ROMAN KERN

A FEATURE ASSOCIATION FRAMEWORK FOR KNOWLEDGE DISCOVERY APPLICATIONS

DISSERTATION

DISSERTATION
zur Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften
der Studienrichtung
Informatik

Graz, 2012

Institute for Knowledge Management
Graz University of Technology

Supervisor/First reviewer: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt
Second reviewer: Prof. Dr. Michael Granitzer

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

Abstract

Most knowledge discovery applications employ a processing pipeline where the knowledge base is transformed into a representation suitable for algorithms to analyse, extract, rearrange and synthesise information. The mainstream approach is the transformation of textual content into a bags-of-words representation to build a documents by terms matrix. This simple, limited representation can be generalised in multiple ways. Documents and terms can be seen as specialised realisations of a more abstract feature concept. The matrix representation itself can be transformed into a bipartite graph structure, which can be generalised as an n-partite graph with nodes that represent generic feature instances. This more general structure allows the integration of additional information, for example the sequence order, the distributional relationships between features and the integration of external services and knowledge sources. The downside of such a generic representation is the associated high computational complexity. In this thesis an algorithmic approach is presented that allows a number of graph operations with contemporary computer resources even for large scale data sets. The usefulness of this approach will be demonstrated by a collection of knowledge management applications with a varying degree of generalisations of the feature space and data structures. Among these applications are recommender systems, information retrieval applications and word alignment algorithms for machine translation. Finally an unsupervised word sense disambiguation scenario is presented, where the statistical, semantic and structural properties of features are exploited for a word sense induction and discrimination task.

Keywords: Feature Associations, Feature Engineering, Natural Language Processing, Machine Learning, Information Retrieval, Social Web

Zusammenfassung

Das Ziel einer Knowledge Discovery Applikationen ist es, aus großen Datenmengen maschinell Information und Muster zu extrahieren. Dabei folgen die meisten Applikationen den gleichen Aufbau. Daten werden eingelesen und transformiert und dadurch in eine Form gebracht, die es maschinellen Methoden erlaubt diese auszuwerten, typischerweise organisiert in sogenannten Instanzen und Features. Hier kommt oft das Vector Space Modell zum Einsatz, in dem die Daten in einer Matrix angeordnet werden. Diese Arbeit beschreibt einen Ansatz, der diese limitierte Daten-Repräsentation in einer Reihe von Aspekten erweitert, um die Information innerhalb eines Daten-Satzes zu extrahieren, im Speziellen jene Information, die in der Beziehungen zwischen Features latent vorhanden ist. Eine zwei-dimensionale Matrix kann in einen bi-partiten Graphen transformiert werden. Diese Daten-Struktur kann dann erweitert werden zu einer n-partiten Graph-Struktur, in der Knoten die Features innerhalb des Daten-Satzes repräsentieren. Zusätzliche Flexibilität kann gewonnen werden, indem die Knoten erweitert werden, um zusätzliche Information aufzunehmen, beispielsweise um externe Quellen anzubinden. Typischerweise geht eine allgemeinere, flexiblere Daten-Struktur mit höheren Laufzeit-Anforderungen einher, die oft einen praktischen Einsatz unmöglich macht. Der vorgestellte Ansatz erreicht dieses hohe Maß an Flexibilität, ohne allerdings ein Laufzeitverhalten aufzuweisen, das durch die theoretische Obergrenze der Laufzeitkomplexität vorgegeben ist. Um die praktischen Nutzen des Ansatzes zu demonstrieren, werden eine Reihe von Knowledge Discovery Applikationen vorgestellt. Diese Applikationen unterscheiden sich in dem Ausmaß an benötigter Flexibilität und Größe der verwendeten Datensätze. Um die allgemeine Nützlichkeit des Ansatzes zu unterstreichen, unterscheiden sich die vorgestellten Applikationen auch hinsichtlich ihrer Domäne. Diese sind Social Web, Information Retrieval und Natural Language Processing. Startend mit einem Recommender System, das eine einfache Daten-Repräsentation verwendet, steigert sich die Flexibilität bis zu einer komplexen Applikation aus dem Bereich der Sprach-Technologien. Hier werden statistische, semantische und strukturelle Informationen ausgewertet um mehrdeutige Wörter mittels eines unüberwachten Lernverfahrens aufzulösen.

Acknowledgements

I would like to thank my supervisor and reviewers Michael Granitzer and Stefanie Lindstaedt.

Furthermore I want to thank my colleagues at the Know-Center and the Knowledge Management Institute at the Technical University in Graz. Especially I want to thank my co-authors (semi-sorted by co-authorship count): Markus Muhr, Andreas Juffinger, Mario Zechner, Markus Strohmaier, Christian Körner, Christin Seifert, Vedran Sabol, Werner Klieber, Viktoria Pammer, Matthias Lux, Hans-Peter Grahsl, ...

Special thanks go to my family and friends for their support and patience!

Everything I know about true bugs I learnt from Thomas Frieß. Stuff, relevant for everyday life, for example the caliber of a Sinn 656, I know due to Rainer. My table tennis skills improved tremendously thanks to Mimo, obviously. // TODO: Add thanks to all relevant people

Contents

Introduction **19**
Motivation . . . 19
Overview . . . 19
Contributions . . . 20
Publications . . . 21
Methodology . . . 24
Feature Association Process . . . 26
Example . . . 28
Structure . . . 29

Knowledge Discovery **31**
Knowledge Discovery Applications . . . 31
Feature Association Framework . . . 37

Feature Engineering **39**
Role of the Feature Association Framework . . . 39
From Data to Features - Preprocessing . . . 40
Feature Extraction . . . 46
Properties of Features . . . 49
Feature Transformation . . . 52
Feature Associations . . . 58
Data-Mining Algorithms . . . 62

Concepts **65**
Overview . . . 65
Configuration & Design Decisions . . . 66
Calculate Feature Associations . . . 75
Feature Association Functions . . . 98
Distributed Environment . . . 137
Feature Association Retrieval . . . 145

Implementation **153**
Introduction . . . 153
Functional Requirements . . . 153
Data Structures & Runtime Environment . . . 160
Design & Main Components . . . 166
Extended Functionality . . . 177
Performance Evaluation . . . 182

Applications **189**
Introduction . . . 189
Social Web . . . 191
Information Retrieval . . . 208
Natural Language Processing . . . 221

Conclusions **243**
Discussion . . . 244
Summary & Outlook . . . 249

Bibliography **255**

List of Figures

1 Application Domains Used for the Evaluation 22
2 The Spiral Development Model 25

3 Overview of the Processing Steps of the KDD Process 32

4 Examples from the Handwritten Digits Data Set 43
5 Growth of the Feature Count in Relation to the Instances of the Brown Corpus 50
6 Occurrence Count of English Words in the Wikipedia 51
7 Example of the Force Directed Placement for the Reuters-21578 Data Set 56
8 Visualisation of an Exemplary SVD Output 63

9 Example of an Input Graph of the Feature Associations 76
10 Sequence of the Execution Phases 78
11 Overview of the Output Collection Phase 82
12 Exemplary Output of the Sort Phase 84
13 Example of a Simple Feature Association Graph 88
14 Example of an Input for the Association Phase 90
15 Example for Contextual Feature Associations 91
16 Example for the Recursive Association Phase 93
17 Topology of the Storage Phase 96
18 Sequence of Execution of Functional Blocks 108
19 Log-Log Diagram of a Feature Frequency Histogram 128
20 Log-Log Diagram of a Smoothed Feature Frequency Histogram 128
21 Output of the Simple Good-Turing Smoothing Algorithm . . . 129
22 Overview of the MapReduce Framework 141
23 Integration of the Feature Association Framework into MapReduce 144

24 Example of the Mapping of the Input Features for the Brown Corpus 163
25 Example of the Output Feature Association Graph Mapping . 164
26 Overview of the Sequence of the Phases with Multiple Cycles 179

27 Popular Tags on Flickr.com 193
28 Screen-shot of the Tagr System 195
29 Compontents of the Tagr System 196
30 Visualisation of the Spreading Activation within the Flickr Association Network 204
31 Results of the Prediction Evaluation on the Flickr Data Set . . 207
32 Example of an Ambiguous Word in WordNet 210
33 Processing Pipeline of the Cross-Language Queries 211
34 Overview of the Retrieval System for CLEF 2008 212
35 Baseline Performance of the CLEF 2008 System 214
36 Performance of the CLEF 2008 System Using WSD Information 215
37 Performance of the CLEF 2009 System for English Queries . . 215
38 Performance of the CLEF 2009 System for Spanish Queries . . 216
39 Overview of the Combined System for Plagiarism Detection . 224
40 List of Grammatical Dependencies for an Example Sentence . 238
41 Phrase Tree for an Example Sentence 239
42 Results of the SemEval-2 WSID Evaluation 241

List of Tables

1 Size of the Input Feature Graph of the Wikipedia Data Set . . 28
2 Main Characteristics of the Output Association Network of the Wikipedia Contextual Collocations 28
3 Top Wikipedia Categories for a Sample Collocation 29
4 Frequency of Wikipedia Articles for a Sample Collocation . . 29

5 Overview of Components for Automatic Spam Filtering 34

6 Main Roles and Tasks of the Feature Association Framework . 39
7 Example for the Feature Selection via Information Gain 54
8 Example for the Topics Generated by the LDA Algorithm . . . 62

9 Execution Strategies Depending on Data Set Size 79
10 Typical Operations of a Storage Backend 96
11 Differences of the Two Types of Components of the Feature Association Function 98
12 Number of Instances in the Wikipedia Data Sets 100
13 Number of Features in the Wikipedia Data Sets 101
14 Average Out-Degree of the Instance Nodes in the Wikipedia Data Sets 102
15 Examples of Infrequent Features in the English Wikipedia . . 103
16 Examples of Frequent Features in the English Wikipedia . . . 103
17 Contingency Table for the Distribution of Two Features in Relation to the Instances 123
18 Definition of Symbols to Reference the Cells Within the Contingency Table 124
19 Co-occurrence Matrix of a Source and a Target Feature 124
20 Matrix of the Expected Frequencies for Independent Features . 124
21 Contingency Table Used as Input for the Mutual Information . 131

22 List of Functional Requirements Depending on the Data Sets Size 158
23 Mapping of the Input Features to the Lucene Data-Structures . 162
24 Mapping of Output Features onto the Lucene Data-Structures 164
25 Example of Common Pruning Parameters 167
26 Overview of the Methods of the Feature Association Function Class 171
27 Overview of the Attributes of the Feature Class 180
28 Main Properties of the Brown Corpus Features 183
29 Output Feature Association Graph of the Brown Corpus . . . 184
30 Performance of the Feature Association Calculations for the Brown Corpus 184
31 Relative Times of the Processing Phases for the Brown Corpus 185
32 Main Properties of the Features for the Reuters RCV-1 Data Set 185
33 Output Feature Association Graph of the Reuters RCV-1 Corpus 185
34 Performance of the Feature Association Calculations for the Reuters RCV-1 Corpus 186
35 Main Properties of the Wikipedia Data Set 186
36 Output Feature Association Graph of the Wikipedia Corpus . 186
37 Performance of the Feature Association Calculations for the Wikipedia Corpus 187

38 Overview of the Application Scenarios 190
39 Overview of the Key Advantages for Integration into the Tagr Application 197
40 Size of the del.icio.us Data Set 200
41 Size of the Flickr Data Set 201
42 Overview of the Main Characteristics of the Flickr Extended Folksonomy 203
43 Results of the Overlap Evaluation for the Flickr Folksonomy . 204
44 Results of the Leave-One-Out Evaluation for the Flickr Folksonomy 205
45 Statistics of the Wikipedia and Europarl multilingual indices . 213
46 Baseline Performance on the CLEF 2009 Test Collection 219
47 Performance on the CLEF 2009 Test Collection Using Query Expansion 220
48 Evaluation of the Block Retrieval Stop of the PAN 2010 System 227
49 Overall performance of the PAN 2010 System 227
50 Overview of Selected WSD Algorithms 231
51 Weights of the Grammatical Features for the WSID System . . 238

List of Algorithms

1 Overview of the Typical Computation of Word Co-Occurrences 59
2 Overview of the Main Processing Steps in the Collect Phase . 84
3 Overview of the Main Processing Steps in the Sorting Phase . 86
4 Overview of the Association Phase when Building Global Associations 89
5 Overview of the Association Phase for Relative Contextual Features 92
6 Overview of the Association Phase for Recursive Contextual Features 94
7 Pseudo-Code for the Map Function as Defined by the MapReduce Framework 139
8 Example of a Simple Implementation of a Reduce Function . . 140

Introduction

This chapter gives an overview of the feature association framework. The list of contributions as well as the related publications are presented. The development process of iterative generalisations will be highlighted. To illustrate the usefulness of the approach, an example is discussed. Finally the structure of the work is presented.

Motivation

In recent times, terms like *data mining* and especially *knowledge discovery* have gained popularity, in the industry as well as in the research community. Even in the mainstream media these terms occasionally start to occur. One might ask what has spurred this interest. The answer is hidden in another question: What is the currency of the digital age? When trying to find answers this question, one quickly gets the main theme of the opinion raised by many people: it is data. Modern technologies make it easy to collect, store and manage huge amounts of data. But its not the data directly, it is the information contained in the data, which carries the true value. The aim of the knowledge discovery process is to help to automatically extract valuable information out of huge amounts of data.

Knowledge discovery summarises a common processing pipeline consisting of a collection of techniques and algorithms. Within this process the input data is refined and converted into so called features. These features are then fed to data mining algorithms and components that visualize the data to the user. Due to limited processing power traditionally the relationship between the features has not been in the focus of research. In some naive approaches, it was even assumed that features are completely independent from each other.

With the increase of processing power and the progress made on the algorithmic side, one can start to exploit the intra feature information. The goal of the feature association analysis is to put features into relation to each other to extract information not directly accessible from processing individual features on their own. Due to the size of the available data and computational complexity of such an analysis, the exploitation of feature relationships proves to be a challenging task on its own. Taking one step further, one may ask on how to improve this process and how to enrich the extraction process to optimise the transformation from plain data into valuable information.

Overview

The feature association framework is a set of algorithms that are tightly integrated to extract information contained in the relationship between features. These features are generated during the life-cycle of a knowledge discovery application. The role of the feature association framework is then

to improve the knowledge discovery process by transforming the features to reflect the extracted information. There are two main use-cases for the feature association framework:

- To **analyse** the characteristics of the relationships between features. This allows to detect previously unknown characteristics and patterns within the data.
- To **synthesise** new features that explicitly represent the information that previously has been only implicitly contained in the relationships. This way the intra feature information can by utilised even by algorithms that treat features as being completely independent from each other.

These two use-cases are highly related in terms of their technical realisation, as both are faced by similar challenges. Furthermore the feature association framework has been developed to serve two different target audiences:

Researcher The analysis gives the researcher a better insight into the nature of the data at hand. This knowledge helps in selecting the appropriate algorithms employed within a knowledge discovery process. Here the main focus is to discover new properties and patterns.

Developer When building a knowledge discovery application, the developer is relieved from the need to have an intimate knowledge of the underlying algorithms to make use of the information contained between features. Instead, the feature association framework can be plugged into existing solutions to transform this information into a representation suitable for many existing algorithms.

These two target audiences are not strictly disjoint, as in many cases people will serve more than one role. For example, researches need to develop applications to help them to uncover empiric evidence for their hypotheses.

Contributions

The main contribution of this work is an approach to extend the traditional way on computing feature associations. This approach allows to integrate and apply various additional algorithms within the process of computing the association network. This opens up a wide array of different ways on how to enrich the processing.

> CONTRIBUTION I: **The feature association framework goes beyond the traditional approach to compute feature associations. Its flexible nature allows to integrate additional processing steps to be applied to enable new ways of research.**

The downside of the feature association analysis is its computational complexity, as each feature needs to be set into relation with each other feature. As features are typically associated with instances, for example documents, these relationship needs to be honoured as well. Therefore for many larger data sets such an analysis has required considerable computational resources or were not possible at all.

> CONTRIBUTION II: Novel algorithmic approaches enable the feature association framework to make the computation of feature associations feasible for a wide range of large data sets.

A common life-cycle of empirical research starts with an intuition. To test whether this intuition holds true in real life, one needs real-world data sets. The researcher therefore needs to analyse the properties of the data set. The confidence in the findings are directly linked with the size of the data.

> CONTRIBUTION III: The feature association framework is a tool for the researcher to gain insights into the properties of features and their relation to each other.

Especially in the field of applied science, one has to rely on existing technologies to answer research questions. In such a scenario often established algorithms are combined to create novel solutions. For example a knowledge discovery pipeline may consist of components developed in the fields of natural language processing or machine learning.

> CONTRIBUTION IV: The feature association framework can be used within existing knowledge discovery solutions to enhance their performance.

In the process of extracting the information between features, it might be important that additional sources of information are integrated as well. Thus not only the properties of the relationships, but also properties of the features themselves are of interest.

> CONTRIBUTION V: The feature association framework proposes a method to integrate a wide variety of external sources of information into the computation of feature associations.

Once the information has been extracted one has to address the question on how to encode, store and manage them. Furthermore, one would want to apply additional processing steps upon the extracted data. Therefore the extracted data needs to conform to data-structures typically found within knowledge discovery applications.

> CONTRIBUTION VI: The result of the feature association computations is transformed into a representation suitable for further processing using existing algorithms.

Finally, the feature association framework is not only an abstract description of algorithms. A reference implementation of the algorithms has been developed. This implementation does not only serve as a prove-of-concept, but has been developed to conform to quality levels demanded by production systems.

> CONTRIBUTION VII: The reference implementation is distributed under an open-source license, thus making it available to every interested party.

Publications

It has been one of the main objectives of the feature association framework to enable and accelerate improvements in the field of applied science. Therefore following an *in vivo* approach for the evaluation appears to be a natural match. A number of use-case scenarios are presented, where feature associations have been used. Each of these use-cases differ in the way the framework is utilised. The knowledge discovery applications were selected from three different domains: Information Retrieval, Social Web and Natural Language Processing The feature association framework can be seen as a link between these disciplines, as indicated in figure 1.

For each of these domains at least two different application scenarios are presented. Each of the covered scenarios is accompanied by peer reviewed academic publications.

Social Web

Recommender System

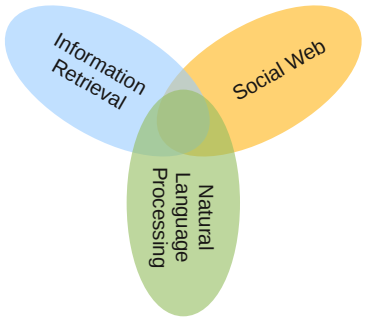

Figure 1: Domains of the applications, which are the base for the evaluation of the feature association framework.

Publication #1: S. Lindstaedt, V. Pammer, R. Moerzinger, R. Kern, H. Mülner, and C. Wagner. Recommending tags for pictures based on text, visual content and user context. In *Proceedings of the Third International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 506–511. IEEE Computer Society Press, 2008

Summary: The feature association framework has been utilised to compute recommendations based on the well known collaborative filtering approach. The contribution of the feature association has been to determine to which extend users tend apply the same tags to tag images.

Tagging Structure Analysis

Publication #2: M. Lux, M. Granitzer, and R. Kern. Aspects of Broad Folksonomies. In *18th International Conference on Database and Expert Systems Applications DEXA 2007*, pages 283–287. Ieee, 2007

Summary: In this paper the distribution of tags, resources and users have been studied. The hypothesis that tags follow a power-law distribution has been found to be true for a large amount of tags. This finding helps improving algorithms dealing with tagged data, for example information retrieval applications.

Publication #3: R. Kern, M. Granitzer, and V. Pammer. Extending Folksonomies for Image Tagging. In *WIAMIS 2008 , Special Session on Multimedia Metadata Management & Retrieval*. IEEE Computer Society, 2008

Summary: Based on a subset of the data available on Flickr, the relationship between tags and other types of meta-data has been studied. Tags have been set in relation to words used in the title, description and comments. It has been found that tags encode information not found in any other type of meta-data. Another finding has implications on the implementation of recommender systems, as as baseline for the performance of the tag recommendation process is presented.

Information Retrieval

Query Expansion

Publication #4: A. Juffinger, R. Kern, and M. Granitzer. Exploiting Cooccurrence on Corpus and Document Level for Fair Crosslanguage Retrieval. In *Working Notes for the CLEF 2008 Workshop, 17-19 September, Aarhus, Denmark*, 2008

Summary: This paper tries to answer the question: Does explicit disambiguation of ambiguous words lead to an improved information retrieval performance? Based on a retrieval system based on the TFIDF approach, improvements were small and not found to be significant. Thus one can conclude, that for a basic retrieval system, the explicit disambiguation information does not help. This insight is relevant, as most of the information retrieval systems in production today follow the basic TFIDF approach.

Publication #5: R. Kern, A. Juffinger, and M. Granitzer. Application of Axiomatic Approaches to Crosslanguage Retrieval. In *CLEF 2009 Workshop*, pages 142–149, 2009

Summary: The existing TFIDF based retrieval application has been extended to include state-of-the-art retrieval techniques, namely the BM25 and an axiomatic approach. This lead to a significant improvement of the retrieval performance, similar to the best published performance numbers for the used data set. Next word sense disambiguation information has been integrated using a query expansion technique. We observed a slight increase in performance. Then we compared the influence of explicit word sense disambiguation information with a query expansion technique based on the distributional semantics of words within a corpus. We found that this approach works more efficiently than exploiting the explicit word sense disambiguation information, but the best performing configuration has been achieved by combing both approaches.

Query Translation

Publication #6: A. Juffinger, R. Kern, and M. Granitzer. Crosslanguage Retrieval based on Wikipedia Statistics. In *Proceedings of 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, 17-19 September, Aarhus, Denmark*, 2008

Summary: In cross-lingual information retrieval a user types in a query in one language, while the searched documents are written in another language. By exploiting the cross-lingual links between the multiple editions of Wikipedia we developed such a system. To assess the impact of the cross-lingual query processing we apply the precessing also on queries written in the same language as the document set. The evaluation showed that our approach works effectively and the cross-lingual information of Wikipedia can be successfully utilized for retrieval applications.

Publication #7: R. Kern, A. Juffinger, and M. Granitzer. Evaluation of Axiomatic Approaches to Crosslanguage Retrieval. In *Multilingual Information Access Evaluation Vol. I Text Retrieval Experiments*, 2009

Summary: We extended our existing cross-lingual retrieval system in two regards. First we added another source of cross-lingual information, namely the Europarl corpus, which offers a higher granularity in terms of aligned text fragments. Secondly we introduced two alternative ways to compute the candidates that are then submitted to search engine. The added information helped to improve the overall performance of the retrieval system. For the candidate selection we were able to identify an approach that consistently outperformed the other two approaches.

Natural Language Processing

Cross-language Plagiarism

Publication #8: M. Muhr, R. Kern, M. Zechner, and M. Granitzer. External and Intrinsic Plagiarism Detection using a Cross-Lingual Retrieval and Segmentation System Lab Report for PAN at CLEF 2010. In *2nd International Competition on Plagiarism Detection*, 2010

Summary: Plagiarism is not only bound to a single language, but might occur across different languages. We have developed an algorithm that extends traditional monolingual plagiarism detection methods to detect cases where the source document has been written in a different language. Therefore a sentenced aligned corpus is processed by applying a word alignment algorithm. The crucial part of our approach is to use more than one translation candidate for each word in the suspicious document. This approach deviates from the mainstream approaches that utilises existing translation systems. The evaluation demonstrates that our algorithm works efficiently while keeping the overhead for cross lingual plagiarism detection reasonable low.

Word Sense Induction & Discrimination

Publication #9: R. Kern, M. Muhr, and M. Granitzer. KCDC: Word Sense Induction by Using Grammatical Dependencies and Sentence Phrase Structure. In *Proceedings of SemEval-2*, Uppsala, Sweden, ACL, 2010

Summary: The task of word sense induction and discrimination is to learn the individual senses of ambiguous words without explicit training data. Based on exploiting distributional properties one tries to detect the number of senses and assign the correct sense to unseen usages. Our approach is based on the output of parser components that produces the parse tree of a sentence as well as the grammatical relationships between words within a sentence. Our aim has been to use few, highly discriminative features as possible to allow a better understanding of the differences that set different senses apart. For this we utilised unsupervised machine learning techniques in combination with a model selection strategy to detect the correct number of senses. In contrast to the mainstream approaches we focused on verbs instead of nouns. In the evaluation we found that our approach delivered satisfying results for a set of verbs. The feature association framework provided the necessary flexibility and performance to achieve this result.

The base for all the presented publications has been the algorithms developed within the feature association framework.

Methodology

The development of the feature association framework has been conducted as an iterative process. Each of these iterative steps has been done based on existing use case scenarios. This way the continuous improvement of existing functionality has been ensured. This type of software development is known as the spiral model, popularised by Boehm[1] and depicted in figure 2.

Alongside this path of development not only functional requirement has been addressed. Other factors, especially the run-time complexity and the necessary computational resources to process the data sets have been guidelines alongside the development cycles.

The feature association framework started out as a simple tool to process a transposed document by term matrix[2]. Many text based knowledge discovery applications utilise such a data-structure. By transposing this matrix the main viewpoint on the data set is transformed. All algorithms

[1] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988

[2] This data-structure records the relation between documents and words, which are stored within the documents. The documents are represented as rows, while the terms (usually simply the words) are represented as columns within this matrix.

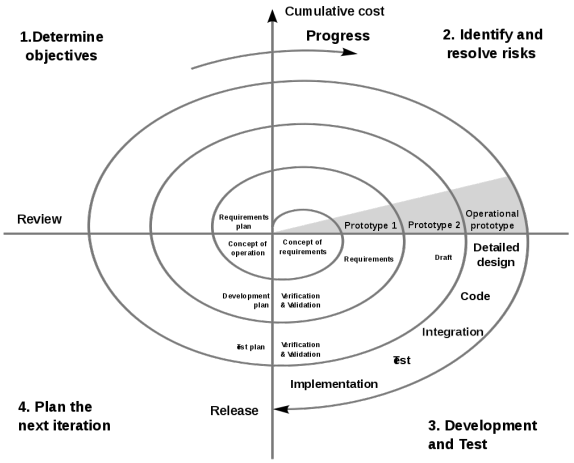

Figure 2: Overview of the spiral development model. The development process consists of a series of iterations to improve and enrich the functionality.

developed to take an document by terms matrix as input are eligible to be applied on the transformed matrix.

While transposing a matrix is mathematically a simple operation, for large scale data sets the realisation of such an functionality poses a number of challenges. As soon as the data out-grows the storage capacity of the main memory of a single machine, the development effort is steeply increased. Strategies to retrieve only the most relevant subset of the whole data set need to be implemented. Naive approaches will no longer work for this amount of data. The feature association framework is equipped with a number of techniques to cope with large data sets, while still preserving a relatively low computational effort.

Soon it became clear that the simple transpose operation of a matrix can be generalised without sacrificing the scalability properties of the feature association framework. Instead of only applying the transpose operation on the input matrix, the output of this operation is additionally multiplied by the original matrix. Thus the input matrix is transposed and multiplied by itself within one invocation of the feature association framework.

The generalisation steps are iteratively repeated by cautiously adding more flexibility to the way the framework processes the data. Real world application thereby served as a driving factor for these adaptations. During the development phase not only the algorithmic side of the framework has been improved. The data-structures has also been modified to a allow a more flexible input as well as output of the computations.

Starting with a simple matrix, the input for the feature association has been generalised to graphs. A matrix can also be seen as a bi-partide graph. Using this viewpoint, the input data-structures have been generalised to cope with n-partide graph structures.

The output of the feature association framework has also undergone modification to allow a greater flexibility. While in the first iteration of the framework the output data-structure has been a matrix, in later versions this data-structure has been made more flexible. The output of the feature association calculations are stored as graph structures, without restrictions on the topology of these graphs. For example this graph may be a directed graph or the relationships between the graph node may be undirected, depending on the requirements of the knowledge discovery application.

Finally the computation of the feature association itself has undergone an evolution starting with the simple copy of values from the input matrix into the transposed output matrix. The feature association function has been steadily refined to allow more sophisticated algorithmic approaches.

The final implementation of the framework defines a number of factors and functional blocks. To integrate a feature association algorithm into the framework it need to be decomposed into these structures. This requirement is a direct consequence to ensure that the execution of the association will scale to the amount of data at hand.

Although this constrains upon the feature association function appear to be strict, it has been shown that a wide variety of existing algorithms can be transformed into a representation suitable for the feature association framework. Among these functions are textbook approaches from the area of information retrieval, for example TFIDF, BM25 and the well known cosine similarity. Other examples are from the field of statistics, like various statistical tests for independence.

Feature Association Process

Although the algorithmic part of the feature association framework as well as the implementation will be covered in great detail in the following chapters, a short overview of the process is given here.

Main Properties

The theoretical upper bound of the computational complexity of the feature association computations is: $\mathcal{O}(n^2 m)$, where n is the number of features and m denotes the number of instances[3]. Thus the main challenge is to introduce measures to make the computation feasible even if the theoretical upper limit does not indicate its usefulness for larger data sets. In order to achieve this, a range of techniques needs to be applied. These techniques try to make optimal use of contemporary computing infrastructure and, more importantly, try to exploit specific properties of the underlying data sets. For example, many features will not be entirely random. In fact, in many data sets, features follow the power law distribution. This property is exploited to reduce the number of computations dramatically.

[3] The number of instances is usually identical to the number of entries within the data set.

Furthermore, the feature association framework allows to apply heuristics to discard features, which carry little information or are too rare to provide reliable statistics. This approach is known as feature selection. Generally it is expected that the features will be sparse on a global level, but locally highly connected. The algorithms and data-structures of the feature association framework are optimized for such a scenario.

Finally, the processes have been developed to allow the computations to be done in a distributed manner. For example, the feature association is compatible with the map-reduce paradigm.

Feature Association Function

The framework allows the usage of different functions to compute the strength of the associations[4]. To keep the computations feasible, a number of constraints are imposed on these functions. The function need to be decomposed into a set of factors and functional blocks, where each of them is responsible for a part of the computations.

[4] The strength of the association is just one possible property of the information contained between features.

More formally, the association strength between two feature nodes in the input graph ($node_i$ and $node_j$) is computed by the function $f(node_i, node_j)$:

$$f(node_i, node_j) = w_{global}(a(node_i, node_j), \mathcal{G}) \quad (1)$$

$$a(node_i, node_j) = w_{aggregate}(\{w_{combine}(l(i), l(j))\}) \quad (2)$$

$$l(node_x) = w_{local}(node_x, \mathcal{L}) \quad (3)$$

Thus, the overall association function needs to be decomposed into the functional blocks w_{global}, $w_{aggregate}$, $w_{combine}$ and w_{local}. Each of these

functional blocks is responsible for a subset of the overall functionality. Thereby the functional block may use the aforementioned factors. These factors allow the integration of additional information, for example global statistics as well as information retrieved from external knowledge bases. As factors are designed to have a minimal set of dependencies, they can be efficiently re-used or computed in advance.

The function arguments that carry this information are denoted as $\mathcal{G}$, for the global statistics of the data set and $\mathcal{L}$ for data about the two features in focus. The framework is responsible to invoke the individual functional blocks, as well as managing the computation of the factors, so that an optimal throughput is achieved. Although the decomposition scheme appear to be strict, there are many examples of functions that have been mapped to this scheme. Among them are the Cosine Similarity, Mutual Information, χ^2, Poisson, Windows Co-Occurrence and many more, which are presented in the following chapters.

Contextual Feature Associations

The integration of global, as well as local statistics is the first way on how the feature associations computations are enriched. The second enhancement relates to the algorithmic aspect. Due to the flexible nature of the feature association framework, one may integrate additional processing steps. Thus, in such a scenario, one can introduce an additional layer to the feature associations.

Starting with a single feature all related instances are collected and passed to a supplied processing pipeline. Within this context all further association calculations are done. For example one may choose to apply a dimensionality reduction algorithm on the context of the feature in focus. The computation of the association strength of all related features is then done in the low dimensional space.

Another, more complex, scenario is to use the contextual processing to invoke another feature association computation. This can be utilised if one is not only interested in the relationship between pairs of features, but on the relationship between three or more features at the same time. For example, if the data contains features which appear to be independent when compared to each other, but correlate for specific states.

The output of the contextual feature associations can be chosen to suit the needs of the other components within the applications. Whether the contextual information is present or not also depends on the requirements.

Reference Implementation

The reference implementation serves two purposes. At first it should demonstrate, that the algorithms do not only work in theory, but also in practice. And secondly the implementation helped developing a series of knowledge discovery applications. These applications did contribute to research in the domains of information retrieval, social web and natural language processing.

The implementation itself has been developed tailored towards contemporary computer architecture. The access times to the main memory have been steadily decreasing in the last years, which is unfortunately not the case for disk access times. Therefore the implementation has been designed to optimise the balance between memory access and storage access[5].

The individual stages of the algorithm will be presented in the following chapters, therefore an in depth description of the components is omitted here. The most important aspect of the implementation is the design decision, that the computations are split into two branches, one responsible to process local sub-graphs of the input data and one responsible to gather the

[5] Currently the implementation is CPU bound as well as IO bound, which can be seen as indicator that a good compromise has been found.

information on a global level. This procedure is similar to many distributed processing frameworks, for example MapReduce[6].

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, pages 1–13, 2008

Example

This example demonstrates how the feature association framework can be used to integrate additional information. Here not only two features are related to each other, but this relationship is further enriched by exploiting the meta-data available in the data set. This meta-data serves as contextual information and represent one possible way on how the traditional approach of computing feature association can be elevated. In this scenario the feature association represent collocations, phrases of two adjacent words, that typically represent a concept, which cannot be deducted from the individual words themselves.

The collaboratively created online encyclopedia Wikipedia[7] is used as data set to build a feature association network. The words contained in each Wikipedia article are mapped as source features, which are the set of features being associated with each other. The final association network will then contain word collocations, where a the first word takes the role of a source feature and the second word is a target feature.

[7] `http://en.wikipedia.org`

Each Wikipedia articles also carries additional information in the form of meta-data. One additional structural information are the Wikipedia categories, which are also collaboratively created and organised. Each article may be assigned to one or more of these categories. The Wikipedia categories serve as proxy for the topics of an article and are therefore can be used as contextual features. Table 1 gives an overview of the main statistics of the feature graph used as input for the contextual feature association calculations.

Node	Type	Avg. In-Degree	Count
Instance	Wikipedia Article	0	3,450,941
Source Feature	English Words	477.0	10,839,070
Context Feature	Wikipedia Category	3.6	460,559

Table 1: Overview of the size of the input feature graph for the collocation detection. Each article consists on average of little less than 500 words and is assigned to about 3.5 categories. The are over 5 billion edges within the graph.

In order to cut down the number of calculation, a number of pruning heuristics need to be applied. For example, categories that are connected to only a handful of articles have been ignored[8]. Words, that occur too frequently or too rarely within the set of all articles were also removed. The computation was executed on a single desktop class machine using a single threaded execution strategy. The whole computation, including storing the results, took 212,939 seconds (2 and a half days). An analysis of the resulting association network is given in table 2.

[8] Statistical significance test need a minimal sample sizes in order to be applicable. Therefore this step is not only necessary to cut down the run-time of the algorithm, but is also a prerequisite to build reliable statistics.

Node	Type	Avg. In-Degree	Count
Source Feature	First Word	0	324,145
Target Feature	Second Word	196.96	315,574
Context Feature	Wikipedia Category	1.21	259,906

Table 2: Main characteristics of the output association network of the Wikipedia contextual collocations. A total of 62,154,056 associations are found for about 250,000 contexts. There are a total of 36,028,078 pairs of category and collocations.

The final collocation association graph can be seen as hyper-graph, where each hyper-edge connects a source feature with an context feature while traversing over a single target feature node. In this graph there are about 36 million of such hyper-edges. As an example, in table 3 a list of the Wikipedia categories in which the words "`freak`" and "`wave`" build a statistically significant collocation. These categories define the context in

which the two constituents carry a new distinct meaning if they are used as a phrase.

Wikipedia Category	Association Weight
Disappeared ships	0.233
Water waves	0.095
Rogue wave incidents	0.095

Table 3: Top 3 Wikipedia categories for the collocation "`freak wave`". Both individual words of the collocation are common in general English. The term "`freak`" occurs 4,432 times and the term "`wave`" 30,685 times in all Wikipedia articles. The combination of the two words describes a distinct concept and the Wikipedia categories describe the context this collocation is used. The association weight is calculated using the normalized Pointwise Mutual Information.

The actual method how the contextual output data-structure is exploited depends on the application. Many collocation are shared between multiple topics and therefore a post-processing step to aggregate the hyper-edges may reveal common collocations when no explicit context information is available.

Optional Information Additional to the association weight, each edge within the output graph may be equipped with a custom payload. Within this payload any information that is necessary for the application can be stored. Therefore the actual content of the payload varies depending on the settings and the implementation.

One example for an payload that could be useful for a real-world application is a list of the instances that contributed the most to the final association weight between two features. The weight of an association is aggregated over all shared instances, but some instances have a higher influence than others. The list of the instances with the highest influence may be stored within the association payload. This information can then be exploited to build better tools especially if the association are directly presented to the users. Table 4 gives an overview of how a list of top instances may look like.

Wikipedia Article	Frequency
Rogue wave	26
Draupner wave	4
MS Bremen	2

Table 4: List of exemplary Wikipedia articles in which the collocation "`freak wave`" occurs in and which contribute to the final association weight between the two words. By adding this information to a visualisation of the output graph a user might gain a better understanding of the feature relationships within the data set.

Structure

The remainder of the work is structured as follows:

Knowledge Discovery This chapter gives an overview of the common process of knowledge discovery and its main challenges. A formal definition of the process is given, as well as a number of examples to illustrate the task of knowledge discovery applications. Readers familiar with the main concepts of knowledge discovery are free to skip this chapter.

Feature Engineering Within the knowledge discovery process, feature engineering plays a major role. This chapter presents an overview of feature extraction, feature transformation and feature association techniques. Furthermore an overview of common properties of features found in many data sets is given. The role of the feature association framework within this process is described.

Concepts A detailed description of the core concepts of the feature association framework is given in this chapter. Its algorithmic approach as

well as the computational properties are presented in a comprehensive manner.

Implementation This chapter gives an overview on how the main concepts are implemented together with a reasoning why specific decisions have been made. On a detailed level the algorithms are analysed and their performance aspects are measured. This is of particular interest for those who wish to integrate the feature association framework into their knowledge discovery applications.

Applications The individual knowledge discovery applications, which serve as a base for the publications are presented. The role of feature associations within these applications as well as the achieved results are reported.

Conclusion The final chapter summarizes the results and gives the answer to the main research questions in a concise manner. It finishes with a short outlook on the future work and possible future research directions.

Knowledge Discovery

The majority of knowledge discovery applications aim at extracting valuable information out of huge amounts of unstructured data. The anatomy of such applications has been studied in the past and it has been found that many of these applications share a common structure.

Knowledge Discovery Applications

As computers grow more and more ubiquitous in the way people conduct their work, the amount of digital data. This also gave rise to a new breed of applications which extract valuable information out of the data.. To refer to this new family of applications, the term *knowledge discovery from databases* (KDD) has been introduced in 1989[9]. This initiative had a big impact, spawned a lot of research and finally founded a new field in the area of computer science. Originally the suffix "from databases" has been used to signify the importance of addressing the issue of increasingly larger data sets.

Previously little attention have been given to fact that algorithms should also cope with amounts of data which exceeds the capacity of the available main memory. Furthermore the Internet has gained momentum and thus became a source for valuable data sets on its own. Therefore algorithms needed to be designed to handle large amounts of data.

Today large data sets are common. These data sets easily exceed the storage capacity of single machines. Therefore the scalability aspect of algorithms and systems is given appropriate attention in the mean time. Traditional databases have been in part replaced by dedicated system to store data in a distributed manner, for example BigTable[10] or Hive[11]. Furthermore the paradigm of relational databases have been extended by a new class of structured storage methods, the so called NoSQL databases. These systems have been developed mainly motivated to increase scalability. Because of fact that big data-set have become commonplace, the suffix "from databases" is now often omitted, and the field is just referred to as *knowledge discovery*.

In 1996 a seminal paper has been published by Fayyad et at.[12], which summarises the main goals of knowledge discovery. Furthermore it gives an overview of the anatomy of a knowledge discovery application together with the biggest challenges that this new research field has to face.

There have have been many alternative names used to refer to the same or similar tasks addressed by the knowledge discovery process. Among them are knowledge extraction, information discovery, information harvesting, data archaeology, data pattern processing and data mining. While the first couple of terms are rarely used anymore, the last term is still being commonly used. But data mining is generally seen only as a single part of the overall knowledge discovery processing sequence.

[9] G. Piatetsky-Shapiro. Knowledge Discovery in Real Databases: A Report on the IJCAI-89 Workshop. *AI Magazine*, 11(5)(5):68—-70, 1991

[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation OSDIâĂŹ06*, 26(2):1–26, 2006

[11] `http://hive.apache.org/`

[12] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996

Knowledge Discovery Definition Data mining is the process of extracting pattern out of unstructured data, by applying algorithms to analyse the data. Pattern extractions represents the core aspect of a knowledge discovery application. Knowledge discovery extends the data mining aspect in many areas. Thus the definition of the whole process can be given as:

> Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data[13].

[13] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: an overview. In U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1, pages 1–34. American Association for Artificial Intelligence, 1996

The definition can be further refined, by making the main concepts more explicit:

Data A set of facts, that can be processed by algorithms

Pattern An expression in a language describing facts, which are contained in the data

Process Knowledge discovery process consisting of multiple steps, which involve processing of the data, data mining and interaction with users

Valid The extracted patters need to have a certain predictive power, they need to generalise

Novel The patterns must not be known in advance

Useful The extracted information needs to serve a purpose

Understandable In order to produce knowledge, the users need to be able to interpret the results of the process

The term interestingness has also been used as an umbrella term for validity, novelty, usefulness and simplicity[14]. Thus the pattern extracted by the knowledge discovery should fulfil the requirement of interestingness to the target audience.

[14] A. Silberschatz and A. Tuzhilin. On Subjective Measures of Interestingness in Knowledge Discovery. In U. M. Fayyad and R. Uthurusamy, editors, *Evaluation*, pages 275–281. AAAI Press, 1995

Another important aspect of knowledge discovery its interdisciplinary nature. Multiple research fields are combined in order to cover the entire knowledge discovery process. Knowledge discovery applications usually require a variety of skills.

Knowledge Discovery Process Within the definition of knowledge discovery special emphasis is laid on the fact, that knowledge discovery itself is a process. This process is similar for many applications and can be formalised as a sequence of multiple consecutive steps. An overview of these steps and their relationships is presented in figure 3. Starting point is a collection of data and the definition of a goal which should be solved by the knowledge discovery application.

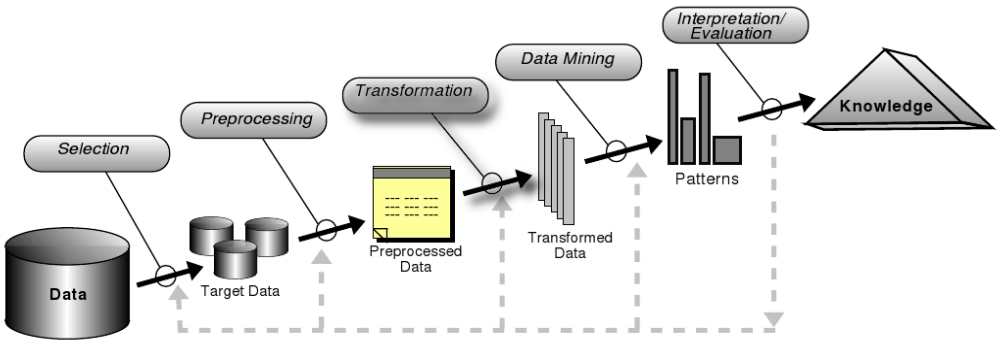

Figure 3: Overview of the processing steps of the KDD process as outlined by Fayyad et al. The transformation step has been highlighted to indicate that the feature association framework is invoked during this step.

[Image taken from the respective publication]

1. The data is then first selected to come up with a selection or sub-set of the complete data on which the processing should be conducted
2. The target data is then cleaned and preprocessed
   - Building a model to assess the amount and characteristics of noise within the data
   - Removal of noise and identification of outliers
   - Implementing measures to handle missing or incomplete data
   - Developing strategies to cope with the dynamics and evolution of the data
3. The preprocessed data is then transformed into data-structures suitable for data mining algorithms. Usually these data-structures are called feature-spaces, which themselves consists of features. These features encode the information contained within the data[15].
   - Identify useful features
   - Analyse the properties of the features and expose this information again as features
   - Algorithms are applied to reduce the complexity of the data, for example dimensionality reduction and projection
4. Choose the data mining task
   - Depending on the overall goal of the application, select the corresponding data mining approach
   - For example choose a clustering approach if no training data is available
5. Select the data mining algorithms
   - Choosing the appropriate algorithm for the data and the overall task
   - Together with the method its parameters algorithms need to be defined
6. Apply the data mining algorithms
   - Processing the data with the selected algorithms will produce an output specific to the chosen method
7. Interpret the results
   - Present the results of the processing to the user
   - This step often requires the development of custom tailored visualisations
8. Consolidation of the results from all processing steps

[15] The feature association framework is applied during this step to analyse the relationship between features and to create features capturing these relationships.

In the most simple case these steps will be executed in a fixed sequence. In typical scenario there will be cycles within the processing chain. This is especially true for application which provide mechanism to allow users to interact with the system.

Examples of Knowledge Discovery Applications

Given the definition of the knowledge discovery process, a number of exemplary application are presented. These examples should give an understanding of the scope of problems being addressed by knowledge discovery applications. In addition an overview if given of the most common data mining approaches and algorithms.

Recommender Engine Recommender systems are arguably the most prominent example for a knowledge discovery system. These have been pioneered by many on-line retailers, mostly motivated to increase their sales volume. The term basket analysis hints at the most common mode of operation of these recommender systems. The main principle can be summed up to: "People, who have bought item X, also bought item Y" Therefore people interested in item X are also given the recommendation to also buy item Y.

Usually these on-line retailers add additional processing steps to this simple principle to increase the likelihood of people following their suggestions. Given the amount of additional revenue these recommendations generate, this kind of knowledge discovery application is probably the most successful, at least commercially. For example, the on-line retailer Amazon[16] reports, that about 35% of product sales are due to recommendations[17]. It is reported, that in the case of Netflix[18], a company that provides on-demand video streaming, about 60% of views are result of personalized recommendations[19].

There are multiple alternative ways, in which data mining algorithms can be employed while computing recommendations. The most popular approach is known as collaborative filtering. Out of these algorithms, the so called item-to-item approach has been chosen by Amazon.com as base for their recommender system. An alternative method is based on the k-nearest neighbourhood method, where similar items are search via shared properties.

[16] `http://www.amazon.com`

[17] `http://www.webcitation.org/62wPepYSA` (accessed on 2011-11-04)

[18] `http://www.netflix.com/`

[19] `http://www.webcitation.org/62wPlcr9x` (accessed on 2011-11-04)

Spam Detection Another example for a knowledge discovery application, that has found its way into most mainstream e-mail clients, is the detection of Spam. Sending e-mails is cheap, which allows to send out commercial e-mails in bulk to millions of recipients. The vast majority of these will consider this kind of e-mail as Spam, thus not being not relevant for them. Machine learning techniques help to automatically detect this kind of misuse and relieve the user of having to deal with these e-mails in a manual way.

Usually a combination of techniques is applied to automatically detect unsolicited commercial e-mails. This combination typically consists of a number of manually assembled rules and components that automatically learnt how such an e-mail looks like. The most popular type of automatic Spam filtering is to use a Bayesian classifier algorithm. In table 5 the output of a Spam detection software[20] is given for an e-mail that has been classified as Spam.

[20] `http://spamassassin.apache.org/`

pts	rule name	description
-0.0	RCVD_IN_DNSWL_NONE	RBL: Sender listed at `http://www.dnswl.org/`, no trust
3.3	BAYES_99	BODY: Bayes spam probability is 99 to 100 [score: 0.9966]
-1.5	SPF_PASS	SPF: sender matches SPF record
0.3	HTML_MESSAGE	BODY: HTML included in message
0.8	MPART_ALT_DIFF	BODY: HTML and text parts are different
0.5	MIME_HTML_ONLY	BODY: Message only has text/html MIME parts
1.7	MIME_BASE64_TEXT	RAW: Message text disguised using base64 encoding
0.0	LOTS_OF_MONEY	Huge... sums of money
0.0	MIME_HTML_ONLY_MULTI	Multipart message only has text/html MIME parts

Table 5: Output of a Spam filter software for a single e-mail. The final decision of the system is a combination of manually crafted rules and a machine learning component.

Web Search Another application of knowledge discovery technologies is web search, were many different data mining activities are being applied.

Users take it as given, that they just have to type in a few words and are automatically directed to a web pages there were looking for. In the background an array of algorithms is responsible to produce the result the users are looking for in a short amount of time.

While there are multiple aspects of web search, only a small part of the whole web search process is picked out here. One of the challenges, a web search engine faces is the fact, that web pages are often copied or duplicated. Additionally the same web page might be available via multiple addresses. Usually multiple version of the same web page are not exactly identical, but minor modifications are present. For example, the header and footer are different, or sometimes some navigational content is added.

Unsupervised machine learning algorithms can be applied to automatically detect near-duplicates and group them together. This technique is called clustering. In contrast to supervised methods, no training examples are required in this setting. In the use-case of web search, all copies of a web site are clustered together and then presented to the user as a single search result.

Main Challenges for Knowledge Discovery Applications

This section highlights some of the main challenges, that need to be overcome when developing a knowledge discovery application. Of course not every single application will need to address all of these issues. The challenges are not all technical in nature.

Multi-Domain A knowledge discovery application is hardly ever developed by a single person. Usually there will be many people involved, usually experts in certain fields. Because of the interdisciplinary nature of knowledge discovery, it is necessary to gather input from different domains in order to solve the problem setting.

Even when looking at the data-mining aspect alone, it might be necessary to bring together experts with different backgrounds, for example machine learning and information retrieval. Naturally the input of domain experts will be required, not only during the starting phase of the development, but also throughout the whole development cycle to provide feedback and sample data-sets.

Because experts from different field speak their own language, use their own vocabulary, it is necessary to agree on a common terminology.

Tool Selection Another issue that needs to be solved early on in the development process is the selection of the tools and algorithms. In the beginning of a knowledge discovery project it is often unclear which approach will work the best.

Having gained experience in previous projects help to avoid the tedious and time consuming process of trial and error. This is not limited to the development of knowledge discovery applications. Nevertheless the performance of the algorithms are hard to predict, even for experts, making the planning and development a hard task.

Scalability - Big Data - Distributed Computing When it comes to huge amount of data, automatic methods do have their advantages in relation to human work. But there are limits. When the size of a data-set exceeds the available memory, there might be the need to adapt algorithms. Sometimes an algorithmic approach is even infeasible in such a setting, for example an algorithm that requires random access to the whole data-set, at any given time during the processing.

Given the growth of data, this problem is no longer affecting only a few knowledge discovery applications. The term "big data" is has raised much interest in recent times. To cope with rising data-sets, traditionally the answer has been to build increasingly bigger machines.

In recent years, an alternative approach has become popular - distributed computing. Instead of using one big machine, multiple commodity machines are tied together and the workload is distributed among them. The cluster of machines is connected via networking infrastructure. Among the challenges in the management of such network topologies is to deal with failures of single machines in a graceful manner and the optimisation of the throughput.

The management of clusters has become a business case on its own. For example Amazon has begun to invested into this business a few years ago and is now one of the leading provides of cluster services. One of their customers recently set-up a cluster consisting of 30,472 cores, 26.7TB of RAM and 2PB (petabytes) of disk space[21].

[21] http://www.webcitation.org/63BwX6DNs (accessed on 2011-11-14)

This movement it not only driven by commercial interest. The open-source community does also provide tools to allow distributed computing, for example Apache Hadoop. This software is for example used by Yahoo, where Hadoop is used to manage a cluster of 45k nodes within 3 data-centers[22].

[22] http://www.webcitation.org/63Bwg0cQT (accessed on 2011-11-14)

Keep the User in the Loop There are cases, where automatic methods achieve a better performance than humans. But these cases are still considered to be the exception. Usually trained humans out-perform even well tuned algorithms, especially for complex problem settings. Because of the huge amount of data, manual processing is often not feasible.

It is one of the challenges of knowledge discovery application to combine the strength of the automatic methods and the high precision of human decisions. Therefore the application should be designed to process the bulk of data using automatic algorithms and to detect corner cases, which are then manually processed.

Another difficult aspect is to reach a certain level of acceptance of the information extracted by data mining algorithms. Users of an knowledge discovery application need a level of trust in the system. Therefore it is important, that the inner workings of the application are presented in a way that can be interpreted by humans.

Feature Association Framework

Having defined the main concepts behind knowledge discovery, the role of the feature association framework within this process can now be presented. As the name implies that it is not a standalone application by itself. Instead it is an algorithmic framework to enable the analysis and synthesis of features. The framework itself deals with the management of the data in order to allow an efficient overall computation.

Within the knowledge discovery process (see figure 3), the feature association computations take place in the transformation step, just before the data mining activities are started. Generally speaking, the aim of the feature association framework is to help in the task of feature engineering.

The input to the feature association computations is the data set already transformed into features. The output of the feature association computations depends on the main use-case:

Feature Analysis The relationship between the features from the data-set are analysed. In this use-case, the features are not modified. The output of the analysis is either stored alongside the features, or directly visualised for the manual inspection by expert users.

Feature Synthesis As in the first use-case, the relationship between features are analysed. But in this mode of operation, the result of the analysis step is transformed into a set of new features. These new features may either be added to the existing set of features, or may replace the old features entirely. In this use-case the newly created features are treated by the succeeding data-mining algorithms as if they were they part of the input data-set.

Feature Engineering

Features are the smallest conceptual entity within a knowledge discovery application. The input data is transformed into structured features, usually by manually crafting a mapping from the raw data to a representation suitable for the domain and used algorithms. In this chapter the task of feature engineering will be presented. Common properties of features will be described and how these can be exploited to efficiently compute feature associations.

Role of the Feature Association Framework

The task of feature engineering is to prepare the data to be processed by data-mining algorithms. Only if the data is transformed into a suitable representation, the results will be optimal. The main challenge in this phase is to uncover and emphasise those aspects, which carry the most valuable information.

The main role of the feature association framework here is to support the process of feature engineering. In table 6 an overview is given of the various strategies that can be applied in this phase. For each of these strategies its main aim and the main role of the feature association framework is listed. Which of these tasks is invoked depends on the data-set and the overall goal of knowledge discovery application.

Name	**Task**	**Main Role**
Preprocessing	Prepare data-set	Support NLP tasks
Feature extraction	Generate features	Provide analysis
Feature normalisation	Modify feature values	Provide global statistics
Feature selection	Modify feature space	Provide measures
Dimensionality reduction	Transform feature space	Support computation
Feature associations	Generate new features	Provide associations

Table 6: Overview of the tasks within the feature engineering phase. For each task the main role of the feature association framework is highlighted.

From Data to Features - Preprocessing

Selecting the data is the first step in the life-cycle within the development of a knowledge discovery application. First the data-sources need to be analysed to define where the most relevant data can be found. Methods to collect the input from these sources need to be developed. In the following step, the data set is pre-processed and cleaned. Finally the data set is transformed into features, organised in feature spaces[23]. This step is crucial as the data-mining algorithms will only work as good as their input allows them to do[24].

[23] Unfortunately the term feature space is sometimes used interchangeably with the term vector space, while the latter one can be seen as just one possible instance of a feature space.

[24] This is sometimes referred to as "garbage in, garbage out".

Select the Data-Sources

The development of a knowledge discovery application usually starts with selecting the data-sources which should be then used to extract information. In the most cases, these data-sources will be heterogeneous and contain data in multiple formats. How this data is organised and how the data is being collected, depends largely on the setting. In the following, a few example settings will be presented.

Example: Enterprise Typically companies manage their documents using internal document management systems (DMS) or content management systems (CMS). These documents are mostly textual documents, images and other unstructured format. Additionally the company runs a set of database-systems, were the data is stored in a highly structured, but proprietary format.

Given this scenario the main challenges in this phase of the knowledge discovery application are:

- Document management systems do not always allow easy access to the stored documents. Often the producers of such system artificially try to avoid making the transition to a competing product too easy. In addition security restriction and limited access to parts of the data are among the common problems that need to be sorted out during this phase.
- Although many file-formats are touted to be standardised and openly documented, in reality most formats other than plain text will cause considerable effort to process. Software manufactures are keen to keep exiting users locked into using their products. Unfortunately this will cause many sources of valuable information to remain untapped.
- Database systems often use an proprietary, often home-grown, scheme. Domain knowledge is necessary to select and transform the data into data-structures suitable for further processing. Sufficient documentation and knowledge on how the data should be accessed cannot be taken as given.
- Combining the unstructured with the structured data can be a challenging task. Aligning multiple sources of information requires often a custom build processing pipeline. Detecting duplicated information and noisy data are among the problem to overcome during this stage.

Most of these problems lie outside of the responsibility of a knowledge discovery application. Still, the quality of the result of the knowledge discovery process may be seriously hampered by some of these problems. There are ongoing initiatives to address these issues[25].

[25] It should be noted, that the improvement of the results of a knowledge discovery process is not always the major driving force behind these activities.

The rise of open-source solutions did also contribute to a change in attitude in regard to the computing ecosystem. The importance of open interfaces and open standards are slowing gaining the awareness of a greater audience. Initiatives like the *Open Document Format for Office Application*

provide file-formats that are openly documented[26]. This enables documents being exchanged by programs from different vendors. In addition this will allow the file-formats to be processed by future software generations, allowing cultural heritage to exist also in the digital domain.

The standardisation of meta-data, for example the *Dublin Core Metadata Initiative*[27], ease the alignment of documents from different sources. Instead of developing custom database schema from the ground up, organisations have started re-using existing schema. Increase in the processing power now allow the additional overhead caused by storing the data in a more structured way, for example using *RDF* and *triple-stores* instead of relational databases.

Example: User Generated Content The World Wide Web and associated technologies have undergone a transformation in the last decade. The changes associated with the Web 2.0 did allow more and more user to contribute. Today a large degree of the information found on the Internet can be categorised as *user generated content.* A diverse set of Blogs, Wikis, Forums and other types of Web platforms allow people to post their opinion, share their photos or simply inform the world of their current mood. These platforms have been recognised to be valuable source of information. For example to detect upcoming trends or to mine the opinion about products.

Because of the abundance of data and its varying degree of quality[28], algorithms need to be developed to automatically extract the relevant information. In this setting the Web can be seen as corpus to build the base for a knowledge discovery applications.

But before any algorithms can be applied, the data needs to be obtained. Because of the volatility of the Web this is a challenging task by its own. Not only does the data change, but also the technologies evolve at a fast pace. Usually the Web pages are scraped by dedicated tools, so called web crawler. In the simplest case, a web crawlers starts with a given web-page, downloads the content, analyses the content for hyper-links and adds the found links to the list of web-pages to visit.

Of course, such a naive algorithm would not work sufficiently well in real life. The crawling process needs to be thoughtfully defined and specifically tailored towards the target web-pages[29]. There are a number of questions that need to be addressed in such a setting:

- Where does the crawling process start?
- What is the strategy to follow outgoing links?
- When to stop the crawling process?
- How to deal with changes of the already visited sites?

Once all relevant web-pages have been collected, the data needs to be processed to extract only the relevant content. Usually a web-pages contain a lot of additional content. For example links for navigational purpose, decoration and advertisements. Methods have been developed to automatically strip away these adorning elements[30].

Often only parts of a web-page are of interest, which can be a challenging task, given the volatile nature of the web. The process of extracting parts of a web page is called wrapper generation[31,32].

Example: Research Corpora Seen from the perspective of the necessary effort to obtain the raw data[33], research corpora lie at the other end of the spectrum. They are deliberately constructed to be easy to parse and easy to process.

[26] And free of patents and other trickery to achieve a vendor lock-in.

[27] `http://dublincore.org/`

[28] In this context, the term quality just reflects whether the content is suitable for the task at hand. Even terrible written rants might be a valuable source for some analysis.

[29] A. Juffinger, T. Neidhart, A. Weichselbraun, G. Wohlgenannt, M. Granitzer, R. Kern, and A. Scharl. Distributed Web2.0 crawling for ontology evolution. In *Proc 2nd International Conference on Digital Information Management ICDIM 07*, volume 2, pages 615–620. Ieee, 2007

[30] P. Fankhauser and W. Nejdl. Boilerplate Detection using Shallow Text Features. *Text*, 2010

[31] N. Ashish and C. A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *coopis*, page 160. Published by the IEEE Computer Society, 1997

[32] J. R. Gruser, L. Raschid, M. E. Vidal, and L. Bright. Wrapper generation for web accessible data sources. In *coopis*, page 14. Published by the IEEE Computer Society, 1998

[33] To clarify, this is only relates to the technical effort to harvest the data. Unfortunately, many research corpora are kelp behind "paywalls" which is a burden not only for open-source projects but also for many research institutions, for instance in development countries.

Their aim is to create a common platform to compare methods and algorithms. Therefore the data stored in these corpora is expected to be clean and easy to parse. In addition, usually these data-structures have been well documented. Examples for such corpora are:

Natural Language Processing The British National Corpus[34], the Brown corpus[35], the American National Corpus[36]

Machine Learning Reuters-21578[37], 20 Newsgroups[38]

Information Retrieval The Cranfield collection, TREC collections[39]

Still, there is often a non-negligible effort associated with adopting such a corpus to a knowledge discovery application. Like in the other presented scenarios, it is important to remove any unnecessary content, while keeping the main data intact.

[34] BNC Consortium. The British National Corpus. *Distributed by Oxford University Computing Services on behalf of the BNC Consortium*, 2007

[35] W. N. Francis and H. Kucera. Brown Corpus Manual. *Brown University*, 1979

[36] N. Ide and C. Macleod. The American National Corpus: A Standardized Resource of American English. In *Proceedings of Corpus Linguistics 2001*, pages 274–280. Citeseer, 2001

[37] D. D. Lewis. Reuters-21578, distribution 1.0. 1997

[38] K. Lang. 20 Newsgroups

[39] D. K. Harman. The TREC test collections. *TREC: Experiment and evaluation in information retrieval*, pages 21–52, 2005

The main task of the first phase of knowledge discovery application is to collected data from various sources and various formats. The output of the this phase is a data-set, which is independent from its original source and free of proprietary file-formats. This data-set is organised in data-structures appropriate for further processing.

Pre-Process the Data

Once the data-sources have been agreed upon and a pipeline to align the input formats have been established, the next step within the knowledge discovery application life-cycle can begin. In this step the data-set is analysed to detect outliers[40] and to remove noise and to find appropriate strategies to handle missing data. If the data-source has been a clean research corpus, this step can be sometimes skipped entirely. Unfortunately this is not the case for the majority of applications.

[40] I. Ben-Gal. Outlier detection for high dimensional data. *ACM Sigmod Record*, 2001

Depending on the nature of the data and the scope of the application, different approaches are applied in this phase. A number of examples should serve as demonstrations which processing steps need to be conducted to pre-process the data-set.

Example: Handwriting Recognition The recognition of handwritten text has been a popular introductory example in textbooks about machine learning[41]. In this example only the recognition of written letters is covered. The input data-set in this case is collection of scanned in handwritten digits. The output should be a cleaned data-set, which should be easy to process by succeeding stages.

[41] P. Harrington. *Machine Learning in Action.* Manning Publication Co., 2012

For the task of recognition of handwritten digit the natural approach is to starts with scanned in samples of written text. The collected samples of text are stored as images, usually using a high resolution and often only in black and white. These images need to be pre-processed before any data mining algorithms can be applied on them.

The scans will contain a number of visual artifacts. For example caused by dust, unevenness of the paper and many other causes of noise. Furthermore the scans may vary in size, the text has been written by different pen of varying thickness. Inevitably different writers will apply different pressure, causing the saturation of the written text to vary. These are among the challenges that need to be resolved during the pre-processing phase.

Once the images have been undergone a cleaning stage, the digits need to be aligned to comparable sizes. This alignment step has been already been done for the MNIST data-set[42]. They applied a number of steps to allow a fain comparison of handwritten digit classification algorithms:

[42] Y. LeCun and C. Cortes. The MNIST Database of Handwritten Digits

- The digits have been scale to cover a box of 20x20 pixels, preserving the aspect ratio

Figure 4: Examples of handwritten digits from the MNIST data-set. These samples represent instances that have been miss-classified using a machine learning approach.
[Image taken from: http://www.cs.berkeley.edu/~smaji/projects/digits/]

- While the original images were bi-colour, grayscale colours were introduced during the scaling process
- The digits have been centred according to their centre of mass within an area of 28x28 pixels

Examples of the cleaned and pre-processed can be seen in figure 4. These examples represent samples that were not successfully recognised using custom SVMs for classification[43].

The authors of the MNIST data-set state that the selection of a different alignment approach does influence the performance of the machine learning algorithms. When the images are centred not by centre of mass, but by applying a bounding box, some algorithms, for instance SVMs and k-NN, improve in performance. This demonstrates that the final output of a knowledge discovery application also depends on the interplay between the individual stages.

Example: Authorship Attribution from E-Mails The base data-set for the next example of a pre-processing pipeline is a collection of E-Mails collected from employers of Enron[44,45]. The use-case here is to automatically assign the E-Mails to one of the authors. As E-Mails are either plain-text or HTML, the data-collecting phase is relatively easy[46]. The pre-processing appears to simple as well - at first glance.

The first step when pre-processing E-Mails is to distinguish between cited text and the main content of the message. A number of heuristics will handle this task sufficiently well, for example to mark each line starting with a '>' character as quote.

Person names and names for organisations need to be masked out. This is motivated by the apprehension that named entities would cause the machine learning algorithms to "learn" the wrong properties. In the task of authorship attribution the writing style of individual authors should be learnt. Named entities, like location names, will also by highly correlated with specific authors. In the presence of entities the algorithm would solely focus on these words[47]. This will ultimately to lead to a problem known as

[43] S. Maji and J. Malik. Fast and accurate digit classification. Technical report, Citeseer, 2009

[44] B. Klimt and Y. Yang. Introducing the Enron Corpus. *Machine Learning*, stitutep1:wwceasccaers2004168, 2004

[45] B. Klimt and Y. Yang. The Enron Corpus : A New Dataset for Email Classification Research. In *Machine Learning ECML 2004*, volume 3201 of *Lecture Notes in Computer Science*, pages 217–226. Springer, 2004

[46] Usually in E-Mails only a subset of HTML is used, making the parsing of the content easier.

[47] The model that is created by machine learning algorithm would just contain of these words. This will cause problem known as over-fitting.

over-fitting, where the algorithms are tuned for only one data-set, but fail to work for other data-sets.

The next step in the pre-processing pipeline is the segmentation of the E-Mail according to is layout. People differ in the way they organise and style their E-Mail messages. Some authors try to keep paragraphs short and others just type in their text without taking formal aspects into consideration. This steps provides additional clues not only for classification algorithms, but also for other pre-processing steps. Other algorithms also profit from the layout information, for example components to split the text into sentences and individual words.

Example: Similarity of Textual Documents Typically when dealing with textual data, the layout does not provide much beneficial information. The semantics of the text is the most value resource in this case, especially documents are compared whether they cover the same topics.

The research field of Natural Language Processing (NLP) tries to develop methods and tools to uncover semantic and syntactic information out of human language. Some of these tools can be seen as knowledge discovery application on their own. More commonly NLP algorithms are applied in the pre-processing phase to analyse the documents.

There are many open-source and commercial tools available to parse text, for example OpenNLP, GATE and UIMA. Each of these tools come with a wide array of methods to extract information out of written text. The output of these methods can later been used to build appropriate features. Among the most commonly used NLP methods are:

Token Splitter Given a sequence of characters, the task of the token splitter to identify boundaries between word. Although this may sound like a trivial task, but the quality of the tokenisation process may indeed have an impact on the performance of the overall system. For some corner cases, even linguists do not agree whether specific language constructs should be counted as a single word, or multiple, for example "isn't" can be either be interpreted as "is not" or a single word[48].

Sentence Splitter Instead of identifying individual words, the task of the sentences splitter it to detect sentence boundaries. Again a task, which requires little effort for humans, but to produce a robust sentence splitter takes considerable effort. These methods need to be robust and produce usable results, even if some preconditions are not met, for example the text does not contain sentences, but for instance only tables of data.

POS Tagger Once individual sentences have been extracted, the grammatical function of each word within each sentences should be detected. This is the task of the Part-of-Speech Tagger. Various approaches have been proposed in the past[49,50,51]. Today an accuracy of over 90% is expected, given clean data and grammatical correct sentences.

Chuncking & Phrase Detection Sequence of words within a sentences can be grouped as they serve a common function within the sentence[52]. For example a noun phrase consists of a head noun and may contain other nouns as well as adjectives. The output of such a parser component is a sentence parse tree, where phrases are hierarchically organised.

Grammar Dependencies Given a sentence the main grammatical relationships between its constituents are extracted[53]. Out of a fixed set of grammatical dependency types, pairs of words are connected. For example, the main verb of a sentence is connected to the subject noun.

Named Entity Recognition Deep parsing is associated with a high computational complexity and its performance is still may not sufficient for many applications. Another approach is to identify only parts of the

[48] Many modern token splitter split this word into two tokens: "is" and "n't".

[49] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics, 1992

[50] H. Schmid. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of International Conference on New Methods in Language Processing*, volume 12 of *Studies in Computational Linguistics*, pages 44–49. Manchester, UK, 1994

[51] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. *Proceedings of the conference on empirical methods in natural language processing*, 1:133–142, 1996

[52] K. Hacioglu. A lightweight semantic chunking model based on tagging. In *Proceedings of HLT-NAACL 2004: Short Papers on XX*, pages 145–148. Association for Computational Linguistics, 2004

[53] J. Nivre, J. Hall, and J. Nilsson. MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, pages 2216–2219. Citeseer, Citeseer, 2006

text. For example, the task of Named Entity Recognition (NER) is to identify person names, location names and other types of proper nouns[54]. Traditionally the entities are known in advanced and stored in so called gazetteer lists. Recently machine learning algorithm have been applied on this problems.

Hearst Patterns Due to the unsatisfying performance of NLP components, alternative approaches have been used by many applications. For example, to extract information out of text, one can start with using manually assembled list of patterns[55]. This approach can then be enhanced by semi-automatic methods.

Stemming For to grammatical reasons many human language use inflections, which often alter the suffix of a word. As the semantic content does not change (or only slightly), one can remove such suffices and reduce the word to its stem. Various approaches for stemming have been proposed in the past[56,57]. Whether stemming does improve the performance of a knowledge discovery application can often only be assessed by trial and error.

To sum up, during the pre-processing phase the data-set is cleaned and prepared for further processing. Therefore the data-set is analysed and the output is stored alongside the data[58]. The individual algorithms being applied here depend on specific properties of the data at hand. Furthermore, this step is not completely decoupled from the succeeding steps, as changes in the cleaning phase might have implications for the performance of the overall system.

Finally the data is ready to be transformed into features. Until this point the data is organised in data-structures that match the requirements of the data-source and formats. The representation of the data as features is due to the requirements of the data mining algorithms. These algorithms hardly ever allow unstructured input, but require their input data to be structured in a specific manner. This is the task of the feature extraction phase and can be seen as part of the overall feature engineering process.

[54] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics ACL 05*, 43(1995):363–370, 2005

[55] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, volume II of *COLING '92*, pages 539–545. Association for Computational Linguistics Morristown, NJ, USA, Association for Computational Linguistics, 1992

[56] M. F. Porter. Snowball: A language for stemming algorithms, 2001

[57] C. D. Paice. Another Stemmer. *SIGIR Forum*, 24(3):56–61, 1990

[58] This step is also called enrichment, as the the data-set now contains additional information.

Feature Extraction

In the context of knowledge discovery applications the task of feature engineering starts with a pre-processed data-set and ends with a set of features, appropriate for the further processing using data mining algorithms. During this process, features are generated, analysed, and transformed. Before the task of feature engineering can be discussed, at first a few terms that relate to features need to be defined: feature space, feature set and instance.

A *feature space* is a general description of a type of feature using within a knowledge discovery application. The feature space restricts the possible range of values of its features. Thus each feature can only be related to just a single feature space.

A *feature set* is a collection of features, used to organise multiple features. Usually a feature set will contain features from just a single feature space. This is not a strict requirement, but it generally considered to be a best practice approach. Only few data mining algorithms can cope with features of different types. Thus mixing different features might hamper getting the optimal performance from the application.

An *instance* is a collection of feature sets. In many cases an instance will only consist of a single feature set. Furthermore, all instances of a data-set will typically contain the same set of features.

To sum up, in the initial feature generation process, the data-set will be transformed into instances and features. Often there will be a simple mapping scheme, where a single input document is represented as a single instance and the data contained within this document will be transformed into features stored within the instance. The exact scheme, how instances and features are generated naturally depends on the application and its algorithms.

Types of Features

There is not a single representation on how a feature is structured. But there are a number of common types of features. In general a single feature simply consists of a identifier and a value. The type of the identifier can be freely chosen, often a integer number of a character sequence is used. The value of a feature is usually one of these:

Binary The value of the feature can take one of two states, true or false. For example, each pixel in a black and white image can be transformed into a binary feature.

Nominal The value is one out of a given set of predefined values. Other names for this type of feature are: categorical, enumeration. For example, a weekday can have one out of a closed set of pre-defined values.

Numeric The feature value is a number, either a integer value or a floating point number. This type of feature value is the most common type.

But the possible type of values are not limited to the types presented here. For example, the value might just be a character sequence or even a complex data-structure. But in this cases the succeeding data-mining algorithms need to be capable to deal with this kind of features.

Example: Vector Space Model

The Vector Space Model (VSM) is one of the most important techniques on how to extract and manage features out of textual resources. The feature value types in this scheme are typically numeric.

In the following, this naive assumption is being made - all words that occur in the data-set are all know in advance. The collection of words is

called *vocabulary*. The feature space defines and describes the properties of its features. In the basic vector space model, the feature space restricts the feature identifiers to be one of the words from the vocabulary, and the feature values to be a positive integer.

Each document in the data-set is represented as a single instance. These instances contain a single feature set. From the documents the features are generated. For each unique word within the document a single feature is produced, and its value is the number of times the word occurs within the document. The sequence information of the words is lost in this process. The term bag of words is also used for this approach.

The feature space itself can also be seen as a high dimensional space, where each feature - unique words in this case - represents a single dimension. Thus the feature space will have as many dimension as there are words in the vocabulary. A single instance can therefore be seen as a single vector. Naturally the instances will be sparse, as document usually only contain a small subset of the complete vocabulary. In this case all missing dimensions are assumed to be zero.

Using this approach, it is easy to compare document with each other. The distance between the two points which represent two document is a common way, as is the angle between the vectors.

There are many extensions to the basic vector space model. For example, the dimensionality of the feature space does not need to be fixed. It is more common to incrementally increase the number of dimensions as new words are introduced.

The scheme, that is used to assign a value to each feature can be made more sophisticated. This process, often referred to as term weighting, may involve complex computations and is some application this step is essential to optimise the performance of the overall system.

Pre-Processing for a Vector Space Model The individual steps within a knowledge discovery application are often tightly linked. In the pre-processing step needs to be modelled to extract all necessary information which are required to build suitable features. For example, when a vector space model is build, simply using words will often not be sufficiently.

Instead of producing a single feature for each word, one can choose to build more complex features. A common approach to combine a sequence of multiple words as a single feature. In this case, feature reflect word n-grams[59]. The most common approach is to generate bi-grams, a combinations of two consecutive words.

[59] In this case, the number of dimension will be very high.

Often one does not want to include all words or bi-grams in the feature space. Only the features with high discriminator power should be generated, as the size of the feature space will have implications on the run-time performance of the whole system. A common approach is to eliminate the most frequent terms, as they do not carry semantics, but serve only a grammatical function. This can be either done by via manually assembled stop-word lists, or by computing the most frequent term in the vocabulary of the data set.

In the case of n-grams, a similar approach can be taken to avoid a combinatorial explosion of the number of dimensions. These n-grams are selected, that actually occur more often than their individual distributions would predict them to occur in a sequence by chance.

Example: Web-Graphs

Due to its vastness, the World Wide Web can no longer be efficiently be navigated without the help of web search engines. The task of a web search engine is to find the most relevant web-page for a given query. There are

many types of factor, that contribute to the relevancy given a specific query. Many of them are subjective, but some can be made more explicit.

The popularity of a web-page does in the many cases can be seen as an indicator for its relevance for a greater audience. Unfortunately the popularity cannot be measures directly, therefore it needs to be assessed in an indirect manner. The number of web-pages that link to a certain web-page can be seen as an approximation of its relevance. This is the base of the PageRank algorithm, which did help Google to improve its search results. The HITS algorithm is another well known representative of this family of algorithms.

In the case of PageRank, the incoming links are not directly counted, instead the originating web-pages of these links are also weighted. The relevance of web-pages is therefore propagated throughout the link graph.

To compute the PageRank, each web-page can be transformed into a instance. Its features are the links to the out-going links. The identifier of these features are the target-URL. The feature value can either be binary or may reflect the number of times a web-site is references[60].

For the PageRank algorithm the feature values need to be normalised. This normalisation needs to be done to ensure that the sum of all incoming links sum to one. This step is done during the feature transformation stage.

[60] Often external web-pages are only linked once, even if there are mentioned multiple times within the text.

Properties of Features

There is a high degree of variations in the properties of features for knowledge discovery applications. Still, there are a number of common properties that appear in many feature-spaces and their features. These common properties can be exploited to improve the quality of the results as well as to speed up the computations. The properties can be categorised into four groups:

- Number of features (size of the feature-space)
- Number of features per instance (size of the feature-sets)
- Distribution of the feature values
- Distribution between the feature values

Sparse vs. Dense A feature space can be classified as being either sparse or dense. This is an important property as it has an influence on which data-structures and algorithms are eligible for the data-mining task. If the feature-space is dense, there is a feature value for each feature of every instance. Thus if there are m features for a single feature-space and the data-set consists of n instances, the number of feature values will be $n \times m$. For many scenarios this number exceeds the available storage capacity making any processing infeasible.

Therefore it is desirable to have sparse feature-spaces. In this case the feature-set of the instances do not contain all features. Instead, the majority of features are missing. The data-mining algorithms need to be adapted to cope with missing features and feature values. It is important that the semantics for left out features is defined in advance. For example, for numeric features, a missing feature value is often assumed to be zero.

Even if the original data-set is dense, the feature-space will often be artificially kept sparse to allow efficient computations. For example, if the value of a feature falls below a pre-defined threshold, the feature is removed from the feature-set and treated as it were missing.

In other cases, the data-set is inherently sparse. A vector space model is commonly build by mapping each unique word to as a feature. As only a few words out of the complete vocabulary occur in a single document, the resulting feature-space will be very sparse. For textual data-sets one can expect less than 10% of all features to be linked to each instance[61].

[61] For example, each Wikipedia article contains on average less than 2% of the entire vocabulary.

Heaps' Law For many data-sets one can find a relationship between the number of features and the number of instances. The number of features does not grow linearly with the number of instances. Instead the growth of the size of the feature-space gets slower with a high number of instances.

Originally this relationship has only been observed for textual documents, where features are words and the feature-space is the vocabulary[62]. In this context the Heaps' law, the size of the vocabulary V in relation to the number of instances - n - can be formalised as:

[62] H. S. Heaps. *Information retrieval: Computational and theoretical aspects*, volume 60. Academic Press New York, NY, 1978

$$V(n) = Kn^{\beta} \tag{4}$$

There are 2 free parameters, K and β which depend on the data-set. For data-set consisting of English text documents, K has been found to be between 10 and 100 and β to be found in the range between 0.4 to 0.6. In figure 5 the growth of the vocabulary for the Brown corpus is given in relation to the number of sentences.

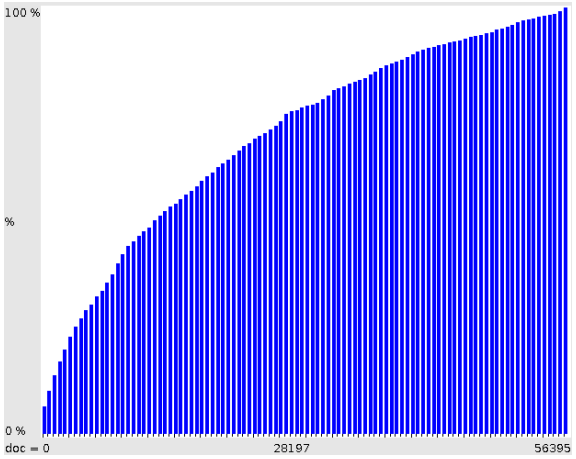

Figure 5: Growth of the size of the feature-space in relation to the number of instances for the Brown corpus. The x-axis reflects the number of sentences and the y-axis shows the number of unique words.

Zipf's Law, Power Law Another well known empirical law has been proposed by Zipf in 1935. It has been first discovered in copora of natural language. It only applies to sparse data-sets as it relates to the distribution of frequencies of features. The frequency of a feature is the number of times a feature occurs within a feature-set of an instance. In the case of documents and words, the frequency of a word is the number of documents in which the word does occur. Therefore this number is usually called *document frequency*. The number of times a word is using in total is usually referred to as *collection frequency*.

The list of all words can be sorted according to their frequencies, starting with the most frequent one. The rank of a word in this context is the position within this list. According to the Zipf's law, the rank is inverse proportional to its frequency. More formally, the document frequency df can be derived from its rank r and two free parameters α and β[63]:

$$df = \alpha r^{\beta} \tag{5}$$

[63] Strictly speaking, the Zipf's law only applies to cases, where $\beta = 1$, but it is has common practice to use the term for the whole class of distributions

By applying the logarithm, the relation between the rank and the frequency become more obvious:

$$\log df \propto \beta \log r \tag{6}$$

Thus, on a log-log plot the relationship between the rank and the frequencies is expected to be a straight line. In figure 6 the according plot is presented for the English Wikipedia. One can see that the Wikipedia data can be closely matched by two lines, where all terms up to rank 10,000 very closely follow the Zipf's law ($\beta = 1$).

The Zipf's law does not only apply to natural language resources. The relationship between a feature frequency and its rank can also be observed for other data as well. Probably the best know case of such a relationship has been published 1913 by Felix Auerbach. The population size of cities can be expressed using equation 5 (and a β of 1.07).

This type of feature distribution is also known a power law. A feature features are very common, and many features are highly infrequent. The term *long tail* has been established for the high number of low-volume features[64]. From a knowledge discovery point of view, the features, that can be found in the middle of the distribution, are often the most important. The most valuable features have a high discriminative power and a sufficiently high frequency at the same time.

[64] C. Anderson. The Long Tail. *Wired*, 12(10):170–177, 2004

The power law is related to the Pareto distribution, which is also known as the $^{80}/_{20}$ rule. For example, the distribution of wealth within a society

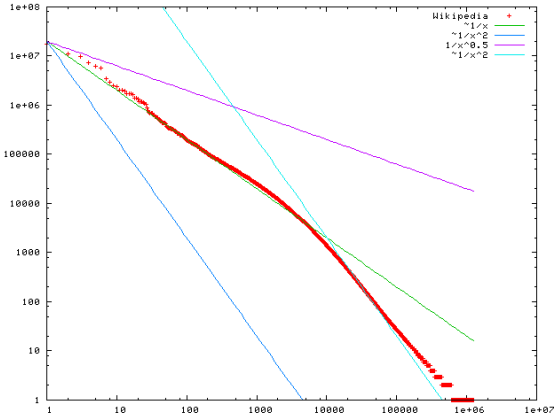

Figure 6: Word frequencies for the English Wikipedia as log-log plot. On the x-axis words are sorted according to their rank and the y-axis indicates the collection frequency of the words. [Copyright Victor Grishchenko]

is assumed to follow this distribution. The main property of a power law distribution is its scale invariance, also termed *scale free*.

Small World Phenomena The power law can also be found in networks, or graphs. The web graph, which consists of web-pages and their links in between, does also follow such a distribution. Within these graph there can be found other types of regularities, which apply to the relationships between the nodes within a graph.

In the context of features-spaces, one can observe that for many data-sets there are common patterns in the way features are connected to each other via instances. Among these patters is the so called small world phenomena, where each node within a network can be reached by any other node within a minimum number of hops, although most of the nodes are not directly connected. More generally, the typical distance between nodes d (mean shortest path) can be put into relation of the overall size of the network n[65,66]:

$$d \propto \log n \tag{7}$$

Within such networks, a number of certain phenomena will typically occur:

Cliques While the overall graph may not be highly connected, there are a number of sub-graphs, that are highly connected. These nodes tend to form clusters with a many connection within the cluster, but only a few that lead to the outside. For example, within a social network one can observe the emerging nature of communities of people.

Hubs Not all nodes are evenly distributed in regards to the number of connection with other nodes. There are nodes with a high degree of neighbour nodes. These nodes are called hubs and their presence is the main reason for the small path length between nodes within the network.

These properties pose additional challenges when applying knowledge discovery algorithms[67]. The algorithms and data-structures should be designed to allow efficient computation, especially within a distributed computing scenario.

[65] D. J. Watts. *Six Degrees: The New Science of Networks.* Vintage, 2004

[66] A.-L. Barabási. *Linked: The New Science of Networks*, volume 71. Perseus, 2002

[67] These data-sets will typically be globally sparse, but locally dense.

Feature Transformation

Directly after the features have been extracted, these features may be further processed, before the data-mining activities take place. This process is called feature transformation. The feature association framework will typically be invoked in this phase. To exploit the relationship between features is just one strategy that may be applied during feature transformation. Some selected alternative methods are briefly discussed.

Feature Value Normalisation

Some algorithms require their input values to be of specific types and to fall within a pre-defined range. For example, some algorithms may only cope with nominal feature values. Therefore all numeric values need to be converted first.

For this purpose data-mining algorithms may applied. For example, clustering algorithms can be applied for vector quantification[68]. Such an approach groups similar feature values into a smaller collection of coherent clusters.

[68] F. Curatelli and O. Mayora-Ibarra. Competitive learning methods for efficient vector quantizations in a speech recognition environment. *MICAI 2000: Advances in Artificial Intelligence*, pages 108–114, 2000

In reverse, some algorithms require a numeric feature values. While binary feature values are easily mapped to a numeric representation, for nominal features this may be a hard task. For example given the names of colours one could use the corresponding wavelength instead.

The optional normalisation step of the feature values also depend on the succeeding algorithms. The feature normalisation may require to scan over the whole data-set, which may cause this computation to be expensive. For example, if the feature values need to be normalised into a pre-defined range. Here the highest and lowest occurring feature value has to be determined. This is often the case, when feature values should represent probabilities. For instance the PageRank algorithm expects the weights of all incoming links to sum to 1. Therefore the corresponding features need to be normalised accordingly.

Another normalisation strategy applies to feature-sets that represent a vector within a vector space model. As the features represent a single vector within a high dimensional space it is common practice to apply a scaling to unit length. This allows the efficient computation of the cosine similarity[69], which is common for many knowledge discovery applications. The succeeding algorithms need to be made aware to make use of this property. This highlights the close relationship between the different stages of the knowledge discovery life-cycle.

[69] For two unit vectors, the cosine similarity can be computed via their dot-product.

Feature Weighting

The feature weighting is even more closer linked with the data-mining algorithms than feature normalising. In this case one tries to optimise the performance of the knowledge discovery process by applying a weighting scheme on the features. This method does only apply to numeric feature values. As with the feature normalisation, no new features are introduced by this approach.

The motivation for the weighting scheme is to boost features, which are supposed to be more helpful than others. For the feature weighting to be effective it has to work hand in hand with the data-mining algorithms.

Example: TF-IDF In the field of Information Retrieval feature weighting has a long tradition and is used in combination with the vector space model. In such a setting the individual features represent words as they occur in documents. The feature weighting is conducted in multiple steps. At first,

the number of times a word occurs within a document is counted. This term frequency builds the base for the first part of the weighting scheme.

Next the document frequency of the terms is collected, which contributes to the second part of the weighting. The number of documents is set in (inverse) proportion to the total number of document in the data-set. Finally the two frequencies are combined to form the TF-IDF[70] term weighting strategy. There are a multiple different ways on how this basic scheme can be applied. Their impact on the performance is typically assesses by conduction evaluation runs on test data-sets.

Another prominent example of a feature weighting algorithm rooted in Information Retrieval is BM25[71]. This retrieval function can be regarded as the state-of-the-art in ad-hoc Information Retrieval. Other feature weighting strategies are based on similar probabilistic approaches. They also integrate the average length of all instances within a data set[72,73,74].

Another set of feature weighting function stems from the field of corpus linguists. The Juilland-F[75] has been developed based on the observation that the occurrence of function words is relatively even across and within textual documents. Whereas content words tend to occur in bursts, so that when they occur, they occur often. Juilland combined the variance and the mean of the frequency of words to create a measure that reflects this property[76]. In recent times this intuition has been picked up again and various measures have been developed to exploit this[77]. Among these measures also the so called dispersion measure has been proposed, which has been successfully applied in a number of applications[78,79,80]. In the context of feature association this property of natural language has been used for model the re-occurrence of terms based on a Bayesian mixture model[81].

Feature Selection

In contrast to the two previous feature transformation strategies, in the case of feature selection, not individual feature values are modified, by the feature-space itself. Out of the set of features, a minimal sub-set of features is computed with still captures the main properties of the data-set. The feature values themselves are not changed in this process. Feature selection is motivated by three main reasons:

- Having fewer features might help to avoid over-fitting. This problem is mostly related to classification and regression problems. If the model, which is generated by machine learning techniques, to closely follows the training data, its predictive power may suffer.

- The assessment of the influence of individual features can prove helpful in the selection of algorithms and their parameter. Having a better understanding of the available data-set may contribute to optimise the quality of the results.

- The computation time can be considerably be shortened, as the search scope is reduced. This is especially important for algorithms that do not scale well with the number of features.

It is important to note that feature selection does not provide benefits for all data-sets and use-cases. Whether it is suited for a given setting needs to be assessed by conducting experiments.

Some classification algorithms already implicitly implement a feature selection stage. For example the family of decision tree algorithms[82,83] employ such strategies during the training phase. These type of algorithms are among the most commonly used classification algorithms[84].

[70] Term Frequency - Inverse Document Frequency

[71] S. Robertson and M. Gatford. Okapi at TREC-4. In *Proceedings of the Fourth Text Retrieval Conference*, pages 73–97, 1996

[72] G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. Inf. Syst.*, 20(4):357–389, Oct. 2002

[73] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '96, pages 21–29, New York, NY, USA, 1996. ACM

[74] K. W. Church and W. A. Gale. Poisson mixtures. *Natural Language Engineering*, 1(02):163–190, 1995

[75] Juilland A., D. R. Brodin, and C. Davidovitch. Frequency dictionary of french words. 1970

[76] This property is sometimes described as burstiness.

[77] S. Gries. Dispersions and adjusted frequencies in corpora. *International Journal of Corpus Linguistics*, 13(4):403–437, 2008

[78] R. Kern and M. Granitzer. Efficient linear text segmentation based on information retrieval techniques. In *MEDES '09: Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pages 167–171, 2009

[79] R. Kern and M. Granitzer. German Encyclopedia Alignment Based on Information Retrieval Techniques. In M. Lalmas, J. Jose, A. Rauber, F. Sebastiani, and I. Frommholz, editors, *Research and Advanced Technology for Digital Libraries*, pages 315–326. Springer Berlin / Heidelberg, 2010

[80] R. Kern, C. Seifert, and M. Granitzer. A hybrid system for German encyclopedia alignment. *International Journal on Digital Libraries*, 11(2):75–89, Sept. 2011

[81] A. Sarkar, P. H. Garthwaite, and A. De Roeck. A Bayesian mixture model for term re-occurrence and burstiness. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, CONLL '05, pages 48–55, Morristown, NJ, USA, 2005. Association for Computational Linguistics

[82] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986

[83] J. R. Quinlan. *C4. 5: programs for machine learning*. Morgan kaufmann, 1993

[84] G. Seni and J. F. Elder. Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions. *Statistics*, 2(1):1–126, 2010

For other algorithms one can conduct a feature selection stage as a separate task. Probably the best known approach to assess the usefulness of features is to compute the Information Gain. It can be formulated as the reduction in entropy for a given class by subtracting the conditional entropy. Here the feature is denoted as f and the class to predict by the classification algorithm as c:

$$IG(f, c) = H(c) - H(c|f) \tag{8}$$

The information gain values for the individual features can then be sorted and the top features are selected. In table 7 the features of the contact lenses data-set[85] are listed, according to their information gain values. To compare the impact of the feature selection, the J48 classification algorithm has been applied two times. The first time using all features and for the second run only the two top ranked features are used (tear-prod-rate, astigmatism). To assess the classification result, a 10-fold cross-validation has been conducted. The F-measure for the J48 classification algorithm rises from 0.84 for all features to 0.88 when using just two features.

[85] J. Cendrowska. PRISM: An algorithm for inducing modular rules. *International Journal of ManMachine Studies*, 27(4):349–370, 1987

Feature	Information Gain
tear-prod-rate	0.5488
astigmatism	0.377
spectacle-prescrip	0.0395
age	0.0394

Table 7: Ranked list of feature from the contact lenses data-set, sorted by information gain.

Another approach for feature selection is the correlation feature selection (CFS) measure. This method is based on the intuition[86]:

> Good feature subsets contain features highly correlated with the classification, yet uncorrelated to each other

[86] M. A. Hall. Correlation-based Feature Selection for Machine Learning. *Methodology*, 21i195-i20(April):17, 1999

A data-set containing cases of diabetes[87] serves as base for the demonstration of the CFS feature selection. By applying the feature selection, the size number of features drops from 8 to 4. When using the Naive Bayes classifier, again within a 10-fold cross validation, the F-measure rises from 0.76 to 0.77.

[87] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pages 261–265, 1988

The CFS feature selection approach indicates that the relationship between features may help to improve the quality of the output of knowledge discovery application. The feature association framework has been developed to allow an efficient analysis of the correlation between features and thus can be applied for feature selection.

Especially in the domain of textual features, where the number of dimensions can be very large, to single out the most distinctive features is essential for many algorithms. An overview of several different feature selections metrics for multiple textual data sets is given by Forman[88]. Furthermore there are a number of pruning strategies used for textual document with natural language features[89,90]. These are also applied as feature selection for classification problems[91].

[88] G. Forman. Choose Your Words Carefully : An Empirical Study of Feature Selection Metrics for Text Classification document Choose Your Words Carefully : An Empirical Study of Feature Selection Metrics for Text Classification. In *Proceedings of the 13th European Conference on Machine Learning (ECML '02)*, number August, 2002

[89] J. Fürnkranz. A study using n-gram features for text categorization. *Austrian Research Institute for Artificial Intelligence Technical Report OEFAI-TR-98-30 Schottengasse*, 3(1998):1–10, 1998

[90] L. Jing, H. Huang, and H. Shi. Improved Feature Selection Approach TFIDF in Text Mining. In *Proceedings of the First International Conference on Machine Learning and Cybernetics*, volume 4, page 5, 2002

[91] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *In Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, 1997

Especially for textual features there exists a list of relatively simple heuristics:

High Document Frequency Words should be removed that are contained in almost all documents. This is based on the observation, that function words tend to occur in nearly all sufficiently long documents. These words serve a grammatical function, but do not carry any semantics and thus are the prime candidates to remove from the set of features. Using the statistical properties of words is a common alternative to manually pre-compiled stop-word lists. There exist many examples

which demonstrate the usefulness of this approach, for example [92] where also low frequent words are pruned away.

Low Document Frequency Words, that only occur in very few documents, should be pruned. Very rare words also tend to be less useful, because they often arise from spelling errors, are noise or artifacts of text processing algorithms. For example, token splitting implementation sometimes do not correctly split two consecutive words and thus create an artificial compound word.

Punctuation As with words that serve solely a grammatical function, punctuation characters also play a minor role in knowledge discovery applications. Depending on the actual processing pipeline, punctuations may have been removed prior to the feature selection processing step. If they are still part of the feature set, they should be pruned away because of their frequent usage that cause many unnecessary calculations. Typically punctuations are detected by specialised parser components, for example the Brill tagger[93] and the TreeTagger[94]. The parser annotates all words with a part of speech tag. These tags vocabularies contain dedicated symbols for punctuations. For the English language the Penn Treebank Tagset[95] is the most popular choice.

Numbers & Time All words that are not listed in a dictionary are candidates to be removed from the feature set. Unfortunately a naive implementation of this scheme is not feasible. Named entities, like person and company names as well as location names, are not always found in common dictionaries. These named entities tend to play an important role in knowledge discovery applications[96,97,98], therefore leaving them out does not appear to be a valid option.

Other types of words, that are not listed in dictionaries, could be removed without reducing the usefulness of the knowledge discovery application. Numbers for example are used to quantify, but they do not carry a semantic information in the majority of cases. Additionally numbers can be easily detected using simple rule based approaches.

The second class of words that can be ignored are date and time specifications. The requirements to reliably detect dates and time are higher then for simple numbers, especially for complex constructs like date ranges. In fact, a workshop series[99] has been established not only to improve detection of temporal expressions, but also to extract events and discover temporal relations. For the task of pruning the features set without losing too much valuable associations between features, a simple approach to detect the most common date and time specifications seems sufficient.

Long Words Very long words are rare in natural languages and therefore they are likely to be artifacts of previous processing steps. The value of the length threshold for this type of heuristic depends on the actual language the documents are written in. Polysynthetic languages are a special class of languages where long words are commonly used. In so call agglutinating languages a new word can be composed by concatenating multiple words to create a new meaning derived from the meaning of the individual words. The most prominent example of an agglutinating language is Japanese, although there exist many more. Furthermore the is another group of language that uses long words, this is the family of fusional languages. In these languages not entire words, but morphemes are combined to construct words with a distinct meaning. Most of the European languages can be attributed as fusional languages, especially Latin, German and Dutch.

URLs Another simple but effective pruning strategy is to filter out all URLs, file names and other resource names. These kind of character sequence

[92] T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Machine learning: proceedings of the fourteenth International Conference (ICML'97)*, 1997

[93] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics, 1992

[94] H. Schmid. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of International Conference on New Methods in Language Processing*, volume 12 of *Studies in Computational Linguistics*, pages 44–49. Manchester, UK, 1994

[95] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):330, 1993

[96] T. Mandl and C. Womser-Hacker. The effect of named entities on effectiveness in cross-language information retrieval evaluation. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 1059–1064, New York, NY, USA, 2005. ACM

[97] W. Klieber, V. Sabol, M. Muhr, R. Kern, G. Öttl, and M. Granitzer. Knowledge discovery using the KnowMiner framework. In *IADIS International Conference Information Systems*, 2009

[98] W. Klieber, V. Sabol, R. Kern, M. Muhr, and M. Granitzer. Using Ontologies For Software Documentation. In *Proc Malaysian Joint Conference on Artificial Intelligence MJCAI2009*, 2009

[99] `http://timeml.org/tempeval/`

is likely to be introduced by the way digital documents are converted to their textual representation. Footnotes and headers commonly contain the name of the file on the local storage. HTML documents frequently contain of URLs that do not provide any benefit for the application. Like numbers these URLs and other resource references are relatively easy to detect and to extract.

Dimensionality Reduction

The aim of feature selection is to reduce the size of the feature-space by removing individual features. Dimensionality reduction also decreases the number of features, but in this case a completely new feature-space is spawned, which replaces the original one. As with feature selection, the goal of dimensionality reduction is to encode the most valuable information with the help of fewer features.

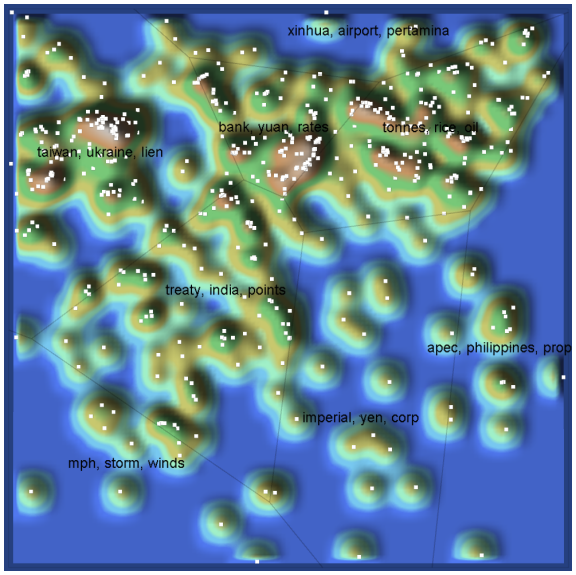

Figure 7: Example of a high-dimensional feature-space build from the Reuters-21578 data set, which is projected into a 2-dimensional space.

One reason for the popularity of dimensionality reduction is its use to visualise complex data sets. Often a high-dimensional feature-space is reduced to two or three dimensions and then visualised for inspection of a user.

The best known approach to dimensionality reduction is the principal component analysis (PCA). Given a square matrix, the axis of the highest variation are identified. These can be sorted and only the first dimensions are kept. The data-set is then projected onto the reduced space giving a new reduced feature-space. The PCA method for dimensionality reduction can be improved by applying non-linear approaches[100]. Singular value decomposition (SVD) is in many ways similar to PCA. But SVD can also be applied on non-square input matrices. Therefore this approach is suitable for a broader range of data-sets.

Another common technique is force directed placement (FDP)[101]. This approach models the forces of particles in physical space to layout the instances, for example gravity. Finally feature vectors which are close to each other in high-dimensional space will end up being close to each other in the low-dimensional projection as well. In figure 7 a sophisticated visualisation of a high-dimensional data-set is depicted[102]. This visualisation gives the feature engineer clues on the distribution of the instances within the data-set.

[100] B. Scholkopf, A. Smola, and K.-R. Muller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10(5):1299–1319, 1998

[101] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software Practice and Experience*, 21(11):1129–1164, 1991

[102] C. Seifert, V. Sabol, and W. Kienreich. Stress Maps: Analysing Local Phenomena in Dimensionality Reduction Based Visualizations. In *European Symposium Visual Analytics Science and Technology (EuroVAST)*, 2010

Language Models

Language models provide valuable information when dealing with features generated out of natural languages. These models can be build independently of the knowledge discovery application using frameworks like SRILM[103] or GIZA++[104]. Language models are typically build for a specific language and in some cases for a specific domain. Sophisticated algorithms have been build to exploit this information, for example in the field of information retrieval[105].

[103] A. Stolcke. SRILM-an extensible language modeling toolkit. In *Seventh International Conference on Spoken Language Processing*, volume 3, pages 901–904. Citeseer, 2002

[104] F. J. Och and H. Ney. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51, 2003

[105] J. M. Ponte and B. W. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998

Feature Associations

The opposite approach to dimensionality reduction is to create features in addition to the existing ones. This may sound counter-intuitive at first, but this step can already be seen as part of the data-mining process itself. Information, that is contained in the relationship between feature should be extracted and made explicit. The main motivation is to improve the overall quality of the results and to improve the traceability of the computations.

Recommender systems are an example where feature associations are an inherit part of the algorithms. Especially in the case of collaborative filtering, where the extracted associations can be directly mapped to recommendations.

Some of the algorithm used within recommender systems stem from the family of association rule mining algorithms. The best known representative of these algorithms is the apriory algorithm[106]. Alternative approaches try to learn first-order logic rules[107]. Among the use-case scenarios for association rule learning algorithms are finding patterns in the shopping behaviour[108].

[106] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Journal of Computer Science and Technology*, volume 15 of *VLDB '94*, pages 487–499. Morgan Kaufmann Publishers Inc., Morgan Kaufmann Publishers Inc., 1994

[107] P. A. Flach and N. Lachiche. Confirmation-Guided Discovery of First-Order Rules with Tertius. *Machine Learning*, 42(1):61–95, 2001

[108] `http://www.webcitation.org/63F1NhYFs` (accessed on 2011-11-16)

Word Co-Occurrences

Computing feature associations has a long tradition in the domain of computational linguistics and methods have been studied by corpus linguists. In the field of natural language feature associations are mainly restricted to word co-occurrences. Given a collection of textual documents, one tries to uncover regularities in the sequence of words.

Researching the relationship between words have been motivated by two well known hypotheses:

- The *distributional hypothesis*, first described by Harris in 1954, which states that words which tend to occur together are semantically related. Firth describes this intuition as "a word is characterised by the company it keeps".

- The *strong contextual hypothesis*, proposed by Miller and Charles in 1991, says that the more similar the contexts of words the more semantically related the words are.

Many approaches have been proposed in the past and here only a brief overview can be given. The mainstream approach to compute word co-occurrences is presented in pseudo-code as algorithm 1.

The different approaches to compute word co-occurrences mostly vary in the way, how the context is defined and how the association strength is computed.

[109] E. Terra and C. L. A. Clarke. Frequency estimates for statistical word similarity measures. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 165–172, Morristown, NJ, USA, 2003. Association for Computational Linguistics

Context The context in this setting defined as the text that surrounds an occurrence. The common approaches to construct a context can be classified as:

Fixed sized window The most common approach is to use a window of fixed size around the occurrence. Thus a fixed number of words are included in the context. The size of the window plays an important role on the association strength[109,110]. Different types of windows, where the influence of a word depends on the distance to the target words plays a role, have been studied as well[111].

[110] J. A. Bullinaria and J. P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510, 2007

[111] M. Patel, J. A. Bullinaria, and J. P. Levy. Extracting semantic representations from large text corpora. In *Proceedings of the 4th Neural Computation and Psychology Workshop*, pages 199–212. Citeseer, 1998

Sentence All words, that are found within a sentence are added to the context. The context will be of different size. Therefore measures need to be developed to allow the association strength to remain comparable[112].

[112] C. Willners and A. Holtsberg. Statistics for sentential co-occurrence. *Working Papers, Lund University, Dept. of Linguistics*, 2001

Algorithm 1 Overview of the typical computation of word co-occurrences.

Require: Textual Corpus - $\mathfrak{C}$
Require: Vocabulary of Words - $\mathfrak{V}$
  procedure BUILDWORDCOOCCURRENCES
    for all $w_i \in \mathfrak{V}$ **do** ▷ Iterate over all words
      $C_i \leftarrow \emptyset$ ▷ Initialise the context

      for all $d_k \in \mathfrak{C}$ **do** ▷ Iterate over all documents
        $O_{i,k} \leftarrow$ GETOCCURRENCES(w_i, d_k)
        for all $o_{w_i,d} \in O_{i,k}$ **do** ▷ Iterate over all occurrences
          $c_{i,k} \leftarrow$ CREATECONTEXT$(o_{w_i,d}, d_k)$
          $C_i \leftarrow C_i + c_{i,k}$ ▷ Add the context of each occurrence
        end for
      end for

      $A_i \leftarrow \emptyset$
      for all $\{w_j | w_j \in \mathfrak{V}$ and $w_j \neq w_i\}$ **do**
        ▷ Compute the association strength
        $a_{i,j} \leftarrow$ COMPUTEASSOCIATION(w_i, w_j, C_i)
        ▷ Based on the collected context
        $A_i \leftarrow A_i + a_{i,j}$
      end for

      YIELD(w_i, A_i) ▷ Report co-occurrences for w_i
    end for
  end procedure

Discourse To reliable detect a discourse structure of a document is still an active research topic. Thus the discourse structure is approximated by including a fixed number of sentences that precedes and succeeds the occurrence are included in the context.

Document Often textual corpora are already organised in documents, where each of them is focused on a single topic. In such a setting, a context that contain the complete document may be the best choice.

Which type of context is best suited depends on the corpus and the method to compute the association strength between co-occurring words.

Association Strength To measure the association strength between co-occurring words researchers have proposed a huge variety of functions. From these different measures, no single function has emerged which can be considered to be the best overall choice. As with the definition of the context, the function to compute the association strength does depend on the data-set and the application scenario.

Many different functions will be covered in a later chapter in detail. Furthermore, comparisons between different functions can be found in the literature[113,114]. Therefore just an enumeration of some of the most popular choices is given here:

Similarity measures Measures that have been traditionally used in many knowledge discovery applications. For example: Cosine similarity, Jaccard coefficient[115], Dice coefficient

Distance metrics Metrics that measure the distance between two points in (typically high-dimensional) space. For example: Euclidean distance[116], city-block distance[117]

Statistical significance tests Measures that have been developed in the field of statistics to assess whether two outcomes are significantly different. For example: χ^2, poison significance measure[118,119], Odds ratio[120]

Information theoretic measures Differences between distributions based on methods to assess the amount of information. For example: Mutual information[121], Kullback-Leibler divergence[122], Jensen-Shannon Divergence[123] and pointwise mutual information[124]

The impact of combining multiple functions has been studied as well[125]. Another approaches utilise the web as corpus and web search engines to compute the similarity between words[126].

Corpus Size The influence of the size of the corpus on the quality of the extracted association has been studied. When computing the association strength between features it is crucial to have sufficiently many samples. If only a few co-occurrences are available in the corpus, the statistics will be less reliable.

It has been found, that an increase in size of the corpus also correlates with an improvement in the quality of the results[127,128]. At a certain point, adding more information does not longer contribute much to an increase in performance.

First and Second Order Co-Occurrences The algorithm listed in 1 directly puts a target word with its surrounding words in context. Thus the associations created by this approach will be restricted to words that jointly occur in documents. Their association strength will be according to their shared distribution. This approach does follow intuition behind the distributional hypothesis. Word associations based on this algorithm can be categorised as first order co-occurrences.

[113] P. Pecina and P. Schlesinger. Combining association measures for collocation extraction. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 651–658, Morristown, NJ, USA, 2006. Association for Computational Linguistics

[114] S. Evert. *The statistics of word cooccurrences: word pairs and collocations*. Stuttgart, 2005

[115] L. Lee. Measures of Distributional Similarity. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 25—-32, 1999

[116] U. Quasthoff and C. Wolff. The poisson collocation measure and its applications. 2002

[117] R. Rapp. The computation of word associations: comparing syntagmatic and paradigmatic approaches. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, number 1992, pages 1–7. Association for Computational Linguistics Morristown, NJ, USA, 2002

[118] U. Quasthoff and C. Wolff. The poisson collocation measure and its applications. 2002

[119] S. Bordag. A comparison of co-occurrence and similarity measures as simulations of context. In *Proceedings of the 9th international conference on Computational linguistics and intelligent text processing*, CICLing'08, pages 52–63, Berlin, Heidelberg, 2008. Springer-Verlag

[120] W. Lowe and S. McDonald. The direct route: Mediated priming in semantic space. In *Proceedings of the 22nd Annual conference of the Cognitive Science Society*, pages 675–680. Citeseer, 2000

[121] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990

[122] I. Dagan, F. Pereira, and L. Lee. Similarity-based estimation of word cooccurrence probabilities. In *of the 32nd annual meeting on*, pages 272–278, 1994

[123] G. Hirst. Distributional Measures as Proxies for Semantic Relatedness. 2005

[124] E. Terra and C. L. A. Clarke. Comparing query formulation and lexical affinity replacements in passage retrieval. In *Proceedings of the ACM-SIGIR workshop on methodologies and evaluation of lexical cohesion techniques in real-world applications (ELECTRA 2005)*, pages 11–17. Citeseer, 2005

[125] P. Pecina and P. Schlesinger. Combining association measures for collocation extraction. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 651–658, Morristown, NJ, USA, 2006. Association for Computational Linguistics

[126] P. D. Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the twelfth european conference on machine learning (ecml-2001)*, pages 491–502, 2001

[127] J. A. Bullinaria and J. P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510, 2007

But the strong contextual hypothesis is not addressed by this procedure. The direct word co-occurrences can be seen as a fingerprint of a word, but they do not yet allow a comparison between the contexts of two words. Thus additional computation is necessary. The computation of the the second order co-occurrences is based on the output of the first order co-occurrences. Words associations for each pairs of word are transformed into a representation which allows comparisons. Thus the fingerprints of two words are compared. Following the strong contextual hypothesis, word pairs with a high similarity can be considered to be semantically similar.

The influence of the order of the co-occurrence on the type of similarity has been studied, making the distinction between paradigmatic and syntagmatic models[129,130]. Co-occurrence of higher order have also been studied in the past and their usefulness has been hinted[131,132].

Applications of Word Co-Occurrences Of course, word co-occurrences are not only computed to test linguistic hypotheses. Feature associations in the form of word co-occurrences have been used in many application scenarios. Only a small fraction of these are covered here to give an impression of the versatility of the information contained in the relationship between features.

Information retrieval In a typical information retrieval scenario a user issues a query to a search system. This query will usually contain just a few words. Therefore there will be a low semantic redundancy and much of the desired information is expected to be encoded in the relationship between the words. Naturally this domain has been addressed by multiple approaches. For example: Query expansion[133], query processing[134], passage retrieval[135].

Natural language processing Various tasks in the area of processing human language appear to profit from the added information gained by co-occurrences. For example: Keyword extraction[136], thesaurus generation[137], text summarisation evaluation[138], collocation extraction[139], word-sense disambiguation[140,141]

[129] R. Rapp. The computation of word associations: comparing syntagmatic and paradigmatic approaches. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, number 1992, pages 1–7. Association for Computational Linguistics Morristown, NJ, USA, 2002

[130] B. Riordan and M. N. Jones. Comparing semantic space models using child-directed speech. *Entropy*, 20:200, 2000

[133] J. Xu and W. B. Croft. Query expansion using local and global document analysis. *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 96*, (Zurich, Switzerland):4–11, 1996

[134] E. Terra and C. L. A. Clarke. Scoring missing terms in information retrieval tasks. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 50–58. ACM, 2004

[135] E. Terra and C. L. A. Clarke. Comparing query formulation and lexical affinity replacements in passage retrieval. In *Proceedings of the ACM-SIGIR workshop on methodologies and evaluation of lexical cohesion techniques in real-world applications (ELECTRA 2005)*, pages 11–17. Citeseer, 2005

[136] Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157–170, 2004

[137] H. Schütze and J. Pedersen. A cooccurrence-based thesaurus and two applications to information retrieval. *Information Processing & Management*, 33(3):307–318, May 1997

[138] C. Y. Lin and E. Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 71–78. Association for Computational Linguistics, 2003

[139] P. Pecina and P. Schlesinger. Combining association measures for collocation extraction. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 651–658, Morristown, NJ, USA, 2006. Association for Computational Linguistics

[140] D. Freitag, M. Blume, J. Byrnes, E. Chow, S. Kapadia, R. Rohwer, and Z. Wang. New experiments in distributional representations of synonymy. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 25–32. Association for Computational Linguistics, 2005

[141] A. Sarkar, P. H. Garthwaite, and A. De Roeck. A Bayesian mixture model for term re-occurrence and burstiness. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, CONLL '05, pages 48–55, Morristown, NJ, USA, 2005. Association for Computational Linguistics

Data-Mining Algorithms

The feature association framework allows the researcher to customise the association computation. One possible application of this functionality is to apply the same methods to each association context, which are traditionally used for the whole data-set. These methods often operate on an input matrix, which in many cases is a matrix representing documents and terms. Within the computation of the feature associations, one can create a similar matrix, but in this scenario not a single matrix for the whole data-set is created, but a dedicated matrix for each feature. These matrices will typically be smaller and consist only of these entries of the data-set, which have a relation with the feature in focus. In the following, a number of algorithm will be presented, that are currently considered to be state-of-the-art. These approaches can then be re-used for the contextual matrices created during the association computations.

Latent Dirichlet allocation (LDA)[142] is one of the methods and has gained popularity in recent years. This method is related to clustering in a sense that LDA can be seen as a fuzzy co-clustering method, where both the instances and the source features are clustered at the same time. The output of the LDA algorithm is a set of topics. Topics are connected to the source features as well as instances via probabilities. See table 8 for an example of 5 topics and the top ranked features[143]. To generate an association graph these topics can serve as output nodes. All source features with a topic probability that exceed a certain threshold are connected to the corresponding output nodes. The LDA method is a member of the family of mixture model based algorithms. A similar approaches is the pLSI method[144], which has been developed to create a probabilistic alternative to the Latent Semantic Indexing[145] method.

[142] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003

[143] M. Steyvers and T. Griffiths. Probabilistic Topic Models. *Handbook of latent semantic analysis*, 427, 2007

[144] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999

[145] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990

Topic #247	**Topic #5**	**Topic #43**	**Topic #56**
drugs (.096)	red (.202)	mind (.081)	doctor (.074)
drug (.060)	blue (.099)	thought (.066)	dr. (.063)
medicine (.027)	green (.096)	remember (.063)	patient (.061)
effects (.023)	yellow (.073)	memory (.037)	hospital (.049)
body (.019)	white (.048)	thinking (.030)	care (.046)

Table 8: Examples for the features (in this case words) with the highest probabilities for four topics. The TASA corpus is used as input data set. For each topic the top 5 words are selected with their probabilities in brackets.

The base of the Latent Semantic Indexing approach is the decomposition of a matrix into its singular values, which is usually referred to as Singular Value Decomposition (SVD)[146]. Similar to the Principal Component Analysis (PCA) the SVD finds an orthonormal base that indicates the orientation of the largest variance within the data set. In the past the SVD has been already been employed for the task of feature engineering and related applications[147,148]. Formally the SVD is defined as follows, where M is the input matrix for n instances and m source features:

$$M_{n\times m} = U_{n\times n}\Sigma_{n\times m}V^*_{m\times m} \tag{9}$$

For many application is it not necessary to calculate all singular values. Instead of calculating the full $\Sigma_{n\times m}$ matrix, a rank approximation can be done which effectively results in a dimensionality reduction. The reduction of dimensions is achieved by removing redundancies between correlating features. If two features share a common distribution among the instances, they can be represented by a single dimension in the reduced space. In the case of textual features the reduced SVD should effectively resolve synonymous relationships between words because all words that carry the same meaning should be represented as a single vector within the reduced

[146] G. E. Forsythe, M. A. Malcolm, and C. B. Moler. *Computer methods for mathematical computations*. Prentice Hall Professional Technical Reference, 1977

[147] H. Schütze and J. Pedersen. A cooccurrence-based thesaurus and two applications to information retrieval. *Information Processing & Management*, 33(3):307–318, May 1997

[148] J. A. Bullinaria and J. P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510, 2007

feature space. This compression of the feature space should also remove noise and reduce the impact of outliers. The number of features is reduced to a lower number of dimensions k[149].

$$M_{n\times k} = U_{n\times n}\Sigma_{n\times k}V^*_{k\times k} \tag{10}$$

See figure 8 for a visualisation of a matrix which has undergone a dimensionality reduction starting with a few thousand dimensions to only three dimensions. All instances have been projected to vectors in the low dimensional space and then projected onto the unit-sphere. The generate this visualisation the ConceptNet[150] corpus was used, from which all sentences that contain the term '`instrument`' were taken. Many of the point lie close to each other, which can be seen as indicator, that these instances share many features and thus are similar to each other. Each of these groups of closely related instances resemble individual senses of the word '`instrument`'. For example individual groups represents string instruments, wood instrument and medical instruments.

[149] Finding the optimal number of dimensions for a given scenario is a hard task and often cited as *curse of dimensionality*.

[150] `http://www.media.mit.edu/~hugo/conceptnet/`

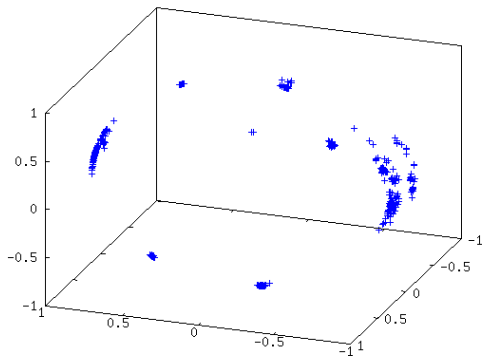

Figure 8: Visualisation of an approximation of a $Matrix_{instances\times features}$ after the number of dimensions have been reduced to 3. Each of the three axis represents one of the three main dimensions of variations within the data set. Each dot in the image represents a single instance as projected onto the unit-sphere of the reduced feature space.

The Singular Value Decomposition not only provides a approach to improve the quality of data and to extract latent structures within the data, there have also methods developed to improve the efficiency of the necessary calculation. Therefore the SVD is suited for even large scale data sets. There exist approaches to efficiently calculate the singular values for sparse matrices[151,152], which is common for many real-world data sets.

The input of the SVD is a matrix build out of the instances within the current context and the source features. Additionally the number of dimensions of the reduced feature space must be determined. The result of the Singular Value Decomposition must then be transformed into an association network. One of the ways to create this graph is to build a single target node for each dimension. For each row in the matrix the dominant dimension is selected, which represents the main characteristic diversion from the average of all instance within the current context. Thus each dimension in the reduced feature space is related to a set of instances. This corresponds to the output of a cluster algorithm, where each cluster consists of a set of instance. Because of this similarity the final association network can be generated the same way for both approaches. The output of the SVD based associations is therefore similar to the example shown in figure 15, but instead of using the clusters, in this case the dominant dimension is employed as means to partition the data set.

As alternative to using the dominant dimensions one could partition the input space, as this approach has already been proposed in the area of clustering of high-dimensional data-sets[153].

[151] M. Brand. Fast online svd revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining*, pages 37–46, 2003

[152] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, 2006

[153] B. L. Milenova and M. M. Campos. O-cluster: scalable clustering of large high dimensional data sets. 2002

Besides the similar input and output data-structures, there is another connection between the Singular Value Decomposition and clustering algorithms. The family of spectral clustering methods represent algorithms that incorporate graph based statistical methods[154]. In this case the input data to be clustered is represented as a graph. Various properties of this graph can then be analysed. The spectrum of the graph is an important source of information on how the data is structured. To exploit this usually the eigenvalues and eigenvectors of the graph are calculated. Alternative to using the eigenvalues the singular values can be used to uncover the intrinsic structure of the data. Various methods based on this idea have been published in recent times[155,156] and can easily be integrated into the calculation of feature associations.

[154] U. V. Luxburg. A Tutorial on Spectral Clustering. Technical Report March, MaxâĂŞPlanckâĂŞInstitut für biologische Kybernetik, 2007

[155] I. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556. ACM New York, NY, USA, 2004

[156] S. Y. Jianbo, S. X. Yu, and J. Shi. Multi-class Spectral Clustering. In *In International Conference on Computer Vision*, pages 313–319. In International Conference on Computer Vision, 2003

Concepts

The work-flow of feature association computation can be split into two phases. At first the associations between features need to be calculated. Then the results of the analysis is made available for succeeding processing within the knowledge discovery application.

Overview

In this chapter the feature association framework is presented from an conceptual point of view. Starting with the use-cases a number of design decisions are derived. The input and output data-structures are defined, as well as the configuration settings. The configuration determines which collections of features are selected to build the associations.

The majority of this chapter is reserved for the detail analysis of the inner workings of the feature association algorithm. Because of the complex requirements and the necessity to to able to process large scale data-sets, the design of the algorithms plays an important role. The basic idea to achieve the desired properties was the decision to split the algorithm into a set of distinct phases. Each of these phases serves a dedicated task and enables task-specific optimisations. The algorithm allows the execution in parallel and within a distributed execution environment. Therefore the feature association framework can be applied on large data-sets with complex feature interactions.

The role of the association function is to capture the semantics of the relationships between the features. The properties of this function relies on the requirements of the application. The feature association framework already provides a set of feature association functions, which have proven to be useful in the context of knowledge discovery applications. This list can be further extended by custom functions, which are tailored towards specific application needs.

The ability to execute the feature associations within a distributed environment is an important prerequisite for processing large data-sets. Therefore the integration of the feature association calculations into established distributed execution framework is demonstrated. The result of the feature association calculations can finally be retrieved and traversed using a dedicated component.

In this chapter at first a number of design decisions are presented. Next the input and output data-structures are described, followed by a detailed overview of the individual processing phases. The anatomy of the feature association function, as well as a list of available functions are reported. The possible modes of traversal within the generated output data-structures will be discussed. Finally the suitability to execute the feature association computation within a distributed environment is demonstrated.

Configuration & Design Decisions

Derived from the real-world use cases the development of the feature association framework is guided by a list of 10 design decisions. These are made explicit in the following section.

Complexity of the Configuration

The feature association framework offers an array of possible way on how features are put in relation to each other. This flexibility is necessary to allow the framework to be used in a wide spectrum of knowledge discovery applications. The diversity of applications is coupled with a variety of possible features and characteristics. In order to offer a unified approach to calculate and analyse the features associations, the configuration settings must be as able to cope with this flexibility. Depending on the actual application and data-set, the framework can be adapted at various levels. For the majority of real-world scenarios the task of configuring the framework should be as simple as possible. If the application is more demanding in its requirements, then deeper knowledge of the internal workings of the framework needs to be assumed.

Design Decision 1 *To configure and apply the feature association framework it should be easy for simple tasks and possible for complex tasks.*

According to this decision the input data-structure, the output data-structure and the configuration need to be defined.

Input Data-Structure

At first the input data-structure for the feature association calculations should be found. A number of requirements serve as starting point for the decision, which data-structure is best suited to represent the features of a data-set.

- This data-structure should be flexible to allow a wide range of different features to be processed.
- There should be no limitation on the size of the input data, to allow even large scale data sets to be analysed.
- The input data-structure should be conceptually simple and easy to use. The last requirement on the data-structure is rooted in the motivation that the framework should be easy to adapt and integrate into existing knowledge discovery solutions. Therefore it should be possible and sufficiently easy to transform any existing data-structure to a representation which can be immediately used as input for a feature association calculation.

The first requirement certainly is the hardest to fulfil, and appears to contradict the requirement, that the input data-structure should be easy to understand and to use. The data-structure has to be generic and should not pose too many restrictions on way features are represented. When considering all these requirements, the graph data-structure appears to be suitable to model the input of the feature association calculation.

Design Decision 2 *The input data-structure for the feature association calculation is a graph. Each node and each edge in this graph may carry any number of additional meta-data, for a example a weight.*

This data-structure is consecutively referred to as **feature input graph**. The framework poses no restrictions on the properties or the size of this graph. It is expected that in the most cases this graph will be an n-partite, weighted and directed graph[157]. As this type of graph is the most commonly used as input to calculate the feature associations, this type will be used to describe the configuration of the role mappings.

Example: When computing word co-occurrences, the documents are represented as nodes in the input graph. The words within these documents are as well represented as nodes. Each occurrence of a word within a documents yields an edge between the corresponding nodes.

[157] The input data-structure of many existing machine learning algorithms can be directly transformed into this kind of graph.

Role Mappings The next required step to adapt the framework for the problem at hand is the mapping of the nodes in the feature input graph to a set of predefined roles. By using roles and a mapping of these roles to the input data-structure a decoupling of the semantics from the data itself can be achieved. This is done because there should be as little limitations as possible on how the input graph is structured. The downside of this approach is the need to separately define which nodes are assigned to the specific roles.

The roles present the different types on how a node is interpreted and processed by the framework. For the input graph, these roles for the nodes are defined:

Source Feature All nodes that are assigned to the role "Source Feature" are treated as origin of a feature association relation. Starting with a single source feature node the algorithm will traverse the input graph to find associated features.

Target Feature The "Target Feature" nodes are candidates for targets of a feature association. Only nodes marked as target feature will be included in the calculation of possible matches for a given source feature.

Instance All nodes mapped as "Instances" are used to find connections between the source feature nodes and the target feature nodes. Only pairs of source and target feature nodes that share at least a single instance node are considered as candidates to be associated. Many functions that calculate the weight of an association take the number of shared instance nodes as an evidence of the strength of the association. The term "instance" is chosen because in many cases the nodes that are mapped to this role represent individual instances within the original data-set[158].

[158] The term "connector node" would also be suitable as it describes the function of these nodes, but it is assumed that the developers of knowledge discovery applications are more familiar with the concept of an instance.

Each node can be assigned to one or more of the three roles. Typically this mapping is not done on the node level, but on partitions of the graph. For example a tripartite graph can be mapped to the three roles by assigning each of the three partitions to one of the roles.

A typical knowledge discovery application operates on a data-set that consists of a set of instances and one or more feature sets[159]. The natural mapping of such a data-set is to build a feature input graph, where each instance and each individual feature is represented by a single node. For each occurrence of a feature within an instance, the two corresponding nodes a connected via an edge. If the occurrence is annotated with an weight, this weight is then attributed to the edge. Other meta-data can also be directly added to the edge and is made available to the algorithms responsible to calculate the associations between the features.

[159] Alternatively the term feature space can be used as label for all features of a specific type.

Example: To compute word co-occurrences within documents, one may start with a bi-partite graph where documents are nodes from one partition and word make up the other partition. The final goal is to find associations between words based on the frequency they appear jointly

within documents. Here the nodes which represent words are mapped to the source and target feature roles. The document nodes are mapped to the instance role. In the process of feature association computation all the word nodes that are connected via at least a single document node are then candidates for feature associations.

Output Data-Structure

For the output data-structure the requirements are similar to the input feature graph. The framework should be able to offer a wide range of possible output representations. It should be possible to enrich feature associations with additional meta-data.

In the most basic configuration a feature association is simply a tuple of two features and an optional weight. More complex application are more demanding and require a richer set of output data. For example it should be possible to model a conditional feature association, where a single association depends on an external state or context.

Design Decision 3 *The output data-structure for the feature association calculation is a directed graph. Each node represents a feature and each edge represents an association between features. Edges may carry a weight, which encodes the strength of the association, and may be annotated with meta-data.*

This data-structure is named **output feature association graph** in the following sections.

Role Mappings As with the input graph, there are also a set of roles for the output data-structure:

Source Feature A nodes marked as source feature serves as the tail of an directed feature association edge. If the feature association network is traversed, it may only start at an node mapping with the source feature role.

Target Feature The head of a feature association relation is marked as target feature node. Starting with a source feature, all edges are connected to a target feature node.

In contrast to the roles of the input feature graph, the rules of the output graph do not need to be externally mapped. There is no need to configure these roles within output graph because there is a direct relationship between the feature nodes of the two graphs. Each feature node in the output graph usually corresponds to a single node in the input graph. In a more complex application setting there might not be a direct relationship between the nodes of the input graph and output graph. For example multiple target features in the input graph are merged into a single feature node in the output graph. Still even in such a scenario the roles are assigned by the feature association algorithms and there is no need for an additional configuration.

Example: In the case of word co-occurrences the output graph solely consists of nodes which represent words. The document nodes of the input graph are no longer needed in the output data-structure. As for the roles, in the output graph each nodes is both a source feature, as well as target feature. Hence, a single word may serve as source or as target of a feature association.

Contextual Features Besides nodes that represent the source and target features, the output feature association graph may also contain additional nodes. The semantics of these nodes depends on the application and the

custom processing steps. This is common for applications that use a set of nodes to represent some sort of context. Examples for contexts are temporal or causal relationships between the feature associations. The presence of a third feature in combination with a source and a target feature could also serve as a context.

Feature Association Function

The overall task is to find which features are associated with each other. In the most cases this relationship is not binary. There should also be weight, which represents the strength of the association relation. An example of such weight is the similarity of two features according to their distributions over the instance nodes.

The selection of which function should be used to calculate this weight depends entirely on the application. Because of the wealth of possible association functions, the limitations imposed on these function should be minimised. Any restriction on the way which functions might be used to calculate the weight between features directly harms the general usefulness of the whole feature association framework. Therefore the algorithms should be open to allow a wide range of feature association functions.

On the other hand, the computation of the association weight should be as efficient as possible, as the execution time of this function has a huge influences on the run-time of the whole process. The feature association function is invoked for all pairs of features and as any feature might be associated with all other features in a data set, the run-time complexity is bounded by $\mathcal{O}n^2$ for n being the number of features. Therefore any long running operation is therefore prohibitive for most of the real-world data sets.

For example a feature association function that queries an external resource[160] for each pair of features will usually yield a bad execution time. To resolve the conflict of efficient computation and the possibility to incorporate additional information into the computation of feature associations, strategies to combine these requirements have to be developed.

[160] For example a complex look-up in a database

Starting point for the efficient integration of complex association functions is the observation that in many cases the individual factors that contribute to the final association weight only depend on one of the two features. If the feature association function can be decomposed into a form, where the majority of factors depend only on a single feature, the computation be done for more efficiently. By following this approach the upper bound of the run-time complexity is decreased to $\mathcal{O}2n$, which can be considered an remarkable improvement, especially for feature rich data sets. The second advantage of this approach is that it allows to pre-compute many of the factors in advance, as they no longer depend on the dynamic relationship between features. Factoring out parts of the computations offers a number of advantages:

- By computing some of the factor in advance, the operations can be executed in bulk. Many systems can be more effectively used if the operations are not invoked individually, but if many operations of the same kind are bundled into one big operation. This is the case for many database systems, which the query execution plan has to be scheduled and locks need to be setup to ensure atomic access for any operation.
- Because of the fact that the computation of a single factor and the computation of the whole association weight is decoupled, the execution sequence of the factor computations can be reordered. In many cases a sequential access to the data is far more efficient then a random on-demand access. This behaviour is rooted in the way contemporary computer systems are build. As the data is accessed in the same sequence

as it is stored, many caching and read-ahead mechanisms are exploited to ensure an optimal run-time behaviour. Thus a carefully crafted application will make use of the full potential of modern computing architectures by choosing the matching execution strategy.

- In a distributed scenario the communication overhead might become the major limitation. This is one of the reason why all necessary calculations should be executed as early as possible. An application that is fully deployed and executed on multiple nodes within a parallel execution framework, needs additional synchronisation to avoid overlapping and redundant calculation might lead to reduced performance[161]. If the some of the factors of a feature association function are pre-computed, no such synchronisation is necessary. The pre-computed data just needs to be distributed to all participating execution units and made available for the association function.

[161] This phenomena is usually referred to as starvation.

The decomposition of the association function and pre-computing of selected factors not only positive side effects. There are downsides to the approach which are twofold:

1. All feature association functions need to be decomposed into independent factors. This might be a tedious work for some functions, and even impossible for other function. For users not familiar with the inner workings of the feature association framework and not aware of the involved performance issues this could prove be a hard task.

2. There is a risk of pre-computing too many factors. For some feature association functions not all factors are needed for any given pair of features. If the factors are pre-computed there is no way to know in advance which factors are consecutively needed. Thus the computation and the distribution of these factors could be spared if the factors were computed on demand.

To overcome the first disadvantage, the framework provides a number of already decomposed feature association function. A wide array of various similarity function, statistical tests and entropy measures have been transformed into a representation usable for the feature association framework and will be discussed in detail later in this chapter.

Design Decision 4 *For each pair of source and target features a weight can be calculated. The function to calculate this weight should be flexible and efficient at the same time. To optimise both contradicting goals, the function should be decomposed into various factors, which can individually computed efficiently. A set of common feature association functions are already provided by the framework by default.*

The decomposed form of the function which calculates the association weight between two features is called **feature association function**.

Example: In the case of word co-occurrences, the conditional probability might be an appropriate choice. The feature association weight between two words would then be the occurrence probability of the target word, given the source word already occurs in a document.

Heuristics

For large data-sets with many features the computation may take a long time caused by the huge amount of potential feature associations. Fortunately most of the real-world applications do not require an exhaustive list of all associations. Many relationships might be spared as they not contribute much to the final result. Only the associations with the highest

association strength are needed as one can assume that they already capture the most relevant information.

In the general case, there is no way to know which associations are the most relevant in advance. Still one may apply some strategies, like for example some feature might be excluded entirely from the feature association calculations. Rare features and high frequent features are candidates to be pruned, as they tend to carry little statistical evidence.

The actual pruning strategy and the employed thresholds depend on the application and the data set. For similar features that represent similar properties one can reuse the same threshold for multiple scenarios.

Design Decision 5 *The feature association framework should be equipped with facilities to prune the feature input space. This should reduce the execution time while not affecting the usefulness of the results. To achieve an optimal compromise the feature association framework should provided ways to fine-tune the pruning steps. For common feature types the framework should provide a set of predefined thresholds to lessen the burden on the configuration effort.*

Example: For word co-occurrences it might not necessary to include words which only serve a grammatical purpose, for example "the", "for", "and". Therefore these words might be skipped all-together. As these words are also among the most frequent, as simple heuristic can be applied, which help to reduce the computation run-time without hurting the quality of the results.

Additional Customisation

For more demanding application scenarios the need arises to further tweak the default behaviour of the feature association computations. For example an application might choose that the output feature association graph should contain not a single target feature node for each input node, but multiple nodes. This is might be motivated to capture different aspects of a feature, which is the case for many real-world data sets.

To make a full customisation possible, the framework allows advanced user to inject additional algorithms into the processing pipeline. Because the need for complex task is regarded as needed only for a minority of use cases, it is completely optional. If such customisation is required a deep understanding of the inner workings of the framework is a prerequisite.

Design Decision 6 *For the majority of use cases, the configuration should be easy. For complex applications that require a specialised processing, it should be possible to customise the algorithms and add application-specific processing steps. Therefore the feature association computations should support additional custom processing steps within the execution of the calculations.*

Example: For the computation of word co-occurrences one might want to add an processing step for each word, for example apply a unsupervised machine learning algorithm. For example, all documents in with a source term occurs in might be clustered to find separate, distinct topics. Then the word co-occurrences are specific to these topics.

Summary of the Configuration

To demonstrate the necessary configuration effort, two extreme scenarios will be given. The first scenario requires minimal effort and represents the majority of real-world use cases. Thereafter a more complex scenario will be presented, which requires a more demanding configuration assembled by an expert user.

Simple Configuration For the most basic configuration only a minimal number of parameters are to be configured:

- **Input Feature Graph**: The application has to supply an input graph, and an additional role mapping. For each of the three input roles there should be a set of nodes (source features, target features and instances).
- **Feature Association Function**: The application may supply a function to calculate the feature association strength. If the application does not set a association function, a fall-back function will be used instead.
- **Output Feature Association Graph**: There is no need for any configuration of the output graph. The output role mappings are inherited from the input graph.
- **Pruning Heuristics**: As the framework is equipped with a predefined set of pruning strategies and thresholds for many typical features, the default setting are already sufficient for the basic use-case.
- **Additional Customisation**: There is no need for additional processing steps with the exception of the most complex application scenarios.

No intimate knowledge of the internal workings of the framework is needed in order to calculate feature associations. Additional insights into the characteristics of the features and their distribution with the data-set will help to fine-tune the calculations. The selection of the matching feature association function is relevant to uncover the significant relationships between the features and detect latent structures. The effects of the pruning strategies will also help to improve the run-time effectiveness. Nevertheless, in an simple application setting even a novice user will generate useful results. An expert user will be able to tune the calculation for optimal results.

Complex Configuration For large data-sets with complex relationships between features, a higher configuration effort will be needed in order to obtain the desired results.

- **Input Feature Graph**: Even for complex data-structures the transformation into a graph like representation is feasible. As in the simple configuration example, the nodes of the input graph are mapped to one of the input roles. The cardinality between nodes and roles is a many-to-many relationship for maximum flexibility.
- **Feature Association Function**: In a complex scenario none of the predefined feature association feature might match the requirements. Therefore a dedicated association function needs to be defined and decomposed into a form suitable for the feature association calculations. Depending on the actual association function the decomposition step is either trivial or a more demanding challenge.
- **Output Feature Association Graph**: Even for a highly complex scenario there is generally no need to change the representation of the final feature associations. The graph data-structure itself is very flexible and can be traversed in multiple ways[162]. Additionally nodes and edges in the output graph might be supplemented with additional meta-data.
- **Pruning Heuristics**: Dedicated heuristics might be necessary for features with properties that deviate from the most common feature types. Users that wish to implement their own pruning strategy should be aware of the complex implications that might be caused by leaving out too many features.

[162] Depending on the customisation, the output feature association graph can also represent a hyper-graph.

- **Additional Customisation**: If needed, users may inject additional processing steps for individual calculations. There custom methods not only control, which algorithms are applied on the data, but they may also manage the output of the feature association calculations. For example a custom implementation might choose to add new roles into the output graph. The customisation of the feature graph generation make it necessary to also adapt the traversal of the output data-structure.

All these task require knowledge in the area of graph structures and special focus should be kept on the performance implications while working with large data sets. To sum up, the configuration setting of the feature association framework are fairly trivial for simple use-cases and complex, but still manageable, for complex applications.

Building the Feature Associations

After the configuration setting have been defined, the actual execution of the feature association calculation need to be specified. Because of the required flexibility a monolithic approach does not seem to be optimal strategy. Therefore a modular and flexible design is appears to be mandatory for complex tasks, like the the association computations.

Design Decision 7 *The whole process of analysing and calculating feature associations is executed as a sequence of phases. Each of these phases servers a single dedicated goal.*

Recently many system architects have changed their strategies to achieve higher performance and throughput. Instead of relying in the increase in speed of a single, central processing unit, strategies to exploit multiple execution units have been studied. Hence, it appears to be reasonable to follow this stream of research in execution environments where multiple execution units are utilised at once. The actual phases and their purpose is described in detail later in this chapter.

Design Decision 8 *Execution of the phases should be done in parallel as far as possible to allow efficient calculations.*

As alternative to pack multiple execution units into a single computer is to use multiple computers wired together via network components. This execution environment is complex to set up and to manage. Furthermore applications and algorithms need to be developed in a way to account for the parallel execution on multiple separate computers.

Design Decision 9 *The architecture of the framework should allow the calculations to be executed in a distributed environment.*

An user of the feature association framework does not need to know the internal workings of the algorithm for many typical application scenarios. If the user wishes to customise the calculations, the same requirements that were used to design the algorithm should also apply on the customisations.

Retrieving the Feature Associations

The final component of the feature association framework is the component to retrieve information from the generated output data-structure. This module is responsible to traverse the output feature association graph. As with building this graph, that actual retrieval operation to a high degree depends on the application as well the data set. In a simple scenario the application starts at a start node and wants to collect all nodes directly connected with the start node. More complex scenarios might involve more sophisticated traversal strategies.

Design Decision 10 *The feature association retrieval component should allow the traversal of the output feature association graph. The start node for the traversal can be determined by the application. Additionally the traversal mode itself is not restricted by framework. A number of common traversal modes should already supplied by the default implementation.*

For most of the common real-world applications the supplied modes of traversal are sufficient. There is no need for the user to gain a deep understanding of the output data-structure. With rising demands the complexity also increases and thus for some applications a customised traversal of the output feature association graph is required.

Example: In the case of word co-occurrences, complex traversal strategies do not appear to be necessary. In the most cases one is only interested in the words with the highest association weight. Therefore the output graph needs to traverse only from the starting to to all directly connected target nodes and sort these according to the association strength.

Calculate Feature Associations

The feature association calculations are executed in a sequence of distinct phases, where each processing step serves one dedicated purpose.

Overview of the Processing Stages

The algorithm to process the features are done in a sequence of 7 phases. These phases are designed to allow decoupling of the individual processing steps and to keep dependencies low. Furthermore it is motivated to allow the whole process to be executed in parallel. Because if two or more processing steps are tightly coupled these no longer can be efficiently be executed independently from each other. Complex dependencies, especially if they contain cycles, severely affect an algorithms ability to scale to large data sets and multiple execution units.

The relative importance of each phases may differ depending on the type of application. The execution order of these stages is common to all application scenarios. The exact calculations done within each of the phases may depend on the desired selection of the algorithms applied on the input features. For example different machine learning approaches can be integrated. Either supervised methods if manually annotated training data is available, or unsupervised approaches for unlabelled data.

The individual processing phases of the feature association framework are:

Preprocess *Reads in the input data and prepares the data structures and execution environment.* Depending on the size of the data and the number of features types different execution strategies are evaluated. Additionally caching data structures are pro-actively filled[163].

Prune *Apply heuristics on the features to filter out noise.* To keep the execution time as low as possible, the majority of unwanted input data should be filtered out in an early processing stage. The actual heuristics largely depend on the data. For textual features (words, stems, ...) derived from human language these heuristics exploit the well-known statistics of word frequencies (e.g. Zipf's law, Heap's law, ...).

Analyse *Gather detailed statistics of the feature distributions over all the instances.* To collect the global statistics all the input data must be available, thus this step cannot be effectively executed in a distributed manner. Therefore this step is executed before the feature association are calculated. The main advantage over of collecting the statistics on-demand is the fact that this kind of operation usually benefits greatly from batch operation. Thus the overall efficiency rises if the global statistics are computed once as one of the initial steps.

Collect *Iterate over the instances and collect all the features while applying local weighting strategies.* In contrast to the global statistics phase starting with this step all further calculations can be done in an distributed execution environment. The focus of locality in this step are the instances[164]. For all instances their features are analysed, collected and weighted according to a local weighting scheme. The collected weighted features and then fed to the next processing step.

Sort *The collected features are sorted according to an internal sorting criterion.* The previous processing step produces a collection of weighted features in a sequence according to the instance sort order. In this phase the locally weighted features are rearranged and sorted[165]. The motivation for this operation is to arrange the data in such a way that it can

[163] On contemporary computer hardware an optimal pre-heating strategy is important to keep the data locally - on the same cluster node as the calculations are done and optimally stored within the fast random access main memory

[164] Locality of reference and sequential access are among the major impact factors on the run-time of an algorithm, especially when dealing with huge amount of data. Both factors are not reflected in the big $\mathcal{O}$-notation, but in an practical application scenario their influence on the run-time must no be neglected.

[165] There exist many sorting algorithms with different characteristics. For this task the algorithm must have the ability to scale with the number of features, instances and parallel execution environments.

be consumed by the succeeding calculations in a streaming, sequential manner.

Associate *Starting with a single feature the association weights with all other connected features are calculated.* The input for this step are the sorted features combined with adorning data. In this stage the main calculations are performed and it is of paramount importance that all necessary data[166] is readily available and provided by the framework.

As different applications yield different demands, the actual implementation of the core algorithms in this phase will vary. Two main modes of operation are presented, the first scenario where feature association are build on a global level and a more complex scenario that involves additional, more sophisticated, computations.

Global Associations The relationship of the source and target features are calculated based on all instances within the data set.

Contextual Associations The set of instances is restricted by a list of contexts. Where a single context is defined by the presence of a target feature within an instance. These contexts can be further processing by two approaches:

Local Associations The source features are associated with different aspects of the contextual target feature.

Recursive Associations The context is recursively processed and associations are build based on a subset of instances.

The output of this stage are the final feature association, again sorted in the sequence according to a deterministic sort criterion.

Store *The result of the calculations are stored in a data structure suitable for fast retrieval.* As the stream of feature associations is produced by the preceding phase these results must be transformed into a dedicated data structure and stored. After all association have been calculated the resulting data structure can be efficiently traversed and searched.

[166] The algorithm might need the local feature weight, statistics or external information

Each of these distinct processing phases will be described in detail the following sections. To demonstrate the inner workings of the algorithms and data structures a simple example is presented, see figure 9. This example shows a simple graph of three instances and two features sets. The first feature set (on the left side) contains of four feature values and the second of two features (right of the instance nodes). Furthermore it is assumed that the first feature set is configured as source and the second of target feature set. At the end of the processing all associations have been calculated between all source features and all target features. See figure 13 for the final association network.

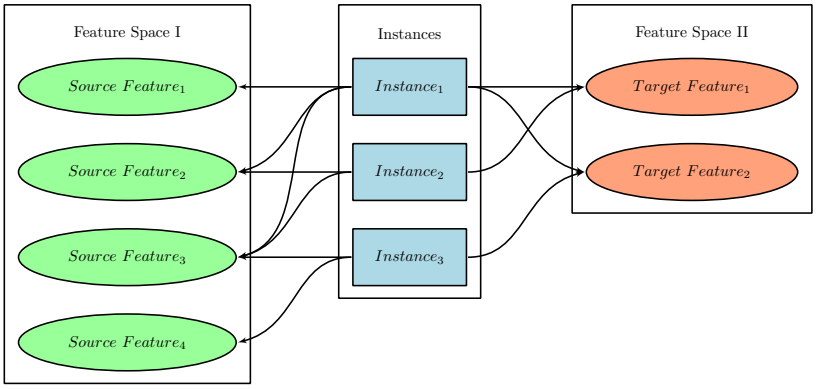

Figure 9: Sample graph, which contains of three instances and two separate feature spaces. For example "Instance 2" is associated with three features, two from the first feature space and one from the second feature space.

Execution Order of the Phases

These processing phases will be executed in a fixed sequence. When viewed from the perspective of a single feature association computation, each phase will logically only start if the directly preceding phased has been completely finished. A simple implementation of the algorithm might choose to not allow two phases to be active at the same time. Up to the sort phase each phase cannot be started as long as the previous phase has not finished. A more sophisticated implementation might choose to deliver the sorted results on the fly and fed them iteratively into the input of the association phase. Results from the association phase can then be stored as soon as the pairs of features have been completely processed.

In figure 10 the individual phases and their exemplary execution sequence is depicted. Most of the phases allow an implementation to choose a distributed computation of the results. In this example the first three phases are executed sequentially and the consecutive phases are executed on multiple execution units. The output of the analyse phase is distributed to multiple execution units, which collect and calculate the local association independently from each other. To achieve the a parallel execution scheme the input feature graph is partitioned into sets of instances. The results of the collect phase is first sorted locally and then the MergeSort algorithm is applied to reorder the data-structures. The reorders data structures are partitioned into sets of target features and distributed among the feature association workers. The associations are then finally collected, merged and stored to build a single feature association graph.

Preprocessing Phase

Before any data can be processed, the actual settings and configuration parameters for the feature association calculation must be parsed and validated. Any inconsistencies in the data, or failed preconditions for the algorithms should be discovered as early as possible. This is especially true for calculations on big data sets as may need several days to process and hence cannot be repeated several times. Because of the required flexibility, there are also many causes which might trigger problems in the computations. Unfortunately many of these settings also in part depend on the actual data being processed.

One cause of such a configuration problems are unsuitable thresholds for the heuristics used for frequency based feature pruning. Because of the nature of such approaches and their tight coupling with assumed feature distributions, they may cause excessive pruning or no pruning at. This might happen if the data set does not match the assumptions. Such a situation should be detected in the preprocessing stage and the user should be informed that the current pruning configuration might not match the data.

Other problems might arise if there is insufficient storage space available or other requirements on the execution environment are not met. Again, the possible cause of errors should be identified and reported to the user as soon as possible.

After successfully validating the configuration settings the execution infrastructure is set up. Depending on the actual settings and the structure of the data, different execution strategies might be employed. For small data sets the executing of the computations can be done on one machine using only a single thread. This type of execution strategy is the most efficient way because there is no communication overhead as the data does not need to be transferred via the network. Additionally there is also no synchronisation bottleneck because the data is not shared between processes or threads. Unfortunately this approach does not scale with the

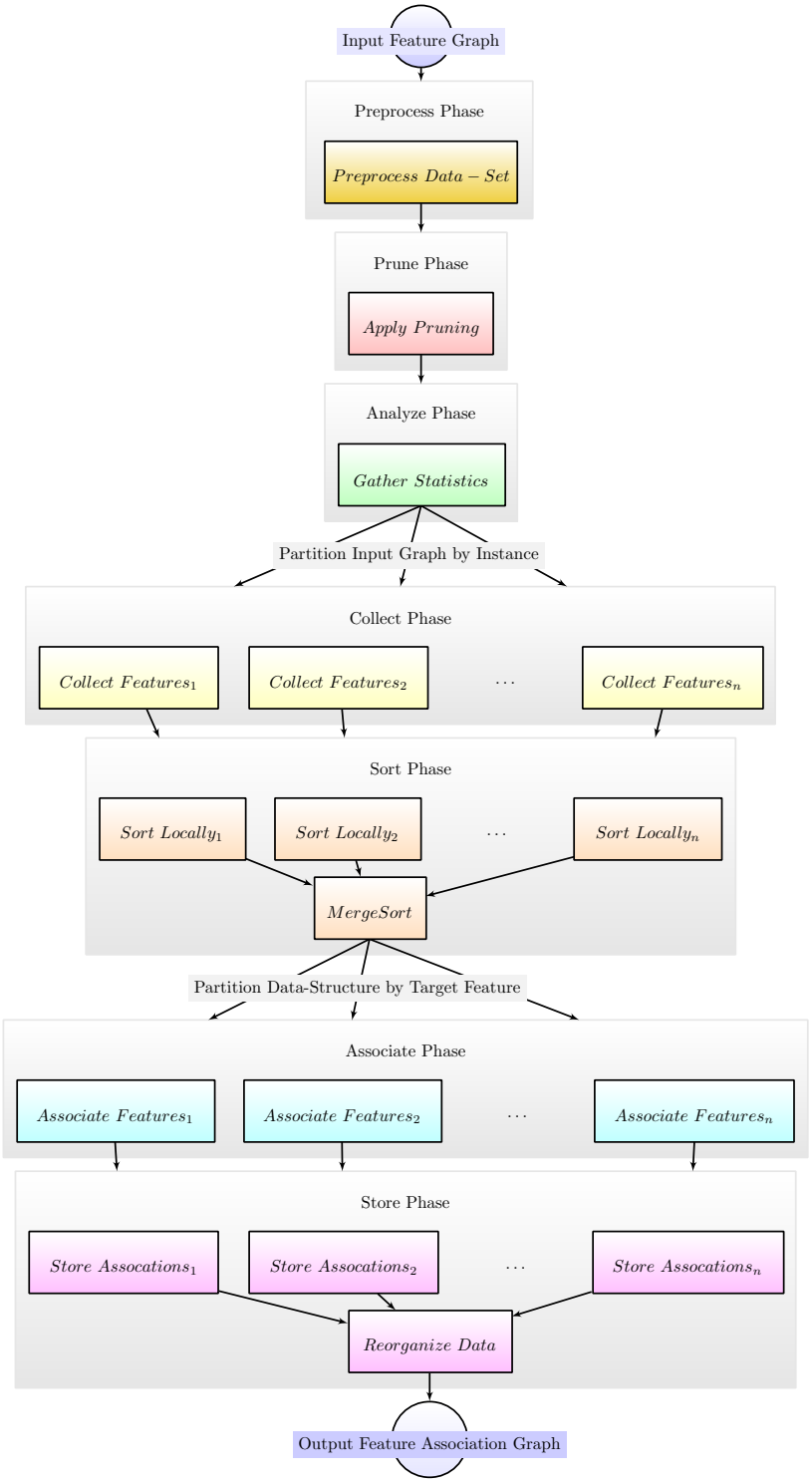

Figure 10: Typical execution order of the phases starting with a feature input graph. Most of the phases may be executed in parallel on multiple execution unit. In this example starting with the collection phase all calculation are conducted partly in parallel. The MergeSort algorithm serves as synchronisation point. Finally a feature association network is created. See figure 9 as an example of an input feature graph and figure 13 for an example feature association network.

size of the data. Therefore for bigger data sets different more complex types of execution strategies need to be developed. For huge data sets the computation is done in parallel on multiple machines using a dedicated cluster management solution and a distributed file system. Table 9 gives an overview of the main characteristics of various executing strategies that can be chosen according to the size and the complexity of the data set and the features.

Data Set Size	Execution Units	Parallelisation	Data Storage	Overhead
Small	Single Machine	Single Thread	Local	None
Medium	Single Machine	Multiple Threads	Local	Synchronisation
Large	Multiple Machines	Multiple Processes	Centralised	Network Communication
Huge	Multiple Machines	Multiple Processes	Distributed	Cluster Management

Table 9: Overview of possible execution strategies for data sets of different size. For larger data sets the complexity and the overhead of the execution infrastructure rises.

Regardless of the chosen type of execution strategies, the execution environment must be prepared for each unit. The data-structures for the algorithms also need to be allocated and initialised. To improve the performance of the feature association computations, the data might be reorganised to optimise the access times[167]. At the end of the preprocessing phase the configuration settings are validated, the matching execution strategy is selected and all data-structures are prepared.

[167] The operating system can be instructed to cache the necessary data to allow faster retrieval.

Pruning Phase

The pruning phase is very important to prevent the combinatorial explosion of feature associations which would make the calculations intractable. Although the pruning step does not directly alter the upper bound on the computational complexity[168], the effective run-time of the calculations as a whole can be massively affected by cutting away parts of the feature space. In order to achieve this, the feature selection process itself must not be associated with high computational costs. The main challenge is to find the sweep spot between the run-time improvements gained by the reduced feature space and the decrease in completeness of feature associations.

[168] The big $\mathcal{O}$notation remains the same

There is no general way to select the optimal strategy a-priory and there is not a single threshold which suits all application scenarios. To achieve this one would need the complete output of the calculations to know which associations are of lesser importance, while the notion of importance depends on the actual application setting. For example, variations of the stepwise regression algorithmic approach[169] cannot be applied in this setting, because of the associated run-time costs. Therefore one has to resort to heuristics as an alternative. The downside of using heuristics is that most of them depend on the data and the distribution of features.

[169] N. Draper, H. Smith, and E. Pownell. *Applied regression analysis*. Wiley New York, 3rd editio edition, 1998

Especially if the number of features is very high, one might want to apply strategies to reduce the number of features. Due to the nature of most of the feature selection approaches (See page 53) they cannot be integrated directly into the pruning phase of the feature associations calculation. These algorithms are often based on the assumption that features carry redundant information and at least in part depend on each other. These redundant features are assumed to be superfluous features and are selected as candidates to been thrown away. But the goal of feature associations is to analyse and extract the interactions between features and therefore the such a pruning approach might introduce an unwanted loss of relevant feature associations.

Feature selection approaches, which are not based on this assumption,

appear to be more suited. For example, pruning strategies based on feature frequencies. Additionally the frequency based methods generally provide a much better run-time complexity and scale well to large data sets.

Common pruning heuristics for textual features include the removal of common words, infrequently used words, number and other words that do not carry semantics. Meta-data that usually accompanies digital documents differs from features that represent natural language. Examples for such meta-data are author names, organisation names and creation dates. Different pruning strategies need to be developed for these kinds of features in order to be efficient.

At the end of the pruning phase all features have been filtered out that do not contribute to the final results but cause a lot of unnecessary calculations.

Analysing Phase

After the features spaces have been preprocessed and various pruning strategies have been applied, all individual features values are analysed. This analysis is conducted not on single instances, but on the data set as a whole. This corpus wide statistics are therefore also named global statistics (opposed to local statistics). Global statistics play an important role in many knowledge discovery algorithms, for example in the field of information retrieval [170].

[170] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of OSIR*, volume 2006. Citeseer, 2006

The global statistics need to be completely calculated prior to any feature association processing. This requirement is derived from the need to be able to parallelise the consecutive algorithmic steps. After the global statistics have been gathered, the data set can be processed in a completely distributed manner.

Depending on the type of features and the association function, different statistics need to be provided. Therefore the set of statistics to calculate needs to be known to the framework. This information may be deduced from the chosen feature association functions in the majority of cases.

Additional to the feature value statistics, other characteristics of the data set are gathered in this phase. The properties of the instances can also be aggregated as they may contribute necessary information for the feature association algorithms.

External knowledge about the feature values is also collected in the analysing phase. For example external databases are queried to collect the necessary information for the feature association computations.

The most common global statistics for feature values and the characteristics of the instances within a data set are:

Instance Count Number of instance within the data set. The amount of instances can be used as proxy for the total size of the data set. It can be used as base to determine how many execution units should be spawned in order to process the data set. Additionally the instance count measure is often used by algorithms in combination with other statistics.

Instance Frequency The number of instances a single feature value is referenced. This number is probably the most common global statistic. In combination with the instance count it is used to calculate the inverse document frequency (IDF). The IDF represents one part of the TFIDF weighting scheme[171], which has been regarded as the state-of-the-art weighting scheme in information retrieval for a long time.

[171] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988

Average Instance Length The arithmetic mean of the number of features per instance. Many algorithms base their calculations on the probability of a feature to occur in the context of a single instance. With the number of features occurrences per instance this probability also tends to rise, at least for many common real-word datasets and of course for the

theoretical random case.

Variance & Dispersion If the edges within the input graph is equipped with additional meta-data, statistics based on these adorning information can be generated as well. For meta-data which represent numbers, the variance of the values can be computed. For example for textual features a meta-data value may reflect the number of times a specific word occurs within a single instance document. The variance then might be suitable to distinguish between words of varying importance to the knowledge discovery application.

Position The individual position of an occurrence of a feature within an instance is used by many feature association functions. This information might not be available for all types of features. For textual features the position is derived from the ordered sequence of words within a text. Depending on the preprocessing this might also include stop-words and punctuation. The data-structure that hold this kind of information must be capable of dealing with multiple values as a word may occur multiple times within a text.

Given the position information of a sequence of features a sliding windows technique can be applied. Using this scheme each features is processed together with its the surrounding context. Features that occur close to each other also tend to be strongly associated. Based on this intuition multiple applications of feature associations have been described in the literature[172,173,174].

The position information not only provides a means to improve the quality of the results, but it also provides means to improve the run-time. Pairs of features that co-occur within the same instance, but their respective positions are far apart are candidates to be left out of the feature association calculations. The cut-off distance between two features can be used to tune the desired behaviour - a low threshold for faster computations or a high threshold for a more complete result. The optimal value depends on the available processing resources and on the data set, in particular on the average instance length. Window sizes of 2, 5 and 10 words around a target word are most common settings in the literature to calculate term co-occurrences.

Sophisticated feature association function do not only exploit the distances between the occurrences to apply a cut-off scheme. The relative distance of two occurrences may be also integrated into the weighting scheme.

[172] H. Schütze and J. Pedersen. A cooccurrence-based thesaurus and two applications to information retrieval. *Information Processing & Management*, 33(3):307–318, May 1997

[173] B. Lemaire and G. Denhière. Incremental construction of an associative network from a corpus. In *Proceedings of the 26th Annual Meeting of the Cognitive Science Society*, pages 825–830. Citeseer, 2004

[174] E. Terra and C. L. A. Clarke. Frequency estimates for statistical word similarity measures. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 165–172, Morristown, NJ, USA, 2003. Association for Computational Linguistics

At the end of the analysing phase a dictionary of all feature values of all feature spaces are build. This look-up data-structure contains the global statistics and provides methods to efficiently retrieve them. Copies of this dictionary are provided to all units within the execution environment to enable a fast access to the needed information in the consecutive phases.

Collect Phase

The goal of the collect phase is to reorganise the data set. Prior to this phase the data set is sorted by instances, at the end of the phase data-structures are produced which are ordered according to the target feature. This data-structure not only contains the target feature, but also a reference to the instance and all co-occurring source features. Additionally a local association weight is calculated for all feature combinations. The design of the algorithm to collect the target features is motivated to allow an sequential access to the data-set. This is especially crucial in a distributed environments, where the data is transferred in a streaming manner.

Starting with the collect phase all computations can effectively been executed in parallel. Only subsets of the complete data set are available after

the global statistics have been gathered. These statistics are distributed to all executing units prior to the collecting phase for faster access.

All instances of the data set are separated into smaller sub-collections and then sent to one of the execution units. The decision which instance to include in a specific collection is done by the execution framework. The basic implementation of such a dispatching functionality simple iterates over all instances in the data set. As soon as a fixed number of instances have been collected, these instances are submitted to an (idle) execution unit. When processing huge data sets this naive algorithm does not provide the optimal throughput. The locality of the data within a distributed storage system should be one of the deciding factors on how to manage the workload.

After a sub-collection is assigned to a single execution unit, the instances within the collection are sequentially processed. The main task of this processing phase is to collect all relevant information for each instance. Depending on the actual configuration, different feature types may be selected as either source or target (or both). The input for the feature collection is an unordered set of features associated with an instance. Each of these features is connected to the instance via a relation. This relation is assigned to one of the feature types and may contain a weight. Additionally each relation may also carry meta-data, for instance a position information.

The collection of features starts by retrieving the list of target feature types from the configuration. Starting with the first target type all assigned source feature types are processed. This is accomplished by a look-up in the configuration to acquire the list of source feature types for the current target type. The source feature type list is iterated and all features of the matching type are collected. This collection of source features is then combined with all target features of the current target type. Then the combined result for each target feature is reported to the framework. These steps are repeated for all configured target types. Algorithm 2 demonstrates the processing steps of the collection phase in pseudo code.

In figure 9 a small sample graph is depicted which serves as an example to illustrate the reordering of the input graph. The output data-structure of the collect phase is depicted in figure 11. All source features are grouped together with all target features connected via a shared instance node. The local weight is additionally stored in the data-structure.

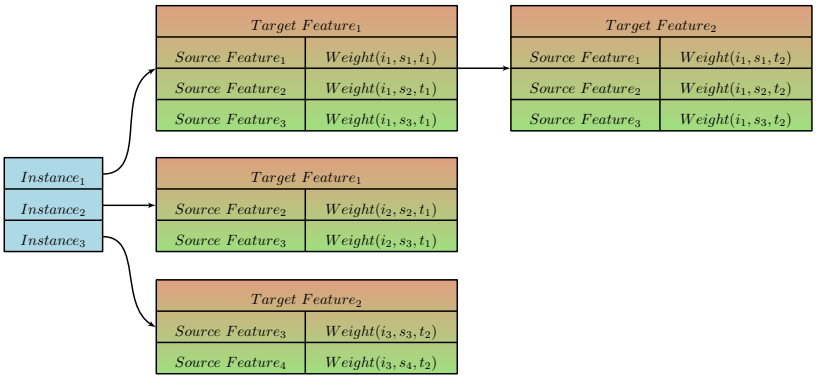

Figure 11: Output data-structure of the collection phase. The instances are now ordered internally by the target features. Each target feature carries each associated source feature together with the local weight.

The run-time complexity of the collection stage is determined by the number of instances, features and the number of target and source feature types. In the worst case each feature is configured to be associated with all other features. Thus the upper bound of the collect phase is $\mathcal{O}(mn^2)$, where n is the number of features for each of the m instances[175]. The average case is expected to be far lower for real-world data sets, because of the prevalent data sparsity. In hardly any real-word data set all instances are

[175] Although the nested for-each loops in the pseudo-code might suggest a higher upper-bound at first glance.

connected to all features. Especially after many feature connections have been removed in the pruning phase. For common data sets the limiting factor of the processing on contemporary computing infrastructures is not the actual computation itself. Instead the run-time is dominated by the time it takes to read and write the data from the storage. Therefore any efficient implementation of the collection phase should rather be I/O-bound than CPU-bound.

The size of the data-structures produced during the collection phase (see $yield()$ function in algorithm 2) may become larger then the input data-structures itself. That may happen if a source feature is connected with multiple target features via a shared instance. This is a common scenario for many applications, for example when building a co-occurrence network. The worst case estimate of the needed storage is equal to the run-time complexity estimate - $\mathcal{O}(mn^2)$.

The feature association function is employed within the collection phase to calculate the local weight of an association between two features. This is done in two steps. At first local weights of all features associated with an instance are calculated. In the second step the source and target weights are combined into one local association weight, which is then stored together with the source feature. In both steps the feature association function has access to all the statistics created in the analysing phase. The local weighting functions are very frequently invoked thus the implementation should be as efficient as possible.

Example The local weighting part of a feature association function that implements the euclidean distance[176] might be realized as:

[176] $d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_i^n (p_i, q_i)^2}$

$$localSourceWeight(x) = x \tag{11}$$

$$localTargetWeight(y) = y \tag{12}$$

$$localWeight(w_x, w_y) = (w_x - w_y)^2 \tag{13}$$

An implementation of the collection phase should apply caching techniques to avoid unnecessary and redundant calculations. The storage infrastructure should apply sophisticated encoding and compression mechanism on the collected data to minimise the disk storage overhead.

At the end of the this phase all target features are grouped for all instances. Additionally together with the target features the associated source features are also aggregated. The source features are accompanied with a local association weight. Although the results of the collection phase are generated iteratively, the consecutive phase can only start as soon as all target features have been fully collected.

Sort Phase

The task of the sorting phase is to sort all grouped target features according to an internal sorting criterion. Because of the large amount of data that needs to be processed special dedicated algorithms need to be adopted. For real-world data sets the data will usually not fit into main memory, therefore a sophisticated memory management is required.

An exemplary input for this phase is depicted in figure 11 and described in the collection phase section. The output of the sort phase for this input data-structure is shown in figure 12. While at the start of the sort phase the primary sort criterion is the sequence of instances, the sorting algorithm reorders the data in such a way that the target features can be read out sequentially.

Although an implementation of the sorting phase is free to choose its own algorithmic approach, a simple but efficient method is presented here.

Algorithm 2 Overview of the main processing steps in the collect phase.

Require: Valid configuration in $config$
Require: Feature association function in $scorer$
Require: Collection of instances stored in I

```
for all i ∈ I do
    F_i ← i.getFeatures()                          ▷ List of all features of i
    T ← config.getTargetTypes()                    ▷ List of all target types
    for all t ∈ T do
        S_t ← config.getSourceTypesForTargetType(t)
        F_{i,t} ← {f | f ∈ F_i and type(f) = t}
        R ← {}                                     ▷ Initialise with empty map
        for all f_y ∈ F_{i,t} do
            O_y ← []
            w_y ← scorer.localTargetWeight(weight(f_y))
            for all s ∈ S_t do
                F_{i,s} ← {f | f ∈ F_i and type(f) = s}
                O_s ← []
                for all f_x ∈ F_{i,s} do
                    w_x ← scorer.localSouceWeight(weight(f_x))
                    w_{x,y} ← scorer.localWeight(w_x, w_y)
                    O_s ← O_s + [< f_x, w_{x,y} >]
                end for
                O_y ← O_y + [O_s]                  ▷ Append source features
            end for
            R[f_y] ← O_y                           ▷ Add to result map
        end for
        YIELD(i, t, R)                             ▷ Report result map
    end for
end for
```

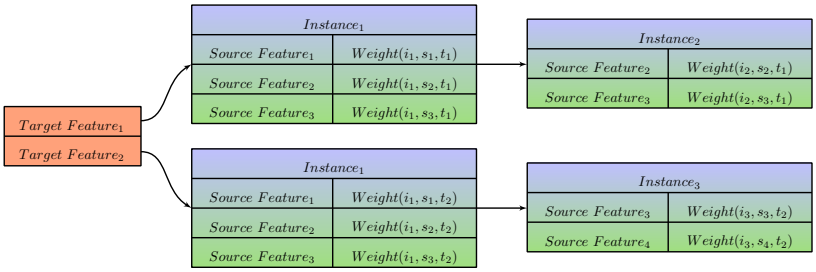

Figure 12: Output data-structure of the sort phase. The target features are used as primary sort criterion, the instances as secondary. For each instance all associated source features and the local association weights are stored.

This method is based on the merge-sort algorithm, which is well suited for this purpose as it incorporates the usage of external storage. This property is mandatory when dealing with large data sets, that do not fit into main memory.

The first step of this algorithm is to write out all the data gathered in the collection phase into a sequence of files. The data is generated iteratively while collecting, and stored in in an in-memory buffer. This buffer has as upper limit of entries, which is defined by the actual resources of the execution environment. Generally speaking, the large this buffer can be made, the faster and more efficient the sorting will be. As soon as the buffer reaches its maximum capacity, the first sort stage is executed.

At the begin of the first sort stage the buffer contains a sequence of local target to source feature mappings, sorted by instance. Now all the entries in the buffer are sorted in-place according to the output sorting criterion, applied upon the target features. Again the actual implementation of the sorting algorithm is not restricted. Examples of common sorting algorithms suitable for this task are Quicksort[177] and Heapsort[178,179]. As soon as the buffer has been sorted according to the target feature, the data can be transferred into secondary storage (on the local hard-drive, a distributed file-system or a centralised storage).

The first stage is repeated as long as the collect phase generates data. After all instances has been processed by the previous phases, the next sort processing step is initiated. The stored buffers are now all read in parallel. Because the buffer are already sorted, they can be consumed in a streaming fashion, which is the preferred operation mode in a distributed environment[180]. The merge-sort picks the buffer with the lowest entry according to the global sorting criterion. This entry is then removed from the buffer and made available for the later processing phases. As long as there is at least one data entry in one of the buffers this procedure is repeated.

The complete algorithm is available as pseudo-code, see 3. Using this approach all entries can be read out in the desired sequence. Additionally the consecutive processing phases can start their calculations in parallel due to the iterative nature of the merge-sort algorithm. After all target features have been read out, the stored buffers they are no longer needed and can be safely removed.

[177] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10, 1962

[178] J. W. J. Williams. Algorithm 232: heapsort. *Communications of the ACM*, 7(6):347–348, 1964

[179] See `http://users.aims.ac.za/~mackay/sorting/sorting.html` for an in-depth comparison of the algorithms.

[180] In contrast to a random access read pattern, where the whole data has to be available in one place - either locally or in a centralised storage.

Association Phase

Up to the this phase the data has been filtered and organised in a way allowing efficient calculations. The association phase finally calculates the actual feature associations and their association weight. The input of this phase is a sequence of target features, the relations with the instance nodes and the associated source features. The results of the sort phase is iteratively consumed, one target feature and its accompanying data-structure at a time. By employing this iterative scheme it is possible to execute the association phase in parallel or distributed among multiple execution units. The output of this phase are the calculated feature associations including an association weight[181] and additional optional information. As with the sort phase, the output of this phase is also produced in an iterative manner.

Due to the flexible design of the framework and the breath of information available[182], there are multiple strategies on how to implement this phase. Two different approaches will now be presented:

Scenario I At first a scenario is described that summarises the available feature data and exploits the global statistics. Associations produced in this scenario represent the pairwise relation between two features based on their distributions on the whole data set.

[181] The weight of an association can also be seen as the strength of their relation (depending on the actual configuration and employed feature association functions).

[182] During this phase the algorithms have full access to all the global statistics, which may include information from various sources, like for example external resources.

Algorithm 3 Overview of the main processing steps in the sorting phase, which is based on the the well known merge sort algorithm.

Require: Sort criterion in f
  $S \leftarrow []$
  procedure SORTINGCONSUMER ▷ **Step 1**: Write out buffers
    $B \leftarrow []$
    while $hasMoreData()$ **do**
      $E_t \leftarrow$ CONSUME ▷ Read data from the collect phase
      if $sizeof(B) + sizeof(E_t) < threshold$ **then**
        $B \leftarrow B + [E_t]$
      else
        SORTINPLACE(B, f) ▷ Sort the buffer in-place
        FLUSH(B) ▷ Flush the buffer to storage
        $S \leftarrow S + [B]$
        $B \leftarrow []$
      end if
    end while
    if $B \neq []$ **then**
      SORTINPLACE(B, f)
      FLUSH(B)
      $S \leftarrow S + [B]$
    end if
  end procedure

  procedure MERGESORT(S) ▷ **Step 2**: Read buffers
    $R \leftarrow []$
    for all $s \in S$ **do**
      $r \leftarrow openStreamingReader(s)$
      $R \leftarrow R + [r]$
    end for
    while $\{r | r \in R$ and $r.peek() \neq nil\} \neq \emptyset$ **do**
      $i \leftarrow \min_i(f(R[i].peek()))$
      $e \leftarrow R[i].pop()$
      YIELD(e) ▷ Report the lowest entry
    end while
  end procedure
  FREE(S)

Scenario II Then a second scenario is presented, where the associations are exploited on a local level. In this case the relations of features are not based on the data set as a whole, but on subsets which represent individual contexts. An example of such context is the presence of a specific feature, so that all instances are included in the context which have a connection with a specific feature. This scenario also allows to incorporate machine learning techniques to detect common pattern within the feature distributions. Additionally this scenario allows a recursive approach to calculate feature associations, where depending on a certain context the local distributions of multiple features in relation to each other can be analysed.

Scenario I: Global Associations This scenario calculates the pairwise feature associations while iterating over the target features as they are produced by the sort phase. The feature associations generated in this scenario are based on the global distribution of the features. The distribution of a feature is the characterised by the way a feature is related to the instances within the data set. The initial input data-structure consisting of two types of roles of nodes - the instance nodes and the feature nodes. All edges that connect a feature with the nodes that represent instances are included in the distribution, thereby collecting the weight of the edges. To calculate the global feature association between two features, their individual distributions are compared.

The actual properties of the distributions and the way the feature association weights are calculates depend solely on the feature association function being used. The feature association function has already been invoked to calculate the local weight of an association between source and target features during the collecting phase. This local weight together with the information of the shared instances is now available in the association phase. This information is made available to the feature association function, which is now invoked to calculate the global association weight. This step is repeated for every source feature that is connected via at least one shared instance with the current target feature. This way a list of associations is generated for each target feature. To keep the number of associations low, this list can be pruned to only contain the most important associations. This pruning can either be done by applying a threshold on the association weight or by restricting the number of entries within the list. Which of the two pruning strategies is chosen, or whether pruning is done at all, depends on the actual application. Generally speaking the fewer feature associations are consecutively stored, the better the retrieval performance will be.

By executing this scheme finally all feature associations within the data set are processed and the output is iteratively passed to the next processing phase.

A simple example is given to illustrate the possible results that may be generated in the global association scenario. In this example a basic feature association function is used. This function calculates the number of shared instances by two features, with I_i being the set of instances in which feature i occurs in:

$$w_{s,t}^{simple} = |I_s \cap I_t| \tag{14}$$

Figure 13 depicts the output graph for the global association while using the number of shared instances as weight. The input graph is shown in figure 9.

No global statistics are necessary in this case, as the information generated during the collection phase is sufficient in this case. Also the local weight calculations can be skipped entirely. The feature association function just needs to count the number of occurrence of a source feature in the

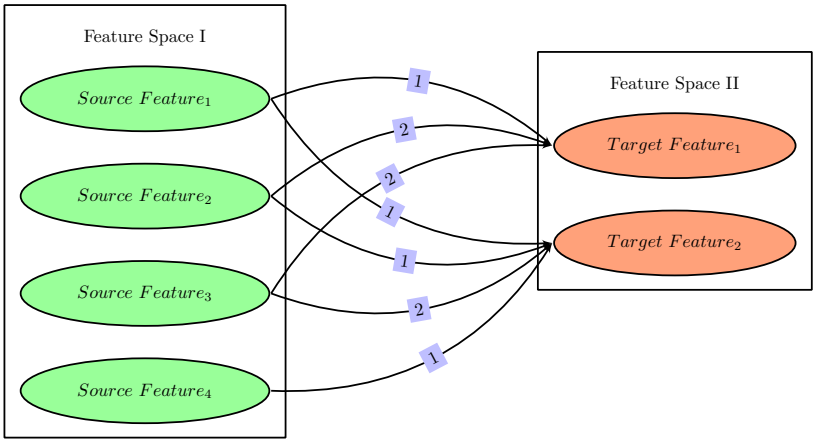

Figure 13: Final feature association network for the example input graph. The edges carry an association weight, which in this case is the number of shared instance between a source feature and a target feature.

list of instances the target feature occurs in. As the association weights will tend to be high for more frequent features, one might want to add a additional normalisation step on the weights. A prerequisite for this normalisation is the presence of global statistics of the features. For example, the total number of occurrences of the features can be taken source for the normalisation to realise the well known Overlap Coefficient.

$$w_{s,t}^{normalized} = \frac{|I_s \cap I_t|}{min(|I_s|, |I_t|)} \tag{15}$$

The basic processing steps of this phase when calculating global associations is presented in pseudo-code in algorithm 4.

Scenario II: Contextual Associations The global association scenario compares distribution of features over the whole data set. In the local association scenario only distribution based upon a restricted subset of the data set are taken into consideration. This is motivated by the intuition that strong associations may not only exist on a global level, but also on a local level. Especially if the data set is heterogeneous, which is a common property of real-world data sets. Depending on a context, there might be strong association between two features, that might not be easily detected when comparing their global distributions.

In the following a context is defined as the set of instances, in which a target feature occurs in. An important motivation for this definition is rooted in the implications on the expected run-time complexity. For local association an increase in computational costs is expected as the sheer the distributions and statistics do not only need to be computed for all feature on the data set as a whole, but again again for all contexts. As there are potentially many contexts within a single data set, the rise in run-time needs to be considered. In the association phase most of the information needed to calculate the context specific statistics and distributions are already present. By using the definition of a context based on the distribution of the target feature, the calculation of contextual feature association becomes feasible also for large data sets. This is made possible by the fact that the data-structures in this phase are already sorted according to the target features.

In figure 14 the input data-structure of the association phase is depicted. For a given target feature, the associated data-structure is the same as the input data-structure of the whole data set, but restricted to the instances in which the target feature actually occurs in.

The scenario of local associations can be further sub-divided, based on

Algorithm 4 Overview of the association phase when building global associations. For each target feature at first the local weights are aggregated over all instances and then the final global association weight is calculated.

Require: List of input data-structures - $\mathfrak{T}$
Require: Feature association function - $\mathfrak{f}$
Require: Global statistics - $\mathfrak{S}$
procedure BUILDGLOBALASSOCIATIONS
while $T \neq \emptyset$ **do**
$t \leftarrow$ CONSUME($\mathfrak{T}$) ▷ **Step 1**: Read a single target term
$I \leftarrow InstanceData(t)$

$A \leftarrow \{\}$
for all $id_i \in I$ **do**
$S_i \leftarrow id_i.sourceFeatureData$
for all $sd_j \in S_i$ **do**
$s_j \leftarrow sd_j.feature$
$w_{i,j}^{local} \leftarrow sd_j.localWeight$

▷ **Step 2**: Aggregate all local associations
$A[s_j] \leftarrow$ AGGREGATE($\mathfrak{f}$, $A[s_j]$, $w_{i,j}^{local}$)
end for
end for

$G \leftarrow \{\}$
for all $a_s \in A$ **do**
▷ **Step 3**: Calculate the global weight
$G[s] \leftarrow$ ASSOCIATIONWEIGHT($\mathfrak{f}$, $\mathfrak{S}$, t, a_s)
end for

YIELD(G) ▷ **Step 4**: Report the final associations
end while
end procedure

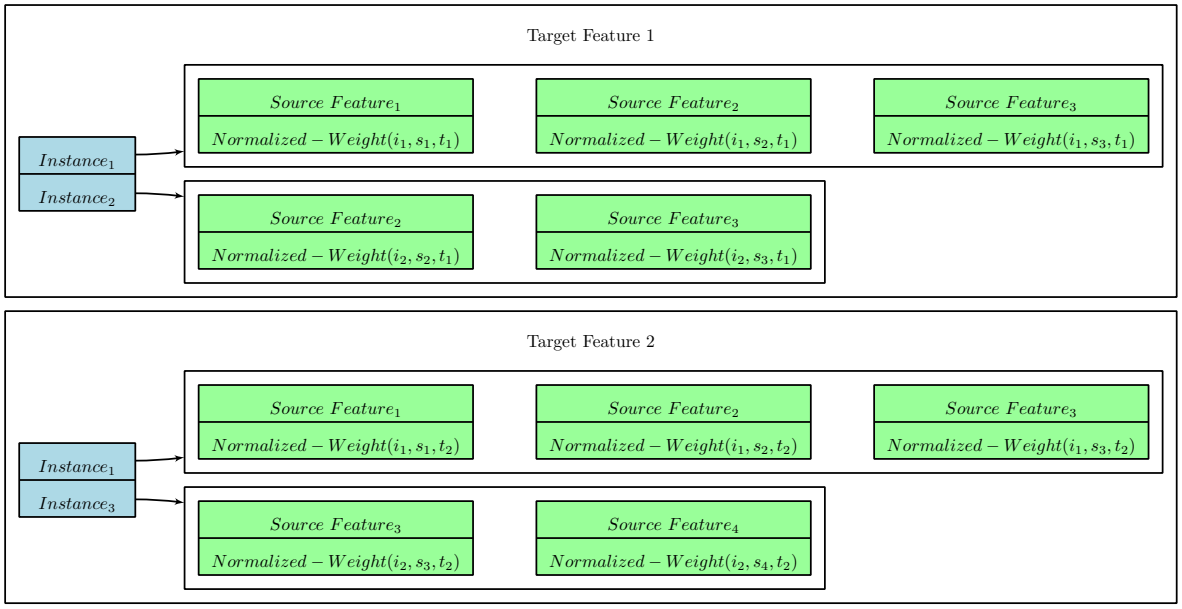

Figure 14: Input data-structure of the association phase. For each target feature all associated instances and source features are accessible for algorithms and feature association functions.

how to process these data-structures:

Relative Contextual Associations The spawned context is used to directly compute association between the current target feature and all associated source features.

Recursive Contextual Associations The context is used to dynamically create a new data set, where the feature association computation as a whole is recursively be applied.

Both approaches are applicable for a number of real-world scenarios[183]. The first approach is more efficient, as all necessary data for the computations are already prepared and quickly available. On the other hand the second approach offers more flexibility, but at the cost of a higher complexity in terms of computational as well as effort to set-up. Examples for both methods are presented in the following section, starting with the less complex model.

[183] See the application chapter for examples

Relative Contextual Associations In the first scenario the output features are directly derived from the target features used to define the context. This can be achieved by allowing a single target feature to be split into multiple output target features, where each of these newly created features represent a different aspect of the target feature. In order to achieve such a mapping of one target feature into multiple derived target output features, the set of instances can be partitioned. All instances that are associated with a single target feature can subsequently be divided into disjoint sets, where each of these sets represents one of the new target output features. Figure 15 exemplary displays the output of the association phase when using the targets terms as contexts as well as nodes in the output graph to build the contextual associations.

Features with multiple aspects are common in many real-world data sets. An example for this are ambiguous features, where a simple feature value represents multiple semantics. In the case of human languages, it is quite common that a single words carries multiple meanings[184]. For example the term '`java`' may refer to an Indonesian island or a popular programming language.

[184] D. Klein and G. Murphy. Paper has been my ruin: conceptual relations of polysemous senses. *Journal of Memory and Language*, 47(4):548–570, Nov. 2002

For this scenario the association phase is divided into two processing steps. In the first step the various aspects of a target feature are detected and

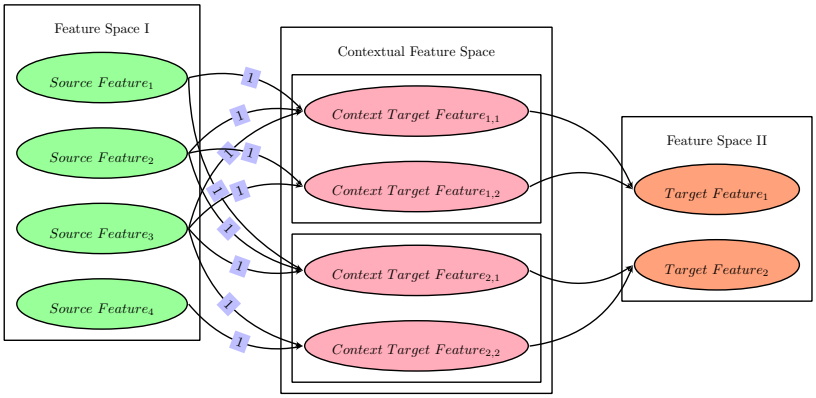

Figure 15: Example for the scenario where the target terms are used to define a context and then split into several nodes, where each of them resembles different aspects of the context. In this example each of the instances represent a different aspect. The source features are not associated directly with the target node, but with the newly introduced contextual target features. The weight of the association is, like in the previous example, the number of shared instances, which is equally 1 for all associations.

in the second step the source features are associated with the new output features. While the second step is similar to the association phase of the scenario for global associations, for the first step new techniques have to be introduced.

In order to detect the distinct subsets of instances that represent the individual aspect of the target features techniques from the field of machine learning can be applied. Unsupervised machine learning algorithms that detect common patterns within a set of instances are especially suitable for this task. These clustering algorithms usually start with a matrix, which is partitioned into coherent groups of rows which are similar to each by detecting pattern within their column values. In the case of feature associations, the a single instance represents a single row and the feature weights are stored in the columns. Popular examples of such a cluster algorithm are agglomerative hierarchical algorithms[185] and variations of the K-means clustering algorithm[186]. Besides hard clustering algorithms, which partitions the data set into not-overlapping parts, fuzzy clustering approaches might be taken into consideration, where a single instance can be assigned to more than one cluster. Both methods are equally suitable for the task of identifying multiple aspects in a given context. In the case of the fuzzy clustering approach the weights between the source and contextual target features should honour the relationship between the identified clusters and the source features.

Another enhancement of the traditional clustering approach are the algorithms that perform co-clustering[187]. This clustering scheme does not only partition the rows of the input matrix, but also the columns are simultaneously organised in groups. The basic scheme of the integration of a machine learning approach into the calculation of relative contextual associations is presented as algorithm **??**. For each target feature a context is created. This context consists of all instances in which the current target feature occurs in. The context is then clustered. The source features are then associated with an output node, that is derived from the clustering solution.

Besides machine learning approaches there are statistical methods that can be employed to structure the instances.

Recursive Contextual Associations The second type of local association presented here is based on a recursive execution pattern. As soon as the context has been established, a new input data set for an association calculation is dynamically created. In this case, the generated feature associations are independent from the configured target feature type. The target features are solely used to create contexts within the data set. This scenarios allows the largest amount of freedom, which emphasises needs a more elaborate configuration. The run-time complexity of this scenario is higher than for

[185] A. El-Hamdouchi and P. Willett. Comparison of hierarchic agglomerative clustering methods for document retrieval. *The Computer Journal*, 32(3):220, 1989

[186] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 34, page 35. Citeseer, 2000

[187] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98. ACM, 2003

Algorithm 5 Overview of the association phase when building relative contextual associations by employing a clustering algorithm.

Require: List of input data-structures - $\mathfrak{T}$
Require: Feature association function - $\mathfrak{f}$
Require: Global statistics - $\mathfrak{S}$
Require: Clustering algorithm - $\mathfrak{c}$

```
procedure BUILDRELATIVEASSOCIATIONS
    while T ≠ ∅ do
        t ← CONSUME(𝔗)                          ▷ Step 1: Read a single target term
        I ← InstanceData(t)

        M ← Matrix(|I|, |𝔖.sourceFeatures|)
        for all id_i ∈ I do
            S_i ← id.sourceFeatureData
            for all sd_j ∈ S_i do
                s_j ← sd_j.feature
                w_{j,local} ← sd_j.localWeight
                                                ▷ Step 2: Fill the context matrix
                M_{i,j} ← MATRIXWEIGHT(𝔣, 𝔖, w_local)
            end for
        end for

        C ← CLUSTER(𝔠, M)                       ▷ Step 3: Cluster the context matrix

        for all c_k ∈ C do
                                ▷ Step 4: Create an output node for each cluster
            t_k ← CREATENODE(t, k)

            A_k ← {}
            for all id_i ∈ instances(c_k) do
                S_i ← id_i.sourceFeatureData
                for all sd_j ∈ S_i do
                    s_j ← sd_j.feature
                    w^local_{i,j} ← sd_j.localWeight
                    w^cluster_{i,j,k} ← CLUSTERWEIGHT(w^local_{i,j}, c_k)

                                        ▷ Step 5: Aggregate all cluster associations
                    A_k[s_j] ← AGGREGATE(𝔣, A_k[s_j], w^cluster_{i,j,k})
                end for
            end for
            L_k ← {}
            for all a_{k,s} ∈ A_k do
                        ▷ Step 6: Calculate the context association weights
                L_k[s] ← ASSOCIATIONWEIGHT(𝔣, 𝔖, t_k, a_{k,s})
            end for

            YIELD(L_k)                          ▷ Step 7: Report the local associations
        end for
    end while
end procedure
```

the other scenarios, with an theoretical upper bound of $\mathcal{O}(m^2 n^5)$[188].

The output of the recursive contextual association scheme is an association network, which consists of source nodes, as well as set of target nodes that were created based on the individual contexts. Each context is then represented a set of newly created output feature nodes. Additionally the target feature, that was used to build the context, may also be present in the final output graph. See figure 17 for an example output. For this example the source features are used as origin as well as target of the edges in the association graph. The basic steps to create such an contextual association graph are outlined as algorithm 6.

[188] It should be noted that this limit is only of theoretical interest, for real-world applications the feasibility of this scenario mainly depends on the configuration and the employed thresholds.

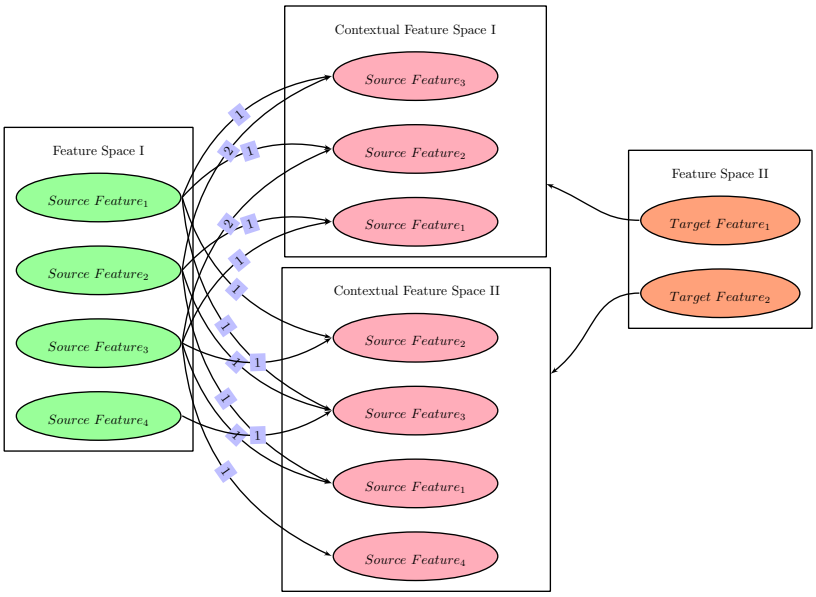

Figure 16: Example for the scenario where the target terms are used to define a context, which is then used to apply a local feature association calculation. Each of the two target features in this example define a context that contains all instances, in which the corresponding target feature occurs in. Within the two contexts all source features are associated with all other source features. The weight of the relations between the features is the number of shared resource.

Because of the complexity of this approach and the number of ways on how to the features are combined the output network may vary depending on the configuration. One of the basic settings of this configuration will be presented by using a real-world example application. The final goal of this example is to find contextual collocation. A collocation is defined as sequence of words that tends to occur more frequently than expected given the frequency of the individual words. Usually collocations carry specific semantics that are not contained in the words that build these sequences. Person names and organisation names are common collocations. The phrase "`New York`" is a typical example of such a collocation[189]. Many collocations are ambiguous and may have different meanings depending on the context. The words "`red hat`" may either refer to a company[190] or to a hat that just has a red colour. There also exist collocations that are only meaningful for specific topics. These collocations can only be detected when analysed on a contextual level. For textual resources the discourse structure has been studied as one of such context. Collocations have been found to be helpful in such a setting [191,192].

The feature association framework can be set up to build such contextual collocations. Two types of features are needed in order to find statistically salient sequences of words. The first feature type are the words contained within a collection of texts. The second feature type should identify the topics that are covered by the texts. Thus the input feature graph can be created with these node types:

- Each individual text within the collection is represented as instance node.
- Words within a single text are mapped as source feature nodes and

[189] Seen from a probabilistic perspective, the phrase "`York New`" should be as likely, if only the frequency of individual words is taken into account.

[190] `http://www.redhat.com`

[191] D. Yarowsky. One sense per collocation. In *Proceedings of the workshop on Human Language Technology*, HLT '93, pages 266–271, Morristown, NJ, USA, 1993. Association for Computational Linguistics

[192] D. Martinez and E. Agirre. One sense per collocation and genre/topic variations. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, pages 207–215, Morristown, NJ, USA, 2000. Association for Computational Linguistics

Algorithm 6 Overview of the association phase when building local associations by recursively calculating the associations. For each target feature a context graph is created. This graph is then used as input for a recursive calculation of feature associations. The result of each contextual association calculation is merged to build one final association network for all contexts.

Require: List of input data-structures - $\mathfrak{T}$
Require: Configuration for building an association graph - $\mathfrak{C}$
  procedure BUILDRECURSIVEASSOCIATIONS
    while $T \neq \emptyset$ **do**
      $t \leftarrow$ CONSUME($\mathfrak{T}$) ▷ **Step 1**: Read a single target term
      $I \leftarrow InstanceData(t)$

      ▷ **Step 2**: Create an empty context graph
      $G_t^{input} \leftarrow$ CREATEGRAPH
      for all $id_i \in I$ **do**
        $S_i \leftarrow id.sourceFeatureData$
        $n_i \leftarrow$ CREATENODE(G_t^{input}, $id_i.instance$)
        for all $sd_j \in S_i$ **do**
          $s_j \leftarrow sd_j.feature$
          $w_{j,local} \leftarrow sd_j.localWeight$
          $n_s \leftarrow$ CREATENODE(G_t^{input}, s_j)

          ▷ **Step 3**: Fill the context graph
          CREATEEDGE(G_t^{input}, n_i, n_s, $w_{j,local}$)
        end for
      end for

      ▷ **Step 4**: Calculate the contextual associations
      $G_t^{output} \leftarrow$ BUILDASSOCIATIONS(G_t^{input}, $\mathfrak{C}$)

      ▷ **Step 5**: Merge the all contextual associations
      YIELD(G_t^{output})

    end while
  end procedure

are connected to the corresponding instance node. As a single word can occur multiple times within a single text, the data-structure that represents the graph must be able to deal with multiple edges between two individual nodes[193]. Additionally each edge carries a position information, which stores the absolute position of the word within the sequence of words.

- The topics are represented as target feature nodes. A single text may cover multiple topics, therefore a single instance node may have multiple connections with target feature nodes.

[193] The terms *multigraph* or *pseudograph* are commonly used for this type of graph.

The feature association calculations are done in a recursive manner. At first association calculation collects the context information. Each context resembles a set of texts that cover a single topic. For each topic all words within the text instances must be collected. This information is then again fed into a features association calculation. For the second association calculation the words are associated with themselves, following two restrictions:

1. Two words must occur in the same text, because collocation cannot span multiple documents.
2. Difference of the positions of two words must not exceed one. This restriction effectively allows only consecutive words to be associated.

The weight of the association between two words can for example be derived from a statistical significance test. Finally for each topic a network of significant collocations is generated.

Store Phase

The final phase of the calculation of the feature associations is the storage of the results. As the associations are iteratively produced by the previous phases, the storage system should be capable of processing multiple results continuously and in parallel. The work that needs to be done by the storage phase can thereby be grouped into two stages, which are sequentially executed:

1. Collect all incoming results and persist them as fast as possible.
2. Reorganise the feature associations for efficient retrieval.

Collect Feature Associations In the first stage the storage function receives the feature associations from the association phase. These associations should be processed as fast as possible to prevent the association calculation to prevent any distributions on the overall system. One possible way to achieve this desired behaviour is a out-of-context execution approach. This execution strategy is often used in situations where a low latency is a priority[194]. In this case the information is received by one module and further dispatched to another dedicated storage module, which is executed in parallel. Using this approach the actual I/O-operations are decoupled from the feature association calculations. This is made possible as in contemporary computational architectures the sub-systems which are responsible for accessing storage devices are autonomous and thus can operate independently from the central processing units.

[194] S. Korsholm, M. Schoeberl, and A. P. Ravn. Interrupt handlers in Java. In *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 453–457. IEEE, 2008

For large data sets it is advisable to distribute the collection of feature association among multiple execution units. The most basic setting of such an architecture is to use one dedicated server which receives the final feature associations via a network connection. The data is then managed by a single server instance. This setting is on the one hand easy to configure

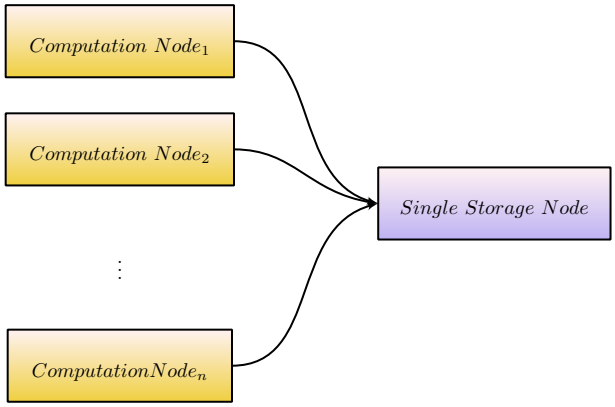

Figure 17: Execution topology of the storage phase for multiple computation units and a single storage unit. Feature associations are computed in parallel using multiple nodes, the results are collected and stored by a single node. This scenario is typical for a configuration where the results are stored in a database.

and to administer, but on the other hand such a configuration poses a single point of failure and thus should not be used for critical computations.

More advanced setting make use of a higher level of distribution. To facilitate an higher amount of parallelism when storing and loading data, a variety of different distributed file-systems have been proposed and implemented[195,196]. These storage mechanism allow access from multiple nodes distributed within a network. The complexities of the management of the data within the network is hidden from the clients. Distributed file-systems do not provide the same performance as local file-systems, but they scale with the amount of data and are therefore well suited for managing huge data-sets.

[195] D. Borthakur. The hadoop distributed file system: Architecture and design. Technical report, 2007

[196] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003

Reorganize Feature Associations In the second stage of the storage phase the collected feature associations are reorganised to allow fast retrieval. The actual processing within this stage depends on the storage backend and on the level of distribution. If the feature association have been collected by multiple nodes within a distributed execution framework, all individual results are now merged. Alternatively if the collected data is stored in a distributed file-system, the collected information can new be read out and further processed by a single instance. For example in such a scenario an merge-sort algorithm might be employed to create a single sorted repository of feature associations.

Maintenance operations are also executed during this stage. For example a dynamic indexing scheme might schedule a reorganisation of the stored data by applying an logarithmic merge algorithm[197]. See table 10 for a brief list of examples of typical maintenance operations for a set of storage backends.

The choice which backend implementation to use and therefore which maintenance operation are necessary depends on the application and the available infrastructure.

[197] S. Büttcher and C. L. A. Clarke. Indexing time vs. query time: trade-offs in dynamic information retrieval systems. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 317–318. ACM, 2005

Storage Backend	Operations
Relational Database	Update Table Index, Compact Tables
Object-oriented Database	Garbage Collect
Sequential File	Sort Entries
Dynamic Index	Logarithmic Merge

Table 10: Examples of typical operations done by different storage backends in the reorganisation step of the storage phase.

Result

The result of the feature association calculation is a data-structure that represents the feature association network. The choice which data-structure to use is up to the implementation, but there are a number of requirements and mandatory properties of the data-structure. Because of the large number of feature associations, the data-structure should be lean and should allow fast and effective storage. The data-structure has to be able to cope with the fact that the complete association network is interactively generated. The output of the association phase are edges between a source feature and a target feature, and for advanced configurations these edges may actually be hyper-edges. As the head of a direct edge may serve also be the tail of other feature association. This may for example happen if the source features and target features originate from the same feature space. The implementation must be able to correctly map these relationships.

Additionally to the association weight each edge may also be associated with a payload, which may contain any data needed for further processing or used for visualisation purpose. To keep the size of the graph small, one could resort to compression technique developed in the field of web graphs[198]. Alternatively one could employ relational database or other storage technique, like for example an inverted index. Algorithms developed in the fields of social web[199] may also be used to manage the association graph.

[198] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602. ACM, 2004

[199] A. Harth and S. Decker. Optimized index structures for querying rdf from the web. In *Web Congress, 2005. LA-WEB 2005. Third Latin American*, page 10. IEEE, 2006

Feature Association Functions

The feature association function is responsible to calculate the final association weight. The semantics of this weight is not constrained by the framework and depend on the function and the application. Feature association functions themselves need to be decomposed into factors and functional blocks in order to be used by the framework.

Overview

The task of a feature association function is to finally deliver a weight for the association of the two features. Although from a technical perspective the association weight is just another meta-data attached to each edge in the output feature association graph, it plays an important role in many application scenarios. The weight usually corresponds to the association strength, but application developers might be interested in different interpretations of the association weight.

In order for the calculations to be efficient on large data-sets, a feature association function is decomposed into a set of independent *factors* and *functional blocks*. The *factors* resemble the smallest entities that make up a feature association function and are carefully crafted to keep the number of dependencies low.

Functional blocks are a combination of multiple factors, where each block serves a distinct purpose. In the execution sequence of the feature association calculations, each functional block is coupled to an individual phase. The output of one functional block is then fed into the input of the next block.

In table 11 an overview of the main differences between factors and functions blocks are given. Both types of components are motivated by two sets of different use-cases and therefore both types deviate from each other.

	Factor	**Functional Block**
Use-Cases & Goals	Reuse Common Factors, Enable Data Caching	Manage Workflow, Enable Code Optimisations
Decomposition Type	Logical Units	Functional Units
Input	Feature Input Graph, External Resources, ...	Factors, Output of Preceding Block
Output	Depends on the Factor	Defined by the Framework
Execution Sequence	Free	Fixed
Execution Phase	All	Single
Optimisation Criteria	Flexibility	Efficiency

Table 11: Key differences of the two types of components of a feature association function. Both types are differently motivated and serve different purposes.

Decomposition into Factors

The decoupling of the calculations of individual factors allow different scheduling techniques in order to determine the factors. For example, some factors may be based on external resources. Access to the these resources may be costly and may cause latencies, as the requests need to be transferred over the network. If the calculations of multiple factors are submitted in batches, the efficiency of the overall process does rise. These factors can be categorised into three groups:

Global Factors : Factors that are independent from any feature. Global factors usually represent statistical measures of the data-set. These factors only depend on the data-set and need to be re-calculate only if the data-set has been changed.

Single Feature Factor : Factors that depend on only a single feature. Examples for such factors are statistics of the distribution of a single feature. Single feature factors also commonly integrate information from external resources. This kind of factor allows caching strategies to improve the run-time of the calculations.

Feature Pair Factor : Factors that depend not only on a single feature, but a pair of features. This is the most complex form of factor. Feature pair factor are usually composed of multiple global and single feature factors. This kind of factor does not lend itself to effective optimisation and therefore it should be avoided or decomposed into simpler factors as much as possible.

Each category allows a different set of optimisation strategies. Among these strategies is the calculation of the some of the factors at the start of the feature association calculation. This boast a couple of advantages over the naive on-demand execution, which would require either redundant calculations or a synchronisation overhead.

Global Factors Factors that do not depend on a single feature, but there might still be a dependency on the data set. Using mathematical notation, a global factor can be written as:

$$\mathfrak{f}^{global}() \tag{16}$$

Examples for such global factors are statistics over all features within the data set. The total number of feature nodes of different type and average out degree of instance nodes are examples for such statistics. Global factor also may integrate external information into the calculation of feature associations.

Additional to the global factors, that take no argument, there are also a set of global factors that may have dependencies. Such a factor may depend on an instance node. For example the factor may represent the out-degree of an instance node, which is equivalent to the number of feature nodes connected to the instance node. This scheme can be further refined by including an additional argument. For example the type of a feature node can be used to restrict the out-degree to all edges that connect the instance nodes with feature nodes of a certain type, the factor function then is:

$$\mathfrak{f}^{global}(i, featureType) \tag{17}$$

The feature association framework executes the calculation of all global factors in a dedicated analysis phase. The results of the calculation are then stored and distribution to all execution units that take part in the actual association calculations.

Single Feature Factors Factors that only depend on a single feature value are referred to as single feature factor. The single feature factor takes a single feature as argument:

$$\mathfrak{f}^{single}(f) \tag{18}$$

The type of factor is common for many feature association functions. The single feature factor typically represent the relationship of feature nodes in regard to other nodes in the input feature graph. For example a

basic factor is the number of connections between a feature node and the instances nodes.

Factors can be calculated for both types of features, source and target features. There is no need to make any distinctions in the factor calculation.

Single feature factors can be effectively calculated prior to the main feature association process. Therefore the single feature factors, as well as the global factors, can not only be used from within the feature association function, but they can already been used during the pruning phase. During the pruning phase all features are removed, that are likely to contribute little to the final result of the calculations. The decision which feature to leave out is based on a set of heuristics, which make use of global statistics and the properties of the features.

Feature Pair Factors The last type of factors are the most complex form of factor. Feature pair factors depend on two features, usually a source and a target feature:

$$\mathfrak{f}^{pair}(f_s, f_t) \tag{19}$$

These features are usually a source and a target feature, although there are no conceptual restrictions. Feature pair factors are typically a combination of various global as well as single feature factors. A example for a feature pair factor is the number of shared instance nodes for a given set of two feature nodes.

In contrast to the other two factor types, this kind of factor is calculated on-demand. There is no way to determine beforehand which feature combinations will be finally associated with each other. Therefore any calculations of feature pair factors should only be done if needed. Due to this, this kind of factor cannot be used by the pruning algorithms.

Example of Common Factors

To illustrate the concept of a feature association function factor, a number of examples are given. These examples are based on functions employed by common knowledge discovery applications. Therefore additional to the definition of the factors the application settings and the mapping to the input feature graph is given.

Number of Instances This global factor is used in many application scenarios, it just represents the number of instance nodes in the feature input graph:

$$\mathfrak{f}_I^N \tag{20}$$

Usually this factor is used for normalisation purpose, for example to determine the proportion of a feature in relation to the instances. Depending on the application, this factor can also be interpreted as number of documents or document count, if the data-set consists of document, which are represented as instance nodes in the input graph. In table 12 a number of examples for this factor is given, based on data-set for the collaboratively created on-line encyclopedia WIKIPEDIA.

Data Set	$\mathfrak{f}_I^N$
English Wikipedia	3,450,942
German Wikipedia	1,403,289
Simple English Wikipedia	81,197

Table 12: Number of instances in WIKIPEDIA data-sets for various languages. Each WIKIPEDIA article is mapped as a single instance node in the input feature graph. Articles without content (redirects, disambiguation pages, ...), category articles and internal articles have been left out.

The result of applying this factor is a scalar. This number can be easily be computed at the beginning of the calculations and then distributed to all execution unit and used within the pruning phase as well as the association phase. The data-structure that holds the value for this factor is simply an integer value.

$$\mathfrak{f}_I^N : \mathbb{Z}^+ \tag{21}$$

Number of Features This factor is independent from an actual feature, but there may be an restriction on the feature type. It counts the number of unique feature nodes within the input graph:

$$\mathfrak{f}_F^N \tag{22}$$

It can also be seen as the size of the input vocabulary[200], if each entry in the vocabulary is modelled as an feature node. Another interpretation of the factor is the number of dimensions of a feature space, where each feature of a type resembles a single dimension in an (typically high-dimensional) feature space.

[200] In mathematical notation: $|\{f|f \in F \cap type(f) = t\}|$, for F being the set of all features and t is a given feature type

This factor is found in many feature association functions. It is easy to compute and therefore the number of features for all feature types are usually pre-calculated by default. The result is stored as an associative array with the feature type as key and the factor result as integer value:

$$\mathfrak{f}_F^N : \{featureType \mapsto \mathbb{Z}^+\} \tag{23}$$

Table 13 lists the value for this factor for various Wikipedia data-sets. The upper limit of possible feature association pairs is $\mathcal{O}(\mathfrak{f}_F^N)$. For the English Wikipedia data-set this results in a theoretical upper limit of feature association calculations that need to be done of 1.16×10^{14}. This emphasises the need for a sophisticated pruning strategy in combination with efficient algorithms and a stream-lined execution approach.

Data Set	$\mathfrak{f}_F^N$
English Wikipedia	10,839,070
German Wikipedia	5,597,398
Simple English Wikipedia	211,063

Table 13: Number of feature nodes in the input feature graph for three different language version of the Wikipedia data-set. The English version is the largest data-set with the richest set of different features.

Average Out-Degree of the Instance Nodes The next important characteristic of the data-set is the average number of features per instances. Again this global factor does not depend on the features, but solely on the features types:

$$\mathfrak{f}_I^{avgout} \tag{24}$$

If the input graph represents a sparse document $\times$ term matrix, then this factor represents the average document length. This measure is commonly used in knowledge discovery application that operate on textual data. It is needed for a normalisation of the probabilities of an feature to occur within an instance (or an term within a document). This motivation is realised in many information retrieval applications[201].

[201] H. Fang, T. Tao, and C. Zhai. A formal study of information retrieval heuristics. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 49–56, New York, NY, USA, 2004. ACM

The Wikipedia as textual resource is an example of an sparse document $\times$ term matrix. In table 14 an overview is given for a selection of Wikipedia data sets in different languages and different characteristics. For the English version of the data set each instance nodes is on average connected to about 500 different feature nodes, giving a total of about 1.65 billion relations within the input feature graph.

Data Set	$\mathfrak{f}_I^{avgout}$
English Wikipedia	476.98
German Wikipedia	329.44
Simple English Wikipedia	136.36

Table 14: Average out-degree of the instance nodes for the WIKIPEDIA data sets in various languages. The input feature graph for the English WIKIPEDIA is more dense than the German version and the relatively sparse Simple English WIKIPEDIA.

The result of this global factor can be computed for all feature types within the data set. The collected results are then stored in an associative array, similar to the "Number of Features" factor:

$$\mathfrak{f}_I^{avgout} : \{featureType \mapsto \mathbb{R}^+\} \tag{25}$$

Out-Degree of the Instance Nodes Sometimes for a feature association function the average out-degree of an instance node is not sufficient. In this cases the individual out-degree values of a set of instance nodes is required. The out degree factor also depends on a given instance node, therefore this factor needs an additional argument:

$$\mathfrak{f}_I^{out}(i) \tag{26}$$

This set can either be simply the complete set of instances nodes in the input graph or a selection of instance nodes. For example the set can be defined as all instance nodes that are connected to a specific feature node.

In the case of an input graph that is created by mapping a document $\times$ term matrix, this factor is the document length, with the document d as argument[202]. As this factor does not depend on a feature as input variable, this factor can still be categorised as global factor.

[202] The factor represents the number of terms within a document: $docLength(d)$

In comparison with the other presented global factors, this factor needs an additional argument - the instance node. Therefore the result of this factor is more complex than the other global factors. It can be realised as associative map of associative maps, where the keys of the maps are the instance nodes and respectively the feature types. The mathematical notation should clarify this, where i being a single instance node:

$$\mathfrak{f}_I^{out}(i) : \{i \mapsto \{featureType \mapsto \mathbb{Z}^+\}\} \tag{27}$$

Whether such a complex data-structure should be completely filled out for all instance nodes as part of the initialisation of the calculations depends on the application scenario. If out-degree of the instance nodes is used in the pruning phase or is configured to be part of the global statistics, then this factor should be scheduled for computation at the beginning of the processing.

In-Degree of the Feature Nodes The number of instances for a given feature node is also common and often used in combination with the total number of instances factor. This factor depend on a given feature and is therefore classified as single feature factor. Nevertheless it can be calculated as soon as the input graph is completely generated. The factor can by used for source as well as target features. The in-degree factor is equivalent to the number of instance nodes a single feature node is connected to and defined as:

$$\mathfrak{f}_F^{in}(f) \tag{28}$$

Similar to the global factors, the in-degree factor can also be used within the pruning phase. Some pruning strategies remove features that are too common or too rare. For example two thresholds can be used to decide with feature to remove from the calculations. The first

threshold defines the lower bound of the in-degree of the feature nodes. This threshold is an integer number - $\mathbb{Z}^+$.

The actual value of the lower threshold depends on the actual data set and the run-time constraints. If there is a limited amount of computational resources is available, it is advised to use a higher number for the lower threshold. Values in the range of 2 to 10 are common for medium to large size data-sets. Features that occur very rare in the data set might be random noise or artifacts generated in the processing of the data-set. In textual corpora, spelling mistakes and parsing errors are the prime source for features that occur only once or twice in the whole data set. Example for rare features in an real-world data-set are given in table 15.

Feature	$\mathfrak{f}_F^{in}(f)$
`blablabla`	4
`zzz-999`	3
`brtitish`	1
`cloos'-ton`	1
`miso-nikomi`	1
`printn(a`	1

Table 15: Examples for rare features, in this case low frequent words from the English Wikipedia. Among these examples are spelling errors, artifacts generated by automatic text extraction methods but also very rare words too.

The lower threshold can be applied on the set of all features **F** to create the restricted set of pruned features $\mathbf{F}_{lower}^{pruned}$.

$$\mathbf{F}_{lower}^{pruned} = \{f | f \in \mathbf{F} \cap \mathfrak{f}_F^{in}(f) \geq \theta_{lower}\} \tag{29}$$

The upper bound is usually not an integer number, but a ratio ($\mathbb{R}^+$). While for the lower bound a constant number is remains a valid choice for a broad array of data sets, an fixed upper threshold would highly sensitive to the size of the data set. Given that in many real-world scenarios the data set is incrementally build, there would be the need that the upper threshold is updated each time a new instance is introduced to the input graph. Therefore the upper bound is defined as a ratio which combines the in-degree factor with the number of instances factor.

For real-world application this threshold is usually set to a ratio less than 0.5, because features that occur in more than the half of all instances rarely yield meaningful results when analysed using statistical methods. The mainstream strategy to cope with high frequent, but relevant feature is to introduce a negation feature, so that each instance is connected with the new feature which has no relation with the frequent feature. The high-frequent feature can then be removed from the data-set. In table 16 examples for high frequent features in a real-world data-set are given.

Feature	$\mathfrak{f}_F^{in}(f)$	$\frac{\mathfrak{f}_F^{in}(f)}{\mathfrak{f}^N}$
`the`	3,032,573	0.879
`in`	2,919,623	0.846
`a`	2,903,352	0.841
`of`	2,888,379	0.837
`is`	2,639,282	0.765
`and`	2,634,096	0.763
⋮	⋮	⋮
`with`	1,703,251	0.494

Table 16: Examples for high frequent features: Top words from the English Wikipedia. Most of these words carry no semantics and serve only a grammatical purpose. Therefore these words can be left out of the feature association function without depriving the quality of the output. The top ranked word (`the`) occurs in 88% of all instances.

As with the lower bound, the upper threshold can be applied on the set of all features to generate a limited set of features, which should be smaller, but still allow the algorithms to detect important associations within the data set:

$$\mathbf{F}_{upper}^{pruned} = \{f | f \in \mathbf{F} \cap \mathfrak{f}_F^{in}(f) \leq \frac{\theta_{upper}}{\mathfrak{f}^N}\} \tag{30}$$

Because of the fact that the $\mathfrak{f}_I^{out}(i)$ factor is a valuable source for the pruning heuristics, the factors for all features should be pre-computed at the beginning of the computation. The data-structure that holds the results of this computation is an associative array, which uses the features as keys an the factors as values:

$$\mathfrak{f}_F^{in}(f) : \{f \mapsto \mathbb{Z}^+\} \tag{31}$$

Weight of the Feature-Instance Edges The next factor depends on a feature as well as on an instance. The factor that reflects the weight of an edge that connects an instance node with a feature node is referred to as:

$$\mathfrak{f}_F^w(f, i) \tag{32}$$

This single feature factor represents the weight of an relationship between a single instance and a single feature. The value of this weight is determined by the process that created the initial input feature graph. Therefore the interpretation of the weight varies according to the application scenario. Usually this weight will be a scalar and therefore the data-structure that holds all values of this factor is an associative array of all features mapped to their factor values:

$$\mathfrak{f}_F^w(f, i) : \{f \mapsto \mathbb{R}\} \tag{33}$$

An application might choose to use a more complex data-structure for representing the weight between instances and features. In this case the data-structure that holds the results of the factor calculations needs to be adapted as well.

This single feature factor depends on a tuple of an instance and a feature, thus the space requirements to store all factor values for a data-set is $\mathcal{O}(nm)$, for n instances and m being the sum of the source and target features. Depending on how dense the input graph is connected, the size requirements might exceed the available resources. Therefore such a factor is usually not pre-computed and cached[203], but calculated on-demand.

The weight of a feature-instance relationship is an important source for calculation feature associations. Therefore this factor is commonly used by many feature association functions.

One example usage is the Euclidean distance between two points, which is defined for n dimensions as:

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{34}$$

If the features are interpreted as dimensions and the instances as points, then each weight represents the value of a single dimension. In the two dimensional case the input graph consists at minimum of two instance nodes. The dimensions are represented by feature nodes (the first is references the X-axis, while the other one represents the Y-axis)[204]. In this bi-partite graph every feature node (point) is connected to all instances, with an connection that carries a scalar weight. In order to calculate the Euclidean distance between point i_1 and i_2 one has to

[203] Although a sophisticated implementation does employ caching methods coupled with adapted eviction strategies.

[204] The application developer can freely choose how to map the points and dimensions to the input graph, for example the points might be encoded as features and the dimensions as instances

combine the individual factors by using the features f_1 and f_2:

$$d(i_1, i_2) = \sqrt{\begin{array}{l}(\mathfrak{f}_F^w(f_1, i_1) - \mathfrak{f}_F^w(f_1, i_2))^2 + \\ (\mathfrak{f}_F^w(f_2, i_1) - \mathfrak{f}_F^w(f_2, i_2))^2\end{array}} \tag{35}$$

In many cases the number of dimensions will be rather higher. A caching strategy can exploit the access pattern to calculate and store all factors for a given feature in a singe batch operation. Because of the architecture and execution sequence the access to the factors will be sequential for both the features, as well as the instances. This property boosts the effectiveness of cache access and was one of the main motivations when planning the feature association algorithm.

Sum-of-Weights of the Feature-Instance Edges While a factor that depends on a feature and an instance is no candidate to be pre-computed, but an aggregation of such a factor can be used in that way. The mathematical notation to denote the aggregated weights factor is:

$$\mathfrak{f}_F^{\Sigma w}(f) \tag{36}$$

The sum of all weights over all instances for given feature is one such example. This factor is defined as sum of the weights of edges between a given feature and the connected instance nodes and can be calculated using the feature weight factor:

$$\mathfrak{f}_F^{\Sigma w}(f) = \sum_{i \in I} \mathfrak{f}_F^w(f, i) \tag{37}$$

Because this single feature factor depends on only the feature, the data-structure that stores the value of the factor is relatively simple. It is an associative array with the features as keys and the aggregated weights as values.

$$\{f \mapsto \mathfrak{f}_F^{\Sigma w}(f)\} \tag{38}$$

If the weights are scalars, then the pre-computed factors should easily fit into main memory. This factor is often used by pruning algorithms to compute indicators for when to remove a feature from the feature association processing. Therefore it is usually completely pre-computed in the initialisation of the feature association calculation.

The sum-of-weights factor is also routinely used within feature association functions for normalisation purposes. A maximum likelihood estimate can be decomposed into various factors where one of these is the sum-of-weights factor. The coin tossing experiment is used as example to illustrate the use of the factor. The input graph consists of only a single instance node that in this example represents the coin. Furthermore the graph contains two feature nodes. The first feature feature node is labelled 'head' and the other one 'tail'[205]. The weight of the edges that connect the instance node with the features is the number of times each of possible outcomes occurred. In this case the sum-of-weights factor is equivalent to the number of tosses. The example estimator for the coin to display 'head' can be written as[206]:

$$MLE_{head} = \frac{\mathfrak{f}_F^w(f_{head}, i)}{\mathfrak{f}_F^{\Sigma w}(f)} \tag{39}$$

If the weight of the relation is not a simple scalar, the aggregation method needs to be adapted. The data-structure that holds the factor values has to be re-factored in this case as well.

Squared Sum-of-Weights of the Feature-Instance Edges The final example factor is derived from the sum-of-weights factor. For the aggregated

[205] This example is presented to illustrate the usefulness of the factor, therefore other outcomes of this experiment are omitted (coin lands on the rim, coin disappears in a block hole, ...)

[206] In this example there is only one instance node which represents a single coin. One could therefore remove the variable i altogether for the sake of brevity.

squared weights the used symbol is:

$$\mathfrak{f}_F^{\Sigma w^2}(f) \tag{40}$$

Instead of aggregating the weight directly, a function is first applied on the weight. In this case the weight is multiplied by itself.

The data-structure, as well as the pre-computation strategy is the same as for the sum-of-weight feature. An associative array is suitable to store the values which have an storage requirement of $\mathcal{O}(n)$ for n features:

$$\{f \mapsto \mathfrak{f}_F^{\Sigma w^2}(f)\} \tag{41}$$

But in contrast to the aforementioned factor the squared weights factor is not as common and usually not employed by pruning strategies. It is mainly used by feature association functions for length normalisation, especially if the data is rooted in the Euclidean space. Finding the magnitude or length of an Euclidean vector[207] plays an important role in many algorithms. Starting point is an input graph which is build from a set of vectors. Each axis in the Cartesian space is mapped as an instance node in the input graph and each feature represents a vector. To normalise a vector (=feature) f to unit length the magnitude is employed, where n is the total number of axis (=instances):

$$f^{normalized} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, a_k = \frac{\mathfrak{f}_F^{w}(f, i_k)}{\sqrt{\mathfrak{f}_F^{\Sigma w^2}(f)}} \tag{42}$$

[207] The length of a vector is defined as: $\|\mathbf{a}\|_2 = \sqrt{\sum_i {a_i}^2}$, and is also called Euclidean norm or Minkowski distance (L^p norm) with parameter 2.

The feature pair factors are similar to the global and single feature factors, but they depend on a pair of features. Therefore they cannot be efficiently be pre-computed due to storage constrains in the most cases. There are also only few feature pair factors that are shared between feature association functions and therefore these factors will be presented in conjunction with the function in which they are employed.

Functional Blocks of a Feature Association Function

The task of the feature association function is to determine whether there is an association relation between two features and determine the weight of this relationship. In order to use any function within the features association framework it has to be decomposed into factors and functional blocks. The factors are the smallest unit of a function. They have a minimum set of dependencies and should be designed to be reusable in different functions and different execution phases. The second decomposition layer are the so called functional blocks. Each feature association function consists of a defined set of blocks. The execution sequence of the functional blocks is also determined by the framework.

Overview of the Functional Blocks The feature association framework defines a set of blocks. Each functional block is assigned to a specific execution phase. The input can be freely defined by the function, whereas the output value of a function block is imposed by the framework. These restrictions by the framework are necessary, because the framework has to manage the data-exchange between the different blocks of a feature association function.

The tasks and semantics of the individual blocks are predefined by the framework. The framework also invokes the execution of the blocks and supplies the necessary input values.

At first a short overview of the six different blocks is given, followed by a detailed description. Examples for the decomposition of feature association functions in conjunction with the presentation of a list of common feature association functions later in this chapter.

Local Source Weight This functional block is defined to calculate a local weight for a given source feature node and a given instance node. This block is scheduled for execution in the collection phase for all instances and their related source feature. s

Local Target Weight Similar to the local source weight block, but instead of source feature node, a single target feature node is given. This block type is invoked at the same time as the local source weight block.

Local Association Weight The two local weights are combined into one weight by the local association weight block. After the local source and target weight blocks for a single instance have been executed, their results can be combined.

Normalise The local weight is transformed into a normalised representation. Therefore not only the weight to be normalised is supplied by the framework, but also all other local weights for a given target feature node. The block type is executed in the association phase.

Aggregate For a single target feature all normalised weights for given associated source features are aggregated into a single weight. The aggregation starts a soon all normalised weights of a single target feature are completely calculated.

Global Association Weight The aggregated weight is finally transformed into the global association weight for a pair of source and target feature. This type of block is also executed during the association phase.

To use a function to calculate the association weights within the feature association framework, every one of the blocks needs to be implemented. The most simple way to implement a block is to output the same data that was used as input. Each implementation of a block has access to all global statistics as they were constructed during the analysis phase. Furthermore a block can employ any combination of factors which satisfy the available dependencies.

In figure 18 the different functional blocks are depicted. The charts gives an overview on how the output of one block is fed as input of the consecutive blocks. If the feature association framework was customised to serve a more complex application scenario, the actual values exchanged between the function block will change. An customisation might for example choose to remove certain feature associations or create new ones, or even introduce new types of features. The sequence of the block execution will still remain the same.

In the following paragraphs the individual block types will be presented in detail. Additionally examples will be given to further illustrate on how the functional blocks are invoked.

Local Source Weight Block The first job of the collect phase is to iterate over the instance nodes in the input graph. For each instance node all connected feature nodes are traversed. The feature nodes are then inspected to which role they are mapped[208]. Only nodes that are mapped to the role "Source Feature" are included in the further processing. During this process the local source weight block of the configured feature association function is invoked.

[208] Please see the "Input Data-Structure" section of an overview of input graph roles earlier in this chapter.

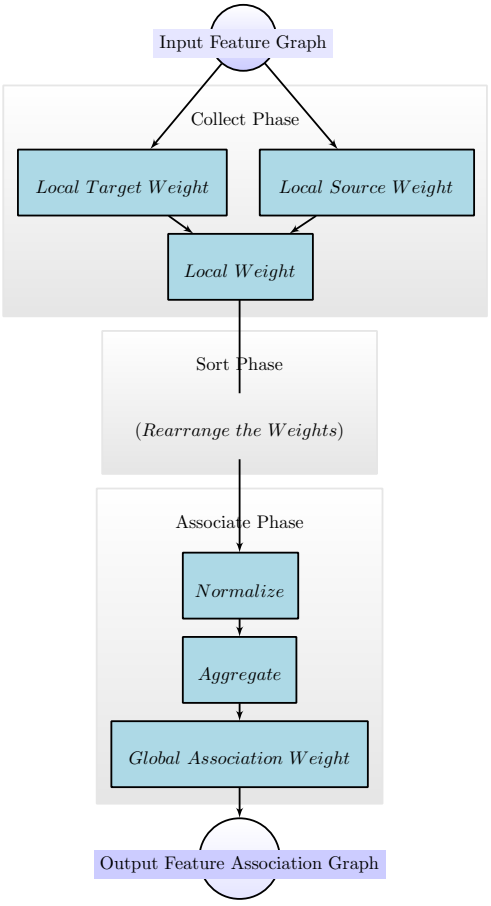

Figure 18: Overview of the sequence of execution of the different functional blocks. The functional blocks are either invoked in the collect phase (local associations) or in the association phase. The last block function determines the final association weight between a source and a target feature.

As input for the local source weight block operation the framework supplies the current instance node, the current feature node, as well as the edge that connects these two nodes. Additionally all meta-data that is attached to either the nodes or the weight is made available to the block implementation.

Formally a local source weight block operation can be written as function that takes an instance node i_k, a source feature f_s^{source} and an edge $e_{k,s}$ as argument:

$$B_{k,s}^{source} = f(i_k, f_s^{source}, e_{k,s}) \tag{43}$$

The result of the block operation is a weight, which is usually a scalar. Alternatively the block might not return a weight at all, which indicates that the current feature-instance relation should be excluded from any further feature association calculations. For example if the block implementation detects that a feature-instance edge in the input graph was erroneously generated or is just noise. In this case the no weight is produced by the block and the edge will be ignored.

Within the local source weight block algorithm any global or single feature factor can be employed. For example a simple realisation of this block may choose to return the weight the edge that connects the instance node with the source feature node. A more complex block implementation puts the weight into relation with the sum of all weights of all edges that connect any instance node with the current source feature node:

$$B_{k,s}^{source} = \frac{f_F^w(f_s^{source}, i_k)}{f_F^{\Sigma w}(f_s^{source})} \tag{44}$$

The numerator of this fraction is the weight of the edge. Operation to obtain this value is not associated with a high run-time overhead. The input graph is traversed in an order according to its internal sequence. To read out the edges and their attached meta-data (and weight) is therefore an cheap operation in terms of computation[209]. The denominator represents the sum of all weights of a all edges in the graph that connects the given feature with all nodes mapped to the instance role. As remarked in the factor section, the value of the weight sums is a valuable input for the pruning phase and is computed in the initialisation of the processing and then distributed to all execution units for fast access. The computational costs are therefore just the look-up operation in an associative array.

[209] Sequential access to stored data-structures is a key element to achieve a high performance on contemporary computer infrastructures.

Local Target Weight Block The block operation for the local target weight is similar to the local source weight block type. The only difference is the selection of feature nodes. While for the local source weight all nodes within the instance graph were selected which are mapped to the source feature role, for this block types all nodes a processed that are mapped to the target role.

The distinction between the two local weight blocks is still necessary because some feature association function are asymmetrical in relation to the two input feature roles. Therefore it should be possible to use different realisations for the local source and target weight blocks. The notation used for the local target weight block is:

$$B_{k,t}^{target} = f(i_k, f_t^{target}, e_{k,t}) \tag{45}$$

Local Association Block Before describing this functional block type, the term “local association” has to be defined first. While a (global) feature association denotes the relationship of two features within the whole data-set, the scope of a local association is determined by a single instance node.

For each instance node a a sub-graph of the feature input graph can be generated. This sub-graph consists of the instance node and all directly connected nodes, which are typically nodes that are mapped to the roles source feature and target feature. The feature associations within this sub-graph are called local associations.

For each pair of globally associated features there is at least one sub-graph, that contains the source as well the target feature. Therefore a shared instance node is a requirement for a pair of features to build an association. Two features without common instance node will be excluded from further processing. This is an important constrained imposed by the feature association framework. This limitation plays hardly any role in real-world applications, because an association can only exist if two feature have something in common. For a given application and data-set the feature inputs graph has to be modelled in such a way, that the instance nodes represent this commonness.

The local association reflects the shared information on a micro level. The corresponding weight represents the strength of a local association. To calculate the weight, the local association block is supplied with the local source and target weight as produced by their respective functional blocks. More formally, the local association block can be expressed as function signature:

$$B^{local}_{k,s,t}\left(\begin{array}{l} i_k, \\ f^{source}_s, e_{k,s}, B^{source}(i_k, f^{source}_s, e_{k,s}), \\ f^{target}_t, e_{k,t}, B^{target}(i_k, f^{target}_t, e_{k,t}) \end{array}\right) \tag{46}$$

Implementations of this block function have, like all other block implementations, access to all global statistics as additional source of information. If external information should be integrated into the association calculation one has multiple ways to achieve this. The first method is to gather the needed information in the initialisation of the processing and make the collected information available via the global statistics. The seconds alternative provide more flexibility. The collected information can be stored as additional meta-data at any node or edge in the feature input graph. This way the external data can be made available to the factors and functional blocks that contribute to the final feature association calculation. Not only external information can be expressed a meta-data, but properties of the data set itself can be encoded as meta-data.

For example, if a relation between a feature and an instance is annotated with a time-stamp within a data-set, this time information can be added as meta-data to the connecting edge. If the same feature is assigned to an instance multiple times with different time-stamp values, the edge meta-data can be filled with an array of time values. The block function can exploit this information for its calculation of the local association weight. When considering only the case of a single time-stamp per edge, a function that filters out all association with time differences of larger than a day could be modelled like this [210]:

[210] Assuming that the time-stamp has a resolution of seconds.

$$B^{local}_{k,s,t} = \begin{cases} 1, & \text{if } |e_{k,s}[timestamp] - e_{k,t}[timestamp]| < 86,400[\text{s}], \\ \emptyset, & \text{otherwise.} \end{cases} \tag{47}$$

In this example the result value of $\emptyset$ indicates that for the current instance sub-graph the local association should be ignored.

The local block functions are invoked by the framework during the collect phase. All local association as collected and fed into the distributed sorting algorithm. The collected data is re-organised in such a way that all necessary information for a single target feature is readily available.

Normalise Block Before the local association weights are merged to form global association, they can optionally be processed. This is applied on the sorted data at the beginning of the association phase. The motivation for the the normalisation process is to level out the local weight in relation to each other. The normalisation block functionality is invoked for each local association weight separately. Besides the local weight to be normalised the function is also supplied with all other local association weights for a given pair of source and target features. The signature of this function can by symbolised as[211]:

$$B^{norm}_{t,k,s} = f(B^{local}_{k,s,t}, \{\forall h\, B^{local}_{h,s,t}\}) \tag{48}$$

[211] Please note the change in the sequence of the indexes of the block function to indicate the change in the internal ordering. The local association weights are organised in sequence of instances, whereas the normalised weights are stored according to the sequence of target features.

For example the normalisation block can be used to transform all weights into a predefined range. This might be needed if any of the succeeding algorithms are only capable to process a limited range of numbers. For example, many similarity measures assume a weight between 0 and 1. The normalisation function can also be used to conduct a discretisation of the local weight, if for example the value needed to arranged in bins. The χ^2 test is an example for an function that relies on bins of equal sizes, where each bin spans a range of values. This statistical test of often used to test whether the distribution of a data sample is equals to an assumed distribution. In many cases the data is expected to follow a normal-distribution[212].

[212] If the data indeed resembles an normal-distribution, the data is suitable for a range of statistical tests, for example the Student's t-test to detect whether a difference in the values can be considered significant

The normalisation block function can also be employed to transform the data into a representation that suited for a family of algorithms that share the same input. Many machine learning algorithms assume the input data to be organised as a matrix. The cells in the matrix represent weights. Some algorithm impose constraints one the value range of these weights. The relation of the weights within the matrix sometimes need to follow certain rules. For example an algorithm that expects a stochastic matrix[213] as input, where all columns in the matrix need to sum to 1. A column in this matrix represents the transition probabilities between a state and all other states. The normalisation block can be implemented to produce such a matrix[214]. One example for an algorithm that expects the input data to be represented as stochastic matrix is the well known PageRank algorithm[215]. The goal of the PageRank algorithm is to identify the most important nodes in a directed graph that have many in-links from other important nodes. The PageRank approach has proven to provide usable results and exhibits an excellent scalability behaviour, which makes is suitable for even large scale data-sets.

[213] The term Markov matrix is also frequently used for this kind of matrix.

[214] In the algorithm 5 the normalisation block has be labelled as $MatrixWeight$ to emphasise the goal of the normalisation process.

[215] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998

After the local weights for a single target feature are normalised the implementation of the association phase can start with the processing of the associations. The input for these processing is a data-structure that contains all the normalised local weights. Because of the fact that most of the real-world data-structures are sparsely connected, the employed data-structures are associative-arrays:

$$\{f^{target}_t \mapsto \{\forall k\, i_k \mapsto \{\forall t\, B^{norm}_{t,k,s}\}\}\} \tag{49}$$

Aggregate The data-structure created within the association phase for the normalised local weights is fed into the association algorithms. The output of this processing is data-structure similar to the input, but the actual entries within the associative arrays may have changed. For example, if the PageRank algorithm has been applied on the normalised weights, the associative arrays holds the individual importance weights of the feature nodes.

In the case that for each pair of features multiple weights are stored within the data-structures, these value have to be finally merged into a single global association weight. The aggregate block represents the

first step in this process. The task of this functional block is to collect all available information of a pair of target and source features and to group this information into a single data-structure. If no sophisticated algorithm is applied on the normalised local weight and the data is just passed through, the input of an aggregation block function is a single target feature and a mapping of a source feature to the collected local normalised weights:

$$B_t^{aggregate} = f(f_t^{target}, \{\forall l\, f_s^{source} \mapsto \{\forall k\, i_k \mapsto B_{t,k,s}^{norm}\}\}) \tag{50}$$

A single invocation of the aggregate function contains all necessary information to build all associations for a given target feature. In the aggregation block function these information are first re-organised based on the source features. Thus the intermediate data-structure of the aggregation - $b_m^{aggregate}$ - can be symbolised as (again assuming that the normalised local weights are not modified by algorithms applied in the association phase):

$$b_t^{aggregate} = \{f_s^{source} \mapsto \{\forall k\, i_k \mapsto B_{m,t,s}^{norm}\}\} \tag{51}$$

This information is then aggregated into a single representation, a single aggregated weight - $w_{t,s}^{aggregate}$. Although the realisation of the feature association function is free to choose any aggregation method and any representation of a weight, usually the aggregation weight simply will be a single scalar for each tuple of source and target feature. Out of the possible aggregation methods, the two most common are:

- *The sum of all weights*: A single scalar is generated by iterating over all weight values for a pair of target and source feature nodes. Thus the aggregated feature association weight is the sum of all weights[216]:

$$w_{t,s}^{\Sigma aggregate} = \sum_k \underbrace{b_t^{aggregate}[f_s^{source}][i_k]}_{\text{equal to } B_{t,k,s}^{norm} \text{ for baseline case}} \tag{52}$$

  To illustrate this aggregation method an example is presented, where the feature input graph consists of feature nodes which represents random events. These source events are connected to a number of instance nodes with an edge that carries the conditional probability of the instance given the source event - $P(i_k|f_s^{source})$. The instance nodes are additionally connected to a set of target feature nodes, which like the source features represent a set of random events. The edges of this relationship may also carry a weight, which in this example is assumed to uniformly 1 for all connections of this type. There is no limitation on the number of instance nodes, as well as target feature nodes.

  If all functional blocks are implemented to simply pass their input values to the next block[217], the normalised local weight will be $B_{t,k,s}^{norm} = P(i_k|f_s^{source}) \cdot 1$. Again assuming no modification on the normalised weight are conducted in the association phase, the intermediate block weights will be $b_t^{aggregate} = \{f_s^{source} \mapsto \{\forall k\, i_k \mapsto P(i_k|f_s^{source})\}\}$ for all target features. The sum of all weights aggregation method will iterate over all normalised weights, while incrementing a summation value. Formally this can be expressed as:

$$w_{t,s}^{\Sigma aggregate} = \sum_k P(i_k|f_s^{source}) \quad \equiv \quad P(f_t^{target}|f_s^{source}) \tag{53}$$

  The final aggregated weight will then be equivalent to the conditional probability of an target event given a source event.

- *The average of all weights*: The second common aggregation method calculates the arithmetic mean of the weights[218]. The resulting aggregation

[216] In the baseline case there are no modification on the local normalised weight.

[217] In the local association block the two local weights (source and target) are multiplied, or alternatively the local association block just returns the local source weight resulting in the same result.

[218] Other functions to calculate an average value (geometric mean, harmonic mean, median, ...) are not presented here, but of course may be used by an feature association function.

weight can be derived from the $w_{m,l}^{\Sigma aggregate}$ by introducing an additional normalisation factor:

$$w_{t,s}^{\mu aggregate} = \frac{\sum_k b_t^{aggregate}[f_s^{source}][i_k]}{|b_t^{aggregate}[f_s^{source}]|} \tag{54}$$

To illustrate this aggregation method, an dice throwing experiment is presented. The experiment consists of multiple persons and a dice. The dice which will always yield a number between one and six[219]. The goal of this experiment is to find out whether certain people have a tendency to throw specific numbers.

[219] It will never remain standing on an edge or will suddenly drawn into another parallel universe.

To build a feature input graph, for each individual throw an instance node will be added. The persons are represented as source feature nodes and the outcome are modelled as target features. Thus the hyper-edge that connects a source feature node with single target feature node while traversing over a selected instance node will represent a single throw event. The weight values for all the edges are initialised with 1.

The local source weight block and the target weight block will not be used in further processing, therefore it may safely return 1 for all invocations. The combined local weight block function also returns a constant value, regardless of the input value: $B_{k,s,t}^{local} = 1$. The association phase does not need to change the local weight or trigger any additional processing. This results in an intermediate aggregation block weight of $b_t^{aggregate} = \{f_s^{person} \mapsto \{\forall k\, throw_k \mapsto 1\}\}$.

Finally the aggregation method can be applied on the intermediate representation, which calculates the arithmetic mean of all local weights for a combination of person and number. This will result in the proportion of throws a person has yielded a specific outcome number.

$$w_{t,s}^{\mu aggregate} \equiv \frac{\#Throws_{person}^{number}}{\#Throws_{person}} \tag{55}$$

Without making any further assumptions it is expected that the aggregate average weight for all person-weight association will be about equal, for a sufficiently large number of throws. Any deviation from this distribution indicates that the person might use a special technique to trigger specific outcomes.

The framework triggers the execution of the aggregation functional block, and also collects a series of aggregation statistics, like for example the number of instances that are shared between the source and target feature. Although the result of the aggregation block function appear to be valid as an final association weight between two features, it will be further relayed to another functional block.

Global Association Weight The global association weight block is the final step in the feature association weight calculations. It is invoked in the association phase after the aggregated weight has been further refined by an optional customisation functionality. The task of this function block is to produce the final association weight between two features. The weight usually reflects the strength of the association relationship between a source feature and a target feature.

The global association weight block takes two features and an aggregated weight as argument. Additionally the implementation might access any global statistics and the aggregation statistics. Formally the global association weight function can be written as:

$$B_{t,s}^{global} = f(f_s^{source}, f_t^{target}, B_t^{aggregate}[f_s^{source}]) \tag{56}$$

The output of the global weight calculation is a weight, which will be

a scalar for most of the use-cases. An application might also choose to use a more complex form to represent the association strength. The weight is then used for the edge that connects the source feature with the associated target feature. If the association between two features should not be part of the output feature graph, the global association weight function should return no value at all for this combination of features. The final output data-structure of the this function block is an associative array. The keys in this array are the target features, the values is a set of tuples. Each tuple in the set is consists of the associated target feature together with the association weight:

$$\{f_t^{target} \mapsto \{\langle f_s^{source}, B_{s,t}^{global} \rangle\}\} \tag{57}$$

A major use-case of the feature associations is to find all associated target features for a given source feature. Therefore one would rather expect that the key of this associative array to be the source feature. For the construction of the final output feature association graph it does not matter how the information is organised to create the association edges. The reason for choosing the target feature as the main access to the association information is rooted in the design decision to allow a fast retrieval. More specially the sort criteria employed in the sort phase of the algorithm determines how the data is internally organised, with the sorting according to the target feature being the default.

The decision to focus on the target features as primary sort criterion instead of the source features is motivated by the expected storage technology to be used. An efficient and scalable store data-structure is needed to manage the output feature association graph. This data-structure does not only have to store the information which nodes in this graph are connected, but also needs to account for the association weight and other additional meta-data[220]. Out of the possible candidates for the storage of the feature association graph, the inverted index appears to be sensible choice, as this data-structure will also often be the input data-structure. When an inverted index is used as storage for the association relations, the direction of these relations is reversed. This is the reason why the global association weight block produces associations in the reverse direction by default. If another type of storage technology is employed, which does not reverse the direction of the associations, the sort criteria needs to be changed and the functional block executed during the association phase need to be adapted[221].

[220] For example a list of instances that contributed the most to the final association weight.

[221] Feature association function that generate symmetric associations are not affected by such a change.

The illustrate the way the different functional block interact with each other, the output of the global association block for a basic configuration is presented. Starting with a feature-instance weight of 1 for all edges, all local functional blocks merely return the input value. The local association weight function calculates the product of the source and target feature block weights. Thus the value $B_{k,s,t}^{local}$ is equal to 1 for all input values that are then feed to the normalisation block function, which just further dispatches this values to the weight aggregation. The aggregation block simply adds a values for an association and the result is delivered as input to the final global association weight processing. If the global association weight function does not modify this value, the final result for the association weight for all source and target features will be (with I being the set of all instance nodes and E the set of all edges in the input feature graph):

$$B_{s,t}^{global} = |\{i_k | i_k \in I \cap \langle i_k, f_s^{source} \rangle \in E \cap \langle i_k, f_t^{target} \rangle \in E\}| \tag{58}$$

For this basic definition of the functional groups the final association weight is equivalent to the number of shared instance node of the source and target features[222]. Although the execution sequence of the functional

[222] If the average-of-all-weights aggregation function were used instead of the sum-of-all-weights function, the final association weight would be 1 for all associations.

blocks and their input and output data-structures are fixed, a wide array of feature association function can be adapted for the feature association framework.

Examples of the Feature Association Functions

The process of decomposing a feature association function into factors and functional blocks can be tedious task. This is especially true for novice users. Therefore the feature association framework provides a collection of already decomposed feature association functions. This functions are selected because of their usefulness in various feature association settings. Many of there have been used in knowledge discovery applications in the past. A selection of functions are presented in the follow section in detail with references to the applications they have been used.

Because of the broad spectrum of functions they are grouped into categories of similar functions. An overview of the groups of presented functions given at first.

- *Similarity Measures:* The features to be associated are compared by applying a similarity measure, for example the cosine similarity.
- *Distance Metrics:* If features are represented as points in an n-dimensional space, one can calculate various distance metrics.
- *Statistical Tests:* In the field of statistics tests have been developed to compare different distribution or samples.
- *Probability Based Functions:* Features can also been seen as random events and as such their probabilities can be used as base for an feature association function.
- *Entropy Based Functions:* Algorithms from the the field of information theory can be applied on the input data-set to generate feature associations.

Similarity Based Functions

The first set of feature association functions are measures of similarity. In contrast to the other categories of functions the similarity measures cannot be attributed to a single discipline or domain. Some were developed in the field of statistics, physics or computational science and some may in fact be additionally be attributed to any of the other group of functions. But they all share common semantics, as the result of the function resembles a similarity value.

Cosine Similarity The cosine similarity is probably the most popular similarity measure used in knowledge discovery applications[223]. Therefore this feature association function will be discussed in detail. The function will serve as an example on the decomposing into factors and functional blocks.

[223] P. N. Tan, M. Steinbach, V. Kumar, and Others. *Introduction to data mining*. Pearson Addison Wesley Boston, 2006

The cosine similarity is often used in conjunction with the so called vector space model (VSM), which has been adopted for applications in the domain of information retrieval (although it is not entirely clear how this actually happened[224]). In the vector state model each record in the data-set is interpreted vector in a n-dimensional space, hence the name. Usually the data-set is sparse, which results in the fact that most of the dimensions of a record will stay empty (or zero). Traditional approaches to measure the distance do not work well for such data (for example the Euclidean distance). Therefore instead of using the distance of two points, the angle between the vectors that connect the origin with the respective points. Even

[224] D. Dubin. The most influential paper Gerard Salton never wrote. *Status: published or submitted for publication*, 2004

if the vectors vary in length, due to missing dimensions, the angle will be largely unaffected.

Applying the cosine function on this angle results in a measure, which produces results in the range of $[-1, +1]$. For orthogonal vectors the cosine similarity will be O. If all coordinates are positive the range of the measure will limited to $[0, +1]$, which is usually the case when the dimensions represent words and their value is the number of times the word occurs in the text.

Formally the cosine similarity between two vectors - f_1 and f_2 - can be expressed as:

$$\cos\theta = \frac{\mathbf{f_1} \cdot \mathbf{f_2}}{\|\mathbf{f_1}\| \, \|\mathbf{f_2}\|} \tag{59}$$

If the two vectors are unit vectors[225], the denominator can be spared and the cosine similarity can be calculation by calculating the cosine of the dot product between v_1 and v_2.

[225] An knowledge discovery application might choose to scale all vectors to unit length at the beginning of the processing.

To apply the cosine similarity for calculating of an association weight between two features, these features need first to be represented as vectors. In traditional information retrieval applications the documents are vectors, while the words or terms resemble the dimensions. In the case of feature associations the roles are exchanged, the features are mapped into a vector representation and the instances serve as dimensions. For the values of each dimension the weights of the edges that connect instances and features are taken. Therefore the weights need to be numbers for the cosine similarity to be applicable.

Generally speaking the cosine similarity will then high for features that share many instances, and low for features with non-overlapping set of instances[226]. In order to use this similarity measure it first needs to be decomposed into the two components of a feature association function: the factors and the functional blocks. While the decomposition into factors is fairly simple in this case, the mapping of the cosine similarity into the block functions appears to be a bit more complex. The cosine similarity for two factors - f_s and f_t- can be expressed by the means of factors like this:

[226] This is based on the assumption that the weights are either all equal or at least all positive.

$$\cos\theta = \frac{\overbrace{\sum_{i_k \in I} \mathfrak{f}_F^w(f_s, i_k)\mathfrak{f}_F^w(f_t, i_k)}^{\mathbf{f_1 f_1}}}{\underbrace{\sqrt{\mathfrak{f}_F^{\Sigma w^2}(f_s)}}_{\|\mathbf{f_1}\|} \underbrace{\sqrt{\mathfrak{f}_F^{\Sigma w^2}(f_t)}}_{\|\mathbf{f_2}\|}} \tag{60}$$

The cosine similarity is now successfully mapped into a combination of factors[227]. The next step is to find a partition of the cosine similarity that matches the sequence of function blocks. The process of mapping a function onto functional block is not unique, there a multiple ways on how to define the block functions. The decision which mapping scheme is realised is influenced by performance considerations.

[227] To improve the performance of the association calculations, the framework is configured to compute factors in advance and make the results available to all factors and functional blocks

The numerator of the cosine similarity as given in formula 60 can be calculated in part by using the local association blocks. The local source weight block employs the source weight factor to calculate a weight. The same is done for the local target weight. These two local weights are then combined by the local association weight. The local weights of the source feature f_s and target feature f_t depend on a single instance node i_k and can be defined as:

$$B_{k,s}^{source} = \text{ſ}_F^w(f_s, i_k) \tag{61}$$

$$B_{k,t}^{target} = \text{ſ}_F^w(f_s, i_k) \tag{62}$$

$$B_{k,s,t}^{local} = B_{k,s}^{source} B_{k,t}^{target} \tag{63}$$

The normalisation block could be used to modify the local weight[228], but to keep the necessary operations minimal the normalisation calculations are postponed to an later block processing. To calculate the complete numerator of the cosine similarity, the sum of all instance specific weights needs to be calculated. This is the task of the aggregation block. More specifically, the sum-of-all-weights aggregation method produces the desired result. The implementation of the normalisation and aggregation functional blocks can therefore be expressed as:

[228] by dividing it by the length of the two features

$$B_{t,k,s}^{norm} = B_{k,s,t}^{local} \tag{64}$$

$$B_{t,s}^{aggregate} = \sum_k B_{t,k,s}^{norm} \tag{65}$$

After the complete numerator is handled by the aggregation block, the denominator still remains to be integrated to end up with a correct implementation of the cosine similarity. The global weight block is responsible to combine the output of previous blocks and to conduct the normalised to unit length. The cosine similarity is symmetric in regard to the input vectors, therefore the global association weight is the same for both association directions.

$$B_{t,s}^{global} = B_{s,t}^{global} = \frac{B_{t,s}^{aggregate}}{\sqrt{\text{ſ}_F^{\Sigma w^2}(f_s)}\sqrt{\text{ſ}_F^{\Sigma w^2}(f_t)}} \tag{66}$$

The global association weight has been successfully decomposed into factors and functional blocks. The feature association framework already provides an implementation of a cosine similarity and therefore any application may use this function without further decomposition steps.

Weighted Cosine Similarity The cosine similarity can be further extended and improved by introducing and additional weighting scheme. Although the weighting scheme is usually applied outside the feature association framework, an example of an integrated weighting will be presented because of its relevance in real-world applications.

The TFIDF approach has become the de-facto standard for many applications in the area of knowledge discovery. It is often combined with the Vector Space Model and serves as the basic implementation of many information retrieval solutions. In this context the Vector Space Model is build based on vectors that represent documents and dimensions that represent terms. Each document consists of a list of terms, where each term may occur multiple times. The number of times a term is listed within a document is the local weighting aspect of TFIDF. The global weighting part is motivated by the intuition that rare terms should have a higher influence on the final weight. To capture this intuition at first the document frequency is determined, which is defined as the number of documents a given term occur in at least once. Although the reciprocal value of the document frequency would be sufficient to satisfy the intuition, it is made robust to changes in the size of the data-set by adding the total number of documents as numerator. Finally the logarithmic value of this ratio is taken as final

global weighting, which is referred to as Inverse Document Frequency.

The combination of the local and global weighting aspects gives the final TDIDF weighting scheme as defined in formula 69. In this equation the document is labelled as i_k, which consists of a list of terms which represent occurrences of the given feature f_s.

$$TF(f_s, i_k) = |\{o_{s,k} | o_{s,k} \in i_k \cap o_{s,k} \in Occur(f_s)\}| \tag{67}$$

$$IDF(f_s) = log(\frac{|I|}{|\{i_k | i_k \in I \cap \langle i_k, f_s \rangle \in E\}|}) \tag{68}$$

$$TFIDF(f_s, i_k) = \underbrace{TF(f_s, i_k)}_{\text{Local Weight}} \underbrace{IDF(f_s)}_{\text{Global Weight}} \tag{69}$$

When building an input feature graph, the number of occurrences is mapped as weight of an relation between an instance and a feature. To be used within the feature association framework the weighting scheme needs to be decomposed into factors. For the TFIDF weighting the factors already supplied by the framework can be re-used. With a combination of these factors a new TFIDF weighting factor can be defined as:

$$\mathfrak{f}_F^{tfidf}(f_s, i_k) = \underbrace{\mathfrak{f}_F^{w}(f_s, i_k)}_{TF(f_s, i_k)} \underbrace{log(\frac{\mathfrak{f}_I^N}{\mathfrak{f}_F^N(f_s)})}_{IDF(f_s)} \tag{70}$$

To integrate the weighting scheme into a feature association function, any references to the $\mathfrak{f}_F^w(f_s, i_k)$ factor need to be replaced by the new weighting factor $\mathfrak{f}_F^{tfidf}(f_s, i_k)$. Applied in the the global weighting block of the cosine similarity is an example of the integration of a weighting scheme into an existing feature association function[229]:

$$B_{t,s}^{global} = B_{s,t}^{global} = \frac{\sum_k \mathfrak{f}_F^{tfidf}(f_s, i_k) \mathfrak{f}_F^{tfidf}(f_t, i_k)}{\sqrt{\mathfrak{f}_F^{\Sigma tfidf^2}(f_s)} \sqrt{\mathfrak{f}_F^{\Sigma tfidf^2}(f_t)}} \tag{71}$$

[229] The factor $\mathfrak{f}_F^{\Sigma tfidf^2}(f_x)$ is the sum of all squared TFIDF weights for a given feature f_x

The enhancement of the cosine similarity measure by integrating a weighting scheme serves as a demonstration the flexibility of the combination of factors and functional blocks. For the following feature association function the decomposition into factors and functional block will be omitted for sake of brevity.

Jaccard Similarity Coefficient In contrast to the cosine similarity which operates on a representation of the feature as vectors in the Euclidean space, the Jaccard similarity operates on sets. The Jaccard similarity is defined as ratio between the size of the intersection and the union of the two sets.

Equation 72 presents the formal definition of Jaccard similarity with I_s and I_t being two sets:

$$Jaccard(I_s, I_t) = \frac{|I_s \cap I_t|}{|I_s \cup I_t|} \tag{72}$$

Due to the definition of the Jaccard similarity measure, it only applicable on binary features. The limitation implies that the weight of the edges which connect an instance with a feature node in the feature input graph are going to be ignored. The existence of a the edges alone serve as determining factor whether a instance is included in a feature specific set. Formally these sets can be constructed for each feature f_s with I being the set of all instances and E the set of all edges[230]:

$$I_s = \{i_k | i_k \in I \cap \langle i_k, f_s \rangle \in E\} \tag{73}$$

[230] Although only the source feature f_s is listed here, the same applies to the target features f_t as well.

The similarity value produced by the Jaccard measure are in the range $[0, 1]$, where a value of 1 indicates that the two features share exactly the same instances and a value of 0 is produced if the two features have no instance nodes in common.

Overlap Coefficient Similar to the Jaccard similarity measure, the Overlap coefficient operates on sets. The two measures deviate in the choice of the denominator. The overlap coefficient puts the magnitude of the intersection set in relation to the size of the smaller of the two sets:

$$Overlap(I_s, I_t) = \frac{|I_s \cap I_t|}{min(|I_s|, |I_t|)} \tag{74}$$

The Overlap coefficient shares many properties with the Jaccard measure. The output range of $[0, 1]$, where a higher number reflects a higher degree of similarity. Like the Jaccard similarity, the Overlap coefficient only applies to binary features.

Dice's Coefficient Similar to the Jaccard similarity and the Overlap coefficient, the base of the Dice's coefficient is the number of shared instances of two features. To calculate the Dice's coefficient the size of the intersection is multiplied by two and divided by the sum of the number of elements in the two sets.

$$Dice(I_s, I_t) = \frac{2|I_s \cap I_t|}{|I_s| + |I_t|} \tag{75}$$

The Dice's coefficient will be close to 1 for highly overlapping sets and near to zero, if the two sets share only few elements. Like the other two set based similarity measures, the Dice's coefficient can only be calculated for binary features. If the input feature graph is weighted, one can apply an threshold on the weights to transform them into a representation suitable for this family of similarity measures.

Although the three presented set based similarity measures appear to be redundant, depending on the application scenario they may provide different results. A detailed graphical analysis of the different similarity measure is given by Jones and Furnas[231].

[231] W. P. Jones and G. W. Furnas. Pictures of relevance: A geometric analysis of similarity measures. *Journal of the American society for information science*, 38(6):420–442, 1987

Pearson's Correlation The Pearson product-moment correlation coefficient has been developed to compare two variables and to test their independence. Therefore this measure is well suited for the task of identifying relevant feature associations.

Although the Pearson's correlation can be applied on distributions, in the context of the feature association framework the data-set usually resembles samples.

In this case the necessary input data for the correlation coefficient need to be estimated. For both variables the mean and standard deviation need to be determined. Therefore the Pearson's correlation can only be calculated, if the data-set is dense - there are no missing values. Within the input feature graph there need to be a connection between each instance node and each feature node. If the weight information is not available for some relations it needs to be substituted[232].

Given a feature f_s and a set of weighted relations, the estimated mean is the arithmetic average of all weights. For the standard deviation the commonly used estimation formula can be expressed via factors as:

[232] Assuming a weight of zero is one way to achieve this, although for very sparse data-sets this does not lead to useful results as the mean and the standard deviations will be close to zero for the majority of the features.

$$\overline{w_x} = \frac{1}{\mathfrak{f}_F^{in}(f_x)} \sum_{i_k | i_k \in I} (\mathfrak{f}_F^w(f_x, i_k)) \tag{76}$$

$$s_x = \sqrt{\frac{1}{\mathfrak{f}_F^{in}(f_x) - 1} \sum_{i_k | i_k \in I} (\mathfrak{f}_F^w(f_x, i_k) - \overline{w_s})^2} \tag{77}$$

Given the definitions of the mean and standard deviation the Pearson product-moment correlation coefficient coefficient is written as:

$$Pearson(f_s, f_t) = \frac{1}{|I| - 1} \sum_{i_k | i_k \in I} \left[\left(\frac{\mathfrak{f}_F^w(f_s, i_k) - \bar{w_s}}{s_s} \right) \left(\frac{\mathfrak{f}_F^w(f_t, i_k) - \bar{w_t}}{s_t} \right) \right] \tag{78}$$

The Pearson correlation coefficient produces values in the range of $[-1, +1]$, where values close to $+1$ indicate a high degree of dependence between the two features. This happens if both features share a similar weight distribution in relation to the instances. In other words, if the weight of a feature-instance relationship is high for the source feature, it will be high for the target feature as well. If the two features are completely independent, the correlation coefficient will be zero. Negative values indicate that the two features inversely correlate. This can be observed if the weight of the source feature is high for a given instance, the weight of the target feature will be low and vice versa[233].

The Pearson's coefficient can also be interpreted as the angle between the axis of variation of the two feature distributions. Therefore this measure is useful to gain a better understanding of the features within a data-set and it represents a valuable tool for the work of feature analysis[234].

Besides the Pearson's correlation that is a number of other correlation functions, most notably the Spearmen's rank correlation coefficient and the Kendall τ. These two correlation measures do not operate on the feature weights directly, but on a ranked list of feature-instance weights[235]. These type of correlation measures is chosen, if only the absolute ranking of the variables is known. In the case of feature associations the Pearson's correlation is generally preferred to these measures, as the weights contain more information as the ranking position and the ranking itself might be non-deterministic (in the case of ties).

[233] The pairs of features represent a kind of XOR like distribution pattern.

[234] Unfortunately this only applies to 'fully-connected' data-sets.

[235] The feature-instance relationship with the highest weight is top-ranked, followed by the second highest weight and so forth.

Tanimoto Correlation The final example for the family for similarity function is the Tanimoto correlation. This measure is not as well known as the other presented measures, but it is included due to its unique properties and its close relationship to other similarity functions. The Tanimoto correlation can be either be seen as extension of the Jaccard index, as an enhancement of the Cosine similarity or as an hybrid version of the Cosine similarity and the Euclidean distance, which is covered in the following section.

Formally, the Tanimoto correlation is defined as relationship between the distribution of a source feature f_s and a target feature f_t:

$$Tanimoto(f_s, f_t) = \frac{\sum_k \mathfrak{f}_F^w(f_s, i_k) \mathfrak{f}_F^w(f_t, i_k)}{\mathfrak{f}_F^{\Sigma w^2}(f_s) + \mathfrak{f}_F^{\Sigma w^2}(f_t) - \sum_k \mathfrak{f}_F^w(f_s, i_k) \mathfrak{f}_F^w(f_t, i_k)} \tag{79}$$

In contrast to the other presented similarity measures, the range of possible output values of the Tanimoto coefficient is not bounded. For example, the magnitude of the resulting scalar for two identical features will depend on the actual feature-instance weight values. This can be seen

as one of the reasons why the Tanimoto similarity measure is not a widely adapted as for example the cosine similarity. The interpretation of the final feature association weight is less intuitive, but it still might be the preferred similarity measure for specific data sets.

Distance Functions

While the similarity measure in general produce low values for independent features and high values for features of similar distribution, the values produced by distance functions follow the reverse logic. They measure the distance of two points which represent the two features. A higher distance between the two points indicates a high degree of independence or dissimilarity.

Euclidean The Euclidean distance has already been covered in this chapter as an example for the factor that composes the weight of an feature-instance relationship. The features are interpreted as points in an n-dimension space, where the number of dimensions is equal to the number of instance nodes in the input feature graph. Thus each dimension maps to one instance. The weigh of the edge between an instance node and a feature node is the coordinate of a point in the n-dimensional space.

The Euclidean distance is also called L_2 norm and is formally defined for the 2-dimensional space as[236]:

$$L_2(i_1, i_2) = \sqrt{\begin{array}{l}(\mathfrak{f}_F^w(f_1, i_1) - \mathfrak{f}_F^w(f_1, i_2))^2 + \\ (\mathfrak{f}_F^w(f_2, i_1) - \mathfrak{f}_F^w(f_2, i_2))^2\end{array}} \tag{80}$$

[236] Please refer to the description of the feature weight factor for detains on the Euclidean distance.

The distance between two points will be zero for identical features, so that each of the two features share the same weights for the same instance nodes. The more the weights deviate, the higher the distance will be. Due to the definition of the distance, the L_2 norm will never yield negative results. Furthermore the Euclidean distance fulfils all requirements of the definition of a metric.

Definition of a metric:

1. The values are non-negative
2. A result of 0 only if both points are identical
3. The sequence of points does not matter (symmetrical)
4. Given three points, the sum of two distances is always larger (or equal) then the third distance (triangle inequality)

The Euclidean distance will be mainly used as feature association function if the data-set itself resembles point in an n-dimensional space. Although this is not the case for the many of real-world data-sets, the data may be transformed by preceding processing into an Euclidean space.

An application scenario for the usage of the Euclidean distance is the family of methods that are based on a k-NN based feature selection. This feature selection scheme starts with a single feature and tries to find the most similar related features. Thus starting with the position of the initial feature the algorithm searches for neighbouring points. A the criteria to define the proximity of points a distance measure is used, in this case the Euclidean distance. Starting with the closet neighbour the algorithm collects as many features until a threshold value is reached. This threshold is usually an upper bound of neighbouring features. Hence the name of this algorithm - k-nearest neighbours.

L_1 Norm The L_1 norm is also known as cab distance, Manhattan distance or city-block distance. These names give a good impression on how the L_1 norm operates.

The L_1 norm is closely related to the Euclidean distance. Their definitions are therefore similar. The $L1$ is defined for the 2-dimension space as:

$$L_1(i_1, i_2) = \begin{array}{l}(\mathfrak{f}_F^w(f_1, i_1) - \mathfrak{f}_F^w(f_1, i_2)) + \\ (\mathfrak{f}_F^w(f_2, i_1) - \mathfrak{f}_F^w(f_2, i_2))\end{array} \tag{81}$$

Although the L_1 and L_2 appear to be nearly identical, they have different properties which makes them suitable for different use-cases. For example, the L_2 is guaranteed to produce positive distance values, the L_1 norm may also generate values.

Minkowski Distances The L_2 and the L_1 norm are examples of the family of so called Minkowski distances. The Minkowski distance is a more general definition of distance and features an additional parameter p. The Euclidean distance is equal to the Minkowski distance with $p = 2$ and for the L_1 norm the parameter is equal to 1. Given two features f_s and f_t, the Minkowski distance can be formally described as.

$$Minkowski_p(f_s, f_t) = \left(\sum_{\{i_k | i_k \in I\}} |\mathfrak{f}_F^w(f_s, i_k) - \mathfrak{f}_F^w(f_t, i_k)|^p \right)^{1/p} \tag{82}$$

Different values of p yield different properties. For specific applications an specialised value of p is beneficial.

Levenshtein Edit Distance Until now all presented feature association function were based on the weight of the connection between instances and features. The Levenshtein edit distance serves as an example that other parts of the feature input graph can be utilised to build meaningful feature associations. In this case the meta-data of the feature nodes are analysed to build a similarity between individual features. The algorithm expects that one of the attached meta-data is a textual representation of the feature. For textual data-sets with will usually be a single word or phrase.

Starting with two strings, the Levenshtein edit distance calculates the number of edit steps required to transform the first string into the second. Each edit step may be an insertion of a character, a removal of a character or an exchange of a single character. Whereas the Levenshtein edit distance determines the minimal number of steps, as there infinitesimal many possible variations of edit steps. So for example the edit distance between 'mouse' and 'mice' is 3 (remove 'u', replace 's' with a 'c', replace 'o' with 'i'). A formal representation of the Levenshtein edit distance is omitted here, but there are many demonstration implementations available online, for example `http://www.merriampark.com/ld.htm` (and an open-source Java implementation is also available: `http://www.merriampark.com/ldjava.htm`).

To use the Levenshtein distance as feature association function, one can create a input feature graph that consists of all features connected to a single instance node. That way all features will be associated will all other features in the output feature association graph with the edit distance as association weight. This is not the most efficient way to find similar written features, as the resulting graph is fully connected, which is usually not needed and takes a considerable time to construct. Alternatively the input feature graph can be modelled to consist of instances which resemble prefixes, for example the first character of each word. This way only features that share the same prefix will be associated with each other. Using character n-grams or syllables are other possible approaches to group the input features into categories. The best representation of the input feature graph relies on the application scenario.

SoundEx The Levenshtein edit distance is way to find similarly written features. When the SoundEx algorithm is used as feature association function, features will be grouped according to the way they are pronounced (in the English language). To use this algorithm the features are expected to be

annotated with meta-data that resembles a textual representation. Usually this will be single words or a sequence of words.

The SoundEx algorithm is particularly useful if the features which identify named entities. Person names, organisation names and location names a typical examples of such named entities. Especially for entity names from foreign countries and different languages there is often no single canonical spelling. Therefore multiple possible ways to write a name may exist and even in one data-set, different spellings might be used. This causes problems and inaccuracies in knowledge discovery applications. To align different spellings into one representation, the sound of a name can be used as criteria.

For real-world data-set the 'reverse' problem exists too. There are word that are written equally, but pronounced differently. For example the name 'Berkeley' is pronounced differently. The surname of the philosopher George Berkeley (/*bɑrkli* :/) is pronounced differently than a well known city in California (/*bɛrkli* :/)

The SoundEx transforms a word into a sequence of characters that uniquely describe the way this word is articulated. Both names yield the same SoundEx code: `G500`.

It has been used by the US Census at the beginning of the 20th century and is still used by genealogists to capture spelling variations of the same surname. The original SoundEx algorithm has a couple of limitations, for example the resulting code has a fixed length regardless of the length of the input word. Therefore multiple schemes to improve SoundEx have been proposed, as well as multiple alternative approaches. For example the NYSIIS[237] (New York State Identification and Intelligence Algorithm), Metaphone[238] and Double Metaphone[239].

[237] R. L. Taft. *Name search techniques.* Bureau of Systems Development, New York State Identification and Intelligence System, 1970

[238] L. Philips. Hanging on the metaphone. *Computer Language Magazine*, 7(12):38–44, 1990

[239] L. Philips. The double metaphone search algorithm. *CC PLUS PLUS USERS JOURNAL*, 18(6):38–43, 2000

Functions Based on Statistical Tests

Statistical tests are a natural match to capture the strength of feature associations. They have been developed to measure how likely the output of an observation is in relation to a predefined hypothesis. The reference for the observation is called null hypothesis. The output of a statistical test is the significance level to which the observation matches the null hypothesis, or an alternative hypothesis is more likely.

To utilise statistical test within the feature association framework, the input data must first be transformed into a representation suitable for the significance tests. A pair of source and target features are assumed to be two random variables. Each of the variables has two states, which resemble a binary value that is true, if an instance node is connected to the feature and false, if not. Using this scheme it is possible to build a contingency table by counting the number of times each of the states occurs. The final contingency table is a 2×2 matrix as in table 17 with I being the set of all instance nodes. The null hypothesis for the statistical tests is the assumption that both features, source as well as target, share a common distributed in regard to the instance nodes. The resulting significance level can then be seen as strength of the association between the two features.

	Source Feature	**Target Feature**
Number of connected instance nodes	$\mathfrak{f}_F^{in}(f_s)$	$\mathfrak{f}_F^{in}(f_t)$
Number of remaining instance nodes	$\|I\| - \mathfrak{f}_F^{in}(f_s)$	$\|I\| - \mathfrak{f}_F^{in}(f_t)$

Table 17: Contingency table used as input for the statistical tests. The output of such an significance test indicates, whether the two features share a common distribution.

To simplify the formula and the discussion of the presented statistical test, symbols will be used to refer to the individual cells within the contingency table instead of referring to the factors itself. The symbols are listed in table 18.

χ^2 Test Out of the various possible statistical test, the χ^2 test[240]is one the

[240] The χ^2 test is also called chi spare test.

	Source Feature	Target Feature
Number of connected instance nodes	a	b
Number of remaining instance nodes	c	d

Table 18: Symbols for the cells of the feature contingency table. These symbols are using in the equations of the consecutive descriptions of the statistical tests..

most popular. The general formula to test the independence of two variables is given as, with $O_{i,j}$ being the observed count and $E_{i,j}$ being the expected counts of the null hypothesis:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}} \tag{83}$$

The feature contingency table can be used as base for applying the so called fourfold test, which is a special case of the χ^2 test. By using the symbols from table 18 the χ^2 test for feature contingency table can be written as:

$$\chi^2 = \frac{(ad - bc)^2(a + b + c + d)}{(a + b)(c + d)(a + c)(b + d)} \tag{84}$$

The result of this formula can be used to look up the significance level for one degree of freedom.

The significance level of the χ^2 test will only be reliable is a number of requirements are fulfilled. Most importantly there must be a sufficient of observations for each state. The generally accepted rule of thumb is to have a minimum frequency of 6 for each cell.

The presented method to use the χ^2 test is not the only possibility. If another null hypothesis is to be tested, the statistical test needs to be adapted. For this alternative version of the χ^2 test the co-occurrences will used again as base for the observations. Thus the table for the observed frequencies will be a 2×2 matrix and given in table 19.

	¬Source Feature	Source Feature
¬Target Feature	none	only source
Target Feature	only target	both

Table 19: Co-occurrence matrix of a source and a target feature. The first column represents the number of instance nodes not connected to the source feature, whereas for the second column instance nodes are counted that are connected to the source feature. The rows are used to denote the same relationship for the target feature.

To employ a statistical test, a matrix with the expect frequency needs to be constructed. Within this matrix the counts will be determined by the estimates based on the null hypothesis. In this case the independence of the two features is taken as null hypothesis. If the two features are truly independent, the occurrence of one feature does not influence the occurrence of the other feature. This must be true for every instance node.

If two variables are truly independent, their joint probability $p(x, y)$ is the result of co-occurrences that are purely caused by chance and thus $p(x)p(y)$.

Based on this assumptions, the 2×2 matrix for the expected frequencies can be constructed, based on the number of instance nodes N and the probability estimates for the two features, $p(f_s)$ and $p(f_t)$ as given in table 20.

	¬Source Feature	Source Feature
¬Target Feature	$N(1 - p(f_s))(1 - p(f_t))$	$Np(f_s)(1 - p(f_t))$
Target Feature	$N(1 - p(f_s))p(f_t)$	$Np(f_s)p(f_t)$

Table 20: Matrix of a expected frequency counts if both features are independent. To calculate the frequencies an probability estimate is put in relation with the total number of instances.

The χ^2 test can then be applied on the two matrices. The degrees of freedom are determined by the size of the matrices, by $(|Columns| - 1)(|Rows| - 1)$. Thus for the 2×2 matrices there is one degree of free-

dom. Finally the significance level can be computed that indicates the independence of the two features.

The χ^2 test is a popular choice to measure the association strength between features. There are many examples for knowledge discovery application that make use of this statistical test. For example the chi square test has been employed to identify keywords in text[241].

[241] Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157–170, 2004

Fischer Exact Test The Fisher exact test is an alternative to the χ^2 test as both are targeted at the same problem. One problem of the chi square test is that is not reliable if the number of counts is low. The Fisher exact test is more precise if there are a limited number of available observations. This advantage is coupled with a drawback. The run-time complexity of the Fisher exact test is higher. Therefore in the past the application of the exact test was hardly ever feasible. With the increase in computational power in recent years the Fisher exact test is now a valid alternative to the less precise χ^2 test.

When using the symbols from table 18 the Fisher exact test can be formally written as:

$$Fisher = \frac{\binom{a+c}{a}\binom{b+d}{b}}{\binom{a+b+c+d}{a+b}} \tag{85}$$

From the definition one can deduce that the computation of the Fisher test needs far more processing instructions than the χ^2 test. Therefore the exact test is only conducted for infrequent features, for all other features the chi square test provides sufficiently good results.

Log Likelihood Test As the name implies the log likelihood test operates on estimates of the probabilities of two hypothesis. The likelihood of the null hypothesis is put in relation to the alternative hypothesis.

There is a close relationship between the log likelihood test and the χ^2 test. In many cases it is hard to compute the likelihood values. If it is possible to derive the null hypothesis from the alternative hypothesis, the the chi square test can be used as an approximation for the log likelihood test.

$$LogLikelihood = -2\sum_{i=1}^{r}\sum_{j=1}^{c} O_{i,j} log(\frac{O_{i,j}}{E_{i,j}}) \tag{86}$$

In the case of feature associations the null hypothesis can be defined as assumption that two features are independently from each other. Therefore the number of shared instances are just results of coincidences and does not significantly deviate from the expected common instances.

When dealing with a small number of observations, the probability estimate will be low, which could lead to problems in the exact calculation of the log likelihood test. Therefore schemes have been developed to make the computation of this test more robust[242].

The log likelihood test has been used in many knowledge discovery application, especially in the field of natural language processing. For example to detect collocations in textual resources[243].

[242] S. Evert. *The statistics of word cooccurrences: word pairs and collocations*. Stuttgart, 2005

[243] P. Pecina and P. Schlesinger. Combining association measures for collocation extraction. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 651–658, Morristown, NJ, USA, 2006. Association for Computational Linguistics

Poisson Test The binomial distribution is the base of the Fisher exact test, which is coupled with a high run-time complexity. Therefore it is desirable to be able to replace the binomial distribution with an approximation. If the probabilities are sufficiently small, the binomial disambiguation can be approximated by the Poisson distribution.

Following the approach by Quasthoff and Wolff[244] the Poisson distribution can be adapted to calculate a significance values:

[244] U. Quasthoff and C. Wolff. The poisson collocation measure and its applications. 2002

$$\lambda = |I| p(f_s) p(f_t) \tag{87}$$

$$k = |\{i_k | i_k \in I \cap \langle i_k, f_s \rangle \in E \cap \langle i_k, f_t \rangle \in E\}| \tag{88}$$

$$Poisson = \frac{-log(\sum_{l=k}^{\infty} \frac{1}{l!} \lambda^l e^{-\lambda})}{log(|I|)} \tag{89}$$

For smaller probabilities this can further be reduced to:

$$Poission_{approx} = \frac{\lambda - k\ log(\lambda) + log(k!)}{log(|I|)} \tag{90}$$

Like the log likelihood test the approach using the Poisson distribution has been used by several applications. Both measures of association have also been compared using multiple languages and both perform equally well when compared with other association measures[245].

[245] S. Bordag. A comparison of co-occurrence and similarity measures as simulations of context. In *Proceedings of the 9th international conference on Computational linguistics and intelligent text processing*, CICLing'08, pages 52–63, Berlin, Heidelberg, 2008. Springer-Verlag

Probability Based Functions

Some of the presented statistical test imply that a feature is coupled with a certain probability. How these probabilities are calculated was not covered in detail in the previous section. The probabilities of the features are not only useful within a significance test. The probability of a single feature and more importantly the probability of pairs of features give insights into the properties of the feature associations. Therefore a selection of feature association function based on probability estimates are presented.

At first an estimate for the probability of individual features needs to be established. The number of instances that are connected to a feature is only possible starting point. As with the similarity measure which operate on binary representations, a threshold value can also be applied in this scenario. This instance count can also be seen as feature frequency. Thus the basic probability estimate for a feature can be defined, where I denotes the set of all instance nodes and E is the set of all edges within the input feature graph:

$$p(f_x) = \frac{|\{i_k | i_k \in I \cap \langle i_k, f_x \rangle \in E\}|}{|I|} \tag{91}$$

This basic maximum likelihood estimate does only consider the number of instances connected to a feature in relation to the magnitude of the set of all features. The correct estimation of the probability of features is a central aspect of the language modelling approach to information retrieval. In this field many alternatives to the basic probability estimation method have been proposed. Ponte and Croft have integrated a term frequency factor into the basic formula[246]. They proposed method can be adapted to the input feature graph. The weights of the edges between an instance node and feature node are used instead of the term frequencies:

$$p^{local}(f_x, i_k) = \frac{\text{ſ}_F^w(f_x, i_k)}{\sum_f \text{ſ}_F^w(f, i_k)} \tag{92}$$

$$p(f_x) = \frac{\sum_{i_k} p^{local}(f_x, i_k)}{|\{i_k | i_k \in I \cap \langle i_k, f_x \rangle \in E\}|} \tag{93}$$

As these probability estimations are based on the occurrences of features within a data-set, the quality of the estimates will rise with the amount of data. Therefore the size of the data-set plays an important role in any language model based application, as well as for the final feature association results.

[246] J. M. Ponte and B. W. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998

The presented probability estimates are tailored towards textual features, but the same approaches are still valid for other types of feature with similar properties. The most important requirements is the sparseness so that a single feature is only connected to a relatively small subset of instances. For dense features the probability estimates can be directly derived from the data. More precisely, the features are assumed to be power-law distributed, a property shared by many real-world phenomena.

If the available data-set is too small for reliable probability estimates, the language model can be build using another dedicated data-set. This approach is valid as long the language used within the data-set is similar to the general language usage. Examples for big data-sets to build are the British National corpus[247], the Gigaword corpus[248] and the Google n-gram corpus[249]. There are also corpora available for specific domains, for example the Reuters RCV-1[250] and RCV-2 for newswire articles.

If the data-set being analysed uses a specialised, domain-specific language where there is no large corpus available, there are other methods to improve the quality of the probability estimates. These methods may even be used for big data-sets, when the number of occurrences for specific features is low. Some features may occur only a few times, or might not occur at all. This problem can arise if the data-set is split into a set of training instances and a set of test instances. For example if the application employed a supervised machine learning algorithm such a splitting might be necessary.

The term smoothing is used to summarise a family of algorithms that try to correctly estimate the probability of rare and missing features. The basic form of smoothing is based on the model that all features are connected to a virtual instance node. So even missing features (features without a relation to any of the instance nodes) are connected to this additional instance. Thus the probability estimate becomes:

$$p(f_x) = \frac{|\{i_k | i_k \in I \cap \langle i_k, f_x \rangle \in E\}| + 1}{|I| + 1} \tag{94}$$

This kind of smoothing is also referred to as *add one* smoothing or Laplacian smoothing. This scheme can be further refined by replacing the constant term 1 by an variable and is then called additive smoothing[251].

Although many different approaches to the problem of smoothing the probability values for features has been proposed, only a limited selection can be covered here. The Simple Good-Turing (SGT) method[252] is used as an example for one of such sophisticated smoothing techniques. The SGT smoothing approach is grounded on the information theory. At first the probability of any missing features is estimated by the ratio of feature connected to only a single instance in relation to the magnitude to the set of features F:

$$p_0 = \frac{|f_x | f_x \in F \cap \mathfrak{f}_F^{in}(f_x) = 1|}{|F|} \tag{95}$$

This estimate assumes that the likelihood of a missing feature is the same as the probability estimate for a feature having just a single connection to the set of instance nodes. The probability of the remaining features, two algorithmic approaches are taken. For low frequent features a recursive approach is employed, for features with a higher count of instances a smoothing method is applied.

First the frequency counts for the feature with many instances are interpolated. In most of the non-synthetic data-sets there will be gaps in the histogram of number of features per instance count. Especially for higher instance counts there will not be a single feature representing a frequency bin. For example there might be a number of features connected to 100 instances and a few features are connected to 103 instances and some features are related to 105 instances, but there is not a single feature in the data-set connected to 101, 102 or 104 instance nodes. These gaps are filled by interpolating over the feature instance-count distribution by using the available data. The available data is first evenly smoothed, where t and q

[247] L. Burnard. Reference Guide for the British National Corpus (XML Edition), 2007

[248] D. Graff, J. Kong, K. Chen, and K. Maeda. English gigaword. *Linguistic Data Consortium, Philadelphia*, 2003

[249] `http://ngrams.googlelabs.com/datasets`

[250] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004

[251] C. X. Zhai. Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, 1(1):1–141, 2008

[252] W. Gale and G. Sampson. Good-Turing smoothing without tears. *Journal of Quantitative Linguistics*, 2(3):217–237, 1995

are assumed to be the closest available instance-counts:

$$Z_r = \frac{N_r}{0.5(t-q)} \tag{96}$$

For example the smoothed frequency of 5 features having an instance count of 103 is $\frac{5}{0.5}105 - 100 = \frac{5}{2.5} = 2$. This is done for all available frequencies and the result is plotted as a log-log diagram. Within the log-log chart one can fit a linear regression curve, which is then used a smoothing of the feature frequency counts. The chart in figure 19 is an example of a log-log view of the unsmoothed feature frequencies. The added regression line does not match the distribution of the data.

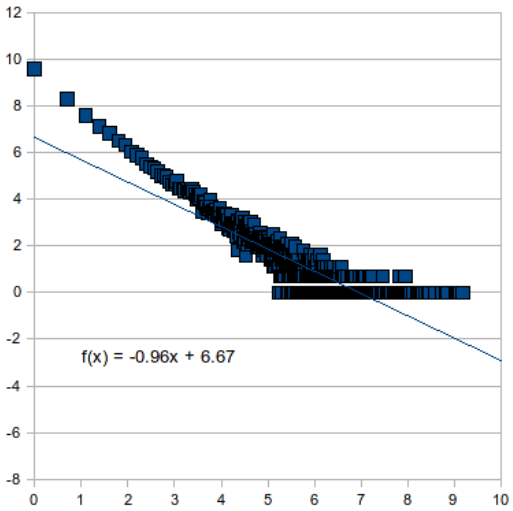

Figure 19: Example for a log-log diagram of a feature count histogram, with textual features generated from the ConceptNet corpus. The x-axis represents the bins of features having a certain instance count. The y-axis are the counts for each bin. Additionally a linear regression curve is fitted into the data. Clearly the curse does not reflect the slope of the frequency distribution.

In figure 19 the same data is smoothed. Now the regression line nicely fits the available data. Using this regression curve as approximation of the frequency distribution, the final smoothing can be calculated.

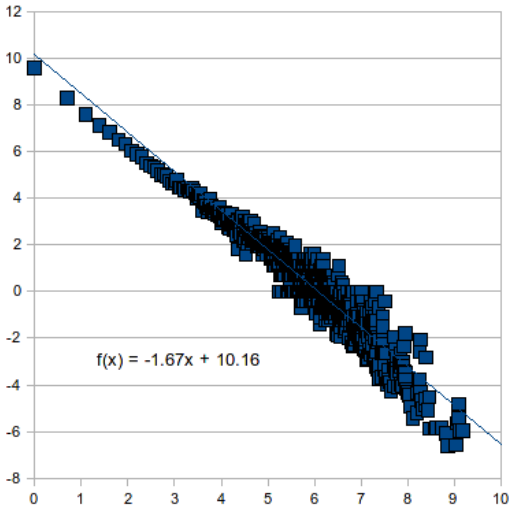

Figure 20: Log-log chart of the smoothed frequencies of the sample data. The frequencies have been evened out by using the index of the neighbouring bins having a count greater than zero. Now the slope regression line and the distribution of the frequency counts are a much closer match. The regression line can now be used to substitute the missing frequency bins.

For higher frequencies the linear regression curve is directly used as approximations. For smaller frequencies a recursive approach is employed that estimates the frequency of features with a certain number of instances by using the estimate of the frequency of features having one more instance

relationship as input. This recursive definition uses the expectation of a frequency count $E(N_n)$ for a given bin probability p_n and for the next higher bin.

$$p_n = p_{n+1} \frac{E(N_{n+1})}{E(N_n)} \tag{97}$$

By using the number of features that are connected to only one instance as expected number ($E(N_1) = |f_x | f_x \in F \cap \mathfrak{f}_F^{in}(f_x) = 1|$), the recursion can be iteratively be resolved. At a certain point the recursive approach is given up and the probability estimates are taken from the regression line. The decision when to swap the strategies can be based on the difference of the two predicted probability. As long as the two methods produce significantly different results, the recursive approach is taken. For all higher frequency counts the regression line based probability estimates are used. Figure 21 gives an overview of the results of the two methods together with the counts for the frequency bins.

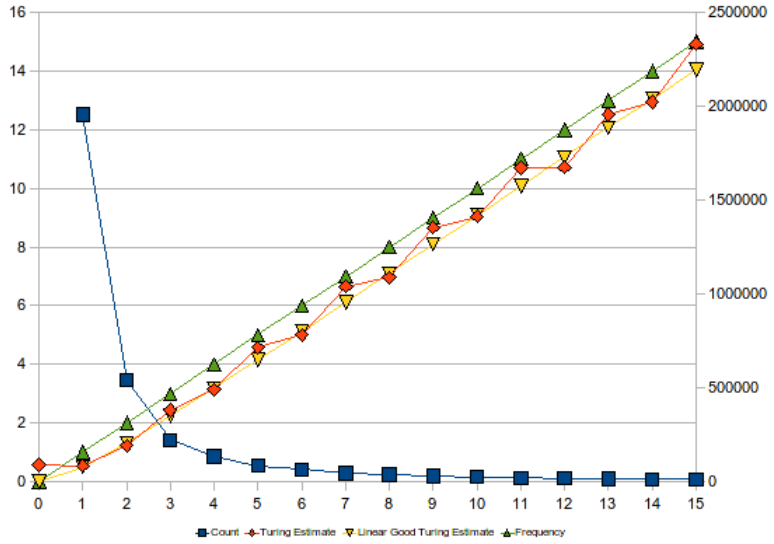

Figure 21: Overview of the results of the Simple Good-Turing smoothing algorithm. The x-axis represents the frequency bins of the frequency-count histogram. The Turing estimate captures the results of the recursive frequency estimation, and the Linear Good-Turing estimate resembles the results from using a linear regression line. Additionally the unsmoothed frequency is given as curve and the frequency counts of the histogram are included using the secondary y-axis. Both smoothing methods underestimate the frequencies, which is expected since they both incorporate the probability of unseen features.

The task of the feature-engineering is to decide whether a smoothing should be applied at all and which smoothing strategy is a sensible choice. The optimal decision largely depends on the actual data-set and the knowledge discovery application itself. Therefore the basic smoothing technique, by simply adding a virtual instance node to the input feature graph, is a good starting point to assessing whether smoothing is beneficial in a given application scenario.

Until now the probability of a feature has been mainly been viewed from the perspective of instances and relations between instance nodes and feature nodes. Depending on the scenario, the probability of a feature can be defined differently. The probability of a feature may be defined in the data-set itself or might be calculated by a pre-processing step. In this case the probability estimate of a feature might be encoded into the weight of a instance-feature relationship. Alternatively an application might assume that all features are equally probable, or the probability is retrieved from an external resource.

For the feature association function the origin of the probability estimate is not relevant. Therefore the scenarios will not be considered in the description of the probability based feature association functions.

Joint Probability The probability of a single feature defines how likely an occurrence of this feature will be. So given a new instance node, there will be an relation to a feature according to its probability.

As long as the features in a data-set are completely independent from each other they will not interact with each other. The joint probability captures the frequencies of two features being related to a single instance node. Thus the joint probability will be relatively high for features with a high correlation. For completely independent features, the joint probability is the product of the individual probabilities:

$$p(f_s, f_t) = p(f_s)p(f_t) \tag{98}$$

Regardless of the correlation of two feature, their joint probability will be bound by the individual probability. Its range is $[0, min(f_s, f_t)]$, with a value of 0 for features that exclude each other. The joint probability is symmetrical, so the sequence of input parameter has no influence on the result.

The joint probability captures the relationship between two feature and is therefore good choice for a feature association function. A downside of this measure is the fact, that two joint probabilities cannot directly be compared, because its range depends on the smaller probability value. To compensate for this, the joint probability can be normalised, which results in the conditional probability.

Conditional Probability The probability of one feature given another feature is called conditional probability. It captures how much the presence of a feature indicates the presence of another. It can be also be seen as a normalisation of the joint probability:

$$p(f_t|f_s) = \frac{p(f_s, f_t)}{f_s} \tag{99}$$

Due to the normalisation on one of the input variables (f_s in this case), the conditional probability is not symmetrical. The sequence of input variables does matter, in the general case: $p(f_t|f_s) \neq p(f_s|f_t)$.

For completely independent features the conditional probability is equal to the probability of the target feature. Thus knowing one feature gives no information about the presence of other features:

$$p(f_t|f_s) = p(f_t) \tag{100}$$

The minimal value of the conditional probability - 0 - is the result of one feature excluding the other. The joint probability reaches its maximum of 1 if the the presence of one feature always yields the other feature. Although this does not implicate that $p(f_s) = p(f_t)$, but it follows that $p(f_t) \geq p(f_s)$.

Because of its fixed output range the joint probability of multiple feature pairs can be compared with each other. Therefore in the general case the joint probability is a better choice as feature association function that the joint probability. Both probabilities can be used in more complex functions to calculate the strength of an association between two features.

Entropy Based Functions

A family of feature association functions is rooted in the information theory. The central element of these algorithms is the concept of entropy. The entropy captures the amount of information and is usually measured in bits[253]. It can be interpreted as uncertainty of an event.

More naturally it can also been seen as minimal number of yes/no answers to needed to answer an information need. One example of such

[253] This depending on the base of the logarithm. Where a base of 2 yields bits, the result of using a base of 10 is called dits and using the natural logarithm yields nats.

information need could be: "Given an instance node, is there a relationship with a specific feature?" The number of answers needed to find out whether this feature is actually related to the instance, depends on the frequency of the feature. If the feature is connected to all instance nodes, the answer is obvious and not a single question is needed - thus the entropy is zero in this case. At the other end of the spectrum - a feature that is not connected to a single instance. The uncertainty is also zero for features, not being related at all.

Intuitively there is a link between the probability of a feature and the entropy. The entropy will be zero for $p(f_x) = 0$ and $p(f_x) = 1$. For all remaining probabilities the entropy can be defined as sum of the two events for the random variable 'feature is related':

1. The feature is related to an instance - $p(f_x)$
2. The feature is not related - $1 - p(f_x)$

Thus the general entropy formula (which is presented with the complete background in many textbooks, for example in "Elements of Information Theory"[254]) can be rewritten for the existence of an instance-feature relationship:

[254] T. M. Cover, J. A. Thomas, and J. Wiley. *Elements of information theory*, volume 1. Wiley Online Library, 1991

$$H(f_x) = (p(f_x)\log_2 p(f_x)) + ((1 - p(f_x))\log_2(1 - p(f_x))) \tag{101}$$

But the information of single feature is not sufficient for the task of finding feature associations, so the entropy alone cannot be used as feature association function. Based on the information theory and the definition of the entropy other measures are more suited and a selection these will be presented in the following paragraphs.

Mutual Information The mutual information captures the amount of information contained in a pair of random variables. Given the entropy of a single feature, the mutual information is the proportion of the uncertainty shared with a second feature. It will be high for features that have a high correlation and low of independent features.

Although there is a certain overlap with the joint probability, the mutual information covers the permutation of all events, not only the joint occurrence. Thus the input of the mutual information is basically the same as for the χ^2 test as given in table 19 and .

	$\neg f_x$	f_x
$\neg f_t$	none	only source
f_t	only target	both

Table 21: Overview of the contingency table used as input for the calculation of the mutual information and other entropy based feature association measures.

Formally the mutual information (MI) for two features can be written as, where $p(\neg f_x)$ denoted the probability of a feature not being related to an instance $(1 - p(f_x))$:

$$\begin{aligned} MI(f_s; f_t) = \; & p(f_s, f_t)\log(\tfrac{p(f_x, f_y)}{p(f_x)p(f_y)}) + \\ & p(\neg f_s, f_t)\log(\tfrac{p(\neg f_x, f_y)}{p(\neg f_x)p(f_y)}) + \\ & p(f_s, \neg f_t)\log(\tfrac{p(f_x, \neg f_y)}{p(f_x)p(\neg f_y)}) + \\ & p(\neg f_s, \neg f_t)\log(\tfrac{p(\neg f_x, \neg f_y)}{p(\neg f_x)p(\neg f_y)}) \end{aligned} \tag{102}$$

The output unit of the mutual information is bits like the entropy itself. A result of zero indicates that the two features are completely independent. The mutual information is symmetrical, thus the result is the same regardless of the ordering of the input variables.

The mutual information has been used in many knowledge discovery applications to study the properties of data sets. Especially in the field of social tagging systems, the mutual information is applied to detect and evaluate the commonness of features[255,256].

Another important aspect of the mutual information is its close relationship to the Kullback-Leibler divergence D_{KL}, which measures the distance

[255] E. Chi and T. Mytkowicz. Understanding the efficiency of social tagging systems using information theory. In *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, pages 81–88. ACM, 2008

[256] B. Markines, C. Cattuto, F. Menczer, D. Benz, A. Hotho, and G. Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *Proceedings of the 18th international conference on World wide web*, pages 641–650. ACM, 2009

of two distributions. In the case of the mutual information these distributions are i) the observed joint distribution of the two features and the ii) the expected distribution if both features were independent:

$$MI(f_s; f_t) = D_{KL}(\overbrace{P(f_s; f_t)}^{Observed\ distribution} \quad || \quad \underbrace{P(f_s)P(f_t)}_{Expected\ distribution}) \tag{103}$$

This approach allows a different view on the mutual information. It can also be seen as distance of true distribution to the expected distribution, if an independence of features is assumed.

If the application requires that the feature association function conforms to the definition of a metric, the mutual information must be further transformed. By combining the individual entropies with the mutual information a measure can be created that satisfies all needed requirements. Formally this new measure, which is referred to as normalised variation of information, is defined as:

$$D(f_s, f_t) = \frac{H(f_s) + H(f_t) - 2MI(f_s; f_t)}{H(f_s) + H(f_t) - MI(f_s; f_t)} \tag{104}$$

This metric produces values in the range of $[0, 1]$, where a value of 1 corresponds to features that are independent from each other.

Pointwise Mutual Information As states in the description of the mutual information it covers the permutation of all possible events of the input variables. In the case of feature associations of low frequent features, the aspect of a non-relationship does not seem to be very intuitive. Statistical speaking the probability of a infrequent feature to be connected to an instance node is far lower than not being connected. The mutual information formula treats all cases equally, thus for many real-world features the counter-intuitive cases will start to dominate the result. Especially for textual resources, the majority of features are just connected to a few instance nodes.

For features that follow a power-law distribution, most of the features will be found in the so called long tail.

In other words, the correlation of two features based on the observation of not occurring is not a reliable indicator for their association strength. Therefore instead of using the complete formula of the mutual information, just the part that deals with the case where both features are tested on existence is used. The remaining measure is called pointwise mutual information (PMI) and defined as:

$$PMI(f_s; f_t) = \log(\frac{p(f_x, f_y)}{p(f_x)p(f_y)}) \tag{105}$$

The pointwise mutual information will be zero for completely independent features. In contrast to the mutual information the PMI may produce negative results for features co-occurring less frequent as expected by chance, for example if two features exclude each other. For correlating features the values will be greater than zero. Like the mutual information, the PMI is symmetric in regard to the input variable sequence order.

The pointwise mutual information has been first used by Church and Hanks[257]. Since then it has been using in many knowledge discovery applications, for example to detect synonym relationships[258], represent word meaning[259] and to predict word re-occurrences[260].

Many modifications to the basic formula of the pointwise mutual information have been proposed. One of these modification is to use only values larger or equal to zero:

$$PMI_{positive}(f_s; f_t) = max(PMI(f_s, f_t), 0) \tag{106}$$

[257] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990

[258] P. D. Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the twelfth european conference on machine learning (ecml-2001)*, pages 491–502, 2001

[259] J. A. Bullinaria and J. P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510, 2007

[260] A. Sarkar, P. H. Garthwaite, and A. De Roeck. A Bayesian mixture model for term re-occurrence and burstiness. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, CONLL '05, pages 48–55, Morristown, NJ, USA, 2005. Association for Computational Linguistics

The resulting measure is called positive pointwise mutual information and has been for example used as lexical association function to measure the semantic relatedness of words[261].

[261] B. Riordan and M. N. Jones. Comparing semantic space models using child-directed speech. *Entropy*, 20:200, 2000

Another modification targets the output range of the pointwise mutual information, which depends on the actual probabilities of the features and is therefore not comparable across associations of different features. Therefore the PMI can be normalised by using its maximum as denominator. To find its upper bound, the $p(f_x, f_y)$ value needs to be maximised. To achieve this, the joint probability is first transformed into a conditional probability, which can be assumed to reach 1. This results in two possible alternatives: $p(f_x, f_y) = p(f_y|f_x)p(f_x) = p(f_x|f_y)p(f_y)$. Therefore there are two different normalisation equations:

$$PMI_{source-normalized}(f_s; f_t) = \frac{PMI(f_s; f_t)}{\log(\frac{1}{p(f_s)})} \tag{107}$$

$$PMI_{target-normalized}(f_s; f_t) = \frac{PMI(f_s; f_t)}{\log(\frac{1}{p(f_t)})} \tag{108}$$

By applying the normalisation on the positive pointwise mutual information, the output range becomes $[0, 1]$, where 0 indicates no or negative correlation and 1 identically distributed features. The normalised variation of the pointwise mutual information has been successfully applied in the field of information retrieval[262,263].

[262] E. Terra and C. L. A. Clarke. Scoring missing terms in information retrieval tasks. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 50–58. ACM, 2004

[263] R. Kern, A. Juffinger, and M. Granitzer. Evaluation of Axiomatic Approaches to Crosslanguage Retrieval. In *Multilingual Information Access Evaluation Vol. I Text Retrieval Experiments*, 2009

Log Odds Although the log odds measure is not related to entropy, there are some similarities with the pointwise mutual information. Both can be derived from the definition of the mutual information and then rearranging parts to match the desired output. While the pointwise mutual information just uses one cells of the input contingency table, the log odds measure uses all four cells. The joint occurrence and the joint non-occurrence are exploited and put in relation to the other two cells. Thus the log odds measure can be formalised as:

$$LogOdds(f_s, f_t) = \log(\overbrace{\frac{p(f_x, f_y)p(\neg f_x, \neg f_y)}{p(\neg f_x, f_y)p(f_x, \neg f_y)}}^{OddsRatio}) \tag{109}$$

Like the mutual information and the unmodified PMI the log odds measure produces results in a range that depends on the input values. Thus the results are hard to interpret and to compare, as indicated by a lively discussion in the area of medical studies[264,265]. The log odds measure has been applied in a wide range of areas and is not limited to computational science.

[264] D. G. Altman, J. J. Deeks, and D. L. Sackett. Odds ratios should be avoided when events are common. *British Medical Journal*, 317(7168):1318, 1998

[265] J. Deeks, M. B. Bracken, J. C. Sinclair, H. T. O. Davies, M. Tavakoli, and I. K. Crombie. When can odds ratios mislead? *British Medical Journal*, 317(7166):1155, 1998

In the field of feature associations the odds ratio or log odds measure has been employed multiple times, for instance to measure the distance in semantic spaces[266].

[266] W. Lowe and S. McDonald. The direct route: Mediated priming in semantic space. In *Proceedings of the 22nd Annual conference of the Cognitive Science Society*, pages 675–680. Citeseer, 2000

Conditional Entropy Like the mutual information the conditional entropy is the result of the combination of two feature distributions. It measures the uncertainty of one feature, when knowing the other feature. So given an instance node with an unknown number of related features, where one of these features is known. The conditional entropy is the number of yes/no questions needed to find out whether a specific feature is also connected to the instance node. Intuitively the uncertainty is highest if the two features are completely independent. As soon as there is a regularity between the distributions of the two features, the information content will decline. The conditional entropy can be defined for the contingency table by

combining the joint probabilities and the conditional probabilities:

$$H(f_t|f_s) = \begin{array}{l} p(f_s, f_t)\log(\frac{1}{p(f_t|f_s)})+ \\ p(\neg f_s, f_t)\log(\frac{1}{p(f_t|\neg f_s)})+ \\ p(f_s, \neg f_t)\log(\frac{1}{p(\neg f_t|f_s)})+ \\ p(\neg f_s, \neg f_t)\log(\frac{1}{p(\neg f_t|\neg f_s)}) \end{array} \quad (110)$$

Although the conditional entropy does not appears to be good indicator for the association strength between two features, due to its properties there are a number of use cases for this measure. One of these use cases is compression. The entropy of a sequence of events is the minimal number of bits required to encode this sequence. Thus it represents the lower bound of what a compression technique can achieve. A simple application of such a compression technique is to throw away all features that carry no additional information. For these features the conditional entropy will be zero for at least a single association.

Redundancy The redundancy is included as one example of a whole family of information theoretic measures that capture the relationship between two features. It is related to the mutual information and is defined as:

$$R(f_s; f_t) = \frac{MI(f_s; f_t)}{H(f_s)H(f_t)} \quad (111)$$

Due to its definition the redundancy can also be seen as normalised variant of the mutual information. The range of the results of the redundancy measure is $[0, min(H(f_s), H(f_t))]$. This property makes the redundancy itself suitable for further normalisation to give a range of $[0, 1]$. Whether this additional normalisation step is conducted depends on the knowledge discovery application and the nature of the features.

As indicated in the introduction to the redundancy, it is only one representative of many proposed alternative modifications to the basic definition of the mutual information and the conditional entropy.

Interaction Information While the presented entropy based measures so far did consider the interaction between a pair of features, some applications might require insights into the relation of the distribution of more than two features. For example the mutual information can be extended for the case of multiple random variables. Instead of describing the extension an already covered measure to the case of three features, a new measure is introduced, the interaction information[267]. This not so well known measure is sometimes also called co-information[268] and boast a number of interesting properties.

The interaction information measure how the mutual information of two features changes in the presence of a third feature. For example two features appears to be completely independent from each other when observed on a global level. But as seen from the perspective of a third feature, regularities in the relationship between the two features begin to arise.

This property can be observed especially in the case of human language, where the semantics of words and their joint usage depends on the topic or domain. Generally speaking feature associations might not occur globally, but only for specific contexts. For the formal definition of the interaction information it is first necessary to define the conditional mutual information:

$$MI(f_s; f_t|f_c) = H(f_s|f_c) + H(f_t|f_c) - H(f_s, f_t|f_c) \quad (112)$$

The conditional mutual information is the sum of the two conditional entropy values for the source and target feature given a context feature minus

[267] W. J. McGill. Multivariate information transmission. *Psychometrika*, 19:97–116, 1954

[268] A. J. Bell. The co-information lattice. In *Proceedings of the Fifth International Workshop on Independent Component Analysis and Blind Signal Separation: ICA 2003*, 2003

the conditional entropy of the joint distribution for the given context. This conditional mutual information is essentially the uncertainty that remains from the joint entropy of the source and target feature when knowing the context feature. In other words, the conditional mutual information will be low if the context "causes" the two features to have something in common and high if the regularity between the source and target feature is independent from the actual context feature. The interaction information II can now be defined as:

$$II(f_s; f_t; f_c) = MI(f_s; f_t|f_c) - MI(f_s; f_t) \tag{113}$$

It takes the conditional mutual information and subtracts the mutual information of the source and target feature. This way the result of the interaction information can either be negative or positive. If the context feature has no impact on the relationship between the two features (f_s and f_t), the interaction information will be zero.

For the negative case of the interaction information, the context feature can be seen as causing the correlation of the distribution of the source and target feature. Its (negative) maximum is reached if the context completely covers the commonness of the two features with $-min(MI(f_s; f_t), MI(f_s; f_c), MI(f_t; f_c))$.

The positive results for the interaction information can best understood if assumed that the mutual information of f_s and f_t is as if the two features are independent. The presence of the context feature causes the two feature to be inversely related, so that the context correlates with either the source or the target feature.

Because of the nature of the interaction information result range, the application is advised to handle both cases differently. A possible approach could be to focus on either the positive or the negative result of the interaction information. For example, an application might choose to specialise on detecting hidden correlations between features and therefore just use negative results. In this case the feature association function might be defined as:

$$II_{negative}(f_s, f_t) = \sum_{f_c \in F} max(-II(f_s; f_t; f_c), 0) \tag{114}$$

The result is the sum of all latent correlations between a source and a target feature.

Alternatively the positive results of the interaction information can be exploited to detect the correct granularity of contexts. Starting with a single context feature, it is subdivided into more refined contextual features as long as the interaction information produces high results for a certain number of pairs of features. The usefulness of such a approach largely depends on the data-set and the actual threshold employed as splitting criterion.

Sliding Window Based Functions

The final presented approach is not a feature association function per-se, but a building block for integrating algorithms into the feature association framework. It also serves as an example, how a data-set that is annotated with a rich set of meta-data can be used to build feature associations tailored towards specific application scenarios.

In this case it is assumed, that the relation between an instance node and a feature node within the input feature graph is annotated with a meta-data that encodes the occurrence position of a feature. For textual resources this will simply be the position of a word within the text. That way each instance node resembles a textual document, a paragraph or a sentence. Words are mapped as features and the edges connecting the features to the

instances are the occurrences.

By adding the absolute position of an occurrence as meta-data it is possible to conduct a keyword in context (KWIC) analysis. Given a specific feature and instance all co-occurring features can be sorted according to their distance. For example given the word "instrument" and its position, all other words within the same sentence can be arranged around this target word. By using five example sentences from the ConceptNet corpus the KWIC analysis looks like:

```
                A stringed instrument can be tuned.
Learning to play a stringed instrument can be part of a school curriculum.
         A barometer is an instrument used to measure air pressure.
      A woodwind is a wind instrument .
                   A wind instrument can produce music.
```

The feature association framework exposes the position of a feature as factor and therefore is it possible for any feature association function to calculate the distance of two occurrences. Usually the distance is used to filter out features too far away from an occurrence. For example only directly adjoining features are selected as candidates for further feature associations. This could be interpreted as an additional pruning strategy, because the number of possible candidates for feature associations is reduced. This is especially effective if the average number of features per instance is high.

This approach is named sliding window because the selection of which occurrences to select can be visualised as an window of words around the position, which is moved by one position for each word. The size of the window is determined by the application, usually just a few words left and right of the target word are used. For example Freitag et al. conducted experiments with various window sizes ranging from 1 to 4[269], others used window sizes up to 32 words[270].

The window does not need to be symmetric, so for example only preceding words might be selected for further processing. The distance can also be exploited, for example as additional weight in the calculation of the feature association strength.

The sliding window approach can also be used as criteria to dynamically generate a distribution needed for all probability based feature association function. For example given a specific source feature the mutual information with all features is calculated that occur in a predefined vicinity to the source feature occurrences. But the sliding window method is not restricted to a single type of feature association function. For example it can also be combined with the cosine similarity or any distance measure.

[269] D. Freitag, M. Blume, J. Byrnes, E. Chow, S. Kapadia, R. Rohwer, and Z. Wang. New experiments in distributional representations of synonymy. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 25–32. Association for Computational Linguistics, 2005

[270] E. Terra and C. L. A. Clarke. Frequency estimates for statistical word similarity measures. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 165–172, Morristown, NJ, USA, 2003. Association for Computational Linguistics

Distributed Environment

Building feature associations is a intensive operation when applied on big data sets. A single execution unit will usually not sufficient for this task. This section will present how the feature association framework can be integrated into existing cluster management solutions.

The Early Days

When dealing with huge data sets and intensive computations the hardware as well as the software need to be capable to cope with the amount of data. The requirements on the employed algorithms are especially high, they need to be efficient and provide an optimal throughput.

Since the invention of the first computers the computational power of the hardware did steadily increase. The famous Moore's law states that the number of transistors per processing unit doubles every two years. Therefore the traditional way to handle huge data sets was to use build bigger machines. To improve the performance even further multiple machines were wired together into clusters. Although these clusters theoretically multiply the available computational resources, the demands on the software to distribute and synchronise the workload increase with every added machine.

In recent years companies and universities started to use commodity hardware instead of expensive dedicated super-computers. This trend has started in the last decade of the 20th century as answer to the challenge of providing retrieval applications that allow users to search the entire web. As the commodity hardware does not provide the computational power comparable with modern super-computers, the number of necessary units increases further. With this increase of hardware the probability of hardware defects and malfunctions also gets higher.

Therefore the software which is executed on a cluster of cheap commodity hardware needs to be developed to expect single nodes to fail. Even worse, some units will not completely malfunction, but will produce the results slower than the rest of the nodes. All these cases need to be detected and appropriate counter-measures need to be deployed. This not only requires a sophisticated cluster management solution, but the algorithms themselves need to be designed in such a way that they can be applied in such a distributed environment.

In the following section a framework for executing algorithms on a cluster of commodity hardware is presented. Thereafter it will be demonstrated how the feature association framework can be configured to be applied in such a scenario.

MapReduce

The most common approach to manage data sets and execute algorithms on a cluster of commodity hardware is called MapReduce. MapReduce[271] is a generic framework developed to efficiently execute algorithms on huge data sets in a distributed environment. It was originally developed to process and store search indices of massive amounts of web pages crawled on the Internet. Starting with this implementation it since became synonymous for a family of approaches tailored towards the efficient dispatching of a workload onto a set of computers. An important aspect of MapReduce and similar methods is that it is not a single algorithm, but a number of components closely working together.

[271] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, pages 1–13, 2008

Network Layer The base of of all components of the MapReduce framework is the layer that manages the communication between the individual nodes of the cluster. Typically this layer employs network communication protocols like UDP/IP and routing techniques like multi-casting. It provides mechanism to detect failures of individual nodes by regular update messages, commonly referred to as heartbeat.

Distributed File System The next higher abstract layer is a so called distributed file system. The basic idea behind such a file system it so enable shared access to common data for each participating unit in the cluster.

A common access pattern to this file system is:

1. The data set is written and distributed on the file system by a dedicated node
2. All nodes consume parts of the data set and execute their respective tasks
3. The output of the distributed calculations are written into the file system by each node
4. The final accumulated result can then be read out of the distributed file system

The distributed file system splits the data into chunks, which are stored on different nodes as the data will typically too big to be stored on a single node. To improve the reliability of the system, chunks are not stores on a single node, but they are redundantly stored on multiple machines. The distributed file system is required to anticipate the access pattern to the data to optimise the throughput. Therefore the chunks should be stored on exactly these nodes, that require that part of the data. Although this is hardly possible for generic applications, still the distributed file system is required to be aware of the network topology. For example, to decrease the risk of data loss, chunks should not only be stored on separate machines, but also on separate racks. In the event of an power failure that affects a part of the cluster, this storage technique increases the chance that the complete data is still completely available.

In an conventional file system, the data can be randomly accessed. For example an algorithm might choose to read out the middle part of any file stored within the file system. The access order has performance implications on a local file system. Algorithms should be specifically designed in respect to the optimal sequence of operations. In a distributed scenario the performance implications are more pronounced. If an algorithm chooses to read a block of data, which is not available locally, it must be retrieved over the network[272]. Therefore the random access to the data is usually not possible within a distributed file system. The data is managed by a dedicated algorithm and the individual nodes process the data in a streaming manner. Therefore the algorithm need to be adapted to consume the input data in a sequence, as well as writing their output in a fixed order.

In the case of the MapReduce framework, the Google File System[273] serves as implementation of the distributed file system component.

[272] The distributed file system has to look up the list of nodes, which store the requested block. Then the best node has to be selected, based on the closeness and the load of the nodes. Then the block has to be transferred to the requesting node.

[273] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003

Data Set Split Before the actual execution of the distributed processing can start, the whole data-set is first split into a sequence of parts. These parts are then assigned to individual execution units. To allow an optimal execution strategy, the assignment should honour the topology of the network.

The size of the individual parts depends on the actual data set and the applied algorithms. If the size is chosen too small, the communication overhead will dominate over the actual processing in terms of execution

time. At the other end of the scale, the upper bound of the part size is the size of the data set divided by the number of available execution units. Any larger part size will cause some nodes to remain idle and the cluster will not be fully utilised. The optimal splitting strategy also takes the size of the chunks within the distributed file system in consideration.

For many data sets the splits cannot be done at arbitrary positions. Usually a data set consists of records and the splitting algorithm has to separate the data set at the boundaries between two records. Thus an adaption of the MapReduce framework for specific application scenarios is necessary at this level.

Map Phase After the data set has been partitioned into subsets, the individual parts are assigned to workers. These workers are started on all participating nodes and execute the map function. This MapReduce framework defines the interface of this function, the input parameter and the output. The input data-structure of the map function is a tuple of an identifier and a payload. In the case of a web search engine, the identifier might be the URL of a web page and the payload is the HTML content.

The task of the map function is to analyse the input data and to transform it into a series of key, value pairs. The MapReduce framework does not dictate the actual type or semantics of these pairs. For building an inverted index the keys are the words, that occur in the text and the values are weights.

An example implementation of the map function is given in algorithm 7. In this example the generated keys are words contained in a Web page and the values are the number of occurrences within the page.

Algorithm 7 Pseudo-code of the map function as defined by the MapReduce framework. In this example it is assumed that the map function tokenises a Web page and reports all occurring words.

```
procedure MAP(url, html)
    words ← tokenize(html)              ▷ Split the HTML into words
    for all word ∈ words do
                                        ▷ Count the number of occurrences
        tf ← termFreq(word, html)

        YIELD(word, tf)                 ▷ Report each word
    end for
end procedure
```

Collecting the Map Results While the execution of the map functions the emitted key, value pairs are locally collected. They are stored on the same node as the map function is executed. For the storage a dedicated data-structure is needed, which is tailored towards two use cases:

1. Fast writing of the result from the map functions

2. Efficient retrieval of all entries for a given key

Additionally the cluster management collects all keys. In the exemplary scenario of a web search engine, all occurring words are centrally collected in a dictionary. Based on the keys, the work-load for the next phase is generated.

Reduce Phase Like the data-set is split into packets assigned to the map workers, the keys are processed and assigned to the reduce workers. These workers are executed in parallel on multiple machines and invoke the reduce function.

Before a reduction function is called, the necessary data needs to be collected. For a given key all values are combined into one data-structure. The collect values are then passed as input to the reduce function by the framework.

$$Input_{reduce} = \langle key, [value_1, value_2, ..., value_n] \rangle \tag{115}$$

The MapReduce framework defines the reduce function as a transformation of the input tuple to an tuple that consists of the key and a single value. This single value then represents the final result for a given key.

In algorithm 8 an example for a reduce function is given. This example is assumed to work in conjunction with the previously given map example. All values for a single word are aggregated and then reported to the framework. Thus the final result of the exemplary MapReduce application is a dictionary of words with the document frequency and total number of occurrences.

Algorithm 8 Example of a simple implementation of a reduce function. The function is invoked for each word and is provided with the collected output of the map phase. Finally the result is reported to the framework, in this case a tuple of the document frequency and the total term count.

```
procedure REDUCE(key, values)
    docCount ← 0                          ▷ Initialise the aggregation variables
    termCount ← 0

    for all tf ∈ values do
        docCount = docCount + 1           ▷ Increment the counters
        termCount = termCount + tf
    end for

    result ← {docCount, termCount}
    YIELD(result)                         ▷ Report the result
end procedure
```

Collect the Output The result of the reduce function is stored within the distributed file-system. Depending on the further work-flow steps of the application, the result is read out and centrally stored. This step is necessary for all applications that need fast random access to the result data, because the distributed file-system only offers streaming access to the data accompanied with high latencies.

Execution Sequence All phases of the MapReduce framework are executed in a fixed sequence. In figure 22 an exemplary configuration of a MapReduce task is depicted. For the actual execution of the distributed execute additional components are necessary, which are omitted in the figure. These components are responsible to dispatch the workload, monitor the health status of the participating nodes and other housekeeping tasks.

Summary of Adaptations To sum up, there are a couple of necessary steps to adapt an algorithm for the execution within the MapReduce framework. If an algorithm or data set cannot be adapted to conform to the

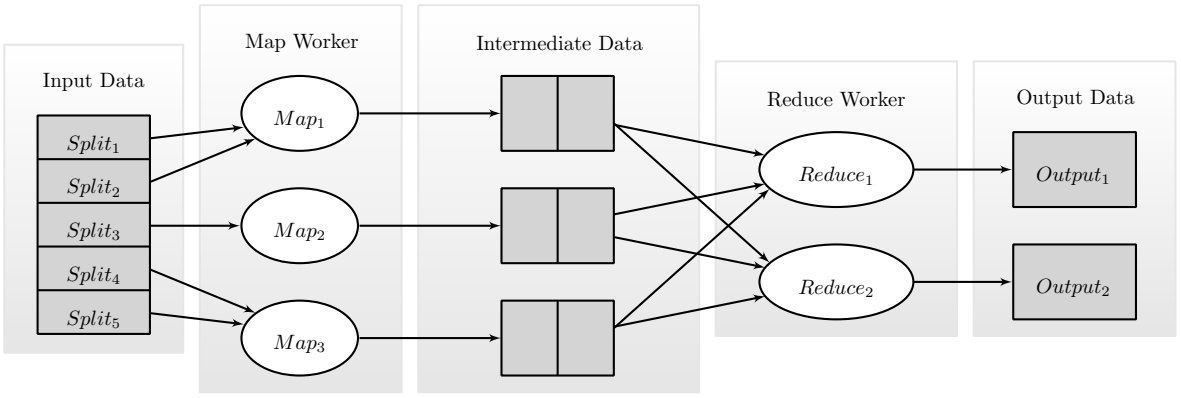

Figure 22: Example of a configuration of the MapReduce framework. First the data set is partitioned into 5 parts. Each part is then assigned to one of three map workers, which are executed in parallel. The output of the map functions is locally stored and then retrieved by both reduce workers. The result is finally stored in the distributed file system.

requirements of the framework, it cannot be executed within the MapReduce framework at all. Only if the algorithm is suitable to be transformed to match the structure and semantics of the framework, the optimal performance can be achieved. Next the requirements will be presented which are necessary to transform an application or algorithm to make it compatible with the MapReduce framework.

1. At first the data set needs to be prepared to be managed and accessed in a distributed manner. Therefore the data has to be partitioned into segments, which can be processed independently from each other. If this requirement cannot be fulfilled and the complete data-set has to be available to all participating nodes at all times, the performance will severely drop due to the network overhead[274]. The size of the partitions and the boundaries of the split are to be defined beforehand and have an influence on the overall throughput of the calculations. Each of these splits contain a number of entries, which consist of an identifier and a payload. The application is responsible on how these data is encoded and organised.

[274] Based on the assumption that the data-set is too large to be locally stored on all nodes of the cluster.

2. The framework dispatches the workload among the nodes and feeds the data to the map workers, which represent the second required adaptation step. An implementation of the map function must be supplied to the MapReduce framework. This implementation has to fulfil the interface requirement of the map function. Additionally the function should not rely on any external data not supplied by the framework as this will usually deter the performance[275]. This requirement arises from the distributed execution on commodity hardware, where defects and hardware faults are likely. Therefore the map function should be work exactly the same way, even when invoked multiple times with the same input variables.

[275] Technically speaking the implementation of the map function should be idempotent and thus should be state-free and have no side effects.

3. The output of the map function can be freely chosen by the application, as long as there as it consists of a key and an value. For the management of the keys the application has to provide a comparator function to the framework. This comparator is then invoked by the framework to find all matching results when collecting the input for the reduce function. In many scenarios the keys will simply be strings (URLs, words, names, ...) or other simple data types which are automatically handled by the framework, which makes the custom comparator function only necessary for more complex keys.

4. The collected output of the map function is collected by exchanging data with all participating nodes of the cluster. For a single key all values are packed into a single data-structure and passed to the reduce function. The framework defines the interface and semantics of this function and

the application has to provide an implementation. As with the map function, the same restrictions on the properties on the reduce function apply, namely it should be idempotent to allow an efficient execution in a distributed execution environment.

5. The final task of the adaptation of an application scenario onto the MapReduce framework is related to the storage of the final result. Each reduce task is executed on a single node within the cluster and its result is stored in the distributed file system. If the MapReduce calculations are only an intermediate step, the result data may remain in the distributed storage. In many scenarios, the output of the reduce workers needs to be collected and then stored in a centralised data-structure. Therefore the application has to provide a functionality to transform the result of the distributed calculations into the desired output format.

The MapReduce is a simple, yet powerful paradigm to perform intensive computations on large amount of data in a distributed manner. One of its main advantages is the ability to rely on cheap commodity hardware instead of dedicated expensive super-computers. But to execute an algorithm on the MapReduce framework, it has first to be adapted to the suit the requirements. Therefore it is only possible for a limited set of application scenarios and data-sets to be effectively applied in such a execution environment.

Relationship to MapReduce

The feature association framework is an approach to analyse and compute the relationships of features within large data-sets. Even when sophisticated pruning strategies and efficient data-structures are employed, the necessary computational resources remain high because of the complex nature of feature associations[276]. The MapReduce framework provides the functionality to execute an algorithm in a distributed manner and is able to handle large data-sets. In order to use the MapReduce framework a number of requirements have to be fulfilled. The following paragraphs cover the main steps to adapt the feature association framework to the the MapReduce execution paradigm.

[276] Even in the basic case, the number of possible feature associations is $\mathcal{O}(n^2)$ for a given set of n features.

The first phases of the feature association framework are responsible to analyse the data-set and to gather global statistics, to apply pruning strategies and to pre-calculate factors for the feature association functions. These operations are typically of low computational complexity are not expected to profit much from being executed in a distributed manner. Therefore the first steps of the feature association calculations can be executed independently from the MapReduce execution framework.

The initial adaptation for the distributed executing is the the partitioning of the input feature graph. Therefore it is necessary to generate sub-graphs that consists of only a small number of nodes. The smallest collection of nodes is a single instance node together will all directly connected feature nodes. Depending on the number of features and the density of the connections, multiple of such sub-graph can be combined to for a single input split.

All input splits are then stored in the distributed file system. In the map function the functionality of the collect phase is invoked. The input of the map function is a single sub-graph, consisting of an instance node and its features. The MapReduce framework requires the output to be pairs of keys and values.

The map function is triggered once for each sub-graph, thus the total number of map function invocation is equal to the number of instances within the data-set. For a instance node, the map function iterates over all connected feature nodes. Feature nodes, which are marked as target nodes are used as keys of the output data-structure. The value for the

target features is a list of all feature nodes marked as source within the current sub-graph. Thus records that consist of a single target feature with all locally co-occurring source features are passed to the MapReduce framework. These records are stored within the distributed file system.

The MapReduce framework collects all records for each target feature and executes the reduce worker. The information passed to a reduce function is a tuple of a target feature together with all locally co-occurring source features for all sub-graphs. Now the association phase of the feature association framework is initiated. Depending on the actual configuration, the information is analysed and combined to generate a list of feature associations. Within a single invocation of the reduce function all feature associations for a given target feature are produced. Thus the reduce function needs to be triggered as often as there are feature nodes marked as target within the input data-set.

The result of the reduce functions is stored in the distributed file system. In order to use the feature associations in in further processing steps, the information has be extracted and written into a central storage. Once the result is read out of the distributed file system, the feature association calculations are finished. Figure 23 gives an overview of the feature association computations when executed within the MapReduce framework.

Alternatives to MapReduce

The MapReduce framework is the most prominent representative of an implementation that manages the distributed execution of algorithms. Many alternatives have been proposed and implemented in recent years, both as commercial solutions as well as open-source frameworks.

Out of the available open-source implementation, the Hadoop project[277] has drawn a lot of attention, especially from the research community. Similar to the MapReduce framework itself, the Hadoop project initially started as part of an system to build a web-scale search infrastructure, called Nutch[278]. Since then it has been steadily improved and has successfully been used in many projects.

[277] `http://hadoop.apache.org/`

[278] `http://nutch.apache.org/`

Its basic operations are very similar to the MapReduce framework. Similar to the Google implementation, the Hadoop project provides facilities to organise files within the cluster. It is called Hadoop Distributed File System (HDFS) and is responsible to store and manage the data within a network of computers. The Hadoop projects is accompanied by a increasingly large set of enhancements and additions. One example of an application build upon Hadoop is the Mahout library[279]. It provides implementation of many machine learning algorithms, which are adapted for the execution within a Hadoop cluster.

[279] `http://mahout.apache.org/`

Besides open-source efforts there are a number of commercially driven projects to develop frameworks similar to MapReduce. The Dryad project[280] is one example of such a framework. Like the other frameworks it uses its own distributed file system, called Cosmos. But unlike the other two presented frameworks, Dryad tries to be more flexible and is not limited to the map and reduce functions. It allows a wider range of distributed functions. As the company that develops Dryad peruses commercial goals, the Dryad framework is only available for a single operating system, which is not commonly used for intensive computations and management of large data-sets.

[280] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59–72, 2007

Besides the mentioned frameworks there exist many other approaches for a distributed execution of application. Most of them are tailored towards business and database applications, with Java EE[281] being the most prominent example.

[281] `http://www.oracle.com/technetwork/java/javaee/overview/index.html`

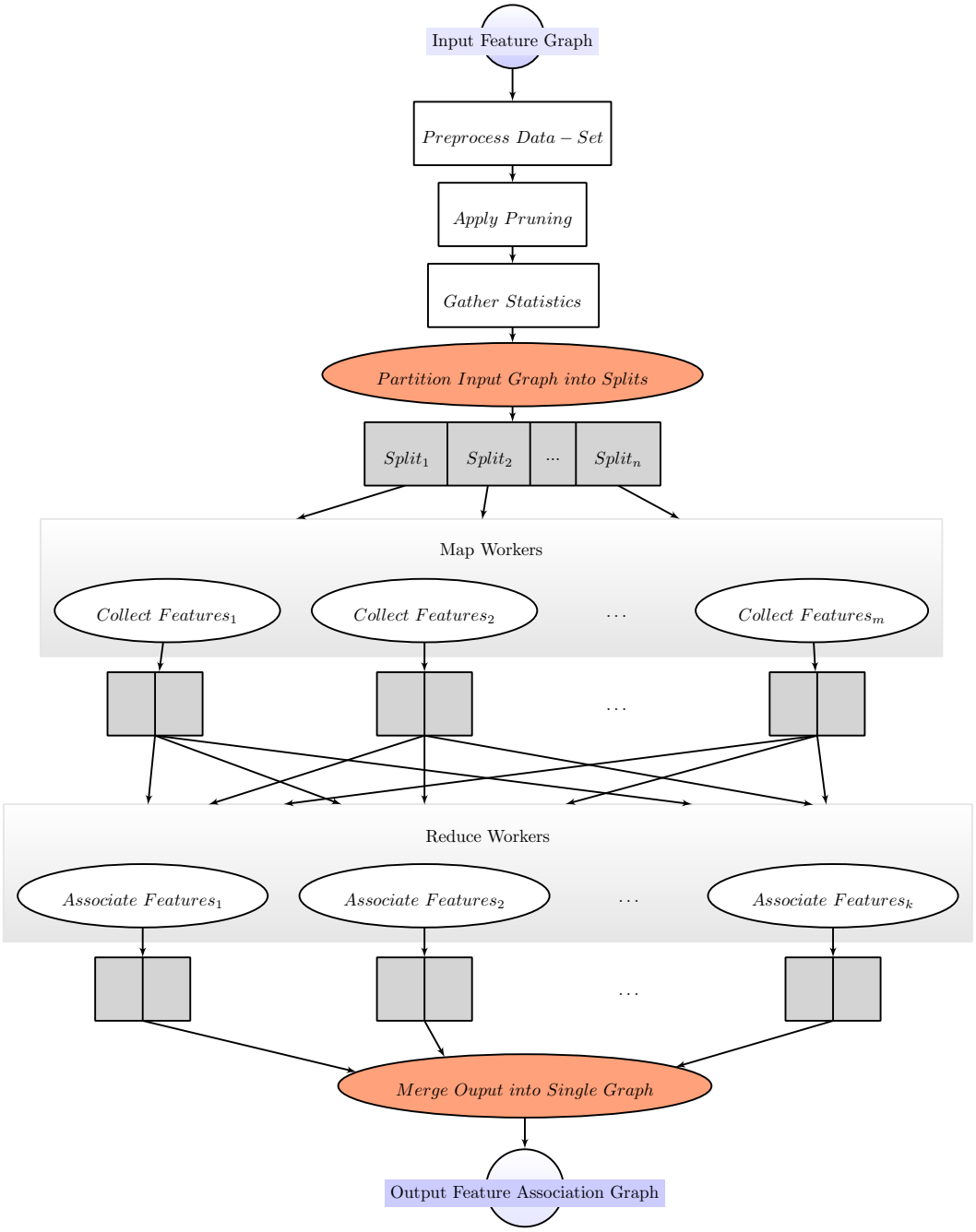

Figure 23: Overview of the sequence of operations if the feature association framework is adapted for the MapReduce paradigm.

Feature Association Retrieval

COMPUTING THE ASSOCIATION BETWEEN FEATURES IS THE FIRST PART OF THE FEATURE ASSOCIATION FRAMEWORK. BASED ON THE FEATURE ASSOCIATION CALCULATIONS THE OUTPUT CAN BE ANALYSED AND FURTHER PROCESSED. THEREFORE THE FRAMEWORK HAS TO PROVIDE FACILITIES TO EFFICIENTLY RETRIEVE AND COLLECT THE NECESSARY INFORMATION.

Overview of the Association Retrieval

The result of the feature association calculations is the output feature association graph. This network[282] contains all relevant relations between the features as determined by the feature association function. This function decides which pairs of features should be associated and can also generate a weight. Depending on the actual implementation of the feature association function, this weight may represent different properties of the relationship between features. In the most common cases the weight will represent the strength of an association of an source feature with a target feature.

[282] Whether the output graph conforms to the formal definition of a network depends on the actual weights of the relationships between the features.

The output graph is expected contain about as many nodes as there features in the input data set. Due to the pruning strategies and the actual implementation of the calculations this number may vary[283]. As the association phase may introduce new synthetic target features, the number of nodes in the output network may exceed the size of the input graph. The relationships between the features are directed and typically weighted, but there are no more than one edge between each pair of feature nodes[284]. Additionally the edges, as well as the nodes, may carry meta-data. There are no further constraints imposed on the properties of the output graph.

[283] It will be lower for more aggressive pruning strategies.

[284] The output feature association graph is not a multi-graph.

The output feature association graph is stored in a data-structure, which has been developed to allow incremental updates. During the execution of the algorithm, the results are iteratively added to the final output graph. Once all feature association have been processed this data-structure may be re-organised and transformed into another representation. This transformation step might be necessary for the forthcoming operations. After all associations have been calculated the output graph is not modified anymore. Instead it will be read out and traversed. The actual retrieval operations depend on the needed information and are presented in the next section.

The feature association framework should provide facilities to traverse the output feature graph. These operations should not only provide the necessary functionality, but they must be developed to provide a satisfying performance. For some operations the retrieval process is as complex as building the association network in the first place.

Use Cases

One of the main motivations to build a generic feature association framework is the advantage of re-using use the same set of technologies and algorithms in many knowledge discovery applications. The requirements imposed by these applications on the retrieval step of the feature associations are diverse. Some applications just require a minimal set of operations, while other application scenarios may only serviced by employing sophisticated algorithms. For both cases the configuration effort should be minimised.

The different retrieval operations of the feature association framework are described in the following section. These operations can roughly be categorised into three classes:

1. The basic operations produce results while traversing over all nodes and

edges of the graph. The result of these operations are for example global statistics and properties of the graph.

2. The second set of operations take a single source feature as starting point. Starting from this point within the graph, the connected feature nodes are traversed while accumulating the necessary information.

3. The final class of operations gather information for a set of feature nodes. The traversal of the graph in this case usually requires sophisticated approaches to avoid excessive processing.

Additionally the application may impose constraints on how the graph is to be traversed. Some of these constraints are presented in conjunction with the main use-cases.

Top Associations The first example of an retrieval function that operates on the whole output feature association graph produces a ranked list of edges. The ranking is conducted by comparing the weights of the edges. Thus the first entry in this list will be the connection with the highest weight in the graph.

Due to its close dependency on the weights the output of this traversal operation is tightly coupled with the feature association function used to generate the weights. When the weight is an representation of the association strength between two feature, the result will list the feature relationships with the strongest associations bounds.

Assuming that the feature association function implements a measure of correlation, the result of the top association traversal operation will contain pairs of highly correlating features. This list can then be used as base to eliminate redundant features. Thus in this configuration the feature association framework can be employed as a means of feature selection.

Global Statistics The retrieval traversal operation is not limited to be based on the weight of the edges. Other global properties of the feature output graph can be computed as well. Other forms of graph analysis are also applicable on the output of the feature association calculations.

The detection of cliques in one example for the analysis of the structure of a graph. To be able to conducted such an analysis, the directed graph needs to be transformed into an undirected one, which can be done as long as the weights are symmetric[285]. Each identified clique then represent a closely related group of features.

Other forms of cluster analysis can also be employed to uncover latent structure within the network of relationships between the features. The spectral clustering approach[286] is well suited for this purpose. These methods transform a graph into its spectrum and tries to detect regularities within the spectral representation. As soon as coherent groups of features have been identified, each group can be replaced by a single synthetic feature. That way the number of feature that encode the instance can be reduced. Other methods of dimensionality reduction can be applied on the output feature association graph as well.

Another family of algorithms do not aim at reducing the number of features, but to aggregate information about single features by exploiting the graph structure. One of the best known of such approaches is the PageRank algorithm[287]. In order to utilise this algorithm, the weights but be computed in such a way that the adjacency matrix of the feature association graph conform to the definition of a Markov matrix. As soon as the prerequisites are fulfilled, the PageRank algorithm can be applied on the network of feature relationships. The output of this analysis is an ordered list of features, ranked according to their relative importance[288].

[285] The feature association function does not depend on the sequence of inputs.

[286] U. V. Luxburg. A Tutorial on Spectral Clustering. Technical Report March, MaxâĂŞPlanckâĂŞInstitut für biologische Kybernetik, 2007

[287] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998

[288] The notion of relative importance depends on the application and the way the feature association function is implemented.

The HITS algorithm[289] is similar to the PageRank approach in regard to its basic aim. Instead of arranging the features in a single sequence, the HITS algorithm generates two lists. These two lists represent the prototypical roles of hubs and the authorities. Features identified as hubs tend to connect and link adjacent features, whereas authorities typically are characterised by many in-links and few out-links.

The affinity propagation algorithm[290] is employs similar techniques as HITS, but its final goal is to identify clusters within a graph structure. Therefore it can be seen as connection between the two presented families of methods to extract valuable information out of a graph.

[289] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999

[290] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science (New York, N.Y.)*, 315(5814):972–6, Feb. 2007

Top Associated Features The retrieval of the list of associated features for a single feature is one of the most important use-cases. Therefore this operation needs to be very efficient as it is expected to be invoked frequently.

As a feature might potentially be associated with many or even all other features, this use-case can be further refined. Some applications might only be interested in the list of the top associations. Thus the retrieval function has to gather all associations, order them according to the sequence of their weights and then pick out the associations with the highest weight. For example a visualisation of the features might choose to present only the ten most strongly connected target features for a single source feature.

If a distance measure is employed as feature association function, the weights needs to be sorted in increasing order. The result list will then contain the nearest features for a given input feature. The top association features retrieval function can therefore be used to conduct a k-nearest neighbour search. This approach is popular choice for classification tasks, where each entry should be assigned to one of a set of predefined classes[291].

Another refinement of the main use-case is the introduction of a threshold instead of an upper bound on the list size. In this case all associated features are included in the result list, where is weight is below a threshold[292].

The top associated retrieval function is not only invoked directly by the application, but it is also employed by other, more complex, retrieval functions. This underscores its importance and therefore its critical for any implementation of this function should be as efficient as possible, both in terms of performance as well as memory consumption.

[291] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991

[292] Or above a threshold, if the feature association function is a distance measure.

Collect Metadata Based on the results of the *top associated features* retrieval function, users might be interested why two features are strongly connected. For example when the feature association framework is employed as a means to generate automatic recommendations. In this case additional information why certain features are recommended could be helpful for the users, to gain a better understanding of the results.

This is the motivation for the *collect metadata* retrieval function. Starting with a pair of directly connected features, additional information of the association between these two can be retrieved. The wealth of the available information depends on the configuration and customisation of the feature association framework. In the aforementioned case, the list of instances that contributed the most to the association strength would fulfil the requirements.

In the output feature association graph, nodes as well es edged can be annotated with meta-data. This retrieval function can be used to expose the information to the knowledge discovery application. It can either be used as enhancement in an visualisation of the data, or as additional input to succeeding processing. The meta-data itself can then be used as features for an machine learning algorithm.

Like the *top associated features* function, the *collect meta-data* function is invoked by more sophisticated traversal algorithms. Therefore its performance will have implications on other retrieval functions as well.

Spreading Activation There are several approaches and algorithms to traverse and search graph structures. One of the most popular of these methods is the breadth first search. Starting with a given node this algorithm iteratively visits neighbouring nodes. At each step there is a set of currently active nodes starting with the single root node. The directly connected neighbour nodes are the active nodes of the next step.

The are two basic modes of operation for this algorithm. The first is to find a specific target node and the second mode is to explore the vicinity of the start node. If this algorithm is employed to search for a specific target node, the iteration is stopped as soon this node has been reached. Without a predefined target node, another termination criteria need to be specified.

The breadth first search serves as technical base for the spreading activation retrieval function. The spreading activation approach has been envisioned by computer scientists as an abstraction to model the information flow within the human brain. Neurons interact with each other via activation signals and as such a network of neurons can store and recall memories[293].

In recent times, the spreading activation has been applied in multiple knowledge discovery applications. It has proven popular to build applications to find relevant information[294] and to combine shallow text data with semantic networks[295].

The basic mode of operation of the spreading activation approach is similar to the breadth first search. Starting with a root node the graph is traversed the same way as in the breadth first search. While doing so, the spreading activation not only records the visited node, but keeps track of a weight for each visited node. The weight of a node is then propagated to all connected neighbour nodes.

A function can be defined to determine the activation weight of a neighbour node. The input for the activation function is a source and a target node, as well as the weight of the source node. For example such a function could compute the propagation weight by multiplying the weight of the source node by a normalisation factor. This normalisation factor is the proportion of the weight of the edge that connects the two nodes by the sum of all weights of the out-links of the source node:

$$f_{out}(n_i, n_j, w_i) = w_i * \frac{weight(e_{i,j})}{\sum_k weight(e_{i,k})} \tag{116}$$

As a single target node can be activated by multiple source nodes, the activation weights need to be combined. For example, the weight combination function could simply calculate the average of the incoming weights:

$$f_{in}(w_1, w_2, ..., w_n) = \frac{\sum_i^n w_i}{n} \tag{117}$$

The intuition for the combination function is similar to the transfer function of neural networks. Therefore all methods used as transfer function can be utilised to combine the activation weights. Popular examples are sigmoid function, linear combinations and step functions.

Additionally the weights can be optionally be discounted by an decay function. As the weight is distributed throughout the network, it is decreased in each iteration until it peters out. For example the distance from

[293] J. R. Anderson. A spreading activation theory of memory. *Journal of verbal learning and verbal behavior*, 22(3):261–295, 1983

[294] F. Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6):453–482, 1997

[295] P. Scheir, C. Ghidini, R. Kern, M. Granitzer, and S. N. Lindstaedt. ARS/SD: An Associative Retrieval Service for the Semantic Desktop. *Networked Knowledge-Networked Media*, pages 95–111, 2009

the start node can be used to achieve such a behaviour:

$$f_{decay}(n_j, w_j, stepCount) = \frac{w_j}{stepCount} \tag{118}$$

The initial weight is iteratively processed and passed to connected nodes until a termination criteria is fulfilled. There are multiple ways to determine when to stop further traversal, for example:

- The number of hops within the network, to limit the region within the feature association graph
- A lower bound on the activation weight, as node with low weight will usually not contribute much to further processing
- Number of nodes visited, to limit the size of the data-structures
- A target node has been reached, where the target nodes can be specified either via their identifier or their assigned role

Usually a combinations of criteria are used to prevent excessive processing. Finally the result of the traversal has to be reported to the application. Depending on the application scenario, only the nodes activated in the final iteration are included in the result or all visited nodes.

Instead of a single result, an realisation of the spreading activation retrieval function may implement a callback pattern. This programming paradigm allows the application to be informed of every step within the traversal operation. The callback driven approach provides the highest flexibility, but the application itself has to choose which pieces of information should be stored for further processing.

The traversal itself can be customised if required by the application. For example the spreading can be limited to nodes carrying a certain meta-data. This way the traversal can be steered to include only nodes, which are relevant for the processing.

Distance Between Features The feature association graph can also be used to gain information about a pair of features not directed connected with each other. In this case the two feature are connected via a set of paths. Each path represents a sequence of feature nodes, starting with a source feature and an end node, which represents a target feature.

For such a traversal the feature graph is required to fulfil certain properties. The graph has to contain paths consisting of more than two nodes in order for this function to generate useful results. The feature association framework needs to be configured to generate such graph structures. One way to achieve a matching output graph is to label all features as source as well as target features.

Once the feature association graph can be traversed, the distance of two features within this graph can be computed. A small distance can be seen as indicator for a higher relatedness between these two features. The distance can either be measures by the length of the path or the aggregated weight. The path length is determined by the number of hops within the graph to reach the target feature while starting at the source feature. To compute the aggregated weight, each edge is visited and the weight is added to the cumulative weight. For feature association function, which represent distance measure the final aggregated weight will be the distance between the source and target feature.

Usually one is not interested in all possible paths which connects two features, but only the shorted path. Computing the shortest path between two nodes within a graph is a well studied problem in the field of graph theory. The Dijkstra's algorithm[296] is probably the best known approach to

[296] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959

this problem. As an additional property this approach requires the weights of the edges to be non-negative, which is the case for most of the distance measure provided by the feature association framework.

The run-time complexity of an simple implementation of the Dijkstra's algorithm is bounded by the number of features within the output feature association graph: $\mathcal{O}(|F|^2)$. For sparse output graphs it is expected that the actual performance of the algorithm will be faster. Still for many applications the cost for such a search will be prohibitive.

To increase the performance of the computation of the shortest path, two basic approaches can be followed. At first, the requirement on the single shortest path can be relaxed and then to optimise the sequence of calculations, several heuristics can be applied. For many application it is sufficient to work with approximations of the length between two features. For example to present a visualisation of the feature association network to the user, the exact distances are not required. In this scenario, related features should be displayed as neighbours and independent features can be placed far away in the visualisation.

The second strategy to improve the performance of the algorithm is to employ heuristics. The execution sequence of the graph traversal does not matter in the theoretical case, where the single best path should be found. But in a scenario where only an approximation to the optimum is needed, resorting to heuristics to govern the sequence of the execution is an effective strategy. For such an approach it is important that the heuristics actually match the available data. In the case of an feature association graph, the heuristics will be most effective if the weights between three features comply with the triangle inequality.

The A* algorithm[297] is based on the Dijkstra's algorithm and incorporates heuristics to speed up the computation. The run-time complexity depends mainly on the employed heuristics. Because of its good performance the A* is a popular choice to find the approximate to the distance between two nodes in a graph. One of the main applications of this algorithm is to find the "sufficiently good" route for navigating within a road network. In such a scenario the edges represent roads between locations and the weights encode the length of these roads. The heuristics incorporate the average expected speed and the direction of road in relation to the goal to guide the search within the road network.

[297] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968

Paths Between Two Features For some applications it might not be sufficient to compute the shortest path between two features. They require more then one path that connects two feature nodes in the association graph. Due to the sheer number of possible connections between the run-time complexity of such an algorithm is very high. Therefore usually other approaches are taken into account to compare two features within the association graph.

Compare Sub-Graphs Instead of collecting the complete information shared between two features, one can limit the traversal on the direct vicinity of the two feature nodes. Two sub-graphs should represents the context of for the two input features. These sub-graphs are generated by starting at a feature node and traversing all edges until a threshold count is reached. If a threshold of one is used, the sub-graph consists of a feature node and all directly connected features. The context sub-graph can then be compared to calculate a similarity of the two features.

In order to distance between a source and a target feature the association graph needs to be build to allow a traversal in both directions. To compare the contextual sub-graphs of two features this limitation is lifted. Therefore this method is generally preferred as a means to compare two features.

The contextual sub-graph reflect the relationship of one feature to other features, typically the associated target features for a given source feature. Thus the comparison of two contextual sub-graph can also be interpreted as indirect relatedness. In the field of natural language processing this approach is used to calculate the similarity of words according to their second order co-occurrences[298].

[298] R. Rapp. The computation of word associations: comparing syntagmatic and paradigmatic approaches. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, number 1992, pages 1–7. Association for Computational Linguistics Morristown, NJ, USA, 2002

The choice of the similar measure to compare two sub-graph depends on the nature of the features and on the application domain. All algorithms which are applicable as feature association function can be used for this purpose as well. The number of common associated features is a simple example for such a similarity function. According to this function, source features that are associated to the same target features are more similar than pairs of features that only have a few associations in common.

Recursive Associations Due to the flexibility of the feature association framework, the output feature association graph can again be used as input. That way multiple feature association calculations can be concatenated and executed as a sequence.

There are a number of scenarios where multiple feature association runs are the preferred way to solve a problem at hand. Two of these scenarios are now presented.

- The input to the feature association calculations is a directed graph. The nodes of this graph are mapped to a set of predefined roles. Thus the input graph consists of instance nodes, source feature nodes and target feature nodes.

  Assuming a bi-partide graph with the instance nodes and source feature nodes as two partitions. The adjacency matrix of the graph will be $M_{instances \times features}$. The feature association framework can be configured generate an output graph of which the adjacency matrix is $M_{features \times instances}$. Effectively the input matrix has been transposed.

  Given an application scenario where the relationships between instances instead of feature should be calculated. To achieve this at first the input matrix can be transposed as a first step. In the second step the output of the transposition operation is used as input for the desired processing.

- The calculation of so called second order co-occurrences is another scenario which illustrates the potential of using the output of a feature association run as input for the next. In this scenario, features are compared to each other. As base for the comparison, their relationship to other features is taken.

  The computation of the second order similarities two processing steps are needed. At first the relationship of all features is computed based on their distribution within the data-set. This will generate a feature association graph where features are strongly associated with feature they often co-occur with.

  In the second step, the target features of the first processing step are taken as instance of the input feature graph. The source feature are again marked as source, and additionally also as target features. Thus all feature of the initial input graph are compared to each other by using the associations of the first step.

  The two feature association run can be conducted using different configurations. For example, the feature association function does not need to be the same for the two runs.

The selection of the retrieval function depends on the application and the amount of information needed for further processing or for an visualisation

for users. An application might choose to invoke multiple different retrieval functions, or to invoke the same function with different input values multiple times. For example the feature association graph might be interactively explored by a user. Regardless of its final use-case, all retrieval function need to be efficiently implemented and provide the result in a minimal amount of time.

Distributed Retrieval

The feature association can be computed in a distributed execution environment. The same requirement is imposed on the retrieval of the feature association graph. Multiple clients may access parts of the association graph at the same time. This section should give a glimpse on how to approach this requirement. The actual implementation of a distributed retrieval architecture depends on the available resources and the expected type of retrieval requests.

No sophisticated execution framework is needed to accomplish a parallel retrieval of the feature association graph. A distributed file system can by utilised to make the association graph accessible from nodes within a cluster[299]. All incoming requests are dispatched to any of the nodes within the cluster. The nodes performs the requested operation and returns the result to the caller.

[299] Alternatively the feature association graph can be store in a centralised database or a Network-Attached Storage (NAS).

The scheme can be improved by various measures. The dispatching algorithm can implement algorithms to evenly distribute the load among the participating nodes. Such software components are called load balancers. The most simple form of dispatching is named round robin, where the sequence of incoming requests to delegated the nodes organised in a queue. For each request the first node in the queue is assigned that task and is then moved to the back of the queue.

The load balancer may also regularly poll the network to detect inactive of defective nodes. Additionally vital information, for example the amount of free memory and the processor utilisation of the cluster participants is monitored. Given the accumulated information the load balancer may use more sophisticated algorithms than just the round-robin method.

An optimal performance can be reached if the load balancer is made aware of the storage topology of the feature association graph within the network. In the distributed file system the feature association graph is partitioned and the individual parts are stored locally on multiple nodes[300]. For an incoming request that addresses a certain part of the association graph, the load balancer might select the node which locally stores that part of the graph.

[300] Usually multiple node will store copies of the same part. This redundancy increases the robustness of the distributed file system.

Even if a load balancer is not informed of the storage locations of the graph, there are heuristics to improve the throughout. For example it might implement a session affinity scheme. Consecutive request from the same client are dispatched to the same node within the cluster. This is motivated by the assumption that two consecutive request are related to each other and address similar parts of the feature association graph. Due to the processing of the first request that part is likely to be present locally in any further retrieval requests.

To sum up, due to the careful design of the feature association framework, the employed algorithms scale well with the size of the data sets. Small data-sets can be processed on a single execution unit. The feature association framework can also be integrated into existing distributed execution framework for the calculation of the feature association as well as the retrieval of the final association graph. The input and the output of the feature association calculations are stored and managed in a distributed file system. Thus even large data-sets can be processed by feature association framework.

Implementation

The reference implementation of the feature association algorithm serves as proof of concept of the usefulness of the proposed framework. The implementation has been developed to conform to a set formal requirements derived from practical and theoretical aspects of knowledge discovery applications. It has then be used in multiple real-world scenarios, where it delivered satisfying performance and demonstrated sufficient flexibility.

Introduction

The algorithms of the feature associations framework were developed and tested in conjunction with a series of prototypes. These prototypes were developed to test certain aspects of the framework and to measure the real-world performance of the algorithms. With each iteration of prototypes the number of feature did grow, making the framework more general and flexible. Thus the final feature association framework is shaped by the results of tests conducted on real-world data-sets.

Due to the importance of the implementations which accompanies the development cycle of the framework, the final reference implementation is presented in an own chapter. This chapter focuses on specific properties of the implementation, without going into details of the programmatic side, but highlighting certain aspects of the design decisions. Therefore the implementation serves as an example on how to solve problems which are not only specific to the problem of efficiently computing feature associations. Many of the challenges which had to be overcome are also found in a variety of knowledge discovery applications from various domains.

A series of functional requirements were defined to steer the development of the implementation. These requirements are motivated to check whether the actual code matches the expectations. The majority of the requirements were derived from practical aspects and experience gained in the field of the knowledge discovery.

The final implementation has then be used to generate and analyse the relationships between feature in various data-sets. Various knowledge discovery applications from different fields has successfully integrated the feature association framework. These applications profit from the effort invested into the creation of the reference implementation.

Functional Requirements

The functional requirements govern and steer the development of the reference implementation. As the design decisions which helped to define and shape the algorithm, the requirements should motivate the decisions on how to model the properties of the implementation. The functional requirements are derived from existing

KNOWLEDGE DISCOVERY APPLICATIONS AND FROM DEMANDS POSED BY THE SIZE OF AVAILABLE DATA SETS.

Functionality

The most important aspects of an implementation of an algorithm is that it follows the concepts and is adheres to the specification of the algorithm. The same implies to the implementation of the feature association framework. The proposed framework for the calculation of association between features specifies a number of properties and algorithmic approaches that should be found in the final implementation. Therefore the first and functional requirement can be stated as:

Functional Requirement 1 *All of the specified properties of the algorithm should be present in the implementation. The implementation should be organised the same way as specified. The execution of the individual phases of the algorithm should be done according to the formal design decisions.*

The next three functional requirement are mere refinements of the first requirement. There are listed as separate items as they represent different aspects of the framework. For the implementation it is important that each of these aspects is accounted for. The first aspect is the usage of data-structures. Any application that wished to incorporate the feature associations will communicate with the framework according to the specification of these data-structures.

Functional Requirement 2 *The data-structures should be implemented as specified by the algorithm, with a graph as basic data-structure for the input and output of the processing.*

As soon as the communication with other components is determined, the framework needs to be configured to produce the desired results. An important part of the configuration is the the feature association function. This function is responsible to calculate the association strength between two features. In the specification of the algorithm, the feature association function has been decomposed into a set of factors and functional blocks. Any function that conforms to this pattern can then by used within the framework.

Functional Requirement 3 *Any function that conforms to the formal definition of a feature association function should be eligible for the usage within the implementation.*

One of the design goals for the framework has been to provide as much flexibility as possible to allow the feature association framework to be integrated in a wide range of knowledge discovery applications. Therefore the framework provides facilities to customise the execution the algorithm. Additional processing can be integrated into the work-flow of the feature association calculation. The association phase is the prime candidate to integrate customised processing, for example to incorporate an machine learning algorithm into the association calculations. The implementation must therefore offer the same level of customisation.

Functional Requirement 4 *Full support of the proposed customisation features of the algorithm should also be provided by the implementation.*

Only if the implementation faithfully follows the formal scheme, it will provide all the proclaimed properties of the specification of the feature association framework. The algorithms are carefully designed to allow the

integration of the feature association calculation into existing frameworks for distributed calculations, for example MapReduce[301]. Therefore any implementation should adhere to the specification in order to provide an predictive behaviour. This will allow the integration of the feature association calculation in a number of different knowledge discovery scenarios.

[301] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, pages 1–13, 2008

Flexibility

The implementation should adhere to the specification, and should allow all operations as proposed by the algorithmic description. Additionally the implementation might choose to provide even more flexibility. Due to the nature of many knowledge discovery applications functional requirements can be formulated that capture a level of flexibility which exceeds the specification. One example for such a flexibility is the requirement to compute different configurations of the feature association framework simultaneously.

Functional Requirement 5 *The implementation should support multiple feature spaces within the data-set. Therefore a scheme should be supplied which allows a mapping of feature spaces of the data set onto the various roles within the input feature graph. This scheme allows a computation of multiple different configurations on the same data at the same time. Technically speaking, the implementation should process multiple independent configurations in one pass over the data.*

The feature association framework is specified to provide a set of retrieval function on the output association network. One of the methods is the recursive invocation of the feature association calculations, where the output of one invocation serves as input for the next. Therefore the choice of data-structures for the input and output is constrained.

Functional Requirement 6 *The data-structures that hold the input data should be compatible to the data-structure responsible of storing the result. This should allow an application to trigger a stacked execution sequence of feature association calculations.*

Scalability

Due to the complexity of the problem of calculating feature associations, even for small data-sets the number of individual relationships between the features can be very high[302]. For larger data sets the calculations are even more demanding. The requirements on the scalability of the algorithms and the implementation are of high priority.

[302] The theoretical upper bound is $\mathcal{O}(n^2)$ for n number of features.

In order to be able to assess the degree of scalability and to check whether the implementation matches the performance demands, a series of criteria have been established. A number of performance targets were defined and represent the next series of functional requirements. These targets depend on the size of the data set to processes. Beside the number of instances and features, the average number of features per instance is equally important. In the presented scenarios it is assumed, that the bi-partite graph of instances and features is relatively sparse, hence each instance references only small subset of all features[303].

[303] Dense instance-feature networks are as well eligible to be processed by the feature association framework, but a) they are not as frequent in common knowledge discovery applications and b) they can often be transformed to sparse networks by applying pruning strategies.

Small data sets may contain a few thousand instances and about as many features. The Brown corpus[304] is have been for a long time one of the main sources for corpus linguists. It consists of 56,766 sentences and 25,432 individual words. Thus the Brown corpus can seen by today standards as a relatively small data-set. Such small data-set should pose no problems when

[304] W. N. Francis and H. Kucera. Brown Corpus Manual. *Brown University*, 1979

processed by the feature association framework even when using moderate computational resources.

Functional Requirement 7 *The feature association calculation should be possible for small data-sets on contemporary commodity hardware*[305].

[305] At the time of writing a typical desktop PC is equipped with a processor, which allows two to four independent parallel computations, about 4 gigabytes of main memory and about one terabyte of secondary storage.

With the increase of computational resources, the size and number of available data set rises. The number of features as well as the instance count is increasing.

A medium size data-set consists of a couple of hundred thousand instances and about as many features. For instance the Reuters RCV-1[306] may still be considered as a medium size data-set. This data-set is made up by articles published by the Reuters new agency in the years 1996 and 1997[307]. The Reuters RCV-1 consists of a total of 806,791 articles and about 1.3 million individual words in the English language. Such data-sets are common and therefore their processing should not require a dedicated expensive computing infrastructure.

[306] T. G. Rose, M. Stevenson, and M. Whitehead. The Reuters Corpus Volume 1-from yesterday's news to tomorrow's language resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 29–31. Citeseer, 2002

[307] Starting with August 1996 to August 1997.

Functional Requirement 8 *Medium size data set are common for many knowledge discovery scenarios. Many organisation are not equipped with dedicated computational resources, for example a cluster of computers. Therefore the feature association calculation should be implemented in such a way to allow the efficient execution of the computations on a single machine*[308].

[308] Currently a server class machine is equipped with two to four processors, each of them about twice as fast as a desktop class CPU. The main memory size is up to 64 gigabyte and the secondary storage is typically a RAID with about 1 terabytes of data.

With the availability of the Web as a resource and the trend of user generated content, the size of data-set did increase even further. Due to this, large size data-set have become more common. Their size exceed the storage resources of commodity hardware. These corpora consist of millions of instances and millions of features. The collaboratively created encyclopedia Wikipedia consists of about 3.5 millions articles and roughly as many terms[309].

[309] The size of the individual articles vary in size. Categories, redirects, internal entries are not counted as articles. The number of entries in the English Wikipedia data-set is about 10 millions at the time of writing.

Therefore the feature association framework should not only be efficiently implemented, but the algorithms should have a low computational complexity. This is an important requirement also to achieve scalability.

Functional Requirement 9 *The implementation of the feature association framework should scale with the data set. Even large data-sets with millions of instances and features should be eligible to be processed. As these data-sets do not fit on a single machine, the implementation should be integrated into a distributed execution framework. This allows the computation to be done on a cluster of machines*[310].

[310] The cluster consists of commodity hardware which is similar in their processing power to desktop class machines.

To sum up this series of functional requirement, the implementation of the feature association calculation should be close to linear in terms of computational complexity in regard to the size of the data set. In other words, the number of machines needed for the computation should be rise equally to the increase of the data set size. For twice as many data one should require about twice as many machines, but not more.

Performance

Scalability is only one aspect of the run-time behaviour, the efficiency of the implementation is another. While the computational complexity is typically expressed using the big O notations, the run-time performance of an algorithm is typically measured in seconds. Even if two implementations are equal in their computational complexity, their run-time might vary.

There are many reasons why an implementation is faster than others. To achieve a high performance, the employed algorithms should be tuned on

the actual problem and the implementation should be tailored towards the execution environment. For example, on contemporary computers the access to the main memory is orders of a magnitude faster than any operation on the secondary storage. Furthermore the sequence of operation and access pattern has a huge influence on the run-time costs of an algorithm.

The functional requirement related to the performance of the implementation are as the scalability issued categorised according to the size of the input data-set. Each of the three functional requirements poses an upper bound on the execution time of the feature association calculations. All these upper bound are developed based on specific assumption about the properties of the input data. They are expected to be sparse and it is assumed that there exist efficient pruning strategies to effectively reduce the number of associations to calculate.

Functional Requirement 10 *Small data-sets should be processed within a time-span of a few minutes. The upper bound for the this category of input size is one hour*[311].

[311] When executed on a single contemporary desktop PC.

Sixty minutes might appear to be much in relation to the fact that contemporary search engine are able to search and rank billions of webpages within milliseconds. One reason for the discrepancy is the different complexity of the two problems. Calculating feature associations is bounded by mn^2 for m instances and n features, while a search operation has run-time complexity of $\mathcal{O}(m)$. Heuristics and optimisation strategies will lead to far better performance in the majority of cases for both problem settings. Still the feature association calculation cannot be integrated into an interactive application in the foreseeable future[312].

[312] This would require the feature association operation to last less then ten seconds. Currently this performance can only be achieved for very small data-sets or if aggressive pruning strategies are applied.

Functional Requirement 11 *The processing of medium sized data set should last no longer than a single day. It expected that for the most common corpora the execution time of the feature association calculation will be a few hours.*

Most of the real-world data-sets will be of medium size. Therefore the performance of the feature association calculations for such data sets appears to be critical. Combining the scalability and the performance requirements leads to an estimated execution time of a few hours when executed on a single server class machine.

Functional Requirement 12 *Large corpora will cause additional overhead for the pre-processing of the data set, the set-up of the distributed execution environment and the management of the parallel computations. In reverse the distributed execution will effectively speed up the computations by nearly a factor equivalent to the number of participants within the cluster. Due to this the computation time can be decreased by adding additional machines to the cluster. The upper bound for the distributed processing of large data set has been defined to be a week.*

Although the performance related function requirement define a set of worst case performance figures, the execution time is influenced by many factors besides the size of the input data set. The configuration of the feature association calculations, especially the employed set of pruning strategies, has implications on the run-time of the computations. The performance of the computers will continue to rise in the future, as will the size of the available corpora. Therefore the computation of the relationships between features within a data-set will require a sophisticated implementation, even if the available computational resource will become more powerful.

In table 22 the functional requirements concerning the scalability and performance are listed. For each category of data-set size an example is given, including the number of instances and features.

Data Set Size	Data Set	# Instances	# Features	Scalability	Performance
Small	Brown Corpus	56,766	25,432	Single Commodity PC	1 hour
Medium	Reuters RCV-1	806,791	1,366,708	Single Dedicated Machine	1 day
Large	Wikipedia	3,450,942	10,839,070	Cluster of Commodity PCs	1 week

Table 22: Overview of the scalability and performance related functional requirements. Depending on the size of the data set upper bounds on the necessary resources and execution time were defined. For the Brown corpus the sentences have been used as instances instead of articles.

The decoupling of the scalability related and performance related functional requirements is motivated by the fact that they require different strategies to achieve the goals. While the scalability is mostly determined by the computational complexity of the algorithms, the final performance is influenced by multiple factors. These factors include the architecture of the platform, the access patterns on the data and the quality of the implementation.

Applicability

The final set of function requirements is derived from the fact that the proposed feature association framework should be applicable for a wide range of knowledge discovery applications. These application will not only vary in the way they operate or their usage of different data-structures, but also on their execution environment. Some applications are developed to be executed on desktop machines, some are fabricated to be run on dedicated cluster and some application may be executed on mobile devices. Additionally no one can foresee which platforms will be available in the coming years.

Functional Requirement 13 *The implementation of the feature association framework should be compatible with as many execution environments as possible. This should allow the support of many knowledge discovery scenarios on multiple platforms. Technically speaking the development of the implementation should be cross-platform.*

It would be theoretically be possible to develop a code base from the ground up[313]. This would require enormous amounts of development effort. A better strategy is to re-use existing solutions and to build upon already developed software components and libraries. A functional requirement captures this strategy:

[313] This attitude of reinventing the wheel is shared among many developers and is commonly referred to as not invented here syndrome (NIHS).

Functional Requirement 14 *The code for the feature association calculation should be based on a solid foundation of existing and proven technologies. This can be achieved by including libraries, which have been developed in public and have established a flourishing community. Furthermore the interfaces of the implementation should conform to existing and established standards.*

The last functional requirement is similar in intuition to the previous one. But instead of using existing components, the feature association framework itself should be implementation to serve as a base for further developments. This should allow researchers and other interested people to integrate feature association calculated into their projects. Additionally a community of users should scrutinise the code and collaboratively enhance the code base.

Functional Requirement 15 *The implementation of the feature association framework should be made available to the public[314]. This will allow contributions from external developers and feedback from people working in the domain of knowledge discovery.*

[314] At the time of writing the task of contributing the implementation to the pool of open-source code has not been fully completed.

The final functional requirements are aimed to increase the practical applicability of the feature association framework. They are motivated to develop an implementation that is suitable for a wide array of use-cases. With the number of participating developers the quality of the implementation will rise. The gathered feedback from practitioners should steer the future development of the implementation.

Data Structures & Runtime Environment

This section covers the technical base and the main libraries of the reference implementation of the feature association framework. For each decision alternative approaches are presented and the reasons for the final decisions are given.

Data Structures

One of the main decisions when developing the implementation is the choice of storage for the input and the output of the calculations. The input for the feature association computations is a n-partite directed graph. Node and edges within the graph may carry additional meta-data and weights. The output is also a directed graph, but it is not partitioned. The edges carry a weight and can be annotated with meta-data.

One of the functional requirements states that the input and output should be the same data-structure. Therefore the first choice to make for the implementation is the selection of a data-structure to store a directed graph. This data-structure should provide means to quickly traverse the graph. It should be flexible to allow the meta-data to be stored alongside the nodes and edges.

Tn the association phase of the algorithm parts of the output graph are iteratively generated. Therefore the graph data-structure must be capable of inserting sub-graphs into an existing graph.

There are many graph libraries available commercially as well as open-source implementations. For example WebGraph[315,316] which employs compression technique to keep the storage space of the graph at minimum. Only few of them provide facilities to transparently load parts of the graph into main memory. The iterative process of building the output graph is also on the list of required features, but only a few existing graph libraries implement such a scheme.

[315] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602. ACM, 2004

[316] P. Boldi and S. Vigna. The webgraph framework ii: Codes for the world-wide web. In *Data Compression Conference, 2004. Proceedings. DCC 2004*, page 528. IEEE, 2005

Instead of using a dedicated graph library, the reference implementation uses an inverted index as basic infrastructure to store and manage the input as well as output graph structures. This may sound unintuitive at first, but in fact it provides a number advantages. An inverted index is the basic data-structure of many search engines. They are well studied and offer a satisfying performance for a set of operations. An inverted index can be build in an incremental manner. Furthermore this data-structure has proven in many applications to provide a good scalability.

The input to build an inverted index is a list of tuples. Each tuple consists of a single identifier and a list of values:

$$\begin{aligned}[&< id_1, [value_1, value_2, value_1, ...] >, \\ &< id_2, [value_2, value_1, value_3, ...] >, \\ &\quad \vdots \\ &< id_n, [value_1, value_3, value_2, ...] >]\end{aligned}$$

The inverted index rearranges the entries and reverses the direction of the mapping. The output of building an inverted index is again a list of tuples. But the output tuples consists of a single value and a set of identifiers.

All identifiers which refer to a specific value are contained in the appropriate entry. Furthermore the number of times a specific value occurs within a single tuple is recorded and stored together with the identifier. No information[317] is removed in this operation, just the main axis of how the data is organised has been changed:

[317] The information that encodes the sequence of values is actually removed. Implementations of an inverted index may capture this information in dedicated data-structured stored alongside the entries.

$$
\begin{aligned}
&[< value_1, [< id_1, 2 >, < id_2, 1 >, ..., < id_n, 1 >] >, \\
&\quad < value_2, [< id_1, 1 >, < id_2, 1 >, ..., < id_n, 1 >] >, \\
&\qquad \vdots \\
&\quad < value_m, [< id_1, tf_{1,m} >, < id_2, tf_{2,m} >, ..., < id_n, tf_{n,m} >] >]
\end{aligned}
$$

The input of the operation can also be seen as matrix, with the identifiers as rows and the individual values are columns. The occurrence count of a value for a single identifier is the cell content. For this matrix the inverted index can be seen as transposition operation.

Inverted Index

After the basic data-structure for the management of the input and output graph has been agreed on, an implementation for this data-structure needs to be selected[318]. The implementation of the inverted index has to provide a level of flexibility to allow the management of the additional information. As the graph elements might be annotated with meta-data and the edges may carry a weight, the implementation has to offer methods to store this information within the inverted index.

[318] Developing an inverted index from the ground up would go against the functional requirement 14.

In the case of the feature association graph the input to the inverted index operation contains an optional weight - $w_{i,j}$ - and an optional data-structure to encode the meta-data - $M_{i,j}$:

$$
\begin{aligned}
&[< id_1, [< value_1, w_{1,1}, M_{1,1} >, < value_2, w_{1,2}, M_{1,2} >, < value_3, w_{1,3}, M_{1,3} >, ...] >, \\
&\quad < id_2, [< value_1, w_{2,1}, M_{2,1} >, < value_2, w_{2,2}, M_{2,2} >, < value_3, w_{2,3}, M_{2,3} >, ...] >, \\
&\qquad \vdots \\
&\quad < id_n, [< value_1, w_{n,1}, M_{n,1} >, < value_2, w_{n,2}, M_{n,2} >, < value_3, w_{n,3}, M_{n,3} >, ...] >]
\end{aligned}
$$

There are a number of different implementations of an inverted index available. Results of an evaluation of the most popular libraries have also been published[319]. When restricting the selection of available libraries to the Java based implementations a list of 4 candidates remains: Lucene, MG4J, OmniFind and Terrier.

[319] C. Middleton and R. Baeza-Yates. A comparison of open source search engines. Technical report, 2008

The last one, Terrier, does not support the incremental indexing and therefore is dropped of the list. The storage demands of the final data-structure is another criteria for selection. OmniFind produces on-disk data-structure that exceed in size the storage requirements of the data-set itself. Out of the remaining two solutions, the MG4J library does not provide facilities to store the additional information needed to manage the meta-data annotations.

Therefore the open-source library Lucene[320] has been the logical choice as basic infrastructure to store, manage and retrieve the input as well as output graph structures.

[320] `http://lucene.apache.org/java/docs/index.html`

Lucene Data-Structures The basic entity for the Lucene inverted index implementation is called document. A single document is made up by multiple fields. Each field is a tuple of a name and a list of values, called tokens.

The tokens are again tuples of a name and an optional payload. This payload can be freely defined by the client of the library and is treated by Lucene as a blob of binary data. The structure of a single Lucene document can be written as:

$$< document > \mapsto < field > *$$
$$< field > \mapsto < field - name >< token > *$$
$$< token > \mapsto < token - name >< payload >?$$

Input Feature Graph The first part of work to integrate the Lucene library into the feature association calculations is to define the mapping of the input feature graph onto the Lucene data-structures. The input is a directed graph with instance and feature nodes. Instances are connected via edges to the features, where each edge may carry a weight. Furthermore a feature node is assigned to one of the feature space of the data-set.

For the mapping, the input feature graph is partitioned into overlapping sub-graphs. Each of these sub-graphs contain a single instance node and all connected feature nodes. The instance nodes are not shared between the sub-graphs, hence there are as many sub-graphs as there are instance nodes within the input feature graph.

The sub-graphs are then mapped to Lucene documents. The fields of this document are determined by the feature spaces of the feature nodes within the sub-graph. Thus the fields represent the available feature spaces. All feature nodes within a single sub-graph are then transformed into a single field.

Feature nodes are represented as tokens. The identifier of the feature is used as token name. The edge weights and all meta-data is stored as payloads which accompany the tokens. Table 23 gives an overview of the mapping of the input feature graph to the Lucene data-structures.

Graph Element	Lucene Data-Structure
Instance Node	Document
Feature Space	Field
Feature Node	Token
Edge Weight	Payload
Meta-Data	Payload

Table 23: Overview of the mapping of the elements of the input feature graph to the corresponding Lucene data-structures.

To illustrate the mapping the Brown corpus has been transformed into a Lucene data-structure. In figure 24 an screenshot is presented that displays a single instance sub-graph. The depicted Lucene document contains a set of tokens which represent the features connected to the instance.

Applications that utilise the reference implementation of the feature association framework are responsible to transform the data-set into an appropriate representation. This will typically require an additional transformation step, which has to be developed once for each data-set. As the inverted index is a common and well understood data-structure this transformation step can be easily conducted.

Output Feature Association Graph The output data-structure is related to the way the input data-set is stored. Functional requirement 6 states that it should be possible to use the output of a feature association calculation as input of a successive iteration. Therefore the implementation also uses an Lucene inverted index as data-structure of the output graph.

While the input graph consisted of instance node which are mapped as single documents, the output feature association graph only contains feature nodes. Therefore a different mapping scheme needs to be developed, but the basic strategy stays the same. The complete output feature association graph is split up into overlapping sub-graph. Instead of using the instance

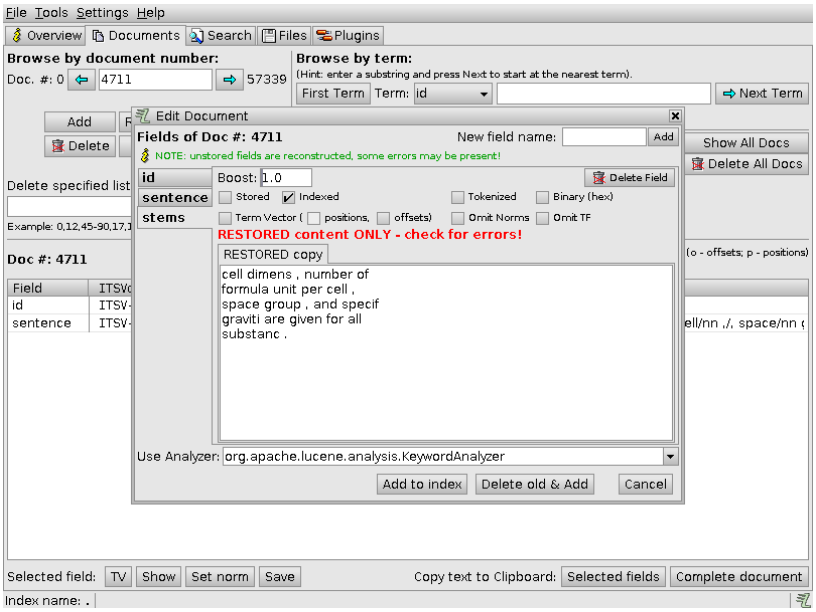

Figure 24: Example of a data-set transformed into the Lucene data-structure. Sentences within the Brown corpus are converted into document. The words are taken as features and the sole feature space has been labelled 'sentence'. The weights and meta-data are not shown in this example screenshot.

nodes, the sub-graphs are created based on the feature nodes labelled as target features. For each target feature node all connected source features are collected and resemble a single sub-graph.

Target feature sub-graphs are then mapped as single Lucene documents in the inverted index. The document is filled with the information contained in the sub-graph. Fields are created for each feature space present in the set of connected source features. For each source feature node, a token is added to the appropriate field. The token name is derived from the source feature identifier. The payload for a single token is holds the association weight and all meta-data attached to the connection between the target and the source feature. Table 24 gives an overview of the mapping scheme used to transform the output feature association graph into data-structure suitable for the inverted index.

One of the main use-cases for extracting information out of the feature association graph is the traversal of the graph starting at a source feature. Given a specific source feature the list of all association target features should be retrieved. Therefore the decision to use the target feature as criteria to partition the association graph may sound counter-intuitive at first. The choice of using the source features as main element of the sub-graphs might appears to be more suited to cater for the main use-cases.

The decision is grounded by the way the inverted index operates. Any implementation of an inverted index will reverse the direction of the relationships between the elements of the index. When building a Lucene index, the relationship between documents and tokens will effectively be inverted. Therefore the input data to the index must be organised in reverse to the expected main retrieval direction. In the case of an feature association graph, the relationship between the target and source features will be inverted. Hence in order to achieve to correct traversal direction one has to fill the inverted index with data organised in the logically wrong direction.

The output of a feature association calculation for the Brown corpus is depicted in figure 25. A single Lucene document represents a sub-graph for a single target feature. All source features, which are associated with the target features are indexed as tokens.

The final inverted index of an output association feature graph can again

Graph Element	Lucene Data-Structure
Target Feature Node	Document
Source Feature Space	Field
Source Feature Node	Token
Association Weight	Payload
Meta-Data	Payload

Table 24: Mapping of the elements of the output feature association graph to the Lucene data-structures. For each node in the graph, which is labelled a target feature, a single Lucene document is created. All associated source features are written as tokens of a field which is determined based upon the feature space of the source feature.

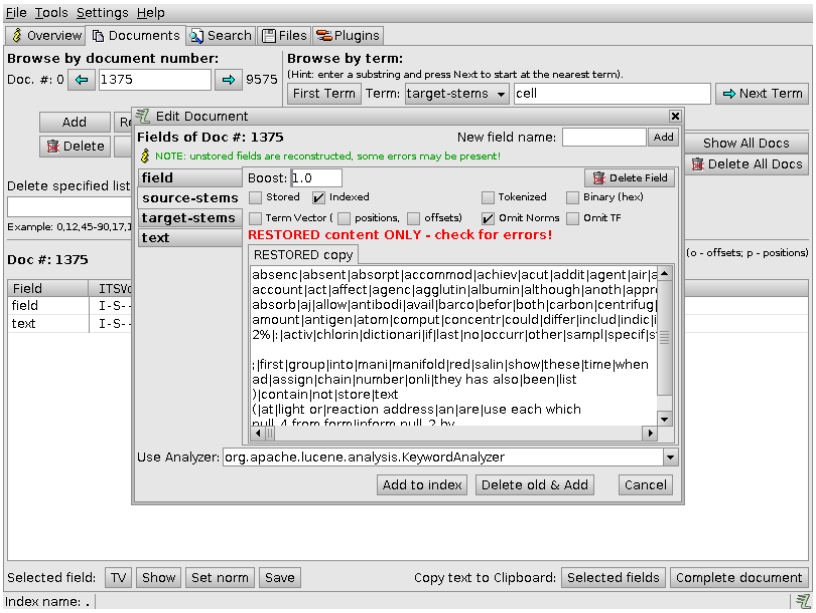

Figure 25: Screenshot of an example for a document that represents a sub-graph of the output feature association graph. The sub-graph contains a single target feature and all associated source features. In this example the target feature is the word 'cell'. The weights and the optional meta-data are not shown in the screenshot.

be used as input for another feature association calculation. Therefore an appropriate configuration needs to be specified. Due to this choice of input and output data-structures many complex use-cases can be supported, for example the computation of second order co-occurrences[321].

[321] There is no limit on how many times this recursive iteration is conducted. Therefore third or fourth order co-occurrences may be calculated as well.

Intermediate Data-Structure The third main data-structure holds the local associations. In the collection phase all features which share at least a single instance are jointly stored. The collected data is then sorted according to the features labelled as target.

The data-structure needs to be efficient in terms of required memory space. This requirement is a consequence of the employed merge-sort step that precedes the association phase. The more local associations fit into main memory, the less merge blocks needs to be created. A merge block consists of a list of already sorted local associations. In the merge step of the algorithm all blocks are read out in parallel. Therefore the size of the intermediate data-structure has a direct impact on the performance of the overall computations.

As this part of the implementation is critical to the whole process, a custom data-structure has been developed. As no already existing library offers the needed flexibility and speed, all algorithms related to this data-structure also need to be implemented. This requires custom implementations of merge-sort algorithm, the block storage logic and the parallel read-out code. To keep the development efforts low, the data-structure should be as simple as possible. The final intermediate data-structure to store the result of the collection phase is organised as:

$$< block > \mapsto < block - entry > *$$
$$< block - entry > \mapsto < target - feature - id >< instance - entry > *$$
$$< instance - entry > \mapsto < instance - id >< feature - space - entry > *$$
$$< feature - space - entry > \mapsto < feature - space - name >< feature - entry > *$$
$$< feature - entry > \mapsto < source - feature - id >< local - association - weight >< meta - data >$$

If the feature associations are calculated within a distributed execution environment, the management of the intermediate data is done by the execution framework. For example the MapReduce framework already provides facilities similar to the merge-sort algorithm to efficiently dispatch the work-load among the participating execution units.

Design & Main Components

Before any code can be written the overall design and architecture of a software component has to be developed first. The software architecture has a huge influence on the performance and the scalability of the implementation. Additionally the usability of the software interfaces contribute to the ease of adaptation on various application scenarios.

Overview

The design of large software projects requires a array of skills and experience. An important prerequisite of any design phase is a set of use-cases and well engineered feature requirements. Additionally a set of implicit assumptions and expectations steer the direction of the software design. Among these soft requirements there is the goal to achieve an aesthetic and elegant architecture. Such an architecture is accomplished by complying to a set of properties. The final software design should be consistent, symmetric, self explanatory and sufficiently simple.

The result of the software design process is a specification of a set of classes[322]. Each of the classes represents a single entity within the software architecture. The design also defines the interaction between the classes. Therefore each class provides a set of operations, where each takes a set of input parameters and produces an output that conforms to a predefined structure.

[322] Although the design process is independent from any specific programming language, there is the assumption that the implementation has to be done using an object oriented programming language.

The implementation of feature association framework can be split into three basic set of classes. These components encapsulate a specified functionality and serve different purposes.

- The first set of classes provide facilities to configure the feature association process. For each data-set and application scenario a suitable configuration needs to be defined.
- Given a data-set and a configuration, the feature association can be calculated. Dedicated classes process the input data, compute the association and produce the final output feature graph.
- The final component consists of classes to traverse the output graph. They provide means to retrieve the feature associations and to collect the required information.

The relationship between the three components is also predefined by the software architecture. The configuration classes are read out by the the two other components. There is no direct relationship between the classes that build the output feature graph and the classes that traverse the graph. Only the data-structure is shared between these two components.

In the next sections the software components will be presented. The main classes of the components will be described and the most important operations are covered. Additionally remarks regarding some aspects of the implementation are made.

Configuration

The task of the classes of the configuration components is twofold. At first the configuration is generated by the application and is tailored towards a specific data set. Then the configuration is read out by the components responsible to build and traverse the feature association graph.

Although the task of the configuration is straightforward and may appear trivial, nevertheless the classes of this component have to be carefully

designed. The configuration classes are the main interface of the feature association framework as seen from clients that wish to make use the functionality. Therefore the configuration should as simple and intuitive to use as possible. Still the whole range of possible computations should be accessible.

AssociativeFeature Class Within the implementation of the feature association framework, a single feature is represented by the class AssociativeFeature. All information concerning a single type of feature is captured by an instance of this class. The type of feature is logically equivalent to the feature space. As feature spaces are mapped as Lucene fields in the input data structure, the field type is equal to the field name within the inverted index.

The AssociativeFeature class is also responsible to manage tuning parameters. For example the threshold for various pruning strategies are stored alongside the instances. In table 25 an overview is given of pruning parameters commonly used for textual data sets. For different kinds of data, the appropriate classes need to be adapted to cater for a different set of pruning strategies.

Name	Description	Examples	Default Value
minTermLength	Remove all short words	Single characters, punctuation marks	2
maxTermLength	Remove all long words	URLs, tokenisation errors	∞
minDocFreq	Remove in-frequent words	Spelling errors, preprocessing artifacts	3
maxDocFreqRatio	Remove high frequent words	Functional words with low semantic content	0.5
maxTermDigitRatio	Remove numbers	Figures, dates	0.5

Table 25: Overview of common pruning parameters managed by instances of the AssociativeFeature class. Default values are given, which can be changed according to the properties of the actual data-set.

The AssociativeFeature class is abstract therefore no direct instances of this can be instantiated. Any realisation of such a class additionally encodes the role of a feature within the input feature graph. Therefore there are two classes, which are derived from the abstract AssociativeFeature class.

SourceFeature Class The first realisation of the abstract associative feature class is used to represent features labelled as source within the input feature graph. The mapping of the roles to the nodes in the graph is done via the feature spaces. Therefore the identifier of a feature space is stored within each instance of the source feature class. This property is inherited from its parent class, as well as all pruning settings.

Instead of introducing a new dedicated class to encode the role of a node in the graph, one could have chosen to resort to an attribute of the AssociativeFeature class. Due to the importance of the information whether a feature node is labelled as source or target feature a single property might not be sufficient to indicate this information. Furthermore this scheme allow separate settings when a single feature space is used as source as well as target of the associations.

TargetFeature Class The target feature class is similar to the source feature class. It is inherited from the AssociativeFeature class and is used to represent target nodes within the input feature graph. Instances of this class contain a member which identifies the feature space of all nodes

labelled as target. The pruning settings are inherited as well and the feature will be processed and filtered out according to these parameters.

The feature association calculation uses this information to generate the appropriate associations. Features identified via the TargetFeature class will then be the target of the association relationships within the output graph.

Feature Association Calculations

The main component of the feature association framework is responsible to analyse the data-set and to build the association network. The major challenge in the design of this component is to create a software architecture which provides the highest level of flexibility without sacrificing scalability or performance. Many aspects of the design are already predetermined by the algorithm and the data-structures. Therefore the main task remains to identify the appropriate mechanism to allow customisations to the basic work-flow.

There is a number of sophisticated design patterns which appear to be suitable for such a use-case. The Java programming language and its run-time environment are well suited to follow these design patterns. A combination of interface specifications and abstract classes build the core of the feature association calculation component.

AssociationBuilder Interface At first an interface is defined, which is exposed to the clients of the feature association framework. This interface is kept very simple and provides a single method. It takes a reference to a data set as input and returns a reference to an association network as output. The method is once invoked to build the complete feature association graph. Any internal operations and processing is shielded away from the clients.

In an simple application scenario, the main method of the AssociationBuilder interface is the only necessary interaction between a client and the framework. Therefore the AssociationBuilder interface is all a client needs to know to be able to create a feature association network. This is a direct consequence of the design decision 1, which has been made while developing the algorithm.

Complex scenarios require adaptation work and the implementation is specifically designed to allow different kinds of customisations. In order to integrate additional functionality and to replace existing code the implementation provides an abstract base class. This base class has been developed to allow any derived class to observe the intermediate result and to modify the process.

AbstractAssociationBuilder Class The AbstractAssociationBuilder class implements the core functionality of the feature association algorithm. It serves as base for any custom functionality as well as the default behaviour. Therefore it provides several hooks to inject custom code. The Java programming language allows the definition of protected member functions, which then can be overloaded by derived classes. This constructs has been used as main means to achieve a flexible and intuitive architecture.

The AbstractAssociationBuilder class is declared to be abstract and cannot be instantiated, only derived classes are eligible for being used by a client. Only clients that wishes to customise the feature association process need to be knowledgeable about the inner workings of this class.

Because the main functionality is made available via this class, its design should reflect the properties of the feature association computation. Therefore its specification is tightly coupled to the algorithm. For each distinct phase of the algorithm, the AbstractAssociationBuilder class

provides an own method. The input and output variables of these method are as predefined by the specification of the algorithm.

The first place for a customisation is to allow clients to gather global statistics of the instances and feature contained in the data-set. These statistics can then by used by the feature association function to determine the association strength.

The global statistics can also be used by pruning algorithms to remove any features and instances which do not contribute any relevant information to the final associations. The pruning strategies depend on the nature of the features and their distribution within the data set. The implementation of the feature association framework already provides a set of pruning approaches. Still it should be possible to plug-in additional customisation algorithms to remove or retain specific features.

The most important phase of the feature association calculations is the association phase. Therefore this part of the framework is a candidate to be customised to adapt the feature association process to a given application scenario. The AbstractAssociationBuilder class provides a two levels of customisation:

- The first level is tailored towards application scenarios which only require small changes to the feature association calculations. In the final part of the association stage sub-graphs are generated for each target feature. These sub-graphs are then inserted into the output feature association graph. The implementation allows the integration of an additional processing of the sub-graph prior to merging it with the rest of the association network. This can be used for example to remove specific pairs of features. The sub-graph may also be enhanced, for example to add meta-data to the output feature nodes.

- The second level allows a modification of the code that is responsible to produce a sub-graph for a single target feature. This part of the algorithm collects all local association for a given target feature. Then the local associations are aggregated and in combination with global statistics the feature association function produces weights for each pair of features. The source features and the target features are then wired together and annotated with the association weight as well as optional meta-data.

  Clients may choose to skip the default behaviour to build the target feature sub-graphs. There are no limitations on the kind of functionality which replaces the implementation of the association phase. It is only required to conform to the specified input as well as the output data-structures. The output needs to be a valid graph structure, where nodes represent features.

  This level of customisation can be utilised to incorporate a machine learning algorithm into the feature association calculations. Therefore the input can be interpreted as matrix, which is the preferred data-structure for many machine learning techniques. The result of the additional computations can then be integrated into the process of finding relevant relationships between the individual features.

Although the AbstractAssociationBuilder class encapsulates the main aspects of the feature association calculations, not all functions are directly implemented by the class. There a many auxiliary classes which are responsible for specialised tasks, which are not covered here due to space limitations. For example a sub-module implements the merge-sort algorithm.

DefaultAssociationBuilder Class The basic functionality of the feature association calculations are implemented by the AbstractAssociationBuilder class. This class is abstract and some methods are just placeholders for the optional customisation. Therefore it cannot be directly instantiated to build the feature associations.

The DefaultAssociationBuilder class is responsible to provide a complete basic implementation. In an simple application scenario this default implementation will be the preferred choice. The behaviour of this class conforms to the formal specification of the algorithm. The results produced by this implementation are labelled as global associations in the description of the algorithm.

ContextualAssociationBuilder Class The ContextualAssociationBuilder class has been created to generate the output graph consisting of local associations. Global associations reflect the relationship between two feature based on the whole data set. The data set can be partitioned into smaller parts, of which each represent a different context. The relationships between features based on this context are labelled as local associations.

The ContextualAssociationBuilder class is the base for building local associations. As there is no single best way to calculate local associations, this class is abstract and thus an additional adaptation effort is necessary. The necessary remaining steps for application scenarios where the contextual associations should be produced are:

- At first the data set has to be partitioned into distinct contexts. The partitioning does not need to be static, but may depend on the target feature. The implementation is free to choose a scheme to define the set of contexts for each target feature within the data-set.

- The context needs to be mapped onto elements of the output feature association graph. There are two basic approaches to deal with contextual features. In the first approach the target feature nodes are split into multiple nodes. Each of the new nodes represent a single context. Edges between the source features and the contextual target feature nodes denote the local association relationships.

  Alternatively the contexts may be mapped as a new type of nodes. For each context a new context node is introduced into the association network. The source features are connected to these context nodes via edges carrying the local association information. The context nodes are then connected to the target feature node. Edges between these two nodes types may be annotated with additional information about the context. The weight of this relationship might reflect the size of the context.

To generate local association between features the adaption effort is higher than for building a feature graph consisting of global associations. Examples for different scenarios where local associations are generated are presented in the application chapter.

AbstractFeatureAssociationFunction Class The last presented class of the feature association component defines the interface for the feature association functions. In table 26 an overview is given of the methods defined by this interface.

All feature association function need to be implemented by conforming to the interface specification. The feature association framework is already equipped with a number of common association functions. Still the application scenario might require to develop custom feature association functions, or need to adapt existing ones. Usually this is done in conjunction

Method	Phase	Description
`getLocalSourceWeight`	Collect Phase	Given a source feature and an instance compute a local weight
`getLocalTargetWeight`	Collect Phase	Given a source feature and an instance compute a local weight
`getLocalAssociationWeight`	Collect Phase	Combine the local weight of a source and target feature which share a common instance
`normalizeLocalAssociationWeight`	Association Phase	Compute a normalised version of a local weight based on statistics of all local weights for a target feature
`aggregateLocalWeight`	Association Phase	Aggregate the local weights of all instances (of a single context) for a given target feature
`getAssociationWeight`	Association Phase	Compute the final association weight between two features

Table 26: List of methods of the AbstractFeatureAssociationFunction class. Each method is invoked once for either a feature or a pair of features. All methods may access global statistics and pre-computed factors.

with a custom implementation of the association phase and a custom set of pre-computed statistics and factors.

Feature Association Retrieval

The third component of the reference implementation of the feature association framework contains classes to retrieve the feature association graph. The main challenge when designing an software architecture for this part of the framework is to take into account all the different use-cases. As the specification of the algorithm states, there should be multiple ways on how to traverse the output feature graph.

The most complex use-cases can be seen as a combination of applying a sequence of basic operations on the association network. Therefore not all use-cases need to be addressed by the implementation. Only the most important retrieval operations need to be solved. Among the criteria which steer the development of implementation, the final run-time performance plays an important part. The speed of the basic operation will have an huge influence on the execution time of more complex operations.

Two basic approaches to the traversal of the feature association graph have been identified. The first is to collect all directly connected feature nodes for a given start node. As the association network is a directed graph, only nodes labelled as source features are eligible as start nodes. The seconds method of traversal is more complex, but provides much more flexibility. In an spreading activation approach the graph is navigated, again using a source feature as start node. During the traversal all the necessary information is collected.

SubGraphAssociationSearcher Class The task of the SubGraphAssociationSearcher class is relatively simple: Given a source feature node, collect all associated target features. As the output feature graph may be very large and highly connected, a number of constraints and threshold should prevent excessive processing. The two main thresholds to limit the amount of data gathered by the implementation are:

- The first threshold defines the upper limit on the number of collected target features. Simply stopping the collection of target feature as soon as this threshold is reached will not be sufficient in most of the use-cases. Usually the desired output is a list of relevant associated features, where

only the feature with low association weights are ignored. Therefore the set of all associated target features needs to be gathered. From this list the top features with the strongest association will be retained, all other target features will be removed.

To achieve such a behaviour, it is not necessary to entirely sort the list of top associations, which would be quite computationally expensive. For example the bounded priority queue algorithm[323] can be utilised in such a scenario. The collection algorithm is initialised with an empty list of target features. While the graph is iteratively read out and all connected features are collected, the list grows with each visited feature. As soon as the number of entries of the list is equal to the threshold, an new feature is only added to the list if its association weight is higher then the smallest weight in the list. If this is the case, the feature with the smallest weight is removed to keep the size of list unchanged.

There are multiple ways to implement an priority queue scheme. In the reference implementation the heap data-structure has been chosen to collect the top associated features.

[323] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT press, 2009

- The second threshold does not require a sophisticated implementation. A fixed minimal association weight is used to constrain the number of reported associations. All connected target features with an higher association weight are included in the result list. This threshold requires a list implementation, which allows incremental growth. The Java programming environment already provides a number of matching data structures.

Although the task of the SubGraphAssociationSearcher class may appear trivial, it is crucial that the implementation is crafted with emphasis on high efficiency. This class is used by other, more complex retrieval operations. The methods of this class are invoked multiple times and therefore their run-time behaviour has a huge impact on the overall performance of the framework.

Various techniques have been used in the implementation of the SubGraphAssociationSearcher class to ensure an optimal performance. The first presented optimisation strategy only applies to small output feature graphs. The latency and transfer times of secondary storage devices are orders of a magnitude higher than accessing the main memory. Therefore the implementation tries to load the complete association network into main memory. This works for sufficiently small feature association graphs.

As soon as the size of the output feature graph exceeds the memory capacity, other strategies have to be considered. The reference implementation combines two main approaches for efficient retrieval: pre-processing and caching.

By default the output feature association graph is stored within a Lucene index as presented previously in this chapter. The graph contains all associations and all meta-data. Usually only the top associations and a limit set of meta-data is requested by the client. Therefore the association network is pre-processed and only the most important information is stored in a dedicated data-structure[324]. Before any requests are passed to the SubGraphAssociationSearcher class, or alternatively on the first request, the pro-computation is initiated. The pre-computed data consists of each source feature together with a fixed number of relevant associated target features. This number needs to be at least as high as the threshold then passed to retrieve the associations for a single source feature.

[324] On contemporary computers it is more efficient if all requested data is located close to each other. If the associations are spread out on the storage area, latencies will be introduced which will cause noticeable delays.

The entries for all source entries is then written as file to the secondary storage. Next the file is opened for reading, but instead of using the default streaming approach of accessing a file, an alternative, more efficient approach is taken. The content of the file is mapped into the address space of

the current processes. This approach allows the operating system to optimise its disk caching algorithms. Therefore the reference implementation itself does not need to cache any data. Modern operating system are more efficiently at caching then any application. The system components may refer to information like the disk geometry, physical storage location and more. It is the responsibility of an application to organise its data to be suited for the operating systems' caching algorithms. The pre-computed data-structure has specifically been designed in such a way.

SpreadingActivationAssociationSearcher Class The first presented retrieval class is optimised on speed, which is made possible by its narrow specified functionality. The SpreadingActivationAssociationSearcher class on the other hand has been designed with flexibility in mind. It provides a rich set of parameters to tune its behaviour. The basic task it to traverse the association network in a spreading activation like manner.

The spreading activation is conceptually similar to a breath first search. But there are a couple of differences, for example in the spreading activation scenario each node may be visited multiple times. The breath first search does not keep track of an activation strength, whereas the spreading activation traversal carries an activation weight and hop count for each thread. Furthermore the spreading activation may incorporate a decay function to simulate that a activation weight peters out after a couple of hops.

The output of the spreading activation retrieval is the list of nodes activated in the terminal hop. As the SpreadingActivationAssociationSearcher class far designed to provide as high flexibility, there are a number of constrains and thresholds to steer the traversal. The main parameters and input variables will be presented in the following paragraphs.

Start Node The minimal required information to trigger a spreading activation of the association network is the start node within the graph. Given an initial source feature the traversal will start and the predefined default settings will control the execution of the operations. If no other parameters are specified the result of this operation will be the same as the basic use-case covered by the SubGraphAssociationSearcher class.

Connected Nodes Threshold For each iteration of the spreading activation algorithm all nodes connected to the currently activated nodes should be visited. The output feature association graph might be highly connected. In the worst case each feature is connected to each other feature of the data-set. To keep the calculations feasible and the keep the memory requirement within certain limits additional thresholds are introduced. The first is the upper limit of nodes visited in the next iteration for each currently activated node. This threshold is basically the same as the first presented upper bound of the SubGraphAssociationSearcher class. Only the feature with the highest association weight will be visited. This threshold is applied for each hop and each node.

Association Weight Threshold As the previous threshold this parameter has already been introduced in the description of the SubGraphAssociationSearcher class. For each hop only those connected nodes are visited, where the association weight exceeds the passed value. The actual best value for this threshold depends on the feature association function employed while building the output graph. Different association function produce values of different ranges. Even the direction of relatedness may be swapped for different functions. For example similar measure produce high values for similar feature, whereas distance measures will produce low values for close features.

Activation Weight Threshold Unique to the spreading activation retrieval function is the specification of the lower limit on the activation weight. The activation weight depends on a number of factors, most of them have already been covered in the previous chapter. This threshold is similar to the association weight threshold, but it is not applied on the weight of the association, but on the result of the activation weight calculation[325]. If the activation weight is lower than the passed threshold value, the traversal is stopped for the current thread. As soon as all threads are terminated by this criteria, the retrieval process is finished.

[325] As a node might be activate by more than one node at the same time, this threshold can only be applied after all candidates for the next iteration have been determined. Therefore this method is less effective to increase the computations as the association weight threshold.

Hop Count Like the previous threshold, this parameter controls the point of termination of the traversal process. Instead of using the current activation weight, the length of a thread is taken as reference criteria. As soon as the number of hops for a thread is equal to the specified value, it is terminated. The last activated node is then added to the result list together with its activation weight.

If an hop count of one is passed to the algorithm, the retrieval will be equal to the way the SubGraphAssociationSearcher class traverses the graph. Which hop count to use depends on the structure of the feature association graph. This threshold can be used in combination with a lower limit on the activation weight to create a combined termination criterion.

Middle Node Feature Spaces A data-set may contain features from different feature spaces. In the input feature graph the features are annotated with the feature space identifier. These annotations are then copied to the output feature association graph. This information can then be used to steer the traversal of the association network.

While spreading the initial activation weight over the network, the selection of which nodes are taken for further traversal can be made upon the feature space annotation. Therefore the client may pass a set of feature spaces to the spreading activation method. This set has no impact on initial start node and also is not applied to constraint the final activated node. If the only termination criteria is the hop count - t^{hop} - the restriction on the feature space for middle nodes will be active only for certain hop counts:

$$0 < hopCount < t^{hop} \tag{119}$$

The middle node restriction is useful for example for data sets which consist of more than one feature spaces. Depending on the role mapping of these feature spaces for the input graph the retrieval of the final association network can then be constrained.

Stop Node Feature Spaces The stop node restriction can be seen as inverse of the middle node feature space constrained. Instead of specifying which feature spaces are valid for further activation, this stop node restriction defines which features should not be used for traversal. The difference between these two restriction is subtle. Nevertheless there are a couple of usage scenarios which require either one of the two methods.

The stop node restriction can be applied if the same feature association network is the base for multiple different use-cases. Is such a case, depending on the scenario a different traversal strategy might be necessary.

Termination Node Feature Spaces The last presented constrain based on the feature space is applies to the final nodes of a traversal trail. These nodes are especially important as they are taken as output of the retrieval operation. The feature association graph may consist of multiple feature spaces and the spreading activation may halt at any of these features. Therefore the traversal can be configured to stop as soon as certain

features are activated. The implementation offers the functionality to specify the feature space identifier of these features.

The SPREADINGACTIVATIONASSOCIATIONSEARCHER class does not only allow to control the traversal process. Equally important are the functions which calculate the activation weight. In the spreading activation scheme the weight is computed by two functions: The proportion of the activation weight which is passed by an node to all connected neighbouring nodes and the combination of these weights for a single target node. The first function can be further split up into a part which is responsible for the distribution of the weight and a part which implements the decay function.

Each of these parts may be customised by clients of the framework. The default behaviour already provides a basic implementation of the activation weight calculation. The output activation weight depends on the current activation weight and the weights of all connected nodes. Each out-going link is weighted by calculating the relative weight of a connection in combination with the current activation weight $w_i^{activation(n)}$, where n denotes the step count:

$$w_{i,j}^{out} = w_i^{activation(n)} \frac{w_{i,j}}{\sum_k w_{i,k}} \tag{120}$$

For each outgoing edge of all currently active nodes the out-weight is computed. Then the set of all active nodes of the next step are collected. For each of these feature nodes, the input weight is calculated by aggregating the respective output weights. The default implementation provides two different basic choices:

$$w_j^{in-max} = \max_k w_{k,j}^{out} \tag{121}$$

$$w_j^{in-sum} = \sum_k w_{k,j}^{out} \tag{122}$$

The decay function is dependent on the data-set and the size as well as complexity of the association network. There is no function which suites the majority of use-cases. Therefore the reference implementation simply returns the input value, which effectively means that the activation weight does not decrease with the number of hops.

$$f^{decay}(w) = w \tag{123}$$

The set of active feature nodes in step n is symbolised as F^n in the following formula. Assuming the w^{in-sum} has been chosen as input weight function, the final spreading activation function of the SPREADINGACTIVATIONASSOCIATIONSEARCHER class can be written as:

$$w_j^{activation(n+1)} = \sum_{\{k|f_k \in F^n\}} (w_k^{activation(n)} \frac{w_{k,j}}{\sum_l w_{k,l}}) \tag{124}$$

The spreading activation retrieval operation does return a set of features including their activation weights. The traversal and the termination of each thread are controlled by the input parameters. Some client do not only need the information which features have been ultimately activated, but they also require detailed information about the traversal process itself. To cater for such use-cases, the retrieval component of the reference implementation provides an additional interface. This interface has been labelled TRACER and allows a client to observe all parallel threads and their route through the association network.

The TRACER interface defines callback methods for each type of event of the traversal operation. The client is free to choose which events are

observed and how much data is recorded during the process. An implementation of this interface is passed to the SpreadingActivationAssociationSearcher class which then invokes the appropriate methods during the spreading activation process. This way the client can completely reconstruct the topology of the association network and the structural relationship of the feature associations.

The framework already provides an convenience implementation of the Tracer interface. This class stores all traversal events in an internal graph based data-structure. For complex association networks and high step counts the memory requirements may become prohibitively high. Therefore this class has a limited applicability. Its main purpose is to demonstrate the behaviour of the spreading activation retrieval process, for example for visualisations.

The SpreadingActivationAssociationSearcher class accepts a set of parameters to control many aspects of the traversal process. Due to the flexibility the performance is not as high as the retrieval of a single subgraph. Both approaches can be combined to gather the desired information from the association network.

Extended Functionality

The reference implementation serves as prove of concept for the feature association algorithms. All specified functionality is covered by the implementation. Furthermore additional functions have been developed which were needed by specific application scenarios.

Overview

The initial decisions made while developing the reference implementation of the feature association framework allowed flexibility for additional functionality. For example, the usage of an inverted index as main data-structure for input and output does allow a variety of methods to process the data set. All operations that may be executed on an inverted index are now open to be used in the context of feature associations.

Another set of additional functionality where added to the reference implementation out of necessity. While developing knowledge discovery applications which use the feature association framework new requirements did arise. Therefore the reference implementation has been enhanced to include these additional methods to improve its applicability.

Data Set Partitioning

The first presented additional functionality is a consequence of using an inverted index as main data-structure. The input feature graph as well as the output feature association graph are stored in a Lucene index. Not only does this data-structure provide all necessary properties needed for the graph storage, it also allows sophisticated retrieval operations, which has been the main use-case for this kind of data-structure.

By choosing the Lucene library as core for the graph storage, all Lucene query functions can now be invoked in the context of feature associations. This applies to the input graph as well as the output association network.

To demonstrate the basic principle a single use-case is presented in detail. The motivation for this use-case is to restrict the feature associations to a sub-set of all instance contained in the data-set. For example the association calculations should only consider instances annotated with a specific meta-data value. In many cases the restriction criteria might not be as simple. Usually multiple clauses are combined to build a single complex constraint.

As described in the previous section, a single instance is stored as single Lucene document. The features are transformed into tokens and each feature space is mapped as a Lucene field. The combination of documents and fields allows the Lucene library to provide a sophisticated faceted search functionality. In a faceted search scenario a user searches for documents while specifying a number of search terms which refer to different aspects of the documents. The actual content of these aspects depends on the data set.

The Lucene library defines a query language which is capable to build hierarchical combinations of sub-query. A sub-query can either be an restriction on a single facet or again a combination of sub-queries. The specification of the restrictions also provide a high level of flexibility[326].

The reference implementation of the feature association framework allows the client to specify a query for the input feature graph. This query is then passed to the Lucene library. The search hits for the query are collected. All instances included in the search result build a subset of the data-set. The calculation of the feature associations is then restricted to this subset.

Many real-world data sets consist of instances annotated with a rich set of additional meta-data. For example instances may carry the name of

[326] The complete query syntax of the Lucene library is available online: `http://lucene.apache.org/java/3_0_3/queryparsersyntax.html`

the person responsible for a specific instance. In a document management system each document is usually annotated with an author name. In order to restrict the data set to all documents belonging to a single author, a Lucene query needs to be specified to include only the appropriate authors. This query is then passed as an additional parameter to the feature association calculations. The final association network will be build based on the selection of documents from a single author.

Processing Cycles

The next presented additional functionality is similar to the query based restriction of the input data set. In this scheme only parts of the the data set are processed at once. But instead of being motivated to increase the flexibility, the processing cycles scheme is born out of necessity.

During the collect phase of the feature association algorithm, the so call local associations are computed. For each instance and each pair of features an entry in a data-structure is created. Entries are iteratively collected and grouped into blocks. The size of a block depends on the available main memory. As soon as a block has been completely filled with entries, it is stored on secondary storage[327]. The merge-sort algorithm is then responsible to reorganise the entries to be sorted according to the target features by reading out the stored block in parallel.

[327] Usually this will be the local hard disk. In an distributed environment the local associations are managed by the execution framework and stored in the distributed file system.

Although the local association data-structure has been carefully developed to be as efficient and compact as possible, the sheer number of entries causes high storage requirements. For large data-sets the number of blocks will be high and might even exceed the available storage space. Due to the way the merge-sort algorithm is executed, all blocks must be completely computed before any further processing might take place.

Of course this problem could be simply solved by adding additional storage space. Unfortunately this approach sometimes cannot be taken. For example if the feature associations are computed on a remote data processing centre and there is no way to increase the number of hard disks per host[328].

[328] Similarly, if the computation is done via cloud computing and the space either limited or far too expensive.

To overcome this burden, the reference implementation provides a parameter to control the maximum number of target features. If the total number of features contained in the data-set exceeds this number, the computation is split into multiple iterations. The implementation tries to keep the overhead of the multiple passes over the data-set as small as possible. For example the gather of the global statistics is only conducted once. The phases that need to be processed in each processing iteration are the collect phase, the sort phase and associate phase. In figure 26 the sequence of processing phases for multiple cycles is depicted.

At the beginning of each iteration a set of target feature is defined. During the the collect phase only local associations are computed, where the target feature is found in the predefined set of features. Thus fewer entries are generated per instance and the space requirements of the sort blocks is decreased.

As the data-set is processed in multiple passes, the complete feature association calculations will take longer than using only a single iteration. Therefore the maximum number of target features should only be specified if the storage requirement of the local associations exceeds the available space.

Auxiliary Classes

The last presented example for additional functionality of the reference implementation did not happen on purpose. While the other examples were motivated by either by necessities or flexibility, the final example has been

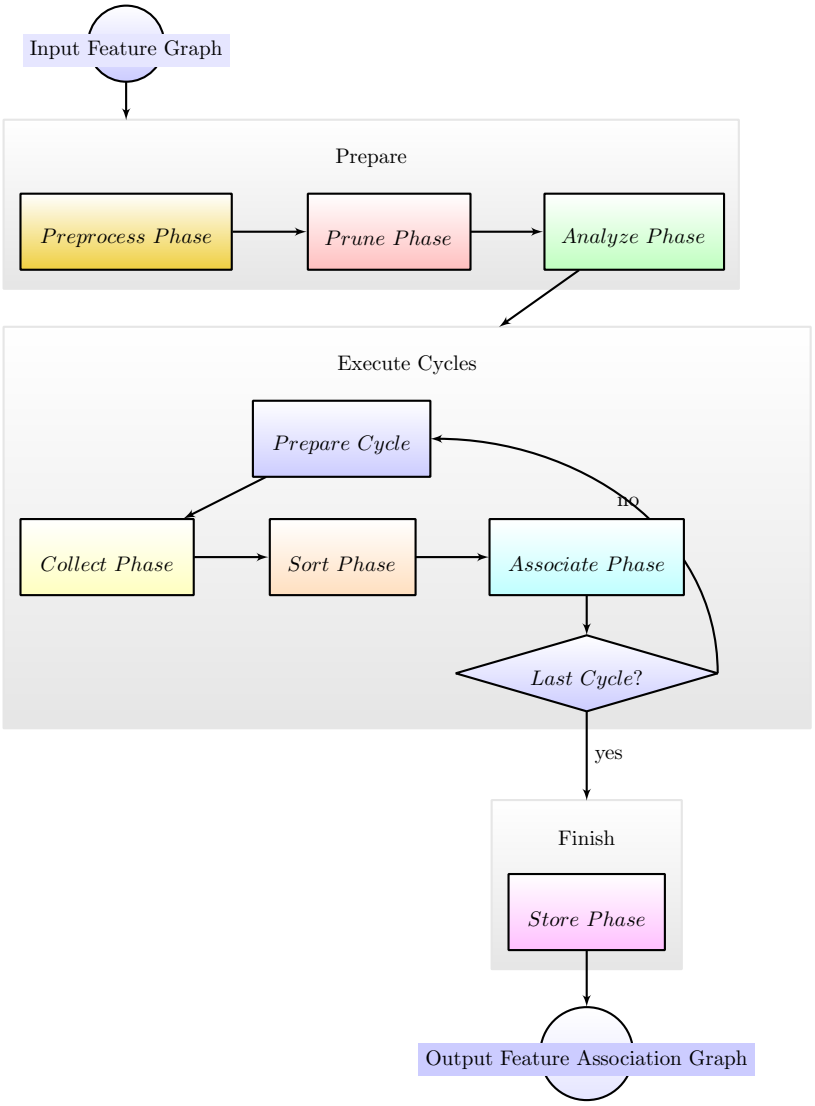

Figure 26: Overview of the phases for multiple passes over the data-set. The main stages of the algorithm are executed in cycles. In each cycle a distinct set of target features are processed. The global statistics and the cached data are shared between the cycles.

developed as a byproduct. To build the reference implementation classes were developed to represent the model of an data set abstraction. Finally a class hierarchy has been created which corresponds to many aspects of the input features and instances. Furthermore it has been necessary to develop structures and algorithms to process and manage the characteristics of the data.

The classes responsible for modelling the statistical properties of the data set are then been re-factored to be usable independently from the rest of the feature association framework. Therefore the functionality supplied by the auxiliary classes can now be used as standalone library. The reference implementation does employ this library for computing the necessary statistical properties and measures. The library itself does not depend on the reference implementation any more.

The functionality of the library is centred around features and their relationship in regard to instances. It can be applied in a number of use-cases. The main functions of the library of auxiliary classes are:

- One of the most important classes of the library models the properties of a feature. The complex nature of feature require an flexible implementation. A single feature consists of a number of aspects, some of the are optional. In table 27 the various attributes of a feature are summarised.

  Out of the attributes, the role of the weight of an feature diverts from the other attributes. The value of the weight depends on the context in with a feature is used. For example the weight might be the result of an statistical function or an similarity measure.

  The main objective of the implementation of this class has been to minimise the memory requirements. A data set may contain millions of features. The performance of the calculations depends on the number of feature which can be simultaneously held within main memory.

Attribute	Description
Identifier	The key for a feature. It must be unique at least within a specific feature space.
Feature Space Identifier	If the data set provides multiple feature spaces, this attribute references the space for the feature.
Weight	The weight of an feature. Depending on the context in which features are used, the weight may carry different semantics.
Meta-Data	A set of additional meta-data. The structure as well as the semantics entirely depend on the data-set and the application. Usually the meta-data is structured as key-value map.

Table 27: Overview of the attributes of a feature. With the exception of the 'identifier' all attribute are optional. Depending on the data set different attributes may be present.

- Features are usually organised is sets made up of multiple features. For example an instance can be seen as set of features. Therefore the library provides a dedicated class for managing a group of features.

  The first main purpose of the feature set class is to be able to apply functions not only on a single feature, but on all within a set. By using such

an batch approach, the efficiency of the computation can be increased. Furthermore the readability of the necessary code is also improved.

The second main use-case involves operations on more than one feature set. For example, a similarity measure can be used to compare two feature sets. If these two sets resemble instances, the resultant similarity value with represent the relatedness of two instances.

- The library is equipped with methods to compute the statistics of features in regard to a whole data-set. The statistics are then held in specialised data-structures. Therefore the library defines a set of interfaces and supplies a set of implementations of common statistics.

  The aim of the data-structures is to provide a flexible and efficient retrieval of the statistical properties. Furthermore they have been designed to allow the statistics to be externally stored. For large data-set the computation of the necessary measures may be time consuming. As long as the data-set is not modified, the statistics may remain valid. Thus the measure only need to be computed once and thereafter they can be re-used by importing them from the external storage.

  An example usage of this facility is to build so called language models. Language models are common in the domain of Information Retrieval, as well as Natural Language Processing. They capture the probabilities of terms given a large collection of textual documents. A term can be defined to be a single word, or a sequence of multiple words. For example a language model might be build based on bi-gram word features in order to detect common phrases.

The development of the reference implementation of the feature association framework has been conducted in close relationship with real-world application scenarios. Therefore it provides additional functionality to fulfil the these demands. Some of the additional methods were inherited from the employed libraries, for example the Lucene query facilities. Finally a dedicated library for dealing with features of large data-sets has been created while developing the feature association reference implementation.

Performance Evaluation

The reference implementation has been developed to conform to the specification of the feature association algorithm. Besides the functional aspects a number of requirements have been specified for the run-time behaviour of the calculations. Upper bound were given for the scalability and performance. In this section a series of real-world application scenarios are presented to illustrate the implementation's performance.

Overview

As stated in the description of the algorithm, the calculations of feature associations are of high computational complexity. The theoretical upper bound is $\mathcal{O}(mn^2)$, for m instances and n features. The basic computational complexity itself cannot be decreased in the general case. The only choice is to decrease the magnitude of the two parameters. To reduce the number of instances will most certainly be coupled with a loss of information. Removing features from the data set should be more effective. Features which appear to contribute little to the final result are candidates for removal. To detect those features is the task of pruning strategies.

Even if sophisticated methods are applied to keep the number of input variables as low as possible, the algorithm remains basically cubic. Therefore the main criteria of whether feature associations are feasible for a given application scenario depends on the efficiency of the implementation. Typically the computation resources are limited in both storage space as well as processing time.

Before the work on the reference implementation of the feature association framework started, a number of requirements have been defined. For different classes of data sets threshold on the maximum allows processing time were specified. They have been stated in the beginning of this chapter, see functional requirement 10, 11 and 12.

Furthermore the implementation should not exhibit an execution behaviour worse than the theoretical run-time complexity. In other words, it should scale with the data set, but not worse then expected. The scalability aspect of the implementation has been captured by functional requirements 7, 8 and 9.

The final aspect of the run-time behaviour is related to the storage demands of the reference implementation. Storage capacities of devices are steadily increasing. Nevertheless in many cases the available storage space is practically limit. Additionally the storage of retrieval of data is time consuming, especially if the data needs to be transferred via a network. Thus the reference implementation should be designed to utilise compact data-structures which can be efficiently be stored.

To test whether the requirements are met, a series of data sets should illustrate the typical run-time behaviour of the reference implementation. The presented numbers should only be seen as guides, as the actual execution times are greatly influenced by many external and internal factors. All data set have been processed using the same hardware and similar configurations. Some key aspect of the execution environment together with the used configuration are summarised in the following list:

- The speed and number of the processors determines the execution time of an application. The scalability of an algorithm is not affected by the processing power. The reported figures were achieved using a contemporary desktop computer[329].
- A salient characteristic of contemporary computer architectures is the

[329] The quad core CPU reports itself as: `Intel(R) Core(TM) i7 CPU 860 @ 2.80GHz`

difference in speed between the main memory and the secondary memory. The access times as well as the throughput of the main memory by far exceeds the read and write times of even the fastest storage devices. Therefore the size of the available main memory highly influences the execution speed. The computer used for all the calculations has been equipped with 8 Gigabytes of main memory. The feature association process has only been allowed to access a fraction of the available space. The small and medium sized data set were processed using only 1 Gigabyte of memory[330].

- The operating system also influences the run-time of an application. Especially the disk caching strategies may impact the execution time. The Linux operating system is generally regarded to be well suited for computational intense applications[331].
- As the reference implementation has been developed using then Java programming language it is executed within a virtual machine. Advances in the area of just-in-time compilers lead to performance improvements in the execution of Java applications. Therefore different virtual machines and different revisions may cause different execution times[332].
- The way the data sets have been pre-processed may have an influences on the number of instances and features. For example, different parsing algorithms may yield to different results due to different tokenisation strategies. Therefore the number of instances and features is reported for each data set. It has been tried to keep the pre-processing as light-weight as possible. Only open-source libraries which are publicly available have been selected for this task.
- Finally the configuration of the feature association framework itself highly impacts the processing time. In order to able to reproduce the results, the default values for the parameters have been used unless stated otherwise.

[330] The Java process has been started with the following parameter: `-Xmx1g`

[331] The installed version of Linux has been: `Linux kcpc-rkern 2.6.35-22-generic #35-Ubuntu SMP Sat Oct 16 20:45:36 UTC 2010 x86_64 GNU/Linux`

[332] The used Java implementation has been: `Java(TM) SE Runtime Environment (build 1.6.0_18-b07) Java HotSpot(TM) 64-Bit Server VM (build 16.0-b13, mixed mode)`

Brown Corpus

The Brown corpus has been one of the prime data sets for linguists for a long time. It consists of 500 textual document which are assigned to 15 different categories. Due to its size it is considered nowadays as an small data set. To come up with about as many instances as features, the documents were further split into sentences. The sentences were taken as instances and the individual words have been mapped as features. The main characteristics of the input feature graph generated out of the Brown corpus are given in table 28

Property	Value
Instance Nodes	57,350 Sentences
Feature Nodes	47,827 Unique Words
Edges	1,004,827 Occurrences

Table 28: Overview of the main properties of the input feature graph of the Brown corpus. Due to is low number of instances and features can be regarded as a small data set. There are a few thousand instances and features and about a million connections between them.

To evaluate the performance of the reference implementation, the feature graph of the Brown corpus has been processed. Because of the small size of the input graph, no pruning strategies have been applied. Therefore the number of feature node of the output association network is identical to the number of input features. In table 29

According to feature requirement 10 small data sets should be processed in less than an hour. This requirement is based on the assumption that the calculations are done on a desktop class computer. The performance of the reference implementation has been summarised in table 30[333]. The

[333] The data easily fits into the cache of the operation system and therefore the time to read and write data are minimal.

Property	Value
Features	47,827 Nodes
Feature Associations	5,291,332 Edges

Table 29: Key characteristics of the output feature association graph for the Brown corpus. Each feature is associated with about 100 other features.

reference implementation needed less than one minute and therefore fulfils the predefined requirements. On average a single feature associations took about ten microseconds to compute. Due to the size of the data set the performance figures can be seen as best case.

Measure	Quantity
Execution Time	47 Seconds
Input Size	37 Megabytes
Intermediate Data	130 Megabytes
Output Size	48 Megabytes

Table 30: Main performance numbers of the reference implementation when computing the feature associations of the Brown Corpus.

The total execution time can be further dissected and analysed. A software profiler has been used to take exact measurements of the individual invocation times during the processing. Based on this data the time consumed by each of the individual phases can be reconstructed.

In table 31 the phases of the algorithm are listed together with their relative proportion of their execution times. At first glance the relative times do not appear to be an important aspect to judge the quality and usefulness of an implementation. For bigger data sets it might be necessary to performed the computation in an distributed execution environment. Some of the phase may be executed in parallel while other should be processed on a single node to avoid excessive synchronisation overhead. Out the phases, the collect and associate phase have been specifically been designed to allow the computations to be distributed among separate machines. These two phases account for over 80% of the total execution time. The sorting of the local associations and the store phase take far less time. Depending on the implementation of the distributed execution environment, the sort phase may also be in part executed in parallel.

The association phase is active for about half of total execution time. During this phase the source features are associated with the appropriate target features. For each association a weight is calculated. The feature association function is responsible to compute the association strength. For the Brown Corpus the Pointwise Mutual Information has been taken as association measure. The computation of this function account for little less than 10% of the total execution duration[334].

[334] The PMI is a relatively simple formula and easy to compute. For more complex feature association functions, the weight calculations will play a more dominant role.

The performance evaluation based on the Brown corpus should give an insight into the behaviour of the reference implementation for small data sets. The computation took less than an minute. Closer inspection of the scalability aspects of the implementation reveals that the most critical parts of the computation will benefit the most from being distributed among multiple machines.

Reuters RCV-1 Corpus

The Reuters RCV-1 corpus has been selected to asses the performance and run-time behaviour of the reference implementation on data sets that can be regarded of medium size. This data set contains of 800,000 news articles, which were further split up into individual sentences. Each sentence has been mapped as instance nodes within the input feature graph. The words

Phase	Relative Time
Preprocess	0.4%
Prune	0.1%
Analyze	0.5%
Collect	32.5%
Sort	13.5%
Associate	48.5%
Store	4.4 %

Table 31: Overview of the relative execution times of the individual processing phases. The collection of the local associations and the calculation of the global associations account for over 80% of the total processing time.

has been taken as features nodes. Words occurring within a sentence were connected to the appropriate nodes.

An overview of the size of the input feature graph is given in table 32. The medium size data set contains about 200 times more instance nodes than the Brown corpus, while the number of feature did grow by the factor of 10.

Property	Value
Instance Nodes	12,488,979 Sentences
Feature Nodes	571,001 Terms
Edges	168,557,387 Tokens

Table 32: Overview of the main properties of the input feature graph of the Reuters RCV-1 corpus. By today's standards the Reuters corpus can be considered to be of medium size.

The output feature graph is considerable bigger for the Reuters RCV-1 data set in comparison to the Brown corpus base scenario. No pruning strategies have been applied and the number of feature nodes in the output matches the feature count of the input graph. About than ten times more feature associations have been identified for the medium size data set. The average number of associations per feature did grow as well in comparison to the first presented data set.

Property	Value
Features	571,001 Nodes
Feature Associations	70,936,143 Edges

Table 33: Key characteristics of the output feature association graph for the Brown corpus. Each feature is associated with about 100 other features.

The computation of the feature association of the Reuters RCV-1 corpus took 90 minutes. As with the Brown corpus evaluation a single desktop class machine has been used for the calculations. The duration of the computation is well below the time limit as specified in functional requirement 11. In table 34 the main characteristics of the computation of the Reuters RCV-1 corpus are given.

The medium size data set is similar to the small data set in many aspects. Sentences are used as instances and words as feature. But the numbers of nodes in the input feature graph is far higher for the Reuters RCV-1 data set. Still the reference implementation provides a satisfying performance and run-time behaviour.

Measure	Quantity
Execution Time	5,178 Seconds
Input Size	5.4 Gigabytes
Intermediate Data	23 Gigabytes
Output Size	635 Megabytes

Table 34: Performance numbers of the reference implementation when computing the feature associations of the Reuters RCV-1 corpus.

Wikipedia Corpus

The final data set of the performance evaluation is generated from the Wikipedia encyclopedia. This online resource is collaboratively generated and provides a wealth of information and continues to grow. The range of covered topics exceed the amount of information found in traditional printed encyclopedias. The length of the articles within this corpus tremendously vary in size.

For the final performance evaluation run the German version of the Wikipedia has been used. The German edition is considerably smaller than its English counterpart. It has been chosen because its smaller size allows the calculations of the feature association to be conducted on the same hardware as the first two data sets[335]. Thus the reported performance figures can be directly compared to gain an understanding of the scalability characteristics of the reference implementation of the feature association framework.

[335] The limiting factor is the required disk space, as the intermediate data for the English Wikipedia would exceed the available free space off the hardware used for the performance evaluations.

In contrast to the other two presented data sets, not sentences, but whole articles are mapped as instances. Therefore the number of instance node within the input feature graph is lower than for the Reuters RCV-1 data set. But the number of features is about one order of a magnitude higher. Additionally the number of connections between instance nodes and feature nodes twice as high. In table 35 an overview is given of the main characteristics of the input feature graph for the German Wikipedia data set.

Property	Value
Instance Nodes	1,403,298 Articles
Feature Nodes	5,102,921 Terms
Edges	388,384,564 Tokens

Table 35: Size of a input feature graph for a large data set. In this case the German Wikipedia serves as source for the data set.

Following the first to evaluation runs, no pruning strategy has been employed to reduce the number of features. But this time the number of local associations has been restricted. A sliding window approach has been integrated into the collection phase. The position of an feature within an instance determines the association candidates. Only feature within a window of 50 tokens are collected as local associations. The window is symmetric and the distance between features within the window is not integrated into the local association weight calculations.

The final output association graph for the German Wikipedia corpus contains as many nodes as there are features in the input graph. The total number of associations is roughly half a billion. In table 36 the exact numbers are listed.

Property	Value
Features	5,102,921 Nodes
Feature Associations	527,853,104 Edges

Table 36: Main characteristics of the output feature association graph for the German Wikipedia corpus.

The execution of the feature association calculation took less than 8 hours. The storage requirement of the resultant graph is similar to that the input graph. Due to the high number of local associations, the intermediate data-structure requires much more disk space. In table 37 an overview is given of the main characteristics of the calculations related to the run-time behaviour.

Measure	Quantity
Execution Time	27,170 Seconds
Input Size	6.1 Gigabytes
Intermediate Data	67 Gigabytes
Output Size	4.9 Gigabytes

Table 37: Results of the performance evaluation based on the German Wikipedia corpus.

Although there are corpora which exceed the size of the German Wikipedia, it is still considered to be larger than the most of common data set in many knowledge discovery applications. This corpus should give insight into the run-time behaviour of the reference implementation for larger data sets. The measured performance is satisfactory and well within the limits defined by functional requirements 9 and 12.

Results of the Performance Evaluations

Before the work on the reference implementation has begun, a series of requirements were specified. The requirements should give a guide on how the run-time behaviour depends on the size of the data set. The run-time behaviour is a blend of two different aspects: performance and scalability. The first one can be measured in time, the second aspect determines the number of resources needed for the computation.

Three types of data set have been identified. Small data set contain a few thousand entries. Typically this type of data set is generated out of a bigger data set. For example such a data set might represent a selection based on a specific users. In such scenario the feature association process will be repeated many times and therefore needs to be a quick as possible.

The Brown corpus has been taken as representative of a small data set. The feature associations were computed in less than a minute on commodity hardware. The execution times depend on the caching strategies of the operating system and other aspects.

The most common type probably can be classified as medium size data set. The characteristics of such a data set is similar to small data set, but the number of entries will be higher. Starting with this type of data set, the scalability aspect starts to be become more important. The data will not fit into main memory and therefore such a data set will be a good indicator of the quality of the implementation.

The Reuters RCV-1 corpus consists of little less than a million articles and can therefore be regarded as a typical example for a medium size data set. Although no pruning strategies have been applied, the overall performance has been satisfactory. The total run-time of the calculations took about one and a half hour on a desktop class machine. This can be seen as indicator that for most of the common data sets no dedicated computational resources will be necessary. Therefore the feature associations can now be computed for many application scenarios without additional administrative overhead, like buying expensive server-class machines.

As a final test of the scalability of the reference implementation a large data set has been taken as analysed. This class of data sets contain millions of instances and millions of features. Due to the enormous amount of necessary operations the processing last longer than for smaller input data

set. This evaluation should test the scalability aspects of the implementation. Therefore the question might be not how long the processing takes, but if it possible at all given the available computational resources.

The Wikipedia contains millions of pages and millions of distinct word, like for example person names and foreign words. Its size is orders of a magnitude larger than the common medium sized data sets. To put the performance figures of this evaluation in relation to the other runs, the same hardware has been used. Still the computations have been successfully conducted within the time-span given by the predefined requirements.

For even larger data set, a single machine will not be sufficient. The feature association algorithms has been designed with such a use-cases in mind. The reference implementation can be adapted to be executed within a distributed execution environment, for example the map-reduce framework.

The run-time behaviour and the performance of the implementation are considered as critical aspects of whether feature association are feasible. Judging by the performance of the reference implementation, the calculation of feature associations is now an option for many application scenarios.

Applications

The feature association framework has been developed to serve two purposes. It should support the task of feature engineering to gain insights into the latent relationships within data-sets. Furthermore it is designed to be easily integrated into knowledge discovery applications to extract information contained in the relationships. In this chapter both scenarios are illustrated by a series of use-cases.

Introduction

The performance evaluation revealed that feature associations can be efficiently calculated. Even large data-sets are now eligible to be analysed to uncover the relationships between the contained features. Now the question is, whether real-world applications will profit from building an association network.

To answer this question a series of application scenarios are presented in the following sections. Although the feature association framework has already been used for commercial projects, the focus now lies on supporting research in the area of knowledge discovery. This is a vast research field which touches many domains. Three of them have been selected as to represent typical areas of knowledge discovery applications:

Social Web The rise of user generated content is one of the main changes happened on the Internet in recent years. People now contribute more to existing web sites and systems. For example they provide product reviews, share their bookmarks and manage their social network. Many web platforms are suitable as sources for valuable information for knowledge discovery tasks.

Because of the dynamics of the social web and its rapid change, the analysis of the users behaviour is currently a hot research topic. Many existing algorithms have been adopted to study the user generated content. Furthermore the social web platforms make use of sophisticated methods to support their users.

Information Retrieval The area of information retrieval has a long tradition. To quickly find the relevant information of of a large amount of data has attracted a lot of researchers and still does. Due to the long history, many approaches has been proposed and information retrieval systems have been reached a mature level.

One of the main challenges in this field is to cope with huge collections of data. Only with the help of sophisticated algorithms and computing infrastructure the retrieval solutions are capable to deal with web-scale data-sets

Natural Language Processing Humans produce and understand natural languages without effort. Computers are still not able to reliably extract

semantics out of text. The task of the natural language processing is to improve the method to analyse unstructured textual content.

Researcher from different fields, for example linguists and computational scientist, work together to gain deeper insights into the structure of human languages. The main challenges in this area are to find and tune algorithms to analyse textual resources.

The motivation for building a framework for feature association is twofold. It should i) help in the process of feature engineering and ii) to support knowledge discovery applications. This categorisation can be further refined to build a scheme of possible modes of the feature association framework:

Analysis In this scenario, the goal of feature associations is to get a better understanding of the data and its intrinsic relationships. The output of the feature association calculations is analysed to detect pattern within the relationship network. For example to identify redundant features or detect mutually exclusive relationships.

Synthesis The feature associations processing can be integrated into an existing knowledge discovery application. In this scenario the feature association framework serves as an transformation operation. The output is then used for subsequent processing steps.

For example, a data set is transformed into an association network, which is then processed by machine learning techniques. This is mode of operation is the most common.

Retrieval The third possible scenario of using the feature association framework is rooted in practical considerations. Here the task is to retrieve information out of the network similar to an association graph.

In this application mode the framework is seen as an specialised data-structure accompanied by a set of generic operations. For data sets similar to an association network, the facilities of the framework can be utilised for processing.

The presented applications can be categorised according to the two axis, the domain and the main application mode. In table 38 an overview is given for the six application scenarios. All applications serve different means and employ different technologies. But they all have in common that the feature association framework is a vital part of the applications work-flow.

Application	Main Focus	Domain
Recommender System	Synthesis	Social Web
Tagging Structure	Analysis	Social Web
Query Expansion	Synthesis	Information Retrieval
Query Translation	Synthesis	Information Retrieval
Cross-language Plagiarism	Retrieval	Natural Language Processing
Word Sense Induction & Discrimination	Analysis, Synthesis & Retrieval	Natural Language Processing

Table 38: Categorisation of the presented application scenarios. There are three different modes of operation and three different domains.

Social Web

In the beginning of the World Wide Web only a few people could publish content. With the advances in technology and improved infrastructure now all user may write blogs, reviews or upload their photos. The research in domain of the social web is focused on two main areas: to get a better understanding of the users' behaviour and to improve existing social web platforms.

Overview

On the World Wide Web many platforms can be found where user may contribute content. User generated content is now regarded as important part of the information found on the Internet. Out of different types of social platforms, a number of groups can be identified:

- One of the first community platforms to appear on the Internet provided facilities to create, manage and share bookmarks. Typically a bookmark consists of an URL, a description and a number of tags. These tags can be freely chosen by the user and typically resemble keywords.

- The second category of community platforms are web-logs, or short blogs. Users post messages which are then displayed in chronological order. Messages can be assigned to categories and annotated with tags. Readers of such blogs use dedicated software to be informed as soon as a new message has been posted.

  Although blogs are mostly used to share personal opinions and experiences, there are a number of professional bloggers. Therefore blogging is sometimes considered as an alternative to traditional journalism. At the other side of the spectrum so called micro-blogging site allow user to post short messages. Typically these messages will contain little factual content, but are mainly used to share a thought or to post the current mood of an user.

- People may not only share personal stories, but also multi-media content. There are portals to upload and share photos or videos. Users may attach text to their uploads, as well as assign tags.

  Visitors to these sites are able to browse the multi-media content and leave comments. Typically such platforms also provide means to rate a photo or video.

- User may not only exchange messages with each other, but also manage their contacts online. Therefore various platforms provide tools to organise and browse the social network of users. The topology of such social network is of special interest to many research groups.

Countless other social applications can be found on the Internet. Most of them are designed to allow users to interact and to contribute content. As the new technology provides new functionality, people make use of the new opportunities.

One of the first conclusions one can drawn from the current state of the social web applications is that users tend to prefer systems that are easy to use. Systems that require the user to follow certain rules or strict work-flows have proven to be less successful. A typical example for this is the popularity of tags. Instead of imposing a classification scheme or a controlled vocabulary, tagging allows the user to use arbitrary content.

For larger tagging systems an interesting observation can be made. Tags from many users are not random or chaotic. Usage patterns and dialects

emerge from the collection of many taggers. Therefore tagging systems have been chosen to be base for the two application scenarios presented from the domain of the social web.

Recommender System

The first presented application is a tag recommender system for photos[336]. There is a common scheme of how people interact with photo sharing platforms. Users upload their pictures to the site. Then they type in a title and a description, or leave these fields blank. Next they add a couple of tags freely chosen by the user.

Typically tags describe the content of the picture. For example users select tags like "eiffel tower", "paris" or "france" for a photo of the Eiffel tower. Tags are also used to add meta-data not directly contained in the content. The date when the photo was shot is often used as tag or details of the settings when the picture was taken. Examples for this category of tags are "2011" and "macro". Tags may also refer to abstract concepts, like "nature", or subjective assessments or mood, for example "beautiful" or "sleepy". In figure 27 the most popular tags of a photo sharing platforms are visualised as a tag cloud.

[336] S. Lindstaedt, V. Pammer, R. Moerzinger, R. Kern, H. Mülner, and C. Wagner. Recommending tags for pictures based on text, visual content and user context. In *Proceedings of the Third International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 506–511. IEEE Computer Society Press, 2008

Figure 27: Tag cloud of the most popular tags at the photo sharing platform Flickr. Many tags refer to geographical locations and events. Although user are free to chose their tags, there is a surprising high overlap between users.

Related Work The task of a tag recommender system is to support the user in the process of adding tags to a recently uploaded photo. An algorithm should present the user suggestions which tags would be appropriate for a given picture. This is an interactive processes as with each tag the user selects, new recommendations should be made.

Taga are just one example for a possible application scenario of recommender systems. Especially for shopping platforms a good recommender engine is a vital part of the system[337]. Therefore recommender systems has been a active field of research for many years.

Generally speaking a recommender system consists of a set of user and items. For a given user a set of items is suggested. Three different main approaches have been identified:

[337] K. Wei, J. Huang, and S. Fu. A survey of e-commerce recommender systems. In *Service Systems and Service Management, 2007 International Conference on*, pages 1–5. IEEE, 2007

- The first approach is called **content filtering**. The properties of the items are compared to the preference of a user. In order for this scheme to work, the properties of the items need to be matched to a users profile.

  The profile needs to be either managed directly by the user or extracted by sophisticated algorithms. For example this approach has been successfully used to recommend articles[338]. In the case of a tag recommender systems for photo sharing sites, tags need to be found for a given image. The semantics of a tag needs to be matched to the semantics of a photo.

- The second method is referred to as **collaborative filtering**. In this scheme the relationship between users are exploited, whereas these

[338] R. Van Meteren and M. Van Someren. Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*. Citeseer, 2000

relationship need not be explicitly made. This method of recommendation has been successfully been used in many applications[339,340]. The collaborative filtering approach can be further split into two methods: User-based collaborative filtering and item-based collaborative filtering.

In the user-base recommendation scenario the current user is compared to other users of the system. A similarity between users is calculated to create a list of related users. In the context of tagging systems, the similarity could be computed based on the set of tags used by users. This approach has shown state-of-the-art performance, but also has a few shortcomings. For example it is hard to compute the similarity of a new users as there are no data available on with to base the similarity calculations, a situation known as the cold start problem.

The item-based collaborative filtering approach is probably the best known approach. Starting with an user and an item, similar items are searched. The similarity is defined by how often other two items are associated via users. In the case of a shopping platform this approach can be expressed as: "User who have bought item X also bought item Y".

[339] J. S. Breese, D. Heckerman, C. Kadie, and Others. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998

[340] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994

- The third approach for recommendation is not as widely known as the previously presented approaches and is sometimes seen as a specific type of content based recommenders. It is called **knowledge-based recommendation** and requires additional information about user and items. This external knowledge needs to be modelled into the recommendation process[341].

  In the case of tag recommendation for photographs, the meta-data stored alongside the picture can be exploited to suggests tags. Many camera models annotate pictures with various in-camera settings and store them within the image file. For example the focus distance can be interpreted and for close-up pictures the tag "macro" might be a candidate of a tag which might be chosen by users.

[341] R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems*, 69(Supplement 32):175–186, 2000

The list of approaches is not exhaustive and for many specialised use-cases, other methods are preferred. Furthermore many of the state-of-the-art recommender system combine multiple approaches[342].

[342] R. M. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix prize. *KorBell Team's Report to Netflix*, 2007

Figure 28: The main user interface of the Tagr system. The user is presented with a selection of similar images, users with shared preferences and a list of tag suggestions.

System Description The TAGR system[343] combines multiple approaches into a single unified user interface. The user interface provides facilities to upload a photo and recommends tags in an interactive manner. A screenshot of the Tagr application is shown in figure 28

Different recommendation strategies build the core of the Tagr system:

1. The image content is analysed, the colour distribution and texture are extracted. The gathered information is then classified into a predefined set of classes. For each of the classes, a set of training instances have been selected as input to a supervised machine learning algorithm. This tag recommendation method effectively implements a content filtering approach.

2. Starting with the current photo, other images are searched in the database. A similarity measure based on the image content is applied in order to visually find similar images. The top five images are then presented to the user. These images are typically annotated with tags, which now serve as suggestions for tagging the current photo.

3. The third component exploits the social network of the current user. The goal of this component is to find users which share the same preferences. For example if two users share the same home town or the same interest they are regarded as similar. Tags from similar users are then also integrated into the tag recommendation mechanism.

4. One can observe that people often start with a specific tag and then add more generic terms. An example of such behaviour has been given previously in this section. Starting with the specific tag "eifel tower" the more general geographical locations "paris" and "france" have been added. The fourth component of the tagr systems tries to mimic this behaviour.

   In order to determine more general terms for a given word, the WordNet[344] resource has been used. This corpus contains a network of words of the English language. The relationships within this graph reflect the semantical relationships of the connected words. In the WordNet corpus each word is connected to each synonyms. Words are also organised in a hierarchical manner. Starting from the root the word are more and more specific.

   The tag recommendation component takes the already assigned tags and searches the WordNet graph for hypernyms. These more general

[343] S. Lindstaedt, V. Pammer, R. Moerzinger, R. Kern, H. Mülner, and C. Wagner. Recommending tags for pictures based on text, visual content and user context. In *Proceedings of the Third International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 506–511. IEEE Computer Society Press, 2008

[344] C. Fellbaum. *WordNet: An electronic lexical database.* The MIT press, 1998

terms are then suggested to the user. This kind of tag recommendation falls into the category of knowledge-based filtering.

5. The final component of the Tagr system employs the feature association framework. This part of the system follows an item-based collaborative filtering approach. Tags are recommended based on the statistical distribution. If two tags are jointly used for many images, they are likely candidates to be used again in conjunction.

The results of the tag recommend approaches is combined and presented to the user. The user than is free to choose one of the tag suggestions or simply type in a new tag. In figure 29 the components of the Tagr system are depicted.

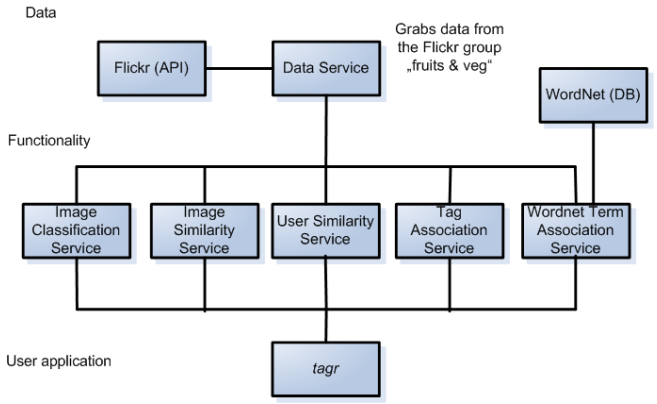

Figure 29: Overview of the main components of the Tagr system. The core of the systems is built by the different tag recommendation strategies. The output of these approaches is then presented to the user.

Feature Association Framework The feature association framework has been integrated into the Tagr application to calculate tag recommendations. In order to be able to make use of feature associations first the mapping has to be defined for features and instances. The feature association functions needs to be defined and finally the retrieval operation has be selected.

Intuitively the tags are mapped as features as one wishes to find related tags as suggestions. For the mapping of the instances there are two choices. Either could the users or the images be represented as instances. For the Tagr application the photos have been mapped as instance within the input feature graph. Within this graph all tags assigned to a single image share a common instance node.

The feature association function determines the relatedness of two tags based on their shared distribution. The cosine similarity has been proposed as a means to conduct an item-based collaborative recommendation[345]. In preliminary tests the cosine similarity did not provide satisfying performance for the test data-set. Therefore a similarity measure has been specifically designed for the use case at hand. Given two tags it puts the number of shared images in relation to the number of times the tags have been individually assigned. The set of images which carries a tag - t_s - labeled as I_s. The similarity function can be then written as:

[345] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004

$$S^{Tagr}(t_s, t_t) = \frac{|I_s \cap I_t|}{\frac{1}{2}(|I_s| + |I_t|)} \tag{125}$$

This weighting scheme is similar to the Jaccard coefficient, which has also been evaluated during the development of the system. Additionally the performance of other denominators, for example $max(|I_s|, |I_t|)$ or simply using $|I_s|$, has been assessed.

The output of the feature association calculations is a network of tags, where similar tags are connected to each other. In order to recommend tags

parts of the output graph needs to be retrieved. Starting with a single tag, all associated tags are collected an ordered according to their weights.

If the photo is already annotated with more than one tag, the associated tags need to be combined. For each of these tags the list of associated tags is retrieved. Then all lists are merged into a singe ordered list. If a tag is found in more than one list, the weights are combined:

$$w_{out} = \sum_{in} w_{in,out} \tag{126}$$

Finally the top associated tags are presented to the user as tag suggestions.

Results In the context of tag recommendation application the feature association framework plays the role of a feature transformation step. For a given list of input features, a set of output features is generated. To make use of the feature associations, tags are mapped as features and images as instances.

The main motivation to integrate the feature association framework into the Tagr application was to add a collaborative filtering based recommender strategy. A statistical approach should complement the existing methods, like content-based and knowledge-based filtering. The alternative to using the feature association framework is to develop a dedicated component from the ground up. This would require a considerable development effort. In contrast to this tedious job, the integration of the feature association framework just required moderate adaptation work. Other advantages of the usage of the feature association algorithm is outlined in table 39.

Property	**Feature Association Framework**	**Alternative Implementation**
Effort	Adaption work	Development from the ground up
Functionality	Inherit all existing functions	Need to implement each function
Flexibility	High flexibility	Requires effort

Table 39: Overview of the main advantages of using the feature association framework instead of a dedicated implementation.

Tagging Structure

Usually the feature association framework is integrated as a part of an existing knowledge discovery application. But it can be also invoked as a standalone tool, mainly to gain insights into the structure of features. In this section two scenarios are presented where the feature associations calculation help to detect patterns and collect characteristics of social tagging systems[346,347].

Social tagging applications have been one of the first widely accepted systems at the beginning of a movement which is now known as Web 2.0. User began to share they favourite bookmarks of websites with other users. These bookmarks are further annotated using tags. As it turned out in the collection of all annotations specific semantic structures emerge. Therefore the term folksonomy was born to reflect these structures which is generated by the collaborative way of tagging.

The bookmarking systems differ from the previously described platforms in regard to the number of users who may tag a single resource. Photos are usually only tagged by the people who created and uploaded them. In a social bookmarking system many users apply tags to the same resources, in this case bookmarks of websites. To differentiate between these two types of tagging systems, the terms broad folksonomy and narrow folksonomy have been introduced[348]. In the previous section a tag recommendation system for a narrow folksonomy has been described. Tn this section at first tags from a broad folksonomy are analysed and then a narrow folksonomy is used as base for an in-depth analysis.

Related Work The analysis of the tagging behaviour and the emerging relationships from folksonomies have been tackled from various sides. The most basic question one might ask is why user tag at all. A number of researchers tried to identify and then classify the different motivations for users to use social bookmarking services[349,350]. This stream of research let to the insight that different types of people also display different tagging behaviours[351,352,353].

Another approach is to try to simulate the users' behaviour by developing models which try to mimic the tagging process. The most common models are the polya urn model[354] and the variations of the Yule-Simon model[355,356]. These models predict a preferential attachment tagging process, such that tags that have been used in the past are likely to be used again. To validate a proposed model one needs to analyse the observed tagging behaviour of users of an existing system[357,358].

Furthermore, the usefulness of tags for navigation via automatically generated hierarchical relationships has been studied[359]. How this approach compares to traditional content based methods to aid hierarchical navigational[360,361] remains an open research question.

[346] R. Kern, M. Granitzer, and V. Pammer. Extending Folksonomies for Image Tagging. In *WIAMIS 2008 , Special Session on Multimedia Metadata Management & Retrieval*. IEEE Computer Society, 2008

[347] M. Lux, M. Granitzer, and R. Kern. Aspects of Broad Folksonomies. In *18th International Conference on Database and Expert Systems Applications DEXA 2007*, pages 283–287. Ieee, 2007

[348] T. V. Wal. Explaining and showing broad and narrow folksonomies, 2005

[349] C. Marlow, M. Naaman, D. M. Boyd, and M. Davis. HT06, tagging paper, taxonomy, Flickr, academic article, to read. In U. K. Wiil, P. J. Nürnberg, and J. Rubart, editors, *Proceedings of the seventeenth conference on Hypertext and hypermedia HYPERTEXT 06*, volume 27 of *HYPERTEXT '06*, pages 31–40. ACM, 2006

[350] M. Heckner, M. Heilemann, and C. Wolff. Personal information management vs. resource sharing: Towards a model of information behaviour in social tagging systems. In *IntâĂŹl AAAI Conference on Weblogs and Social Media (ICWSM)*, 2009

[351] M. Strohmaier, C. Körner, and R. Kern. Why do Users Tag? Detecting UsersâĂŹ Motivation for Tagging in Social Tagging Systems. In *International AAAI Conference on Weblogs and Social Media (ICWSM2010)*, number Coates 2005, 2010

[352] R. Kern, C. Körner, and M. Strohmaier. Exploring the Influence of Tagging Motivation on Tagging Behavior. In *Research and Advanced Technology for Digital Libraries*, pages 461–465, 2010

[353] C. Körner, R. Kern, H.-P. Grahsl, and M. Strohmaier. Of categorizers and describers: An evaluation of quantitative measures for tagging motivation. In *HT '10: Proceedings of the 21st ACM Conference on Hypertext and Hypermedia*, pages 157–166, 2010

[354] S. Golder and B. A. Huberman. The structure of collaborative tagging systems. *Arxiv preprint cs/0508082*, 2005

[355] C. Cattuto, A. Baldassarri, V. D. P. Servedio, and V. Loreto. Vocabulary growth in collaborative tagging systems. *arXiv*, 704, 2007

[356] C. Cattuto, V. Loreto, and L. Pietronero. Semiotic dynamics and collaborative tagging. *Proceedings of the National Academy of Sciences*, 104(5):1461, 2007

[357] K. Dellschaft and S. Staab. Understanding the Dynamics in Tagging Systems. *Proceedings of the European Future ...*, 2009

Broad Folksonomies The first presented analysis of a folksonomy using the feature association framework did target a so called broad folksonomy. Not only a single user may tag a resource, but multiple users may add tags to a single resource. The goal of the analysis is to gain a better understanding of the tagging process by studying the distribution of tags in relation to users and resources.

Previous work[362] has indicated that the frequency of tags follows a power law distribution. This type of distribution is found in many natural and man-made phenomena, for instance the size of cities[363]. If the collaboratively generated folksonomies would also exhibit this property, this would allow to re-use existing methods from other domains. For example, many techniques in the field of information retrieval exploit the power-law distribution of words within text. Given that tags follow the same distribution as word within text corpora do, the search and retrieval methods may be applied on tagging data.

[362] C. Cattuto, V. Loreto, and L. Pietronero. Semiotic dynamics and collaborative tagging. *Proceedings of the National Academy of Sciences*, 104(5):1461, 2007

[363] G. K. Zipf. *Human Behaviour and the Principle of Least-Effort.* Addison-Wesley, 1949

The process to assess whether the distribution of tags follows the power law is made up of two steps:

1. At first the parameters of the distribution need to be estimated. The formal definition of the power law combines the coefficient α and the exponent β:

$$y = \alpha x^{\beta} \tag{127}$$

   In order to estimate the two parameters, we employed the linear least squares method (LLS). Therefore a logarithm is applied to the power law relation:

$$\log(y) = \log(\alpha) + \beta \log(x) \tag{128}$$

2. Once the parameters have been estimated a statistical test is conducted to asses whether the observed data matches the expected distribution. For this purpose we employed the well known χ^2 test to calculate the significance level. The chi square test is defined as:

$$\chi^2 = \sum_{i \in Bins} \frac{(O_i - E_i)^2}{E_i} \tag{129}$$

   In order to apply the chi square test, the data need to be grouped into bins. This is rooted in the requirement of the significance test that there need to be at least 5 samples per bin. Other approaches to estimate the goodness-of-fit have been discussed by Goldstein et al[364].

[364] M. L. Goldstein, S. A. Morris, and G. G. Yen. Problems with fitting to the power-law distribution. *The European Physical Journal B-Condensed Matter and Complex Systems*, 41(2):255–258, 2004

In order to analyse the tagging behaviour, at first a data-set needs to be acquired. The social bookmarking site Del.icio.us[365] has been selected as base for the analysis. A crawler has been developed which has been used to periodically collect any recent activity. Resources which were bookmarked only by a single user have been ignored in the crawling process. The complete data set finally contains over three million bookmarks. For the analysis a subset of all bookmarks has been created which covers a specific time-span. In table 40 an overview is given for the full data-set and the sub-set.

[365] `http://del.icio.us`

Based on the data set and the method to detect pow-law distributions a series of analysis have been conducted to answer a set of questions:

- Does the distribution of co-tags follow a power law?
- Does the number of users for specific resources follow the power-law?

	Full Data Set	Time-Based Sub-Set
Bookmarks	3,234,956	838,804
Tag-Assignments	9,241,878	2,408,935
Tags	365,838	135,473
Users	84,121	26,919

Table 40: Size of the del.icio.us data sets used to analyse the tag distribution.

- Does the relation between bookmarked resources and users follow a power-law?

The first analysis is motivated by the proposed Yule-Simon model with time dependent memory by Cattuto et al[366]. They showed for a few tags that the co-occurring tags follow a power-law distribution. Furthermore the selected tags have in common that they been used by many users. Therefore the question arises whether the power-law assumption is also valid for a whole folksonomy including less frequent tags.

[366] C. Cattuto, V. Loreto, and L. Pietronero. Semiotic dynamics and collaborative tagging. *Proceedings of the National Academy of Sciences*, 104(5):1461, 2007

The feature association framework has been employed to conduct the necessary statistical calculations. For each tag the set of co-occurring tags has been collected and counted. The frequency counts have then be taken to build a histogram for each tag to asses its co-tag distribution. Out of the set of 135,473 tags, about 80% have been identified to show a power-law distribution. This analysis fosters previous observations. Furthermore deeper analysis reveals that for 90% of all cases, the exponent β falls into the range of $[-1.5, -0.5]$[367]. For the co-occurring tags one can conclude that the majority of tags follow the power-law distribution with similar parameters.

[367] A value of β close to zero indicates a more even distribution (shallow curve).

To answer the next two questions we split the data-set into two sub-sets. The first sub-set contains tags used by more than 30 users, while the second sub-set contains tags used by less than 30 users. The data-set which contains only the most popular tags has a size of 15,835 (the remaining 341 thousand tags are found in the second data-set).

We restricted our analysis of the relation between the users and the resources to the first data-set. For each tag within the data-set we ranked the resources according to the number of users, who have bookmarked a site. Next we calculated a ranking of the users based on how many resources have been tagged by each tag. Thus for each tag we generated two histograms, one based on resources and the other one based on users. On each of these histograms we applied our power-law distribution assessment.

Our analysis shows that for 18.4% of all popular tags the frequency of the resources exhibit a pow-law distribution (using a threshold of 99% for the statistical significance test). The value of β mostly falls into the range $[-0.5, -0.1]$. For only 13% of all tags the histograms of the users reflect a power-law distribution. The same statistical significance threshold as for the resource analysis has been applied. The gradient parameters of the user histograms are found in the similar range as the β parameter of the resources.

When relying on the overall statistics only we were not able to detect any correlations between the resource and the user histograms. For the set of tags where both histograms follow a power-law distribution we made an more detailed analysis. This is the case for about 5.6% of the most popular tags. We found these tags to be specifically descriptive and of high quality, for example *RFC*, *Technorati*, *X86*.

Additionally we conducted an analysis of those tags which do not yield resource and users histograms with a non power-law distribution. Out of all tags a percentage of about 53% are only used once by a single user for one resource. Many of these tags are the result of misspellings or a highly

specific vocabulary. Tags that are only used by a single person, but on multiple resources, account for 3.95% of the most popular tags and 19% of the remaining tags. Finally about 12% of all popular tags and 38.7% of all remaining tags have been used by multiple users, but only a single time.

Based on the results of our analysis the three questions, which have been stated at the beginning of this section can be answered. For the majority of all tags, co-tags do follow a power law distribution. This cannot be said about the relation between the users and resources. But, if a tag yields its resource and users histograms to follow a power-law distribution, it is likely to be a high quality tag suitable for retrieval or tag recommendation.

Narrow Folksonomies As previous research has show, most of the tags within a folksonomy follow a power-law distribution. This is also been found to be the case for words used within textual documents. Many algorithms have been developed to exploit this property, for example to compute the similarity of two texts. Therefore the question arises how tags and words do relate, as this would allow to apply text based algorithm on tagging data. Do tags and other types of meta-data cover the same information or do the user generated tags differ from other data found within a folksonomy? This question is the starting point of an detailed analysis of an existing, real-world folksonomy. The feature association framework has thereby been used as a tool to get a better insight into the data.

The base for this analysis has been a so called narrow folksonomy, where only a single user may add tags to a single resource. We selected the online photo-sharing platform *Flickr* as source for our data set[368]. A focused web crawler has been developed to gather pictures from the group *fruits & vegs*[369]. Finally the data set for our analysis consisted of more than ten thousand images. In table 41 the main characteristics of the data-set are listed.

[368] Although this platforms allows user to tag photos of other users, this feature is hardly ever used.

[369] *Flickr* allows users to add their photos to groups. The fruits & vegs group contains photographs somehow related to fruits and vegetables.

Table 41: Size of the Flickr data sets used to compare tags with other meta-data provided by users.

	Amount
Photos	13,651
Users	4,336
Avg. Photos per User	3.14

Additionally to the name of the users and the photos, a number of other data has been gathered. Finally our data set contained six different types of data and meta-data:

Photo The actual data of the photo together with an unique id. In the calculations we only used the id of the photo, the actual content of the photo is only used when displaying the results.

User The data set contains more then 4000 users. A single user is represented as a pair of an identifier and a name. The names of these users are not used for the computations, they are used when visualising the association network.

Tag All tags that have been used to annotate the photos were collected. As the Flicks folksonomy represents a narrow folksonomy, the user who added the tags is identical to the user who uploaded the photo.

Title When uploading a photo the user is asked to provide a title. Usually the title will contain a short summary of what is depicted in the picture.

Description Additionally to the title, the use may also provide a description of the image. Not all photos carry a description, but if present, it will usually contain more text than the title.

Comments In contrast to the title and description, the comments do not originate from the same user who has uploaded the photo. Other users of the *Flickr* platform may add their personal comments to other peoples images. These comments are expected to be more noisy in comparison with the title and descriptions.

The title, description and comments meta-data differ from the other meta-data, as they represent textual content. Therefore the content of this meta-data fields are first preprocessed. The textual content is split into individual tokens. The tokenised words are not further processed, no stemming or stop-word filtering has been conducted.

A folksonomy is formally defined as tri-partite graph, consisting of users, resources and tags[370]. In the case of the Flickr folksonomy, the resources are photos. In this graph structure, users are connected to photos and photos are connected to tags. In contrast to the traditional definition of a folksonomy, our data-set contains additional information. To find out how textual information and tag relate, we developed an evaluation scenario using multiple representations of our data set.

[370] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag Recommendations in Folksonomies. *Knowledge Discovery in Databases PKDD 2007*, 4702(May):506–514, 2007

The base of our analysis is the output of the feature association framework applied on the crawled data. The raw data from the *Flickr* crawl was transformed into a graph-structure, which is suitable to compute the associations. Depending on the desired output graph, the input graph has been restricted to contain just those nodes and relations that are needed to construct the necessary feature associations. The cosine similarity has been chosen as feature association function. The association graph contains relations with a weight close to 1 for nodes that share a similar distribution. For nodes that differ in their distribution, the weight will be close to zero. For example, two tags that were exclusively used together will be connected via an edge with an association weight of one. As the cosine similarity is symmetric, the feature association network is an undirected graph.

In order to analyse the difference between tags and other kinds of meta-data we build three feature association graphs:

G^t The first graph represents a traditional folksonomy graph. There are three types of nodes within the graph: the users, the photos and the tags.

G^e This graph is build using all the available information. Additionally to the users, photos and tag nodes this graph also contains node types for title terms, description terms and comment terms. Thus this graph is the most complete.

G^p The last feature association graph consists of all nodes of the G^e with the exception of the tag nodes. This graph simulates the situation before tags were introduced to annotate resources.

Thus G^e graph contains nodes build from all available data and meta-data types together with their relationships. An overview of all relations within each of the three graph is given in table 42. The "Total Number" column lists the total number of associations between two nodes. The arithmetic mean of all weight of a specific relation type is given in the column "Avg. Weight". In column "Avg. Weight Top-10k" the average weight of the top the thousand relations, when ordered by weight. The last column indicates which of of the three graph contains the relation type.

Judging by the average weight for the top ten thousand associations there are obvious differences in the association weights. The relation between

Relation	Total Number	Avg. Weight	Avg. Weight Top-10k	Graph
Tag ↔ Photo	99,911	0.08	0.2	G^t, G^e
Photo ↔ User	52,258	0.13	0.3	G^t, G^e, G^p
User ↔ Comment	398,023	0.09	0.84	G^e, G^p
Comment ↔ Photo	346,535	0.05	0.34	G^e, G^p
Photo ↔ Title	27,882	0.16	0.33	G^e, G^p
Photo ↔ Description	77,262	0.1	0.34	G^e, G^p

Table 42: Overview of relations within the association network of the three graphs.

users and comment terms is by far the highest. The connection between comment terms and photos is much lower. This indicates that there many terms within comments only by single users, as if users have their own vocabulary for commenting photos. The relationships between tags and photos appear to be the weakest. This stirs the question whether tag are suited for retrieval or navigation of photos. The evaluation section will address this issue using a more detailed analysis of the properties of the folksonomy graph.

For the evaluation of the association networks we used a retrieval method based on a spreading activation scheme. The spreading activation technique is widely used to query graph based structures[371]. Starting with a single node and an initial activation weight the graph is iteratively traversed. For our analysis we used 1 as initial activation weight. Starting with a single node all outbound edges are navigated and the target nodes are marked as activated. Their activation weight is the product of the activation weight of the start node and the edge weight. The set of activated nodes serve as start nodes in the next iteration. If a node gets activated via multiple edges, the highest activation weight is taken. Because of performance considerations, the set of start nodes is limited to 1024[372].

In figure 30 a screen-shot is shown of a visualisation of the spreading activation process. In this example the G^e graph has been traversed, starting with the user node "pizzodisevo". A total of 5 iterations have been computed. The nodes with the highest activation weight are selected for the visualisation. Additionally the path containing the strongest activation from the start node to the target nodes is shown. In this example it can be seen that some tag nodes are reached via title and description nodes. For example the tag "courgette"[373] is activated by traversing the node for the title term "falling". This node has been activated by the photo labelled with the title containing the term "nocciola"[374].

The main goal of our analysis has been to investigate how the additional meta-data differs from the traditional folksonomy data-structure. Additionally we tried to evaluation how the additional meta-data would be helpful for a recommender system. We split the evaluation into three separate task. At first we assessed the overlap between the traditional folksonomy with the association network built using all available information. Next we tried to find out the importance of single relation typed by following a leave-one-out approach. Finally we computed the predictive quality of the relation types to give estimates which then can by used when building a tag recommender system.

Overlap Evaluation In the first evaluation the overlap between the traditional folksonomy graph G^t and the extended graph G^e is computed. Additionally the pruned graph G^p should give further insight into the information contained in the user supplied tags.

In this evaluation we restricted the set of initial starting nodes just to

[371] F. Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6):453–482, 1997

[372] Due to the sparseness of the association graphs, this limit is only infrequently reached.

[373] French word for zucchini.

[374] Italian term for hazelnut

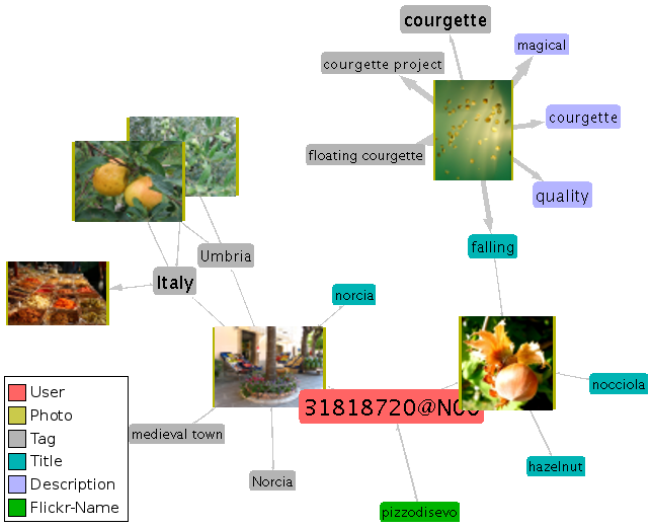

Figure 30: Screen-shot of a visualisation for a spreading activation within the G^e, starting with the user node "pizzodisevo". From there various path through the association network are taken. The nodes with the highest activation weight are displayed.

nodes that represent users and photos. For each instance of these node types we started the spreading activation process. The traversal of the association network stops when reaching a node of the same type as the initial start node. The result of a single activation run is then a set of target nodes.

The traditional folksonomy graph G^t serves as base for the comparison against G^e and G^p. The overlap between two results sets is calculated by dividing the number of common nodes by the sum of all distinct nodes of both sets. To complement the overlap figure, we also calculated the recall, which is the number of common nodes in relation to the size of the result set generated for the G^o graph.

Graph	Node Type	Overlap	Recall
G^e	Photo	79.8%	100%
G^e	User	15.0%	67.5%
G^p	Photo	6.0%	11.5%
G^p	User	3.8%	24.0%

Table 43: Result of the overlap evaluation for the *Flickr* folksonomy. The extended graph and the pruned graph are compared against the traditional folksonomy.

In table 43 the result of the overlap evaluation are listed. The first obvious difference is that the overlap values for the extended graph are higher than for the pruned graph. One can therefore infer that the "Tag $\leftrightarrow$ Photo" relation plays an important role to associate photos, as well as users.

The high overlap and recall between G^e and G^t for the photo node type corroborates that the relation between photos and tags carries a high information content. The additional nodes for description and title term do not dramatically change the results of the activated photo nodes. Between the node types for users and photos there is huge difference in terms of overlap. This can be in part attributed to the comments nodes, which allow the discovery of many additional user nodes.

The comparison between G^p and G^t shows that there is low overlap and a relatively low recall. This again stresses the importance of the tag

information, as the extended and pruned graph vary only in terms of the "Tag ↔ Photo" relation.

Leave-one-out Evaluation The second evaluation tries to measure the importance of individual relation types. For this evaluation the extended association network - G^e - is used.

For all relation types the top 10,000 relations have been selected, based on their association weight. For each of these relations we run two spreading activation traversals, starting at each nodes that are connected via this relation. Thus if a relation connects two nodes - n_1 and n_2 - at first an traversal is conducted starting a n_1 and the second starts at n_2. In this spreading activation process the relation itself is excluded from the graph traversal. Thus this analysis measures the necessary detour needed once a relation is removed from the graph.

Using this approach a number of statistics can be calculated to asses the importance of the individual relation types. The first is the average activation weight of the node at the other end of the relation - *AAD*. If the traversal started at n_1, the association weight of n_2 is recorded. As the initial activation weight is 1, the activation weight of n_2 is expected to fall between 0 and 1. A high value indicates that the relation is mostly redundant and does only contribute little information content to the association graph.

The next statistic is the sum off all weights along the shortest path until n_2 is reached, averaged over all relations of a specific type - *AW*. This value will be higher than *AAD*. The difference between *AW* and *AAD* should give insights how well the nodes alongside the alternative path are connected. A large gap between these two statistic would indicate the the relations cannot be easily reconstructed using the remaining association graph.

The final statistic is the average length of the alternative path - *APL*. The values for this statistic will be mainly influenced by the structure of the association network. This measure should give a baseline for the information that is represented by the left out relations. The more the *APL* of a relation type deviates from the theoretical minimum, the more relations exists that cannot be recovered once removed.

In table 44 the result of the leave-one-out evaluation are presented. For each of the relation types within G^e the average of the main characteristics are given. Additionally for the *APL* measure, the theoretical minimum is given in brackets.

Relation Type	AAD	AW	APL
Comment ↔ Photo	0.13	0.37	2.00 (2)
Photo ↔ User	0.08	0.31	2.16 (2)
User ↔ Comment	0.06	0.24	2.00 (2)
Photo ↔ Title	0.01	0.22	3.16 (3)
Photo ↔ Description	0.001	0.16	3.09 (3)
Tag ↔ Photo	0.001	0.14	3.04 (3)

Table 44: Result of the leave-one-out evaluation for the Flickr folksonomy. The main characteristics of the path once a relation has been removed from the graph.

From the results of the leave-one-out evaluation one can see that the relation between comment terms and photos carries the least information. As already found in previous analysis, the comment terms correlate with users. The relation between users and photos appears to be at least in

parts redundant as this relation type scores the second highest values for *AAD* and *AW*. Again the strong connection between comment terms and user can be attributed to this findings. The *APL* value shows that are a considerable amount of user/photo combinations which are only found via longer association chains.

The connection between a user and a comment term is reached within a minimum number of hops for the majority of cases, but its average activation it relatively low. This can be interpreted that users who tend to comment on the same set of photos do not agree on a common vocabulary. Finally the connections between a photo and the other types of meta-data (title, description and tags) appear to carry unique information. Especially the relationship between tags and photos delivers the lowest values of the average activation weights, although its *APL* value is close to the theoretical limit. To summarise, on average the vocabulary of the title, description and tags meta-data appear to differ according to their similarity in regard to the photos.

Prediction Evaluation The previous evaluation run tried to gather insights into the relation between different relation types. The last evaluation focuses on the relation between nodes of the same type. Its aim is to provide a baseline when building a recommendation engine by measuring the predictive quality of individual node types.

To accomplish this task the data set has been split into two parts, a training set and a test set. The training set has been generated by randomly choosing 60% of all photos, the remaining photos are used for testing. Out of the training set all available data has been used to compute a G^e feature association network.

The test set has been then used to assess the predictive quality by traversing the feature association network. All photos within the test set are evaluated by collecting all associated meta-data. For each of the meta-data types all values are iterated and individually assessed. All other values of the meta-data type are taken as input to the associated retrieval. Finally the result of the traversal is compared to the start value.

For example the photo with the id "10547374" carries the tags "2002", "tomato" and "nature". For the first of the three round the tag "2002" selected as start value and the remaining tags "tomato" and "nature" are used to build the start nodes for the spreading activation. From this traversal all tag nodes are collected and their activation weight serve as sorting criteria. The weight is used to generated an ordered result set of tags. The position of the tag "2002" within the ranked list is recorded and the test is repeated for the other two tags. Not only tag relations are tested this way, but all other meta-data types as well. Although they will probably be never been used directly in a real-world recommender system, they still should give a better understanding for the available data.

The overall results of the prediction evaluation is summarised in figure 31. On average the correct tag can be found in the first 10% of the result set for about 40% of all tests. This can be seen as a baseline performance for building a tag recommender system.

Other meta-data types did perform similar to the tag prediction. Especially good performance can be observed for users. This indicates that the combination of all available meta-data does help to predict which users will be interested in photos given their past behaviour. At the other side of the scale, the title terms yielded the lowest results. This hints that the combination of term within a single title is more diverse then for example the words within comments.

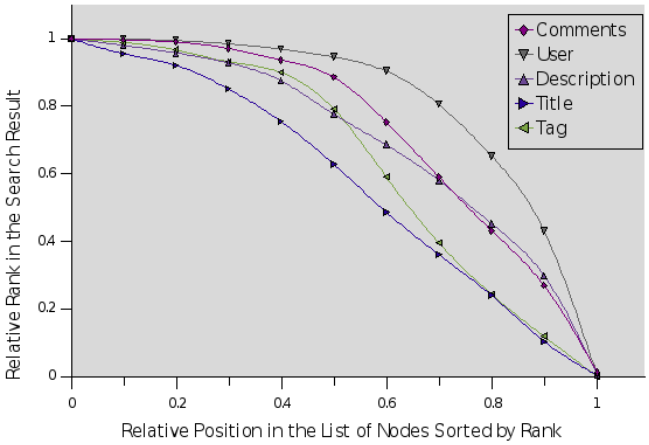

Figure 31: Results of the prediction evaluation that simulates a recommender system. On average the desired node can be found in the first 10% of the result set for about 40% of all nodes (with the exception of the *Photo* ↔ *Description* relation)

Results This section covered the analysis of tagging data taken from real-world folksonomies. The feature association framework has been utilised to gather statistics about the internal relationships found in the tagging data. Sophisticated retrieval models allowed a detailed analysis to gain insights into the usage of this kind of social web applications.

The presented application scenarios highlight the suitability of the feature association framework to be used for feature analysis. The rich dataset of the Flickr data set demonstrates that feature associations do not need to be limited to bi-partite graph structures. In the next section the feature association framework will be mainly be featured as a way to transform existing features and to synthesise new features.

Information Retrieval

NOWADAYS PEOPLE ARE USED TO FOLLOW A SEARCH BASE STRATEGY TO INTERACT WITH KNOWLEDGE BASES. MANY TECHNOLOGIES HAVE BEEN DEVELOPED TO ENABLE AND IMPROVE THIS INTERACTION. THEREFORE THE CHALLENGES IN FIELD OF INFORMATION RETRIEVAL HAVE BEEN TACKLED BY MANY RESEARCHERS AND ENGINEERS ALIKE. TODAY MANY INFORMATION RETRIEVAL METHODS AND ALGORITHMS HAVE REACHED A MATURE LEVEL.

Overview

Most of the underlying technologies of contemporary search engines have been proven to work reliably and provide sufficient performance. Therefore in more recent times the research in the area of information retrieval has begun to tackle developing solutions for specific information needs. Furthermore another stream of research tries to improve and optimise existing technologies.

In this section a the feature association framework is featured as a part of an information retrieval application[375,376,377,378]. Within this application association network are use to serve two separate purposes, query translation and query expansion.

[375] R. Kern, A. Juffinger, and M. Granitzer. Evaluation of Axiomatic Approaches to Crosslanguage Retrieval. In *Multilingual Information Access Evaluation Vol. I Text Retrieval Experiments*, 2009

[376] R. Kern, A. Juffinger, and M. Granitzer. Application of Axiomatic Approaches to Crosslanguage Retrieval. In *CLEF 2009 Workshop*, pages 142–149, 2009

[377] A. Juffinger, R. Kern, and M. Granitzer. Crosslanguage Retrieval based on Wikipedia Statistics. In *Proceedings of 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, 17-19 September, Aarhus, Denmark*, 2008

[378] A. Juffinger, R. Kern, and M. Granitzer. Exploiting Cooccurrence on Corpus and Document Level for Fair Crosslanguage Retrieval. In *Working Notes for the CLEF 2008 Workshop, 17-19 September, Aarhus, Denmark*, 2008

Query Translation Associations between features have been utilised to translate individual query terms from a source language into a target language. This task is can be seen a special case of machine translation. Machine translation in general usually deals with whole documents to translate sentence between different languages. The output of such a system are grammatically correct sentences. For the task of query translation, the grammar of a language does not play an important part. Queries submitted by user to a search engine often consist of just a few keywords[379]. We have developed a system based on the Wikipedia online encyclopedia to find corresponding query terms for a cross-lingual information retrieval system.

[379] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *ACM SIGIR Forum*, 33(1):6–12, 1999

Query Expansion As the queries generated by the users tend to be short and partly omit necessary keywords, strategies have been developed to cope with this challenge. The focus of these activities has been the processing of the query string. After an user has issued a query, it is analysed and modified before it is sent to the search engine. Additionally to the query terms entered by the user, more terms are added to the query. This technique is known as query expansion. Usually these terms are semantically related to the original query terms, for example synonyms. The feature association framework has been adapted to generate those semantically related query terms.

Robust WSD @ CLEF The retrieval application itself has initially been developed for the CLEF[380] 2008 campaign and then refined and in parts rewritten for the CLEF 2009 campaign. The aim of the CLEF campaigns has been to evaluate the quality and progress in the field related to cross-language information retrieval. It consisted of multiple tracks, where one has been dedicated to assess the impact of word sense disambiguation on cross-lingual retrieval: Robust WSD Task[381]. This task is motivated by the intuition that knowing the correct sense of an ambiguous word could improve the performance of an information retrieval application. It consisted of two separate evaluations, one for a mono-lingual task additionally a cross-lingual task.

[380] `http://clef-campaign.org/`

[381] `http://ixa2.si.ehu.es/clirwsd/`

Evaluation Setting The organisers of the Robust WSD task supplied a data-set consisting of more than 170,000 document. These documents are a collection of articles from the Los Angeles Times (1994) and the Glasgow Herald (1995). The articles are all written in English, although there is a slight difference in the vocabularies of these two sources[382].

Each article consisted of an short title and the actual textual content. Both parts have already been tokenised, lemmatised, contained POS tags and were annotated with senses using WordNet[383] synsets. These senses are computed using two different word sense disambiguation systems - labeled UBC[384] and NUS[385]. The output of both systems has been added as annotations to the tokens. These annotations do only contain the most probable sense according to the system, but may contain multiple senses. Each of the senses carries the WordNet synset id and a score.

Additionally there have been more than English and Spanish 100 queries[386] for training and testing. For each of the training queries a list of relevant documents have been supplied. The query itself consisted of three parts: a title, a description and a narrative - with increasing level of verboseness. The participants of the Robust WSD task were asked to just use the title and description information. Just like the articles, all parts of the queries were annotated with word sense disambiguation information.

Relation of WSD and Information Retrieval The intuition that determining the correct sense of ambiguous words could improve the performance of information retrieval systems has generated a lot of research in the last couple of years. Results in the area of monolingual retrieval could not life up to the expectations[387,388]. Short queries and the skewed distribution of senses are among the explanations for the observed results.

Despite moderate results for monolingual retrieval tasks, the question is still open whether WSD also has minimal impact on other areas of information retrieval, like for example Question Answering (QA) and Cross-language Information Retrieval (CLIR). Some researchers[389] indicate that word sense disambiguation could help in multilingual retrieval.

WordNet Synsets The WordNet corpus has been built to model how words relate to each other. Therefore word are organised in a graph structure. There a multiple different types of relations, for example hypernyms and hyponyms. From the word sense disambiguation perspective there are two types of relations that are play the most important role.

Each sense of an ambiguous word is organised as a synset within the WordNet graph. These synsets may carry auxiliary information, for example the glosses and example sentences. But most importantly synset also contain synonyms for each individual sense.

For example the noun 'row' is related to seven different synsets, whereas for the verb row there is just a single sense. In figure 32 the entries of the WordNet graph for the word 'row' are shown. For three of these senses, there are synonyms. For example the sense #2 (an angry dispute) contains five alternative words that capture the same meaning. An in-depth description of the process of word sense disambiguation is given in the last presented application scenario within this chapter.

[382] A. Juffinger, R. Kern, and M. Granitzer. Crosslanguage Retrieval based on Wikipedia Statistics. In *Proceedings of 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, 17-19 September, Aarhus, Denmark*, 2008

[383] C. Fellbaum. *WordNet: An electronic lexical database*. The MIT press, 1998

[384] E. Agirre, O. L. D. Lacalle, and B. Country. UBC-ALM : Combining k-NN with SVD for WSD Eneko Agirre and Oier Lopez de Lacalle. *Computational Linguistics*, (June):342–345, 2007

[385] Y. S. Chan, H. T. Ng, and Z. Zhong. NUS-PT: Exploiting Parallel Texts for Word Sense Disambiguation in the English All-Words Tasks. *Computational Linguistics*, (June):253–256, 2007

[386] The actual number of queries varies between the task held in 2008 and 2009.

[387] M. Sanderson. Word sense disambiguation and information retrieval. *Intelligent Information Management*, 01(02):122–127, 1994

[388] E. Voorhees. Natural language processing and information retrieval. *Information Extraction*, page 724, 1999

[389] D. W. Oard and B. J. Dorr. A Survey of Multilingual Text Retrieval. *Electrical Engineering*, (UMIACS-TR-96-19):1–31, 1996

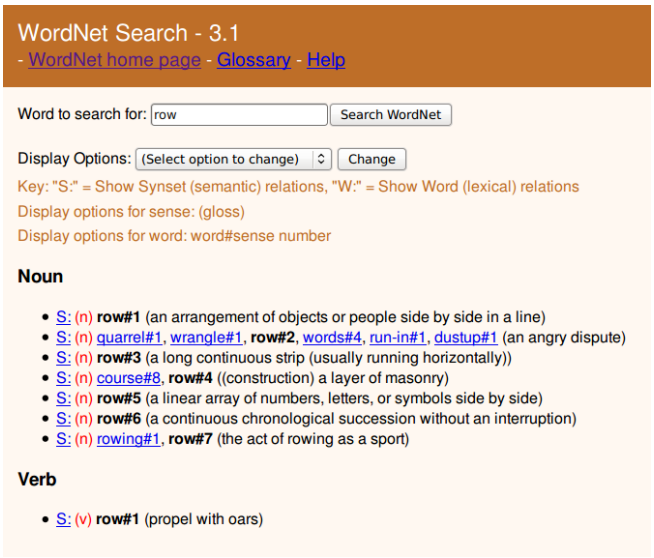

Figure 32: The ambiguous word 'row' can either take the role of a noun or a verb within a sentence. When used as a noun, there are seven different distinct senses according to WordNet.

Query Translation

In this section feature associations are highlighted as a way to translate query terms from a source language to a target language. Therefore an information retrieval system will be presented which has been used to participate in the CLEF 2008 Robust WSD task.

System Description The main focus of the information retrieval application has been to build a system where each query is treated equally, regardless of its language. As the language of the corpus is English, one can assume that there is a natural bias towards English queries. To remove this bias in order to measure the influence of the word sense disambiguation on the retrieval quality we applied the same pipeline to all input query languages. In figure 33 an overview is given of the main query processing steps.

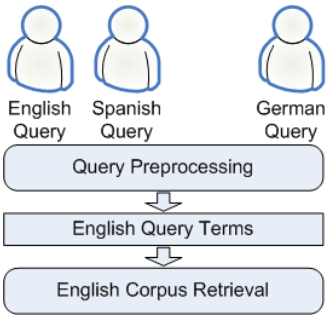

Figure 33: Overview of processing flow of the queries as issued by users. All queries are treated the same way, even if the language of the query matches the language of the corpus.

Additionally to the query translation we also applied a query expansion, which will be covered in the next sections. Therefore in this section this part of the query processing pipeline will only be briefly covered.

To test the impact of the word sense disambiguation on the performance of the retrieval we built multiple search indices. We created three different indices, each of them contains all articles from the data-set. But they vary in the way they are constructed.

Plain The first index is build using the default Lucene text processing and indexing chain. The plain text has been tokenised and transformed to lower-case. The additional annotations for the word sense disambiguation information have been ignored for this index. This index contains about 600,000 terms.

WSD For the second index we tried to exploit the word sense disambiguation annotations. Therefore we add the synonyms for each token that carries a Synset annotation. Tokens are often annotated with more than one sense. In that case we picked the sense with the highest score.

For the highest ranked sense annotation we used the Synset id to identify the synonyms for a single token. All the found synonyms are then added to the search index. Additionally we took care that phrase queries still work with the added terms. For example a document which contains the token sequence “baby food” will be retrieved by either the phrase query “baby food” or the phrase query “infant food”.

MST The final index has been built by analysing the noun co-occurrences within the articles. From these co-occurrence network, the minimum spanning trees have been computed and stored as tokens within a Lucene full-text index. As this index configuration does not directly allow assessments to whether word sense disambiguation may improve the

retrieval process, this index will not be taken into consideration in the results section.

An overview of the overall query processing of the CLEF 2008 system is given in figure 34. English and Spanish queries are both treated equally by our query translation processing. Depending on the configuration either the plain index or the index built using the word sense disambiguation is used. The query expansion techniques are described and evaluated in the next section. The final step of the retrieval chain is the search itself using either the plain index or the index that additionally contains the synonyms of the highest-scoring sense.

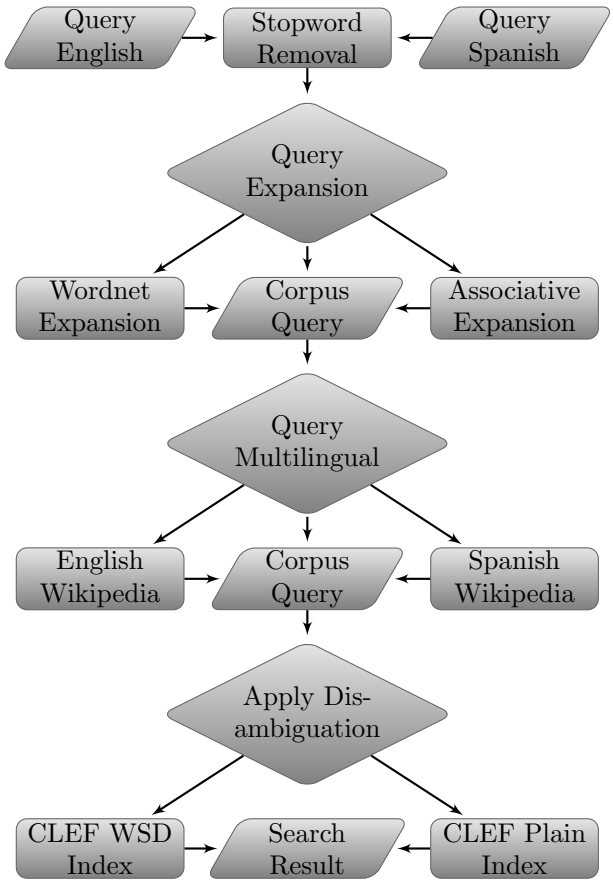

Figure 34: Overview of the retrieval pipeline for our CLEF 2008 system.

Multilingual Associative Index The goal of the multilingual associative index is to find the best matching terms in a language that is different to the original language of an input term using information retrieval techniques. Within the index data-structure all the entries are made up of multiple fields, where each of these fields corresponds to a single language. The multilingual associative index can be created using various multilingual resources. For the CLEF 2008 system we used the *Wikipedia*[390] as a corpus.

The free encyclopedia *Wikipedia* exists in various editions in different languages that also contain links between corresponding articles. We exploited this link infrastructure to automatically build a multilingual index for all query languages, namely English and Spanish. We took the XML dumps[391] provided by *Wikimedia* organization were parsed by the *Wikipedia Java API*[392] as our data-source. The *Wikipedia* multilingual index

[390] http://en.wikipedia.org/wiki/Main_Page

[391] `http://download.wikimedia.org/backup-index.html`

[392] `http://matheclipse.org/en/Java_Wikipedia_API`

thus finally contains aligned articles that are available in the two target languages.

In the CLEF 2009 system we added an additional source, the *Europarl* corpus[393]. The *Europarl* corpus[394] is created using the proceedings of the European parliament taken from the years 1996-2006. This resource again enables us to build a sentences aligned multilingual associative index. As the multilingual transcripts first need to be aligned, we applied the Church and Gale algorithm[395]. The Europarl corpus contains versions in 11 European languages, but for our system we used only the English and Spanish versions.

[393] `http://www.statmt.org/europarl/`

[394] P. Koehn. Europarl: A parallel corpus for statistical machine translation. *MT summit*, 5:12–16, 2005

[395] W. A. Gale and K. W. Church. A Program for Aligning Sentences in Bilingual Corpora. *Computational Linguistics*, 19(1):75–102, 1991

Table 45 gives an overview of the two multilingual indices. The Wikipedia index consists of whole articles whereas the Europarl index is build out of sentences. One can observe that there is huge gap in the number of terms between the two resources.

	Entries	**English Terms**	**Spanish Terms**
Wikipedia	2,896,802	5,139,238	1,365,908
Europarl	1,304,243	88,370	146,537

Table 45: Statistics of the Wikipedia and Europarl multilingual indices.

The intuition behind our term translation approach is similar to select terms for query expansion using the top ranked documents in pseudo relevance feedback methods [396]. For each term, which can either be a single word or a phrase, a query is build. This query is then used to search for relevant documents in the query source language. From the top hits of the results - D_{top} - the aligned documents in the target language are retrieved. From the terms contained in these document the term candidates for translation are calculated.

[396] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, volume 61. Cambridge University Press, 2008

We finally implemented three different scoring algorithms for estimating the best translation for the input term. The first is based on the well known *TFIDF* weighting scheme. For each term the weight w_i is calculated using the score of the most relevant documents D_{top} ($docFreq$ is the number of documents the translation candidate is contained in, N is the total number of documents):

$$w_i^{TFIDF} = log(\frac{N}{docFreq_i + 1} + 1) * \sum_{d \in D_{top}} score(d) \tag{130}$$

For the CLEF 2009 system we added two additional weighting schemes. The intuition behind the next scoring algorithm is to maximise the likelihood that a term has caused the document to be relevant. To accomplish this the same formula that is used to calculate the score of a document in the source language is applied on all target language terms found in the most relevant hits. The aggregated difference between the actual score and the reconstructed score serves as base for the weight of a single term:

$$w_i^{reconstruction} = \frac{1}{\sum_{d \in D_{top}} |tf_{i,d} * idf_i - score(d)| + 1} \tag{131}$$

$$idf_i = log(\frac{N}{docFreq_i + 1} + 1) \tag{132}$$

The final scoring algorithm is based on the well-known cosine similarity. The vector of scores for the top scoring documents v^S in the result set is compared with a vector v^i, which contains the $TFIDF$ weights calculated from the aligned document.

$$w_i^{cosine} = \frac{\sum_{d \in D_{top}} v_d^S v_d^i}{\|v^S\| \|v^i\|} \tag{133}$$

$$v_d^i = tf_{i,d} * log(\frac{N}{docFreq_i + 1} + 1) \tag{134}$$

Results The organisers of the Robust WSD track also provided an extensive set of test queries. By applying our information retrieval system on the test collection should provides insights mainly in regard to two questions.

- Does the retrieval performance increase by adding synonyms to the indexing chain?
- How well does our query translation scheme work?

While the fist question can answered relatively easy, by comparing the results of the performance on the test collection for the plain and the synonym index. To measure the quality of the query translation step, we first run our system on the English queries with the additional translation step. Next we compare the performance of this configuration with the performance without any translation.

First we compare the performance of our system on the CLEF 2008 test corpus using the plain index and the index that contains the word sense disambiguation information. In figure 35 the performance for the baseline configuration is shown. This configuration does not integrate any synonyms into the retrieval process.

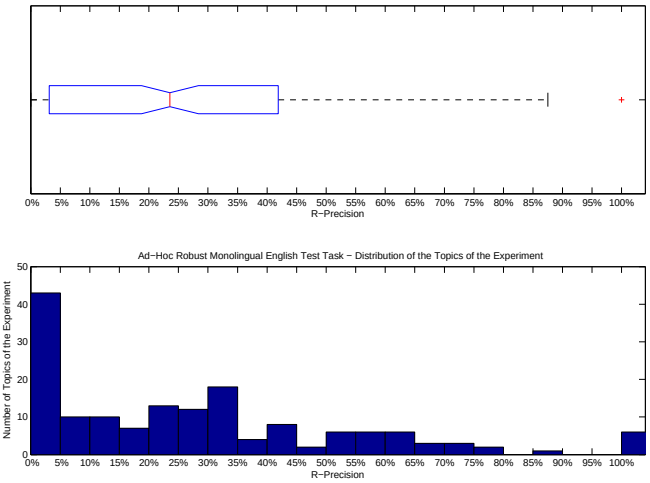

Figure 35: Performance of our information retrieval system for the CLEF 2008 campaign without using the word sense information into account. This configuration serves as baseline to assess the impact of the synonyms on the systems' performance.

In figure 36 the performance of the configuration using the Synset information is depicted. The inclusion of synonyms into the search index does not improve the overall retrieval performance. Instead, the results of the two configuration appear to be very similar.

The second main question that should be addressed by the evaluation on the test collection is the assessment of the quality of our query translation scheme. Therefore we recorded the results using multiple configurations. The baseline is defined by the results of using no translation at all for the English queries. In figure 37 the results for the set of English queries is given.

The various translation schemes provide similar levels of performance. Only the cosine weighting scheme appears to deviate from the other two translation weighting functions. When comparing the best performing

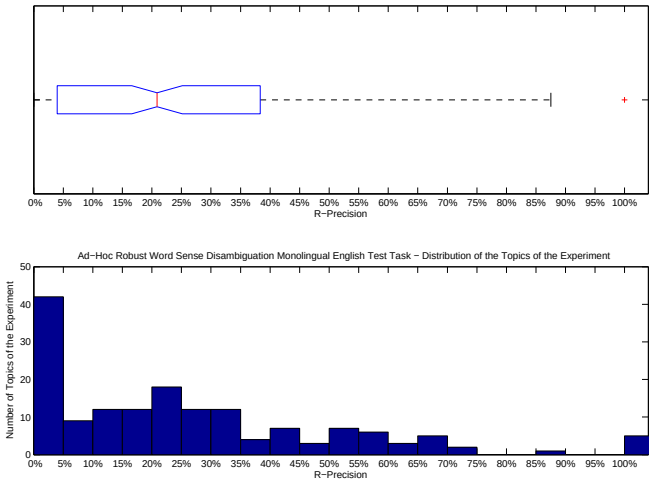

Figure 36: Performance of our information retrieval system when integrating the word sense disambiguation information into the retrieval process. Clearly the added information does not improve the overall performance on the test collection.

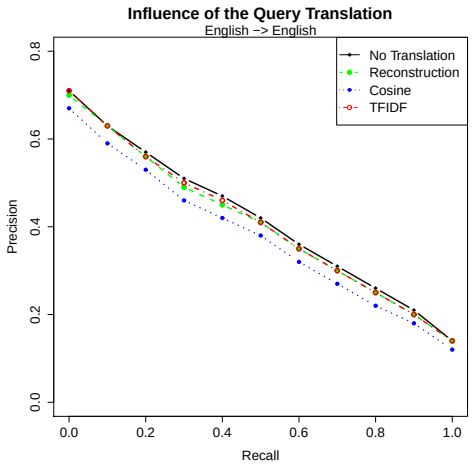

Figure 37: Comparison of the performance for the English queries using different translation strategies. There is no pronounced difference in the performance of the various weighting scheme. All translations provide a retrieval performance close to using no translation at all.

translation scheme in relation to using no translation at all one can observe that the translation does not have a negative impact.

Finally we compared the performance of the different translation weighting function for the Spanish queries. The results are presented in figure 38.

For the Spanish queries one can observe a more pronounced difference between the three weighting schemes. Using the w_i^{TFIDF} function gives the best results on the test collection. When comparing the performance of this weighting function with the results of the English queries one can observe a drop in performance. When using the CLEF 2009 system in the baseline configuration, for the English queries a MAP of 0.4022 is achieved, while for the Spanish queries the MAP is calculated as 0.2885.

Conclusions The query translation processing demonstrated good performance in the CLEF 2008 and CLEF 2009 information retrieval systems. It is based on the multilingual associative index to find corresponding terms. This data-structure has been populated using data gathered from *Wikipedia* and the *Europarl* corpus. In the evaluation the performance of the system when applying the translation step has been found to be similar to using no translation at all. When using the Spanish queries we observed a more pronounced difference in the results when using different translation weighting schemes.

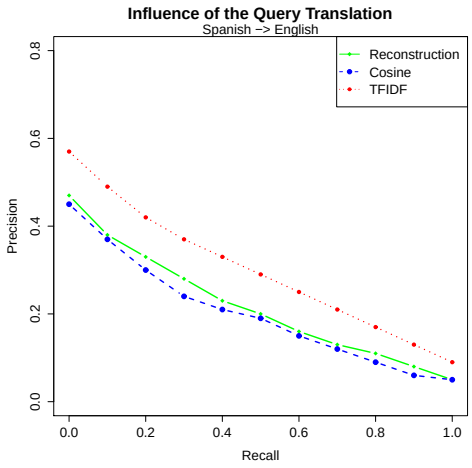

Figure 38: Performance of the CLEF 2009 system for the Spanish queries. The configuration using the TFIDF weighting scheme for translation outperforms the other two configurations.

Given that our translation algorithm has not been explicitly developed as a general machine translation tool the results can be considered to be satisfying. In a later section the feature association framework will be presented to efficiently store and retrieve the result of component that has been deliberately designed to be part of a statistical machine translation pipeline.

Query Expansion

In this section the feature association framework is presented as a tool to improve the quality of an information retrieval system. Initially we have developed a system[397] to take part in the Robust WSD track of the CLEF 2008 campaign. For the CLEF2009 challenge we customised and improved the original version of the system[398]. It has been modified to integrate different types of retrieval and ranking functions. The system now contains multiple *TFIDF* weighting schemes, the BM25 [399] weighting function and additionally a retrieval function utilising an axiomatic retrieval approach [400]. In our experiments we evaluated these different retrieval functions and implemented different strategies to add the WSD information to the retrieval process.

Feature Associations for Query Expansion By using WordNet and the annotated sense of ambiguous terms it is possible to determine the synonyms for a specific sense. The relation between synonymous words are one of many semantic relatedness relationship types between words. Statistical methods provide unsupervised means to detect word pairs with a high semantic relatedness without restriction to a specific relationship type. One of these methods is based on the co-occurrence statistics of words within a corpus. Many algorithms have been proposed to accomplish this task, using different weighting functions to measure the relationship between words. The Pointwise Mutual Information (PMI) has been found to provide good performance in this regard[401].

For our system, we use semantically related words to conduct a specific form of query reformulation. Query expansion[402] is a technique to extend an user query with additional terms. There are many variation of this technique, which can be categorised into two groups:

Global Query Expansion When using global query expansion, terms are added to the query based on external sources. Usually these sources are knowledge bases like a thesaurus or a dictionary. For instance expanding a query with the synonyms found in WordNet would be a typical application scenario for a global query expansion approach[403].

Alternatively to resort to external sources, one can also exploit the document corpus itself. All documents that are stored in the search index are analysed and serve as base to expand the original query.

Local Query Expansion While the global query expansion can be seen as an independent pre-processing before the query is submitted to the search index, the local query expansion technique integrates the actual retrieval into the query processing itself. At first the original query is used to retrieve the highest ranked documents. These documents are assumed to be highly relevant. Out of these documents additional query terms are extracted. Therefore this technique could also be seen as a type of pseudo relevance feedback (or blind relevance feedback). In the past this kind of query expansion did show improvements on the retrieval performance[404,405].

These two types of query expansion do not exclude each other. In fact both approaches have already been combined in the past[406]. In our information retrieval system we opted for a global query expansion approach. Therefore we have employed the feature association framework to compute semantically related terms based on the co-occurrence statistics of the documents' terms. Exploiting the distributions of terms in relation to other terms has already been applied for information retrieval tasks[407]. The co-occurrence statistics based on the CLEF2009 article corpus were calculated by using a modified Pointwise Mutual Information (PMI) measure for all

[397] A. Juffinger, R. Kern, and M. Granitzer. Crosslanguage Retrieval based on Wikipedia Statistics. In *Proceedings of 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, 17-19 September, Aarhus, Denmark*, 2008

[398] R. Kern, A. Juffinger, and M. Granitzer. Evaluation of Axiomatic Approaches to Crosslanguage Retrieval. In *Multilingual Information Access Evaluation Vol. I Text Retrieval Experiments*, 2009

[399] S. Robertson and M. Gatford. Okapi at TREC-4. In *Proceedings of the Fourth Text Retrieval Conference*, pages 73–97, 1996

[400] H. Fang and C. Zhai. An exploration of axiomatic approaches to information retrieval. *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 480–487, 2005

[401] P. D. Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the twelfth european conference on machine learning (ecml-2001)*, pages 491–502, 2001

[402] E. N. Efthimiadis. Query Expansion. *Annual Review of Information Systems and Technology ARIST*, 31(1):121–187, 1996

[403] E. M. Voorhees. Query expansion using lexical-semantic relations. In W. B. Croft and C. J. V. Rijsbergen, editors, *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 61–69. Springer-Verlag New York, Inc., 1994

[404] C. Buckley, G. Salton, J. Allan, and A. Singhal. Automatic Query Expansion Using SMART: TREC 3. In *In Practice*, pages 69–80. NIST, 1994

[405] D. Carmel, E. Farchi, Y. Petruschka, and A. Soffer. Automatic query refinement using lexical affinities with maximal information gain. *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 02*, page 283, 2002

[406] J. Xu and W. B. Croft. Query expansion using local and global document analysis. *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 96*, (Zurich, Switzerland):4–11, 1996

[407] E. Terra and C. L. A. Clarke. Scoring missing terms in information retrieval tasks. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 50–58. ACM, 2004

words that occur at least 2 times and less than in 50% of all documents. The similarity between two words w_i and w_j is defined as:

$$S_{CondPMI}(w_i, w_j) = \frac{log_2 \frac{P(w_i|w_j)}{P(w_j)}}{log_2(\frac{1}{P(w_j)})} \tag{135}$$

When processing a query these associations are used to expand the query by additional terms. For each term in the users query, the top co-occurring terms are searched and then added to the query.

Using statistically associated terms is just on of three possible query expansion strategies of our information retrieval system. We tried to include the word sense disambiguation information into the query processing as well. Finally our system offers four possible query processing configurations:

None No query expansion, the users input is submitted directly to the search index.

Synset ID From the word sense disambiguation annotation of the queries, the Synsets with the highest scores are selected and added to the query. The search index has been built to also contain this information for all indexed articles.

Synonyms Again the highest ranked Synset are selected. But not the ID of the Synset, but the synonyms itself are added to the query.

Co-Occurrence Terms The query terms are submitted to the feature association framework to retrieve associated terms. These associated terms are then added to the query.

All All query expansion strategies (Synset ID, Synonyms and Co-Occurrence Terms) are applied.

The query expansion strategies that exploit the word sense disambiguation annotations can be configured to either use the NUS or the UBC annotations. After the query has been processed and optionally been expanded it is submitted to the search index to retrieve a ranked list of documents.

Retrieval Functions The *TFIDF*[408] weighting scheme and the *BM25*[409] approach are textbook methods to retrieve relevant document for a given query. Both demonstrated robust and reliable performance in the past. A variant of the *TFIDF* retrieval model did provide good, but not state-of-the-art performance in the CLEF2008 Robust WSD task[410]. Many of the CLEF2008 participants incorporated the BM25 approach into their retrieval systems with great success[411,412]. We therefore also report the performance our system using an implementation of the BM25 weighting scheme[413] (In the evaluation runs we set k_1 to 0.8 and b to 0.5):

$$S_{BM25}(Q, D) = \sum_{t \in Q \cap D} \frac{tf_{t,D}}{k_1 lengthNorm_D^{BM25} + tf_{t,D}} * idf_t^{BM25} \tag{136}$$

$$lengthNorm_D^{BM25} = (1 - b) + b * \frac{docLength_D}{averageDocLength} \tag{137}$$

$$idf_t^{BM25} = log \frac{N - docFreq_t + 0.5}{docFreq_t + 0.5} \tag{138}$$

For our main experiments we have chosen to apply findings in the area of axiomatic approaches to information retrieval. Fang and Zhai presented[414] several variations of weighting functions build using a set of

[408] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988

[409] S. Robertson and M. Gatford. Okapi at TREC-4. In *Proceedings of the Fourth Text Retrieval Conference*, pages 73–97, 1996

[410] A. Juffinger, R. Kern, and M. Granitzer. Exploiting Cooccurrence on Corpus and Document Level for Fair Crosslanguage Retrieval. In *Working Notes for the CLEF 2008 Workshop, 17-19 September, Aarhus, Denmark*, 2008

[411] L. Dolamic, C. Fautsch, and J. Savoy. UniNE at CLEF 2008: TEL, and Persian IR. *Evaluating Systems for Multilingual and Multimodal Information Access*, pages 178–185, 2009

[412] J. Guyot, G. Falquet, S. Radhouani, and K. Benzineb. UNIGE Experiments on Robust Word Sense Disambiguation. In *Evaluating Systems for Multilingual and Multimodal Information Access*, 2008

[413] `http://nlp.uned.es/~jperezi/Lucene-BM25/`

[414] H. Fang and C. Zhai. An exploration of axiomatic approaches to information retrieval. *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 480–487, 2005

axioms that constrain the properties of a weighting function. The authors did recommend one of their derived retrieval functions which has shown promising performance in their evaluation. We did adapt this function for our retrieval system. The score of a document D out of N documents given a set of query terms Q is build using the tuning parameter α and β:

$$S_{Axiomatic}(Q,D) = \sum_{t \in Q \cap D} idf_t^{Axio} * \frac{tf_{t,D}}{tf_{t,D} + lengthNorm_D^{Axio}} \quad (139)$$

$$lengthNorm_D^{Axio} = 0.5 + \beta \frac{docLength_D}{averageDocLength} \quad (140)$$

$$idf_t^{Axio} = (\frac{N}{docFreq_t})^{\alpha} \quad (141)$$

Using the training topics we found the setting of 0.25 for α and 0.75 for β to provide a satisfying performance.

Results In the evaluation using the CLEF 2009 test collection we tried to assess two main questions: a) does incorporating the word sense disambiguation information improve the retrieval performance and b) how does the query expansion using associated terms compare to the other query expansion methods?

In the first set of evaluation runs the difference retrieval functions are compared to find the best performing configuration that does neither use query expansion nor exploits the word sense disambiguation annotations. This configuration will then be used as a baseline to measure the impact of the query expansion strategies. In table 46 the different retrieval functions are compared using the CLEF 2009 test collection. Although this comparison gives no insights into the question whether WSD information could improve the performance, it demonstrates that the results of the axiomatic approach is indeed a valuable contribution to the arsenal of information retrieval techniques. The according GMAP measure is improved over the BM25 run, which indicates that especially low performing topics did improve using the axiomatic approach.

Retrieval Function	MAP	GMAP	Notes
TFIDF1	0.3083	0.1182	*Lucene Boolean Query*
TFIDF2	0.3313	0.1331	*Lucene Disjunction Max Query*
BM25	0.3889	0.1566	*Using* $k_1 = 0.8$ *and* $b = 0.5$
Axiomatic	0.4022	0.1805	*Using* $\alpha = 0.25$ *and* $\beta = 0.75$

Table 46: Performance of the monolingual system without query expansion.

For the comparison with the configurations that utilise the WSD information we only report the performance figures achieved using the axiomatic retrieval function. Table 47 lists the performance measures of the various query expansion configurations.

The best performing configuration combines the synonym, synset and term co-occurrence information, labeled “all” in the table. The performance figures do show that integrating the words sense disambiguation data into the retrieval process of our system does improve performance. Not only does the baseline configuration benefit from the sense annotations, but also the configuration that already uses a (successful) query expansion technique is improved further. The difference between the two WSD data sets (NUS and UBC) and between the Synonym and the Synset features are too small to allow any conclusions.

Query Expansion	MAP	GMAP	Wilcoxon	Randomized
Synonyms (NUS)	0.4061	0.1849	0.0376	0.0517
Synonyms (UBC)	0.4036	0.1837	0.3286	0.2070
Synset IDs (NUS)	0.4047	0.1856	0.0303	0.0944
Synset IDs (UBC)	0.4070	0.1869	0.0119	0.0147
Co-occurrence Terms	0.4170	0.1864	0.0001	0.0196
All (NUS)	0.4222	0.1947	0.0174	0.0554
All (UBC)	0.4212	0.1942	0.1603	0.0730

Table 47: MAP and GMAP measures of the monolingual system using a combination of features together with the p-values of two significance test. Both significance tests agree that for most of the configurations the improvement is not achieved by chance.

We conducted two significance test to assess whether the improvement over the baseline (the axiomatic configuration for the first five runs and the query expansion using co-occurrence statistics for the last two runs) is statistically significant. These tests were the Wilcoxon signed rank test and randomised tests, which according to [415] should be the preferred way to test for significance in information retrieval applications.

The evaluation on the CLEF 2009 test dataset demonstrates that adding word sense disambiguation information to an existing information retrieval system can improve its performance. This is also the case for the query expansion technique which uses the feature association framework to find semantically related terms.

Conclusions There are a number of conclusions that can be drawn from the information retrieval system which has been developed for the CLEF 2008 and CLEF 2009 campaign. For example it has been shown that our query translation algorithm achieves a satisfying performance. The impact of incorporating words sense disambiguation information into an existing information retrieval application has been studied. Finally it has been found that using the co-occurrence of terms to feed a query expansion algorithm outperformed the baseline on the CLEF 2009 test data set.

The feature association framework did serve as tool for multiple tasks within this information retrieval application. The focus of this application scenario has been on the analysis between features and their retrieval. The flexibility and wide range of configuration setting of the feature association framework helped to build a state-of-the-art retrieval system in a minimum amount of time.

[415] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management - CIKM '07*, page 623, New York, New York, USA, 2007. ACM Press

Natural Language Processing

Applications which employ natural language processing are particularly demanding. In the most cases the amount of data is large and the data itself tends to be noisy. Furthermore the distribution of the features vary between data sets and within data sets. Therefore the requirements on exploiting feature associations are accordingly high.

Overview

In the majority of cases natural language processing (NLP) is applied on unstructured plain text. Usually the textual content is organised in documents or varying length. In this setting features will represent single words within these documents. Thus the association network often reflects how word relate to each other.

In this section two natural language processing applications will be presented. In both cases the feature association framework has been integrated into these applications.

Cross-Language Text-Reuse Intellectual property has begun to play an increasingly important part in the global economy. Today patents and copyrights are considered by many as a vital part of the assets of a company. Even in the research community these topics have gained importance. For example, cases of plagiarism have drawn the attention of mass media towards this topic.

Plagiarism is a specific case of text-reuse. Additionally to this kind there are also alternative, legitimate forms of text-reuse.

The detection of text-reuse poses a number of challenges. The amount of data that needs to be analysed is huge in the most cases. Paraphrasing and other types of modification to the text increases the effort to uncover the original source of a text. One special case for such a modification is cross-language text-reuse. Given a text in one language, its content is simply translated to another language.

In the first application scenario a system to detect such cross-lingual text-reuse will be presented[416]. Feature associations between corresponding words in multiple languages are utilised to find translated pairs of text fragments. Within this application setting the focus lies on the efficient and fast retrieval of associations.

Word Sense Induction and Discrimination In the second presented application scenario the feature association framework is used to detect and identify senses of polysemous words[417]. This use-case combines all major features of the feature association framework, making use of all available generalisations. It is used for feature analysis as well as feature synthesis. Furthermore the flexibility to integrate additional processing steps into the feature association pipeline is exploited.

The detection of multiple senses of an ambiguous word is one of the most challenging tasks of natural language processing. Word sense induction is a specific type of word sense disambiguation. In this case the number of distinct senses are not known, therefore the algorithm has to induct the different senses in an unsupervised manner. Due to this approach, the data sets need to be large to contain sufficiently many examples of each sense for the machine learning to deliver meaningful results. This leads to a set of requirements on the feature association framework, not only in terms of quality of the results, but as well as on the run-time needed to compute these results.

[416] M. Muhr, R. Kern, M. Zechner, and M. Granitzer. External and Intrinsic Plagiarism Detection using a Cross-Lingual Retrieval and Segmentation System Lab Report for PAN at CLEF 2010. In *2nd International Competition on Plagiarism Detection*, 2010

[417] R. Kern, M. Muhr, and M. Granitzer. KCDC: Word Sense Induction by Using Grammatical Dependencies and Sentence Phrase Structure. In *Proceedings of SemEval-2*, Uppsala, Sweden, ACL, 2010

Cross-Language Text-Reuse

In this section the feature association framework will be featured as a part of an solution to detect cross-lingual text reuse. This solution has been developed to take part in a competition where multiple plagiarism detection algorithms are compared against each other. Text reuse is a special form of plagiarism, where the original author of a text is not given and the text is not properly quoted. With more data easily available on the Internet, plagiarism has become an increasingly important topic in research and industry alike[418]. Text reuse is not strictly limited to plagiarism, other forms have been investigated as well. For example the reuse of text written by news agencies within newspapers has been studied[419].

[418] H. Maurer, F. Kappe, and B. Zaka. Plagiarism - A Survey. *Journal Of Universal Computer Science*, 12(8):1050–1084, 2006

[419] P. Clough, R. Gaizauskas, S. S. L. Piao, and Y. Wilks. METER: MEasuring TExt Reuse. *Annual Meeting of the ACL*, page 152, 2002

PAN 2010 The main goal of the PAN series has been to assess the current state-of-the-art in the area of plagiarism detection. The PAN initiative started out as a workshop series and more recently has been accompanied by a competition. In 2010 this competition has been conducted for the second time. The organisers of this challenge tried to provide an evaluation data set which covers multiple settings for plagiarism detection and multiple types of plagiarism[420].

[420] M. Potthast, B. Stein, and T. Holfeld. Overview of the 1st International Competition on Wikipedia Vandalism Detection. *Notebook Papers of CLEF 2010 LABs and Workshops*, pages 22–23, 2010

External Plagiarism Detection In the external plagiarism detection setting a collection of documents have been provided, labelled *source documents*. Additionally another set of documents have been given to the participants. The second set were the *suspicious documents*, which may contain plagiarised passages from the source documents. The length of these passages varies between a couple of words to multiple paragraphs. Furthermore some suspicious documents contain multiple plagiarised sections, while others contain none.

These suspicious documents were not generated on real cases of plagiarism, but were artificially generated. This allowed the organisers to control the level of modification which have been made to the plagiarised passages. Three different degrees of modifications have been applied.

None Some of the plagiarised passages were direct copies from the source documents. No modification on the text were made. These type of plagiarism is expected to cause the least effort to be detected.

Low The next level of modification introduces small changes to the source passages. They reflect an author who puts little effort into concealing the text copy.

High The final type of modification mimics an author who puts considerable amount into changing the original text. The order of the words within sentences are changed, as well as synonyms are inserted to replace some words in the source passage. It is expected that this kind of plagiarism will be the hardest to be detected.

Additionally to the different levels of modifications, the organisers also added multiple documents to the pool of source document in a language different to the language of the suspicious documents. While the suspicious documents are all written in the English language, the source collection contains documents in English, Spanish and German. The task of the plagiarism detection algorithm is to be able to cope with this type of cross-lingual plagiarism.

The main challenges to build a system which operates on a given corpus of source document is to make it scale for huge amounts of data. Even if only simple algorithms are employed to identify copied text passages, the number of operations needed is the dominating factor in the systems' run-time complexity.

To group similar source documents and then apply a cluster pruning technique has been proposed to reduce the number of necessary operations[421]. In this section an alternative approach based on information retrieval techniques will be presented.

Intrinsic Plagiarism Detection Often the source document for a plagiarised passage might be not available. Humans will still be able to a certain extend to detect a change in the writing style and to conclude that a certain text has been written by another author. This setting is simulated in the intrinsic plagiarism detection setting. In this setting only a list of suspicious documents is given. The task of the plagiarism detection algorithm is to be able to spot differences in the writing style within a document.

[421] M. Muhr, M. Zechner, R. Kern, and M. Granitzer. External and Intrinsic Plagiarism Detection Using Vector Space Models. In *Proceedings of the SEPLN'09 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse*, 2009

Due to the success of the first PAN series of workshops and competitions, the scope has been broadened in recent years. In 2010 the PAN workshop also incorporated tasks for vandalism detection[422]. The deliberately manipulation of Wikipedia articles serves as use case for this kind of task.

In the following year the PAN competition has been further expanded to host tasks related to authorship identification. These tasks have been authorship attribution and authorship verification. Both task are similar to the intrinsic plagiarism detection setting.

In the case of authorship attribution the algorithm should identify which author out of a known set of authors has fabricated a given document. Therefore the organisers provided a set of training documents, where each of the document has been written by a single author. The test set is made up of previously unseen document from these authors. The algorithm should identify the original authors of these documents. A variation of the test set additionally contained documents from authors who were not present in the training set.

[422] M. Potthast, B. Stein, and T. Holfeld. Overview of the 1st International Competition on Wikipedia Vandalism Detection. *Notebook Papers of CLEF 2010 LABs and Workshops*, pages 22–23, 2010

The authorship verification task just focuses on a single author. In this scenario the algorithm has to decide whether a given document is written by a specific author or by someone else. Although these tasks appear to be similar to the intrinsic plagiarism detection, the underlying techniques vary. For example, the authorship identification setting is suited for an approach that employs supervised machine learning algorithms[423].

[423] R. Kern, C. Seifert, M. Zechner, and M. Granitzer. Vote/Veto Meta-Classifier for Authorship Identification. In *In 3nd International Competition on Plagiarism Detection*, 2011

System Description In this section a system will be presented which has been used for the PAN 2010 competition. It consists of algorithms and modules to tackle all different scenarios and settings as available in the data-set. The system uses techniques developed in the field of information retrieval to provide a scalable solution to the external plagiarism setting. An algorithm originally developed for the task of splitting long documents into shorter coherent parts has been adapted to detect changes in the authorship within texts.

The feature association framework has been integrated into the module responsible to search for possible candidates of plagiarised passages. In this use-case the association network has not been built by the feature association framework itself. The associations were computed by adapting tools developed as part of statistical machine translation systems. The efficient retrieval technique for associated features is the main focus in this use-case. The feature association framework thereby provides a method to compute translation candidates for all words of all documents in the set of source documents which are not written in English.

The processing pipeline of the plagiarism detection system can be outlined as:

- The detection of plagiarism from a known sources is reformulated as an information retrieval task on text blocks. These text blocks are

created out of the source documents. The output of this stage is a list of candidate blocks.

- In the post-processing multiple blocks are filtered out or merged to build the list of plagiarised passages from known sources.
- Source documents which are not written in the English language are processed by applying an additional translation step. Within this step the feature association framework is responsible to retrieve translations for individual words.
- For the detection of changes of authorship within a single document a sliding window approach has been adapted.
- Finally the results from the external as well as intrinsic stages are combined to come up with a unified list of plagiarised passages

In figure 39 the overview of the system is depicted. On the left part within this picture the processing of the source documents is shown. If a source document is not written in the English language, all words are replaced by translation candidates.

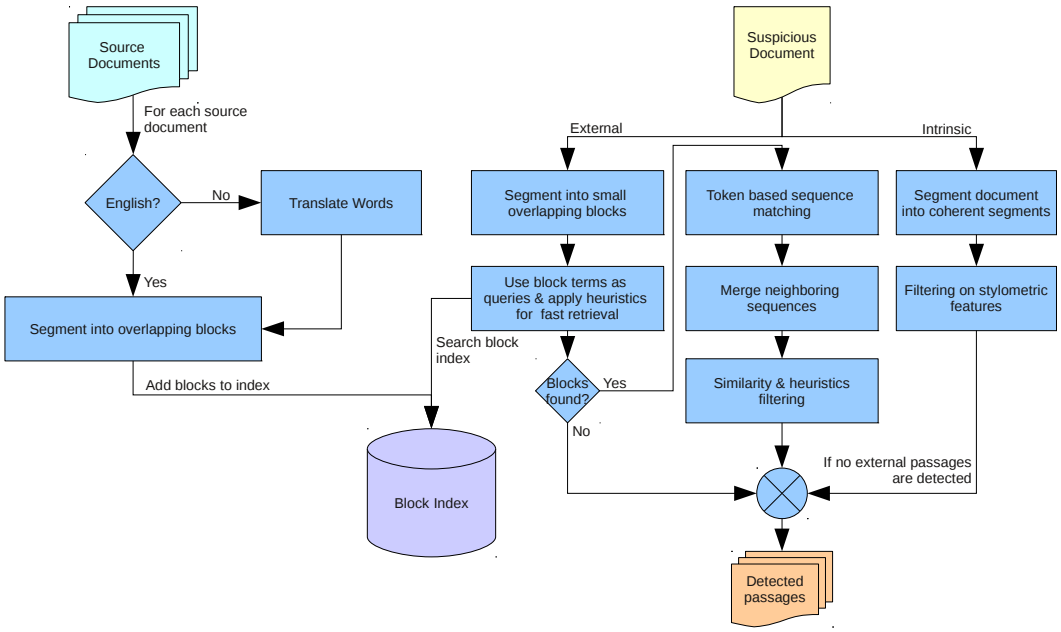

Figure 39: Overview of the processing for the combination of external and intrinsic plagiarism detection. For cross-lingual plagiarism an additional translation component is integrated.

Word alignment algorithms are commonly used within statistical machine translation systems. The task of word alignment is for find corresponding pairs of word from two different languages. The source for this family of algorithms is a corpus which has been aligned on sentence level. For the PAN system we selected the Europarl aligned corpus (Release 5)[424] as base to calculate the word alignments. The computation of the word translation candidates were conducted by adapting the BerkeleyAligner[425] software package[426].

The output of the word alignment process is a rank list of English alignment candidates for each German and Spanish word. The top 5 results from is list were taken to build an association network. Within this graph the German and Spanish word are the start nodes connected to the English translation candidates. The association network is then used during the processing of the source documents.

The source documents are processed by splitting their content into overlapping blocks of words, which are then indexed by a full-text search

[424] P. Koehn. Europarl: A parallel corpus for statistical machine translation. *MT summit*, 5:12–16, 2005

[425] P. Liang, B. Taskar, and D. Klein. Alignment by agreement. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 104–111. Association for Computational Linguistics, 2006

[426] `http://code.google.com/p/berkeleyaligner/`

engine. Each document is first tokenised and a sliding window approach is employed to traverse of the stream of words. The windows size has been set to 40 and the increment has been set to 20. For each 20th token within the source document a new block of 40 consecutive words is generated. Each block is then passed to the search index as an own document, together with the id of the originating source document.

If the source document is not written in English, the words are translated before being passed to the search engine. Each word is selected as start node within the association network and the top 5 translation candidates are retrieved. Finally each block will not contain 40 words, but up to 200 English words.

When processing the suspicious document a similar processing is conducted. Again a sliding window approach is taken, but using a different set of parameters. The window size is 16 words, while the step size is 6 terms. There parameters have been manually assigned based on the results on a training data set.

Out of each block from the suspicious document a search query is formulated. The document frequency of the words within the query is taken to optimise the retrieval process. Infrequent words, that only occur in few documents, will generally lead to a faster query execution. Therefore the words within the query are reordered so that the most frequent terms are the last to filter out the set of matching documents.

To cut down the number of requires search request, a heuristic to discard certain queries has been developed. Again the document frequency is evaluated for all terms within a query. If the query does not contain a single term, which frequency falls below a predefined threshold, the whole query is skipped.

For each suspicious block query the search engine returns a ranked list of source blocks. The score of these results are then used to find a cut point to prune the list to a minimum number of candidates. After all blocks of a suspicious document are processed, the candidates are then passed to the post-precessing component of the system.

In the post-processing step the candidates from the block retrieval stage are filtered out and merged. The result of this processing is a list of passages within the suspicious document together with the corresponding text in the source documents. To accomplish this task the candidates are used to create word by word matrices between suspicious documents and source documents. Within this matrix matching sequences are detected. To cope with a high obfuscation the matches are allowed to contain gaps and changes in the sequence of words. Finally the Jaccard similarity is calculated between a passage within the suspicious document and its corresponding segment in the source document. If this similarity values exceeds a certain threshold, this passage is reported as being plagiarised according to our systems criteria.

Additionally to the cases of external plagiarism, where the source of the text is known by the system, the PAN data-set contains plagiarised passages from unknown sources. To detect this kind of text reuse, our system contains a component to detect changes in the writing style within a document. Therefore a linear text segmentation algorithm has been adapted[427]. This algorithm has been developed to detect topical changes within a document, by tracing the lexical cohesion of words.

The output of the text segmentation algorithm for the suspicious documents serves as input to the intrinsic plagiarism detection component. To detect changes in the writing style the words within the documents are first transformed into features. These features should reflect the stylometric properties of the authors writing style. In our system we integrated two types of stylometric feature transformation functions:

[427] R. Kern and M. Granitzer. Efficient linear text segmentation based on information retrieval techniques. In *MEDES '09: Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pages 167–171, 2009

Stop-Words This feature transformation is motivated by the intuition that different authors tend to resort to different stop-words to construct grammatically correct sentences. For this transformation function to work all words need to be annotated as either function word or content word. This is accomplished by looking up all words in a manually crafted stop word list. All words that have been identified as stop-word are added to the feature set and their frequency is additionally recorded. The remaining function words are ignored by this feature transformation function. As stop word list we took the stop-word lists from the Snowball stemmer project[428]. These list are available in a number of languages and also contain the set of pronouns.

[428] http://snowball.tartarus.org/

Stem-Suffix The last characters of words have already been used to identify specific author styles. The motivation for this feature transformation is the assumption that different authors may differ in their use of flections. One possible approach for this kind of function is to pick the last n characters of each words, where n is usually set to 3. For our system we used a flexible number of suffix characters. The suffix was determined by the number of characters a stemming algorithm would cut off or replace (we utilised the Snowball stemmer for this task). Finally this function produces a set of word suffixes.

For the detection of plagiarised passages we first build centroid feature set for the whole document. This is motivated by the assumption that only parts of the document are actually plagiarised, while the majority of the document is written by a single author. For each segment as produced by the text segmentation algorithm a feature set is constructed. These segment features are then compared to the centroid features via the cosine similarity. If the distribution of the features for a single segment falls below a threshold in regard to the similarity measure, the entire segment is marked as being potentially plagiarised. The final output of the intrinsic plagiarism detection component is a list of passages which deviate from the writing style of the complete document.

After applying the external and intrinsic plagiarism detection components on the suspicious document, their results are combined. The combined list is then compared against the test-data set to assess the quality of the algorithm.

Results During the development of our system we used a subset of the data set to tune the performance of the algorithms and parameter settings. A total of 500 suspicious document have been selected as base for this development corpus. For the competition itself a separate test data-set with previously unseen documents has been passed to the participants.

At first we investigated the raw performance of the retrieval step to find the candidate passages within the source documents. This should give an upper bound on the overall performance of our system on the external plagiarism setting.

In table 48 the result of this analysis is given. The three different levels of obfuscation and setting of cross-lingual plagiarism are reported separately. In the column 'Hit-Count' the number of passages is listed, which were successfully found by at least a single block query. The 'True-Count' specifies the number of real plagiarised passages, while the last column is the ratio between the two counts. Passages, that are not found in this stage will not be identified as plagiarised in later processing stages. Therefore this ratio is an indicator for the overall performance of our system.

All but the translated passages are retrieved in more than 90% of all cases. An additional processing to integrate synonyms into the search index could prove beneficial to cope with a high degree of modifications.

Obfuscation	Hit-Count	True-Count	Ratio
high	13,348	14,756	0.9046
low	14,832	14,883	0.9966
none	16,784	16,784	1.0
translated	5,462	6,314	0.8651

Table 48: Results for the evaluation of the block retrieval step of the PAN 2010 system. The ratio reflects the upper limit of the complete system for external task in terms of recall.

To compare the performance of different systems, the organisers of the PAN competition proposed a set of evaluation metrics. Starting with the precision and the recall for identifying plagiarised passages they also integrated a granularity based measure into the final score. The reported plagiarised passages should be fragmented as little as possible therefore a low value for the granularity score leads to a higher score.

Table 49 gives an overview of the performance of our plagiarism detection system on the evaluation data set. Each type of configuration and some selected combinations are reported in a separate row. The various settings for the external configuration without any cross-lingual results are then combined, as well as a combined result for all types of external plagiarism. Together with the intrinsic configuration the overall performance of our system is reported.

Task	Precision	Recall	Granularity	Score
non-translated all	0.9299	0.8967	1.0553	0.8785
non-translated high	-	0.8122	1.0771	-
non-translated low	-	0.9207	1.0968	-
non-translated none	-	0.9497	1.0025	-
translated	0.8036	0.6162	2.1655	0.4195
external	0.9053	0.8631	1.1611	0.7949
intrinsic	0.212	0.1566	1.0	0.1802
Overall	0.8417	0.7057	1.1508	0.6948

Table 49: Overview of the overall performance of the PAN 2010 plagiarism detection system. Each type of plagiarism is reported separately using the evaluation metrics proposed by the organisers of the PAN competition.

Our system was finally ranked third out of 18 participating systems based in the combined score. Furthermore out of all systems that tackled the problem of cross-lingual plagiarism, our system was the only one not to resort to Google Translate[429]. The most striking result of the evaluation is the difference between the external and the intrinsic plagiarism settings. Due to the missing source documents in the intrinsic task, a gap in performance has been expected.

[429] `http://translate.google.com/`

In terms of precision and recall the cross-lingual plagiarism did not achieve the same level of performance as the mono-lingual passages. But especially in terms of granularity there is a pronounced difference. This implies that the post-processing and merging step may profit from additional refinements. Still 80% of all cross-lingual passages were successfully identified, which can be seen as satisfying given the relative complexity of the task.

Conclusions For the PAN 2010 plagiarism detection system, the feature association framework has been used to store and retrieve pairs of word translation candidates to uncover cases of cross-lingual plagiarism. In this scenario the feature association themselves were computed by an tool, which has been developed as an part of a statistical machine translation system. Therefore the focus lies on the retrieval part of the association network. Because of the number of documents and their size, the retrieval

operation needs be efficient to allow the algorithm to be executed even on moderate computational resources.

The plagiarism detection system covered multiple plagiarism settings, and integrated several approaches to detect this kind of text-reuse. In the evaluation based on the PAN 2010 data-set it provided a satisfying performance. The ease of integration of the feature association framework did contribute to the versatility of the final system.

Word Sense Induction and Discrimination

The final presented use-case of the feature association framework combines all possible modes of operation. In this scenario the feature association serve as base to analyse the data set, to synthesise new features as well as retrieve the result of these calculations. Furthermore advanced features of the framework, for example building contextual local associations by applying machine learning techniques are explored. The application scenario is rooted in the domain of Natural Language Processing, where the feature association framework is used to detect and assign senses of ambiguous words[430].

The task of word sense induction and discrimination (WSID) is to identify distinct senses of ambiguous words in a set of text documents. These documents are not annotated and there are no training examples, thus WSID algorithms have to learn the senses in a completely unsupervised fashion. The motivation for developing such algorithms is the assumption that the detection of senses could help in a number of scenarios, for example information retrieval and machine translation. In recent years initiatives have been formed to develop not only test corpora, but also platforms to compare the quality of different algorithms.

[430] R. Kern, M. Muhr, and M. Granitzer. KCDC: Word Sense Induction by Using Grammatical Dependencies and Sentence Phrase Structure. In *Proceedings of SemEval-2*, Uppsala, Sweden, ACL, 2010

Related Work Natural languages consist of many ambiguous words, which poses no apparent problems for humans in text understanding. For machine based algorithms the situation is differently. The performance of many computational tasks that deal with natural languages could be improved if the sense distinctions could reliably be exploited. Word sense disambiguation (WSD) is the process of automatically detecting the correct sense of an ambiguous word within a textual context. Where the context can be a document, a paragraph or even only a single sentence. Traditionally the individual senses are known in advance and determined by linguists and lexicographers. Therefore the task of WSD is to find the correct sense of a word out of the set of predefined senses. Three main approaches towards this problem have been developed:

- Supervised methods that are trained on annotated data.
- Unsupervised approaches where no labelled training data is necessary.
- Knowledge-based algorithms integrate external knowledge sources, for example dictionaries or thesauri.

From these methods the supervised methods provide the best performance so far. The disadvantage of the supervised approach is the reliance on annotated data, which is usually time-consuming and expensive to create.

The unsupervised methods on the other hand no not utilise such training data, usually only a plain textual corpus is sufficient. Another advantage of unsupervised methods is their ability to automatically discover the distinct senses. This property is preferred when the word senses differ from the general language, which can be the case for specialised domains, for example medical records. The main task for this family of algorithm is not the disambiguation aspect. It is the identification of senses, the term word sense induction and discrimination (WSID) has been established for this class of unsupervised methods.

In recent years multiple resources have become available that provide an overview over the field of word sense disambiguation in general. In 2007 a book has been published[431], which covers all areas of the WSD research field. At first the linguistic background on word senses is presented. Each chapter is written by selected experts in their respective research areas. All mainstream approaches are covered, the supervised, the unsupervised and

[431] E. Agirre, P. Edmonds, A. Kilgarriff, N. Ide, Y. Wilks, M. Stevenson, J. Gonzalo, L. Màrquez, F. Verdejo, G. Escudero, P. Buitelaar, B. Magnini, C. Strapparava, P. Vossen, P. Resnik, D. Martínez, M. Palmer, G. Rigau, H. T. Ng, H. T. Dang, R. Mihalcea, and T. Pedersen. *Word Sense Disambiguation: Algorithms and Applications.* Springer, 2007

the knowledge-based approaches. Additionally the domain specific WSD is presented together with an overview of existing applications of WSD.

More recently a survey in the ACM computing surveys journal has been published[432]. Again this survey tries to cover all different approaches to the WSD process. The more specialised word sense induction and discrimination methods are not described in detail. One important area that is covered in this survey is the evaluation methodology of WSD and WSID algorithms.

[432] R. Navigli. Word Sense Disambiguation: A Survey. *ACM Computing Surveys (CSUR)*, 41(2):10, 2009

In order to evaluate and compare different word sense disambiguation systems the SenseEval series has been initiated. In 1998 the first SenseEval took place, a competition where various approaches were systematically compared. Furthermore SenseEval also contributed data sets to the research community, which were very scarce until then. The results of the first SenseEval together with SenseEval-2 (2001) and SenseEval-3 (2004) are covered in great detail by Martinez[433] and Agirre et al[434].

[433] D. Martinez. *Supervised Word Sense Disambiguation: Facing Current Challenges*. PhD thesis, 2004

[434] E. Agirre, P. Edmonds, A. Kilgarriff, N. Ide, Y. Wilks, M. Stevenson, J. Gonzalo, L. Màrquez, F. Verdejo, G. Escudero, P. Buitelaar, B. Magnini, C. Strapparava, P. Vossen, P. Resnik, D. Martínez, M. Palmer, G. Rigau, H. T. Ng, H. T. Dang, R. Mihalcea, and T. Pedersen. *Word Sense Disambiguation: Algorithms and Applications*. Springer, 2007

The scope of the SenseEval series has been broadened to other tasks similar to the word sense disambiguation, like for example semantic role labelling. To reflect the new scope the 2007 edition of the competition was renamed to SemEval. In SemEval-2007 the word sense induction and discrimination was covered by an own dedicated task[435]. In this task the participants had to induce senses for 100 words (65 verbs and 35 nouns). The evaluation was split into two parts:

[435] E. Agirre and A. Soroa. Semeval-2007 task 02: Evaluating word sense induction and discrimination systems. *Proceedings of the 4rth International Workshop on Semantic Evaluations*, (June):7–12, 2007

- An unsupervised evaluation, where none of the participants were able to outperform the baseline, which was created by assuming that the ambiguous target words have just one sense (most frequent sense baseline).
- A supervised evaluation, where 3 of the 6 systems did provide a better performance than the baseline.

Ambiguous words with multiple senses are an intrinsic feature of natural languages. While humans usually cope with these ambiguities effortlessly, the exact distinction and definition of these senses is hard, even for lexicographers and linguists. This can be contributed to the varying degree of granularity of senses distinctions. Linguists have defined two main levels of granularity:

- The term homonym is used for words that have more than one sense, with no semantic overlap. The word "bank" for example has two senses: a) the financial institute and b) the side of a river. Both senses have entirely different meanings. These two senses also tend to co-occur with different words. This observation is the motivation of one the main approaches to automatically distinguish between meanings.
- The term polysemy is used for finer grained differences between the senses. In the case of the word "bank", there are at least two senses: a) the financial institute as abstract concept and b) a building in which an instance of a financial institute resides. These distinctions between the senses depend on the context and the domain. A clear separation is not always possible as in many cases senses form a continuum. Due to this property, this type of granularity is very hard to detect with an algorithmic approach.

The second type of sense granularity is also an area of research in the linguistic community. One of the results of this research is the insight that the sense distinction itself can be made starting from different viewpoints. These viewpoints have been developed while trying to systematically classify the reasons why words are perceived as similar. The similarity of words does also apply to the similarity of different senses of a single word.

- Taxonomic - Similarity is defined via a classification taxonomy, for example "cat" and "dog" are connected via the "pet" class within a taxonomy.

- Thematic - Similarity is found via the overlap of shared contexts, as for example the words “doctor” and “nurse” are similar as they are often used in the same contexts.
- Goal-derived - words become related as they are associated with the same goal, for example “airplane” and “hotel” are put in relation as they are required for the goal of going on vacation.
- Radial - Words with different semantics might still be considered similar as they share common properties in the real word. The distinction of senses the word term “bank” - financial institute vs. building - is an example where the word is reused but with a slightly different sense.

The perception of senses by human and the influence of the different viewpoints have been studied by cognitive scientists[436]. The results of this research is encouraging as it appears as if humans do treat polysemous word similar as homonyms. As a consequence the algorithmic approaches towards separating the senses of homonyms can serve as a starting point to develop methods to discriminate finer sense distinctions.

[436] D. Klein and G. Murphy. Paper has been my ruin: conceptual relations of polysemous senses. *Journal of Memory and Language*, 47(4):548–570, Nov. 2002

Many of the different approaches towards word sense induction and discrimination are based on two well known hypotheses: the distributional hypothesis and the strong contextual hypothesis (a description of these two concepts is given on page). In the case of words with multiple senses, the distributional hypothesis would lead to the assumption that each sense will differ by the words it co-occurs with. Based on these assumptions, word senses can be identified via finding distinct groups of words that reflect the context of each of these senses.

The main approaches towards word sense disambiguation can be grouped into three categories. These are the family of cluster based algorithms, approaches which operate on graph structures and methods resorting to statistical methods. In table 50 the presented algorithms are listed and grouped according their type of approach.

Algorithm	Category
Automatic Word Sense Discrimination	Clustering Based
SenseClusters	Clustering Based
I2R	Clustering Based
UPV-SI	Clustering Based
HyperLex	Graph Based
UoY	Graph Based
UBC-AS	Graph Based
Bayesian Word Sense Induction	Statistical

Table 50: Overview of selected word sense disambiguation algorithms, together with their classification according to their approach.

Automatic Word Sense Discrimination One of the most influential papers in the field of word sense discrimination is by Schutze in 1998[437]. This paper introduces the separation of the sense discrimination and the sense labelling task. The first task is crucial for any application that deals with multiple senses of words, the second task is often only optional.

[437] H. Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123, 1998

The approach presented in this paper is rooted on the Distributional Hypothesis and the Strong Contextual Hypothesis. Multiple senses of one word are associated with multiple distinct topical contexts. Identifying these contexts will lead to the individual senses. Contexts are defined by the text surrounding the occurrences of the target word within a textual corpus. Short documents that consist of a few sentences are generally regarded as a best practice for these contexts.

In this approach the target word contexts' are processed by computing the so called 2nd order co-occurrences. Next each context word - all words that occur in the context the target word - are replaced by a weighted vector, called word vector. The dimensions of this vector represent words that occur within a certain vicinity of the context word. A corpus with sufficiently many instances of all context words is required for this computation. The weight of the dimensions is calculated by exploiting statistical properties of the co-occurrences. The raw number of co-occurrences is a simple way to represent the strength of the association between words. Because of the fact that words are now represented by a weighted vector, the similarity between two words can be calculated using standard techniques. The cosine similarity is a very common example of these techniques therefore it serves as proxy for the semantic similarity of two words.

Each context of a target word instance can now be represented via the context words co-occurrence vectors. The original feature space of each context is very sparse as it is populated only by the individual words within the context. By using the co-occurrence vector representation, the feature space becomes more densely populated and additionally is also smoothed, as information from the training corpus is integrated. To honour the fact that not all words are equally suited to discriminate between topics, a weighting scheme is applied by merging all word vectors into the context feature space. The inverse document frequency has proven in many information retrieval applications to provide a good approximation to the discriminatory power for individual words.

The actual task of identifying individual senses is then transformed into a machine learning task, more precisely a clustering algorithm is employed to group the context vectors. The Buckshot algorithm[438] has been selected, which is a combination of an EM algorithm and agglomerative clustering. Additionally they processed the feature space by applying Singular Value Decomposition (SVD), where the input space is compressed by eliminating noise and redundant information. As training data the New York Times news service was utilised and they finally ended up with a corpus of about 60 million words, together with a smaller corpus of about 5 million words for testing. Evaluation was conducted on 10 ambiguous words and additionally on 10 artificial words which were created by concatenating two words with entirely different semantics. Each occurrence of one of the two words within the corpus was replaced by the artificial word.

[438] D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 318–329. ACM New York, NY, USA, 1992

In their evaluation they note that methods utilising the SVD into the processing of the feature space provided the best performance. The normalisation the feature space is most likely reason for this behaviour in the authors opinion. They reason that the clustering algorithm can provide better results if the input data is more uniformly distributed.

SenseClusters One of the most popular software packages for word sense induction and discrimination is SenseClusters[439]. This library is implemented in the Perl programming language and released under an open-source license. SenseClusters supports various options and parameters for tuning the algorithms. In this section only the configuration used for SemEval-07 will be covered[440].

[439] `http://senseclusters.sourceforge.net`

This approach taken by SenseClusters is similar to *Automatic Word Sense Discrimination*[441], were unsupervised clustering of the input data plays the central role. The machine learning community has developed a wide range of different algorithms for task of finding patterns in data. Probably the best known of these cluster algorithms is k-Means. This algorithm is well understood by the research community and has demonstrated in the past to provide good results in a variety of domains.

[440] T. Pedersen. Umnd2: Senseclusters applied to the sense induction task of senseval-4. *Proceedings of the 4th International Workshop on Semantic Evaluations*, (June):394–397, 2007

[441] H. Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123, 1998

Additionally the k-Means algorithms has a low run-time complexity. The main disadvantage of k-Means and related clustering algorithms is the number of clusters - k - to be found, which has to be defined in advance. Choosing the correct k is crucially important and in the case of word sense induction it directly correlates with the number of senses that will be detected. For the estimation of the correct number of senses for a specific word, the SenseClusters packages provides the Adapted Gap Statistic[442].

The configuration of the SenseClusters algorithms for SemEval-07 are based on the 2nd order co-occurrences. A window based approach was chosen as selection criteria for the words that build the co-occurrence vectors for the context words. All words within a window of 12 words around the context word instance are selected. For each of them, the Pointwise Mutual Information (PMI) is calculated and a threshold of 5.0 is applied. The motivation for this lower bound is to filter out all co-occurrences that are not results of an underlying semantic relationship, but were merely produced by stylistic or grammatical properties of the training corpus.

Each context of a target words instance is now represented via the context words co-occurrence vectors. The individual clusters that are the result of this algorithm should resemble distinct, thematically coherent contexts. Following the Distributional Hypothesis, each of these found topical contexts can be interpreted as individual sense of the target word.

In comparison with other approaches of SemEval-07, the SenseClusters approach fares very well. In the supervised evaluation only one system provided a better performance than SenseClusters. The authors of SenseClusters conclude[443] that the unsupervised approach towards words sense induction and discrimination requires sufficiently many training instances for the machine learning algorithms to detect distinctions between the senses.

I2R The I2R system[444] takes the same basic approach as the SenseClusters system. Both are based on the hypothesis, that distinct senses are coupled with different context they appear in. Again a machine learning algorithm is employed to find these clusters. Like the SenseClusters approach the number of clusters to be found is the central aspect of their system, for which they have created a stability criterion.

As features they took a subset of the ones described by Lee and Ng[445]:

- Part-of-speech of words within a window of 3 around the target word, where the position information is encoded into the feature
- All words from the whole context of the target word
- Words with a window of size 11 around the target word (collocations)

The SenseClusters system employs the k-Means as clustering algorithm, while the I2R system uses the sequential Information Bottleneck algorithm (sIB)[446]. The k-Means algorithm is a centroid based method and similarities between contexts are usually calculated via the cosine similarity, whereas sIB estimates the inter-context similarity via their conditional distribution.

To guess the correct number of clusters and therefore senses, they adapt the method of cluster validation. A clustering solution with a fixed number of clusters is considered valid, if the result is stable if applied to multiple random sub-samples of the input data. In other words, if the cluster result varies from run to run the cluster number does not fit the nature of the data.

The performance of their system was tested on the SemEval-2007 data set. For the supervised evaluation their approach performed best of all participants. The ability of their system to detect rarely used senses was

[442] T. Pedersen and A. Kulkarni. Automatic cluster stopping with criterion functions and the gap statistic. *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology companion volume: demonstrations* -, pages 276–279, 2006

[443] T. Pedersen and A. Kulkarni. Automatic cluster stopping with criterion functions and the gap statistic. *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology companion volume: demonstrations* -, pages 276–279, 2006

[444] Z.-y. Niu, D.-h. Ji, and C.-l. Tan. I2R: Three Systems for Word Sense Discrimination, Chinese Word Sense Disambiguation, and English Word Sense Disambiguation. *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval-2007)*, (June):177–182, 2007

[445] Y. K. Lee and H. T. Ng. An Empirical Evaluation of Knowledge Sources and Learning Algorithms for Word Sense Disambiguation. *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 41–48, 2002

[446] N. Slonim, N. Friedman, and N. Tishby. Unsupervised document classification using sequential information maximization. *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '02*, page 129, 2002

attributed to be the main reason for the relatively good performance.

UPV-SI The main idea behind the UPV-SI[447] is the enrichment of text by replacing each term by a weighted vector. They refer to this in their paper as self-term expansion.

For each term in the training corpus they calculated the co-occurrence statistics. As measure for the strength of the association between two words they have chosen the Pointwise Mutual Information (PMI). Additionally they applied a number of restrictions on the terms and the co-occurrences:

- Only word are processed that occurred at least 3 times in the corpus.
- A window size of 5 was used for the co-occurrences.

Again a clustering algorithm was employed to find the sense of the ambiguous target words. For this task they have chosen the KStar algorithm[448]. As input for this method the similarity matrix is required, which was populated using the Jaccard measure. In contrast to other clustering algorithms, the KStar algorithm is not initialised with a fixed cluster count, but a stopping criterion has to be provided. In the UPV-SI system, this is accomplished by calculating the average similarity of all sentences.

In comparison with other participants of SemEval-2007 the UPV-PI system provided a good performance, being the third best system in the supervised evaluation. The authors conclude that using a term selection method might further improve their system.

[447] D. Pinto, P. Rosso, and H. Jiménez-Salazar. UPV-SI: Word Sense Induction using Self Term Expansion. *acl.ldc.upenn.edu*, (June):430–433, 2007

[448] K. Shin and S. Han. Fast clustering algorithm for information organization. *Computational Linguistics and Intelligent Text Processing*, 2588:221–226, 2003

HyperLex Véronis states[449] that approaches based on term co-occurrence, like for example the algorithm proposed by Schutze[450] face the challenge of a high variation of the observed frequencies. Especially very rare event are not detected and are likely to be treated erroneously as noise. As an alternative the authors propose a method based on word co-occurrence graphs.

Such a graph can be constructed by inserting all words as nodes and all co-occurrences as edges between them. This is done for all words that co-occur with a single target term. Findings from the area of graph theory can then be applied. As the authors point out, such a co-occurrence graph exhibits properties of a small-world network. The World Wide Web is a prominent example of such a network.

For the evaluation they used a search engine to help gather data for a set of predefined target words. Only nouns and adjectives were retained and words with a frequency of less than 10 were filtered out. Co-occurrences were restricted to a single paragraph and discarded if their frequency was below 5.

The weight for the association between two words was based on the conditional probabilities of their occurrences. Again a threshold was applied on the weights as otherwise the graph would become too highly connected making the computation infeasible.

Once the co-occurrence graph has been build, high-density components are identified and their root hubs are calculated. Each of the identified components represents a single sense of the ambiguous target word.

Finally they compared the performance of their system with a baseline, where only most frequent sense was used. In their test the baseline obtained a precision of 73%, whereas the HyperLex algorithm provided a precision of 97%.

[449] J. Véronis. HyperLex: lexical cartography for information retrieval. *Computer Speech & Language*, 18(3):223—-252, 2003

[450] H. Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123, 1998

UoY The authors of the UoY system[451] note that graph based word sense discrimination systems have disadvantages on two regards:

- Words are treated as independent units, so concepts that are represented as a sequence of multiple words are not detected.

[451] I. Klapaftis and S. Manandhar. UOY: a hypergraph model for word sense induction \& disambiguation. *Proceedings of the 4th International Workshop on Semantic Evaluations*, (June):414–417, 2007

- Graph based methods operate on a 2 dimensional space.

To address these problems they propose an extension of existing graph based methods by using a hypergraph as their main data structure. Within such a graph the edges, called hyperedges, can carry more information than edges in a traditional graph. A single hyperedge does not only connect two nodes, but multiple nodes.

When building such a hypergraph they apply a number of restrictions:

- Only nouns are extracted from the text
- Each of these nouns must occur more often than a predefined threshold
- Another threshold controls the minimal size of a paragraph
- The size of a hyperedge can only vary between 2 and 4 nodes
- Each potential hyperedge must exceed a minimal frequency

Next a weight is assigned to each hyperedge, which is based on its frequency in relation to the frequencies of its sub-hyperedges. Finally hyperedges with a low weight are filtered out.

Graphs build using the HyperLex algorithm exhibit the small-world network properties. The same hold true for the hypergraphs of the UoY systems according to the authors. To identify senses of the ambiguous target word they modified the HyperLex algorithm to detect areas of high density and to identify the root hub nodes.

In the SemEval-2007 supervised evaluation the UoY was able to outperform the most-frequent-sense baseline for nouns. When applied on verbs, the performance of the hypergraph approach and the baseline was about the same.

UBC-AS The UBC-AS system[452] features a two stage clustering approach that operates on a co-occurrence graph. The initial stage of the algorithm is the construction of the context co-occurrence graph, where each word is represented by a node. Each pair of words that occur in the same context are connected via an weighted edge. The weight if the edge depends on the strength of the association of the two words.

Co-occurrence graphs are assumed to exhibit small-world properties. Within these graphs densely connected areas should represent distinct senses of an ambiguous word. Each of these areas is represented by a root hub. The UBC-AS system utilises two algorithms to identify these hubs: HyperLex[453] and HITS[454]. For each of the identified hubs a minimum spanning tree is computed from the co-occurrence graph. The authors note that this procedure is equivalent to a single-link clustering.

The next step is the assignment of the contexts to the hubs. The minimum spanning trees serve as a base to calculate the assignment weights. Although the information gained by this step is sufficient to assign each context to the hub with the highest weight, the authors remark that based on their experience the number of hub exceeds the number of senses. Therefore they have chosen to add another processing on top of the context-hub assignments, again by resorting to a clustering technique.

The input of the second stage is a context-context similarity matrix. Each context is represented via its assignment vector, which are then pairwise compared via the cosine similarity. To keep the matrix sparse only the top similarity scores for each context are preserved. At this point external information about the context similarity could be integrated into the matrix. The matrix is then clustered by the Markov clustering algorithm[455].

For the evaluation runs they used different configurations depending on the word class of the target word. For nouns the HyperLex algorithm

[452] E. Agirre and A. Soroa. UBC-AS: A Graph Based Unsupervised System for Induction and Classification. *Proceedings of the 4rth International Workshop on Semantic Evaluations*, (June):346–349, 2007

[453] J. Véronis. HyperLex: lexical cartography for information retrieval. *Computer Speech & Language*, 18(3):223—-252, 2003

[454] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999

[455] S. van Dongen. A Cluster algorithm for graphs. *Report - Information systems*, (10):1–40, 2000

is used to detect root hubs and the association weight is calculated via the conditional probability. For verbs the hubs are identified by applying the HITS algorithm and the χ^2 test defines the association strength. The higher frequency of verbs in relation to nouns has been cited as reason for the two different processing methods.

In the SemEval-2007 evaluation the UBC-AS system performed as well as the most-frequent-sense baseline.

Bayesian Word Sense Induction Brody and Lapata[456] proposed to approach word sense induction via a probabilistic generative model. Each sense of a word is drawn from a distribution, similar to distribution of topics in the Latent Dirichlet Allocation (LDA) algorithm[457]. The assumption of this approach is that the underlying distribution of senses follows a Dirichlet distribution. One potential downside of this algorithm is the sensitivity to the choice of the initial parameters for the calculation. As one of the key advantages of their approach they mention that it is possible to integrate various features into the word sense induction process.

[456] S. Brody and M. Lapata. Bayesian Word Sense Induction. *Computational Linguistics*, (April):103–111, 2009

[457] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003

In their evaluation they studied a range of different features extracted from the textual content:

- A window based selection of word surrounding the target word (they use two window sizes, 10 and 5 words).
- Collocations (word with distance of 1 in relation to the target word)
- Word n-grams and part-of-speed n-grams
- Dependency relations

These features were extracted by the RASP system[458]. For the word based features they have taken the lemmatised version of the word form. Their evaluation is conducted on the data set provided by the organisers of SemEval-2007, based on the Penn Treebank II corpus which is made up by articles from the Wall Street Journal (WSJ). In this data set each instance of a target word is surrounded by a short content, usually a few sentences. They restricted the disambiguation process on nouns only. To train their model they selected two different corpora:

[458] T. Briscoe, J. Carroll, and R. Watson. The second release of the RASP system. *Proceedings of the COLING/ACL on Interactive presentation sessions* -, (July):77–80, 2006

- The British National Corpus (BNC), were the topics differ from the test data set
- The WSJ corpus, but without the articles contained in the test data set

These data sources were processed to finally contain about 730 thousand instances of the target words for each of the two corpora.

In the first part of their evaluation they found that the optimal number of senses per word differs between the two training corpora. While for the WSJ training set the optimal number of senses was 4, which is about the same as the correct number of senses in the test data set, the best performance of their algorithm trained on the BNC data was twice as high.

In their next evaluation the influence of the different features were studied. The dependency relation feature had a negative effect on the overall performance. The reason for this unexpected is sparse nature of the feature according to the authors. The best performing configuration of their system has been achieved when only the two window based features were used for the sense induction.

Finally, they trained their model with the best performing combination of features and the WSJ corpus to compare its performance with the results published by the participants of SemEval-2007. They found that their system did provide better results that both the baseline and the best systems, although the difference between the best performing system of SemEval-2007 was not found to be statistically significant.

System For the second installment of the SemEval competition we developed a system for word sense induction and discrimination which makes heavy use of the functionality provided by the feature association framework. In this scenario the associations calculation do not only connect existing features, but also synthesise new features. For each input feature a number of nodes in the output graph are generated, where each of these new nodes represents a singe sense of the input term.

This WSID system exploits syntactic and semantic features based on the results of a natural language parser component. To achieve high robustness and good generalisation capabilities, we designed our system to work on a restricted, but grammatically rich set of features. Based on the results of the evaluations our system provides a promising performance and robustness.

The goal of the SemEval-2 word sense induction and discrimination task[459] is to identify the senses of ambiguous nouns and verbs in an unsupervised manner and to label unseen instances of these words with one of the induced senses. The most common approach towards this task is to apply clustering or graph partitioning algorithms on a representation of the words that surround an ambiguous target word[460,461]. We followed this approach by employing a clustering algorithm to detect the individual senses, but focused on generating feature sets different to the mainstream approach. Our feature sets utilise the output of a linguistic processing pipeline that captures the syntax and semantics of sentence parts closely related with the target word.

The base of our system is to apply a parser on the sentence in which the target word occurs. Contextual information, for example the sentences surrounding the target sentence, are currently not exploited by our system. To analyse the sentences we applied the Stanford Parser[462,463] (Version 1.6.2), which is based on lexicalised probabilistic context free grammars. This open-source parser not only extracts the phrase structure of a given sentence, but also provides a list of so called grammatical relations (typed dependencies)[464]. These relations reflect the dependencies between the words within the sentence, for example the relationship between the verb and the subject. The exploitation of grammatical dependencies for word sense disambiguation have already been investigated in the past[465].

The phrase structure and the grammatical dependencies are sources for the feature extraction stage. To illustrate the result of the parser and feature extraction stages we use an example sentence, where the target word is the verb “file”:

> Afterward , I watched as a butt-ton of good , but misguided people **filed** out of the theater , and immediately lit up a smoke .

The feature extract step consists of two separate steps, the first only takes the typed dependencies into consideration while the second extraction information out of the parse tree.

Grammatical Dependency Features The Stanford Parser provides 55 different grammatical dependency types. Figure 40 depicts the list of the grammatical dependencies identified by the Stanford Parser for the example sentence. Only a limited subset of these dependencies are selected to build the grammatical feature set. This subset has been defined based on preliminary tests on the trial data set. For verbs only dependencies that represent the association of a verb with prepositional modifiers and phrasal verb particles are selected (`prep`, `prepc`, `prt`). If the verb is not associated with a preposition or particle, a synthetic “missing” feature is added instead (`!prep`, `!prt`). For nouns the selected dependencies are the prepositions (for head nouns that are the object of a preposition) and noun compound modifiers (`pobj`, `nn`). If the noun is associated with a verb the grammatical dependencies of this verb are also

[459] S. Manandhar, I. P. Klapaftis, D. Dligach, and S. S. Pradhan. SemEval-2010 Task 14: Word Sense Induction & Disambiguation. In *Proceedings of SemEval-2*, Uppsala, Sweden, ACL, 2010

[460] Z.-y. Niu, D.-h. Ji, and C.-l. Tan. I2R: Three Systems for Word Sense Discrimination, Chinese Word Sense Disambiguation, and English Word Sense Disambiguation. *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval-2007)*, (June):177–182, 2007

[461] T. Pedersen. Umnd2: Senseclusters applied to the sense induction task of senseval-4. *Proceedings of the 4th International Workshop on Semantic Evaluations*, (June):394–397, 2007

[462] D. Klein and C. D. Manning. Accurate unlexicalized parsing. *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - ACL ’03*, pages 423–430, 2003

[463] D. Klein and C. Manning. Fast exact inference with a factored model for natural language parsing. *Advances in Neural Information Processing Systems*, pages 3—-10, 2003

[464] M. de Marneffe, B. MacCartney, and C. Manning. Generating typed dependency parses from phrase structure parses. In *LREC 2006*, 2006

[465] P. Chen, W. Ding, C. Bowes, and D. Brown. A fully unsupervised word sense disambiguation method using dependency knowledge. *Human Language Technology Conference*, 2009

added to the feature set.

relation	gov ▲	dep
pobj	as-5	butt-ton-7
det	butt-ton-7	a-6
prep	butt-ton-7	of-8
nsubj	filed-14	people-13
prt	filed-14	out-15
prep	filed-14	of-16
cc	filed-14	and-20
conj	filed-14	lit-22
advmod	lit-22	immediat...
prt	lit-22	up-23
dobj	lit-22	smoke-25
pobj	of-16	theater-18
pobj	of-8	good-9
amod	people-13	misguide...
det	smoke-25	a-24
det	theater-18	the-17
advmod	watched-4	Afterwar...
nsubj	watched-4	I-3
prep	watched-4	as-5
cc	watched-4	but-11
conj	watched-4	filed-14

Figure 40: List of grammatical dependencies as detected by the Stanford Parser for the example sentence.

The name of the dependency and the word (i.e. preposition or particle) are used to construct the grammatical features. The different features are weighted. The weights have been derived from their frequencies within the trial data set and listed in table 51. For the example sentence the extracted grammatical features are:

```
'out', 'of', prep, prt
```

Phrase Term Features The second set of features are generated from the sentence phrase structure. In figure 41 the parse tree for the example sentence is depicted.

Again we tried to keep the feature set as small as possible. Starting with the target word only phrases that are directly associated with the ambiguous word are selected. To identify these phrases the grammatical dependencies are exploited. For nouns as target words the associated verb is searched at first. Given a verb the phrases containing the head noun of a subject or object relationship are identified. If the verb is accompanied by a preposition, the phrase carrying the object of the preposition is added as well. All nouns and adjectives from these these phrases are then collected. The phrase words together with the verb, prepositions and particles are lemmatised using tools also provided by the Stanford Parser project.

The weights of the phrase term features are based on the frequency of the words within the training data set, where N is the total number of sentences and N_f is the number of sentences in which the lemmatised phrase term occurs in:

$$weight_f = log(\frac{N}{N_f + 1}) + 1 \tag{142}$$

In our example sentence the extracted phrase term features are:

```
of, misguided, file, theater, people, out
```

Feature	Weight
prepc, prt, nn, pobj	0.9
prep	0.45
!prep, !prt	0.5
'prepositions', 'particles'	0.97

Table 51: Weights of the grammatical features, which were derived from their distribution within the trial data set.

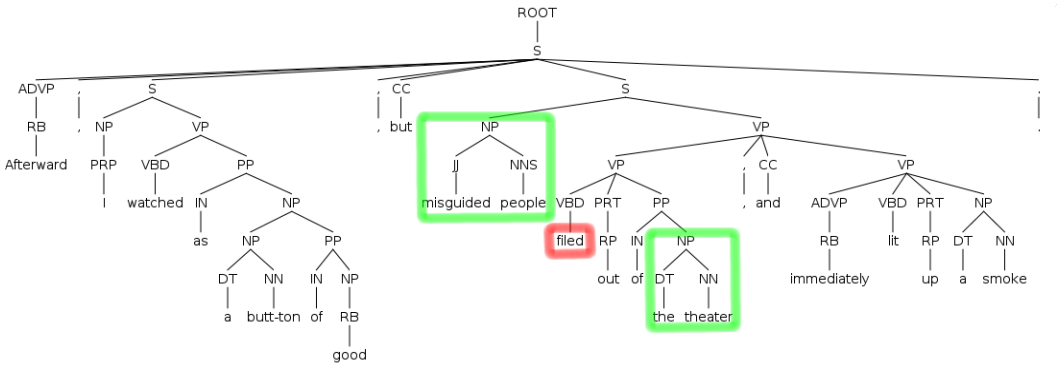

Figure 41: Phrase tree of the example sentence. The noun phrase "misguided people" is connected to the target word via the `nsubj` dependency and the phrase "the theater" is associated with the target verb via the `prep` and `pobj` dependencies.

The feature space of the phrase terms is expected to be very sparse. In addition phrase terms may have different representations but similar semantics. Therefore the phrase terms are optionally expanded with associated terms, where semantically similar terms should be associated with the same terms. Again the feature association framework comes to use for this task.

To calculate the statistics for term expansion we used the training data set (although other data sets might be more suitable for this purpose). The data set is split into sentences. Stop-words and rarely used words, which occur in less than 3 sentences, were removed. The remaining words were finally lemmatised. For a given phrase term the top 100 associated terms are used to build the feature set. The association weight between two terms is based on the Pointwise Mutual Information:

$$weight_{pmi}(t_i, t_j) = \frac{log_2(\frac{P(t_i|t_j)}{P(t_j)})}{log_2(\frac{1}{P(t_j)})} \tag{143}$$

For example the top 10 associated terms for `theater` are:

`theater.n, movie.n, opera.n, vaudeville.n, wnxt-abc.n, imax.n, orpheum.n, pullulate.v, projector.n, psychomania.n`

To detect the individual senses within the training data set we integrated unsupervised machine learning techniques into the feature association calculations. For each ambiguous word a matrix - $M_{|Instances|\times|Features|}$ - is created and a clustering algorithm is applied, namely the Growing k-Means[466]. This algorithm needs the number of clusters and centroids as initialisation parameters, where the initial centroids are calculated using a directed random seed finder[467]. We used the Jensen-Shannon Divergence function for the grammatical dependency features and the Cosine Similarity for the phrase term feature sets as relatedness function.

For each cluster number we re-run the clustering with different random initial centroids (30 times) and for each run we calculate a cluster quality criterion. The overall cluster quality criterion is the mean of all feature quality criteria, which are calculated based on the set of clusters the feature occurs in - C_f - the number of instances of each cluster - N_c - and the number of instances within a cluster where the feature occurs in - $N_{c,f}$:

$$FQC_f = \frac{weight_f}{|C_f|} * \sum_{c \in C_f} \frac{N_{c,f}}{N_c} \tag{144}$$

$$QC_{run} = \overline{FQC_f} \tag{145}$$

The cluster quality criterion is calculated for each run and the combination of the mean and standard deviations are then used to calculate a stability criterion to detect the number of clusters, which is based on the intuition that the correct cluster count yields the lowest variation of QC

[466] M. Daszykowski, B. Walczak, and D. L. Massart. On the optimal partitioning of data with K-means, growing K-means, neural gas, and growing neural gas. *Journal of chemical information and computer sciences*, 42(6):1378–89, 2002

[467] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2007

values:

$$SC_k = \frac{mean(QC)}{stdev(QC)} \tag{146}$$

Starting with two clusters the number of clusters is incremented until the stability criterion starts to decline. For the cluster number with the highest stability criterion the run with the highest quality criterion is selected as final clustering solution. The result of the sense induction processing is a list of centroids for the identified clusters. Alternative methods to identify the correct number of clusters rely on external information, not present in the clustering process[468].

[468] R. Kern, M. Zechner, and M. Granitzer. Model Selection Strategies for Author Disambiguation. In *IEEE Computer Society: 8th International Workshop on Text-based Information Retrieval in Procceedings of 22th International Conference on Database and Expert Systems Applications (DEXA 11)*, pages 155–160. IEEE, 2011

The final processing step is to assign an instance of an ambiguous word to one of the pre-calculated senses. The sentence with the target word is processed exactly like the training sentences to generate a set of features. Finally the word is assigned to the sense cluster with the maximum relatedness.

Results The system can be configured to use a combination of feature sets for the word sense induction and discrimination calculations:

a) *KCDC-GD*: Grammatical dependency features,

b) *KCDC-PT*: Phrase terms features,

c) *KCDC-PC*: Expanded phrase term features,

d) *KCDC-PCGD*: All training sentences are first processed by using the expanded phrase term features and then by using the grammatical dependency features with an additional feature that encodes the cluster id found by the phrase features.

In the evaluation we also submitted multiple runs of the same configuration to assess the influence of the random initialisation of the clustering algorithm. Judging from the results the random seeding has no pronounced impact and it influence should decrease when the number of clustering runs for each cluster number is increased.

All configurations found on average about 3 senses for target words in the test set (2.8 for verbs, 3.3 for nouns), with exception of the *KCDC-PT* configuration which identified only 1.5 senses on average. In the gold standard the number of senses for verbs is 3.12 and for nouns 4.46, which shows that the stability criterion tends to underestimate the number of senses slightly.

In figure 42 the results for three configurations are given. In the charts the y-axis represents the V-Measure[469].

[469] This measure is biased towards multiple senses, thus systems that create more fine grained senses will profit.

To compare the performance of the different configurations, one can use the average rank within the evaluation result lists. Judging from the rankings, the configurations that utilise the grammatical dependencies and the expanded phrase terms provide similar performance. The configuration that takes the phrase terms directly as features comes in last, which is expected due to the sparse nature of the feature representation and the low number of detected senses.

Comparing the performance of our system with the two baselines shows that our system did outperform the random baseline in all evaluation runs and the most frequent baseline (MFS) in all runs with the exception of the F-Score based unsupervised evaluation, where the MFS baseline has not been beaten by any system. Although none of our submitted configurations was ranked first in any of the evaluations, their ranking was still better than average, with the exception of the *KCDC-PT* configuration. In addition the result of our system remains stable in regard to the evaluation method, which can be seen as an indicator that focusing on a limited set of features results in a robust behaviour.

Another observation that can be made is the difference in performance between nouns and verbs. Our system, especially the grammatical dependency based configurations, is tailored towards verbs. Therefore the better

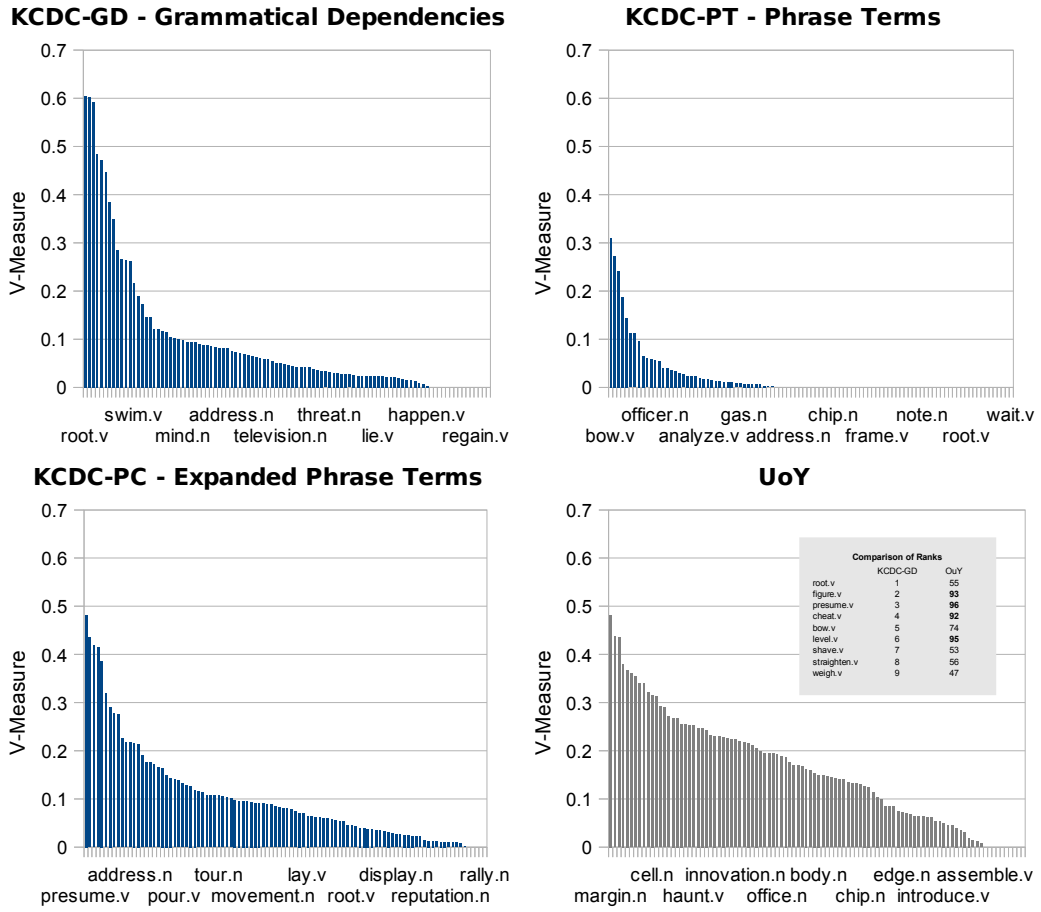

Figure 42: Results of the SemEval-2 evaluation for three different configurations of our system. Additionally a comparison with a different approach is given.

performance of verbs in the evaluation is in line with the expectations.

When looking at the results of the individual target words one can notice that for a set of words the quality of the sense detection is above average. For 16 of the 100 words a V-Measure of more than 30% in at least one configuration was achieved (average: 7.8%). The best performing target words are: `root.v`, `presume.v`, `figure.v`, `weigh.v`, `cheat.v`. This can be seen as indicator that our selection of features is effective for a specific group of words. When looking at the comparison in figure 42 one can see that the top ranked words of our system do not show up in the top results of an alternative approach. This results is very encouraging as it indicates that combining our features extraction with a more traditional approach should yield better overall results.

Conclusions In this section a word sense induction and discrimination system has been presented. The feature association framework is a central part of this system. It is first used to enrich the feature spaces and then to compute the different senses of ambiguous words. An unsupervised machine learning has therefore been integrated into the feature association framework in order to analyse the contextual associations.

This application scenario highlights the versatility of the feature association framework. Furthermore its run-time performance and its scalability properties will allow to apply the system on even larger data-set, which should prove beneficial for the quality of the results.

Conclusions

Knowledge discovery applications are diverse in nature. They vary in a number of areas, for example the amount of data to be analysed and the complexity of data itself. The feature association framework has been designed to cope with the high demands of such applications, while still being able to produce results of high quality. This chapter summarises the key insights gained from integrating the framework into real-world knowledge discovery applications to solve a diverse set of research problems.

In this work the feature association framework has been presented. The starting point for the development of this framework has been the computation of associations between features. This process has then iteratively been extended to allow additional processing as part of the process.

Due to the high theoretical run-time complexity of the feature association analysis, special attention has been given to ensure that the computations are still feasible for large amounts of features. This ensures that the high degree of flexibility provided feature association framework is still applicable for real world data sets.

To evaluate whether the goals have been reached, a number of knowledge discovery applications have been presented, where the feature association framework did play a vital part. Given the results from these scenarios one now can answer the raised research questions, assess the usefulness of the development approach and measure the run-time performance of the implementation:

- Do feature associations carry additional information?
- Did the incremental generalisations lead to the desired outcome?
- What are the key benefits of the feature association framework?
- What are the run-time costs of computing association networks?
- What are the limits of this approach?

Discussion

In this section the main questions are discussed based in the experience gained from the evaluations. At the end of this section is should become clear whether the development of the feature association has been an worthwhile endeavour.

Do Feature Associations Carry Additional Information?

Feature associations do contain additional information, which is not contained within the features themselves. But it is unclear whether this information is of use for today's knowledge discovery applications, especially given the additional processing overhead. Therefore this question is not only a theoretical one, but also practical aspects need to be taken into consideration.

In this work a number of knowledge discovery applications have been presented, which utilised the feature association framework. For some applications the feature association were more vital than for others.

For application within the Natural Language Processing domain it is more obvious that the additional information did benefit. While in other domains shallow approaches and simple features traditionally did work well, for example information retrieval. But in recent times it has become increasingly harder to improve information retrieval performance by tweaking only the algorithms. Exploiting the relationship between the features need to be taken into consideration if the performance should be further optimised.

Out of the six scenarios presented in the applications chapter, three directly make use of feature associations: *Tagging Structure*, *Cross-language Plagiarism* and *Word Sense Induction & Discrimination*. For these applications there is no direct alternative approach to using feature associations as they play a central part within the whole system. To conclude, feature association do not only theoretically carry information not contained in the features itself, but also help to solve problems of real-world knowledge discovery applications.

Did the incremental generalisations lead to the desired outcome?

The assessment of the usefulness of the approach does not allow a trivial answer. Therefore the question can be reformulated into two separate parts:

- The functional part addresses the aspect of whether the generalisations provide the expected level of flexibility. In short, did the generalisations go far enough?
- The non-functional part includes the justification of the efforts needed to map existing problem onto the formal requirements of the framework. In short, did the generations go too far?

Flexibility Although the iterative generalisations were always driven by existing problems at hand, there are indicators that the current state of the framework can be applied on a wide array of unseen use-cases. The first evidence for this is the fact, that the latest stage of development is still suitable to solve all previous problems. Thus, in the process of the cyclic improvements to add more flexibility no previously existing functionality has been lost.

In section the constrains imposed on the feature association functions were discussed. Each function that can be reformulated to match these constrains is eligible to be used to compute feature associations. Although

it is clear that there are limits to the number of possible function to be integrated, the most common similarity measures, statistical significance tests and measures from the field of information theory fit this scheme. Furthermore, the feature association function may access external information, for instance statistics not provided by the feature association framework itself.

Finally, the feature association framework has been used in applications, which are not covered here and in applications which has been developed since this thesis has been written[470]. Given the more flexibility is usually coupled with higher computational costs, it is likely that more flexibility will lead to higher run-times making some computations unfeasible.

[470] For example in an currently unpublished scenario the BM25 weighting function has been combined with the cosine similarity as part of a recommender system

Effort A higher degree of flexibility usually correlates with a higher amount of effort to accomplish more simple tasks. This rises the entrance burden of choosing a technology when tackling a problem at hand. For sure the flexibility and the high number of parameters of the feature association framework will discourage its use for first time users.

An application has been presented[471], where the feature association framework has been integrated by the developers of the application scenario. During this process no modifications to the framework appeared to be necessary. In addition the provided functionality has proven to be sufficient to address all issues raised by the application scenario.

[471] See page 199 for a the analysis of the distribution of tags within a broad folksonomy.

Judging by the feedback of the developers of all presented applications, there integration work did not pose any difficulty, once the basic principles has been understood. In practice the extraction of features will be the more tedious work.

What are the key benefits of the feature association framework?

The feature association framework has been developed with two main use cases in mind. First to provide a tool to help analyse the available features, and second to exploit the information contained in the relationship between features. Both use cases ultimately contribute in the work of a researcher to solve problems in the area of knowledge discovery.

Assess the potential impact of feature association When tackling a problem, it is often unclear in the beginning, which feature will prove beneficial. This is especially true for the information contained in the relation graph between the features. It turned out that for many problem settings, the assumption that features are independent from each other, does not deter performance. For other problems the most important information is not directly contained in the features itself, but in the relation to each other.

Therefore having the possibility to quickly check whether this is the case for a problem at hand is a clear benefit. Given the analytical tools provided by the feature association framework allow for an in depth evaluation of the network of features.

In the context of narrow folksonomies[472] the feature association framework has been used to analyse the relationship between different metadata of a folksonomy. This analysis has revealed that terms used in comment to photos often contain words specific to the author of the comment.

[472] Starting on page 201 this application is presented in detail.

Generate new features Traditionally many algorithms in the area of knowledge discovery operate on data-structures like vectors, matrices and tensors. Integrating additional information is hard to accomplish in such a setting. Having a tool at hand where such a information can be seamlessly integrated into an existing infrastructure is a considerable asset.

The information contained in the relations between the feature can be transformed to build a new set of features. In addition during the process of computing the association network, external sources and knowledge bases can be exploited. All these aspects are weld together to build new features, allowing succeeding processing steps to make use of this information.

In the case of query expansion[473] existing query terms have been used as input of the traversal of the association network. The result of this operation is then transformed into features, which are then treated like the input features by the search engine.

[473] Find a detailed presentation of this scenario starting with page 217.

Easy to integrate into existing solutions In order to conduct an additional analysis or processing step it is mandatory that the necessary modifications are kept at a minimum. This is especially true for the data-structures as they need to be flexible and easy to understand. The graph data-structure is well understood and much research has been conducted in multiple areas. For instance, the properties of graph like structures has been investigated. Many algorithms have been proposed that make use of these structures. It can be therefore assumed that the main input and output data-structures are suitable in the most cases.

For example, an existing monolingual retrieval application has been enriched with facilities to conduct cross-lingual query processing[474]. During this process the necessary adaptations have been kept at a minimum.

[474] See page 211 for the description of the query translation application scenario

Scaleable and high-performace It is not sufficient that an algorithms serves its purpose, it equally important that this algorithm satisfies the requirements in regard to its run-time costs. Ideally the time to perform the computations is as little dependent on the size of the data. For big data sets this property of an algorithms may become the deciding factor whether an approach is feasible or not.

The feature association framework has been developed to cope with big data sets. Furthermore the framework has been modelled to allow to be deployed within a distributed environment. For example the feature association computations are compatible with the map-reduce computing paradigm.

The most demanding application of the feature association framework has been to discover senses of ambiguous words[475]. Due to the high amount of data and the heavy processing the presented approach could only be realised due to the efficient algorithms and the lean data-structures.

[475] See page 229ff for details on the word sense induction and discrimination algorithm.

Feature associations as data-structure The output of the feature association can itself be used as data-structure. This way only the retrieval functionality is utilised. In some scenarios the output of existing tools need to be adapted to fit into the processing pipeline. To accomplish this task, the output data-structure - the feature association network - might be created not by the feature association framework itself, but might be filled by external sources.

For the detection of cross-lingual text reuse[476] a word-alignment algorithm has been used to create a feature association network, with is then stored and retrieved via the feature association framework. From the side of the application there has been no need for modifications.

[476] See page 222 for a detailed coverage of this application scenario.

Open-Source Although this is not a functional aspect, from an academic viewpoint it is certainly a benefit that not only the algorithm itself has been documented, but also the implementation is openly available. Making the source code available to the open-source community allows for external peer-review. In addition external contributors may add new functionality, as well as provide optimisations and to improve

the existing code-base. This way the future of the framework is not dependent on single persons or entity, but is open to input from the community.

What are the run-time costs of computing association networks?

The previous question as been directed towards functional aspects, while this question targets the run-time behaviour of the framework. Due to the multiple ways feature associations can be computed here only a broad overview is given.

In the most simple case, one is only interested how much information is contained in the relationship between the features of a data set. For a small data set, for instance the Brown corpus consisting of a few thousand textual documents, the run-time is less than one minute[477]. With larger data-sets, the computational costs rise. The Reuters RCV-1 corpus consists of 800,000 news messages, and about 12 million sentences. For this medium size data set computing the association network takes about 86 minutes[478].

The English Wikipedia consists of about 1.5 million articles (m) and over 5 million individual terms (n). The computational complexity of the association computations is theoretically bounded to $\mathcal{O}(mn^2)$. By exploiting the distribution of the features the amount of operations which need to be executed can be considerably reduced. Finally the computation of a feature association graph for the Wikipedia data-set takes 7.5 hours on a desktop class machine[479] for this data set.

Given that the computation of the feature associations can be conducted in a distributed manner[480], even larger data set are feasible to be processed.

[477] On page 183 the detailed figures are reported.

[478] See page 184 for more details about the data-set and the configuration.

[479] See page 186 for a complete run-time analysis.

[480] The section starting on page 137 is dedicated on how the feature association framework can be integrated into existing distributed execution environments, like for example map-reduce.

What are the limits of this approach?

This section does not address potential problems of the approach, but tries to discuss the limits of the feature association framework. These limits can be categories into two groups:

- Limits of the algorithmic approach
- Limits of the reference implementation of the algorithm

The inherent computational complexity of computing association between features is at least quadratic. Reducing these computational costs can only be successful when exploiting characteristics of the nature of the features at hand. In many cases the features will follow certain distributions. If the distribution is known, a specialised approach can provide considerable benefits in order to reduce the number of necessary operations.

In many knowledge discovery applications the features turn out to be power-law distributed[481]. If no such distribution is found for the data, one has to resort to a brute force approach or accept a loss of quality, if the data set exceeds a certain size.

Another potential bottleneck of the feature association computations may reside outside the main algorithm. One of the key features of the feature association framework is the ability to integrate external information sources. If these sources fail to be able to deliver the requested information in a timely manner, the computation as a whole will suffer.

As discussed in the section dedicated to the feature association function[482], not all functions may be eligible for their use within the feature association framework. Certainly such functions do exists, although none were encountered so far for any given application scenario. Reformulating an existing function into the structure required by the feature association functions appears to be the more probable cause to not consider integrating the feature association framework.

[481] Please see section starting on page 199, where this property is shown for a broad folksonomy whereby the feature association framework has been used for the computations.

[482] See pages 98ff for an in depth description of the constrains imposed on the feature association function

The limitations of the reference implementation are much less severe, as they can easily be counteracted by adding the required functionality. For instance the BM25 weighting function[483] has not been supported by the reference implementation. Therefore the implementation needed to be adapted.

In addition one is not restricted to use the reference implementation. For dedicated purpose one might choose to implement just a subset of the required functionality of the feature association framework.

[483] S. Robertson and M. Gatford. Okapi at TREC-4. In *Proceedings of the Fourth Text Retrieval Conference*, pages 73–97, 1996

Summary & Outlook

This section gives a short overview of the feature association framework in the context of knowledge discovery applications. The main concepts behind the algorithms of the framework and its implementation are highlighted. Finally the main results from the evaluations are presented.

Many knowledge discovery applications share the same basic structure. In the beginning plain data is collected and stored. Next this data is pre-processed, for example file-formats are converted and noise is removed. The output of this step is managed using data-structures, which depend on the data itself as well as on practical constrains, like for example available storage space and data access times.

The pre-processed data is then transformed into features. This step is crucial for many applications, as the features should provide a distilled view upon the raw data. Due to its importance, usually this feature engineering step is conducted in multiple stages.

In feature analysis phase one tries to identify patterns within the data, which may help solving the problem at hand. At the end of the feature engineering the data set has been transformed into a representation suitable for knowledge discovery algorithms. The output of these algorithms can then be either be fed to further processing stages or presented to users. Therefore visualisation techniques are applied to allow the user to navigate the results, often allowing interactive interactions. Alternatively reports are generated which sum up the results of the information extraction processes.

The feature association framework is a tool to analyse the relationship between features within a data set. It may provide benefits at two stages of this processing pipeline. At first, during the feature analysis phase, it contributes in the processes of gaining a better understanding of the available information contained in the data. Secondly, the information encoded in the relationships can be exploited and made available to the further processing steps. In this scenario, the feature association framework serves as an additional feature transformation stop within the pipeline of the knowledge discovery application.

Iterative Generalisations

The feature association framework has been developed in an iterative way. Guided by increasingly more complex problems, the framework and its algorithms have been improved to finally allow a high degree of flexibility. Starting with simple setup to compute the association between features, the framework has been generalised in three areas:

- Algorithmic
- Input and output data-structures
- Integration of external knowledge sources

The algorithms within the feature association framework have been iteratively been reworked allowing increasingly more flexibility. This is especially true for the functions that are responsible to compute the strength of the association between two features. There are multiple ways where custom algorithmic steps can be integrated into the computations. This way a complete knowledge discovery processing pipelines can be used internally to build the feature associations.

The most frequent usage of the feature association framework is it to integrate it into existing knowledge discovery applications. Therefore it needs to be flexible in terms of its input and output data sources. It should be easy to adapt the specific features of a data set to make them suitable to extract the most relevant relationships between features. The same rationale can be applied on the output data-structure, as succeeding processing steps need to access the results of the feature association computations.

The input data-structure originally started out as a simple matrix. In many knowledge discovery applications, the data is represented as matrix of instances and features[484]. This representation is also known as vector space model. Although its wide use this data-structure is limited and does not allow to store additional meta-data. This meta-data may prove to be beneficial and therefore the input data-structure has been remodelled to allow a more richer representation of the data.

[484] For textual data, instances will often be documents and features are usually words that occur in the documents.

A matrix can also be seen as a bi-partide graph. Therefore the logical generalisation of the input data-structure is to increase the level of abstraction of the graph structure. The input data-structure of the feature association framework is a n-partide graph, where n is not limited to a specific upper bound[485]. In addition each relation within this graph may carry additional meta-data. The structure of this meta-data is up to the knowledge discovery application and the used feature association functions.

[485] Alternatively this generalisation step can also be seen as a transition from 2D matrices to n-dimensional tensors.

The output data-structure holds the results of the feature association calculations. This data-structure has also undergone a generalisation step. In its most simple form, the output consists of pairs of features with an additional weight, which reflects the strength of the association. This output structure has been remodelled into a graph. By using this representation it is possible to efficiently traverse the network of associations.

Application, which integrate the feature association framework, may now choose to directly work with the association network or make use of the available retrieval operations. The most sophisticated retrieval method is based on a spreading activation scheme, where the graph of features is traversed in order to detect common patterns within the feature relationships.

The final generalisation aim at integrating external sources into the association computations. In some scenarios, the statistics gathers from the data set are not sufficient and one want to exploit existing knowledge bases. To address this use-case, the feature association functions may choose to make use of the internal statistics or resort to external sources for additional information about the features being processed.

Efficient Implementation

The feature association framework does not only exist as an abstract algorithm, but there is also an reference implementation. This implementation is available as open-source allowing external contributors to improve the existing code base. During the development the main focus has been to provide an implementation that is as efficient as possible without sacrificing the quality of the results. Having a efficient implementation should help to foster the acceptable of the feature association framework.

The algorithms have already been developed to provide a foundation for an efficient implementation. For example the fact the many feature follow a power law distribution is honoured by the way the input is read-out by the algorithm. Furthermore the algorithm is compatible with existing distributed execution paradigms. This should allow the computations to scale with the size of the data-sets.

A number of approaches has been followed to ensure a low run-time cost, which can be categorised into groups:

Heuristics A number of heuristics restrict the search scope to find relevant feature associations. For example, from a feature, which is just used a few times, no robust statistical properties can be deducted. The individual threshold of these heuristics can be controlled by the application and can therefore be tuned to the specific data set at hand.

Algorithmic Approach The central part of the implementation is to efficiently weld together information from many different parts of the data set. For contemporary computing infrastructures it is essential that as much data as possible it kept in memory and disc access should always be conducted in a sequential manner.

Storage Coupled with the algorithmic approach is the choice of how the data-structures are managed. The storage should be efficient and allow the data to be stored in a distributed manner. Therefore the implementation make use of existing open-source libraries, which have been used in many projects and proven to be reliable.

Diverse Applications

To demonstrate the usefulness of feature associations in knowledge discovery applications, a series of evaluation scenarios have been presented. These scenarios stem from three different domains in the field of knowledge discovery: social web, information retrieval and natural language processing. For each of these fields at least two applications have been selected.

Out of the domain of **Social Web** the application scenarios were:

Tag Recommender The feature association framework is used in this scenario as an item based recommender system. The data-set consists of images taken from a popular photo sharing site[486]. Additionally to the photos, additional meta-data like tags and user-ids were acquired. In this scenario the most basic configuration of the feature association framework has been applied.

[486] http://www.flickr.com/

Broad Folksonomy Analyzis The folksonomy of a popular bookmarking site[487] serves as base for the analysis of a broach folksonomy. In contrast to narrow folksonomies, these kind of folksonomies allow multiple users to tag specific resources. For the data set the resources are web-sites.

The outcome of this analysis has been that most tags follow a power law distribution. Furthermore the relationships between resources and tags has been studied.

[487] http://del.icio.us/

Narrow Folksonomy Analysis The relationship between photos and tags in contrast to the relationship of multiple types of meta-data has been studied in the next application scenario. A traditional folksonomies is defined as a tri-partide graph of users, resources and tags. This folksonomy has been expanded to contain other meta-data, for example the title, description and comments.

One of the key finding has been that the words people use for comments are to a large extend more specific to the author then to the photo which has been commented.

The second application domain has been **Information Retrieval**, where two application have been covered:

Query Translation In this application scenario, a collaboratively written encyclopedia serves as base for query translation. The application started out as an monolingual information retrieval system. The query translation processing has later been added to allow queries to be formulated in language other than the language of the document in the search index.

To assess the impact of the query translation step, it has been also applied on the original language of the document. Three different weighting strategies were evaluated and it has been found, that the additional translation does not seriously deter the information retrieval performance.

Query Expansion Global query expansion is a standard technique in the field of information retrieval where the original query of a user is automatically expanded by related terms before sending it to the search engine. These related terms may be taken from an existing knowledge base, or alternatively computed by the documents within the search index. The latter approach has been taken for the presented application.

The feature association framework has been utilised to analyse the documents and to compute semantically related words. During the retrieval stage, the input query is taken as starting point in the association network. The words with the highest association weight are then added to the query.

In the evaluation configurations which employ the query expansion step did outperform the configurations without query expansion.

The third domain is **Natural Language Processing**. Here two application have been presented which made use of the feature association framework.

Cross-Lingual Text Reuse Automatic plagiarism detection has stirred much interest in recent times. The technical base of such a system is an algorithm to uncover reuse of passages of text. In such a scenario the scalability of the system is the most demanding problem to solve. Each suspicious document needs to be compared with all documents in the database. This is true even if sophisticated heuristics to limit the search scope and thus the number of comparison operations needed.

In the case of cross-lingual plagiarism detection one has identify passages that were lifted from document written in a different language. The feature association framework has been used for such a system to retrieve translation candidates. By taking this approach the core algorithms of the text-reuse system need only be moderately be modified.

The evaluation did show that the translation processing is effective, both in terms of quality of the results, as of the time needed to retrieve the translation candidates.

Word Sense Induction and Discrimination The final presented application scenario is the most demanding in regard to the requirements imposed upon the feature association framework. Furthermore word sense induction and discrimination is a task, where human excel, but machines still struggle to provide usable results. To tackle this problem the word sense induction problem has been reformulated into a problem to find the most significant associations between an ambiguous word and accompanying function words.

Therefore the computations of the associations has been enhanced by integrating an unsupervised machine learning algorithm. The input has been modelled to contain not only the plain textual information, but also carries a number of meta-data, for example syntactic information gathered from linguistic analysis tools. In addition external sources have been adapted to integrate semantic information into the calculations. On top of the functional requirements, the feature association computations need to scale to the amount of data available.

The evaluation of the word sense induction and discrimination showed that this approach works sufficiently well for a number of ambiguous words. Especially satisfying is the fact that the words which demonstrated the best performance are complementary to the set of ambiguous words whose senses are best distinguished using a more traditional approach.

Future of the Feature Association Framework

There are a number of ways, the feature association framework can be further developed. On the other hand the generalisation steps appear to driven the core algorithms to point, where each added flexibility will have pronounced side-effects on the applicability. One of the main advantages of the feature association framework is its easy integration into existing applications. Therefore the further expansion of the core algorithms and data-structures does not appear to be the preferred way for future development.

In the area of distributed computing there is still room for improvement. As the available data-sets grow bigger, the demand for scalable solutions is growing. Much research will be directed at improving existing algorithms to cope with big data. Therefore the development of the feature association framework should anticipate improvements in the way distributed computing is conducted.

Relationship between features for certain will have their use in future applications. Therefore in addition to the presented application scenarios, there will be multiple use-cases, where the feature association framework will prove beneficial. This will be the case for extending existing applications as well as developing application with feature associations as their main component.

In the future the feature association framework should also broaden its scope for domains, which are not yet covered. For example, it will be integrated into application that are not directly linked with knowledge discovery. Whether this step will require fundamental changes to the algorithms remains to be seen.

Bibliography

[1] E. Agirre, P. Edmonds, A. Kilgarriff, N. Ide, Y. Wilks, M. Stevenson, J. Gonzalo, L. Màrquez, F. Verdejo, G. Escudero, P. Buitelaar, B. Magnini, C. Strapparava, P. Vossen, P. Resnik, D. Martínez, M. Palmer, G. Rigau, H. T. Ng, H. T. Dang, R. Mihalcea, and T. Pedersen. *Word Sense Disambiguation: Algorithms and Applications*. Springer, 2007.

[2] E. Agirre, O. L. D. Lacalle, and B. Country. UBC-ALM : Combining k-NN with SVD for WSD Eneko Agirre and Oier Lopez de Lacalle. *Computational Linguistics*, (June):342–345, 2007.

[3] E. Agirre and A. Soroa. Semeval-2007 task 02: Evaluating word sense induction and discrimination systems. *Proceedings of the 4rth International Workshop on Semantic Evaluations*, (June):7–12, 2007.

[4] E. Agirre and A. Soroa. UBC-AS: A Graph Based Unsupervised System for Induction and Classification. *Proceedings of the 4rth International Workshop on Semantic Evaluations*, (June):346–349, 2007.

[5] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Journal of Computer Science and Technology*, volume 15 of *VLDB '94*, pages 487–499. Morgan Kaufmann Publishers Inc., Morgan Kaufmann Publishers Inc., 1994.

[6] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[7] D. G. Altman, J. J. Deeks, and D. L. Sackett. Odds ratios should be avoided when events are common. *British Medical Journal*, 317(7168):1318, 1998.

[8] G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. Inf. Syst.*, 20(4):357–389, Oct. 2002.

[9] C. Anderson. The Long Tail. *Wired*, 12(10):170–177, 2004.

[10] J. R. Anderson. A spreading activation theory of memory. *Journal of verbal learning and verbal behavior*, 22(3):261–295, 1983.

[11] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2007.

[12] N. Ashish and C. A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *coopis*, page 160. Published by the IEEE Computer Society, 1997.

[13] A.-L. Barabási. *Linked: The New Science of Networks*, volume 71. Perseus, 2002.

[14] A. J. Bell. The co-information lattice. In *Proceedings of the Fifth International Workshop on Independent Component Analysis and Blind Signal Separation: ICA 2003*, 2003.

[15] R. M. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix prize. *KorBell Team's Report to Netflix*, 2007.

[16] I. Ben-Gal. Outlier detection for high dimensional data. *ACM Sigmod Record*, 2001.

[17] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[18] BNC Consortium. The British National Corpus. *Distributed by Oxford University Computing Services on behalf of the BNC Consortium*, 2007.

[19] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[20] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602. ACM, 2004.

[21] P. Boldi and S. Vigna. The webgraph framework ii: Codes for the world-wide web. In *Data Compression Conference, 2004. Proceedings. DCC 2004*, page 528. IEEE, 2005.

[22] S. Bordag. A comparison of co-occurrence and similarity measures as simulations of context. In *Proceedings of the 9th international conference on Computational linguistics and intelligent text processing*, CICLing'08, pages 52–63, Berlin, Heidelberg, 2008. Springer-Verlag.

[23] D. Borthakur. The hadoop distributed file system: Architecture and design. Technical report, 2007.

[24] M. Brand. Fast online svd revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining*, pages 37–46, 2003.

[25] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, 2006.

[26] J. S. Breese, D. Heckerman, C. Kadie, and Others. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

[27] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics, 1992.

[28] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[29] T. Briscoe, J. Carroll, and R. Watson. The second release of the RASP system. *Proceedings of the COLING/ACL on Interactive presentation sessions -*, (July):77–80, 2006.

[30] S. Brody and M. Lapata. Bayesian Word Sense Induction. *Computational Linguistics*, (April):103–111, 2009.

[31] C. Buckley, G. Salton, J. Allan, and A. Singhal. Automatic Query Expansion Using SMART: TREC 3. In *In Practice*, pages 69–80. NIST, 1994.

[32] J. A. Bullinaria and J. P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39(3):510, 2007.

[33] R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems*, 69(Supplement 32):175–186, 2000.

[34] L. Burnard. Reference Guide for the British National Corpus (XML Edition), 2007.

[35] S. Büttcher and C. L. A. Clarke. Indexing time vs. query time: tradeoffs in dynamic information retrieval systems. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 317–318. ACM, 2005.

[36] D. Carmel, E. Farchi, Y. Petruschka, and A. Soffer. Automatic query refinement using lexical affinities with maximal information gain. *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 02*, page 283, 2002.

[37] D. Carmel, H. Roitman, and N. Zwerdling. Enhancing cluster labeling using wikipedia. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '09*, pages 139–146, New York, New York, USA, 2009. ACM Press.

[38] C. Cattuto, A. Baldassarri, V. D. P. Servedio, and V. Loreto. Vocabulary growth in collaborative tagging systems. *arXiv*, 704, 2007.

[39] C. Cattuto, V. Loreto, and L. Pietronero. Semiotic dynamics and collaborative tagging. *Proceedings of the National Academy of Sciences*, 104(5):1461, 2007.

[40] J. Cendrowska. PRISM: An algorithm for inducing modular rules. *International Journal of ManMachine Studies*, 27(4):349–370, 1987.

[41] Y. S. Chan, H. T. Ng, and Z. Zhong. NUS-PT: Exploiting Parallel Texts for Word Sense Disambiguation in the English All-Words Tasks. *Computational Linguistics*, (June):253–256, 2007.

[42] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation OSDI'06*, 26(2):1–26, 2006.

[43] P. Chen, W. Ding, C. Bowes, and D. Brown. A fully unsupervised word sense disambiguation method using dependency knowledge. *Human Language Technology Conference*, 2009.

[44] E. Chi and T. Mytkowicz. Understanding the efficiency of social tagging systems using information theory. In *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, pages 81–88. ACM, 2008.

[45] K. W. Church and W. A. Gale. Poisson mixtures. *Natural Language Engineering*, 1(02):163–190, 1995.

[46] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[47] P. Clough, R. Gaizauskas, S. S. L. Piao, and Y. Wilks. METER: MEasuring TExt Reuse. *Annual Meeting of the ACL*, page 152, 2002.

[48] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT press, 2009.

[49] T. M. Cover, J. A. Thomas, and J. Wiley. *Elements of information theory*, volume 1. Wiley Online Library, 1991.

[50] F. Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6):453–482, 1997.

[51] F. Curatelli and O. Mayora-Ibarra. Competitive learning methods for efficient vector quantizations in a speech recognition environment. *MICAI 2000: Advances in Artificial Intelligence*, pages 108–114, 2000.

[52] D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 318–329. ACM New York, NY, USA, 1992.

[53] I. Dagan, F. Pereira, and L. Lee. Similarity-based estimation of word cooccurrence probabilities. In *of the 32nd annual meeting on*, pages 272–278, 1994.

[54] M. Daszykowski, B. Walczak, and D. L. Massart. On the optimal partitioning of data with K-means, growing K-means, neural gas, and growing neural gas. *Journal of chemical information and computer sciences*, 42(6):1378–89, 2002.

[55] M. de Marneffe, B. MacCartney, and C. Manning. Generating typed dependency parses from phrase structure parses. In *LREC 2006*, 2006.

[56] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, pages 1–13, 2008.

[57] J. Deeks, M. B. Bracken, J. C. Sinclair, H. T. O. Davies, M. Tavakoli, and I. K. Crombie. When can odds ratios mislead? *British Medical Journal*, 317(7166):1155, 1998.

[58] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[59] K. Dellschaft and S. Staab. Understanding the Dynamics in Tagging Systems. *Proceedings of the European Future ...*, 2009.

[60] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[61] I. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556. ACM New York, NY, USA, 2004.

[62] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98. ACM, 2003.

[63] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[64] L. Dolamic, C. Fautsch, and J. Savoy. UniNE at CLEF 2008: TEL, and Persian IR. *Evaluating Systems for Multilingual and Multimodal Information Access*, pages 178–185, 2009.

[65] N. Draper, H. Smith, and E. Pownell. *Applied regression analysis*. Wiley New York, 3rd editio edition, 1998.

[66] D. Dubin. The most influential paper Gerard Salton never wrote. *Status: published or submitted for publication*, 2004.

[67] E. N. Efthimiadis. Query Expansion. *Annual Review of Information Systems and Technology ARIST*, 31(1):121–187, 1996.

[68] A. El-Hamdouchi and P. Willett. Comparison of hierarchic agglomerative clustering methods for document retrieval. *The Computer Journal*, 32(3):220, 1989.

[69] S. Evert. *The statistics of word cooccurrences: word pairs and collocations*. Stuttgart, 2005.

[70] H. Fang, T. Tao, and C. Zhai. A formal study of information retrieval heuristics. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 49–56, New York, NY, USA, 2004. ACM.

[71] H. Fang and C. Zhai. An exploration of axiomatic approaches to information retrieval. *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 480–487, 2005.

[72] P. Fankhauser and W. Nejdl. Boilerplate Detection using Shallow Text Features. *Text*, 2010.

[73] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[74] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: an overview. In U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1, pages 1–34. American Association for Artificial Intelligence, 1996.

[75] C. Fellbaum. *WordNet: An electronic lexical database*. The MIT press, 1998.

[76] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics ACL 05*, 43(1995):363–370, 2005.

[77] P. A. Flach and N. Lachiche. Confirmation-Guided Discovery of First-Order Rules with Tertius. *Machine Learning*, 42(1):61–95, 2001.

[78] G. Forman. Choose Your Words Carefully : An Empirical Study of Feature Selection Metrics for Text Classification document Choose Your Words Carefully : An Empirical Study of Feature Selection Metrics for Text Classification. In *Proceedings of the 13th European Conference on Machine Learning (ECML '02)*, number August, 2002.

[79] G. E. Forsythe, M. A. Malcolm, and C. B. Moler. *Computer methods for mathematical computations*. Prentice Hall Professional Technical Reference, 1977.

[80] W. N. Francis and H. Kucera. Brown Corpus Manual. *Brown University*, 1979.

[81] D. Freitag, M. Blume, J. Byrnes, E. Chow, S. Kapadia, R. Rohwer, and Z. Wang. New experiments in distributional representations of synonymy. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 25–32. Association for Computational Linguistics, 2005.

[82] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science (New York, N.Y.)*, 315(5814):972–6, Feb. 2007.

[83] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software Practice and Experience*, 21(11):1129–1164, 1991.

[84] J. Fürnkranz. A study using n-gram features for text categorization. *Austrian Research Institute for Artificial Intelligence Technical Report OEFAI-TR-98-30 Schottengasse*, 3(1998):1–10, 1998.

[85] W. Gale and G. Sampson. Good-Turing smoothing without tears. *Journal of Quantitative Linguistics*, 2(3):217–237, 1995.

[86] W. A. Gale and K. W. Church. A Program for Aligning Sentences in Bilingual Corpora. *Computational Linguistics*, 19(1):75–102, 1991.

[87] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[88] S. Golder and B. A. Huberman. The structure of collaborative tagging systems. *Arxiv preprint cs/0508082*, 2005.

[89] M. L. Goldstein, S. A. Morris, and G. G. Yen. Problems with fitting to the power-law distribution. *The European Physical Journal B-Condensed Matter and Complex Systems*, 41(2):255–258, 2004.

[90] D. Graff, J. Kong, K. Chen, and K. Maeda. English gigaword. *Linguistic Data Consortium, Philadelphia*, 2003.

[91] S. Gries. Dispersions and adjusted frequencies in corpora. *International Journal of Corpus Linguistics*, 13(4):403–437, 2008.

[92] J. R. Gruser, L. Raschid, M. E. Vidal, and L. Bright. Wrapper generation for web accessible data sources. In *coopis*, page 14. Published by the IEEE Computer Society, 1998.

[93] J. Guyot, G. Falquet, S. Radhouani, and K. Benzineb. UNIGE Experiments on Robust Word Sense Disambiguation. In *Evaluating Systems for Multilingual and Multimodal Information Access*, 2008.

[94] K. Hacioglu. A lightweight semantic chunking model based on tagging. In *Proceedings of HLT-NAACL 2004: Short Papers on XX*, pages 145–148. Association for Computational Linguistics, 2004.

[95] M. A. Hall. Correlation-based Feature Selection for Machine Learning. *Methodology*, 21i195-i20(April):17, 1999.

[96] D. K. Harman. The TREC test collections. *TREC: Experiment and evaluation in information retrieval*, pages 21–52, 2005.

[97] P. Harrington. *Machine Learning in Action*. Manning Publication Co., 2012.

[98] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[99] A. Harth and S. Decker. Optimized index structures for querying rdf from the web. In *Web Congress, 2005. LA-WEB 2005. Third Latin American*, page 10. IEEE, 2006.

[100] H. S. Heaps. *Information retrieval: Computational and theoretical aspects*, volume 60. Academic Press New York, NY, 1978.

[101] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, volume II of *COLING '92*, pages 539–545. Association for Computational Linguistics Morristown, NJ, USA, Association for Computational Linguistics, 1992.

[102] M. Heckner, M. Heilemann, and C. Wolff. Personal information management vs. resource sharing: Towards a model of information behaviour in social tagging systems. In *Int'l AAAI Conference on Weblogs and Social Media (ICWSM)*, 2009.

[103] G. Hirst. Distributional Measures as Proxies for Semantic Relatedness. 2005.

[104] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10, 1962.

[105] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[106] N. Ide and C. Macleod. The American National Corpus: A Standardized Resource of American English. In *Proceedings of Corpus Linguistics 2001*, pages 274–280. Citeseer, 2001.

[107] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59–72, 2007.

[108] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag Recommendations in Folksonomies. *Knowledge Discovery in Databases PKDD 2007*, 4702(May):506–514, 2007.

[109] S. Y. Jianbo, S. X. Yu, and J. Shi. Multiclass Spectral Clustering. In *In International Conference on Computer Vision*, pages 313–319. In International Conference on Computer Vision, 2003.

[110] L. Jing, H. Huang, and H. Shi. Improved Feature Selection Approach TFIDF in Text Mining. In *Proceedings of the First International Conference on Machine Learning and Cybernetics*, volume 4, page 5, 2002.

[111] T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Machine learning: proceedings of the fourteenth International Conference (ICML'97)*, 1997.

[112] W. P. Jones and G. W. Furnas. Pictures of relevance: A geometric analysis of similarity measures. *Journal of the American society for information science*, 38(6):420–442, 1987.

[113] A. Juffinger, R. Kern, and M. Granitzer. Crosslanguage Retrieval based on Wikipedia Statistics. In *Proceedings of 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, 17-19 September, Aarhus, Denmark*, 2008.

[114] A. Juffinger, R. Kern, and M. Granitzer. Exploiting Cooccurrence on Corpus and Document Level for Fair Crosslanguage Retrieval. In *Working Notes for the CLEF 2008 Workshop, 17-19 September, Aarhus, Denmark*, 2008.

[115] A. Juffinger, T. Neidhart, A. Weichselbraun, G. Wohlgenannt, M. Granitzer, R. Kern, and A. Scharl. Distributed Web2.0 crawling for ontology evolution. In *Proc 2nd International Conference on Digital Information Management ICDIM 07*, volume 2, pages 615–620. Ieee, 2007.

[116] Juilland A., D. R. Brodin, and C. Davidovitch. Frequency dictionary of french words. 1970.

[117] R. Kern and M. Granitzer. Efficient linear text segmentation based on information retrieval techniques. In *MEDES '09: Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pages 167–171, 2009.

[118] R. Kern and M. Granitzer. German Encyclopedia Alignment Based on Information Retrieval Techniques. In M. Lalmas, J. Jose, A. Rauber, F. Sebastiani, and I. Frommholz, editors, *Research and Advanced Technology for Digital Libraries*, pages 315–326. Springer Berlin / Heidelberg, 2010.

[119] R. Kern, M. Granitzer, and V. Pammer. Extending Folksonomies for Image Tagging. In *WIAMIS 2008 , Special Session on Multimedia Metadata Management & Retrieval*. IEEE Computer Society, 2008.

[120] R. Kern, A. Juffinger, and M. Granitzer. Application of Axiomatic Approaches to Crosslanguage Retrieval. In *CLEF 2009 Workshop*, pages 142–149, 2009.

[121] R. Kern, A. Juffinger, and M. Granitzer. Evaluation of Axiomatic Approaches to Crosslanguage Retrieval. In *Multilingual Information Access Evaluation Vol. I Text Retrieval Experiments*, 2009.

[122] R. Kern, C. Körner, and M. Strohmaier. Exploring the Influence of Tagging Motivation on Tagging Behavior. In *Research and Advanced Technology for Digital Libraries*, pages 461–465, 2010.

[123] R. Kern, M. Muhr, and M. Granitzer. KCDC: Word Sense Induction by Using Grammatical Dependencies and Sentence Phrase Structure. In *Proceedings of SemEval-2*, Uppsala, Sweden, ACL, 2010.

[124] R. Kern, C. Seifert, and M. Granitzer. A hybrid system for German encyclopedia alignment. *International Journal on Digital Libraries*, 11(2):75–89, Sept. 2011.

[125] R. Kern, C. Seifert, M. Zechner, and M. Granitzer. Vote/Veto Meta-Classifier for Authorship Identification. In *In 3nd International Competition on Plagiarism Detection*, 2011.

[126] R. Kern, M. Zechner, and M. Granitzer. Model Selection Strategies for Author Disambiguation. In *IEEE Computer Society: 8th International Workshop on Text-based Information Retrieval in Procceedings of 22th International Conference on Database and Expert Systems Applications (DEXA 11)*, pages 155–160. IEEE, 2011.

[127] I. Klapaftis and S. Manandhar. UOY: a hypergraph model for word sense induction \& disambiguation. *Proceedings of the 4th International Workshop on Semantic Evaluations*, (June):414–417, 2007.

[128] D. Klein and C. Manning. Fast exact inference with a factored model for natural language parsing. *Advances in Neural Information Processing Systems*, pages 3—-10, 2003.

[129] D. Klein and C. D. Manning. Accurate unlexicalized parsing. *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - ACL '03*, pages 423–430, 2003.

[130] D. Klein and G. Murphy. Paper has been my ruin: conceptual relations of polysemous senses. *Journal of Memory and Language*, 47(4):548–570, Nov. 2002.

[131] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[132] W. Klieber, V. Sabol, R. Kern, M. Muhr, and M. Granitzer. Using Ontologies For Software Documentation. In *Proc Malaysian Joint Conference on Artificial Intelligence MJCAI2009*, 2009.

[133] W. Klieber, V. Sabol, M. Muhr, R. Kern, G. Öttl, and M. Granitzer. Knowledge discovery using the KnowMiner framework. In *IADIS International Conference Information Systems*, 2009.

[134] B. Klimt and Y. Yang. Introducing the Enron Corpus. *Machine Learning*, stitutep1:wwceasccaers2004168, 2004.

[135] B. Klimt and Y. Yang. The Enron Corpus : A New Dataset for Email Classification Research. In *Machine Learning ECML 2004*, volume 3201 of *Lecture Notes in Computer Science*, pages 217–226. Springer, 2004.

[136] P. Koehn. Europarl: A parallel corpus for statistical machine translation. *MT summit*, 5:12–16, 2005.

[137] A. Kontostathis and W. M. Pottenger. Detecting Patterns in the LSI Term-Term Matrix. *Workshop on the Foundation of Data Mining and Discovery IEEE International Conference on Data Mining*, 2002.

[138] C. Körner, R. Kern, H.-P. Grahsl, and M. Strohmaier. Of categorizers and describers: An evaluation of quantitative measures for tagging motivation. In *HT '10: Proceedings of the 21st ACM Conference on Hypertext and Hypermedia*, pages 157–166, 2010.

[139] S. Korsholm, M. Schoeberl, and A. P. Ravn. Interrupt handlers in Java. In *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 453–457. IEEE, 2008.

[140] K. Lang. 20 Newsgroups.

[141] Y. LeCun and C. Cortes. The MNIST Database of Handwritten Digits.

[142] L. Lee. Measures of Distributional Similarity. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 25—-32, 1999.

[143] Y. K. Lee and H. T. Ng. An Empirical Evaluation of Knowledge Sources and Learning Algorithms for Word Sense Disambiguation. *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 41–48, 2002.

[144] B. Lemaire and G. Denhière. Incremental construction of an associative network from a corpus. In *Proceedings of the 26th Annual Meeting of the Cognitive Science Society*, pages 825–830. Citeseer, 2004.

[145] D. D. Lewis. Reuters-21578, distribution 1.0. 1997.

[146] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[147] P. Liang, B. Taskar, and D. Klein. Alignment by agreement. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 104–111. Association for Computational Linguistics, 2006.

[148] C. Y. Lin and E. Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 71–78. Association for Computational Linguistics, 2003.

[149] S. Lindstaedt, V. Pammer, R. Moerzinger, R. Kern, H. Mülner, and C. Wagner. Recommending tags for pictures based on text, visual content and user context. In *Proceedings of the Third International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 506–511. IEEE Computer Society Press, 2008.

[150] W. Lowe and S. McDonald. The direct route: Mediated priming in semantic space. In *Proceedings of the 22nd Annual conference of the Cognitive Science Society*, pages 675–680. Citeseer, 2000.

[151] M. Lux, M. Granitzer, and R. Kern. Aspects of Broad Folksonomies. In *18th International Conference on Database and Expert Systems Applications DEXA 2007*, pages 283–287. Ieee, 2007.

[152] U. V. Luxburg. A Tutorial on Spectral Clustering. Technical Report March, Max–Planck–Institut für biologische Kybernetik, 2007.

[153] S. Maji and J. Malik. Fast and accurate digit classification. Technical report, Citeseer, 2009.

[154] S. Manandhar, I. P. Klapaftis, D. Dligach, and S. S. Pradhan. SemEval-2010 Task 14: Word Sense Induction & Disambiguation. In *Proceedings of SemEval-2*, Uppsala, Sweden, ACL, 2010.

[155] T. Mandl and C. Womser-Hacker. The effect of named entities on effectiveness in cross-language information retrieval evaluation. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 1059–1064, New York, NY, USA, 2005. ACM.

[156] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, volume 61. Cambridge University Press, 2008.

[157] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):330, 1993.

[158] B. Markines, C. Cattuto, F. Menczer, D. Benz, A. Hotho, and G. Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *Proceedings of the 18th international conference on World wide web*, pages 641–650. ACM, 2009.

[159] C. Marlow, M. Naaman, D. M. Boyd, and M. Davis. HT06, tagging paper, taxonomy, Flickr, academic article, to read. In U. K. Wiil, P. J. Nürnberg, and J. Rubart, editors, *Proceedings of the seventeenth conference on Hypertext and hypermedia HYPERTEXT 06*, volume 27 of *HYPERTEXT '06*, pages 31–40. ACM, 2006.

[160] D. Martinez. *Supervised Word Sense Disambiguation: Facing Current Challenges*. PhD thesis, 2004.

[161] D. Martinez and E. Agirre. One sense per collocation and genre/topic variations. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, pages 207–215, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[162] Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157–170, 2004.

[163] H. Maurer, F. Kappe, and B. Zaka. Plagiarism - A Survey. *Journal Of Universal Computer Science*, 12(8):1050–1084, 2006.

[164] W. J. McGill. Multivariate information transmission. *Psychometrika*, 19:97–116, 1954.

[165] C. Middleton and R. Baeza-Yates. A comparison of open source search engines. Technical report, 2008.

[166] B. L. Milenova and M. M. Campos. O-cluster: scalable clustering of large high dimensional data sets. 2002.

[167] M. Muhr, R. Kern, and M. Granitzer. Analysis of structural relationships for hierarchical cluster labeling. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '10*, page 178, New York, New York, USA, 2010. ACM Press.

[168] M. Muhr, R. Kern, M. Zechner, and M. Granitzer. External and Intrinsic Plagiarism Detection using a Cross-Lingual Retrieval and Segmentation System Lab Report for PAN at CLEF 2010. In *2nd International Competition on Plagiarism Detection*, 2010.

[169] M. Muhr, M. Zechner, R. Kern, and M. Granitzer. External and Intrinsic Plagiarism Detection Using Vector Space Models. In *Proceedings of the SEPLN'09 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse*, 2009.

[170] R. Navigli. Word Sense Disambiguation: A Survey. *ACM Computing Surveys (CSUR)*, 41(2):10, 2009.

[171] Z.-y. Niu, D.-h. Ji, and C.-l. Tan. I2R: Three Systems for Word Sense Discrimination, Chinese Word Sense Disambiguation, and English Word Sense Disambiguation. *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval-2007)*, (June):177–182, 2007.

[172] J. Nivre, J. Hall, and J. Nilsson. MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, pages 2216–2219. Citeseer, Citeseer, 2006.

[173] D. W. Oard and B. J. Dorr. A Survey of Multilingual Text Retrieval. *Electrical Engineering*, (UMIACS-TR-96-19):1–31, 1996.

[174] F. J. Och and H. Ney. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51, 2003.

[175] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of OSIR*, volume 2006. Citeseer, 2006.

[176] C. D. Paice. Another Stemmer. *SIGIR Forum*, 24(3):56–61, 1990.

[177] M. Patel, J. A. Bullinaria, and J. P. Levy. Extracting semantic representations from large text corpora. In *Proceedings of the 4th Neural Computation and Psychology Workshop*, pages 199–212. Citeseer, 1998.

[178] P. Pecina and P. Schlesinger. Combining association measures for collocation extraction. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 651–658, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[179] T. Pedersen. Umnd2: Senseclusters applied to the sense induction task of senseval-4. *Proceedings of the 4th International Workshop on Semantic Evaluations*, (June):394–397, 2007.

[180] T. Pedersen and A. Kulkarni. Automatic cluster stopping with criterion functions and the gap statistic. *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology companion volume: demonstrations -*, pages 276–279, 2006.

[181] L. Philips. Hanging on the metaphone. *Computer Language Magazine*, 7(12):38–44, 1990.

[182] L. Philips. The double metaphone search algorithm. *CC PLUS PLUS USERS JOURNAL*, 18(6):38–43, 2000.

[183] G. Piatetsky-Shapiro. Knowledge Discovery in Real Databases: A Report on the IJCAI-89 Workshop. *AI Magazine*, 11(5)(5):68—-70, 1991.

[184] D. Pinto, P. Rosso, and H. Jiménez-Salazar. UPV-SI: Word Sense Induction using Self Term Expansion. *acl.ldc.upenn.edu*, (June):430–433, 2007.

[185] J. M. Ponte and B. W. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

[186] M. F. Porter. Snowball: A language for stemming algorithms, 2001.

[187] M. Potthast, B. Stein, and T. Holfeld. Overview of the 1st International Competition on Wikipedia Vandalism Detection. *Notebook Papers of CLEF 2010 LABs and Workshops*, pages 22–23, 2010.

[188] U. Quasthoff and C. Wolff. The poisson collocation measure and its applications. 2002.

[189] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[190] J. R. Quinlan. *C4. 5: programs for machine learning*. Morgan kaufmann, 1993.

[191] R. Rapp. The computation of word associations: comparing syntagmatic and paradigmatic approaches. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, number 1992, pages 1–7. Association for Computational Linguistics Morristown, NJ, USA, 2002.

[192] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. *Proceedings of the conference on empirical methods in natural language processing*, 1:133–142, 1996.

[193] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[194] B. Riordan and M. N. Jones. Comparing semantic space models using child-directed speech. *Entropy*, 20:200, 2000.

[195] S. Robertson and M. Gatford. Okapi at TREC-4. In *Proceedings of the Fourth Text Retrieval Conference*, pages 73–97, 1996.

[196] T. G. Rose, M. Stevenson, and M. Whitehead. The Reuters Corpus Volume 1-from yesterday's news to tomorrow's language resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 29–31. Citeseer, 2002.

[197] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[198] M. Sanderson. Word sense disambiguation and information retrieval. *Intelligent Information Management*, 01(02):122–127, 1994.

[199] A. Sarkar, P. H. Garthwaite, and A. De Roeck. A Bayesian mixture model for term re-occurrence and burstiness. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, CONLL '05, pages 48–55, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[200] P. Scheir, C. Ghidini, R. Kern, M. Granitzer, and S. N. Lindstaedt. ARS/SD: An Associative Retrieval Service for the Semantic Desktop. *Networked Knowledge-Networked Media*, pages 95–111, 2009.

[201] H. Schmid. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of International Conference on New Methods in Language Processing*, volume 12 of *Studies in Computational Linguistics*, pages 44–49. Manchester, UK, 1994.

[202] B. Scholkopf, A. Smola, and K.-R. Muller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10(5):1299–1319, 1998.

[203] H. Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123, 1998.

[204] H. Schütze and J. Pedersen. A cooccurrence-based thesaurus and two applications to information retrieval. *Information Processing & Management*, 33(3):307–318, May 1997.

[205] C. Seifert, V. Sabol, and W. Kienreich. Stress Maps: Analysing Local Phenomena in Dimensionality Reduction Based Visualizations. In *European Symposium Visual Analytics Science and Technology (EuroVAST)*, 2010.

[206] G. Seni and J. F. Elder. Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions. *Statistics*, 2(1):1–126, 2010.

[207] K. Shin and S. Han. Fast clustering algorithm for information organization. *Computational Linguistics and Intelligent Text Processing*, 2588:221–226, 2003.

[208] A. Silberschatz and A. Tuzhilin. On Subjective Measures of Interestingness in Knowledge Discovery. In U. M. Fayyad and R. Uthurusamy, editors, *Evaluation*, pages 275–281. AAAI Press, 1995.

[209] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *ACM SIGIR Forum*, 33(1):6–12, 1999.

[210] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '96, pages 21–29, New York, NY, USA, 1996. ACM.

[211] N. Slonim, N. Friedman, and N. Tishby. Unsupervised document classification using sequential information maximization. *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '02*, page 129, 2002.

[212] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pages 261–265, 1988.

[213] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management - CIKM '07*, page 623, New York, New York, USA, 2007. ACM Press.

[214] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 34, page 35. Citeseer, 2000.

[215] M. Steyvers and T. Griffiths. Probabilistic Topic Models. *Handbook of latent semantic analysis*, 427, 2007.

[216] A. Stolcke. SRILM-an extensible language modeling toolkit. In *Seventh International Conference on Spoken Language Processing*, volume 3, pages 901–904. Citeseer, 2002.

[217] M. Strohmaier, D. Helic, D. Benz, C. Koerner, and R. Kern. Evaluation of Folksonomy Induction Algorithms. *Transactions on Intelligent Systems and Technology (ACM TIST)*, 2011.

[218] M. Strohmaier, C. Körner, and R. Kern. Why do Users Tag? Detecting Users' Motivation for Tagging in Social Tagging Systems. In *International AAAI Conference on Weblogs and Social Media (ICWSM2010)*, number Coates 2005, 2010.

[219] R. L. Taft. *Name search techniques*. Bureau of Systems Development, New York State Identification and Intelligence System, 1970.

[220] P. N. Tan, M. Steinbach, V. Kumar, and Others. *Introduction to data mining*. Pearson Addison Wesley Boston, 2006.

[221] E. Terra and C. L. A. Clarke. Frequency estimates for statistical word similarity measures. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 165–172, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[222] E. Terra and C. L. A. Clarke. Scoring missing terms in information retrieval tasks. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 50–58. ACM, 2004.

[223] E. Terra and C. L. A. Clarke. Comparing query formulation and lexical affinity replacements in passage retrieval. In *Proceedings of the ACM-SIGIR workshop on methodologies and evaluation of lexical cohesion techniques in real-world applications (ELECTRA 2005)*, pages 11–17. Citeseer, 2005.

[224] P. D. Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the twelfth european conference on machine learning (ecml-2001)*, pages 491–502, 2001.

[225] S. van Dongen. A Cluster algorithm for graphs. *Report - Information systems*, (10):1–40, 2000.

[226] R. Van Meteren and M. Van Someren. Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*. Citeseer, 2000.

[227] J. Véronis. HyperLex: lexical cartography for information retrieval. *Computer Speech & Language*, 18(3):223—-252, 2003.

[228] E. Voorhees. Natural language processing and information retrieval. *Information Extraction*, page 724, 1999.

[229] E. M. Voorhees. Query expansion using lexical-semantic relations. In W. B. Croft and C. J. V. Rijsbergen, editors, *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 61–69. Springer-Verlag New York, Inc., 1994.

[230] T. V. Wal. Explaining and showing broad and narrow folksonomies, 2005.

[231] D. J. Watts. *Six Degrees: The New Science of Networks*. Vintage, 2004.

[232] K. Wei, J. Huang, and S. Fu. A survey of e-commerce recommender systems. In *Service Systems and Service Management, 2007 International Conference on*, pages 1–5. IEEE, 2007.

[233] J. W. J. Williams. Algorithm 232: heapsort. *Communications of the ACM*, 7(6):347–348, 1964.

[234] C. Willners and A. Holtsberg. Statistics for sentential co-occurrence. *Working Papers, Lund University, Dept. of Linguistics*, 2001.

[235] J. Xu and W. B. Croft. Query expansion using local and global document analysis. *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 96*, (Zurich, Switzerland):4–11, 1996.

[236] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *In Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 412–420, 1997.

[237] D. Yarowsky. One sense per collocation. In *Proceedings of the workshop on Human Language Technology*, HLT '93, pages 266–271, Morristown, NJ, USA, 1993. Association for Computational Linguistics.

[238] C. X. Zhai. Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, 1(1):1–141, 2008.

[239] G. K. Zipf. *Human Behaviour and the Principle of Least-Effort.* Addison-Wesley, 1949.