Master's Thesis

Semantic Recommendation Systems in Research 2.0

Patrick Thonhauser
patrick.thonhauser@gmail.com

Graz University of Technology

Institute for Information Systems and Computer Media
Advisor: Assoc.Prof. Dipl.-Ing. Dr.techn Martin Ebner
Co-Advisor: Selver Softic, MSc

Graz, July 2012

Masterarbeit

Semantische Empfehlungssysteme für Research 2.0

Patrick Thonhauser
patrick.thonhauser@gmail.com

Technische Universität Graz

Institut für Informationssysteme und Computer Medien
Betreuer: Univ-Doz. Dipl.-Ing. Dr.techn. Martin Ebner
Mitbetreuer: Selver Softic, MSc

Graz, Juli 2012

Abstract

In this age of data processing, data mining, topic related data-search and the recommendation of data and information is already a common, wide spread as well as profitable and trend-setting topic. The content of this master's thesis deals with a recommender system, which recommends users from a specific social networks to each other. These recommendations are based on the analysis of text artefacts and aim to connect similar minded or interested people. By applying many existing methods and technologies, a system for making recommendations in the form of user profiles of social platforms was developed. The fact that the amount of internet users has now exceeded the one billion user mark, made developing an appropriate strategy to limit this huge set of people to a manageable and promising subset, one of this thesis' main tasks. Finding appropriate metrics for accomplishing this filtering task was essential for limiting the amount of potential recommendations. The analysis of user profiles was mainly realized by analyzing user generated text. As part of this, it was possible to find a way to classify content whilst staying aware of its context and therefore, filtering scientific content or finding people that produce such content. One very important task of this thesis and the implemented proof of concept system, was to find and identify metrics and to develop comparable ratios, which enabled the system to classify user profiles at a satisfying rate of error. The systems and strategies that were developed delivered unmistakably positive results. This leads to the opinion that the system developed, along with the methods and strategies, would prove useful for future Semantic Recommender Systems for Research 2.0 and beyond.

Keywords *Recommender Systems, Semantic, Twitter, Social Media, Data Mining, Classification, Part of Speech Tagging, Research 2.0, Linked Data, Semantic Web, Python, Knowledge Discovery in Databases, Microblogging*

Zusammenfassung

Gezielte und themenbezogene Datensuche- und die Empfehlung von Daten und Information, ist in diesem Zeitalter der Datenverarbeitung bereits ein alltägliches, breitgefächertes, aber auch lukratives und richtungsweisendes Thema. Der Inhalt dieser Arbeit beschäftigt sich mit einem Empfehlungssystem, dessen Aufgabe es ist, Benutzer eines bestimmten sozialen Netzwerkes einander zu empfehlen. Diese Empfehlungen basieren auf der Analyse und Klassifizierung von Text und haben die Aufgabe, ähnlich oder sogar gleich interessierte Benutzer einander bekannt zu machen. Mit Hilfe von vielen bestehenden Technologien und Methoden wurde so ein System geschaffen, das es einem Benutzer ermöglicht, Empfehlungen in Form von anderen Benutzern der selben sozialen Plattform auf der er oder sie sich bewegt, zu erhalten. Da die Anzahl von Menschen die das Internet benutzt die Milliardengrenzen bereits bei Weitem überschritten hat, war eines der Hauptziele im Zuge dieser Arbeit eine Strategie zu finden, dieses riesige Menge an potentiellen Empfehlungen so gut wie möglich auf eine überschaubare und möglichst vielversprechende Menge einzuschränken. Um diese Einschränkung möglich zu machen, war das finden von Metriken zur Beschränkung Empfehlungszielgruppe eines der essenziellen Punkte während der Konzeptionierung des entwickelten Systems. Die Analyse von Benutzerprofilen wurde hauptsächlich an Hand der Analyse von benutzergeneriertem Text realisiert und im Zuge dessen, konnte ein Weg gefunden werden, um die Thematik des Inhaltes zu klassifizieren und so vor allem wissenschaftliche Inhalte bzw. Benutzer die wissenschaftliche Inhalte verbreiten, aus der Masse heraus gefiltert werden. Das Hauptaugenmerk dieser Arbeit und des implementierten *Proof of Concept* Systems lag auf dem Entwickeln von Kennzahlen und Erkennen von vergleichbaren Merkmalen um Benutzer zu einem zufriedenstellenden Anteil an Falschklassifikation als potentiell interessant oder uninteressant klassifizieren zu können. Es konnten durchaus positive Ergebnisse erzielt werden, die das entwickelte System bzw. die entwickelten Strategien für zukünftige Realisierungen von Semantischen Empfehlungssystemen für Research 2.0 und darüber hinaus sehr von Nutzen sein können.

Keywords *Recommender Systems, Semantic, Twitter, Social Media, Data Mining, Classification, Part of Speech Tagging, Research 2.0, Linked Data, Semantic Web, Python, Knowledge Discovery in Databases, Microblogging*

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.

Place, 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 entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum	Unterschrift

Acknowledgements

This master's thesis would not have been possible without the guidance and help of several individuals, starting with the people who directly influenced this thesis.

Martin Ebner and Selver Softic, who always had an open ear for all my questions and did their very best in guiding me through this thesis and the associated work. Judith Dohr for giving me great and highly engaging advice and assistance at the beginning of the writing process. Also a big thank you to all the people who volunteered themselves as testers for my proof of concept system.

The following people did not have any direct association with this thesis, but were essential to me throughout the time I attended university.

All my family members who supported me during my studies, in particular my parents who always believed in me and offered me the opportunity to study. The great support of my beloved girlfriend Sonja Guntschnig and also her family. Some of my student colleagues who attended most of the practical parts of our studies with me. Especially Georg Kitz who always encouraged me to be better than average inside and outside the borders of university.

Patrick Thonhauser
Graz, Austria, July 2012

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Twitter has grown tremendously in the last few years and is generating 300 million Tweets and 1.6 million search queries each day. As of now (2011), Twitter has over 300 million users. These are pretty impressive numbers for a microblogging/social-network platform and Twitter has already become a cultural phenomenon[1]. Every day people all over the world are communicating via Twitter, exchanging the latest news and discussing millions of diverse topics. They are spreading their knowledge and personal opinions all over the web. An indicator for making this behavior possible, is that nearly every web- or native social application provides the option to tweet about your activities. Working on a specific project, writing on a scientific paper, attending a workshop or just enjoying a tasty piece of pizza. The list of tweetable actions is almost infinite and everybody who is interested in a specific person or a specific topic, has the ability to consume the information by reading certain tweets or exploring the tweeted resources[2]. However the interesting questions for researchers are, how to make use of the information contained within millions of tweets and what can be extracted from those 140 character microblogs? How much useful information is in a single Tweet and how can we separate useful information from noise?

Anyway, the questions asked aren't new but keep researchers from all over the world busy. Many known and approved researchers try to find answers and some of them, like [Softic et al., 2010], [Mika and Laniado, 2010] or [Choudhury and Breslin, 2010], indeed were able to resolve parts of this challenging puzzle. For Semantic Web researchers, Twitter has become one of the most popular applications for the dissemination of information [Kraker et al., 2010] and it is therefore a legitimate candidate to serve as the main source for mining data that concerns users and providing information of scientific interest. Although Twitter is very popular, there are still a lot of tasks that require attention for Semantic Web researchers. Doing semantic analysis, classifications of Tweets, trend detection, mining of Tweets etc. are all very important, interesting and substantial parts to discovering the sense and meanings of Tweets and Twitter in general. Nonetheless there is a lack of workable and useful real world examples. The content of this master's

[1] | http://prsay.prsa.org/index.php/2012/03/23/friday-five-looking-back-at-twitters-seminal-moments/ (June 2012)

[2] Weblinks, fotos etc.

thesis isn't just about providing a valid explanation of how to extract information from Twitter, but rather valuable guidance and an example of how to use existing research and gained scientific findings to build a useful recommendation system which allows the discovery and utilization of interesting content and user information. The main questions this thesis addresses are listed below:

- Is Twitter useful for discovering new connections between researchers in similar or the same subject areas?
- Is it possible to recommend Twitter users based on their Tweets?
- Is it possible to separate useful content from noise to a satisfying level of success?
- Are there any appropriate classifiers and metrics to measure the significance of Twitter users and Tweets?

In general, the improvement and optimization of recommendation systems is still a challenging and topical subject. Nonetheless, whilst recommending people from diverse research branches is one of the most interesting aspects of this issue, it is not the only significant reason for exploring this topic in detail. Another highly interesting aspect is the opportunity to focus the stream of information a user is searching in. Undoubtedly, this represents a very powerful kind of online personalization in which Twitter accounts are able to play a massive role in the bundling of information that is obtained.

The reason for the major role of Twitter is for one, that Twitter has an already huge and exponentially growing number of users and secondly the selective and conscious (although unfortunately not always) usage of *hashtags* by Twitter users. Twitter uses hashtags to tag a Tweet with a topic. e.g. one main topic of my #masterthesis is #semanticweb. Consequently, my Tweet is tagged with the words *masterthesis* and *semanticweb* and can be found by other users by searching for these keywords. Besides hashtagging, the ability to retweet somebody's information gives the user the opportunity to highlight information that could be of further interest for other users that may work in the same research community. All the aforementioned indicators and features of this microblogging platform build a very promising basis for recommendations in general.

Obviously, there are a lot of application areas for Semantic Recommendation Systems. Not only for individuals, but also for companies or universities. The ability to access a personalized environment, that is tailored to particular needs and topic areas, is of benefit to anybody who is seeking interesting information or expertise. Furthermore, it's also very useful for detecting trends, especially in technology and science [Kraker et al., 2010].

Within this thesis, the basics of Semantic Web, its tools and standards are discussed. Linked Data, Recommendation Systems and the classification of microtext artefacts will also be discussed as part of those basics. The next step is introducing a concept and the definition of building such a *Semantic Recommender System for Research 2.0*, followed by the design and implementation of this system. Finally, the proof of concept is presented and discussed.

Chapter 2

Fundamentals

Before being able to develop and further to explore *Semantic Recommendation Systems* in detail, some pre-work has to be done. There are some very important and interesting topics that should be discussed and mentioned to avoid the risk of "reinventing the wheel". The following sections represent the basis of the forthcoming system and led this thesis and it's associated work in specific directions. At the very beginning it was necessary to work through a substantial amount of existing research to obtain a fundamental knowledge of the things that should be addressed before undertaking the development and design of *Semantic Recommendation System for Research 2.0*. Reading through this section serves as a good grounding for gaining important background knowledge in order to understand the related topics and practical parts of the thesis.

2.1 Semantic Web and its standards

In the early 90s the prolific emergence of pervasive and user-friendly digital technologies in our information society, made it nearly impossible to manage all the data on the Web. Also machine processing couldn't be established under the preveiling circumstances [Leger et al., 2006]. Obviously something had to change. Around this time, the first concept of a Semantic Web was invented and finally it became one of the major improvements of the World Wide Web. None other than Tim Berners-Lee, the pioneer of the World Wide Web came up with the idea of Semantic Web in 2001 in his *"The Semantic Web"* article in the *"Scientific American"* magazine [Berners-Lee et al., 2001]. He defined the word Semantic Web as follows:

> "The Semantic Web is an extension of the current web in which information is given a well-defined meaning, better enabling computers and people to work in cooperation."

In other words, Berners-Lee, who is also the director of the World Wide Web Consortium[1] (W3C), was talking about an additional layer of information, which describes the meaning

[1] `http://www.w3.org/` (January 2012)

of data. Combined with the Web 2.0 [O'Reilly, 2005] concept, Semantic Web defined a new era of the World Wide Web: *Web 3.0* [Markoff, 2006]. Semantic Web was and still is a new idea of mining human intelligence to make a computer reason in a human fashion. The possibility of doing so, opened a lot of promising doors, especially for industrial and scientific facilities.

The World Wide Web Consortium is leading this collaborative movement called Semantic Web and promotes common formats for describing and mining resources on the Web. W3C aims to convert the current unstructured documents to a so called *"Web of Data"*. The most important and fundamental Framework for the realization, is the *Resource Description Framework*[2], which will be discussed later. However the term *"Semantic Web"* is also referred to in W3C's vision of linked data, which is defined by W3C as follows:

> "The Semantic Web is a Web of Data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. The collection of Semantic Web technologies (RDF, OWL, SKOS, SPARQL, etc.) provide an environment where applications can query that data, draw inferences using vocabularies, etc."

In the following subsections these technologies and terms are discussed in depth. Starting with one of the base concepts of Semantic Web: Linked Data.

2.1.1 Linked Data

Linked Data is nothing other than creating typed links between data from different sources [Bizer et al., 2009]. While the lion's share of the web consists of HTML documents, Linked Data relies on documents written in RDF. This sounds quite familiar when someone thinks of the term Semantic Web. But there's an important difference between Linked Data and Semantic Web. More precisely, Linked Data is the data, that makes resources semantically readable for computers [Stankovic et al., 2010].

A simple way to explain the relationship between them is by using the example of ontologies. Let's say there exists an ontology called "Reptiles". In an object oriented way of thinking, you can say, "Reptiles" is our base class. Instances of the base class such as "Snake", "Gecko" or "Alligator" are semantic data of this ontology. Now when you link these instances, for example,via DBpedia[3], you will get Linked Data. By exploring and following those links, a computer is able to notice, that an alligator, snake or gecko, is a reptile.

The famous Tim Berners-Lee defined some very important and still valid rules for publishing data on the web. The aim of these rules is, to make this published data part of a single global data space [Berners-Lee, 2006]. These rules are:

[2] RDF

[3] DBpedia is a project that is extracting structured information from Wikipedia and making this information explorable for web applications. The DBpedia system offers the possibility, of linking this data with information from those web applications.

- Use URIs[4] as names for things.
- Use HTTP URIs so that people can look up these names.
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL[5]).
- Include links to other URIs. so that they can discover further information.

> "These rules have become known as the 'Linked Data principles', and provide a basic recipe for publishing and connecting data using the infrastructure of the Web while adhering to its architecture and standards."[Bizer et al., 2009]

By following these 4 simple rules, everybody has the ability and responsibility to make the data and resources that are provided semantically useful.

2.1.2 Resource Description Framework(RDF)

So what is RDF and what is it good for?

RDF is a standardized model for describing information related to objects (resources) in the World Wide Web. These resources are identified via URIs. The syntax of RDF is quite distinct from XML's tree-based infoset and therefore, quite easy and straightforward to learn and read. RDF is a collection of triples. Each triple is consisting of a subject, a predicate that denotes the relationship and an object. A predicate always points from the subject to the object and never in the opposite direction. Subjects and objects are nodes that can be URIs. Subjects and objects can both be blank nodes[6] but only an object can also be a literal. For a detailed description of the concept and design of RDF, see the W3C definition of this concept[7].

The following example is a description for the resource "`http://www.w3schools.com/rdf`" and serves to display how RDF is used to describe resources in general.

```
<?xml version="1.0"?>

<RDF>
  <Description about="http://www.w3schools.com/rdf">
    <author>Jan Egil Refsnes</author>
    <homepage>http://www.w3schools.com</homepage>
  </Description>
</RDF>
```

Listing 2.1: RDF Example Code

[4]URIs are unique identifiers, which consist of character strings and describe a physical resource on the web.

[5]RDF and SPARQL will be discussed in the following subsections.

[6]The blank nodes in an RDF graph are drawn from an infinite set. This set of blank nodes, the set of all RDF URI references and the set of all literals are pairwise disjoint.

[7]`http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/#dfn-literal` (January 2012)

Let's say this resource: `http://www.w3schools.com/rdf` is a subject, predicate is *"author"*(see line 5 in the code example 2.1) and the object is *"Jan Egil Refsnes"*. So this piece of RDF-Code says, that *Jan Egil Refsnes* is the author of this resource. By using this standardized description of resources, computers are now able to link the name *Jan Egil Refsnes* to this resource and can also figure out, in which relationship he interrelates to it.

After defining what RDF is and how it can be used to describe simple resources, there is also the need for a source that describes the properties and classes of RDF resources. Concluding, one needs a common vocabulary and the *RDF Schema*[8] offers such a vocabulary for formalizing ontologies. Because RDF only supports *type*-elements for typecasting, one needs some additional concepts, for example, to create a taxonomy of terms. Therefore RDF Schema supports the concept of *Classes and Properties*. A class represents an abstract object. By joining the concept of classes and types, RDF Schema is able to create instances. Classes have different properties, which represent the second additional concept in comparison to RDF, to describe an instance of a class. To be able to query RDF and RDF/S resources, you a need a query language similar to that of SQL for accessing and manipulating relational databases.

2.1.3 SPARQL Protocol and RDF Query Language

Another very important Semantic Web standard is SPARQL. SPARQL is recommended by W3C and is entirely designed to meet the use cases and requirements identified by the RDF Data Access Working Group[9] . This query language is also mentioned in Tim Berners-Lee's four "Linked Data Principles"[Berners-Lee, 2006] and plays a major role when talking about Semantic Web and Linked Data. What SQL is to databases, SPARQL is to Linked Data formalized in RDF. There are four different query forms to form different result sets for RDF graphs.

- SELECT: Returns specified or all variables bound in a query match pattern and those results are returned in a table format.
- CONSTRUCT: Returns an RDF graph, which was extracted from a SPARQL endpoint.
- ASK: Returns true or false according to whether the defined query pattern matches or not.
- DESCRIBE: Returns an RDF graph, that describes the resources found.

For a detailed description of the usage and definition of the four query forms, see the W3C documentation of SPARQL[10].

[8] RDF/S
[9] `http://www.w3.org/TR/rdf-dawg-uc/` (January 2012)
[10] `http://www.w3.org/TR/rdf-sparql-query/` (January 2012)

2.1.4 Web Ontology Language (OWL)

The first thing someone recognizes when he or she reads the acronym for Web Ontology Language is that OWL doesn't represent the first letters of the standard's name. But there is an easy and comprehensive explanation for that. It simply sounds better than WOL and refers to the animal that symbolically stands for wisdom. Although this is not the most interesting fact about OWL, it may cause confusion.. To also avoid any confusion around RDF Schema and OWL from the beginning, OWL in comparison to RDF/S adds more vocabulary for describing properties and classes. However both standards can and do coexist within the very same RDF document. At the end of this section, there will be an example combining RDF, RDF Schema and OWL to illustrate the interactions of these essential Semantic Web standards.

The idea behind OWL is, that some applications might not just need to represent information to humans, but rather need to process the content of the information. By providing additional vocabulary along with formal semantics, the language's machine interpretability of web content is much bigger than the machine interpretability of other well known languages. OWL was developed to exactly meet the needs of a Semantic Web. The main purpose is, to make it easier for machines to automatically process and integrate information, which is available on the Web. If machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF/S.[11] More precisely, it describes the meaning of the terminology used in Web documents, by using more detailed and specially designed concepts for fulfilling the task of linking data.

OWL is split into three sub languages. The following descriptions are taken from W3C's definition of OWL's sublanguages[12]:

- **OWL Lite:** This sub language is the light version of OWL. To be utilized by users that just need a classification hierarchy and simple constraints. The advantage of OWL Lite is the easy and quick migration path for thesauri and other common taxonomies and it's less formally complex than the two other OWL versions.

- **OWL DL:** OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL derives from its correspondence with description logics, a field of research that has studied the logics that formed the formal foundation of OWL.

- **OWL Full:** OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example,

[11] `http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.2` (January 2012)
[12] `http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.3` (January 2012)

in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full

Deciding which of these sublanguages is the right choice is by far more difficult than one might think and every OWL developer has the ordeal of choice. Nevertheless, every developer who plans to use OWL should think about his individual expectations and should read the W3C documentation very accurately before beginnung with the development[13].

Since the definition of OWL in early 2004, **OWL 2** has been released and officially documented and recommended by W3C in late 2009. OWL 2 is very similar to OWL 1 and fully backwards compatible with it's ancestor. The role of RDF/XML and other syntaxes have not changed. OWL 2 just adds new functionality to OWL 1 and therefore, makes the adoption of the new standard very easy and straight forward. Similar to OWL 1, there are also three sublanguages, which are now referred to as *Profiles*. The following listing of OWL 2's sublanguages are cited from OWL 2's official definition of W3C[14].

- **OWL 2 EL** enables polynomial time algorithms for all the standard reasoning tasks; it is particularly suitable for applications where very large ontologies are needed, and where expressive power can be traded for performance guarantees.
- **OWL 2 QL** enables conjunctive queries to be answered in LogSpace (more precisely, AC0) using standard relational database technology; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to access the data directly via relational queries (e.g. SQL).
- **OWL 2 RL** enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples.

Several new features like keys, property chains, richer data types and ranges, asymmetric, reflexive, and disjoint properties etc. make OWL more powerful and offer more expressivity than ever. To conclude the big topic of OWL, figure 2.1 shows the overall structure and main building blocks of OWL 2. The center of this figure is an OWL 2 ontology, which can be seen as a RDF graph. On top of the center, you see various

[13] `http://www.w3.org/TR/2004/REC-owl-features-20040210/` (January 2012)

[14] `http://www.w3.org/TR/owl2-overview/http://www.w3.org/TR/owl2-profiles/` (January 2012)

concrete syntaxes. These different syntaxes can be used to exchange and serialize ontologies. The two blue marked boxes at the bottom of this graph represent the two semantic specifications that define the meaning of OWL 2 ontologies[15].

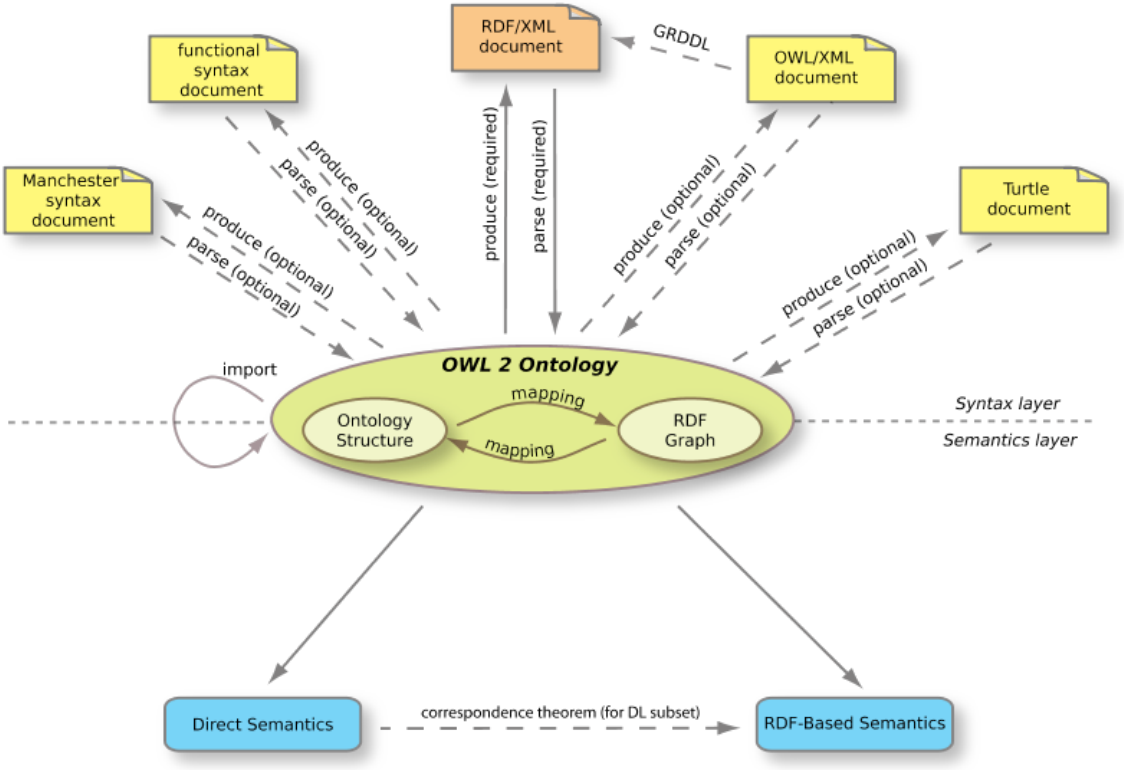

Figure 2.1: OWL2 structure (`http://www.w3.org/TR/owl2-overview/`)

2.1.5 RDF, RDF/S and OWL combined

As already mentioned, OWL, RDF and RDF/s can and do coexist in even a single document. The following listing 2.2 displays an example of the usage of RDF, RDF/S and OWL[16]:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:plants="http://www.linkeddatatools.com/plants#">
  <!-- OWL Header Omitted For Brevity -->

  <!-- OWL Class Definition - Plant Type -->
```

[15]Image is taken from W3Cs OWL 2 documentation `http://www.w3.org/TR/owl2-overview/` (January 2012)

[16]`http://www.linkeddatatools.com/introducing-rdfs-owl` (January 2012)

```
10    <owl:Class rdf:about="http://www.linkeddatatools.com/plants#
          planttype">

12      <rdfs:label>The plant type</rdfs:label>
        <rdfs:comment>The class of all plant types.</rdfs:comment>
14
    </owl:Class>
16
    <!-- OWL Subclass Definition - Flower -->
18  <owl:Class rdf:about="http://www.linkeddatatools.com/plants#flowers"
        >

20      <!-- Flowers is a subclassification of planttype -->
        <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/
            plants#planttype"/>
22      <rdfs:label>Flowering plants</rdfs:label>
        <rdfs:comment>Flowering plants, also known as angiosperms.</
            rdfs:comment>
24
    </owl:Class>
26
    <!-- OWL Subclass Definition - Shrub -->
28  <owl:Class rdf:about="http://www.linkeddatatools.com/plants#shrubs">

30      <!-- Shrubs is a subclassification of planttype -->
        <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/
            plants#planttype"/>
32      <rdfs:label>Shrubbery</rdfs:label>
        <rdfs:comment>Shrubs, a type of plant which branches from the base
            .</rdfs:comment>
34
    </owl:Class>
36
    <!-- Individual (Instance) Example RDF Statement -->
38  <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#
        magnolia">
      <!-- Magnolia is a type (instance) of the flowers classification
          -->
40    <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#
          flowers"/>
    </rdf:Description>
42  </rdf:RDF>
```

Listing 2.2: OWL Example Ontology

This example represents three plant classes. Flowering plants and shrubs are both subclasses of the planttype class. This planttype class is at the top of the plant hierarchy in this example. As you see, OWL ontologies are defined by RDF resources, as well as RDF/S properties and classes. OWL is used as a semantic extension for RDF and OWL ontologies are defined by extension of RDF/S meaning.

2.1.6 More Standards

This subsection presents some more Semantic Web Standards that are used very frequently and should also be mentioned when talking about Semantic Web Standards.

2.1.6.1 Simple Knowledge Organization System Reference (SKOS)

SKOS is a family of formal languages and is designed for expressing the basic structure and content of many types of controlled vocabulary. SKOS is built upon RDF and RDF/S and its main purpose is to provide a model for composing and publishing concepts on the World Wide Web and linking such concepts semantically. Just as in RDF, concepts are identified with URI's and are described with strings in many different languages. SKOS's fundamental elements become concepts by introducing the class concept. This class makes it possible to identify a given resource as a concept. For describing concepts, SKOS offers a variety of labels, where you can choose preferred, alternative and hidden labels. In comparison to OWL, SKOS is a stand alone vocabulary and is widely used beyond the librarians world. The reason for this is the aforementioned labeling feature that can be used with any kind of real world data. SKOS is also very popular for interchanges between different social networks [Bleier et al., 2011].

2.1.6.2 Friend of a Friend Project (FOAF)

The *Friend Of A Friend* project aims to create a Web of machine readable pages. Each page of this web describes an internet user itself as well as users, where each user is connected to other users by different kinds of relationships defined by Social Networks and similar connections. FOAF is not only used to describe how people are connected to each other, but rather to make visible, what they do and create on the World Wide Web. Every user on the Web has the ability to generate such a FOAF file and upload it to the Web. FOAF is also an RDF application and therefore can be combined with many other vocabularies. This vocabulary is widely used for enriching Social Network accounts by integrating new data, described by FOAF profiles. Services like IYOUIT [Boehm and Luther, 2009] try to provide personalized services by context aware user profiling and the adoption of common semantic representation formats. For a detailed introduction to this standard, take a look at the official Friend of a Friend website[17].

2.1.6.3 Semantically-Interlinked Online Communities (SIOC)

Semantically-Interlinked Online Communities are also a mentionable Semantic Web technology. The SIOC vocabulary is also based on RDF and recommended by W3C. This standard was designed to meet the requirements of a Semantic Web ontology representing rich data from Social Web in RDF. Usually it's used hand in hand with FOAF for describing Social Network accounts and information. SIOC has become a standard for expressing

[17] `http://www.foaf-project.org/docs` (January 2012)

user generated content and offers new ways and possibilities for creating Semantic Web applications, built on top of user generated data. Online blogs, message boards, wikis, etc. have replaced traditional methods of publication on the web and make it possible to build bridges between the huge amount of published information.

> "Developers can use this ontology to express information contained within community sites in a simple and extensible way."[18]

A detailed documentation of this powerful vocabulary can be found on their official SIOC homepage[19].

After now having had a brief look at Semantic Web and its standards, we can move on to the next basic topic for building a Semantic Recommender System.

2.2 Twitter and Microblogging

In the last few years, microblogging and especially Twitter[20] have gained strong importance and their usage and adoption rate have grown dramatically [Softic et al., 2010]. More and more people use microblogging for sharing their knowledge, experiences, opinions and feelings.

> "People are connecting with each other in a very smart way, sharing and discussing their topics, exchanging information,using it with their devices over mobile Internet access and evolved different techniques to enhance their online communication."[Ebner, 2010]

Twitter is the largest microblogging platform of 2012 so far and is used as a reference microblogging platform throughout the whole thesis. But what exactly is microblogging and what can we extract from that huge amount of user generated data?

> "Microblogging is a small-scale form of blogging, generally made up of short, succinct messages, used by both consumers and businesses to share news, post status updates and carry on conversations."[Templeton, 2008]

Extracting useful data from microblog platforms like Twitter, was and is a very popular and promising field of research and many researchers have found ways to make microblogs semantically useful. However, to be able to make Tweets and the information contained readable to machines, the computer has to understand the meaning of individual Tweets. Many researches like [Choudhury and Breslin, 2010], [Softic et al., 2010] or [Milikic et al., 2011] have come up with different approaches for detecting connections between Tweets in various categories like sports, science, events, etc. In this section I'm going to explore

[18] `http://rdfs.org/sioc/spec/` (January 2012)
[19] `http://sioc-project.org/` (January 2012)
[20] `http://www.twitter.com` (January 2012)

some very interesting previously undertaken work and research concerning this topic, starting with formalizing and defining tweets in general and explaining how they differ from usual information streams in the World Wide Web.

2.2.1 Social Awareness Streams

Any user of the World Wide Web who has ever used Twitter, Facebook[21] or a similar Social Network has certainly seen Social Awareness Streams without even noticing them. When users login to a Social Network website, they usually see a stream of information generated by other users they might follow or are friends with, in reverse chronological order. Knowing what Social Awareness Streams are, makes it pretty obvious that one of the major features of Twitter is such a Social Awareness Stream. However this isn't anything unusual for a Social Network. In comparison to other social streams of similar systems, the stream of Twitter seem to be kind of special. In networks like Facebook or Google+[22], the data structure of messages is defined by developers. In contrast, Twitter messages have a user defined data structure, because users have the ability to use hashtags, retweets[23] and replies or mentions[24] on their own terms. This has made Social Awareness Streams in Twitter much more complex and implies a dynamic data structure. Therefore, [Wagner and Strohmaier, 2010] introduced and analyzed a model for formalizing Tweets as tripartite models named *Tweetonomies*. Fig. 2.2 visualizes an example Tweet formalized as Tweetonomy.

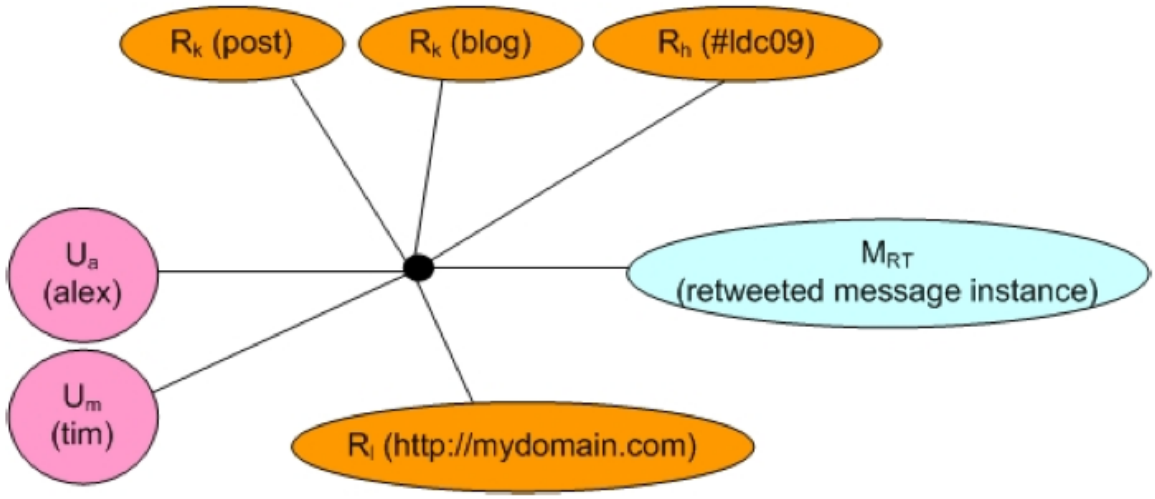

Figure 2.2: Example Tweetonomy for the Tweet: *"RT@tim new blog post: http://mydomain.com #ldc09"*

[21] `http://www.facebook.com` (January 2012)
[22] `https://plus.google.com/` (January 2012)
[23] RT
[24] @

U, R and M are finite sets containing users (U), resources (R) and messages (M). By using this kind of formalization, three mode networks can be generated and analyzed for visualizing relationships between users, resources and messages. This isn't just interesting for doing research on specific tweets, but rather for revealing how much different and dynamic information a tweet can contain.

After defining Tweets theoretically, the next step is using this knowledge to bring Tweets inti a semantically useful type of machine read- and interpretable presentation by using technology and standards already mentioned in section 2.1

2.2.2 Making Tweets semantic

To make Tweets semantically useful, twitter data has to be brought into a machine readable form of presentation. RDF is is a good example for doing such a thing. The next step would be the description of the relationships between the different Tweets and tweeters. Semantic Web standards like SIOC (see section 2.1.6.3), FOAF (see section 2.1.6.2) or OWL (see section 2.1.4) are appropriate tools. One example of such a system is Grabeeter [25]. Grabeeter was developed by TU Graz and stores as well as indexes Tweets of a Twitter account for further processing and usage [Softic et al., 2010].

> "The Grabeeter web application enables users to archive their tweets in the Grabeeter database and to perform a search on the stored tweets through a web interface."[Muehlberger et al., 2010]

With systems like DBPedia[26], the semantic version of Wikipedia, sources can be mapped to each other as described in section 2.1.1. However, after extracting the data, Tweets can be used for analysis by using SPARQL endpoints and lookup services.

Nevertheless, this represents only one possible way of making microblogs semantically useable. The real challenge hides in detail and that's the linking of data and the definition of Tweets and the information that is contained in them, however that is related to the identification of users that are of a certain scientific interest.

2.2.3 Semantic Relatedness and Metrics

The big task here is to define rules, which make it possible to extract as much useful information as possible and identify, in which context information is linked to each other [Milikic et al., 2011].

The meaning of the term is always defined by the social context. Social systems are dynamic when they depend on their context as are the meaning of terms. This is a big problem concerning the linking of individual terms to each other because they are constantly changing and they also have different meanings within different circles of

[25] `http://grabeeter.tugraz.at/` (January 2012)
[26] `http://dbpedia.org/About` (January 2012)

people. It's quite easy to detect trends in the world of Twitter by searching for hashtags or simply using the integrated trend feature. One question that arises at this point is how are these trends related to each other? Let's say the terms *iPad3*, *launchday* and *Austria* are the current trending topics on Twitter. The dynamic changing of relationships between terms makes it very hard to define how these terms relate in different contexts.

[Milikic et al., 2011] also offer a very interesting solution for this problem. They developed a function named *Normalized Micropost Distance*, which makes it possible, to measure the relatedness of terms over a defined period of time. Therefore, identifying the sense of context between two terms is not enough for filtering useful information. The reason is that information about the importance of single terms, the Twitter user and hashtags is still lacking.

2.2.4 Trend detection

Trend detection is a very widespread field of research especially concerning microblogging. Researchers all over the world are developing operations to detect what people are actually talking about and what bothers them. The aim is not only to detect general trends, but also to gain customized information, which is of interest to a certain target audience (e.g. soccer fans, computer geeks, music nerds, etc.). As a fact, the number of Tweets increases with every second. This however, reflects such an unbelievably huge amount of information that it is really impossible to read all the Tweets. Additionally, a lot of Tweets are just noise and thus have to be eliminated from the evaluation. [Kraker et al., 2010] developed such a trend detection system by using either (a) a specified taxonomy of keywords, (b) specified list of users, or (c) a combination of both. After the crawled Tweets are scanned for informative nouns or hashtags, using POS-tagging[27], which will be discussed in depth in section 2.4, the obtained information can be used for the evaluation of Tweets.

Another technique to detect hashtags, which exhibit the desirable properties of strong identifiers or shortly summarized, which are probably trends, is introduced by [Mika and Laniado, 2010]. The first step they took to "rate" hashtags was to define metrics and scales for different attributes concerning Tweets and hashtags. They present four very interesting attributes in their paper.

- **Frequency:** Hashtags are used by a circle of people with some frequency. This frequency is measured in number of users using this hashtag and the number of messages sent by those users containing this hashtag. By exploring the correlation between these two numbers, they calculate the frequency of a hashtag.
- **Specificity:** The extent to which the usage of a hashtag deviates from the usage of the word without a hash.
- **Consistency in usage:** Hashtags are used consistently by different users and in different messages to indicate a single topic or concept.

[27] Part of Speech tagging

- **Stability over time:** The hashtag should become a part of the persistent vocabulary of Twitter users, i.e. it should have sustained levels of usage and a stable meaning over a period of time.

They defined a vector space model for evaluating the metrics of a hashtag and studied the evolution over time. *Vector Space Models* are discussed in section 2.3.2.1 of this chapter.

2.2.5 Categorizing users

Another significant discussion concerns the categorization of individual Twitter users and the possibility of detecting single Tweets that fit the nature of a certain user. [Horn et al., 2011] for example, used a supervised classification to deal with this problem and they also implemented an application following the path of supervised learning(see section 2.3.2.3). However the difficulty regarding the huge amount of information still remains. [Choudhury and Breslin, 2011] also addressed this kind of problem by developing a prototype for discovering sports Tweets and finding appropriate classifiers, which allow for identification of Tweets and tweeting users that belong to and/or tweet about a specific sports event. And indeed they found classifiers which were almost completely fulfilling their needs. There are many different approaches to solving these problems and this hints at a growing economical importance and potential. Twitter is also referred to as the electronic word-of-mouth. Indisputably, getting information about customized needs, e.g. about user preferences regarding car brands and concerning expectations, brings valuable competitive advantage among others [Jansen et al., 2009].

Nonetheless, the question of high scientific interest that remains is indeed a very delicate one. Is it necessary to develop classifiers for each event or category you want to filter, or is it indeed possible to define a set of rules or metrics or classifiers that are able to automatically readjust according to changing circumstances? In other words, the question deals with the difficulty of not losing the context in this tremendous, dynamic and ever progressing pool of information.

2.3 Recommender Systems

The idea of Recommender Systems emerged in the 90s, when more and more people started using the internet and consequently the associated amount of information available for everybody who had access to information provided by other internet users, increased dramatically. In 1992, the PARC Tapestry System [Terry, 1992] was introduced. Eventually, it was the first system for collaborative filtering. This system was supposed to help internet users to find more interesting and useful data related to their personal needs. In the first place, it demonstrated how explicit data and implicit behavioral data, can be stored into a queryable database. By combining these two kinds of data[28], personal

[28]Explicit data is data, shared by users, like writing an article or rate something. Implicit data is data that is derived from explicit data, like when you say: *"Marco was tired when he came home from*

filters were born. Nonetheless, the big disadvantage of the system is, that the generation of personal filters aren't working automatically and therefore, every user had to define his own filter. In 1994, the GroupLens System [Resnick et al., 1994] made the usage of recommender systems easy for everyone, by providing an automated mechanism for the generation of collaborative filters in a network for news articles. Over the years there have been four states in advancing Recommender System, which will be discussed in the following sections.

The way to commercialization wasn't a broad one. In the mid 90s, the internet business expanded rapidly and so did the need for Recommender Systems in real world systems. Nonetheless integrating them into existing or even new real world systems also brought some new challenges, including changing items, new and growing userbase, trust, transparency etc. Between the year 2000 and 2006, research on Recommender Systems literally exploded, eventually caused by the burst of the so called internet bubble in 2000[29] and the fear of a second burst in 2006. A lot of big companies, like Amazon [Linden et al., 2003], had already integrated and were using Recommender Systems in their online system and thus, gained huge advantages over websites and systems which weren't using those systems and the huge related benefits.

Netflix gave this field of research a big boost by introducing the *Netflix Prize*[30] in 2006, which ended in 2009 by declaring "BellKor's Pragmatic Chaos"[Bell et al., 2008] as the winner. However, the boost of popularity and the interest regarding this topic did not peter out after the competition ended. Now, in the very beginning of the year 2012, this hot topic is indeed on its way to the best accuracy levels obtained so far. This section serves as an overview of state of the art Recommender Systems. As already mentioned, there are four basic concepts for such systems, starting with the father of the Recommender System concepts: *Collaborative Filtering*. Reference should be made to [Jannach et al., 2011] which serves as the relevant source concerning the following sections of the thesis. However, this work provides a very detailed overview of the most important and preferably applied Recommendation System techniques. However, Collaborative Filtering and Content based Recommendations are of huge relevance to the practical part of this master´s thesis and are thus discussed in more detail.

2.3.1 Collaborative Filtering

Collaborative Filtering[31] deals with similarities between individual users. If one considers that user A bought bananas and user B also did, it suggests that they may have something in common. If they both buy the same or mostly akin products, the possibility that they will act in similar ways in the near future is conceivably high. Consequently, it is comprehensible that if A buys a coffee mug, it's very likely, that user B also wants to buy one. This reveals that the users collaborate with each other in an implicit way.

university", implicit data is for example: Marco is a student.

[29] `http://articles.latimes.com/2006/jul/16/business/fi-overheat16` (March 2012)

[30] `http://www.netflixprize.com/` (March 2012)

[31] Collaborative Filtering = CF

Another very interesting fact about CF is, that the system doesn't need to know anything about the items that are recommended, meaning the data doesn't have to be semantic. All recommendations are based on individual user behaviors. Nevertheless, there is one big disadvantage known as the so-called *cold-start problem*. For example, if you imagine books that haven´t been bought on a certain platform before, there are logically no recommendations available. In particular, there are three major problems in CF that one has to deal with, including the *cold-start problem*. The two remaining are *scalability* and *sparsity*. Millions of users and items occur in systems like Last.fm[32] or Facebook[33]. The computational complexity for calculating recommendations based on all those items and users in such a big environment is in fact huge. Sparsity however, originates from the fact that platforms like Amazon are selling a large variety of items, but just a few users really rate those items. Eventually, as a result, some of the most popular and most sold items are never rated by the customers. However, Collaborative Filtering/Recommendation, which represents a long-standing leading theme in the community, has been investigated for nearly 20 years now. Consequently, there is a huge wealth of knowledge regarding the advantages and disadvantages. Hence, there are a lot of solutions that are aimed at an efficient decrease of the impact caused by the three problem factors that have been mentioned.

A collaborative recommender approach always produces the same types of result:.

- A comparable numeric quantity of information based on the user behavior, meaning, i.e. if a user likes or dislikes an item.
- A list of n items that are recommended.

Most of todays successful Recommender Systems rely on CF and its techniques because it simply is the best-researched for predicting recommendations. There are two basic concepts for Collaborative Recommendations. User-based and item-based nearest neighbor recommendations.

2.3.1.1 User-based nearest neighbor recommendation

As indicated by the name, user-based nearest neighbor recommendation predictions are generated by analyzing users. Generally, this technique works as as follows:

Let R_{m*n} be a matrix where m are users of the system and n are items to be sold. The value $R_{i,j}$ in R_{m*n} represents ratings from a user i of an item j. Now the task is to find users that have similar likes or dislikes. These users are referred to as neighbors, where the nearest neighbor to a single user A is the one that has the most accordances with likes and dislikes of user A [Chen et al., 2010]. Next, it's time to predict which items a user will like in the future. This prediction is based on analysis of nearest neighbors. Similarities between users can be measured and calculated using several rating ratios. Some of the

[32] `www.last.fm` (March 2012)
[33] `www.facebook.com` (March 2012)

most frequently used and for this master's thesis important ratios are discussed later on. Companies or learning platforms with very specific environments need well adapted metrics and techniques for filtering recommender relevant information in order to weight different ratios so that a maximum amount of accuracy is obtained. The *APOSODLE* project for example aims to connect interesting people providing interesting information. To be able to acquire such a behavior, environment specific metrics and weights have to be found or developed [Beham et al., 2010].

2.3.1.2 Item-based nearest neighbor recommendation

Although user-based nearest neighbor recommendation is applied successfully in many different use cases, the problem of scalability in huge e-commerce systems still remains.It's at this point that item-based nearest neighbor recommendation comes into play. Real time predictions of recommendations in huge systems like Amazon are practically impossible. Therefore, preprocessing and precomputation has to be done to allow for the supply of real time predictions. *Item similarity matrices* are calculated in advance to lower the amount of possible items. At runtime, predictions are calculated by only taking into account the number of nearest neighbors, equal to the amount of ratings a user has produced [Linden et al., 2003]. In advance, there are many other useful additional ways to lower the complexity of prediction matrices, or to filter the most important factors for recommending items. Particularly in an environment where ratings are submitted by users, the choice concerning the scale of rates is an essential factor for developing a successful and satisfying Recommender System. Unfortunately, there's no panacea for defining an appropriate scale in advance.

Something that all Recommender Systems have in common is that testing and test data is a key factor for developing such a system. This however leads to the first big problem regarding the development of a Recommender System, as already mentioned in section 2.3.1, the lack of test data provided for several fields of interest, especially learning platforms [Drachsler et al., 2010].

While user-based nearest neighbor recommenders are referred to as *memory based*, because the database is held in memory for instant prediction processing, item-based nearest neighbor recommenders are referred to as *model based*. This can be accounted for by offline precomputation of pre defined models, which are used to calculate predictions in real time.

There are several model based approaches such as matrix factorization, which is used by the winners of the *Netflix Prize* [Bell et al., 2008], association rule mining [Romero and Ventura, 2010] or probabilistic recommender approaches [Jannach et al., 2011]. With the current success of using matrix factorization for predicting recommendations, it's nearly impossible to not recognize them as a potential way to go. However, for further information concerning model based approaches the above mentioned references might be of interest.

2.3.1.3 Matrix Factorization

As already mentioned, in the world of collaborative filtering, *matrix factorization* has stirred things up in the last couple of years. This kind of model based recommendation aims to derive latent factors from rating patterns. Those latent factors can be seen as *semantics*. They are calculated by approximating a matrix X to the product of two smaller matrices W and H. Thereby, X is the partially observed rating matrix. W contains vectors of latent factors which describes a user u_k and H describes the items i_k with feature vectors [Thai-Nghe et al., 2010]. w_{uk} and h_{ik} are elements of W and H. The result is the rating given by a user u to an item i.

$$\sum_{k=1}^{K} w_{uk} h_{ik} = (WH^T)_u, i \tag{2.1}$$

Formula 2.1 serves to calculate the rating, a user u has given for an item i. Researchers have shown, that the success of matrix factorization is strongly bound to the adequate elimination of data noise. Herein the key factor is, to train the system. Actually, there are several kinds of stochastic methods to sharpen the sense of such noise filters [Jannach et al., 2011]. Nonetheless, in a usual memory based or item based Recommender System, so-called similarity measures are commonly used for detecting similarities between users or items. The following section will discuss the three most commonly used types.

2.3.1.4 Similarity Measures

Choosing and defining similarity measures, plays a key role in developing a Recommender System. It's quite usual, that multiple different similarity measures are calculated to reach a higher level of accuracy in predicting recommendations. The most important and most common similarity measures are presented in the following.

Cosine Similarity Measure: In item-based recommender approaches, *cosine similarity* is established as the most accurate metric for describing the similarity between two items. In fact, it's also commonly used in the field of information retrieval and text mining, this however makes it even more interesting and useful for a Semantic Recommender System based on Tweets. The similarity between ratings of two item vectors, $\vec{a}$ and $\vec{b}$ is defined as follows[34] [35] :

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|} \tag{2.2}$$

Nonetheless cosine similarity has one big disadvantage, meaning that the average rating behavior is neglected.

[34] The symbol | stands for euclidean length of a vector
[35] The symbol · stands for product of vectors

Adjusted Cosine Similarity Measure: This advanced kind of cosine similarity takes average user ratings into account by subtracting it from the ratings. In the following formula, U is a set of users, who rated item a and b:

$$sim(\vec{a}, \vec{b}) = \frac{\sum_{u \in U}(r_{u,a} - \bar{r_u})(r_{u,b} - \bar{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \bar{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \bar{r_u})^2}} \tag{2.3}$$

The cosine similarity is predicted for every user or item entry in our R_{m*n} matrix and the rating entries are now replaced by adjusted or unadjusted cosine similarities. Predictions can be done in both cases by calculating a weighted sum of a user's rating for items that are similar to a specific item.

Pearson Correlation Coefficient What cosine similarity is for item-based nearest neighbor recommenders, the *Pearson correlation coefficient* is for user-based nearest neighbor recommenders. It represents a commonly used and outperforming measure for calculating similarity, in this case, for users instead of items. In the following formula, U represents a set of users and I a set of items. Our R_{m*n} matrix remains in the previously defined state of representing the ratings from users for items. User a's and user b's similarity is calculated according to the formula below.

$$sim(\vec{a}, \vec{b}) = \frac{\sum_{i \in I}(r_{a,i} - \bar{r_a})(r_{b,i} - \bar{r_b})}{\sqrt{\sum_{i \in I}(r_{a,i} - \bar{r_a})^2}\sqrt{\sum_{i \in I}(r_{b,i} - \bar{r_b})^2}} \tag{2.4}$$

By applying this formula for all users compared to each other, it's pretty straight forward to analyze the nearest neighbors. The next task is to make a prediction for a particular item in the system. This, however, includes the decision, regarding the nearest neighbor ratings that should be taken into account and how these ratings should be valued. In almost all use cases, this depends on the type of environment, that the system has to be adapted to. Consequently, it is possible to make predictions for an individual user, revealing/ suggesting possible new items of interest which the user might buy in the future.

In fact, collaborative filtering, which was the subject of detailed investigations throughout the last 20 years, represents a huge scientific field. This makes the aforementioned approach of developing Recommender Systems a very promising one, especially concerning semantic approaches. Experts in this field, like [Mödritscher, 2010] or [Solskinnsbakk and Gular, 2011], often take the advantages of using this technology to recommend information or also users in diverse environments. Classic collaborative filtering methods can be extended by tags for discovering similarities between users and items. But these tags aren't seen as semantics, but rather are seen as additional users in our user-item matrix for predicting similarities. Ratings in this matrix are set to 1, if a tag, belongs to this item.

2.3.2 Content-based Recommendation

In contrast to collaborative filtering, content-based recommendation systems are aware of the items they are recommending. Content-based recommenders know characteristics of items and users. For example, descriptions of video games are used to identify users, who are known to have similar affectations and who the system thus identifies as possible customers for a new video game. When a single user bought the videogame Halo 1 and Halo 3 and has bought several other shooter games, it's very likely that the user is interested in the newly added Halo Reach, although no one else has bought this game so far. The big challenge concerning the development of such systems is the identification of useful information about items, which might be of certain interest to a specific user. In fact *content-based* doesn't always mean that items and their characteristics are stored in relational databases and then are checked for similarities.

This recommendation technique has been developed for the purpose of Semantic Recommender Systems by analyzing news feeds and other documents, and checking for their semantic relatedness. So the main task, is to find relevant keywords within a document. Things like *social tagging*, which for example is used in Twitter (see section 2.2), helps in finding relevant keywords. Unfortunately, you can´t always rely on social tags and therefore, it requires another approach, which at least describes and measures useful information. One approach is *vector space models*.

2.3.2.1 Vector Space Models (VSMs)

Vector space models are often used for analyzing folksonomy systems like Twitter.

> "VSMs are commonly used in information retrieval as a representation of documents, where each dimension corresponds to term in the collection and each value measures the weight of that term for the document." [Mika and Laniado, 2010]

Content of a document can be encoded in keyword lists by using *TF-IDF*[36]. TF is a measure of the appearence of a certain term in a document. IDF is responsible for reducing the weight of keywords, that appear very often in all documents. The following definition the term frequency $TF(i,j)$ of a keyword i in a document j is just an example taken from [Jannach et al., 2011] to point out, how it can be done. $freq(i,j)$ is the absolute frequency of i in j. $OtherKeywords(i,j)$ denote the set of other keywords, appearing in j. First we need to know $maxOthers(i,j)$, which is defined by $\max(freq(z,i))$, where $z \in OtherKeywords(i,j)$.

$$TF(i,j) = \frac{freq(i,j)}{maxOthers(i,j)} \tag{2.5}$$

The idea behind IDF is, that keywords, which appear very often in nearly all documents, aren't as important as keywords occuring very infrequently in just a few documents. N

[36] term frequency-inverse document frequency

is the number of documents and $n(i)$ are the documents, in which the keyword i appears. Now we want to calculate $IDF(i)$ to be able to weight the keyword according to the idea of RDF.

$$IDF(i) = \lg \frac{N}{n(i)} \tag{2.6}$$

$TDF - IDF(i, j)$ where i is the keyword again and j is the document in which the keyword appears is defined as follows:

$$TDF - IDF(i, j) = TF(i, j) * IDF(i) \tag{2.7}$$

Our final vector space model consists of TDF-IDF weighted vectors for keyword i in document j. Nonetheless, the weighting of keywords reveals some major problems. Imagine a music store for guitars. In the description of the store it reads: *"No left-hander guitars are sold in our store"*. Now, because the word *"left-handers"* occurs just once in the whole description, the system weights it by far heavier than the word would be weighted in a description of a store which sells only lefthander guitars. The outcome would be, that someone who searches for lefthander guitar stores, will likely be guided to the store, which doesn't sell lefthander guitars. This non context awareness is a big problem, especially for recommendations regarding scientific papers.

2.3.2.2 k-nearest Neighbor (k-NN)

This approach also deals with the analysis of user profiles and is very similar to collaborative approaches. For the evaluation of an item, the likes/dislikes of a user have to be tracked and remembered. Next, cosine similarity (previously discussed in section 2.3.1.4) is used to measure the similarity of two documents. For a newly added item in the systems, we need to find the k most similar rated items. By analyzing those k-nearest neighbors in comparison to the user's behaviors, the system is able to decide whether the user is probably interested in the newly added item or not. K-nearest neighbor[37] approaches are also used for tag recommendations in folksonomy systems [Gemmel et al., 2009] to develop, for example, automatic approaches for annotating documents with concepts extracted from social data[Solskinnsbakk and Gular, 2011].

Summarized, k-NN systems have the big advantage of a simple implementation and it's easy to adapt the system to recent changes. Another big advantage is that the cold start problem is all but eliminated. But researches have shown, that pure kNN approaches often don't reach the accuracy of more ambitious techniques.

2.3.2.3 Classification

Content-based recommendation tasks can be formulated as classification problems. Doing that, allows the application of many different machine learning techniques. [Horn et al., 2011] for example, developed a system for detecting user types and tweet quality based

[37] k-NN

on supervised machine learning classification. Supervised means that the algorithm relies on training data where input and output is known. Finding such representative training sets is by the way one the most challenging tasks in supervised learning. Especially in text classification, it's a big problem to label a document as relevant or irrelevant. That's because new documents have to be assigned to predefined classes. A typical use case for something such as this is an anti spam system, which has to analyze a document if it fits into the spam scheme, know and classified by the system.

The next thing that may causes problems in classifying documents is, that there aren't only true or false options, but rather there can be by far more classification possibilities. Supervised based classification and machine learning techniques tend to overspecify classes and recommendations over time. A lack of recommendations containing fresh information would be the consequence, as recommended documents become too similar. So it's very important for developers of a recommender system based on supervised learning, to provide a mechanism for also recommending, let's say, unexpected documents. In advance of the Recommender Systems section 2.3, classification of microtext artefacts is discussed in section 2.4.

The following classification approach is based on Bayes. Bayes based systems occured very frequently during the research prior to this master's thesis and they are a classic classification approach for recommender systems.

2.3.2.4 Bayes Classifiers

In particular, let's start with *Multinomial Naive Bayes*. These approaches are commonly used for text classification [Eibe and Boukaert, 2004] and are based on the *Bayes Theorem* which is defined as follows:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.8}$$

$P(A)$ and $P(B)$ are *a priori* probabilities of A and B. A priori probability can be explaind with an easy to understand real world example. Imagine a bag that contains green and yellow stones. What is the probability of picking a green one? Therefore it's necessary to know what happened before we wanted to know what the probability of getting a green one is. $P(B|A)$ is the probability of B on condition, that A has happened. So $P(A|B)$ describes the probability relationship between A and B. In Naive Bayes for Text Classifiers, each document is viewed as a collection of words by neglecting the arrangement of words in this document. In the following formula, we use the Bayes Theorem for describing the probability relationship that class value c fits a test document d. Word w occurs n_{wd} times in document d. $P(d)$ and $P(c)$ are a priori probabilities for a document d or a class c.

$$P(c|d) = \frac{P(c) \prod_{w \in d} \cdot P(w|c)^{n_{w,d}}}{P(d)} \tag{2.9}$$

A detailed definition of all variables and probability relations, can be found in [Eibe and Boukaert, 2004]. [Seth et al., 2008] used Naives Bayes for developing a subjective credibility model for participatory media, like Twitter. This is in fact not POS (see section 2.4) but also a very interesting approach for making sense of Twitter. By defining several rules for identifying a particular message as useful and defining the credibility metrics, they created a bayesian user model, that computes credibility of a document in relation to a specific user. This system was developed to improve existing recommender systems or to filter previously executed recommendations. In Seth et al.'s paper, they talk about applying this technique to Collaborative Filtering techniques, which is a good example for a *hybrid recoomender* (see section 2.3.4).

Nonetheless, before talking about hybrids, it's important to talk about the last pure kind of recommendation techniques first: *Knowledge-based Recommendations*.

2.3.3 Knowledge-based Recommendation

Knowledge-based recommendations only rely on user ratings and demographic information. The two previously discussed techniques have their strengths and advantages but they don't fit all the requirements a recommender system could have. One typical use case for when a knowledge-based approach should be used is when you plan to do something that happens very infrequently and the decision has to be made very carefully. For example buying a new guitar or TV. In cases such as this, it's important to take knowledge about a quite long time span into account. People have very different needs and requirements for choosing their guitar and so they want to have the option to define their own rules for choosing the products. Such a task isn't typical for collaborative filtering or content-based approaches. Sometimes there aren't even ratings needed.

Knowledge-based recommender systems recommend items, for one specific user who defined the rules and metrics that a recommended item has to meet to be classed as interesting. These types of recommender systems are divided into two sub domains. *Constraint-based recommendations* and *case-based recommendations*. Constraint-based Recommenders rely on user predefined recommendation rules and recommend items from a given set that fulfill these requirements. Case-based recommendations focus on retrieval of similar items. This is done by comparing different types of similarity measures.

2.3.4 Hybrid Recommendations

Just like the name says, this section is about hybridization. All three of the presented recommender approaches have their strengths and weaknesses. Some of the weaknesses can be absorbed by combining different techniques of developing a recommender system like that mentioned in section 2.3.2.3. The main task here is to identify the strengths and weaknesses when you implement a recommender system for a particular need and of course, find an alternative that is able to overcome the weaknesses of your system.

[Jannach et al., 2011] define some of most used and common patterns for combining different algorithms and models and present a table for categorizing the input data requirements of recommendation algorithms. This table aims to make it easier to choose a particular algorithm for specific needs. But meeting all the prerequisites is just one aspect of system development. When it's possible to apply different design patterns, the overall design of a system becomes a very precarious task. Nearly all recommender systems in real world examples are hybrids, like the previously mentioned example of the Netflix Prize winners [Bell et al., 2008] or [Seth et al., 2008]. A very interesting and well written summary of how recommender systems can be combined and the different approaches that a are widely used is presented by [Adomavicius and Tuzhilin, 2005]. They also present a table in their publication, for choosing the right algorithms and recommender approaches for specific needs.

2.4 Classification of Microtext Artefacts and Part of Speech Tagging.

The classification of microtext artefacts is the basis of being able to build a Twitter-based Semantic Recommender System. This section deals with the question, what exactly has to be classified and can you make a machine context aware? How is a machine able to understand that a sentence like: *"This feature is killer"*, has nothing to do with a human who kills people? One major and commonly used step in developing context awareness in text classification is the usage of *part-of-speech*[38] tagging techniques, which is part of the broad topic of *Natural Language Processing*.

POS tagging aims to tag every single word in a sentence with predefined tags, which describe the meaning of a word in context. This enables a machine to detect semantic relationship between words in different contexts. POS tagging is just a part of the big topic of *Natural Language Processing*[39]. As we already heard in section 2.3 techniques like *cosine similarity* or *TF-IDF* don't have any semantic knowledge about the things they recommend. Therefore, it's essential to process written words in their very own context. So this technique makes it possible, to compare the text or information by its meaning and measure and recommend in advance.

2.4.1 NLP Pipeline

POS-tagging is just a part of a typical NLP pipeline. [Russel, 2011] describes a typical NLP pipeline as follows:

- **End of Sentence Detection:** The first part of the pipeline is responsible for breaking a text into meaningful sentences.

[38] POS
[39] NLP

- **Tokenization:** Tokenization means, making each word in a sentence a token.
- **POS tagging:** Assigning part of speech information to each token.
- **Chunking:** This step aims to analyze tagged tokens and detect logical concepts within them.
- **Extraction:** This step analyzes chunks and tags those chunks as named entities. Entities can be just about anything, such as animals, gadgets, people etc.

This is an example of how such a pipeline can be realized. In real world examples, the elements of the presented pipeline differ slightly. The information that is obtained by applying these steps can be used for measuring similarities (see section 2.3 Recommender Systems). POS is either an unsupervised or supervised machine learning approach. Unsupervised learning is defined by [Ghahramani, 2004] as follows:

> "Finally, in unsupervised learning the machine simply receives inputs x1, x2, . . ., but obtains neither super- vised target outputs, nor rewards from its environment."

The difference lies in the ability to tag a completely unknown word. [Gimple et al., 2011] for example, developed a POS-tagger especially for the needs of Twitter messages.

2.4.2 Hidden Markov Models

As already mentioned, the meanings of microtext artefacts in systems like Twitter, change dynamically over time and are used differently in various contexts. A classic and commonly used technique for defining the changing circumstances and dealing with probabilities that occur from unpredictable happenings in the system are HMMs. HMMs describe the probabilities for implicit(hidden) information. The following simple real world example demonstrates how HMMs work and are built:

First, imagine a coin. A coin has two sides, one displaying a symbol or person and the other one defines the value of the coin by displaying a number. The coin is flipped by another person and this person tells you the results after he/she flipped the coin. In addition to that, you can't see how the coin is flipped. So all you know about this experiment, is the result. Let's say H is head and N is number. A possible result is:

$$O = H, H, N, H, H, H, ..., N \tag{2.10}$$

So there are two possible states in this model, where each state has a probability of 0.5. We don't have any hidden elements in this model. But what if neither state is uniquely associated with either heads or numbers? The probability for each state would still remain 0.5, but the outcome sequences are independent from the state transitions. In other words, this model would be *hidden* [Juang and Rabiner, 2003]. This very simple sounding approach, can be applied to tasks like POS tagging, as described in [Goldwater

and Griffiths, 2005]. In the case of a POS tagger, HMMs help to identify a word, in relation to its sentence. Let's say we have a sentence like this: *"This is @PathonHauser's first tweet for AuroraApps"*.

In a real world POS taggers, every possible POS tag represents exactly one state $y \in s_1, ..., s_n$ in a HMM. Because all word categories can emerge after another, our graph would be fully connected. All edges have transition probabilities. Each state also produces a word of the sequence and a sentence can be generated by a walk through the graph. Figure 2.3 displays a possible HMM for classifying words in four different categories. Transition probabilities are fictional.

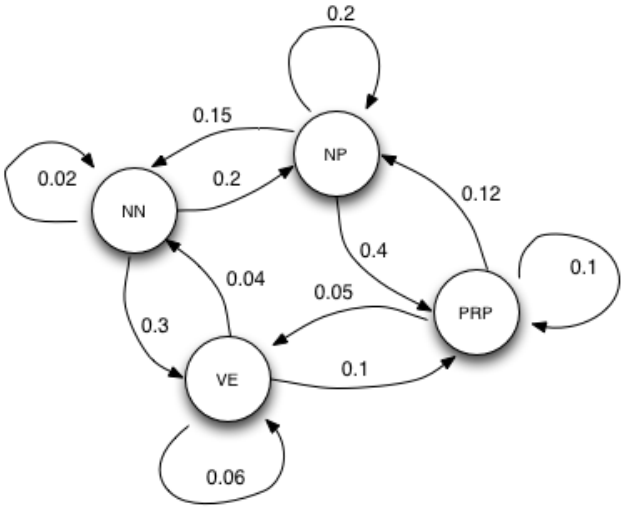

Figure 2.3: Example HMM for POS tagging with four different kinds of POS tags (NP, NN, PRP, VE) and its transition probabilities from one state to another.

Output symbols are defined as $x \in (o_1, ..., o_m)$, which are also referred to as observations. The probability for the starting sequence is defined as $P(Y_1 = y_1)$ and the probability for a state transitions is $P(Y = y_i|Y_{i-1} = y_{i-1})$. $P(X_i = x_i|Y_i = y_i)$ is representing output/emission probability. A word is equated with an observation. Every output and state sequence, has its probability:

$$P(\bar{x}, \bar{y}) = P(x_1, ..., x_l, y_1, ..., y_l) \tag{2.11}$$

After that, it's time to estimate the probability for transitions and emissions. Transitions can be estimated by the number of time state a followed state b in divided by the number of occurrences of state b. Emissions are estimated by the number of times output a is observed in state b divided by the number of times state b occurs. These estimations have to be smoothed in advance to obtain proper results. So far we we assume, that our system has input and output. But in an unsupervised approach, the system doesn't have an output.

What can be done, is to estimate expected frequencies and use them to calculate the

parameters, because the system doesn't know these hidden parameters. In Advance to applying this technique we define *forward-backward* probabilities.

The forward probability is defined by wanting to know, at a time t, what the probability that the system is in state s and the observation thus far has been $o_1, .., o_t$ is. $\lambda = parameters(A, B)$ is.

$$\alpha_t(s) = P(s_t = s, o_1, ..., o_t|\lambda) \tag{2.12}$$

Backward probability deals with the question, what is the probability that while the system is in state s at time t that the observation that follows will be $o_{t+1}, ..., o_T$. Mathematically formulated you get the following:

$$\beta_t(s) = P(o_{t+1}, ..., o_T|s_t = s, \lambda) \tag{2.13}$$

The following list represents the idea of the Baum-Welch algorithm, taken from a lecture by [Hahn, 2012].

1. Initialize your parameters with some random values.
2. Calculate the *forward-backward probabilities* based on your current parameters.
3. Forward-backward probabilities can now be used to calculate the expected frequencies.
4. Use the expected frequencies to estimate the parameters.
5. Repeat this until the parameters converge.

For a complete mathematical formalization, see [Hahn, 2012]. But applying the technique that has just been presented for overcoming unsupervised learning problems is one of many different possible ways to go.[Goldwater and Griffiths, 2005] for example developed a fully Bayesian approach for unsupervised POS tagging, which performs quite well. But because of relevance for this thesis, *support vector machines* are discussed next.

2.4.3 Support Vector Machines

The topic of support vector machines is indeed a very complex one. There are several different ways how to apply this classification technique. This section provides a look at the very basics of this concept to point out how this kind of classification can be useful for POS tagging and classification of words and also for chunking. [Nakagawa et al., 2001] was the first to introduce SVM[40] for unknown word guessing and his work is used as a reference for discussing SVMs regarding POS tagging. In their paper they define the principle of SVM as follows:

[40]Support Vector Machines

Support Vector Machines (SVMs) are supervised machine learning algorithm for binary classification on a feature vector space $x \in \mathbf{R}^L$.

$$w \cdot x + b = 0, w \in \mathbf{R}^L, b \in \mathbf{R} \tag{2.14}$$

SVMs aim to build a model based on training data. This model predicts the target values of test data, given test data attributes [Hsu et al., 2010]. Hyperplanes (2.14) separate sets of objects in classes, aiming to produce the most possible free space near class borders. The distance between the first object of a class and a hyperplane is call *margin*. A hyperplane is defined by training sets (2.15).

$$\{(x_i, y_i) | x_i \in \mathbf{R}^L, y_i \in \{\pm 1\}, 1 \leqslant i \leqslant l\} \tag{2.15}$$

SVMs try to find hyperplanes with maximum margins. Figure 2.4 visualizes the problem of finding maximum margins:

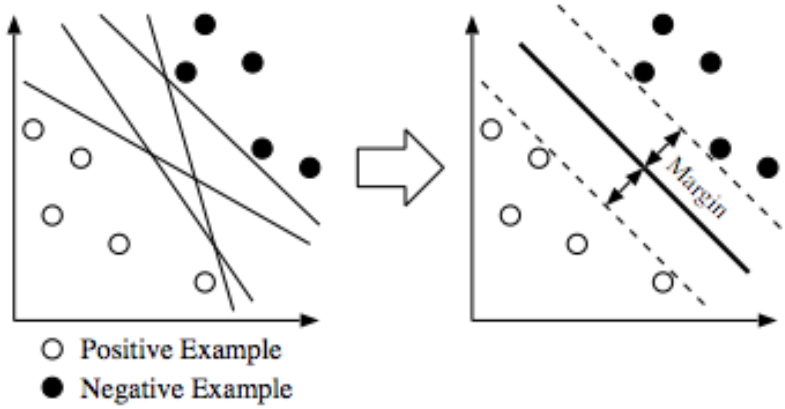

Figure 2.4: Margin maximization by training hyperplanes

But as one can imagine, not all objects can be classified by two dimensional lines. Therefore, non-linear tasks have to be mapped onto a higher dimensional space. This done by mapping the feature vectors. Because all data points appear as inner products from a linear point, we also only need the inner product in a higher dimensional space. These values aren't mapped to higher dimensions. They are calculated in $\mathbf{R}^L$ by so called *kernel functions*.

Kernel functions allow the calculation of linear functions (in this case hyperplanes) implicitly in higher dimension spaces. There are several kinds of kernel functions that can be applied, but there are four, let's say basic functions that are applied commonly [Hsu et al., 2010]. Linear, polynomial, radial basis function and sigmoid. SVMs are binary classifiers and in the case of POS tagging more than two classes have to be classified. K classifiers are created to separate a class from all other classifiers. For predicting a POS tag the following features are used [Nakagawa et al., 2001].

- **POS context:** The words next to the unknown word.
- **Word context:** The lexical forms of the two coterminous words.
- **Substrings:** Prefixes and suffixes(up to 4 chars) of the unknown word and the existence of numerals, capital letters and hyphens in the unknown word.

Back to the example presented in section 2.4.2: *"This is @PathonHauser's first tweet for AuroraApps"*. Let's say the word *AuroraApps* is the unknown word in this sentence. In fact there are two known approaches for solving this problem in the case of SVMs. Both make usage of the previously presented additional features. The first approach uses only preceding POS tags. Because there are no probabilities calculated as in HMMs(see section 2.4.2), a deterministic method has to be applied. A dictionary containing all previously used and known POS tags is used and all tags that are contained are used as possible candidates. The one that best fits the features, is taken. The second approach is using preceding and succeeding POS tags. So you treat unknown words the same way known words are treated, but by using the three previously presented features.

SVMs can also be used for chunking approaches [Kudo and Matsumoto, 2001]. The concept for this approach is quite the same, with of course slight adaptions of hyperplanes, kernel functions and features for detecting so far unknown phrases. In this case SVMs aim to identify common English phrases.

2.4.4 Clustering

Clustering is one the most important tasks of unsupervised learning problems[41]. The concept of clustering relies on one fundamental ideal and that's the application of grouping similar items to clusters. [Witten et al., 2011] defines clustering as follows:

> "These [obtained] clusters should reflect some mechanism at work in the domain from which instances or data points are drawn, a mechanism that causes some instances to bear a stronger resemblance to one another than they do to the remaining instances."

Figure 2.5 visualizes what the task of clustering aims to achieve in a given set of data items. In this example the clustering technique builds four clusters.

Applications of cluster techniques and methods are pattern recognition, classification, compression and classic disciplines like marketing or psychology [Fung, 2001]. There are many approved and commonly used different ways to express and formulate the problem of clustering.

- **Exclusive Clustering:** One data item belongs to exactly one cluster.
- **Overlapping Clustering:**An item has the possibility of belonging to several clusters.

[41] `http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/` (April 2012)

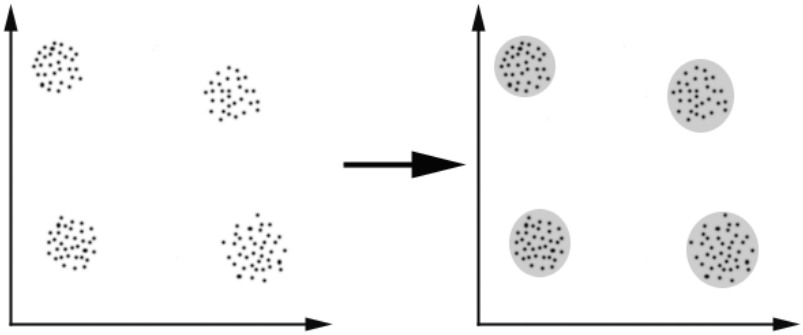

Figure 2.5: Example of clustering data points into four clusters.

- **Probabilistic Clustering:** Items have probabilities that define to which degree of probability this item belongs to a specific cluster.
- **Hierarchical Clustering:** This kind of clustering is based on the union between the two nearest clusters.

There is not only one path that leads to proper results when formulating the problem of clustering, but rather there exists several methods and algorithms to deal with their problem specific circumstances. A list of commonly used algorithms can be found in [Fung, 2001]. The most important measures in clustering techniques are distance measures between data points. Therefore, the base for applying a clustering technique is converting all data points to the same physical unit. In the case of developing a *Semantic Recommender Systems for Research 2.0* the problem of clusters can be found in tasks such as identifying users for different categories (Probabilistic Clustering) or simply in sorting multidimensional ration vectors (Hierarchical Clustering).

This chapter presented the very basics and some important techniques a developer needs to know for developing a *Semantic Recommender System*. Combining the knowledge of Semantic Web in general with the technology of Recommender Systems, opens the door for many application areas and straightens the path, for the constant improvement of any kind of information preparation. The next chapter's content, is the presentation of an idea for a system that aims to filter and recommend context aware and user relevant data based on Twitter feeds for a defined set of categorized users.

Chapter 3

Concept

After digging more deeply into some basic concepts and techniques of Semantic Web, Recommender Systems and Natural Language Processing, this chapter serves as a basic concept definition of this maser's thesis practical part. Semantic Recommender Systems aim to mine knowledge and deliver it as an end product. That's in fact a very rough generalization of the indeed quite complex task of developing a system such as this, but points out the overall benefit very well.

This chapter describes and defines the stages that lie in between developing and delivering knowledge as an end product. The following section introduces a model that aims to describe and define all stages of a development process for identifying and discovering valid, novel and potentially useful patterns in data.

3.1 Knowledge Discovery in Databases (KDD)

The term *KDD* became popular in the mid 1990s, when the volume of data on the *World Wide Web* grew so big that a normal human being wouldn't be able to stay aware of the fast growing volumes of digital media. Manual detection of changing circumstances is slow, expensive and subjective and therefore not the right task for a human. Such work needs to be automated and this automated process always follows the same or very similar rules and traces of development stages. According to these findings, the process model of KDD became more applicable than ever. This raises the question: what exactly makes KDD in comparison to similar sounding processes processes, such as usual classification or recommender task as discussed in chapter 2, so different?

The answer is indeed a very simple one. KDD serves as an overall model for displaying the path from the very beginning of developing a system, which is in most cases a raw and big set of data, to the very last step of a development process [Fayyad et al., 1996]. Therefore, this model serves as a skeletal structure for this forthcoming system. The extraction of high level knowledge from low level data is the major goal of every data mining application and also, a very vital part of KDD. Nonetheless, the task of data mining, or classification, or recommendation, is just a *part* of a complete model. Generally, these

process models are split into five particular main stages of development and conception.

1. The **Selection** stage deals with the task of understanding and exploring the relevant domain knowledge and identifying the goals of the forthcoming application. By being aware of what the system should be capable of, the developer should also be able to select an initial data set that serves as a basis for extracting knowledge.

2. **Preprocessing** is the next step in this process chain. This stage's task is to prepare the data for further processing. A typical step within this stage would be cleaning the data of noise.

3. The next task is to format and transform preprocessed data that makes further usage of the initial data as manageable and convenient as possible. According to the goals of the application, the extraction of potentially useful features is also part of this development stage. As a matter of fact, this stage is called **Transformation**.

4. This stage contains classic **Data Mining** techniques like classification, clustering, etc. The aim of this stage is to find a technique or algorithm that meets the expectations of this system.

5. One of the final steps is **Interpretation**. This stage involves the possibility and ability to return to any former stage for improving applied techniques and algorithms. The last step is the **Evaluation** of results and findings that were discovered during development.

Figure 3.1 visualizes the KDD process model. Note that every stage can and should be re-entered during development to continuously improve the system.

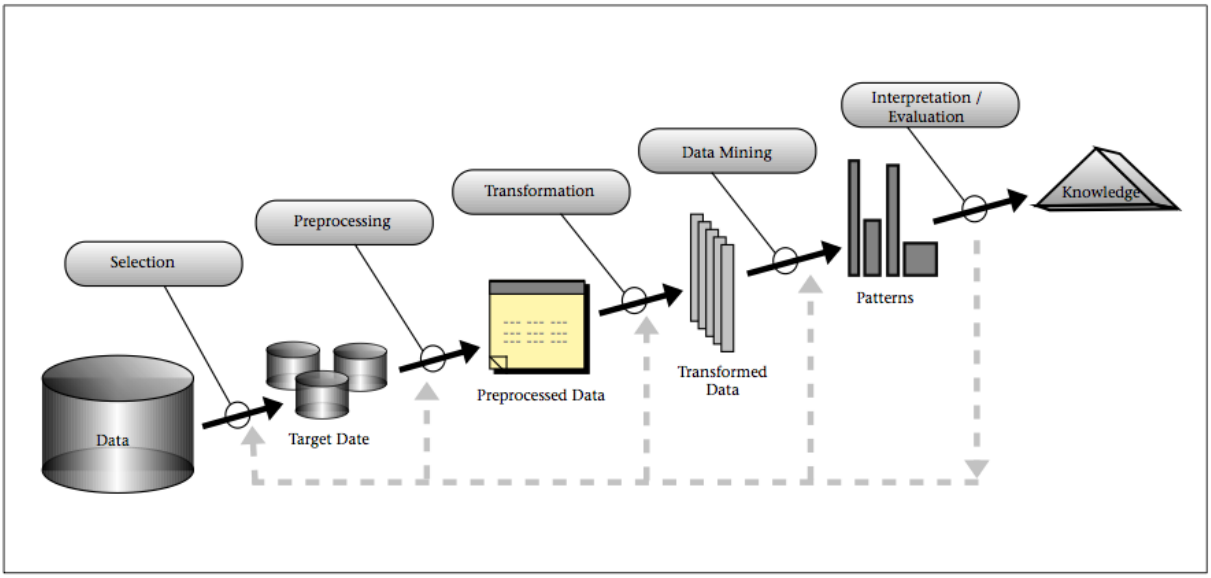

Figure 3.1: An overview of the steps that compose the KDD process.

In the following sections, the concept for the practical part of this master's thesis is presented and defined in every stage of KDD. By following the path of KDD, a detailed insight in all fundamental software design, implementation and data acquisition decisions that were made from the very beginning of this project is given. Starting with the Selection Stage, which serves as the initial starting point.

3.2 Selection

A main part of this stage invlolved digging one's nose into the topic of *Semantic Web*, *Recommender Systems*, *Natural Language Processing* and the topic of micro blogging itself. This pre-work took a considerable amount of time and ensured a fundamental understanding of how things work in this huge field of research was gained. The results and findings of this literature search are summarized in chapter 2 of this master's thesis. Whilst carrying out this extensive pre-work, the idea for the proof of concept application arose and consequently defined the goals of this project. The definition of goals led to a proof of concept application that is discussed in advance of defining the initial idea. In advance of discovering the goals of the system, one also has to find out, what form and kind of data would be a solid and promising initial dataset.

3.2.1 Thought Bubbles

The concrete idea for a proof of concept application in the form of a Semantic Recommender System for Research 2.0 came up during spending spare time on searching for potentially interesting information spreading Twitter users on the Twitter platform. Whilst doing that, it became obvious that when one browses for new and interesting users, one nearly always does this by having a look at who a friend is following or a follower list. People you consider potentially interesting, open doors to other people, which are might of interest as well. By being conscious of this, the idea of, let's call it *Thought Bubbles*, emerged.

Twitter users follow other users for specific reasons. In the majority of cases these reasons are due to similar fields of interest. Nonetheless, this doesn't mean the connection between similarly interested Twitter users is bidirectional. When social network connections aren't bidirectional, an individual user doesn't implicitly have to know his followers. Obviously, the follower is interested and involved with similar topics, to the person he or she follows. Therefore, there is a high probability that friends and other colleagues of the followed user have similar connections, which can be of certain interest to a particular user.

A user is active in several kinds of topic based bubbles, where the participating users do not necessarily know all participants of such a bubble. However, in most cases, one doesn't have just one special kind of interest and he or she is part of several topic based subsets of users. Hence, users within one user's specific bubble, might be of interest to

each other. Figure 3.2 shows an example of a network graph,which reveals the sphere of activity within diverse *Thought Bubble*.

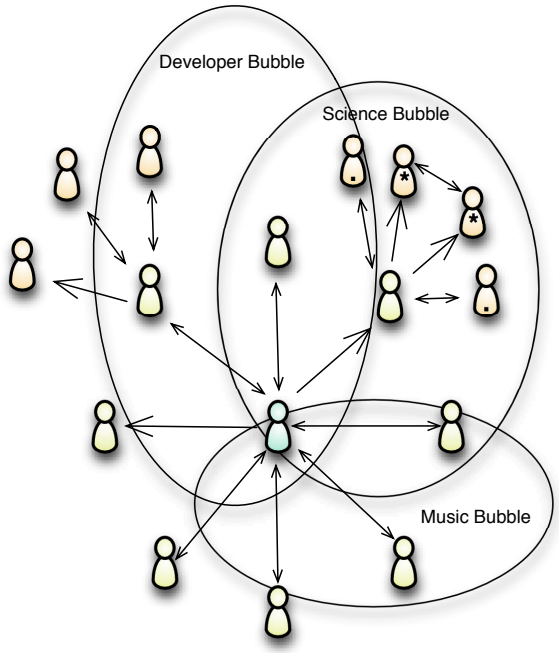

Figure 3.2: Example Twitter Network Graph with Thought Bubbles.

Users marked with a star (*) are potentially of high interest for the centered account in figure 3.2. These users belong to the same topic specific bubble, as illustrated here, to the *Science Bubble*.

This implies that following a specific user within a certain field increases the probability of finding further relevant users who are also engaging in a field of specific relevance. The missing bidirectionality of certain user connections, hints at interest only relationships. Being conscious of this, led to the concept of Thought Bubbles. This presents the possibility of recommending people and information, which is contained within a bubble and is yet to be explored by a specific Twitter user.

3.2.2 Proof of Concept Application

The proof of concept application that is being developed during this master's thesis, deals with the realization of recommending Twitter users according to the idea of *Thought Bubbles* mentioned in section 3.2.1. In fact, the focus of this is proof of concept application relies on filtering potentially interesting persons for a specific Twitter account. Figure 3.3 visualizes the applications architecture.

A client application is using the REST[1] API[2], which is supplied by the *Thought Bubble Server* and serves as the main access for third parties to this system. The usage of common

[1]Representational state transfer (REST)
[2]application programming interface (API)

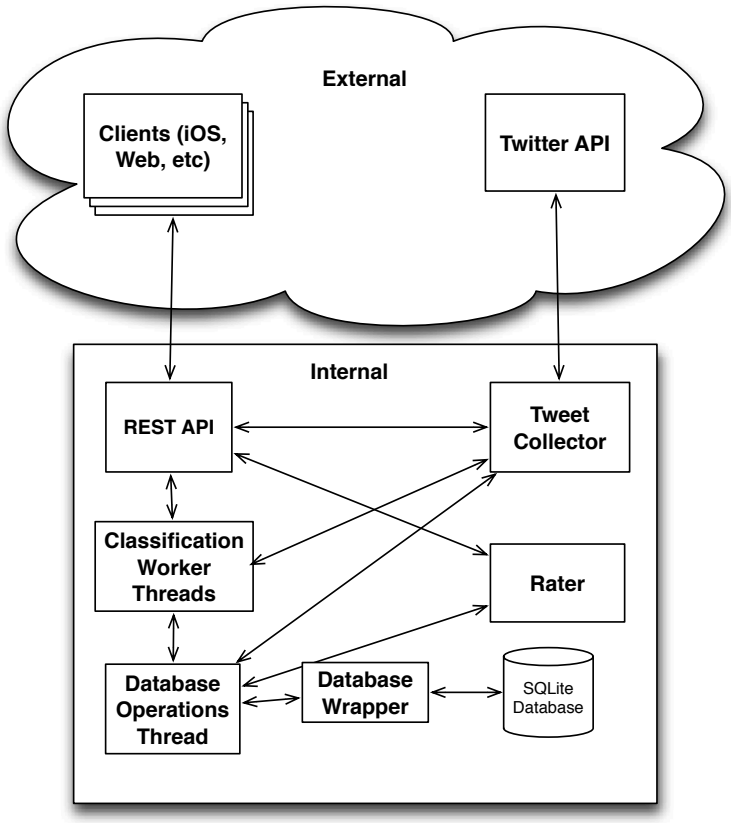

Figure 3.3: Proof of Concept Application schematical Architecture.

http requests, enables the client to subscribe to the *Thought Bubble Service* or in advance, receive recommendation data from the server. A detailed list of implemented API-calls can be found in chapter 5. When a user subscribes to the *Thought Bubble Service*, the *Tweet Collector* instance starts to fetch user lists, shown on the previously discussed example of *Thought Bubbles*. When the *Tweet Collector* is finished fetching user lists for the users that are using the service, the information for potential recommendations is stored to the database for further processing. All database operations are executed via the *Database Operation Thread*. After that, *Classification Worker Threads* are started to calculate the similarity between users. The similarity calculation is based on user specific ratios, which are discussed in depth in section 3.5. All results are again stored with help of the *Database Operation Thread*. The architecture presented is just a brief overview of the technical functionality of this proof of concept application, in order to visualize the different parts of the system that need to be developed and planned in advance. This section serves as a fundamental basis skeleton for designing and planing all particular software modules that are needed to realize the system. A detailed technical documentation can be found in chapter 4.

3.2.3 Initial Data

Based on findings describes in section 2.2, micro blogging data created by Twitter users on the according platform serves as the initial data source for further processing. To be able to categorize a Twitter account, it's necessary to take a look at one's Tweets, Retweets and in particular, one's hashtags. Consequently, a user's Tweets will serve as an initial dataset for categorization. Not only the Tweets themselves can be identified as useful data, but rather information about the entire Twitter account of a user. The following listing displays a sample response from Twitter REST API[3].


```
{
  "contributors_enabled":false,
  "created_at":"Fri Apr 04 09:02:32 +0000 2008",
  "default_profile":false,
  "default_profile_image":false,
  "description":"Semantic Web, Linked Data, Social Media, Reputation
     Systems enthusiast and researcher. Phd Student @ Graz University
     of Technology & a very bad guitar player ;)",
  "favourites_count":3,
  "follow_request_sent":false,
  "followers_count":131,
  "following":false,
  "friends_count":243,
  "geo_enabled":false,
  "id":14301066,
  "id_str":"14301066",
  "is_translator":false,
  "lang":"en",
  "listed_count":12,
  "location":"AUT",
  "name":"Selver Softic",
  "notifications":false,
  "profile_background_color":"022330",
  "profile_background_image_url":"http:\/\/a...",
  "profile_background_image_url_https":"https:\/\/si0.twimg.com...",
  "profile_background_tile":false,
  "profile_image_url":"http:\/\/a0.twimg.com\/profile_images
     \/1135082711\/semweb-softic-small_normal.JPG",
  "profile_image_url_https":"https:\/\/si0.twimg.com\/profile_images
     \/1135082711\/semweb-softic-small_normal.JPG",
  "profile_link_color":"0084B4",
  "profile_sidebar_border_color":"a8c7f7",
  "profile_sidebar_fill_color":"C0DFEC",
  "profile_text_color":"333333",
  "profile_use_background_image":true,
  "protected":false,
  "screen_name":"selvers",
  "show_all_inline_media":false,
```


[3]The following request URL was taken: `https://api.twitter.com/1/users/show.json?screen_name=selvers&include_entities=false`

```
  "status":
36 {....},
  "statuses_count":676,
38 "time_zone":"Vienna",
  "url":null,
40 "utc_offset":3600,
  "verified":false
42 }
```

Listing 3.1: Example Twitter User JSON response for an existing Twitter account.

The representation or respectively format of this API response is called JSON[4]. Only one, pre-prepared by the Twitter REST API, http request brings so much additional information for a specific Twitter account. Ratios like follower and following count, location, listing count, preferred language, etc., can all help, on the one hand to categorize a user and on the other hand, to measure the influence of Twitter accounts. One of these mentioned ratios leads directly to the pre selection stage of this KDD stage.

3.2.4 Language selection

The fact that Twitter is a multilingual platform, makes it pretty hard to handle all possible languages that occur within Tweets. Statistics based on a dataset of 5.6 billion tweets from all over the world, collected over a period of 16 months, from July 2010 to October 2011, showed that English is still the main language of tweeted content. Followed by Japanese and Portuguese. In addition to these Twitter statistical facts, the usage of the English language for scientific papers and reports for international conferences or journals, to make written science and the according information accessible for the broadest audience possible, led to the decision, to only use Twitter accounts marked as English for evaluation. Every Twitter user has to select the language he or she is tweeting in when they sign up for Twitter. Figure 3.4 visualizes the aforementioned language statistics[5].

After defining the fundamental details for the data sources to be used, the stage of preprocessing defines the first steps of preparing the initial data for further processing.

3.3 Preprocessing

The elimination of noise within the dataset is an important step during development.

Noise elimination is a very elementary preprocessing step and deals with filtering data that can be disregarded from recommendation right from scratch. The operative point in this task lies in identifying data features that point to the identification of data as noise. Let's assume we observe one specific Twitter account. Which facts could hint to

[4]JavaScript Object Notation is a lightweight object notation that is also human readable. `http://javascript.about.com/library/bljson.htm` (March 2012)

[5]Statistics are taken from `http://reyt.net/twitter-61-of-tweets-are-not-in-english/8683` (March 2012).

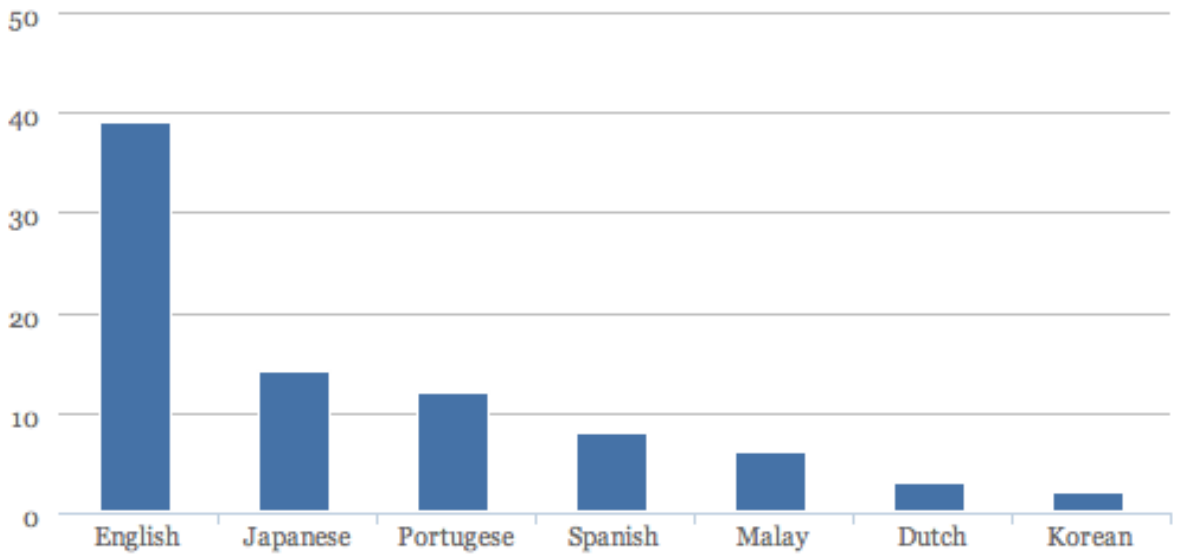

Figure 3.4: Twitter language statistics from July 2010 to October 2011 (`http://reyt.net/twitter-61-of-tweets-are-not-in-english/8683`)

a quite big probability, that a Twitter account is potentially not useful or improper for recommendations? Figure 3.5 visualizes these steps as filter chain.

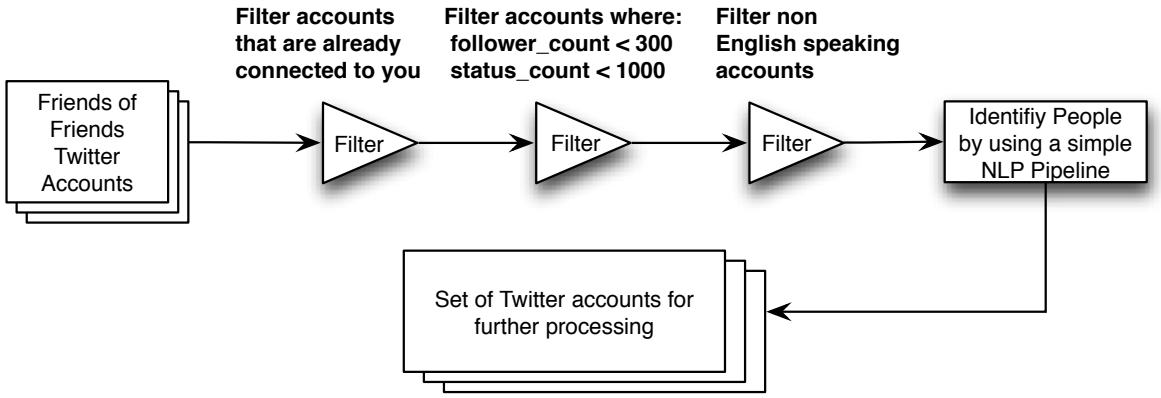

Figure 3.5: Preprocessing chain for filtering potential recommendation candidates.

- In the first place, when the service user is already following the provided Twitter account, recommending such a person, wouldn't make any sense at all and can be neglected.
- Twitter accounts that follow a huge amount of other Twitter accounts, but in comparison have very few or no followers, hints at a spamming, or just very unpopular user. Such accounts are often advertisers for various dubious purposes and are often

referred to as *Blast Followers*[6]. As a consequence of that, a user has to reach a specific limit of followers to be considered as a potential recommendation[7].

- Just signed up, unused or non data producing Twitter accounts should also be disregarded. As already mentioned in section 3.2.3, information regarding such circumstances can quite easily be extracted from Twitter REST API responses by observing tweet frequencies and the friends and follower ratio (see listing 3.1).

- As mentioned in section 3.2.4, accounts that don't use English as their main language can also be disregarded.

- What does it mean when the currently observed account represents an institution or company? In this case, it's very likely, that the account is doing a lot of advertising and therefore, it is also quite uninteresting for a recommendation within the realms of Research 2.0. Consequently, one needs to find a way to eliminate as many accounts as possible that don't belong to, at least fictionally, real people.

By filtering all accounts according to these criteria, leads to a manageable and potentially interesting set of Twitter accounts for further categorization and data mining.

After preprocessing the potential set of Twitter accounts that could be used as initial data for the task of recommendation, also the Tweets of those people also have to be preprocessed. The very first test runs of this proof of concept application led to discovering the fact, that not all Tweets in a users timeline are potentially useful for classification. In fact, Retweets of specific users influenced the identification of interests of a user. Therefore, Retweets are stripped in advance of the classification task. Within tweets, also mentions and URLs are stripped because they aren't useful for classification tasks and only tend to increase the computation time of. That's because mentions aren't contained in any test set for POS taggers or chunkers and therefore can't be properly identified. Nonetheless, personal and overall like in the case of Twitter mentions, fictional names, don't expose any information about a specific account. This finding is based on the fact that users usually mention others, whom they communicate with or are already followed by[8].

3.4 Transformation

This stage of transformation aims to transform the initial data to a manageable, easy and convenient process format.

Although Semantic Web and its technologies and standards were discussed in section 2.1, none of them were used during or for development so far. The reason for this is either that the official Twitter REST API is delivering responses packed in JSON objects and

[6] `http://www.makeuseof.com/dir/blastfollow-mass-follow-twitter-users/` (April 2012)

[7] A factor of min. 300 followers was chosen during development.

[8] `http://www.momthisishowtwitterworks.com/` (April 2012)

the fact that the commonly used web technologies like Python[9], Ruby[10] or PHP[11] are very well prepared for dealing with JSON. Also the most third party libraries support JSON by default.

Another valid reason for choosing JSON as the main data format is the fact, that JSON can and will also be used for the *Thought Bubble REST API*. Internally, JSON objects will be casted into platform specific objects. Details for the technical realization can be found in chapter 4.

Although semantic technologies like FOAF (see section 2.1.6.2) or SIOC (see section 2.1.6.3) weren't used so far, it is planned to make use of them in advance of finishing this proof of concept application. This would indeed enable this system to link people beyond the borders of Twitter. [DeVoch et al., 2011] for example, already conceived and partially approved a system using these classic semantic approaches to mine specific science related events and their participants.

3.5 Data Mining

After fetching a pre-filtered set of potential future recommendations for a *Thought Bubble* service user, it's time to pick and identify the most similar and potentially interesting connections within this set. Each of the following steps present one major operation on the initial dataset of potential Twitter users and its Tweets.

3.5.1 Tweet Frequency

The request limits of Twitters REST API forced us to filter the set of potential recommendations in advance to the main classification and categorization task. Therefore, we made use of a very popular Twitter statistic: the *Tweet Frequency*[12]. Let t_e be the time in days since a user has created his or her account and Tw_n be one specific Tweet. The result is the *Tweet Frequency* of one Twitter account tf_u :

$$tf_u = \frac{t_e}{\sum_{i=0}^{n} Tw_i} \tag{3.1}$$

The top n accounts that tweeted the most Tweets within a day, are considered for the classification task. Each *Tweet Frequency* is stored as a ratio in the database for further calculations.

[9] `http://www.python.org/` (March 2012)
[10] `http://www.ruby-lang.org/en/` (March 2012)
[11] `http://www.php.net/` (March 2012)
[12] `http://blog.kissmetrics.com/science-of-social-timing-1/` (April 2012)

3.5.2 NLP Pipeline

One major part of classifying a user's account is to apply a simple NLP Pipeline. Figure 3.6 visualizes all necessary preprocessing steps that have to be executed to get a set of comparable feature vectors.

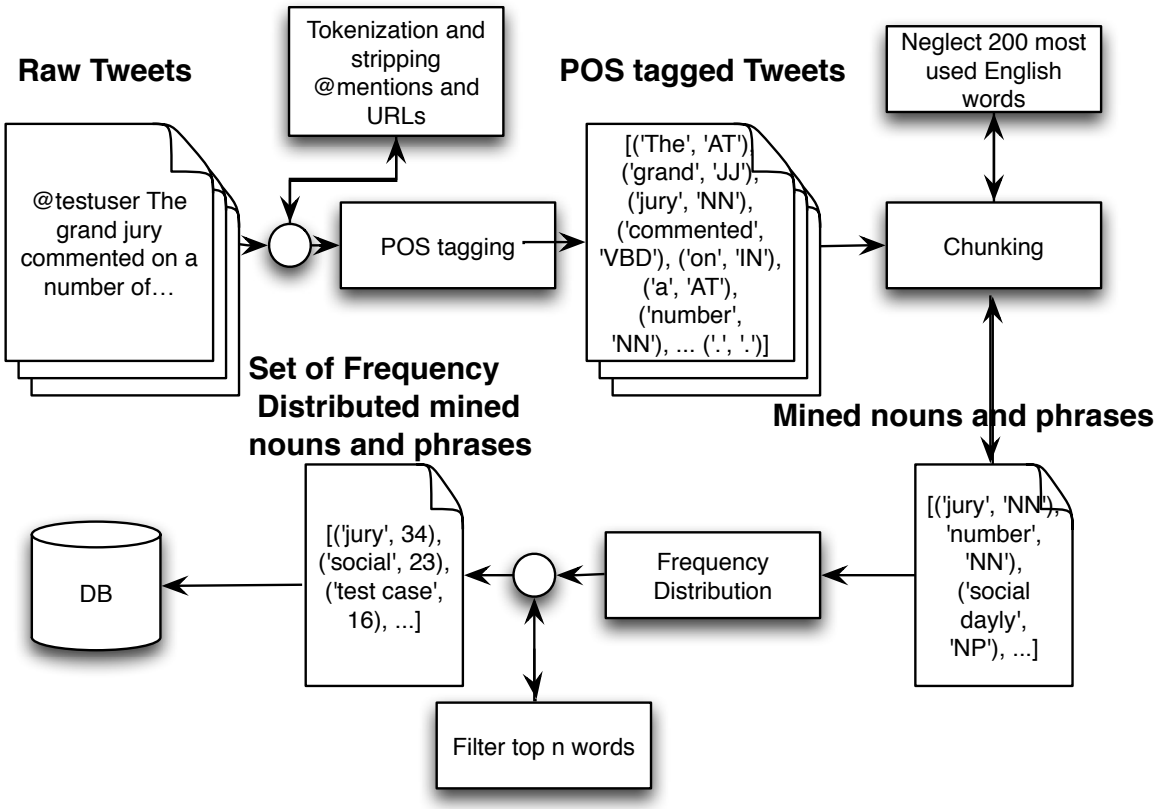

Figure 3.6: NLP Pipeline of *Thought Bubbles* proof of concept application.

The first step in this NLP Pipeline is to tokenize all accumulated Tweets of a specific user. During tokenization, mentions and URLs are stripped from the Tweets as described in section 3.3.

Based on these results, all tokenized Tweets are POS tagged, similar to the description in section 2.4. When all accumulated Tweets of an account are POS tagged, the tagged Tweets are chunked to identify common word phrases and names within the Tweet set.

The found words and phrases are processed for the 200 most commonly used English words[13]. Those 200 words are stripped from the result set because they don't have any influence in categorizing accounts. The result set contains all detected proper nouns and names, excluding mentions and URLs. Although most hashtags are recognized by the chunker as proper nouns, they are added additionally to the result set. A side effect of that is that hashtags are counted twice within a set and therefore, are twice as influential

[13] `http://www.world-english.org/english500.htm` (April 2012)

as commonly detected words. *Frequency Distribution* method is then applied to the result set to identify the top n influential and used words and phrases. In advance to that, they are stored as feature vectors for each account in the observed user set and of course, for the user, who is requesting recommendations.

Nevertheless, this part of the system is where NLP is applied. As mentioned in section 3.3, it's necessary to identify whether a Twitter account belongs to a person or something else, like a company or institution. Therefore all screen names[14] of accounts are tokenized, POS tagged and chunked. For this approach, the NLP Pipeline is very simple. The chunked nodes are observed as to whether the chunker has identified *persons*. If so, a user is taken into account for further processing. A detailed description of the chosen methods can again be found in chapter 4.

3.5.3 Retweet ratio

During the tokenization step in the NLP Pipeline that mines the feature vector of a user, another very important and useful ratio is mined. The so called **Retweet ratio**. Let $rt()$ be a function for determining the number of Retweets one Tweet has and TW_i be exactly one observed Tweet of a user. n is the amount of observed Tweets for an account and the result is the average amount of Retweets one gets per Tweet tr_u.

$$tr_u = \frac{\sum_{i=0}^{n} rt(Tw_i)}{n} \tag{3.2}$$

After computation, this ration is also stored for further comparison tasks. This ratio correlates directly with the amount of Tweets that are fetched from the Twitter Rest API, so it doesn't consider all Tweets a user has ever tweeted.

3.5.4 Measuring Similarity

After processing feature vectors for each user, it's time to compare those vectors. A classic approach for item-based recommender systems is *Cosine Similarity* (see section 2.3.1.4). This approach seems to fit the needs of this comparison task perfectly, because in comparison to systems where different items are rated by users, it's not necessary to take average ratings into account. The reason for that is based on the fact that average occurrences of words in other users feature vectors aren't interesting. Those vectors shouldn't be influenced by the usage of a specific noun or phrase of other users.

In the first place, it's important to define a specific size of comparable vectors, because both vectors have to be of the same size. To accomplish that, it's necessary to merge the features of a vector and define a specific set of features. The corresponding vector is then filled with the ratings of each users top n words and phrases vectors. Formula (2.2) is applied to compute the similarity of two Twitter accounts. *Cosine Similarities* are stored

[14]Name of a Twitter user that is displayed for other users. In most cased the real name of a person.

in the database. It's the main ratio for making a decision as to whether an account should be recommended or not.

Basically, there are four important tables within this database design:

3.5.5 Clustering

Once all potential recommendations have been measured for their similarity, it's time to compare all computed ratios. A vector consists of three so far calculated ratios:

1. Cosine Similarity of compared feature vectors of the users (section 3.5.4).
2. Tweet Frequency (section 3.5.1).
3. Retweet ratio (section 3.5.3).

Those vectors can be seen as points in a common three dimensional cartesian coordinate system. The challenge now is to find a clustering method that is able to find the best scoring ratio vectors without neglecting one of them. This conceptual formulation hints at a hierarchical clustering (see section 2.4.4). Nevertheless, the more precarious task is to find an algorithm that meets all the requirements of this task. Hierarchical clustering treats every single point as a singleton cluster. During the clustering task, all singleton clusters are merged, until all data points occur in one single cluster. [Olson, 1996] lists the four most used variations of clustering, based on several different distance measures.

- **Single Linkage Clustering:** This form of distance measuring always takes the shortest distance between data points as distance between two clusters. One major disadvantage of this method is the danger of chaining. This phenomenon is caused by only measuring the shortest distance between two clusters based on only one data point from each cluster. The two nearest data points. That means that within the merged clusters, other data points can be very distant from each other.
- **Complete Linkage Clustering:** In contrast to Single Linkage Clustering, Complete Linkage Clustering takes the two most distant data points for measuring the distance. This form of clustering tends to produce very tight clusters.
- **Average Linkage Clustering:** Just like the name says, Average Linkage Clustering computes the average distance between clusters, so all data points within a cluster are taken into account. Based on this computation, the two nearest clusters are merged during one iteration step.
- **Centroid Linkage Clusters:** This variation is very similar to Average Linkage Clustering, but uses the group centroid of clusters, to measure their distance.

These four types of hierarchical clustering will be tested to determine, which method of determining the distance between clusters is most suitable for the needs of this clustering task. Chapter 5 discusses the performance of the four techniques and explains the motivation for choosing one of them.

3.5.6 Categorization

The task of categorizing recommendations according to their topics is the last stage in this proof of concept application. This stage aims to classify the rated set of users that are recommended according to their ranks and were calculated in step 3.5.5. The fact that the content of this master's thesis deals a with question genuinely concerning the practicability of Twitter for building a Recommender System for Research 2.0, this proof of concept application aims to identify Research 2.0 content. As a matter of fact, the recommendations will be categorized as Research 2.0 be they relevant or not.

Technically, this will be accomplished by applying Bayes classifiers, as discussed in section 2.3.2.3. Therefore, test and training data sets in the form of Tweet lists have to be collected in advance of applying the classification tasks to determine, whether a user's Tweets are relevant for Research 2.0 or not. In the sense of *Thought Bubbles*, this classification task can be adjusted for additional topics. By implementing and training Bayes classifiers for each of the pre-selected topics, topic related *Thought Bubbles* can be built.

3.6 Interpretation and Evaluation

Interpretation and Evaluation is the last and maybe most important stage in a KDD process model. Within the development and first tests of this proof of concept application, many techniques and methods of data mining, preprocessing and fundamental architectural concepts had to be sharpened, reconsidered and newly adopted. Although this chain changed during development, the preceding sections of this chapter described the stages of selection, transformation and data mining in their very last state of development within this master's thesis. During early stages of development, a scientific paper was written about the first test runs and results of this proof of concept application. These first results and a brief overview of the *Thought Bubble* concept were committed to the *iKnow Conference in 2012 of Knowledge Management and Knowledge Technologies*[15].

3.6.1 Semantic Benefits

Additionally, it's planned that future *Thought Bubble* client applications will have the ability to rate recommendations as useful, ok or simply useless. Based on this continuously collected evaluation data, recommendations can be filtered and compared according to these user evaluations. Based on those rated recommendations, projects like FOAF (discussed in section 2.1.6.2) can be used to semantically link newly connected people beyond the borders of Twitter and so, enrich and support the movement of *Linked Data* and *Semantic Web*. In contrast to [DeVoch et al., 2011] who uses semantic technologies to aggregate potential connections between people who attend the same conferences or other Research 2.0 related events, it's planned to add knowledge to the *Semantic Web* and

[15] `http://i-know.tugraz.at/` (April 2012)

newly link data. The next chapter will also present a possible implementation strategy for realizing such a feature.

3.6.2 Evaluation Technique

In general, evaluation is done by analyzing Twitter accounts of researchers and colleagues, who are personally known by the developers and advisers of this project. By presenting the results to those pre-selected test users, it was possible to measure the satisfaction of the users with their recommendations. Based on their feedback and of course based on our own experiences with the recommended set of users, led us to develop increasingly better proof of concept applications. This kind of evaluation is commonly referred to as *precision and recall informational retrieval*[16]. A final evaluation and interpretation of the test results, is presented in the last and concluding chapter of this thesis.

The next chapter deals with the technical details of this system and presents methods, techniques and algorithms used that enable the system to find new and useful people, especially regarding the community and topic of Research 2.0.

[16] `http://thenoisychannel.com/2009/03/17/precision-and-recall/` (April 2012)

Chapter 4

Technical Details and Implementation

After defining the system architecture and describing the rough usage of techniques and methods for filtering potentially interesting users in the previous chapter, this chapter serves as a detailed description of all applied operations and technologies for realizing the proof of concept application. According to the KDD process model, this chapter starts with the stage of preprocessing. Nonetheless prior to that, the frameworks and libraries that are used to realize this application are presented and discussed to clarify why and for what purpose they were used.

4.1 Development Platform and Frameworks

All development tasks are coded and run on an iMac Mid 2010 with the following hardware setup:

- **Processor** 2.93 GHz Intel Core i7
- **Memory** 12 GB 1333 MHz DDR3
- **Software** Mac OS X Lion 10.7.3 (11D50)

In advance of starting to implement, Python[1] was chosen as the main programming language. In particular, Python version 2.7, because it was at time of development the most recently released and supported Python version for Mac OS. The decision to use Python is based on the fact, that NLTK[2] is a Python only library and therefore there was hardly a way to find that leads past Python. Speaking of NLTK, this framework is used for realizing all NLP tasks, like POS tagging, chunking, etc. The decision to use NLTK is based on the huge amount of provided methods and techniques in the field of classification and NLP. Also the fact that O'Reilly[3] published some very useful documentation and howto guides, played a major role during the technology pre-selection stage.

[1] `http://www.python.org/` (April 2012)
[2] `http://www.nltk.org/` (April 2012)
[3] `http://oreilly.com/` (April 2012)

The proof of concept application is implemented as Django[4] web app. The fact that a Python IDE[5] named *Pycharm 2.5*[6] provides high class capabilities for the Django framework, leads to choosing it as main development IDE. Assuming the fact that Pycharm also has integrated SQLite[7] support, SQLite was chosen as database engine.

Another very important factor when developing an application where the main task is analyzing Twitter data, is of course a reliable and fast python wrapper around the Twitter REST API. Therefore, the *Python Twitter*[8] is used for fetching Twitter data. As already mentioned in chapter 3, Twitter data is processed and received as JSON objects.

Also additional frameworks are used during development that aren't mentioned in this section. These frameworks and libraries are referenced during the entire system walkthrough according their very own use case within this application.

4.2 Database design and implementation

The usage of a database is basically necessary for storing connections between Twitter accounts and ratios, which describe similarities between people. Also feature vectors need to be stored. Assuming the fact that SQLite isn't built for storing generic objects, it was necessary to look for another solution. The usage of a third party library called *y_serial* enabled the system to store objects such as lists or dictionaries into our SQLite database[9]. An alternative to *y_serial* was also found in *pickle*. This framework enables to serialize and de-serialize python object structure from and to text files. After testing both alternatives, the decision for using *y_serial* came close because of the ability to compress and additionally annotate objects. Also the byte overhead with *pickle* was significantly bigger.

Basically, there are four important tables within this database design:

- **TwitterAndServiceuser:** An Entry of this database table represents one Twitter account. This Twitter user can whether be a usual Twitter user who is considered for a recommendation, a Twitter user who is signed up for the *Thought Bubble* service or both. Each Twitter user can be in a *n to n relationship* with each other. That means, a user can have a set of n possible recommendations but can also occur as recommendation in another service users potential recommender list.

- **ComparisonRating:** A ComparisonRating describes a similarity rating between two TwitterAndServesUsers.

- **ImportantTweet:** When a Tweet of a user has, compared to other Tweets of this user, a significant high Retweet count, a Tweet is identified as potentially interesting

[4] `https://www.djangoproject.com/` (April 2012)
[5] Integrated Development Environment
[6] `http://www.jetbrains.com/pycharm/` (April 2012)
[7] `http://www.sqlite.org/` (April 2012)
[8] `http://code.google.com/p/python-twitter/` (April 2012)
[9] `http://yserial.sourceforge.net/` (April 2012)

and also stored.

- **Recommendation:** When a user is identified as potentially interesting, a Recommendation entry is stored.
- **Category:** All categories that are implemented as classifier instances own such a table entry. These entries are to tag a recommendation for a specific *Thought Bubble* category.

Figure 4.1 displays the complete database design with all elements of a table entry. The figure was visualized by using *graphviz*[10] library.

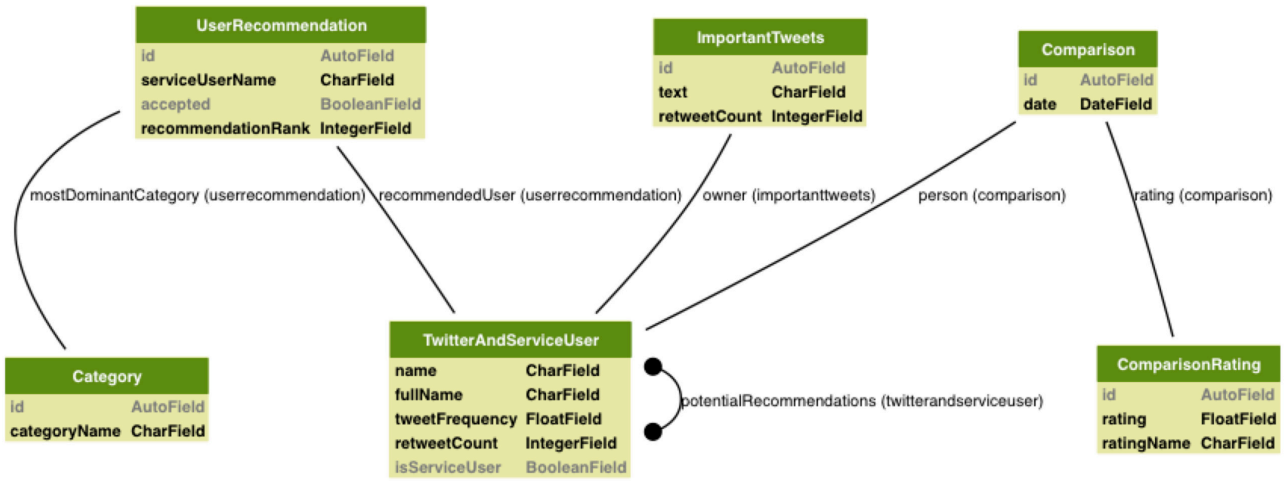

Figure 4.1: Entity Relationship diagram of database design visualized with graphivz

All database storing operations are processed by one single thread to prevent race conditions. A thread object is started when the web server starts the application and waits for jobs that are pushed as predefined objects into a common job queue[11].

4.3 Data Preprocessing implementation

Pseudo code of the applied data preprocessing steps that are previously visualized in figure 3.5 are is listed in listing 4.1.

```
list crawlPotentialRecommendationsListForUser(TwitterUser user){
  potentialRecommendations = [];
  serviceUser = TwitterAPI.getFriendsOfUser(user);
  for (friend in serviceUser){
    friendsOfFriend = TwitterAPI.getFriendsOfUser(friend);
```

[10] `http://www.graphviz.org/` (May 2012)

[11] Implemented according to the *Producer-Consumer* design pattern: `http://zone.ni.com/devzone/cda/tut/p/id/3023` (April 2012)

```
6     for (f in friendsOfFriend){
        if (checkIfUseable(f))
8         potentialRecommendations.append(f);
      }
10  }
    for (user in potentialRecommendation){
12    tokens = nltk.tokenize(user.Realname);
      postaggged = nltk.postag(tokens);
14    chunks = nltk.chunk(postagged);
      for (node in chunk){
16      if (not node.hasAttribute('PERSON'))
          potentialRecommendations.remove(user);
18      else
          user.tweetFrequency = calcTweetFrequency(user);
20    }
    }
22  potentialRecommendations = sort(potentialRecommendations, '
       descending');
    store100BestScoringUsersToDB(potentialRecommendations);
24  }
```

Listing 4.1: Pseudo code for preprocessing data.

The code lines from two to ten visualize the crawling task of friends of the *Thought Bubble* service user. All found Twitter accounts are preprocessed by calling the function *checkIfUseable()* in line 7. This function parses for the main language, the follower count, the amount of Tweets a user has tweeted so far and whether the currently observed Twitter account is already in the *PotentialRecommendation* list. The next step is to identify that the Twitter account is an account from a real person. This is done by applying a simple NLP Pipeline by using the NTLK library. The functions *nltk.tokenize.word_tokenize*, *nltk.pos_tag* and *nltk.ne_chunk*[12] are used to mine person entities. *nltk.ne_chunk* is a pretrained chunker. This kind of chunker whether just tags NE[13] or adds category labels like 'PERSON' or 'ORGANIZATION'. By parsing for 'PERSONS' it's possible to eliminate other entities. If a user is classified as a person, *Tweet Frequency* is calculated as described in section 3.5.1. In advance of this, all remaining accounts in the *potentialRecommendations* list are sorted according to their *Tweet Frequency*. The top n accounts are used for further data mining tasks.

4.4 Data Mining implementation

This is the main implementation area of this proof of concept application. Data is the most challenging part of implementation across this master's thesis practical task. The first step is to mine the comparable data of a Twitter account.

[12] A detailed API documentation about these functions ca be found here: `http://www.nltk.org/documentation` (April 2012)

[13] Named Entitiers

4.4.1 NLP Pipeline Implementation

NLTK was also used for realizing the NLP Pipeline. Basically, the task of choosing broadly high performing and reliable techniques is indeed a very challenging task. The fundamental investigation presented in chapter 2 was an essential part of this selection task and delivered the base knowledge in order to judge the usefulness of the massive amount of techniques that are supported by NLTK. The aim of this NLP Pipeline, is to mine the most used words within the users Tweets by only acknowledging words that are relevant for classification. The words are stored into user related lists.

This pipeline starts like every other NLP pipeline, fetching initial data for mining. Therefore the most recent 200 Tweets of an account are fetched. The acquisition of data is done by whether using the Grabeeter or Twitter REST API. Unfortunately, the Twitter REST API is only used when the currently observed user isn't subscribed to Grabeeter. This data acquisition strategy is motivated by Twitter REST APIs 350 requests per hour limit. All fetched Tweets are tokenized and POS tagged as described in section 4.3. Although the NLP Pipeline is so far fit for the task of identifying Twitter accounts as people, the task of chunking is different and way more complex. Only identifying named entities isn't enough. Word phrases should also be identified. Prior to the POS tagging task, mentions and URLs are stripped from the tweets. This is accomplished by applying *Regular expressions*[14]. The developed expression for eliminating mentions is visualized in listing 4.2 and listing 4.3 displays the Regex for eliminating URLs:

```
r"([@?])(\w+)\b"
```

Listing 4.2: Regex for stripping Twitter mentions.

```
1 r"(http|ftp|https):\/\/[\w\-_]+(\.[\w\-_]+)+([\w\-\.,@?^=%&amp
    ;:/~\+#]*[\w\-\@?^=%&amp;/~\+#])?"
```

Listing 4.3: Regex for stripping URLs.

NLTKs *Trigram tagger* is used[15] for POS tagging. Trigram Taggers need to be trained and as a matter of fact, an appropriate training and test set is needed. Although it was discussed to build a custom text corpora for the task of tagging Tweets, the decision was made to go with as to whether to and well approved corpus. *CoNLL2000* corpus is the product of a shared task of the corresponding conference in the year 2000 [Tjong Kim Sang and Buchholz, 2000] and is recommended and known as a solid corpus even for relatively short text artefacts[Perkins, 2010]. The fact that this chunking corpus is delivered with NLTK led to the decision of using this corpus for the first proof of concept application. In advance of applying the *Trigram tagger*, the text is chunked.

This POS tagging and chunking task is implemented as Singleton class and therefore, only one instance of this object is created, trained and used during runtime. It is therefore

[14] A detailed explanation of how Regular Expressions. word can be found here: `http://www.regular-expressions.info/` (April 2012).

[15] A trigram tagger is POS tagger based on a second order HMM (see section 2.4.2)

the bottleneck of this application. When a Tweet was chunked, the chunked data is traversed for NPs[16] and NNs[17]. Those words and phrases are then stored as list for the corresponding user. Nonetheless, before the words and phrases are stored to the database, they are filtered for common English words as already described in section 3.5.2. Although hashtags are usually detected by the chunker, they are additionally stored into the same list to strengthen the hashtagged words influence. During the task of chunking the particular tweets, their Retweet count is stored for further computation of the users *Retweet ratio*. When all fetched Tweets are processed, *Frequency Distribution* is applied to count the occurrence and frequency of different words. The top n words are stored as vectors. This NLP Pipeline task is implemented according to the *Producer-Consumer* pattern to enable partially parallel computation. A complete implementation of the core classes for this proof of concept application can be found in the submission data package corresponding to this master's thesis (see appendix A). The pseudo code visualized in listing 4.4 shows the scheme of the implementation of worker threads that process the NLP Pipeline for one user.

```
1
class TweetObserverThread(){
3
  (void)run() {
5   while true{
      job = getJobFromJobQueue();
7     if (job == nil)
        stopThread();
9     retweetFrequency = 0;
      tweets = fetchTweetsfromGrabeeterAPI(job.comparedUser()) ?
         fetchTweetsFromTwitterAPI(job.comparedUser()) : self.sleep
         (60);
11    for (tweet in Tweets){
        stripURLSandMentions(tweet);
13      tokens = nltk.tokenize(tweet.text);
        tags = nltk.pos_tag(tokens);
15      chunkTree = TringramTagger.getInstance().tagAndChunk(tags);
        featureVector = traverseTree(chunkTree);
17      retweetFrequency += tweet.retweets();
        featureVector.addHastags(tweet.hashtags());
19    }
    featureVector = applyFrequencyDistribution(featureVector);
21  user.save(featureVector, retweetFrequency);
    jobQueue.addJob(Job(user));
23 }
}
```

Listing 4.4: Pseudocode of a NLP Pipeline worker thread.

In advance of this, the cosine similarity of users is calculated and stored into the database. This task is also implemented using Pythons thread API and follows the same

[16] noun phrases
[17] Nouns

producer-consumer principle as NLP Pipeline threads.

4.4.2 Clustering implementation

After mining three promising and comparable ratios, each user is present as a vector built from those three significant values. *Cosine Similarity* followed by *Tweet Frequency* and the *Retweet ratio*. As discussed in section 3.5.5, four hierarchical clustering approaches for sorting the vectors that correspond to their ratios were tested. Notably, common sorting algorithms could not be applied as the sorting task should consider all three values within a vector. In fact, particular values of the vector had to be weighted to be able to define the influence of a ratio.

Cosine Similarity is defined as the ratio with the most influence, so *Tweet Frequency* and *Retweet ratio* should never be able to overrule this similarity ratio. In contrast to that, the other two ratios should have the same weight and therefore influence. A higher *Tweet Frequency* should overrule a lower *Retweet ratio* and vice versa. Therefore a custom distance function is developed to meet these requirements. Listing 4.5 displays the pseudo code for the developed distance function:

```
1 lambda (x, y, z, o),(u, v, w, p): (float(abs(x-u)), float(abs(y-v)
    *0.2), float(abs(z-w)*0.2)
```

Listing 4.5: This distance function defines how the distances are calculated and compared with each other.

The letters x and u represent the *Cosine Similarity*, y and v *Tweet Frequency* and z and w *Retweet ratio*, where the last two ratio's influence is lowered by weighting them by 20 percent. This aims to always take *Cosine Similarity* as the prime ratio. o and p are placeholder names to be able to identify the compared vectors according to their recommendation objects.

Now to the four different hierarchical clustering approaches. *Complete Linkage Clustering* couldn't meet our requirements. Basically, it was based on the fact that this technique of hierarchical clustering merges clusters according to their most distant points within one cluster. This behavior leads to an effect where the clustering result isn't sorted according to the nearest distance of data points, caused by not necessarily merging the nearest clusters. *Average Linkage Clustering* and *Centroid Linkage Clustering* also tend to break the strict sorting rules, but perform much better than *Complete Linkage Clustering*. Nonetheless, a *Single Linkage Clustering* approach brought the expected results. Therefore, this approach is applied for the realization of this proof of concept application. The usage of a third party Python library named ladida [18] helped to realize this computation step.

[18] `http://pypi.python.org/pypi/cluster/1.1.0b1` (April 2012)

4.4.3 Categorization implementation

After clustering was completed, all potentially interesting Twitter accounts that had been identified were classified by their topic relationship. For the needs of testing and as this is a proof of concept application, only one topic related classifier was initialized and trained so far. Nonetheless, it's a quite easy to add more topic related classifiers to the system by firstly providing topic related test and training data for the classifier and also by adding an instance of the classifier class and feeding it with this data.

As mentioned in section 3.5.6 a *Bayes* classifier is used to serve as topic classifier. NLTK also provides a ready to use *Bayes* classifier implementation, is part of the realization of this feature. Listing 4.6 contains the corresponding pseudo code for such a classifier instance. This class serves as a base class for developing other topic related *Bayes* classifiers that might have to meet slightly different requirements.

```
class TweetCategorizer(){

-(void)loadCategoryData(self, cateogry):
  document = fopen(category)
  for (line in document){
    data.append(nltk.tokenize(line))
  return data
}

-(void)trainClassifier{
  data = loadCategoryData(category)
  testSet = nltk.FrequDist(data) #get most used words for training

  trainingData = loadCategoryData(categoryTest)
  trainSet = nltk.FrequDist(trainingData)
  #train classifier
  self.classifier = nltk.NaiveBayesClaissifier.train(trainSet)
  nltk.classify.accuracy(self.classifier, testSet)

}

-(void)categorize(self, featureVector){
  #calculate probability for feature vector to be part of category
  prob = self.classifier.prob_classify(featureVector)
  return prob.prob('category')
}
}
```

Listing 4.6: Base class of Bayes classifier for classifying probability of whether a Twitter user is part of a specific category.

The stored feature vectors that contain the ranked top n words of a user's Tweets, were used for classifying whether a user belongs to a topic related *Thought Bubble* or not. In advance of this, a threshold for every single different topic related classifier instance had to be defined. Based on the title of this masters's thesis a classifier for *Research 2.0* related

content was implemented, or rather, fed with test and training data. When the probability that a users feature vector is part of this topic related bubble is at a satisfying level, a user becomes symbolically part of this specific *Thought Bubble*. The implementation of a *Twitter Bayes Classifier* is taken from an open source command line tool called *Twitter Comander*[19]. *Twitter Commander* is a twitter client with smart filtering and the ability to statistically classify Tweets. All one has to do to create a topic specific classifier is to feed an instance of the *Bayes* classifier base class with topic specific test and training data until the error rate is minimized to a satisfying level. In this case, we chose to set the maximum error limit to 10%.

4.4.4 FOAF document creation

As already mentioned in chapter 2 of this thesis, FOAF utilizes OWL to make interested inferences between people and FOAF was chosenfor the project to link people semantically. In this specific use case, the aim is to connect people according to their Twitter account names, which are unique and therefore perfectly suited for this FOAF approach. Rather than fetching and mining information from existing semantic data sources, this system generates *Linked Data*.

The idea behind this approach follows a very simple principle. Link people semantically who accepted *Thought Bubble* recommendations. The creation of FOAF files can be realized by making usage of the third party Python library *foaflib*[20]. The following listing 4.7 displays what the creation and linking of two Twitter users looks like by using *foaflib*.

```
# Create a new person
me = Person()

# Set basic attributes
me.name = "Bill Bloggs"
me.gender = "male"
me.homepage = "http://www.bill.bloggs.net"
me.add_mbox("mailto:bill@bloggs.net")
me.add_mbox("mailto:bill@megacorp.com")

# Add an account
twitter = OnlineAccount()
twitter.accountServiceHomepge = "http://www.twitter.com"
twitter.accountName = "bbloggs42"
me.add_account(twitter)

# Add a friend (someone you foaf:know), from scratch
wife = Person()
wife.name = "Belinda Bloggs"
wife.gender = "female"
wife.homepage = "http://www.belinda.bloggs.net"
wife.add_mbox("mailto:belinda@bloggs.net")
```

[19] `https://github.com/hmason/tc` (April 2012)
[20] `http://code.google.com/p/foaflib/` (May 2012)

```
23
# Add Twitter account for wife
25 twitterW = OnlineAccount()
twitterW.accountServiceHomepge = "http://www.twitter.com"
27 twitterW.accountName = "belinda42"
wife.add_account(twitterW)
29
# add wife as friend
31 me.add_friend(wife)

33 me.save_as_xml_file("my_foaf.rdf", format='xml')
```

Listing 4.7: Example of creating a FOAF connection between two Twitter users. The example is taken from foaflib documentation.

This code whether creates a new foaf profile for a user or enriches an existing one with information about their newly established Twitter friendship. Systems such as [DeVoch et al., 2011] are now enabled by applying common reasoning to detect new connections between people. Listing 4.8 is *RDF*-output for executing the data presented in listing 4.7.

```
1 <?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
3    xmlns:ns1="http://xmlns.com/foaf/0.1/"
   xmlns:ns2="http://webns.net/mvcb/"
5    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
7   <rdf:Description rdf:about="#me">
     <ns1:mbox rdf:resource="mailto:bill@megacorp.com"/>
9    <ns1:mbox rdf:resource="mailto:bill@bloggs.net"/>
     <ns1:name rdf:resource="Bill Bloggs"/>
11   <ns1:holdsAccount rdf:nodeID="_70cb2451-21aa-45e9-a454-3
        afee1dee55b"/>
     <ns1:homepage rdf:resource="http://www.bill.bloggs.net"/>
13   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
     <ns1:gender rdf:resource="male"/>
15   <ns1:knows rdf:nodeID="_e2fc3984-8cc1-4667-b1c8-e16593af7f82"/>
   </rdf:Description>
17 <rdf:Description rdf:nodeID="_e2fc3984-8cc1-4667-b1c8-e16593af7f82">
     <ns1:mbox rdf:resource="mailto:belinda@bloggs.net"/>
19   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
     <ns1:name rdf:resource="Belinda Bloggs"/>
21   <ns1:homepage rdf:resource="http://www.belinda.bloggs.net"/>
   </rdf:Description>
23 <rdf:Description rdf:nodeID="_70cb2451-21aa-45e9-a454-3afee1dee55b">
     <ns1:accountName rdf:resource="bbloggs42"/>
25   <ns1:accountServiceHomepage rdf:resource=""/>
   </rdf:Description>
27 <rdf:Description rdf:nodeID="_857b5126-69ab-4bad-a2f1-83500ce34fdf">
     <ns1:primaryTopic rdf:resource="#me"/>
29   <ns2:generatorAgent rdf:resource="http://code.google.com/p/foaflib
        "/>
     <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/
```

```
        PersonalProfileDocument"/>
31    </rdf:Description>
</rdf:RDF>
```

Listing 4.8: RDF output.

This code establishes a connection between two people who own a Twitter account and describes the relationship between them. As a byproduct, this linked data can be used again as a source for discovering new connections. By exploring the connections of the newly linked person, new connections may be found within the set of other connections this person has.

4.5 Complete Architecture and UML Diagram

The overall architecture and software design of the system part that is responsible for the complete recommendation task is displayed in figure 4.2.

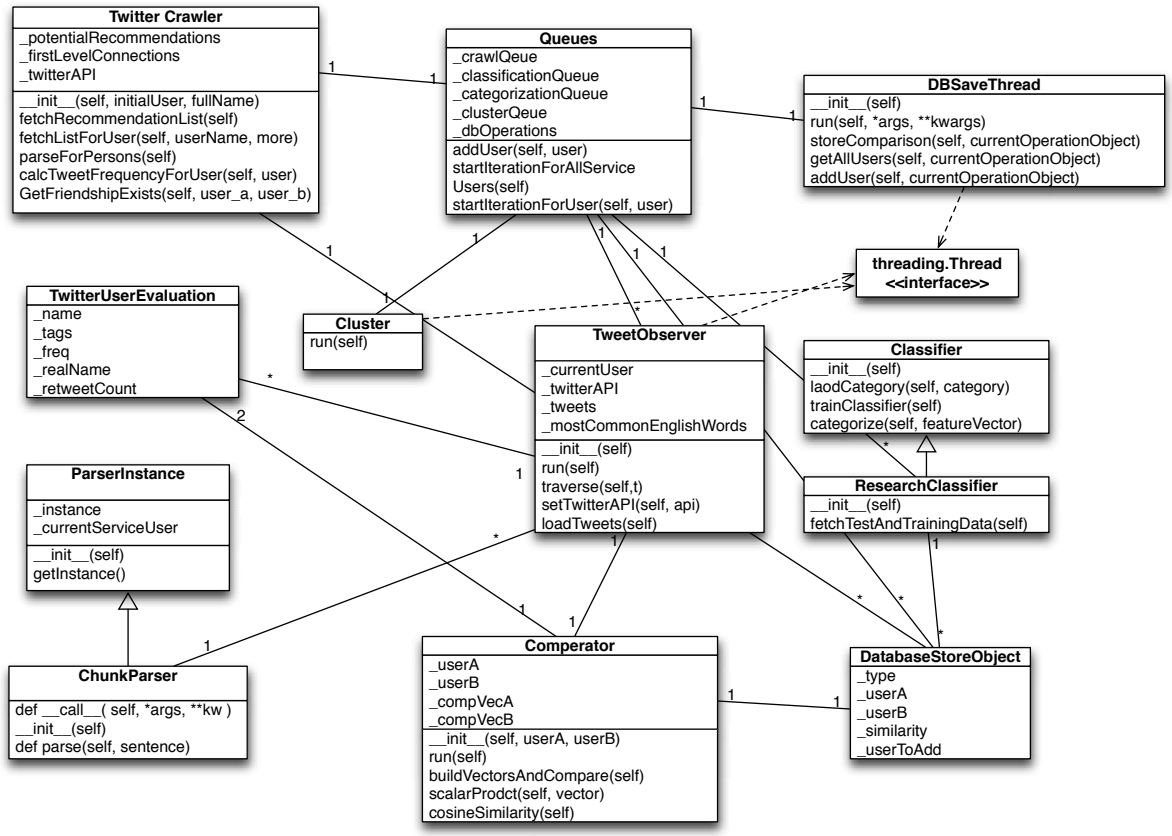

Figure 4.2: UML class diagram of core components.

One iteration for computing the top-n recommendations for a user follows the following sequence description:

1. A user is added to the corresponding queue via the *addUser(self, user)* method. By initializing a *DatabaseStoreObject*, the user is added to the database queue and stored automatically by the *DBSaveThread*.

2. Subsequently, the classification can be started for one specific user by calling the method *startIterationForUser(self, self)* or by starting an overall new iteration for all service users by calling the method *startIterationForAllServiceUsers(self)*.

3. When an iteration is started, the *Twitter Crawler* fetches the potential recommendation list and stores it to the database. *DatabaseStoreObject*s and the *DBSaveThread* are again used to process this task.

4. When a user's potential recommendation list is ready, a new job is added to the corresponding queue.

5. Three *TweetObservers* threads are started per service user for applying the NLP-pipeline and computing their similarities. When all ratings for a potential recommendation are computed, a *Comperator* object is used to compare the users and store the results to the database.

6. When all data mining methods have been applied for all potential recommendations, a cluster task is added to the corresponding queue.

7. The results of the clustering task are again stored to the database and a categorization job is added to the corresponding queue.

8. In advance of this, the Tweets of the top-n recommended users are classified by applying topic related *Classifier*s.

9. After identifying the Twitter users main tweeted about topics, one iteration for one user is complete.

The following chapter presents the test results for this proof of concept application.

Chapter 5

Results and Evaluation

All test runs that were conducted on the proof of concept application that has been previously described, serve as data for this chapter's content: The evaluation and interpretation of recommendations for several Twitter users.

This chapter also describes the setup of the test runs and how the results were evaluated and performed overall. In addition, this chapter closes with an in depth discussion about the evaluated results and future work.

5.1 Evaluation Technique

To be able to measure and judge the test results, a classic *Precision and Recall* approach is applied, which is kind a standard in pattern recognition and information retrieval systems, especially NLP systems [Melamed et al., 2001]. *Precision* is a fraction of the retrieved instances that are relevant to the search task. Y is the set of relevant instances and X is a set of retrieved instances. Precision ($precsion(X|Y)$) is defined as follows:

$$precision(X|Y) = \frac{\mid X \cap Y \mid}{\mid Y \mid} \tag{5.1}$$

Recall ($recall(X \mid Y)$) is the fraction of relevant instances to the amount retrieved instances and is defined as follows:

$$recall(X|Y) = \frac{\mid X \cap Y \mid}{\mid X \mid} \tag{5.2}$$

In the case of determining the accuracy of a classification task, such as in the case of identifying Twitter accounts as potentially interesting or not, the terms *true-positive*, *true-negative*, *false-negative* and *false-positive* are commonly used to describe a classification result.

- **true-positive (tp):** A user who is classified as interesting and also really is.

- **true-negative (tn):** An account that was not useful and therefore was not classified as interesting.
- **false-negative (fn):** A Twitter account that is interesting for a user, but wasn't classified as such.
- **false-positive (fp):** This user is not interesting but was classified as such.

Nonetheless, one of these terms isn't really useful for measuring the quality of a recommender system and therefore, doesn't say much about the quality of such a system. When one thinks of a search engine that delivers results for a specific search query, one never cares about potential matches that weren't presented as results. All that matters to the user in terms of quality of service are items that are delivered to the user and consequently faces directly. Therefore, *Precision and Recall* are very useful ratios for determining the quality of recommender systems because none of these ratios includes *tn* results. Figure 5.1 visualizes the confusion matrix for these terms.

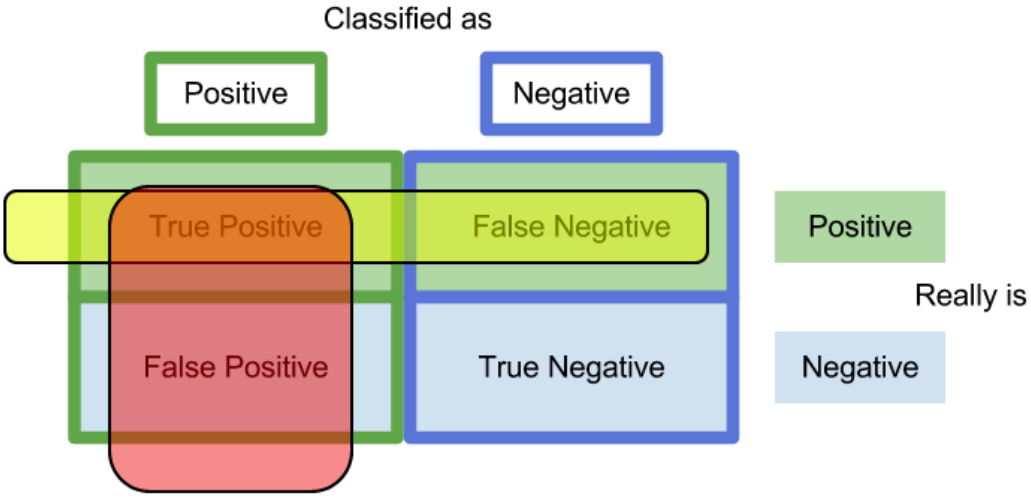

Figure 5.1: Visualization of Precision and Recall.

Red highlighted parts (tp and fp) identify the properties of *Precision* in this context and yellow marked terms (*tp* and *fn*) are parts of *Recall*. In fact, *Precision* and *Recall* are defined as follows:

$$precision = \frac{tp}{tp + fp} \tag{5.3}$$

$$recall = \frac{tp}{tp + fn} \tag{5.4}$$

The probabilistic interpretation for *Precision* is the probability that a Twitter user is identified as relevant. *Recall* can be seen as the probability that a relevant Twitter user

is also identified as relevant. Additionally, the *Accuracy* ratio is calculated by using the following formula:

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{5.5}$$

Accuracy describes the proportion of all true results to the amount of tested data. Although *tn* results don't say much about a recommender system, they are very important for evaluating a classifier.

Moreover, the statistic test accuracy is calculated by applying the traditional *F-measure* or also referred to as *F1-score* [Olsen and Delen, 2008]. This measure considers *Precision* and *Recall* and calculates their *harmonic mean* [1]. One can also refer to this ratio as weighted average of *Precision* and *Recall*. It is calculated by applying the following formula:

$$F1 - score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.6}$$

All four mentioned ratios follow the same rating principles. 0 is the worst possible rating and 1 the best.

5.2 Test Setup

All participating users of the test runs were given a set of top-25 recommendations for their Twitter account. Consequently, all evaluated results rely on the subjective opinions of those test-users, which evaluated whether a recommendation is of use to them, or irrelevant. The decision to use a third party library for categorizing the recommended Twitter accounts led to the decision not to include categorization in the test run that was presented. That means, only self written system modules are part of this test run. All that has to be done to be able to categorize Twitter accounts according to their main tweeted about topic, is to provide test and training data for a specific topic and feed it to an instance of the base class described in section 4.4.3. This task has to be repeated until the *Bayes classifier* is able to categorize users at a satisfying error rate.

All test-users had to fulfill pre requirements to be considered as potential testers.

- Using English as their main Tweet language: English only preferred.
- Following at least 50 other accounts.
- Tweeted at least 2000 status updates (Retweets and mentions do not count as status update).
- Followed by minimum 300 other Twitter users.
- The test-users should have an academic background to enable the verification of whether their topic related interests were met.

[1] `http://mathworld.wolfram.com/HarmonicMean.html` (May 2012).

Ten Twitter accounts were chosen to serve as test-users. The numbers for these threshold ratios were chosen during development based on the amount of average fnumber of recommendations found for a Twitter account. Several test runs showed that these numbers lead the system to find less than 1500 potential recommendations, which are used for further processing. Consequently, the steps of *Preprocessing* cuts this set of Twitter accounts down to approximately 300 accounts. Thereupon, the remaining recommendations are analyzed by the *NLP-Pipeline*. By applying the *Clustering* task, 25 accounts were recommended per test-user.

As a matter of fact, it is very difficult to determine concrete figures for *fn* and *tn* results within this test set because tmore than 1000 potential accounts were fetched for further processing per test user. Hence, a supervised test set was prepared to determine how well the system is able to separate relevant data from others. The test setup for this second test-run looks as follows:

- 50 Twitter users
- 20 of those should be classified as relevant
- 30 of them shouldn't be classified as relevant

Usually, the top-25 accounts are taken as recommendations. Previously executed tests during development have shown, that when a recommendation similarity ratio scores lower than **0.9**, the currently observed Twitter account should be disregarded and presented as a potential adjuvant account. Figure 5.2 shows the test results for 22 tested users that were compared to each other and to a test set of noise users to determine a threshold, where positive classified users shouldn't fall below this value. All optimal thresholds visualized in figure 5.2 identify the threshold for a specific user, where 75% of accounts that should have been classified as potentially interesting also were classified as such. As a matter of fact, a mean of all those optimal thresholds was calculated and now serves as an additional threshold to the top-25 k-NN approach. These tests were executed in April 2012.

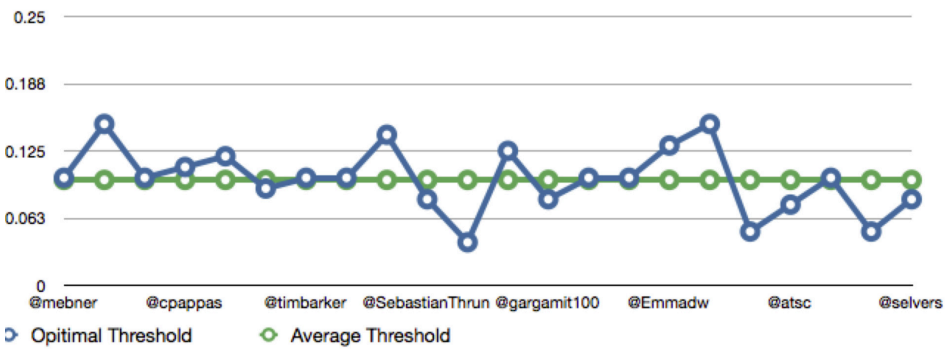

Figure 5.2: Average threshold for 22 hand picked test users.

5.3 Test Results

The following two subsection present the test results for the supervised and the real world test. Both tests were run between May 23rd 2012 and May 31st 2012. Repeating those tests would probably bring different results because new tweeted content would influence the similarity measures and of course the set of recommended accounts. Therefore, the test results are significant within the mentioned period of time but all manually selected accounts for applying this test will have to be reviewed again to guarantee a supervised test environment.

5.3.1 Real-world-test

After delivering the personalized top-25 recommendations to the test users, except in the case of @selvers who got ten, on average it took 2 to 3 days for the users to supply feedback. The user evaluations are listed in table 5.1. One test run took ten to fifteen minutes.

Table 5.1: Test results for 10 test users

account	**# recommendations**	**tp**	**fp**	**precsion**
@gekitz	25	22	3	0.88
@richmarkt	25	23	2	0.92
@PathonHauser	25	19	6	0.76
@selvers	10	4	6	0.4
@bitfluter	25	18	7	0.72
@florianlionel	25	15	10	0.6
@timbuckteeth	25	4	21	0.16
@alexmarkt	25	24	1	0.96
@andidol	25	23	2	0.92
@mebner	25	3	22	0.12
10	235	155	80	Ø0.644

155 of **235** recommendations were evaluated by the test users as interesting, of which **80** were rated as *not really useful*. The average *Precision* was computed by dividing all *Precision* ratio through the amount of participating test users.

In advance of this real world test a supervised test was executed to back these figures. All 10 of the test users were asked if their recommendations interests and tweeted content fits their own interests. They all answered that the recommendations they described as useful, all fit their interests and aren't just interesting in general.

In addition to observing the average *Precision*, *Standard Deviation* is a very interesting ratio that displays, how far the test values are spread out. One can see this ratio as a mean deviation from an average value. In particular, *Sample Standard Deviation* is applied and is defined as follows:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{K}(x_i - \bar{x})^2} \tag{5.7}$$

N is the amount of test samples, ten in our case. The average *Precision* is represented by $\bar{x}$ and x_i is the *Precision* of a test run for one of ten test users. The *Standard Deviation* for the ten tested users is **31.5%**. Figure 5.3 visualizes the corresponding *Gaussian bell curve*[2].

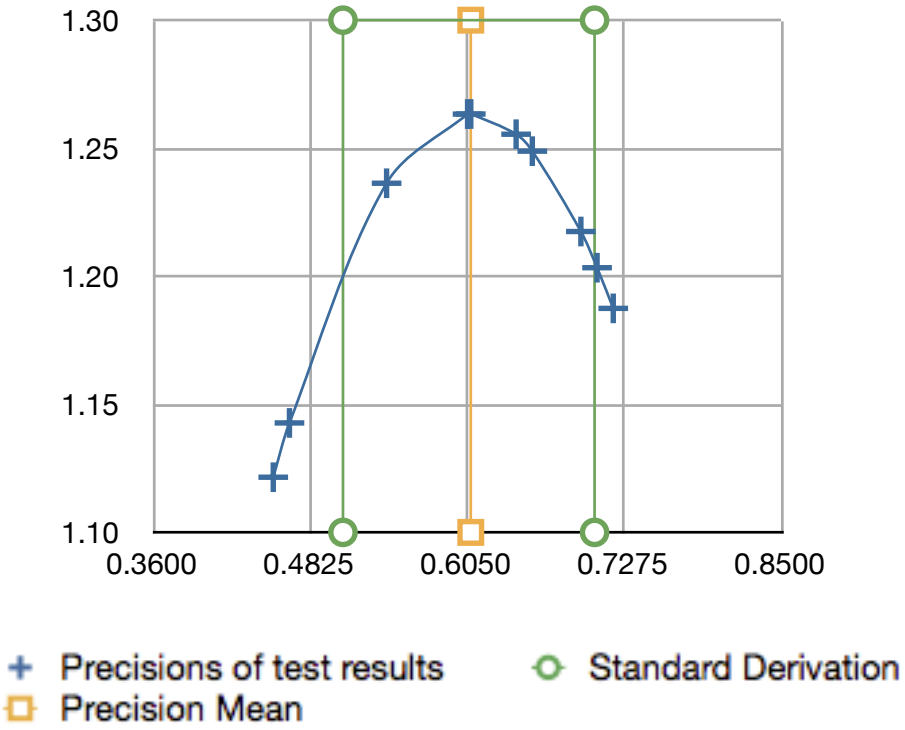

Figure 5.3: Gaussian bell curve and Standard Deviation of ten test users

5.3.2 Supervised test-run

The supervised test-run returned the results visualized in figure 5.4 . Twitter accounts marked with a star *(*)* should be classified as relevant. Within this test-run, only the similarity ratio is used for evaluation whether a recommendation is relevant or. The reason for this decision is based on the fact that *Tweet Frequency* and *Retweet Count* are pure statistical ratios, whereas the similarity ratio relies on classification tasks and the result of the implemented *NLP-Pipeline*.

16 accounts were classified as relevant, where **3** of them weren't marked with a star

[2] `http://mathworld.wolfram.com/NormalDistribution.html` (June 2012)

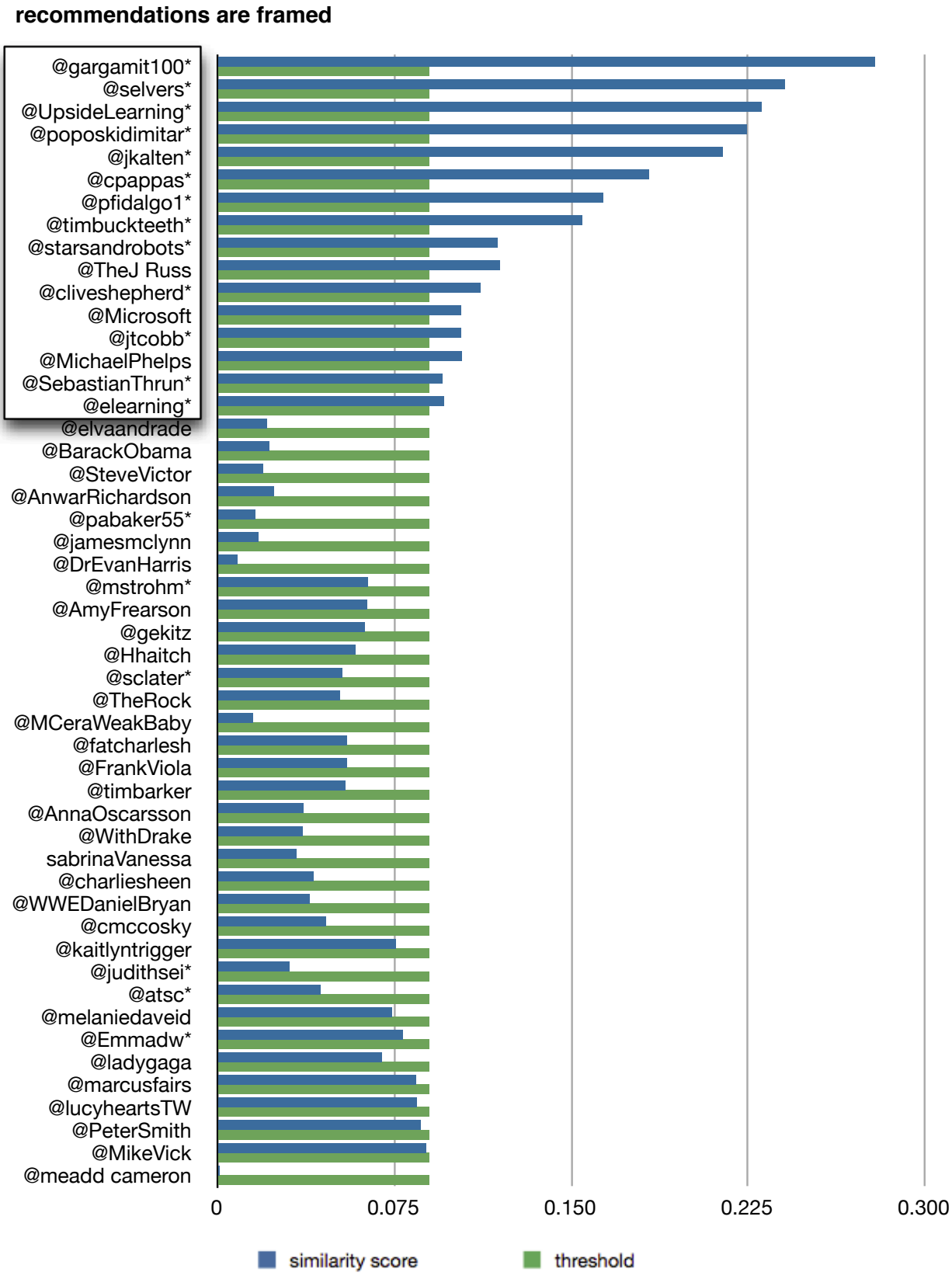

Figure 5.4: Supervised test run for @mebner.

and therefore, should't have been classified as such. Usually the system filters the top-25 recommendations, but only 16 had a better similarity score than **0.9**. Seven of the accounts that should have been classified as interesting, weren't. Table 5.2 summarizes the results in a confusion matrix.

Table 5.2: Confusion matrix for supervised test run.

-	**Positive**	**Negative**
Positive	13	7
Negative	3	27

Based on these figures the following ratios were calculated (see table 5.3).

Table 5.3: Evaluation ratios for supervised test for @mebner.

ratio	*score*
Precision	0.813
Recall	0.65
Accuracy	0.8
F1-Score	0.722

5.4 Test evaluation

The probability that a recommended item is relevant is **81.3**%[3]. Although this figure looks very promising the real world test scored significantly lower (64,4%) than the supervised one. In particular @mebner and @timbuckteeth scored far beyond the rest of the tested users. After taking a closer look at their Twitter accounts and talking to them personally, the bad ratings seemed to be caused by the set of people they are following. The two users don't seem to follow only people they are interested in, but rather follow people for status or business motivated reasons. Therefore, the pre-selection of potential recommendations often delivers a set of users that don't contain as many satisfactory accounts as usual[4]. Additionally, the similarity scores of the top-25 recommendations were significantly lower than the scores of the recommendations of the other 8 test users. On average, the difference was up to **0.15**. Bearing in mind that the pre-defined threshold for considering a recommendation as potentially useful or not (0.9), this gap is quite large. As a matter of fact, the presented recommendations are a kind of reflection of which accounts one follows and therefore, the following of users for reasons other than topic or tweet content harms the recommendation results. A way of overcoming and respectively

[3]Precision

[4]800 to 1500 accounts per iteration

improving the bad results for those users would be for example, to limit the recommendations to user specified topic related *Thought Bubbles*. Nonetheless, this circumstance doesn't harm the integrity of the system. The reason for that is associated with fact that per iteration, only a small section of all users that are potentially interesting are observed. Subsequently finding a set of useful recommendations, takes more iterations and so more time. In these cases, it is probably more effective to recommend less people, mainly those with a high similarity score.

Another interesting observation made during the process of generating recommendations for @selvers was that this account allows an online service to tweet automated tweets. In this case, this account tweeted the same sentence every day. Additionally, when this user has a very low tweet frequency, this single sentence gets an enormous weight when it is processed by the NLP-pipeline. Therefore, nearly all recommendations are based on this single Tweet. As a matter of fact, recommendations computed by this *Thought Bubble* service are reciprocal. Also the *Standard Deviation* computed in section 5.3.1 emphasizes, that the results are relatively wide spread around the average *Precision* rate. This behavior is caused by the quality of tweeted content and following behavior of the test users. Although, the fact is that the lions share of the tested users had a *Precsion* ratio above the average.

Another related problem is (as already mentioned in chapter 4) request limitations of Twitters REST API. During these tests, multiple Twitter developer accounts were used for all different Twitter API calls to maximize the request limit. Although these actions were very useful for fetching more information and potential recommendations for further processing, the limit still seems to be a show stopper when it comes up to generate recommendations for multiple accounts. The fact that *Grabeeter*s user base isn't as big as expected, increases this problem. As a matter of fact, it's hardly imaginable that the *Thought Bubbles* service will ever be a live service, but rather a services where one is able to sign up and receive recommendations from time to time. One possible use case that was implemented during this test phase, is to to add the recommendation list in form of a Twitter list to the subscribed user's account, which generally generated good feedback.

The rate of relevant classified items (13) in comparison to all available relevant items (20), is referred to as *Recall* and is **0.65**. Consequently 65% of available interesting accounts were recommended. Although this figure doesn't seem to be particularly high, *Recall* hasn't that much of weight and importance in the field of recommender systems. Although this ratio is quite useful for supervised environments, it loses importance in real world application. In the case of this system, only the top-25 recommendations are presented to the user. So it doesn't matter to the user how many relevant users are really in the set, but rather only the set one has access to. When there is a set of 1000 observed and compared users and this set contains for example 99 relevant accounts, what would that mean for the *Recall* ratio? Let's imagine all top-25 recommendations are part of those 99 relevants, the *Recall* would still be very low. 25.3% to be precise.

Also the *accuracy* is only helpful in a supervised test environment when one talks about recommender systems in general. With a rate of **80%** this ratio seems to be satisfying. As it displays the overall correctness of classification taking *tp*s and *tn*s into account, there's not much more to say about this ratio.

The combination of *Precision* and *Recall* called *F1-Score* scored **0.722**. This commonly used ratio for binary classification tasks describes the test accuracy of this supervised test run.

So far, these test results and ratios seem to be very promising and ok but how do similar services perform in similar tests perform? [Hannon et al., 2010] developed a similar recommender system for Twitter related content. They also used *Content* and *Collaborative Filtering* approaches for realizing their system based on the *Lucene*[5] platform. Their test results were interpreted as very positive and promising with a precision between 11% and almost 25% while they tested several different implementations for 15 test users. In comparison, the *Thought Bubble* proof of concept application reached a precision of up to 81.3% in the supervised test and an average of 64.4% during the real-world-tests. That means we have a twice as successful *Precision* where the lowest, which scored @mebner, is still in-between their average precision rates.

[5] `http://lucene.apache.org/core/` (June 2012)

Chapter 6

Conclusion and Future Work

Classification and recommendation of user profiles in social networks isn't just a Twitter related topic but can be used and applied for similar networks as well. The appliance of such services aims to connect similarly interested people, especially regarding their scientific interests or expertise [Stankovic et al., 2010]. Nonetheless, the tests in chapter 5 have shown, that this service isn't just useful in the field of Research 2.0, but rather can be used for connecting and recommending people beyond the borders of a specific topic. After all, by applying classification according the users strongest interests (see section 4.4.3) scientific motives of users can be detected and classified.

The *Thought Bubble* service and its concept aims to mine potentially useful new Twitter accounts, no matter what interests the service user has. Hence, the feature of categorization, still enables the user to limit recommendations to a specific topic.

Recommending novel and useful new users and following them, automatically generates a personalized stream of information, similar to a search engine like *Google*[1]. This isn't just a fast and convenient way for finding new interesting people, but rather a way to create one's personal network of people, which might be able to answer your questions or influence your work.

Although this proof of concept applications test results (see chapter 5) are very promising and of course have proved the concept, there is still a lot of work to do, to make this system applicable for real world usage. Major limitations like the Twitter REST API request limit have to be overcome and also the accuracy of classifications has to be sharpened and approved to meet the user's requirements for an outstanding recommendation tool that's as best as possible.

As mentioned in section 4.4.4, linking accepted recommendations with the service user by using FOAF technology could lead to new users beyond the borders of Twitter. Although the possibility that these newly found connections own Twitter accounts is existent, additional ways have to be found to filter again potentially interesting content from noise. [DeVoch et al., 2011] also used FOAF for mining connections between users on social platforms and it's there that this work could serve as an example how to realize

[1] `http://www.google.com` (May 2012)

this feature.

After doing all the theory research presented in chapter 2 and after implementing and testing this proof of concept system, the base questions that led to this master's thesis (see chapter 1) can be answered:

- Twitter is indeed useful for discovering new connections between researchers in similar or same subject areas. The test results of the *Thought Bubble* service have shown that recommendations for specific interested Twitter users can reach a satisfying rate of precision (see section 5.3.1).
- It is possible to recommend Twitter users based on their tweeted content, although Tweets aren't the only factor for determining, whether an accounts is a potential recommendation or not. Multiple factors like *Tweet Frequency*, *Retweet Count*, language, the amount of followers and the amount of status updates in general all influence this circumstance (see chapter 4).
- This proof of concept application is able to separate noise from useful data at a rate of up to 80% (see chapter 5). Although the real world tests scored partially lower than the supervised one, noise can be eliminated at a satisfying rate of error.
- There were indeed appropriate metrics found to measure the significance of Twitter users and Tweets. The main ratios for recommendations are *Cosine Similarity, Tweet Frequency and Retweet Count* (see chapter 3).

The fact that the developed system started to move away from using semantic technologies (see section 2.1) for discovering new connections between users, was motivated by the existence of the system developed by *DeVocht* [DeVoch et al., 2011] one year before this master's thesis started. Nonetheless, the additional feature for enriching online available semantic data (see section 4.4.4) supports the semantic web movement by semantically linking new established Twitter connections.

Another notable fact of the *Thought Bubble* concept is, that recommendations and the metrics for filtering those recommendations change dynamically in every new iteration, because only the 200 newest Tweets of a service user are used for processing one's potential recommendations. That means, that the system is able to stay context aware over time and changes the recommended accounts according to new and fresh tweeted content.

The next step will be implementing a web app to make the *Thought Bubble* service usable for everyone. Also a decision of which topic related classifiers will finally be implemented and supported will play a major role in the next few weeks and months. The collection of test and training data for every single supported topic related classifier will take a lot of time in order to guarantee an appropriate level of accuracy. Also an iOS Twitter client that supports and integrates the *Thought Bubble* concept is being planned. All in all, the proof of concept system has turned out to be a very useful and promising recommender tool that probably has a bright future as a tool for finding new connections and interesting information on Twitter not only for scientists, but rather for everyone.

Appendix A

Appendix

The attached .zip file contains:

- This master's thesis as LaTeX project.
- Concrete figures for all diagrams in a *Numbers* file.
- All test results as *.txt* files.
- The source code implemented in *Python*.
- This master's thesis as *.pdf* file.
- A *readme.txt* file that describes all necessary steps for setting up a test environment for the *Thought Bubble* service.

Bibliography

Adomavicius, Gediminas and Alexander Tuzhilin, 2005. *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions*. (6). http://ids.csom.umn.edu/faculty/gedas/papers/recommender-systems-survey-2005.pdf.

Beham, Günter, Barbara Kump, Tobias Ley, and Stefanie Lindstaedt, 2010. *Recommending Knowledgeable People in a Work-Integrated Learning System*. http://www.sciencedirect.com/science/article/pii/S1877050910003169.

Bell, Robert M., Yehuda Koren, and Chris Volinsky, 2008. *The BellKor 2008 Solution to the Netflix Prize*. http://www2.research.att.com/~volinsky/netflix/Bellkor2008.pdf.

Berners-Lee, Tim, 2006. *Linked Data - Design Issues*. http://www.w3.org/DesignIssues/LinkedData.html.

Berners-Lee, Tim, James Hendler, and Ora Lassila, 2001. *The Semantic Web. The Scientific American*. http://www.scientificamerican.com/article.cfm?id=the-semantic-web.

Bizer, Christian, Tom Heath, and Tim Berners-Lee, 2009. *Linked Data - The Story So Far*. http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf.

Bleier, Armin, Benjamnin Zapilko, Mark Thamm, and Peter Mutschke, 2011. *Using SKOS to Integrate Social Networking Sites with Scholary Information Portals*. http://ceur-ws.org/Vol-830/sdow2011_paper_4.pdf.

Boehm, Sebastian and Marko Luther, 2009. *FOAF on Air: Context-aware User Profiles for the Social Web*. http://ceur-ws.org/Vol-520/paper11.pdf.

Chen, Zhimin, Yi Jiang, and Yao Zaho, 2010. *A Collaborative Filtering Recommendation Algorithm Based on User Interest Change and Trust Evaluation*. http://www.aicit.org/jdcta/ppl/13_JDCTAS19-557001.pdf.

Choudhury, Smitashree and John G. Breslin, 2010. *Extracting Semantic Entities and Events from Sports Tweets*. http://ceur-ws.org/Vol-718/paper_17.pdf.

Choudhury, Smitashree and John G. Breslin, 2011. *Extracting Semantic Entities and Events from Sports Tweets.* http://ceur-ws.org/Vol-718/paper_17.pdf.

DeVoch, Laurens, Selver Softic, and Martin Ebner, 2011. *Semantically driven Social Data Aggregation Interfaces for Research 2.0.* http://www.scribd.com/mebner007/d/65189313-Semantically-driven-Social-Data-Aggregation-Interfaces-for-Research-2-0.

Drachsler, Hendrik, Toine Bogers, Riina Vuorikari, Katrien Verbert, Erik Duval, Nikos Manouselis, Günther Beham, Stefanie Lindstaed, Hermann Stern, Friedrich Martin, and Martin Wolpers, 2010. *Issues and Considerations regarding Shareable Data Sets for Recommender Systems in Technology Enhanced Learning.* http://www.sciencedirect.com/science/article/pii/S1877050910003236.

Ebner, Martin, 2010. *Is Twitter a Tool for Mass-Education?* http://www.scribd.com/doc/72687166/Is-Twitter-a-Tool-for-Mass-Education.

Eibe, Frank and Remco R. Boukaert, 2004. *Naive Bayes for Text Classification with Unbalanced Classes.* http://www.cs.waikato.ac.nz/~eibe/pubs/FrankAndBouckaertPKDD06new.pdf.

Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth, 1996. *From Data Mining to Knowledge Discovery in Databases.* http://www.kdnuggets.com/gpspubs/aimag-kdd-overview-1996-Fayyad.pdf.

Fung, Glenn, 2001. *A Comprehensive Overview of Basic Clustering Algorithms.* http://patwa.googlecode.com/svn/trunk/docs/ref/clustering.pdf.

Gemmel, Jonathan, Thomas Schimoler, Maryam Ramezani, and Bamshad Mobasher, 2009. *Adapting K-Nearest Neighbor for Tag Recommendation in Folksonomies.* http://www.dcs.warwick.ac.uk/~ssanand/itwp09/papers/gemmell.pdf.

Ghahramani, Zoubin, 2004. *Unsupervised Learning.* http://mlg.eng.cam.ac.uk/zoubin/papers/ul.pdf.

Gimple, Kevin, Nathan Schneider, O'Connor Brendan, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogamata, Jeffrey Flanigan, and Noah A. Smith, 2011. *Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments.* http://www.ark.cs.cmu.edu/TweetNLP/gimpel+etal.acl11.pdf.

Goldwater, Sharon and Thomas L. Griffiths, 2005. *A Fully Bayesian Approach to Unsupervised Part-of-Speech Tagging.* http://cocosci.berkeley.edu/tom/papers/bhmm.pdf.

Hahn, Koo, 2012. *Baum-Welch algorithm.* http://www.sjsu.edu/faculty/hahn.koo/teaching/ling124/slides/ling124_forward_backward.pdf.

Hannon, John, Mike Bennett, and Barry Smyth, 2010. *Recommending Twitter Users to Follow Using Content and Collaborative Filtering Approaches.* http://irserver.ucd.ie/dspace/bitstream/10197/2524/1/john-hannon-recsys-v4-April-25.pdf.

Horn, Christopher, Elisabeth Lex, and Michael Granitzer, 2011. *WHO TWEETS: DETECTING USER TYPES AND TWEET QUALITY USING SUPERVISED CLASSIFICATION*. http://know-center.tugraz.at/download_extern/papers/iadis2011-whotweets-chorn.pdf.

Hsu, Chih-Wei, Chang Chih-Chung, and Chih-Jen Lin, 2010. *A Practical Guide to Support Vector Classification*. http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

Jannach, Dietmar, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich, 2011. *Recommender Systems*. Cambridge. ISBN 978-0521493369.

Jansen, Bernard, Mimi Zhang, Kate Sobel, and Abdur Chowdury, 2009. *Twitter Power: Tweets as Electronic Word of Mouth. Journal of the American Society for Information Science and Technology*.

Juang, B. H. and L. H. Rabiner, 2003. *An Introduction to Hidden Markov Models*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1165342.

Kraker, Peter, Claudia Wagner, Fleur Jeanquartier, and Stephanie Lindstaed, 2010. *On the Way to a Science Intelligence: Visualizing TEL Tweets for Trend Detection*. http://know-center.tugraz.at/download_extern/papers/science_intelligence.pdf.

Kudo, Taku and Yuji Matsumoto, 2001. *Chunking with Support Vector Machines*. http://acl.ldc.upenn.edu/N/N01/N01-1025.pdf.

Leger, Alain, Johannes Heinecke, Lyndon Nixon, Pavel Shivaiko, Jean Charlet, Paola Hobson, and Francois Gosadoue, 2006. *The Semantic Web from an Industry Perspektive*. http://www.technologyreview.com/video/semantic.

Linden, Greg, Brent Smith, and Jeremy York, 2003. *Amazon.com Recommendations*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1167344&userType=&tag=1.

Markoff, John, 2006. *Entrepreneurs See a Web Guided by Common Sense. The New York Times*. http://www.nytimes.com/2006/11/12/business/12web.html.

Melamed, Dan, Ryan Green, and Joseph P. Turian, 2001. *Precision and Recall of Machine Translation*. http://acl.ldc.upenn.edu/N/N03/N03-2021.pdf.

Mika, Peter and David Laniado, 2010. *Making sense of Twitter*. http://iswc2010.semanticweb.org/pdf/352.pdf.

Milikic, Nikola, Jelena Jovanovic, and Milan Stankovic, 2011. *Discovering the Dynamics of Terms' in Semantic Relatedness through Twitter*.

Muehlberger, Herbert, Martin Ebner, and Behnam Taraghi, 2010. *Try out Grabeeter to Export, Archive and Search Your Tweets*. http://www.scribd.com/doc/40231851/twitter-Try-out-Grabeeter-to-Export-Archive-and-Search-Your-Tweets.

Mödritscher, Felix, 2010. *Towards a Recommender Strategy for Personal Learning Environments*. http://www.sciencedirect.com/science/article/pii/S1877050910003157.

Nakagawa, Tetsuji, Taku Kudoh, and Yuji Matsumoto, 2001. *Unknown Word Guessing and Part-of-Speech Tagging Using Support Vector Machines*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.2808.

Olsen, David L. and Dursun Delen, 2008. *Advanced Data Mining Techniques*. Springer. ISBN 978-3540769163.

Olson, Clark F., 1996. *Parallel Algorithms for Hierarchical Clustering*. http://faculty.washington.edu/cfolson/papers/pdf/pc95.pdf.

O'Reilly, Tim, 2005. *What is Web 2.0*. http://oreilly.com/web2/archive/what-is-web-20.html.

Perkins, Jacob, 2010. *Python Text Processing with NLTK 2.0 Cookbook*. PACKT publishing. ISBN 978-1849513609.

Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Berstrom, and John Riedl, 1994. *GroupLens: an open architecture for collaborative filtering of netnews*. http://dl.acm.org/citation.cfm?id=192905.

Romero, Cristobal and Sebastian Ventura, 2010. *Educational Data Mining: State of the Art*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5524021.

Russel, Matthew A., 2011. *Mining the Social Web*. O'Reilly. ISBN 978-1449388348.

Seth, Aaditeshwar, Jie Zhang, and Robin Cohen, 2008. *A Subjective Credibility Model for Participatory Media*. http://www.cse.iitd.ac.in/~aseth/credibilityv7.pdf.

Softic, Selver, Martin Ebner, Herbert Muehlburger, Thomas Altmann, and Behnam Taraghi, 2010. *@twitter Mining Microblogs Using Semantic Technologies*. http://iswc2010.semanticweb.org/pdf/352.pdf.

Solskinnsbakk, Geir and Jon Atle Gular, 2011. *Semantic Annotation from Social Data*. http://ceur-ws.org/Vol-830/sdow2011_paper_3.pdf.

Stankovic, Milan, Claudia Wagner, Jelena Jovanovic, and Philippe Laubert, 2010. *Looking for Experts? What can Linked Data do for You?* http://events.linkeddata.org/ldow2010/papers/ldow2010_paper19.pdf#.

Templeton, Mike, 2008. *Micorblogging defined*. http://microblink.com/2008/11/11/microbloggingdefined/.

Terry, Douglas B., 1992. *Replication in an Information Filtering System*. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=242615&tag=1.

Thai-Nghe, Nguyen, Lucas Drumond, Artus Krohn-Grimberghe, and Lars Schmidt-Thieme, 2010. *Recommender System for Predicting Student Performance*. http://www.ismll.uni-hildesheim.de/pub/pdfs/Nguyen_et_al_RecSysTEL2010.pdf.

Tjong Kim Sang, Erik F. and Sabine Buchholz, 2000. *Introduction to the CoNLL-2000 Shared Task: Chunking*. http://acl.ldc.upenn.edu/W/W00/W00-0726.pdf.

Villata, Serena, Nicolas Delaforge, Fabien Gandon, and Amelie Gyard, 2011. *Social Semantik Web Access Controll*. http://ceur-ws.org/Vol-830/sdow2011_paper_5.pdf.

Wagner, Claudia, 2010. *Exploring the Wisdom of Tweets: Towards Knowledge Acquisition from Social Awareness Streams*. http://www.joanneum.at/uploads/tx_publicationlibrary/WAC_2010_ESWC.pdf.

Wagner, Claudio and Markus Strohmaier, 2010. *The Wisdom in Tweetonomies: Acquiring Latent Conceptual Structures from Social Awareness Streams*. http://kmi.tugraz.at/staff/markus/documents/2010_SemSearch2010_Tweetonomies.pdf.

Witten, Ian h., Frank Elbe, and Mark A. Hall, 2011. *Practical Machine Learning Tools and Techniques*. Morgan Kaufmann. ISBN 978-0123748560.