Masterarbeit

Design and Integration of an adaptive Information Retrieval System for MS SharePoint

Julia Gebetsberger

Institut für Technische Informatik
Technische Universität Graz

Begutachter: Dipl.-Ing. Dr. techn. Christian Josef Kreiner
Betreuer: Univ.-Ass. Dipl.-Ing. Bakk.techn. Philipp Maria Glatz

Graz, im September 2011

Kurzfassung

Das Produkt *SharePoint Customer Service Desk* der Firma *Solvion information management GmbH & Co KG* dient zur Unterstützung von Service-Mitarbeitern. Das Auffinden von Lösungen von einem bestehenden System erweist sich jedoch als schwierig und ineffizient. Um dieses Problem zu lösen wird ein System benötigt, das automatisch Lösungsvorschläge für neu eintreffende Anfragen ermittelt. Ebenfalls stellt das Zuweisen von Anfragen zu Service-Mitarbeitern ein Problem dar, da dies diese selber oder eine dritte Person erledigen muss. Dabei gibt es keine Unterstützung oder Information darüber, welche Problemstellungen für einen Service-Mitarbeiter geeignet sind. Das System sollte von der immer wachsenden Wissensbasis profitieren und somit adaptiv sein.

Bekannte Methoden zum Lösen dieser Aufgabenstellung sind Information Retrieval Systeme und Klassifizierungsalgorithmen. Information Retrieval Systeme ermöglichen es aus einer großen Datenmenge bestehende Information zu extrahieren. Ein Qualitätsmaß dieser Systeme sind Parameter wie Prediction und Recall. Klassifizierungsalgorithmen stammen aus dem Bereich der künstlichen Intellegenz und machen es möglich Kategorien bzw. Klassen zu bestimmten Objekten zuzuweisen.

Die Aufgabe meiner Masterarbeit besteht darin ein System zu verwirklichen, dass die oben beschriebenen Probleme löst. Dazu wurde eine Literaturrecherche vollzogen und zwei Implementationen durchgeführt. Eine in Matlab, um die diversen Algorithmen zu evaluieren und schließlich eine für Microsoft SharePoint 2010, die in den bestehenden *SharePoint Customer Service Desk* integriert werden kann.

In meiner Arbeit findet man einen guten Überblick über Information Retrieval Systeme und diverse Klassifizierungsalgorithmen, die sich zum Verarbeiten von Texten eignen. Ebenso findet man eine kurze Beschreibung zu Mircorosoft SharePoint und dessen Suchalgorithmus. Ein Schwerpunkt dieser Arbeit bezieht sich auf vektorbasierende und probabilistische Modelle. Außerdem wird eine fertige funktionierende Lösung gezeigt, die sehr generell ist und somit für verschiedene Einsatzbereichen verwendet werden kann.

Abstract

The company *Solvion information management GmbH & Co KG* developed a product called *SharePoint Customer Service Desk* . The problem exists that the finding of existing solutions in an existing system is difficult and inefficient. The problem requests should be assigned to a specific support assistant. Currently this must be done by the support assistant itself or by a third person. These two different problems should be solved by a new system, which should be able to aid the support assistants by searching existing solutions of problem requests and to assign these requests to specific assistants. The performance of the system should enhance by the growing knowledge base.

Information Retrieval Systems and classification algorithms are convenient methods to solve such problems. Classification algorithms are from the area of artificial intelligence and make it possible to assign classes or categories to specific objects. Information Retrieval Systems are used to extract information from a huge amount of data. Parameters like precision and recall are used to evaluate such systems.

My exercise is to develop a system that solves the above described problems. Therefore algorithms that are appropriate are searched, described and evaluated. A Matlab implementation is developed for that evaluation and finally the system is implemented for Microsoft SharePoint 2010 and integrated within the *SharePoint Customer Service Desk*.

This master thesis gives you a good overview of Information Retrieval Systems and different classification algorithms, that are appropriate for text processing. Probabilistic and vector based models are focused. Microsoft SharePoint and its search service are also mentioned. The implemented systems are described and the final system shows a very generic realisation, which can be used for different areas.

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.

....
date (signature)

Danksagung

Diese Diplomarbeit wurde im Studienjahr 2010/11 am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt.

Die Arbeit ist mit der Kooperation der Firma *Solvion information management GmbH & Co KG* enstanden und ich möchte mich hiermit bei DI(FH) Stefan Schnuderl für seine Idee und Unterstützung bedanken. Ebenfalls gilt der Dank meinen Betreuern Dipl.-Ing. Bakk.techn. Philipp Maria Glatz und Dipl.-Ing. Dr.techn. Christian Josef Kreiner.

Besonders bedanken möchte ich mich bei meiner ganzen Familie insbesonders bei meinen Eltern Andreas und Mag. Andrea Gebetsberger, die mir das Studium ermöglichten und mich finianziell und seelisch dabei unterstützten. Weiters möchte ich mich besonders bei meiner Schwester Mag. Jennifer Gebetsberger für die Unterstüzung bei der Korrektur und Tips bei der Masterarbeit bedanken. Ebenfalls gilt der Dank meinem Freund Werner Arnus für das Verständnis und die Unterstützung während des Studiums.

Widmen möchte ich die Arbeit Maria Arnus, die Großmutter meines Freundes, die leider während dem Entstehen dieser Masterarbeit verstorben ist.

Graz, im Monat Jahr Name des Diplomanden

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 History of Information Retrieval Systems

Since around the year 3000BC humans started to write down information. With the help of the first printing press of Gutenberg in the year 1450AD and later with the first invention of the computer, the amount of information increased vigorously. In 1945 the idea of information retrieval was born and till 1950 methods to enable text searches were described. 1957 H.P. Luhn showed the idea of using words for indexing documents. To retrieve documents the overlap was calculated. In 1960 Gerard Salton described the idea of the SMART (System for the Mechanical Analysis and Retrieval of Text) system. This system defines different weighting schemes. Also the Cranfield paradigm to evaluate information retrieval systems was defined by Cyril Cleverdon. Between 1970 and 1980 different developers implemented various models for document retrieval based on the knowledge of 1960 [Sin01]. In the year 1990 and the inventing of the world wide web information retrieval systems became very important[Got09]. Parallel to these search engines, recommender systems got developed, which were using information about the users to improve the results. [TH01] To be able to evaluate all these different realisationsm, the text retrieval Conference TREC was established in 1992 [Sin01].

1.2 Motivation of this Work

The product *SharePoint Customer Service Desk* by company *Solvion information management GmbH & Co KG* helps support assistants to administrate customer requests. Customers are able to communicate through different communication channels, like the telephone or e-mail, with the support assistant. For each new incoming problem a *ticket* is created. The communication between the assigned support assistant and the caller is saved via actions, so the history for every problem is stored and can help to solve new incoming problems, which are similar to the past ones. A further part of the *SharePoint Customer Service Desk* is to enable an incident manager to verify the efficiency of the support assistance by means of expressive statistics.

The information for solving requests is growing with each enquiry. At the beginning this is a great advantage for each support assistant, but later this information overload possibly leads to time lacks. Reviewing different requests to find an appropriate solution

Figure 1.1: SharePoint Customer Service Desk Overview

wastes a lot of time. Also for new support assistants it is very hard to get along with such an overloaded and complex system. An intelligent system should be developed to solve this problem, which is able to find solutions automatically and which assists the support agents.

1.3 Challenges of an Information Retrieval System

All problems can be described with the help of their tickets and the corresponding actions, which have a title and body field, containing textual information. Based on that, it should be possible to classify problems and to retrieve potential solutions. The system should have following characteristics:

Adaptiveness: the quality of the system should enhance over time

Efficiency: the solutions and the classification should compute fast

Autonomy: the users should not have to set much configurations

General applicability: the system should be as generic as possible to allow it also for other applications and machine learning parts of the system should not tend to overfitting

1.4 Goals of the Thesis

The goal of this master thesis is to develop a system to enhance the functionality and usability of the product *SharePoint Customer Service Desk* of *Solvion information management GmbH & Co KG*. By the implementation of the system the characteristics described above should be considered. Following goals should be achieved:

Classification: A request should automatically be categorised. This should help the support assistants to find easier solutions for a problem and reduce the complexity of the system. With the help of this mechanism also support assistants can be assigned automatically to tickets.

Recommendation: Each request adds proposals for appropriate solutions. These could then be used to recommend solutions for following requests.

Feedback Function: The retrieval quality of the system can enhance by applying a feedback function.

Microsoft SharePoint Integration: The system should be integrated completely within Microsoft SharePoint 2010.

1.5 Outline

In the next chapter *related work*, literature which was studied to build the system, is described. This includes a description of recommender and information retrieval systems. Furthermore, algorithms to perform a classification and suggestion for text based items are shown. Some popular information retrieval systems, as well as methods for the evaluation of them, are described.

The successive chapter contains the *evaluation* of the different algorithms, which were described. This includes the quality of the returned classes accordingly to the suggestion and the run time behaviours of these algorithms. The design of the evaluation system and the data, which have been used, are described.

In the chapter *Implementation* the chosen algorithms and the design of our system are shown. Also rough comparisons to existing information retrieval systems, which are described in the related work, are made.

Finally this thesis includes a *conclusion*, that contains a discussion of completeness and possible future work.

Chapter 2

Related Work

Nowadays systems, like information retrieval or recommendation systems, are used to find informations. These systems are able to filter, search or suggest information. The difference between recommender and information retrieval systems is, that recommender systems are using user specific information [Bur02]. The experience of other users could also be used to determine recommendations [TH01].

In this chapter, recommender systems are described at first. After that information retrieval systems and the different models, that are used to realise them, are shown. The inverted index is explained, which is used to enhance performance. Implicit and explicit feedback mechanisms are described, that give the user the chance to influence the behaviour of the systems. Classification algorithms for documents are explained and evaluation methods are shown.

2.1 Recommender Systems

The independent research of recommender systems arose in the mid-1990s [AT05].

Items can be recommended by splitting them into two categories. One category defines the users like, and the other one, that he or she does not like the item. [PB07].

A very popular recommender system is the book recommendation system of Amazon. Favourite categories are retrieved by the purchasing behaviour of the user [PB07]. Recommendations are also used within Amazon to personalise the web site to interests of the users. The algorithms, which have been developed, can handle a large volume of user data [LSY03]. In the paper of [RV97] a lot of other recommender systems give an overview of the field of applications.

Different types of recommender systems are known [TH01, AT05, Bur02]:

- **Content-based systems:** recommend items because of the user history.
- **Recommendation support systems**: helps people to share recommendations.
- **Social data mining system:** mining information from computational records of social activity.
- **Collaborative filtering systems:** use the information of the preferences of other users to recommend items to a specific one.

- **Demographic-based systems:** Demographic information is used for the recommendation.
- **Utility-based systems:** The features of an item are used to suggest other ones
- **Knowledge based systems:** With the help of a description of the user needs, items are recommended.
- **Hybrid systems:** are combination of different recommendation systems, to profit from the advantages of each of them.

Items are described with the help of vectors. A matrix which is created out of these is called vector space model. Each entry contains a feedback value, which can be binary, numeric or nominal [MS06]. The item vectors can include calculated term weights or also feedback information. With the help of an utility function, which is based on the comparison of the document vectors, a value, that represents the similarity, called relevance ranking, can be computed. Sorted by this ranking a result list of recommended items can be established [AT05].

Machine learning techniques, like clustering, decision trees and neuronal networks are also an alternative [AT05, Bur02].

For binary recommender systems the goal is to classify items with the help of a function, based on the feature vectors, to determine the utility for a user [MS06].

2.1.1 Collaboration-based Recommender System

A collaboration-based recommendation system determines, if an item should be recommended or not for a specific user, with the help of the information about the item ratings of all other users. For this calculation users with a similar rating scheme are obtained. Based on their ratings for a particular item, the recommendation for an other user can be estimated. These ratings can be binary or not. Binary means that an item is suggested or not, otherwise a ranking value, which indicates the relevance, is calculated. The problems of such kind of recommendation systems are following [Bur02]:

New user problem: A new user has not rated items and no other users, which have a similar rating scheme, can be obtained.

New item problem: New items are not recommended because they never have been rated before.

Sparsity: In recommendation system the rating information of each item is used. Users in the most cases can hardly rate all existing items and so for a lot of items no information is available. This case is called sparsity.

2.1.2 Content-based Recommender System

Content-based recommender systems use the history, for example the preferred items from the past of a user, to suggest new items. Information retrieval systems are often very similar to this kind of recommendation. Collected user information is used to improve the results [AT05].

2.2 Information Retrieval System

Information retrieval systems are used, as the name says, to retrieve information out of a large collection of data. These data can be available in the web or be a controlled document collection [SL98].

With the help of a user defined query, comparisons between all existing items can be made in order to retrieve similar ones. A drawback of that is, that the user has to know what he or she is interested in, to be able to build an appropriate query. This means that he or she has to know appropriate keywords, which occur in the documents, designated to obtain. Query expansions, that means to adding automatically keywords to a query, could help to obtain a better recall. That keywords can be generated with the help of thesauri, domain models or feedback informations [GLWW00, Got09].

In Figure 2.1 the important parts of information retrieval systems are shown. For a defined query of the user, relevant documents have to be retrieved. Therefore a given document collection has to be processed and represented to be able to calculate matching between the different documents. To improve the information retrieval process, the user has the possibility to evaluate the obtained result with the help of a feedback function [Got09].

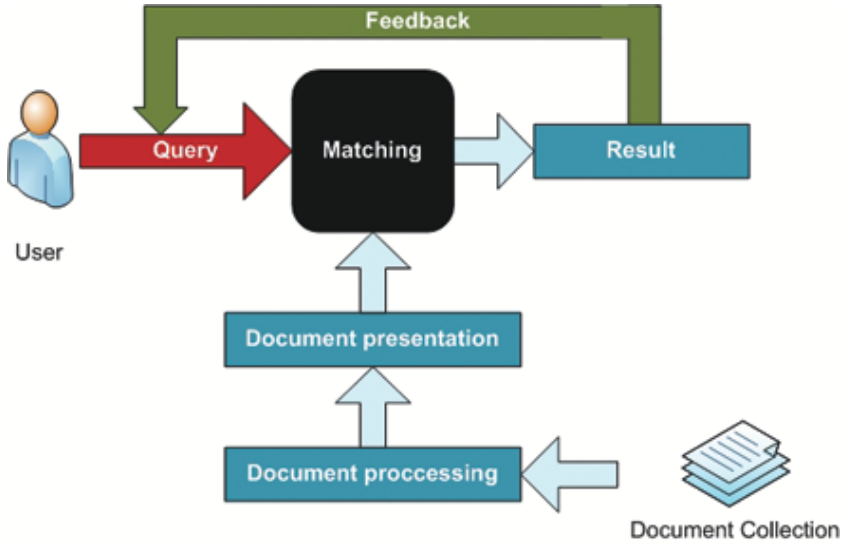

Figure 2.1: Components of an Information Retrieval System [Got09]

2.2.1 Document Collection Characteristics

Document collections are called corpus. With the help of Heap's and Zipf's law fundamental characteristics of document collections can be estimated. Heap's law says that the vocabulary size M of a specific corpus grows as shown in Equation 2.1.

$$M \approx k * T^b \tag{2.1}$$

T is the amount of tokens in the whole corpus. Tokens are the words, which are extracted of the documents. Numbers, names and so on are often filtered before. The parameters k

and b are two constants, which have to be chosen and which are dependent of the corpus. b is often set approximately to 0,5 and k is a value between 30 and 100. Zipf's law describes the occurrence of the terms t_i in the whole corpus. Terms are words, that represent the vocabulary. Therefore the terms have to be sorted by their frequency in the corpus cf. The corpus frequency is the count of documents in which the term t_i occurs. So t_1, for $i = 1$, is the term with the highest corpus frequency and t_2 has the second highest cf and so on. The Equation 2.2 shows Zipf's law.

$$cf(t_i) * i \approx const \tag{2.2}$$

These two laws imply, that the more documents are added, the less new terms are appended to the vocabulary. Furthermore they show that terms, which occur very often or infrequently, have less information value for a document. In Figure 2.2 two bound cf_{min} and cf_{max} are shown. Words occurring very often are called stopwords and can be removed in most cases. Infrequent words are mostly not deleted for giving the user the opportunity to search items with the help of these unique ones. Stopwords are below the cf_{min} bound. This bound can be defined via a specific list of words, which should be removed, or via a given threshold [Got09].

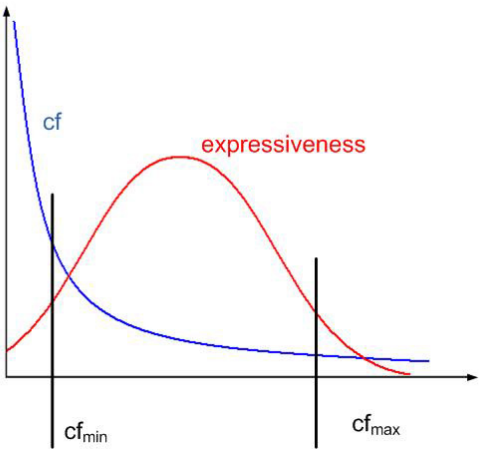

Figure 2.2: Diagram of Zipf's Law, the blue line shows the frequency of a specific word in the corpus and the red line the expressiveness of it

2.2.2 Vector Space Model

Documents can be represented as vectors. For each entry weights are calculated, which are indicating the importance of each term in a given document. So a $m \times n$ matrix is created, where m is the vocabulary size and n the amount of items (see Figure 2.3).

These are for example the TF–IDF weights. TF–IDF stands for *Term Frequency–Inverse Document Frequency*. A retrieval function calculates the similarity between the documents via their vectors. The higher the similarity the higher is the relevance for the given query [Got09].

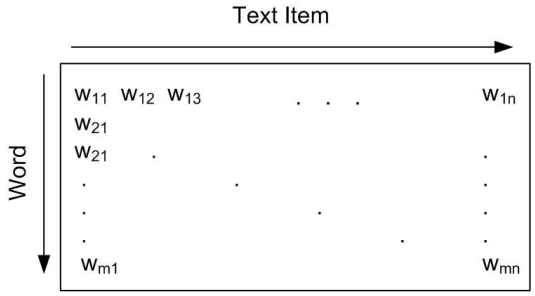

Figure 2.3: Vector Space Model, for each word and text item a weight is set [Got09]

Weightings

To build a vector space model different weighting schemes can be used. Three different factors are relevant for its computation:

- Global weighting g_i
- Local weighting l_i
- Normalisation factor n_i

The global weight indicates the importance of a specific term in the corpus. Local weights are representing the relevance in a specific text and the normalisation factor is used to normalise the vector lengths. The final weight w_i is computed with the help of a multiplication of these three factors (see Equation 2.3) [Got09].

$$w_i = g_i * l_i * n_i \tag{2.3}$$

The so called SMART system contains a definition of possible weightings [Got09, ZG05]. A very common scheme is the TD–IDF weighting. The frequency of the occurrence is determined in a specific text and the entire collection. In Equation 2.4 the calculation of the weight for the term i is shown. tf_i stands for the number of appearances of the term in the document. n is the amount of all documents and f_i is the number of documents in which the term occurs [MS06].

$$w_i = tf_i * log(\frac{n}{df_i}) \tag{2.4}$$

If a term occurs several times in a document it is more relevant and so the weight w_i is higher. This value is called TF, what stands for term frequency. Terms, which occur in the corpus very often have a small information value for a specific document. Because of that, the weights are smaller if a word occurs often in the whole document collection. This value is the inverse document frequency, short IDF [MS06].

Different types of TF–IDF measurements exist. To weaken terms with a high term frequency following Equation 2.5 can be used. The highest term frequency of the document is used for normalisation [Got09].

$$w_{local}(t_j, d_i) = \alpha + (1 - \alpha) * \frac{tf_{d_i}(t_j)}{\max k = 1, ..., M tf_{d_i}(t_k)} \tag{2.5}$$

Similarity Measurements

To determine if two documents are similar their vectors are compared via similarity measurements. For instance the cosine similarity for two documents, which are described via their vector $\vec{vec}_1$ and $\vec{vec}_2$, can be computed (Equation 2.6).

$$cos(\vec{vec}_1, \vec{vec}_2) = \frac{\vec{vec}_1{}^T * \vec{vec}_2}{\|\vec{vec}_1\| * \|\vec{vec}_2\|} \tag{2.6}$$

A great advantage of the cosine similarity is its independence of the vector length. This means that the vector space model must not be normalised. In Figure 2.4 an example for the cosine similarity for two dimensions is shown. Two dimensions mean that in the document collection only two words occur, $word_1$ and $word_2$. On the x–axis the weight for the $word_1$ is assigned and on the y–axis for the $word_2$. The $\vec{vec}_1$ shown in Figure 2.4 indicates that the weight for the $word_1$ is larger as for $word_2$. For instance that can be if the $word_1$ occurs more often in a specific document. The word weights for the $\vec{vec}_2$ are vice versa. The cosine value can be between minus one and plus one. The number one means that the angle between the two vectors is null.

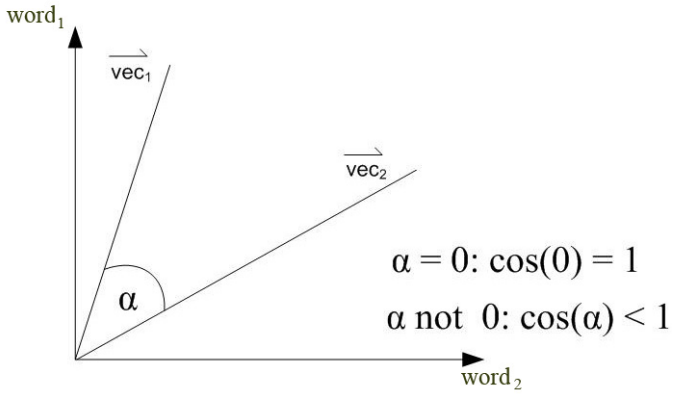

Figure 2.4: Cosine Similarity Example

Further similarity equations for the document vectors d_i with the words w_i can be found in Table 2.1 [Got09].

Pseudo-Cosine Similarity	$sim_{pseudo}(\vec{d_i}, \vec{d_k}) = \frac{\sum_{j=1}^{M} w_i^{[j]} * w_k^{[j]}}{\sum_{j=1}^{M} w_i^{[j]} * \sum_{j=1}^{M} w_k^{[j]}}$
Dice Similarity	$sim_{dice}(\vec{d_i}, \vec{d_k}) = \frac{2 * \sum_{j=1}^{M} w_i^{[j]} * w_k^{[j]}}{\sum_{j=1}^{M} w_i^{[j]} + \sum_{j=1}^{M} w_k^{[j]}}$
Jaccard Similarity	$sim_{Jaccard}(\vec{d_i}, \vec{d_k}) = \frac{\sum_{j=1}^{M} w_i^{[j]} * w_k^{[j]}}{\sum_{j=1}^{M} w_i^{[j]} + \sum_{j=1}^{M} w_k^{[j]} - \sum_{j=1}^{M} w_i^{[j]} * w_k^{[j]}}$
Overlap Similarity	$sim_{Overlap}(\vec{d_i}, \vec{d_k}) = \frac{\sum_{j=1}^{M} min(w_i^{[j]}, w_k^{[j]})}{min(\sum_{j=1}^{M} w_i^{[j]}, \sum_{j=1}^{M} w_k^{[j]})}$

Table 2.1: Different Similarity Measurements [Got09]

2.2.3 Boolean Information Retrieval

Boolean Information Retrieval systems are using a vector space model. For each term of the vocabulary V an entry exists that indicates if a specific term exists in the document or not. V is the set of all terms of the document collection. The user has to define a query with a set of keywords. Also boolean operations like OR, AND, etc. can be added to the query to define for instance if a keyword must be contained in the searched document or not. The result of a query is a sub set of the documents of a certain document collection, which fits the given query. The retrieval function looks like $\rho : D \times Q \Rightarrow \{0, 1\}$. D is the document collection and Q the set of all queries. The retrieval function means that a document fulfills the query or not [Got09].

2.2.4 Probabilistic Information Retrieval

Probabilistic information retrieval calculates a probability for each document, which indicates the relevance. In this section the binary independence model is described more precisely to understand the functionality of such kind of systems [Got09].

Binary Independence Model

The probabilistic information retrieval only considers the terms, which describe the query and the document for the information retrieval (same as boolean information retrieval). These ones have directly an importance for the obtained result. The result of such an information retrieval system is also only the information if a document is relevant or not. It is also assumed that the terms are independent to each other. The probability $P(R = 1|\vec{d}, \vec{q})$ (Equation 2.7), if a document $\vec{d}$ for the given query $\vec{q}$ is relevant, R=1 or not R=0, has to be determined. The document and query are represented as vectors.

$$P(R = 1|\vec{d}, \vec{q}) = \frac{P(\vec{d}|R = 1, \vec{q}) * P(R = 1|\vec{q})}{P(\vec{d}|\vec{q})} \tag{2.7}$$

The result is a sorted list, where the most relevant document with the highest $P(R = 1|\vec{d}, \vec{q})$ is on top. To consider also the information of the irrelevant documents the so called *chance* O can be used. The chance of an event A can be written as Equation 2.8. $P(A)$ is the probability that the event A occurs and $P(\bar{A})$, the probability that this event not occurs. The chance has the advantage that some computations become irrelevant and so it is easier to compute. Also it can be used for the ranking because if $P(A_1) > P(A_2)$ also $O(A_1) > O(A_1)$.

$$O(A) = \frac{P(A)}{P(\bar{A})} \tag{2.8}$$

This value also can be used to rank the documents via their relevance. Equation 2.9 shows the calculation of the chance for the relevance of the document.

$$O(R = 1|\vec{d}, \vec{q}) = \frac{P(R = 1|\vec{d}, \vec{q})}{P(R = 0|\vec{d}, \vec{q})} = \frac{P(\vec{d}|R = 1, \vec{q}) * P(R = 1|\vec{q})}{P(\vec{d}|R = 0, \vec{q}) * P(R = 0|\vec{q})} \tag{2.9}$$

The second part of the equation is the chance $O(R = 1|\vec{q})$, which is not important for the ranking, because it is a constant.

$$O(R = 1|\vec{q}) = \frac{P(R = 1|\vec{q})}{P(R = 0|\vec{q})} \tag{2.10}$$

The probability $P(\vec{d}|R = 1, \vec{q})$, shown in Equation 2.11, can be written as multiplication of the conditional probabilities $P(t_i|R = 1, \vec{q})$ of terms t_i, which occur in the document d.

$$P(\vec{d}|R = 1, \vec{q}) = P(t_1|R = 1, \vec{q}) * ... * P(t_k|R = 1, \vec{q}) \tag{2.11}$$

To consider also the information about all terms of the vocabulary, which are not occurring in the document, the probability $P(\vec{d}|R = 1, \vec{q})$ can be written as Equation 2.12 and also as Equation 2.13. $P(\vec{d}|R = 0, \vec{q})$ is shown in Equation 2.14. p_{ti} is the conditional probability for a term t_i, if the document is relevant for the given query $\vec{q}$. u_{ti} is the conditional probability, if the document is not relevant for the given query.

$$P(\vec{d}|R = 1, \vec{q}) = \prod_{i \epsilon T(d)} P(t_i = 1|R = 1, \vec{q}) * \prod_{i \notin T(d)} P(t_i = 0|R = 1, \vec{q}) \tag{2.12}$$

$$P(\vec{d}|R = 1, \vec{q}) = \prod_{i \epsilon T(d)} p_{ti} * \prod_{i \notin T(d)} (1 - p_{ti})) \tag{2.13}$$

$$P(\vec{d}|R = 0, \vec{q}) = \prod_{i \epsilon T(d)} u_{ti} * \prod_{i \notin T(d)} (1 - u_{ti}) \tag{2.14}$$

So Equation 2.10 can be written as Equation 2.15.

$$O(R = 1|\vec{q}) = \frac{\prod_{i \epsilon T(d)} p_{ti} * \prod_{i \notin T(d)} (1 - p_{ti})}{\prod_{i \epsilon T(d)} u_{ti} * \prod_{i \notin T(d)} (1 - u_{ti})} * O(R = 1|\vec{q}) \tag{2.15}$$

In Equation 2.15 every word of the document is considered ($i \epsilon T(d)$). This is not required because it can be assumed that the probability for terms, which do not occur in the query is the same. If a term does not occur and if it is relevant or not, the probabilities p_{ti} and u_{ti} are equal. The new Equation 2.16 considers that.

$$O(R = 1|\vec{q}) = \prod_{i \epsilon T(q) \bigcap T(d)} \frac{p_{ti}}{u_{ti}} * \prod_{i \epsilon T(q)\ T(d)} \frac{(1 - p_{ti})}{(1 - u_{ti})} * O(R = 1|\vec{q}) \tag{2.16}$$

With the help of Equation 2.17 the chance can be written as Equation 2.18.

$$\prod_{i \epsilon T(q)\ T(d)} \frac{(1 - p_{ti})}{(1 - u_{ti})} = \prod_{i \epsilon T(q))} \frac{(1 - p_{ti})}{(1 - u_{ti})} * \prod_{i \epsilon T(q) \bigcap T(d)} \frac{(1 - u_{ti})}{(1 - p_{ti})} \tag{2.17}$$

$$O(R = 1|\vec{q}) = \prod_{i \epsilon T(q) \bigcap T(d)} \frac{p_{ti} * (1 - u_{ti})}{u_{ti} * 1 - p_{ti})} * \prod_{i \epsilon T(q))} \frac{(1 - p_{ti})}{(1 - u_{ti})} * O(R = 1|\vec{q}) \tag{2.18}$$

The second product in Equation 2.18 and $O(R = 1|\vec{q})$ can be ignored in the retrieval function, because they are constant. To make the computation more efficient the multiplication could also be written as summation of logarithms. The logarithm has no impact on the ranking. The retrieval function is shown in Equation 2.19.

$$\rho(d,q) = \sum_{i \epsilon T(q) \cap T(d)} log(\frac{p_{ti} * (1 - u_{ti})}{u_{ti} * (1 - p_{ti})}) = \sum_{i \epsilon T(q) \cap T(d)} w(t,d) \tag{2.19}$$

$w(t,d)$ can be interpreted as the weighting for a term t in a document d. It is assumed that the collection of relevant documents for a query is known and so the probabilities p_t and u_t can be estimated, see Equation 2.20 and 2.21.

$$\hat{p_{ti}} = \frac{s_i}{S} \tag{2.20}$$

$$\hat{u_{ti}} = \frac{df(t_i) - s_i}{N - S} \tag{2.21}$$

s_i is the amount of relevant documents for the term t_i. S are all relevant documents. N is the number of all documents in the corpus and $df(t_i)$ is the document frequency of term t_i. The retrieval function can be written as Equation 2.22.

$$\rho(d,q) = \sum_{i \epsilon T(q) \cap T(d)} log(\frac{s_i * (N - S - df(t_i) + s_i)}{(df(t_i) - s_i) * (S - s_i)}) \tag{2.22}$$

If s_i is null, that means that the specific term is not relevant in any document, the logarithm can not be computed. Also if the term $(df(t_i) - s_i)$ is null, what happens when a term only occurs in relevant documents or if $(S - s_i)$ is null (when the term occurs in all relevant items), the divisor is null and the retrieval function is undefined. To avoid these problems a constant is added to s_i, which is called smoothing parameter. In the most applications it is set to 0.5. Equation 2.22 describes only a model because S, the amount of relevant documents for a query, can only be estimated. Therefore different probability estimations can be computed. A common variant is the BM25 model [Got09].

$$w(t,d) = log(\frac{p_t * (1 - u_t)}{u_t * (1 - p_t)}) = log(\frac{p_t}{1 - p_t}) + log(\frac{1 - u_t}{u_t}) \tag{2.23}$$

To estimate $w(t,d)$, Equation 2.23 and following assumptions should be considered. The number of relevant documents is very small compared to the amount of all documents of the corpus. Considering that, the calculation of u_t can be reduced to Equation 2.24. The probability of a term, that occurs in a not relevant document, can be written as shown in Equation 2.25.

$$\hat{u_t} = \frac{df(t)}{N} \tag{2.24}$$

In the most cases the amount of documents is much larger than the number of documents, where a specific term occurs and so the document frequency $df(t)$ is very small compared to N. Because of that it is possible to neglect $df(t)$ in the numerator and the equation can be approximated, see Equation 2.25.

$$log(\frac{1 - \hat{u_t}}{\hat{u_t}}) = log(\frac{N - df(t)}{df(t)}) \approx log(\frac{N}{df(t)}) \tag{2.25}$$

This approximated probability is similar to the IDF- weight. A popular weighting scheme is the BM25 algorithm [Got09]. The BM25F is an extension of this algorithm.

Okapi BM25F

This algorithm is a newer version of the BM25 algorithm. BM25F considers the document structure, that means different fields, for instance an anchor text, body and so on to calculate a ranking value. In the paper [CZR05] the algorithm is explained for searching e-mails. Different field types are distinguished:

- Subject
- Body
- Quoted

For each field f a normalised term frequency $\bar{x}_{d,f,t}$ is calculated, see Equation 2.26.

$$\bar{x}_{d,f,t} = \frac{x_{d,f,t}}{(1 + B_f(\frac{l_{d,f}}{l_f} - 1))} \tag{2.26}$$

$x_{d,f,t}$ is the term frequency of the term t in a specific field within a document d. $l_{d,f}$ is the field length of the specific field f an l_f the average field length within the corpus. B is a specific function for the normalisation of a field type. If B_f is null the term is not normalised. If it is one, the average field length is used for normalisation. The pseudo frequency $\bar{x}_{d,t}$ is calculated as a linear combination of the different term frequencies of the various field types, see Equation 2.27.

$$\bar{x}_{d,t} = \sum_f W_f * \bar{x}_{d,f,t} \tag{2.27}$$

W_f is a weighting parameter for the respective field type. The ranking value for each document is calculated with the help of Equation 2.28.

$$BM25F(d) = \sum_{t \epsilon q \bigcap d} \frac{\bar{x}_{d,t}}{K_1 + \bar{x}_{d,t}} * w_t^{(1)} \tag{2.28}$$

The $w_t^{(1)}$ is the so called RSJ relevance parameter, which is used to reduce the IDF weight, if no relevance information for a specific term exists. Static features can be added via linear combination to the BM25F ranking value. With the help of exploration and gradient descent the parameters for different test- and trainings- sets are obtained [CZR05].

2.2.5 Inverted Index

To obtain relevant documents it is not very efficient to compare all documents in a corpus. Only documents should be compared, which have any similarity with the given query [Got09]. The index process is independent of the query. An advantage is that the documents could be processed parallel and so a very high efficiency can be achieved [GLWW00]. To build an inverted index, following steps have to be made:

At first terms are extracted from the documents and duplicates are removed. For each term a posting list is created which contains the document IDs, which including that term, see Figure 2.5. The number of these document IDs is the so called document frequency, which is used for instance for the IDF calculation.

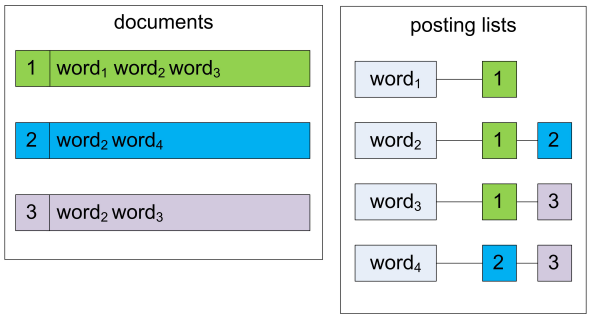

Figure 2.5: Posting Lists [Got09]

With the help of posting lists and search trees an inverted index can be created. The search trees make it possible to get the appropriate posting lists for a given query. Different operations like AND and OR can be performed to combine posting lists and to retrieve a set of documents, which is matching a given query. An AND operation will compute the difference between these lists and an OR operation will merge these ones. That means, if you want to obtain all document IDs, which are containing for instance any word of the query, an OR operation will be executed. The law from Zipf, see Equation 2.29, estimates the occurrence probability of each term. The terms are sorted descending via their term frequency in the corpus. $cf(t_i)$ is the frequency of a term t_i in the corpus and K is a constant.

$$cf(t_i) * i \approx K \tag{2.29}$$

Words which occur very often, called stopwords, and which occur very seldom are not proper terms to extract [Got09].

2.2.6 Evaluation

For the evaluation of document retrieval systems test collections are required, which are containing the possible queries and their results. The results are a set of relevant documents with a relevance weight for the given query. A very popular method to create such a test collection is the Cranfield paradigm [Got09].

Cranfield Paradigm

The results of a given query of the information retrieval systems are evaluated from a jury. Each juror appoints a relevance for each document. This relevance weighting is done by each juror individually. The opinions of these jurors differ very often. This has an effect to the absolute evaluation schema, but not to the relative one. That means if a system is evaluated as a bad or good system, it is a bad or good one. In the most cases it is not possible to evaluate all documents, so that the so called pooling method is used [Got09].

Pooling Method

Different information retrieval systems are delivering k result documents for a given query. These results are merged together to a document pool. Each of these documents are evaluated by the relevance with the help of the jurors [Got09].

Different evaluation parameters exist like precision, recall, MAP and further ones [Got09].

Precision and Recall

Precision and recall are common parameters for evaluating information retrieval applications. Equation 2.30 and 2.31 show the calculation of these parameters. $|relevant \bigcap positive|$ is the set of correct retrieved relevant items [Rij79]. *relevant* is the set of known relevant items. *positive* is the set of items, which are retrieved from a query.

$$precision = \frac{|relevant \bigcap positive|}{|positive|} \tag{2.30}$$

$$recall = \frac{|relevant \bigcap positive|}{|relevant|} \tag{2.31}$$

The relevance and precision are values between null and one. The recall indicates how many items are retrieved from the system which are relevant. All relevant items can be retrieved if all items of the system are returned, so the recall would be one. The precision is high if all obtained items are relevant. Not relevant retrieved items minimise the precision.

With the help of the so called confusion matrix (Table 2.2) the calculation of precision and recall can be written as Equation 2.32 and 2.33. TP stands for **t**rue **p**ositives, which means items occurring in the result being relevant. FN are the **f**alse **n**egative ones, which are not retrieved from the systems but are relevant [Got09].

	relevant	$\overline{relevant}$
result	TP	FP
$\overline{result}$	FN	TP

Table 2.2: Confusion Matrix

$$recall = \frac{TP}{TP + FN} \tag{2.32}$$

$$precision = \frac{TP}{TP + FP} \tag{2.33}$$

Accuracy

A further evaluation parameter is the accuracy. In Equation 2.34 the calculation of this value is shown. $|D|$ means the number of documents of the corpus. This parameter is not very suitable for information retrieval system evaluation because the number of non

relevant items is very high. Very high results can be obtained by assigning all documents as irrelevant [Got09].

$$acc = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{|D|} \tag{2.34}$$

F- Measure

Precision and Recall alone do not give a clear understandable information about the information retrieval system. Via a combination of these parameters the so called F-Measure can be calculated, see Equation 2.35.

$$F_\beta = \frac{(\beta^2 + 1) * recall * precision}{(\beta^2 * precision + recall} \tag{2.35}$$

With the help of the constant β, a weighting of the precision and recall can be applied. Commonly β is set to one (Equation 2.36).

$$F_\beta = \frac{2 * recall * precision}{precision + recall} \tag{2.36}$$

Micro- and Macroaveraging

If the precision and recall are calculated for more than one query the values have to be averaged. This can be done in two different ways. The first one, the macroaveraging, is to add the determined evaluation parameters for each query and to average them. The other method, microaveraging, is to merge all retrieved items and after that, calculate the evaluation parameters. In Equation 2.37 and 2.38 these methods are shown for the precision value [Got09].

$$p_{micro} = \frac{\sum_{i=1}^{k} |relevant_i \bigcap positive_i|}{\sum_{i=1}^{k} |positive_i|} \tag{2.37}$$

$$p_{macro} = \frac{1}{k} \sum_{i=1}^{k} p_i = \frac{1}{k} \sum_{i=1}^{k} \frac{|relevant \bigcap positive|}{|positive|} \tag{2.38}$$

Precision at k

Precision at k, short $p@k$, is used to calculate the precision for a ranked result list. That means that the first k items from this list are used to determine the precision. If the amount of relevant items is smaller than this value k, a precision of one can not be established. To avoid that, k is set to the amount of relevant items [Got09]. This calculated precision is called *R- Precision.*

Precision Recall Graph

Another possibility to see the relation between precision and recall is the precision recall graph. To get a smooth result the retrieved values are interpolated in the most cases. With the help of the eleven point recall precision graph it is possible to combine multiple of such graphs. This is done by averaging the precision values for each recall between null and one with a step size of 0.1 [Got09].

Mean Average Precision

Mean Average Precision, short MAP, represents the information of a precision recall graph with only one number. For a given set of queries Q the MAP value can be calculated. For each query a ranked result list is obtained. $|Q|$ is the number of queries. $p@k_{i,j}$ is the precision for the query i to the position j. The calculation of the MAP value is shown in Equation 2.39 [Got09].

$$MAP = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{m_i} \sum_{j=1}^{m_i} p@k_{i,j} \tag{2.39}$$

2.3 Feedback

User feedbacks are often used to describe the interest of the user. A feedback can be explicit or implicit [TH01].

Explicit Feedback: the user rates items actively

Implicit Feedback: extracted from the behaviour of the user

Explicit feedback is made from the user and depends on him or her motivation. Implicit feedback is harder to implement [MS06].

2.3.1 Rocchio Relevance Feedback

A very common method for the vector space model is the Rocchio Relevance feedback. From the result list of an initial query $\vec{q_0}$, selected relevant and non-relevant items can be used to calculate a new query vector $\vec{q_1}$. This new query will rank these previous selected relevant items higher and the non-relevant lower. It is achieved that the new generated query vector is than more similar to the relevant ones. The computation of $\vec{q_1}$ is shown in Equation 2.40. This method only adds or modifies the term weights of the original query via additions and subtractions [SB90].

$$\vec{q_1} = \alpha * \vec{q_0} + \beta * \frac{1}{|relevant|} \sum_{d_i \epsilon relevant} \vec{d_i} - \gamma * \frac{1}{|notrelevant|} \sum_{d_k \epsilon notrelevant} \vec{d_k} \tag{2.40}$$

The parameters α, β and γ are application dependent. In the most cases it should be that $\alpha > \beta > \gamma$ [Got09].

In the article [SB90] different feedback approaches have been tested and evaluated. The results of the less effective probabilistic feedback schemes are shown as not completely competitive to the vector space methods. Different Rocchio methods were evaluated. That means that the weights α, β and γ, the number of iterations and the amount of relevant and not relevant items was changed. The *"Ide dec-hi"* method was the best overall approach. This method uses only one not relevant document vector, see Equation 2.41 .

$$Q_{new} = Q_{old} + \sum_{allrelevant} \vec{d_i} - \sum_{onenonrelevant} \vec{d_k} \tag{2.41}$$

In the most cases only one iteration leads to the best results. That means, that the first calculated query vector is not modified afterwards [SB90].

In Figure 2.6 an example for a two dimension Rocchio relevance feedback is shown. If a cosine similarity would be computed in this example, $\vec{vec}_1$ would be the most similar one. The user might mark $\vec{vec}_1$ as not relevant and $\vec{vec}_3$ as relevant The new vector $\vec{Query}_{new}$, which is created with the help of the summation and subtraction, is much closer to the relevant one than the original query vector $\vec{query}$.

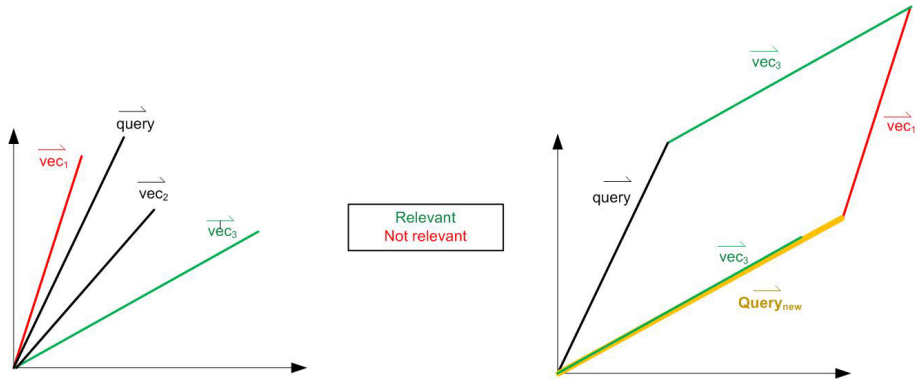

Figure 2.6: Relevance Feedback Example for two Dimensions

2.3.2 Probabilistic Relevance Feedback

Based on the selected set of relevant and irrelevant documents the probabilities p_t and u_t can be estimated, which are required for the probabilistic information retrieval. Following factors are needed to compute the probabilities:

- R the amount of relevant documents
- N the amount of not relevant documents
- $df_R(t)$ the document frequency of the term t in the set R
- $df_N(t)$ the document frequency of the term t in the set N

In Equation 2.42 the computation of p_t is shown.

$$\hat{p_t} = \frac{df_R(t)}{R} \tag{2.42}$$

In the most cases the amount of relevant documents is very small and with that also $df_R(t)$. Terms, which are not contained in the feedback, are not considered. To avoid this problem, smoothing is used, see Equation 2.43. That means that to all relevant documents a "half document" (0.5) is added.

This "half document" is added to documents in which the term exists and in which not. So all together one whole document is added and because of that, 1.0 has to be added in the dividend.

The same can also be assumed for the not relevant ones, see Equation 2.44.

$$\hat{p_t} = \frac{df_R(t) + 0.5}{R + 1} \tag{2.43}$$

$$\hat{u_t} = \frac{df_N(t) + 0.5}{N + 1} \tag{2.44}$$

A problem of these assumptions is that the not relevant documents, which were defined by the prior estimation functions, are not considered. To solve that the so called pseudo relevance feedback can be used. Therefore the top k documents of the initial query are relevant and the others are not.[Got09].

2.4 Document Classification

Document classification is a content based assignment of documents to predefined categories [GLWW00]. That classification can be single- or multi label task [Seb05]. That means each document can be assigned to one or more categories [GLWW00].

Document retrieval can also be treated as a classification problem, for instance by separating the documents into two classes. The classes are indicating if a document is relevant or not.[LJ98].

Automatic document classification is realised via supervised machine learning methods. Algorithms like Nearest Neighbour, Rocchio Centroid are state of the art classifiers. SVMs, support vector machines often establish better results [GLWW00].

Two phases are required for supervised machine learning [GLWW00]:

- Learning Phase
- Classification Phase

In the learning phase following different data sets are required [Seb05]:

- Training Set
- Validation Set
- Test Set

Those sets are created by a user, who has to select documents and assign them to categories by hand. Some classification algorithms need counter examples, what implies more effort for the user [GLWW00]. With the help of a given training set, which consists of sample documents that are assigned to defined categorises, a model can be learned. A validation set can be used to tune required parameters. The result of the *learning phase* is a model for each category. The test set is used to evaluate the under- or overfitting of the classification. If the model complexity is very high, overfitting can be a problem [Seb05].

In the *classification phase* categories are assigned automatically to new documents. This phase has to be efficient [GLWW00].

All algorithms that are used to classify documents have to use extracted features. *Feature vectors* have to be constructed, which are representing the documents. Different types of features can be extracted out of a document, like n-grams or morphological analysis features. The amount of features is in the most cases very high and with that the originated feature space. To get rid of unimportant features, a feature reduction can be applied [GLWW00]. Further problems, like the high dimension feature space exist. It

is very difficult to get all the information from a few keywords because of the complexity of the natural language. Also document characteristics like variable length and quality are known challenges. Classifiers should also be as efficient and accurate as possible [LJ98].

2.4.1 Naive Bayes Classifier

If a document should be assigned to a specific class is computed with the probability calculation given in Equation 2.45. c^*_{NB} is the class, which should be assigned to the document with the help of the naive Bayes classifier (NB).

$$c^*_{NB} = \arg\max_{c_j \epsilon C} P(c_j) \prod_{i=1}^{d} P(w_i|c_j) \tag{2.45}$$

$c_j \epsilon C$ are the different document classes. For each document D a word list W is dedicated, that is containing words w_i. $P(c_j)$ is the a priori probability for a class c_j and $P(w_i|c_j)$ the conditional probability of a word w_i to a class c_j. The assumption is that the probability of the words, which are occurring in the document are independent. If a word does not appear in a training set, the conditional probability is null. To avoid that the Laplace law of succession can be used (Equation 2.46). It makes a uniform prior assumption, that the words which are corresponding to a class, are equally probable.

$$P(w_i|c_j) = \frac{n_{ij} + 1}{n_j + k_j} \tag{2.46}$$

n_{ij} is the word frequency of w_i in c_j. k_j is the vocabulary size of c_j and n_j the number of words in class c_j [LJ98].

2.4.2 K Nearest Neighbour Classification

Each class is represented via its training documents, which are assigned to this one. The distance between a new document and all documents of the training set is calculated. Then the nearest K documents are selected and the class, which is referred to most of these documents, is chosen. To determine K a test set can be used [Got09].

Each document refers a feature vector which contains a weight for each word. This weight for instance can be the TF–IDF weight. To compare the similarity $S(D_1, D_2)$ of two documents D1 and D2 the cosinus similarity can be computed, see Equation 2.6.

In Figure 2.7 the algorithm is illustrated. If the black point is the new document and K is four, the documents, which are within the light blue circle, would be retrieved as most similar ones. Three of them are members of the "red star" class and so this one would be assigned [LJ98].

2.4.3 Subspace Model

The entire feature space is divided into m sub regions, which are representing the feature spaces for each class. The feature space of a class is for instance a vector of words. With the help of a projection matrix, the vector given in the original vector space is translated

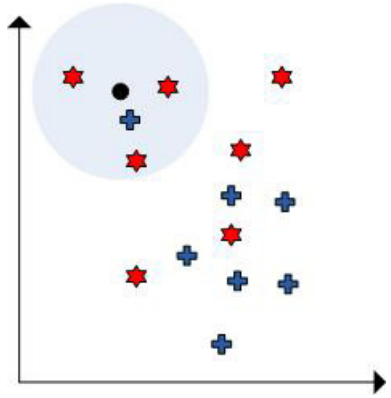

Figure 2.7: K Nearest Neighbour Classification

to a sub region. If a word occurs in a certain class, the entry in the projection matrix is assigned with the weight δ_j^k, see Equation 2.47, otherwise it is null.

$$delta_j^k = \frac{CLASSFREQ_{jk}}{log_2(DOCFREQ_j + 1)} \tag{2.47}$$

$CLASSFREQ_{jk}$ is the ratio of the count of documents which are corresponding to the class c_k and contain the word w_j, to the number of all documents, which are labelled with c_k. $DOCFREQ_j$ is the ratio of the number of all documents of the training set containing the word w_j to the count of all documents. Each document is projected to each feature vector. The projected feature vector is called $\vec{T_k}$. The sub space decision rule assigns the class, which has the largest euclidean norm $\|\vec{T_k}\|$, see Equation 2.48 [LJ98].

$$\|\vec{T_k}\| = \sqrt{\vec{T_k}^T * \vec{T_k}} \tag{2.48}$$

2.4.4 Decision Tree Classifier

The decision tree classifiers are robust to noisy data and suitable for learning disjunctive expression. Decision tree algorithms are for instance ID3 and it's successors called C4.5 and C5. Recursively a decision tree is constructed with the help of a top-down approach. At each level the attribute with the highest information gain is selected. C5 has the advantage of the so called rule- sets, which make the computation faster and less memory is used. Also adaptive boosting can be used to enhance the results. This means n classifiers are used to examine the errors of the $(i-1)$th classifier with the help of the ith one. This n classifiers build a voting scheme, which obtains the class to which the document corresponds [LJ98].

In Figure 2.8 a decision tree is shown. The leaves of the decision tree are the classes, that should be assigned. The nodes contain the attributes and with the help of specified rules an appropriate child node can be selected. The classification process starts at the root and based on the attribute and the corresponding rules the next node is selected, till a leave is reached [R.86].

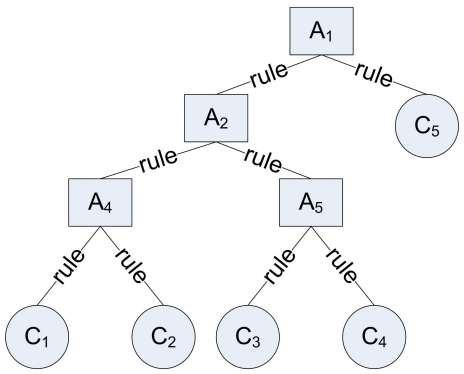

Figure 2.8: Decision Tree Classification

2.4.5 Rocchio Algorithm

The documents are represented as vector $\vec{d}$ containing weights, for instance TF–IDF ones. The centre of the given training data for each class j is the centroid z_j of the documents D_j that correspond to that class. The calculation of z_j is shown in Equation 2.49.

$$\bar{z_j} = \frac{1}{|D_j|} * \sum_{d \epsilon D_j} * \vec{d} \tag{2.49}$$

To assign a class to a new document the distances to these centroids are calculated. This distance for instance can be the number of hyperlinks or a similarity function (see Section 2.2.2). In Table 2.4 possible distance measurements, via the similarity function, are shown.

$$\begin{aligned} dist(d_1, d_2) &= 1 - sim(d_1, d_2) \\ dist(d_1, d_2) &= -log(sim(d1, d2)) \end{aligned}$$

Table 2.3: Distance Calculation via Similarity Function

It is also possible to get similarity parameters from a distance measurement (see Table 2.1).

$$\begin{aligned} sim(d_1, d_2) &= 1 - \frac{dist(d_1,d_2)}{dist_max} \\ sim(d_1, d_2) &= e^{-dist(d_1,d_2)} \\ sim(d_1, d_2) &= \frac{1}{1+dist(d_1,d_2)} \end{aligned}$$

Table 2.4: Similarity via Distance Calculation

The distance between two documents can be for instance the euclidean metric $\lambda(d_1, d_2)$, see Equation 2.50 [Got09].

$$\lambda(d_1, d_2) = \sqrt{\sum_{i=1}^{M} (w_1^{[i]} - w_2^{[i]})^2} \tag{2.50}$$

After the distance calculation, the centroid with the shortest distance is chosen and the corresponding class is assigned therefore.

In Figure 2.9 an illustration of the rocchio classification is shown. The green star and the cross indicate the centroids of their classes. If a new document, the green circle, should be classified the distances to these centroids are determined. In this picture the star class is closer and would be assigned [Got09].

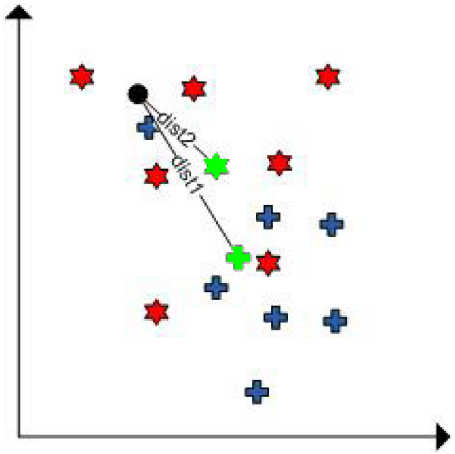

Figure 2.9: Rocchio Classification

2.4.6 Support Vector Machines

SVMs, support vector machines, are obtaining the best results for document categorisation. The algorithm determines, in a high dimensional space among all surfaces, a separation by the widest possible margin between positive and negative training examples. This separation is called maximum-margin hyperplane and accomplishes that the general error is minimised. A great advantage of this algorithm is that dimension reduction is not required [Seb05]. Support vector machines are appropriate for high dimensional classification tasks [Joa98]. It is also very robust against overfitting [Seb05].

In Figure 2.10 the maximum-margin hyperplane is shown. The vectors, which are very near to that, are called support vectors [Got09].

Normally, SVMs are used for binary classification, but recently also for multi-class classifications [Seb05]. To be able to use more classes, following methods can be used.

The method *"One versus All"* means, that for n classes n SVMs are used. The documents which are not referred to a specific class are combined into one class. Each SVM is trained with a specific class and the rest. A SVM can return the certain class or the one of the rest. A resulting class is obtained, if one SVM returns a specific class and the other SVMs the class of the rest. The methods are illustrated in Figure 2.11 [bDK05].

The second method is called *"Winner Takes All"*. Therefore $\frac{(n-1)*n}{2}$ SVMs are trained with the data of two different classes. Each class is compared to each other. The class, that was classified most, is chosen. An example is shown in Figure 2.11 [bDK05].

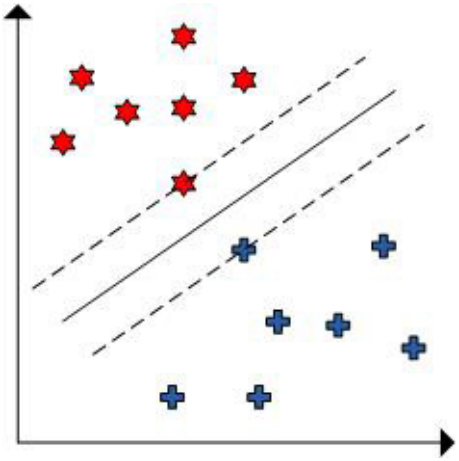

Figure 2.10: Support Vector Machine Classification

2.4.7 Combination of Multiple Classifiers

To combine the results of various classifies, different methods exist. Via a weighted linear combination, where the weights are defined by a user, more classifiers can be considered. A further approach is called "single voting". The class, that is assigned by the majority of the classifiers, is chosen. The method dynamic classification selects the class with the highest local accuracy for the given test samples. For instance the k-nearest neighbour algorithm is applied to find the neighbours of a document. The leave one out method is used for the training data to find the local accuracy by calculating the soft measure. This measurement determines the weight of each neighbour by the cosinus similarity to the document. Adaptive classification combination uses the highest accuracy among all used classifiers to determine the to assigning class [LJ98].

2.4.8 Boosting

Boosting is also an approach, which accomplishes better classification results with the help of more classifiers. These classifiers have to be from the same learning type. Each classifier is trained sequentially. The n-th classifier retrieves the results of the previous classifiers and tries to get the worst classified samples. Each document class pair is assigned to a weight, which indicates the effort for classifying a document to its correct class. The result of the boosting process is a weighted linear combination rule [Seb05].

2.4.9 Evaluation

Classifiers can be evaluated with the help of different parameters.

- **Training Efficiency:** time to build the classifier with the help of the given training set
- **Classifier Efficiency:** average time to classify documents
- **Effectiveness:** correctness of the classification

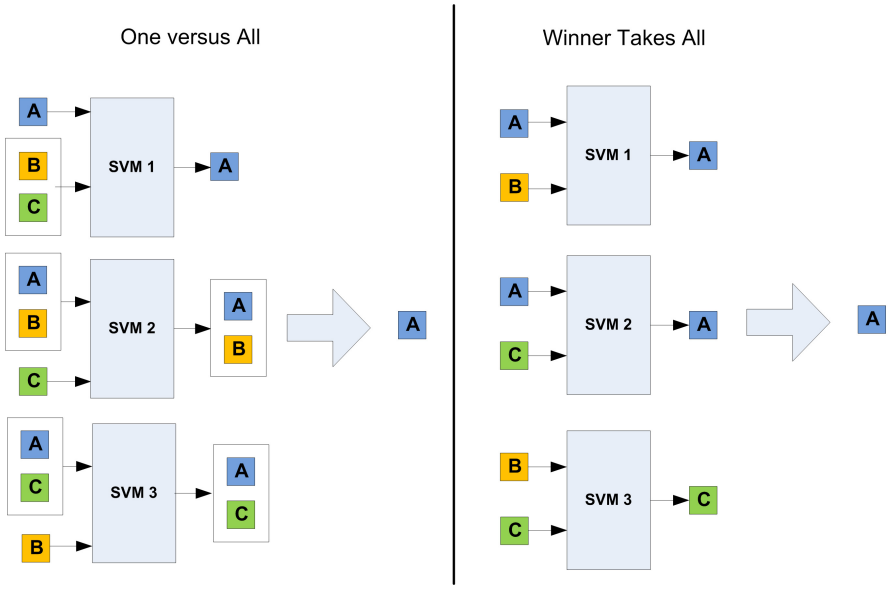

Figure 2.11: Multiclass SVM

For *Single- labelled* text categorisation the so called *accuracy* is used to measure the effectiveness. It is the percentage of correct classification decisions. *Multi- labelled* categorisations are evaluated with the help of *precision and recall*. Two different kinds of averaging among the different classes have to be considered, what is done via *micro- and macro- averaging* [Seb05].

2.5 Text Preparation

Text often must be preprocessed because of following problems[FHM08]:

Spelling mistakes

Polysemy: word with multiple meanings, for instance bank

Synonyms: words with the same meaning

Inflection: for instance park and parking

Following methods can be used to reduce these problems.

N-Grams: N-grams can be used to avoid problems like spelling mistakes and inflection. Instead of splitting the text into words, it is divided into more parts, that are used as features. The N defines the length of the created string parts. Trigrams, $N = 3$, are often used for similarity measurements in the information retrieval. The trigrams, for example for the text *test software* are: *_te tes est st_ t_s _so, sof, oft, ftw, twa, war, are, re_*. The underline represents a white space [MMC02].

Stemming: Stemming is used to reduce inflected words like play, playing and so on to their stem. For instance the word *playing* should be reduced to its root *play*. The

Porter stemming is a common algorithm. Stemming algorithms must not reduce a word to its morphological stem. The created stems only have to indicate related words [MS06, Got09].

Thesaurus and Dictionary: Dictionaries can be used to correct spelling mistakes. A thesaurus helps to get the synonyms for a specific word. With the help of query expansion, these words are added and a similarity computation can be improved [Got09].

Stopwords: Luhn said that words, which occur in a corpus too frequent or rarely, are not important for the search. The words, which are very frequent, are called stopwords [Got09]. These words should be removed from a text, because they are often irrelevant and so the noise of a text. Words like *is* and *a* are common stopwords [MS06]. The amount of stopwords in retrieval systems can be very different. Google for instance defines only 36 stopwords, in contrast a MySQL database declares 550. Stopwords removal can also lead to poor information retrieval. For instance, if a word like *no* is removed, the resulting text has a different meaning [Got09].

2.5.1 Keyword Extraction

Keywords can be extracted with the help of several approaches [oel09]:

Statistical Approaches: With the help of different statistical measurements, like the word occurrence in the document, keywords can be extracted. Familiar methods are TF–IDF and N–Grams. These methods are simple and don't need linguistic knowledge, so that they can be applied to different languages.

Linguistic Approaches: These mechanisms extract keywords with the help of linguistic knowledge. For instance, stopwords list, linguistic features-like the syntactical structure and thesauruses are used to improve the quality of statistical methods.

Machine Learning Approaches: With the help of a training set containing documents and a set of keywords per document, a model can be learned, which makes a keyword extraction possible.

Keyword Extraction Evaluation

In the paper [Ass07] a data set called DUC2002 is used. With the help of a tool, called ROGUE, they have evaluated the results. ROGUE is a toolkit for evaluation of document summation applications. It counts the overlapping units like n-gram, word phrases and so on. To test the keyword extraction, they used 34 documents from the data set and extracted manually about ten keywords per document (average 6.8 keywords per document). Via different keyword extraction methods, they obtained ten keywords per document. With the help of WordNet they reduced each word to it's basic form. Precision, Recall and F-Measure, see Equation 2.51, were calculated to evaluate the methods.

$$F - Measure = \frac{2 * precision * recall}{(precision + recall)} \tag{2.51}$$

In the paper [MI03] manually extracted keywords are used to evaluate their results. 20 authors where asked to select important keywords form their documents. They also obtained 15 terms by different keyword extractors and calculated the precision and recall to evaluate their results. Via Equation 2.52 the precision is calculated.

$$precision = \frac{terms_{important}}{terms_{obtained}} \tag{2.52}$$

To determine the coverage, the authors were asked to select five or more indispensable keywords.

$$coverage = \frac{terms_{indispensable}}{terms_{obtained}} \tag{2.53}$$

Similar evaluation approaches could be found in [vdPPRG04] and [Dep].

2.5.2 Feature Reduction

Feature reduction is used to lower the dimension of the feature space. This can help to avoid overfitting and to reduce the computation time [Seb05]. Different kinds of feature reduction techniques exist. A possibility is the selection of terms, which have the highest weights. This approach is called individual best feature. The weights can be calculated via mutual information. Other well known approaches are sequential "forward/backward" and "plus l-take away r" selection. These methods have a better performance, but the computation costs are high.

A further one is the feature extraction [LJ98]. To reduce the dimension of the features, PCA, principal compound analysis, which is called latent semantic indexing for document retrieval, can be used [GLWW00]. Words are correlating within a group of documents. So for instance k-means clustering can be applied within a category to find different classes of words. This approach is called term grouping in subspace [LJ98]. Supervised approaches, like the term clustering, often achieve better results [Seb05].

SVD

Singular value decompostion (SVD) can be used to perform a feature reduction.

$$B = U\Gamma V' \tag{2.54}$$

Equation 2.54 shows the computation of a SVD (singular value decomposition). Γ is a diagonal matrix, which includes the sorted eigenvalues of B, see Equation 2.55. Sorted means, that $eig_1 > eig_2 > ... > eig_m$.

$$\Gamma = \begin{pmatrix} eig_1 & 0 & 0 & ... \\ 0 & eig_2 & 0 & ... \\ ... & & & \\ 0 & 0 & ... & eig_m \end{pmatrix} \tag{2.55}$$

The columns of V contain the eigenvectors of $B' * B$. If B is a $m \times n$ matrix, the components of the SVD would be following ones:

- Size of U is $m \times m$

- Size of Γ is $m \times n$
- Size of V' is $n \times n$

To reduce the dimension of B from m to k, the bottom rows, which are containing the smallest singular values, are deleted from Γ and V and the corresponding columns from U, so that the sizes are as following:

- Size of $\tilde{U}$ is $m \times k$
- Size of $\tilde{\Gamma}$ is $k \times k$
- Size of $\tilde{V}'$ is $k \times n$

The data can be projected by the multiplication of $\tilde{U}'$ with B. The size of the resulting matrix of the multiplication of this $k \times m$ matrix and the $m \times n$ data is $k \times n$. So the dimension is reduced from m to k [PWWB09].

In an information retrieval system B is the vector space model of a document collection, see Section 2.2.2.

PCA

Principal component analyse helps to reduce the dimension of feature spaces without the loss of much information. That means that data with the dimension n will be reduced to the dimension m. In the case of a vector space model the whole data is represented as $n \times k$ matrix, where n would be the number of features and k the amount of text items. After the PCA, this matrix can be reduced to a size of $m \times k$ where $m < n$. To perform a PCA, at first the mean value of each dimension has to be subtracted, so that the data is unbiased. After that the covariance matrix of the $data_{unbiased}$ is calculated, see Equation 2.56, and the eigenvalues eig_i and eigenvectors vec_i are computed. High eigenvalues indicate that the specific dimension has a high contribution. So the corresponding eigenvectors are called principal components. To reduce the dimension of the data only large eigenvalues are chosen. With the help of these, a feature vector, see Equation 2.57 and an eigenvector matrix can be created.

$$Covariance = \frac{1}{n-1} * data_{unbiased} * data'_{unbiased} \tag{2.56}$$

$$FeatureVector = [eig_1, eig_2, eig_3, ..., eig_n] \tag{2.57}$$

$$FeatureMatrix = [vec_1, vec_2, vec_3, ..., vec_n] \tag{2.58}$$

The feature vector is a row vector with the length m and the feature matrix has the size $n \times m$. The unbiased data has the size $n \times k$. The dimension reduction can be performed via Equation 2.59. The resulting matrix size of the multiplication of a $m \times n$ and $n \times k$ matrix is $m \times k$ and so the dimension is reduced [Smi02].

$$data_{reduced} = FeatureMatrix' * data_{unbiased} \tag{2.59}$$

If Y is calculated via the Equation 2.60 and the SVD is computed the resulting V would be the $FeatureMatrix$. This is because V are the eigenvectors of $Y' * Y$ and this is the $Covariance$, see Equation 2.61.

$$Y = \frac{1}{\sqrt{n-1}} * data'_{unbiased} \tag{2.60}$$

$$Y' * Y = \frac{1}{n-1} data_{unbiased} * data'_{unbiased} = Covariance \tag{2.61}$$

Following steps have to be performed to calculate the PCA:

1. Subtract the mean of each dimension
2. Calculate covariance of the unbiased data
3. Calculate the eigenvectors and eigenvalues of the covariance
4. Select the eigenvectors with the highest eingenvalues and create the Feature Matrix, see Equation 2.58
5. Project the data, see Equation 2.59

The PCA can also be calculated with the help ofthe SVD:

1. Subtract the mean of each dimension
2. Calculate Y, see Equation 2.60
3. Calculate V via the SVD, see Equation 2.54
4. V is equal to the FeatureMatrix and the data can be projected, see Equation 2.59

2.6 Microsoft SharePoint

Microsoft SharePoint is a web platform for business collaboration. SharePoint interacts well with Microsoft Office, which is a great advantage for the end user. The creation of new sites, for instance blogs, wikis, teamsites and many more, is very simple. Lists can be modified easily, like adding columns or changing the current view. The view of a list defines which list items should be displayed. Files, for instance documents, can be uploaded with the help of document libraries. Workflows that are representing processes, can be created. This ones can for instance send an e- mail if a list item has changed. A great advantage of SharePoint is, that a lot can be modified and adjusted to the requirements you need.[MB10] A new feature of SharePoint 2010 is the possibility of rating and tagging [RAS+10]. SharePoint can also be seen as developer platform. Common tools are Visual Studio and the SharePoint Designer. The Visual Studio 2010 provides templates for SharePoint directly. Web and .Net technologies build the foundation of SharePoint. Linq and CAML can be used to query lists [RAS+10].

The architecture of a Microsoft SharePoint has a tree like structure, see Figure 2.12. Within each web list, content types and so on, can be created [RAS+10].

2.6.1 Webpart

Webparts are custom elements, which can be added to each page. Each webpart can be personalized, this means that properties can be set, which influence its appearance and

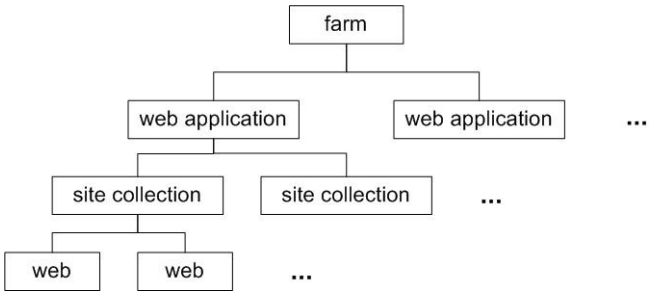

Figure 2.12: Microsoft Sharepoint architecture

can be saved. Custom webpart controls can be created by deriving from the base WebPart control. Also existing ASP.Net–, standard Web server–, custom server– and user controls can be used as webpart control [RAS+10].

2.6.2 Features

With the help of features the user can modify and enhance the functionality of SharePoint. Different tasks like adding items, copying files and so on, can be executed with the help of features. Physically features are placed within the *Program Files \Common Files \Microsoft Shared \Web Server Extensions \14 \TEMPLATE \FEATURES* folder. In each folder under that path, different files are contained, which provide the function for each specific feature. The feature is defined via an XML file [LMV08]. With the help of a scope attribute, the level, where the feature should be available can be adjusted. Features with the scope farm are available from the SharePoint central administration and within the entire farm. They can be activated or deactivated under the manage farm features setting, which is placed in the system setting in the central administration. Web application features can be set within the web application configuration, which is available under the Application Management → manage Web Applications page. After the selection of a specific web application, the Manage Feature page can be opened from the ribbon. Site feature can be adjusted under the site collection settings and web features under the site settings. These site settings can be opened via the site actions, which are normally placed at the top left of each site, see Figure 2.13 [RAS+10].

Dependencies between features can also be declared within the feature xml. This means that a feature can only be activated, if other declared features, which are at the same or higher level, are activated first. A feature receiver can be added to each feature, which makes it possible to add a code to different events:

- FeatureInstalling
- FeatureActivated
- FeatureDeactivating
- FeatureUninstalling

The life cycle of a feature is shown in Figure 2.14 [LMV08].

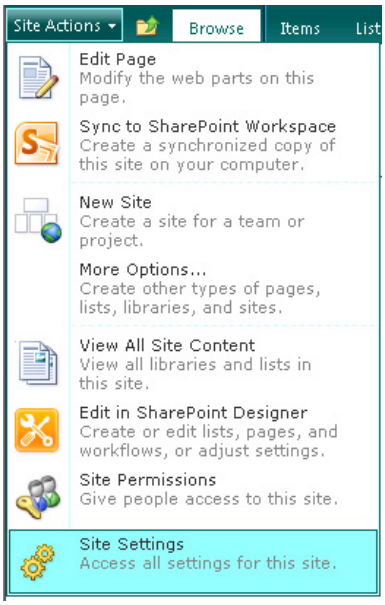

Figure 2.13: Microsoft Sharepoint Site Settings

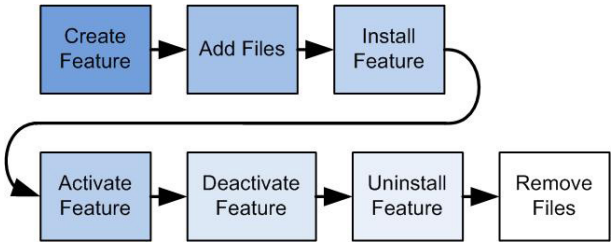

Figure 2.14: Feature Receiver Life Cycle [LMV08]

2.6.3 Event Receiver

With the help of event receivers a custom code can be added to different events. Following types of event receivers can be implemented:

- List Events
- List Item Events
- List Email Events
- Web Events
- List Workfow Events

List Item Events are for instance ItemUpdated, ItemAdded and so on.

2.6.4 Timer Job

A timer job is executed at a specific time or within a time span, which is defined with the help of a schedule. The intervals can be defined minutely, hourly, daily, weekly or monthly [RAS+10].

2.7 Search Engines

Different kinds of search engines exist like enterprise and web search engines. Information retrieval in the web or for a controlled collection differs. The data, that is available on the web, is much more various. The language can be any of the human or programming one. The text can be machine generated, like log files, and the file formats can differ, like images, HTML and so on. Also web search engines have to deal with manipulations. Numerous companies are trying to influence the rank because of profit. Metadata, that is not visible to the user, is abused. Algorithms, which are working very well for a controlled collection, often achieve bad results for the web search [SL98].

In this section, some search engines are described. Only few are listed, because it is not easy to find specific information about their relevance ranking algorithms.

2.7.1 Google Search

Google is a very effective web search engine. An essentially used algorithm is the PageRank, which was developed by Larry Page and Sergey Brin, the Goolge founders, on the Standford University. By determining the link structure in the WWW, it brings order into the web [Mol09].

Major components are the crawler, indexer and the search unit. The list of the URLs, which is obtained from the different distributed crawlers, is saved. To each crawled URL an ID (docID) is assigned. After that, these new pages are indexed. Each document is represented by a set of word occurrences, the so called hits. Each hit also contains information, like the position of the word in the text, the font size and capitalisation. Then the indexer creates a partially forwarded index with these sorted hits. Important information of the links, which occur on the page, are also saved in a separate anchor file. With the

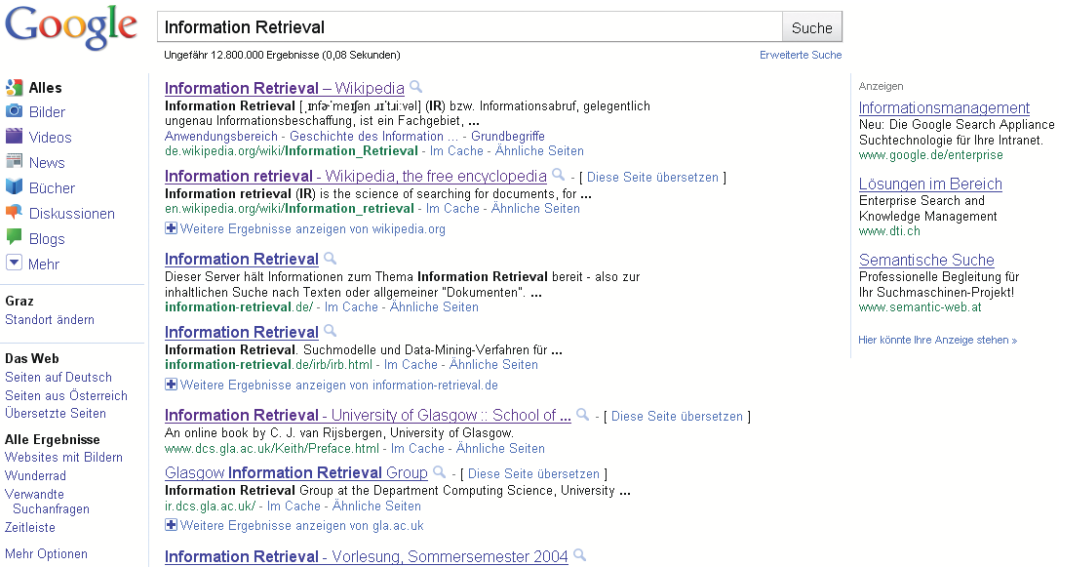

Figure 2.15: Google Website

help of this information, the anchor text and the information about the link points, can be determined. An URL resolver stores the information, which is available in the anchor file, in a link database, that is later used to calculate the PageRank. The anchor text is also added to the forward index with its corresponding docID. A so called sorter generates an inverted index out of the forward index [SL98].

With the help of the inverted index, the web search engine obtains documents, which are matching a given query. For each of these documents a relevance score is calculated and an output page is generated, which shows the documents sorted with their title and a short summary. Also a spell- checking and an ad-serving system is executed. The ad-serving system determines relevant advertisements. To provide a short computation time the search is highly parallelized. The inverted index is randomly divided into the so called index shards. Each of them contains a subset of documents of the full index [BDH03].

2.7.2 Google Scholar

Google Scholar is a search engine for academic literature for all academic disciplines. A citation extraction is performed and with the help of the autonomous citation indexing (ACI), the ranking of the papers is influenced. Papers, which are cited often, are ranked higher. Google Scholar also executes a full text index. The relevance ranking considers the full text, the author, the publications in which the paper is cited, and how often it was cited generally [MW06].

2.7.3 Apache Lucene

Apache Lucene is an OpenSource information retrieval system. It is implemented in Java. A document is prepared via different kinds of parsers. These parsers can be applied to different document types. The unstructured text is filtered and added to the index. These language dependent filters can remove stop words, perform stemming or transform the text

to lower case. After that, a tokenizer splits the text with the help of specific tokens, for instance white space. The inverted index can be created ad hoc or incremental. Incremental means that the existing index is upgraded. Information like the local term frequency, the term position, the field name and the term are also saved. The Lucene ranking function is shown in Equation 2.62. $tf_d(t)$ is the rooted term frequency of the term t in the document d. $idf(t)$ is the inverse document frequency of the term t. The $boost_d(f(t))$ is a factor for a field f in the document d. $norm_d(f(t))$ is a normalisation value for the field f by the amount of terms in that specific field. This value is calculated during the indexing phase. $coord(q, d)$ is a factor, which is based on the term frequency in the document d and q. $norm_q(q)$ is also a normalisation value for the query q, see Equation 2.63 [Kür06].

$$rank(q,d) = coord(q,d) * norm_q(q) * \sum_{tinq} (tf_d(t) * idf(t)^2 * boost_d(f(t)) * norm_d(f(t))) \quad (2.62)$$

$$norm_q(q) = \frac{boost_q(q)^2}{\sum_{tinq}(idf(t) * boost_q(f(t)))^2} \quad (2.63)$$

2.7.4 Microsoft SharePoint Search 2010

Microsoft Sharepoint provides a search application, which consists of three different parts.

Crawler

The Crawler browses the given content sources, which can be added in the central administration. For each site collection a content source is automatically created. With the help of defined rules it determines the URL addresses of the sub webs and the content below [RAS+10].

Indexer

Each content, which is found by the crawler, is indexed. Indexing is used to make queries faster. Different content, like images, documents and so on, are represented as text. Therefore different IFilters are used, which extract the textual information from the various file types. Wordbreaker and stemming are also used to filter the obtained data. When a query is executed, this data is used to search relevant items [RAS+10].

Query

Two different query types are implemented:

KeywordQuery: With the help of a set of keywords relevant items are computed.

FullTextSQLQuery: A specific SQL syntax helps to search relevant items. This type of query enables a more specific search opportunity.

For each item a rank value is calculated, which indicates the relevance for the given query.

Ranking Models

Static and dynamic information is used to retrieve relevant items. Static information is provided before the query is executed, like the filetype, language, click distance and URL depth. The anchor text, text representation, property weighting and the URL matching are used for the dynamic ranking. The base of the content rank relevance calculation is the so called Okapi BM25F algorithm, which is described in Section 2.2.4 [MSD09]. It's a state of the art TF– IDF algorithm. The adjustable parameters, like weights, can be adjusted in the ranking models. These are defined XML data, which are saved on the SQL Search Service Application database in the MSSRankingModels table. In Figure 2.16 a part of such a xml data is shown.

```
<?xml version="1.0" ?>
- <RankingModel2Stage name="MainResultsDefaultRankingModel" description="Default ranking model for main results" id="8F6FD
    xmlns="urn:Microsoft.Search.Ranking.Model.2NN">
  - <RankingModel2NN id="1EAD08A0-A521-4ef1-826D-9F73C0790C11">

            ...
    - <RankingFeatures>
      - <Static name="ClickDistance" pid="96" default="5">
          <Transform type="InvRational" k="0.0900786349287429" />
        - <Layer1Weights>
            <Weight>1.86902034145632</Weight>
          </Layer1Weights>
        </Static>
            ...
      - <BM25Main name="ContentRank" k1="1.0000000000">
        - <Layer1Weights>
            <Weight>0.40306699650789</Weight>
          </Layer1Weights>
        - <Properties>
            <Property name="Body" pid="1" w="0.00966159625448698" b="0.0504303620214618" />
            <Property name="Title" pid="2" w="1.29526330954758" b="1.20975920969803" />
            <Property name="Author" pid="3" w="0.314466873313388" b="1.13417184851726" />
            <Property name="DisplayName" pid="56" w="0.39785658626107" b="0.253098660308997" />
```

Figure 2.16: XML Ranking Model of MS SharePoint

Wordbreaker

The wordbreaker is used to divide a given text into words. Also stemming, noisy words (stopwords) filtering and a thesaurus can be applied. If stemming is enabled, the word stems are also added to the query. Noisy words and the thesaurus for different languages are defined via files , that are saved on the server. These files can be edited [RAS+10].

Chapter 3

Design of the Implemented Systems

In this chapter the context and the design of our implementation are described. The system for the evaluation of the different kinds of algorithms is implemented in Matlab. The chosen algorithms are implemented with the programming language C# .Net. The system is also integrated within Microsoft SharePoint 2010. The components, which have to be realised, are described within this chapter.

3.1 Context

The information retrieval system must be implemented within MS Sharepoint 2010 and integrated within the *SharePoint Customer Service Desk* of the company Solvion. Administrators and normal users can use this system. The administrators have to configure the system and can remove feedback information.

Most of the available data sets of *SharePoint Customer Service Desk* contain only about 50 tickets, except one other which contains 1370. This one is used to evaluate the system.

Each ticket has a title, description and category field. The mean word count and dispersion are shown in Table 3.1.

25 different categories are assigned, for instance 'Windows', 'Office', 'Hardware' and so on. To generate a second test set, a subset of 255 tickets is extracted and eight categories are assigned by hand.

Field	Mean Word Count	Dispersion
Title	4.8584	1.6997
Body	6.1920	4.1181
Title and Body	11.0504	4.0761

Table 3.1: Mean Word Count and Dispersion of the Test Set

In Figure 3.1 the dictionary size corresponding to the ticket count is shown. Also the approximation of Heap, see Equation 2.1, is plotted with $b = 0.7324$ and $k = 3.3492$.

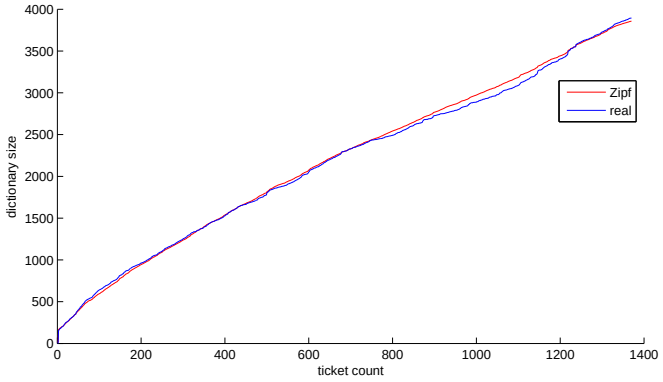

Figure 3.1: Dictionary Size of Data Set

3.2 Evaluation System

The classification and similarity measurement for a given data set have to be evaluated. The *MAP* value and the percentage of the correct classified *correct*[%] queries are calculated to compare the different implemented algorithms, see Figure 3.2.

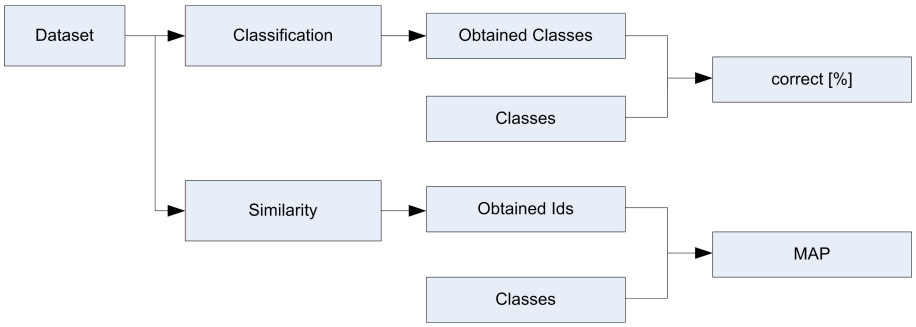

Figure 3.2: Components of the Evaluation System

The correct percentage is determined by a "one to n cross validation" . That means that $n-1$ items are used to train the system and one item, which is not a member of the training subset, is classified. After that the result is compared to the real assigned class and over all items a percentage of "correct classified" can be calculated.

In Figure 3.3 the methods that are evaluated, are shown.

3.2.1 Data

For the similarity and classification evaluation two different files are needed. One file includes per row the text of the ticket and the other one the assigned class to the corresponding ticket, see Figure 3.4. In that example, the classes are described by different numbers. It is also possible to set a name like "Telefon" as class name.

To evaluate the SharePoint Search a file is generated. Therefore the list is crawled and indexed with the help of the *Search Service Application*. After that, each list entry is used

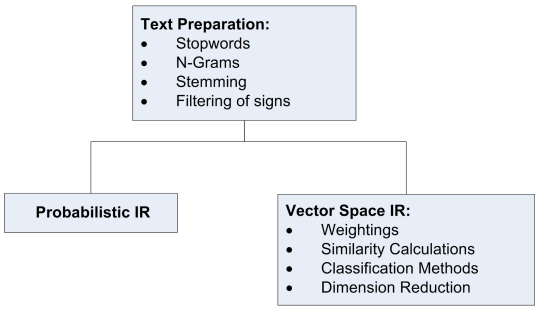

Figure 3.3: Methods to Evaluate

line

Ticket File	Class File
Telefon Telefonhörer hat aussetzer	**1**
Visual Desktop Bewerber - Auswertungen - Fachkombinationslisten für Englisch funktioniert nicht!	2
Drucker Verknüpfung für den Drucker im Sek. Schulpsych. (B) benötigt	3
Drucker Drucker sehr langsam!	3
Schuluser BHAK Vo User Rowanschek neu anlegen	4
Drucker Toner für Drucker HP LJ 1300 tauschen	3
Notebook Benötigt Notebooks für Veranstaltung in der Fasseiche am 22.09.2009!	5
Handy Proxy	5
WLAN Einstieg div. Notebooks Fremde Notebooks zu Schulungszwecken mit dem Gäste-WLAN verbinden	5
Zeiterfassung BSI Lyssy Anlegen, Einrichten, Ersteinstieg, kurze Einschulung	4
VD - Änderung e-Mail-Adresse Änderung e-Mail-Adresse	4
...	...

Figure 3.4: Evaluation Files

to create a query. Therefore the user has to set the list name and the field. This extracted query is executed with the help of the *KeywordQuery* with the option, that any of those words should be contained in the result. Only results are accepted, which are within the source list. The result list contains a ranking value and the URL to the item. The item ID is extracted from the URL. Each row of the result file, which is used for the evaluation, represents a query and contains those retrieved IDs, see Figure 3.5.

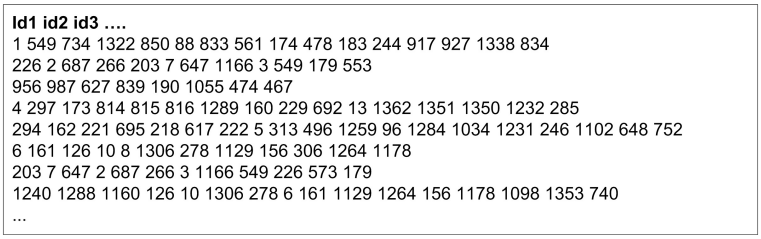

Figure 3.5: Generated Microsoft Result File

3.2.2 Similarity Evaluation

The structure of the implementation of the similarity calculation evaluation is shown in Figure 3.6. The red arrows indicate the control flow and the black ones the functions, that are executed. The file *evaluationSimilarity* performs following steps:

1. **TestDefintions:** This Matlab script contains the paths to the text files, which contain the tickets and class assignments. Also the weighting schemes and the folder, to which the results should be written to, are declared.

2. **Init:** This function reads the ticket file, creates the vector space model and can normalise the given data.

3. **DimensionReduction:** In the folder dimension reduction different scripts, for instance *MyPCA*, exist. Those can be called and the vector space model will be modified.

4. **ReadMicrosftSearchResultFile:** To evaluate the MS SharePoint Search a file is created, which contains the information of the query index and the retrieved document IDs. This file is read with the help of this function.

5. **EvaluateMicrosoftSearchMAP:** After reading the file, this function is used to calculate the MAP value.

6. **EvaluateSimilarityMethodMAP:** This function can execute different similarity calculations and return the MAP value.

At first we used the TMG- toolbox, which is described in [ZG05], to build the vector space model. To be able to evaluate N-grams also, an own calculation was implemented. The $p@k$ value, which is used for the MAP calculation, is determined with the help of the knowledge about the assigned classes. If a retrieved item has the same class as the query, it is assumed to be relevant.

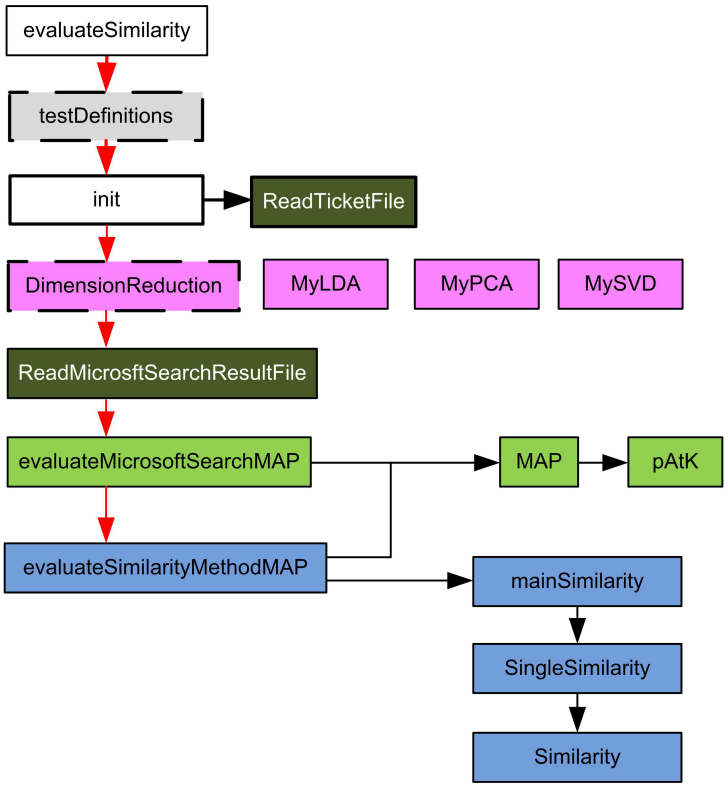

Figure 3.6: Similarity Calculation Evaluation System

3.2.3 Classification Evaluation

The classification is congeneric to the similarity evaluation. Instead of the *MAP* value the correct classified percentage is calculated. In Figure 3.7 the system without the initial components is shown.

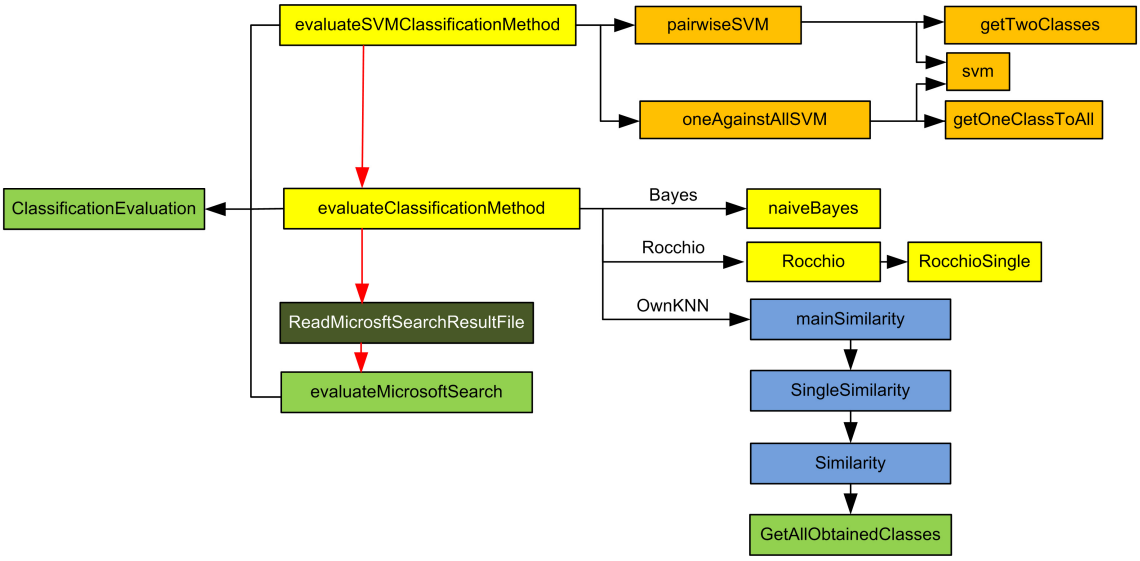

Figure 3.7: Classification Calculation Evaluation System

The implemented k nearest neighbour (KNN) algorithm first calculates a similarity measurement for all items, then obtains the class by getting the k most similar items and the most common class is selected. The Rocchio calculation also uses the similarity measurement to get the nearest centroid.

3.3 SharePoint Information Retrieval System

After the evaluation of the different algorithms the information retrieval system must be implemented in SharePoint and integrated within the *SharePoint Customer Service Desk*.

Following use cases are exist:

- A ticket arrives and it is automatically assigned to a support assistant or to a category.
- Possible solutions for an incoming problem should be retrieved automatically.
- By interacting with the system, it should enhance the quality of the receipt solution and classification.

To realise the system following components have to be implemented.

3.3.1 Components

The system is divided into different components, see Figure 3.8. In this figure it is observable that the infrastructure is a central component, that access the data for the information

retrieval. The content processor is application dependent, that means it must be implemented if an other list type, for instance a task list should be used. The other components should be as generic as possible, to be able to use it also for other applications.

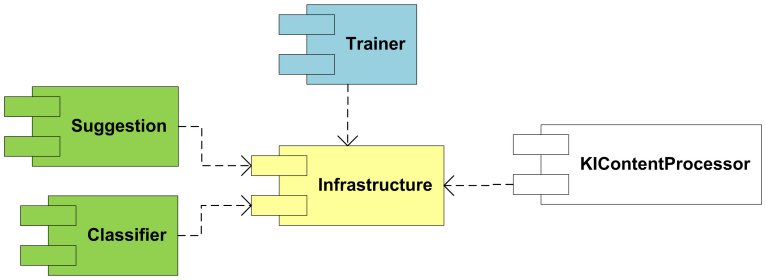

Figure 3.8: Components of the Information Retrieval System within Sharepoint

The interaction of the components is shown in Figure 3.9. An incoming ticket is handled by the content processor and the text information is represented. The trainer also modifies that representation and saves information of the classes. The text representation is used for the similarity and classification process. The feedback functionality, which is based on the similarity measurement, also influences the text representation.

In Figure 3.10 the different SharePoint features and their scopes are shown. The logging feature provides the infrastructure for the logging. The password generator creates the password for the SQL database login and saves it to the farm property bag. The functionality of the other features are described later in this chapter.

3.3.2 Infrastructure

The infrastructure creates and deletes the environment, where the required data for the information retrieval are saved. Tables, users and schemas are created within a Microsoft SQL database with the help of a web scoped SharePoint feature receiver. When it is activated the tables are created and by the deactivation all tables are removed. A login user is required to avoid permission problems by connecting to the database. At first we used windows authentication. With that the problem occurred, that only the user, who had activated the feature, was able to connect and save data to the *SolvionKIDatabase*

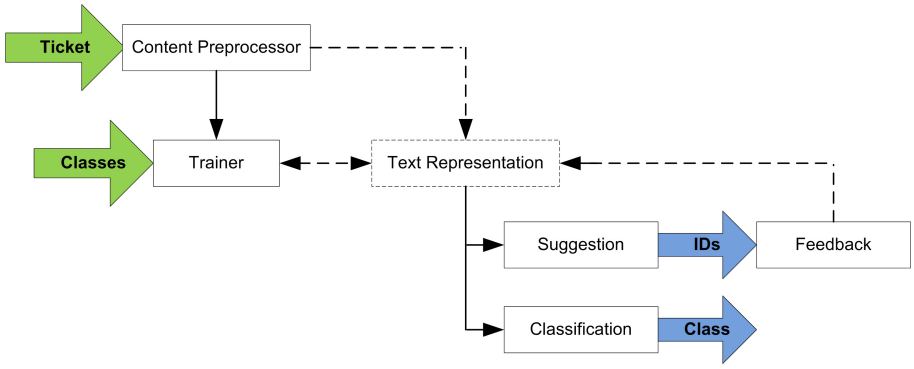

Figure 3.9: SharePoint Information Retrieval System Overview

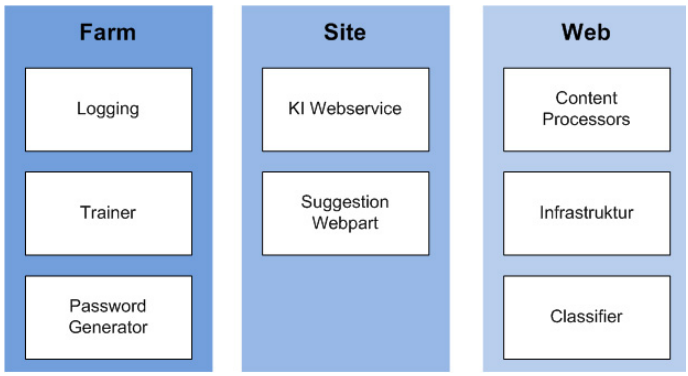

Figure 3.10: Feature Scopes of the different Components

database. Because of that, a login user was created, with the name *SKILoginUser*. The required password is a GUID, which is saved in the farm property bag. The infrastructure is a web scoped feature. For each web a schema is created. The name of that schema is the web GUID. To be able to switch between that schemas, the default schema of a user, called *SKIUser*, which executes the queries, has to be changed. That user also must be created before the specific tables are established. The steps which should be executed are following ones:

1. Create login user *SKILoginUser*.
2. Create database *SolvionKIDatabase*
3. Create "global" tables
4. Create schema
5. Create user *SKIUser*
6. Alter schema and create web dependent tables

In Figure 3.11 the relations between the tables are shown.
Tables which are not "global" are following ones:

- **Content Processors:** Contains the information about the installed content processors. The information of the web and list are saved. Also an ID is created to be able to get particular items from different lists.
- **Trainer Settings:** In the training settings the URLs of the webs are saved which should be trained.

The tables, which are unique for each web, are:

- **Classifier Settings:** The classifier setting include information which are required to perform the classification.
- **Contents:** The content table contains the preprocessed text. The content is not required for further computations, but is helpful for debugging and demonstration.

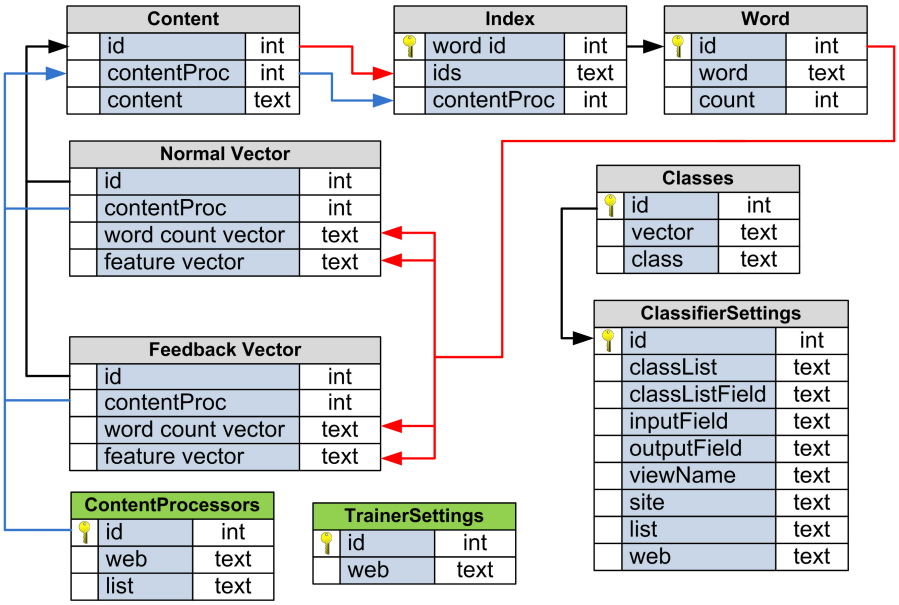

Figure 3.11: Database Entity-Relationship Model

- **Words:** That table contains the dictionary that means words with a generated ID and the global word count.
- **Index:** The inverted index contains for each word the text IDs, which include it.
- **Normal Vectors:** The feature vector and local word count vector are saved within this table, referred to a specific ID.
- **Feedback Vectors:** This table includes the vectors which are generated with the help of the feedback function.
- **Classes:** In this table the classes and their vectors which are created during the training phase are saved.

When a new content processor is created, it has to register in the *ContentProcessors* table. That means the URL of the web and the list GUID have to be saved. It is possible that a processor is attached to more than one list, for instance for a specific list type. For each of the saved lists an ID is generated automatically. With the help of that ID, it is possible to obtain within the *Contents*, *NormalVector*, *FeedbackVector* and *Index* table, data from a specific list. The relations are shown in Figure 3.11 with the blue lines. The black lines which point to the ID of a specific entry show the relationships between the tables. With the help of that ID, which corresponds to the item ID of the SharePoint list, the required data of a specific list item can be determined.

For each SharePoint list more than one classifier can be applied. The required settings are saved within the *ClassifierSettings* table. The centroids, which are required for the Rocchio classification algorithm are placed within the *Classes* table. With the help of the ID, which is generated for the classifier setting, the corresponding class information can be obtained.

The red lines in Figure 3.11 are indicating that the IDs of the source list are contained within the pointed text field. This is used for the index and vector tables. These item IDs are saved in the inverted index to each word and content processor ID and indicate if this word occurs in the text or not. The words table, more precise "dictionary" includes the word referred to an ID and the global word count.

3.3.3 Content Processor

The content processor has to transform and save the data, which should be available for the information retrieval, to the infrastructure. It is realised as a list event receiver and fired if a list item is added, updated or deleted.

Following calculations have to be executed:

1. If a ticket or action is added or modified the ID of a ticket is obtained and the subjects and descriptions texts are concatenated and saved in the content database.
2. Calculate word count vector and save it to normal vector table with the ticket and content processor ID:
   (a) Split text into words
   (b) If the word does not exist in the dictionary (word table), insert it and set the global word count to null
   (c) Get the word ID and count the words in the text
   (d) Create a vector with the help of the IDs and counts and save it
3. Add ID to the inverted index.
4. With the help of the global word counts and word count vector calculate and save the feature vector.

If the ticket is removed its entries are deleted from the content and vector tables and the ID is removed from the inverted index.

Very important to mention is the fact, that the global vector counts of existing words in the dictionary are not changed. This is not done, because otherwise all feature vectors would have to be recalculated. The global word counts are updated during the training phase.

3.3.4 Trainer

The trainer includes the computationally intensive calculations. A schedule defines when it should be executed. It is realised as SharePoint timer job. Following calculations are performed:

- update the global word count in the word table by counting the IDs in the inverted index,
- delete words form the word table with a global word count of null,
- recalculate the feature vectors of the vector tables, and

- calculate the class centroids for the Rocchio classification.

In Figure 3.12 the configuration page is displayed. This page is used to add web URLs and to set the schedule. It can be opened within the Central Administration → Monitoring → Timer Jobs → AI Trainer Settings.

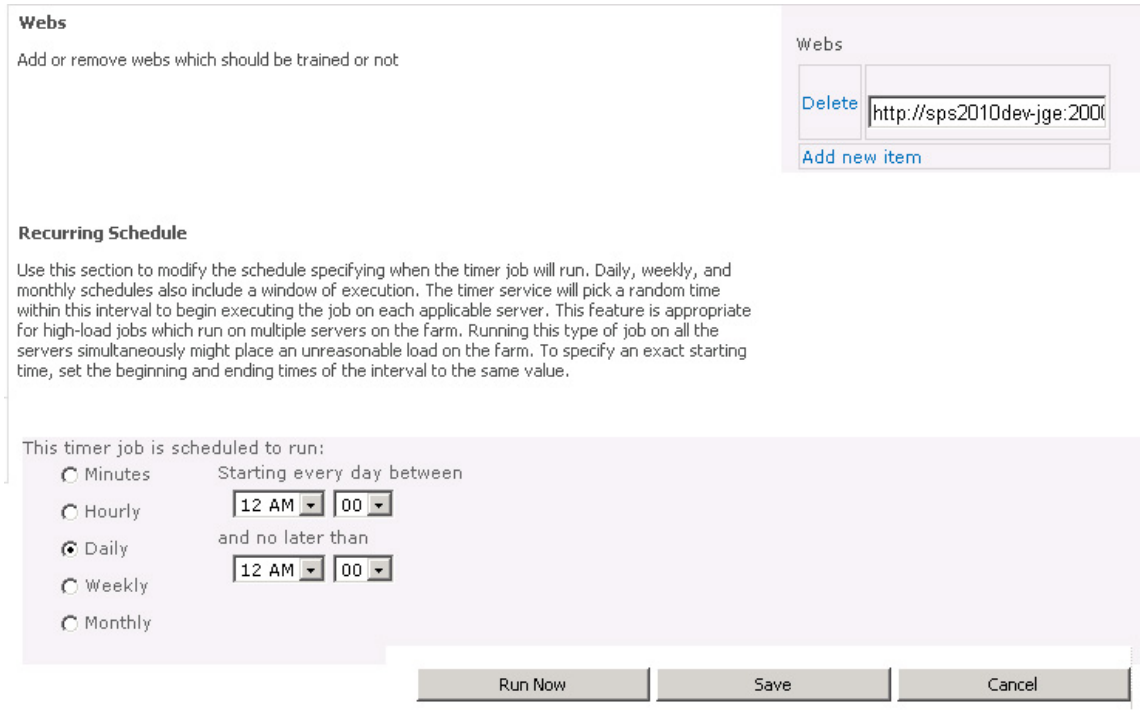

Figure 3.12: Trainer Setting Page

3.3.5 Similarity Measurement

The similarity measurement is used to search possible solutions. Therefore the saved data of the content processor are used. With the help of the item ID and the list GUID an entry is obtained in the MS SQL database. It is also possible that the vector with the feedback or the normal one can be obtained. To search similar entries the similarity between vectors is calculated and the IDs of the most similar ones are returned. These operations are implemented with a WSP web service, which has following interfaces:

- *int[] GetNearestNeighbors(int itemId, string sourceListGuid, string destListGuid, bool includeFeedback, int num, string webUrl);*
- *void Feedback(int itemId, string listGuidSource, int[] relevantItemIds, int[] notRelevantItemIds, string listGuidDest, string webUrl);*
- *void ClearFeedback(int itemId, string listGuid, string webUrl);*
- *string GetWebUrl(string listGuid, string webUrl);*

The method *string GetWebUrl(string listGuid, string webUrl)* searches the *listGuid* in the content processor table and returns the saved web GUID. The *webUrl* of the method parameters is used to get information of the used SQL database.

The GUI for information retrieval system is realised via a SharePoint webpart. This one communicates with the web service and displays the results. The presentation of the webpart can be modified via an xslt file. Also implicit and explicit feedback functionalities are provided. In Figure 3.13 the web part is shown for the implicit feedback. There the user can rate items and apply feedback. In Figure 3.13 the rating and the result after the button click are shown. In the result the relevant item is ranked higher and the not relevant item falls out of the range of visible items.

Figure 3.13: Suggestion Webpart With Implicit Feedback

In Figure 3.14 the suggestion webpart with explicit feedback functionality is shown. The star controls can't be used for the rating like in the implicit variant. They only show the rating information of the list entry, which has no influence on the relevance ranking. If a user clicks on the action or answer button the corresponding item is used for the feedback calculation. Also a popup window is opened by clicking on the action, info or answer button.

In Figure 3.15 the webpart properties are displayed. The information about the item, which should be searched within the database, is retrieved over the URL: *"http://url/site.aspx?List=**LISTGUID**&ID=**ItemID**"*. The field name of the list or id parameter can be adjusted in the properties.

The list, in which an item is searched, is defined via its GUID, the *"Search List GUID"*.

The *"Number"* defines the amount of displayed results and the dialog height and width define the size of the popup window, which is opened, if you click on an info, mail or action button.

Within the administrator section users can be added, which should be able to delete existing feedback information.

The Action, Mail and Info URL settings define the URL, which should be opened within the popup dialog box. The expressions within the curly brackets are replaced with the field of the source or destination list item.

When a suggestion webpart is displayed, following steps are performed:

1. get similar IDs via the web service,

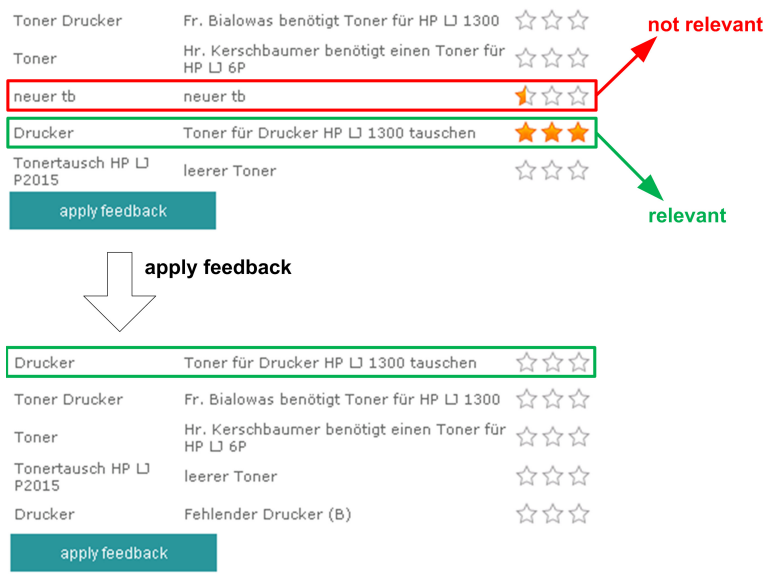

Figure 3.14: Suggestion Webpart With Explicit Feedback

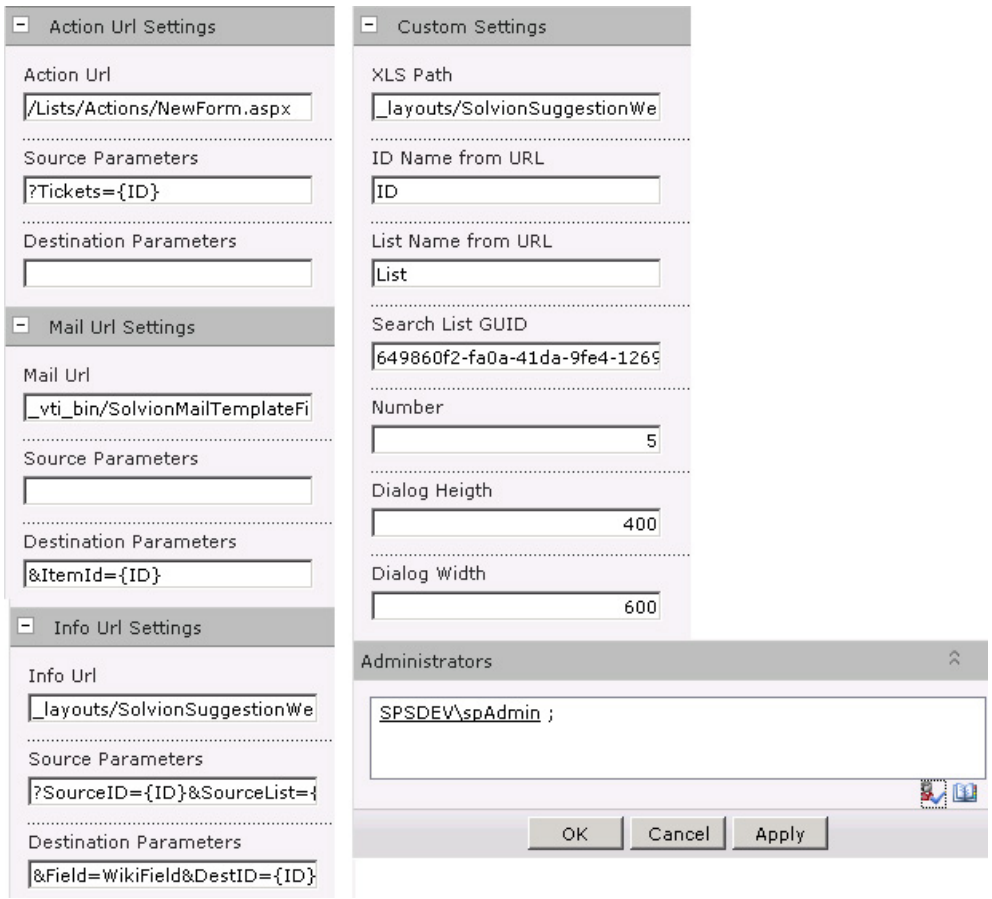

Figure 3.15: Suggestion Webpart Properties

2. get the data from the destination list as *DataTable*,
3. add columns, which are containing the IDs for controls with events, to the table,
4. get the *DataTable* as XML text,
5. transform the XML with XSLT style sheet, and
6. find controls and attach events

3.3.6 Classifier

The classifier is used to classify items with the help of the data saved in the infrastructure. Computational intensive calculations, which are required from this component, are handled by the trainer. The trainer saves for each classifier setting the Rocchio centroids in the class table.

The classifier is realised as list event receiver, which fires if an item is added or updated. With the help of a configuration page, see Figure 3.16, which is available through the configuration site of each list, following settings have to be made:

- Class list: the list contains the classes that should be classified
- Class field: the field in the class list where the different classes are saved
- List view: the list view which should be used to get the list items
- Input field: the field where the assigned classes are saved
- Output field: the field where the result of the classifier should be written to

The class, which is set from the user, will not be overwritten. If the output field change its value, a class will be assigned.

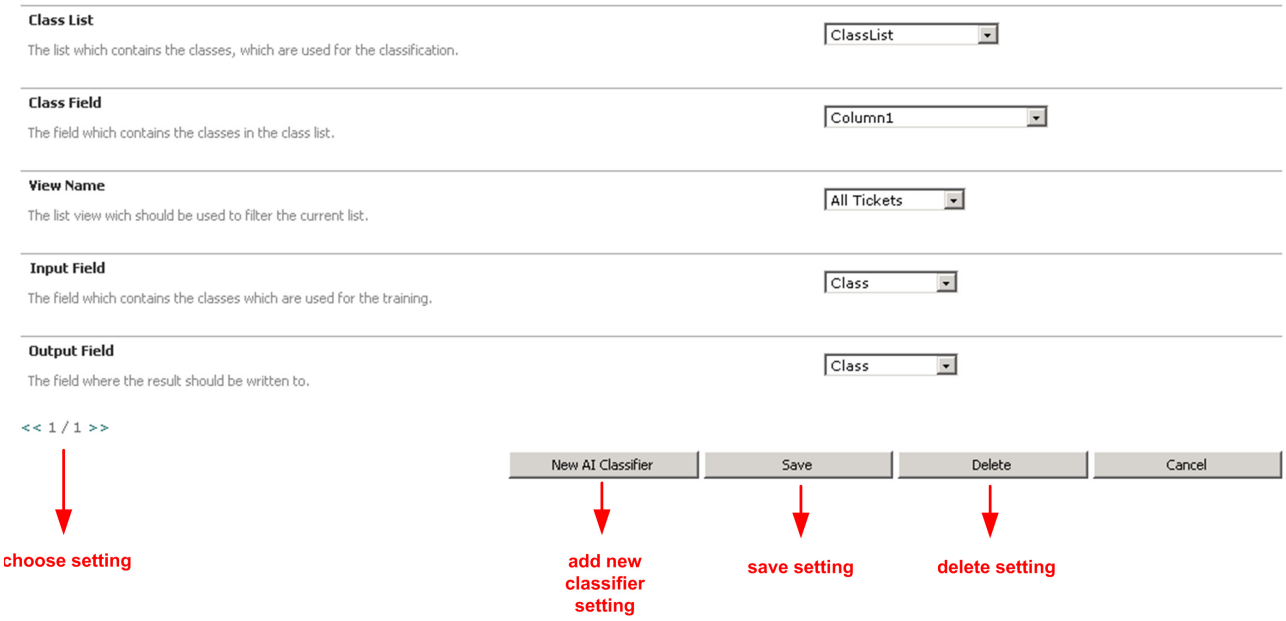

Figure 3.16: Classifier Settings

Chapter 4

Implementation

In this chapter the implementation is described and some details are mentioned.

The system is implemented with the programming language C# .Net as Visual Studio 2010 solution. Following projects are created:

SolvionKI: Contains the Business Logic.

KIInfrastucture: Feature that creates the SQL Infrastructure in the Microsoft SQL database.

KIContentProcessorFAQ: Content Processor for FAQ entries.

KIContentProcessor: Content Processor for the *SharePoint Customer Service Desk* entries

KIClassifier: Contains the event receiver that assigns the classes and classifier setting page.

KIClassifierServiceDesk: Implementation of an event receiver that is used to update the ticket list if an action is added or changed (Ticket and Action List).

KITrainer: Contains the trainer setting page and the timing job implementation.

KIWebservice: Webservice that communicates with the webpart.

SuggestionWebpart: Webpart that shows the search results.

KILogging: Feature that creates the Logging Infrastructure.

In Figure 4.1 the components and interactions, that are realised in the projects, are shown.

The implementation of the SolvionKI is divided into two parts, the DAL (data access layer) and the logic. The DAL is used to communicate with the Microsoft SQL database. In Figure 4.2 the class diagram of the DAL is shown. The *SQLDALManager* creates with the *SQLConnectionCreator* an appropriate connection and instantiates the classes, which are used to communicate with the individual tables. The *SQLInfrastructure* creates or deletes the required database and tables. Two different kinds of an *ISQLConnection* are implemented. The *SQLNormalConnection* is used to connect to the MS SQL server and

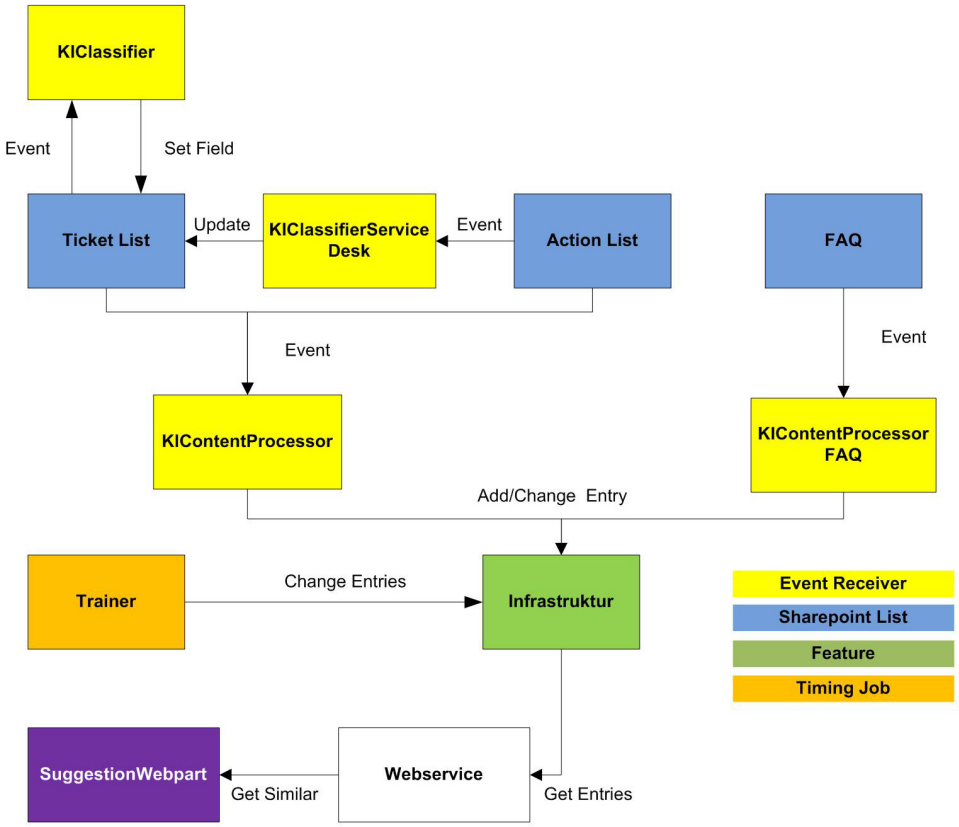

Figure 4.1: Implemented Components

to create the database, the schema and so on. The *SQLSchemaConnection* opens a normal connection and executes the SQL statements as *SKIUser* to alter the default schema.

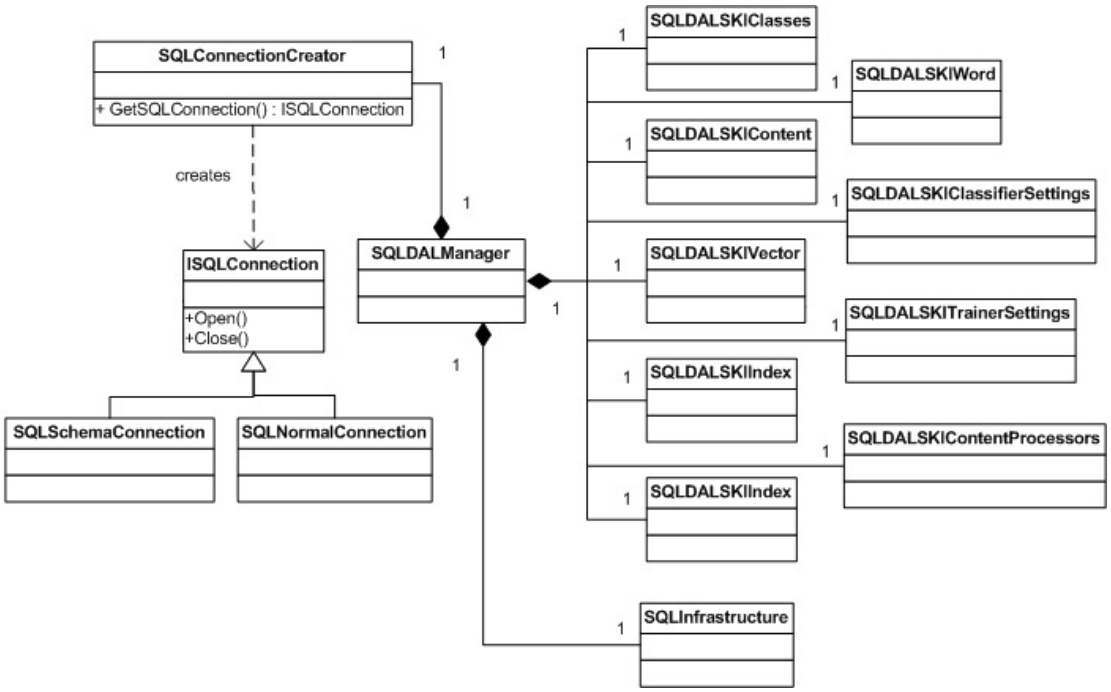

Figure 4.2: Class Diagram of the DAL

The class diagram of the information retrieval logic is shown in Figure 4.3. Via the *SQLDALManager* the logic communicates with the SQL server. The *Trainer* and the *TextProcessor* are using the *Weighting* class to build or update the feature vectors. The methods of the *Text* class, that filter the text and split it into words, are used from the *TextProcessor*. The *Similarity* and the *TextProcessor* are using the *Index* to achieve a higher performance. The *Feedback* class creates a new feature vector by a combination of other ones. To apply new global word counts the word count vector must be determined via the *Weighting* class.

4.1 Sparse Vector

Each item is represented as vector. These vectors are very sparse, that means only a few values are not equal to null. Only maximal the number of words in the text values have to be saved. In the vector space model the length of the vectors would be the amount of words n in the dictionary, see Equation 4.1.

$$\vec{vec}_{normal} = [v_1, v_2, v_3, ...v_n] \tag{4.1}$$

Because of that the saved vectors in the tables only contain values, which are not equal to null. For that, the indices and their corresponding values are saved.

$$\vec{vec}_{sparse} = (i_1, v_1)(i_2, v_2)... \tag{4.2}$$

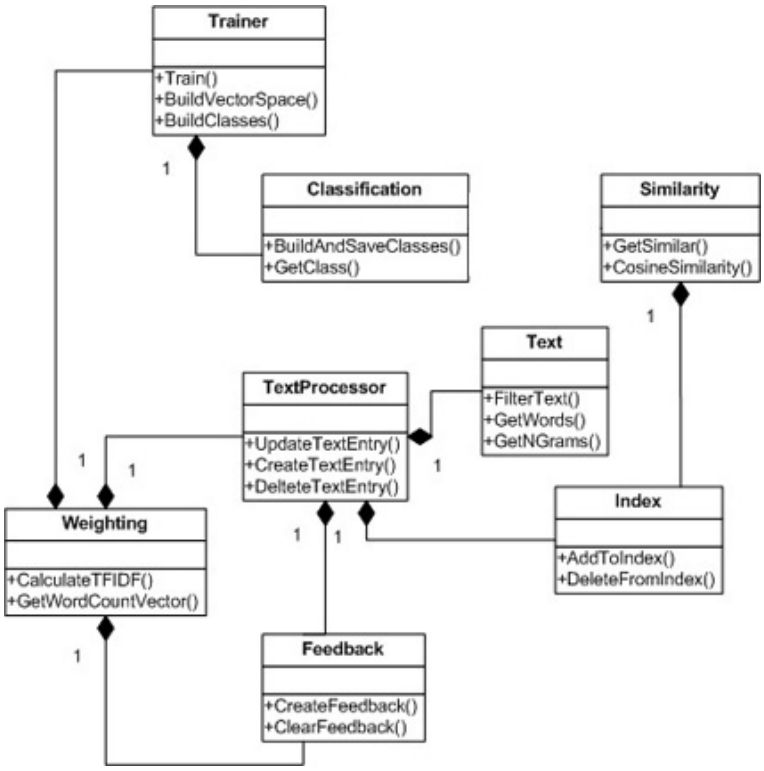

Figure 4.3: Class Diagram of the Logic

For this purpose a class was implemented which has following functions:

- List <int >GetAllIndizes()
- void SetValue(int index, double value)
- double GetValue(int index)
- void AddVector(SparseVector vec)
- void SubstractVector(SparseVector vec)
- double ScalarProduct(SparseVector vec)
- void Divide(double value)
- double GetDistance()

With the help of these functions the cosine similarity is computed:

```
public double CosineSimilarity(SparseVector vec1, SparseVector vec2)
{
    double scalar = vec1.ScalarProduct(vec2);
    double dist1 = vec1.GetDistance();
    double dist2 = vec2.GetDistance();
    return scalar / (dist1 * dist2);
}
```

4.2 Example

Following text entries are saved:

- Das ist ein Test.
- Das ist kein Test.
- Das ist ganz was anderes.

The content preprocessor saves the text items in the content table with a given ID. After that the words are saved into the dictionary and the word count and feature vectors are created. Also the text IDs are saved to the inverted index. After the first training phase the global word count is set and not equal to null.

word	global word count	id
das	3	1
ist	3	2
ein	1	3
test	2	4
kein	1	5
ganz	1	6
was	1	7
anderes	1	8

Table 4.1: Words Table

content	id
das ist ein test	1
das ist kein test	2
das ist ganz ganz was anderes	3

Table 4.2: Content Table

In the vector table you can see that the words *das* and *ist* are not occurring in the feature vector. This is because these words appear in all texts in the corpus. The feature weights are calculated via the Equation 4.3 and because of that the weight of these words is null.

id	word count vector	feature vector
1	(1,1)(2,1)(3,1)(4,1)	(3,1.5850)(4,0.5850)
2	(1,1)(2,1)(5,1)(4,1)	(5,1.5850)(4,0.5850)
3	(1,1)(2,1)(6,2)(7,1)(8,1)	(6,3.1699)(7,1.5850)(8,1.5850)

Table 4.3: Vector Table

$$w_i = termFrequency_i * log_2(\frac{documentCount}{globalCount_i}) \tag{4.3}$$

The similarities between these vectors are shown in Table 4.4. There you can see that doc1 and the doc2 are similar, but doc1 and doc2 to doc3 not.

	doc 1	doc 2	doc 3
doc 1	1.0	**0.1199**	0
doc 2	**0.1199**	1.0	0
doc 3	0	0	1.0

Table 4.4: Similarity Matrix

4.3 Installation of the Implemented System

The information retrieval system is installed, integrated within the *SharePoint Customer Service Desk* and developed within a Sun Virtualbox image. This image is a Microsoft Server 2008 and the SharePoint 2010 installation. Visual Studio 2010 is the used developing environment. Following steps have to be executed to install the information retrieval system:

- Deploy wsp packages with the help of the console command *stsadm* or with the Visual Studio 2010.
- Activate Features in the correct order on the different sites (scopes).
- Place the Suggestion Webpart on an appropriate page.
- Set Configurations (Trainer, Webpart, Trainer)

The features should be activated in following order:

- Logging Feature (enables logging infrastructure)
- Infrastructure Password Generator Feature (creates password for the SQL database)
- Infrastructure Feature (creates user, database and tables)
- Trainer Feature (installs trainer timer job and configuration page)
- Content Processors Feature

- Classifier Feature (installs the event receivers and configuration pages for lists with a content processor)
- Service Desk Classifier Feature (adds an event receivers, that updates the corresponding ticket list)
- Suggestion Webservice Feature (installs suggestion webservice)
- Suggestion Webpart Feature (adds suggestion webpart)

Chapter 5

Evaluation Results

In this chapter the results of the evaluation of the different kinds of algorithms for the similarity measurement and the classification are shown.

For the evaluation three different datasets are used, see Table 5.1.

	ticket count	mean word count	div word count	dictionary size	categories count
a	255	11.5961	4.4047	1201	8
b	250	11.608	4.299	1180	8
c	1370	10.4146	4.0137	3439	25

Table 5.1: Evaluation Datasets

The categories of the datasets *a* and *b* are assigned by hand. The dataset *c* uses categories from an existing service desk application of the company *Solvion*.

5.1 Similarity Measurement Evaluation

Different kinds of text preparation methods, weighting schemas and algorithms are evaluated. The MAP value is calculated to compare the results. The result list contains the top five ranked items.

5.1.1 Text Preparation

Different text preparation methods are evaluated. *Hunspell*, a spell checking tool, is used to make spelling correction, find stems and synonyms. Also the calculation of *N- Grams* is evaluated.

Hunspell

Hunspell is used to add stems, synonyms and spelling suggestion to the text to perform a kind of query expansion. It is an open source toolbox, which is also used from Open Office [PL10]. To evaluate these methods the weighting scheme TF–IDF and the cosine similarity are used. The result list contains the five most similar items. The MAP value of that list is calculated and shown in table 5.2. The dataset ***a***, see Table 5.1, is used.

	MAP
Spelling Correction	0.46617
Stems	0.47551
Synonyms	0.46743
Spelling Correction and Synonyms	0.46759
Spelling Correction and Stems	0.46659
Spelling Correction, Synonyms and Stems	0.46723
without	0.62248

Table 5.2: Hunspell Evaluation

N-Grams

For the N-Grams evaluation only one weighting scheme is used, the TF–IDF. Also only one similarity measurement (cosine) is used. N is a number from one to seven. The MAP values for the dataset ***a*** are calculated. In Table 5.13 and 5.4 the sizes of the dictionaries and the MAP values for the best five results are shown.

N	Dictionary Size	MAP
1	60	0.57156
2	1143	0.61853
3	6278	0.61539
4	16483	0.61052
5	27916	0.60661
6	38271	0.60324
7	46774	0.60039
words	3439	0.6088

Table 5.3: N- Grams Evaluation for the dataset ***c***

N	Dictionary Size	MAP
1	59	0.57681
2	805	0.64979
3	3461	0.64972
4	7005	0.63873
5	9938	0.62602
6	12113	0.62007
7	13647	0.61284
words	1201	0.62248

Table 5.4: N- Grams Evaluation for the dataset ***a***

5.1.2 Weighting Schemas

To evaluate the weighting schemes following similarity algorithms are used:

- Cosine

- Tanimoto
- Euclidean
- Pseudo-cosine
- Dice
- Overlap

The dataset ***c*** is used as input. The mean value of the MAP value is calculated for each weighting schema over all similarity measurements. In Table 5.5 the evaluation results are shown.

rank	global weighting	local weighting	MAP
1.	a	g	0,6166
2.	t	g	0,61603
3.	l	g	0,61581
4.	n	g	0,61561
5.	l	x	0,60818
6.	a	x	0,6076
7.	t	x	0,60727
8.	n	x	0,60645
9.	b	g	0,60120
10.	b	x	0,60120
11.	t	e	0,59431
12.	a	e	0,5943
13.	l	e	0,59418
14.	t	f	0,59404

Table 5.5: Weighting Evaluation

In Table 5.6 the mean MAP values of the different local weightings are shown.

local weighting	MAP
x	0,606
f	0,592
g	0,612
n	0,572
p	0,559
e	0,592

Table 5.6: Weighting Evaluation of Local Weighting Schemas

In Table 5.7 the mean MAP values of the different global weightings are shown.

5.1.3 Dimension Reduction

For the dimension reduction evaluation the weighting schema TF–IDF and the cosine similarity are used. The dimension reduction algorithms PCA, LDA and SVD are evaluated.

global weighting	MAP
t	0,5906
b	0,5848
l	0,5909
n	0,5894
a	0,5907

Table 5.7: Weighting Evaluation MAP of Global Weighting Schemas

In Table 5.8 the MAP values for different dimensions and algorithms for the dataset *a* are shown. In Figure 5.1 the MAP values for the different dimensions are plotted.

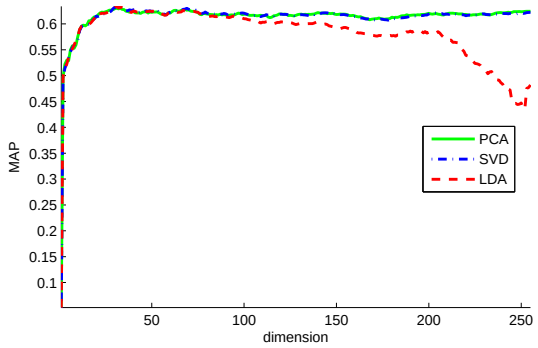

Figure 5.1: Dimension Reduction Graph

Method	Dimension	MAP
LDA	31	0.6340
PCA	31	0.6302
SVD	34	0.6324
	1201	0.6225

Table 5.8: Dimension Reduction Evaluation of Cosine Similarity

In Figure 5.2 the behaviours of the different similarity algorithms are shown. The similarity between two vectors is computed. In *Measurement 1* the distance and in *Measurement 2* the angle of the compared vector grow linearly. In *Measurement 3* both, angle and distance, are increasing linearly. It is shown that all algorithms change their value if the angle between the compared vectors is changing. The overlap and cosine algorithm behaves the same. If only the distance is modified the cosine, pseudo– cosine and overlap algorithm are constantly equal to 1.0. By changing the distance and angle linearly the pseudo– cosine and overlap algorithm compute the same results. The cosine and pseudo–cosine algorithm yield the same functions if only the angle or if the angle and distance are changing.

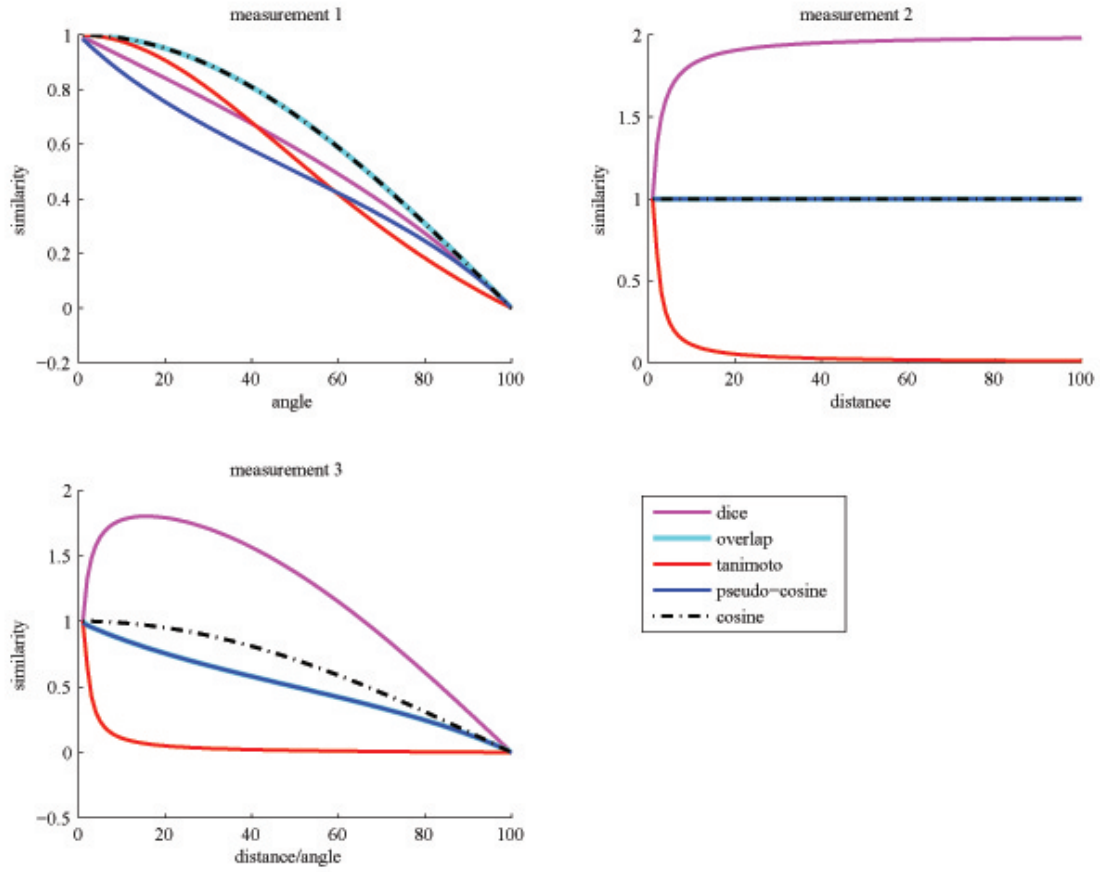

Figure 5.2: Similarity Algorithm Comparison

5.1.4 Similarity Algorithms

In Table 5.9 the MAP results of the different similarity measurements for the set ***c*** are shown. *Microsoft SharePoint Search 17.3* means that the MAP value for the result list, which contains about 17.3 items, is calculated. For the *Microsoft SharePoint Search 5* only the top five result items are used. In Table 5.9 the mean MAP values of all different weighting schemas for each algorithm are shown.

algorithm	MAP
cosine	0,60266
tanimoto	0,60237
euclidean	0,54300
pseudo-cosine	0,59265
dice	0,60087
overlap	0,59419
Microsoft SharePoint Search 17.3	0.5668
Microsoft SharePoint Search 5	0.6125

Table 5.9: Similarity Algorithms Evaluation

In Table 5.10 the best ten results of the different similarity algorithms and weighting schemas are shown.

rank	global weighting	local weighting	algorithm	MAP
1.	a	g	cosine	0,62756
2.	l	g	cosine	0,62711
3.	t	g	cosine	0,62653
4.	n	g	tanimoto	0,62638
5.	n	g	dice	0,62631
6.	n	g	cosine	0,62610
7.	l	g	tanimoto	0,62604
8.	a	g	tanimoto	0,62525
9.	t	g	tanimoto	0,62464
10.	a	x	cosine	0,62393

Table 5.10: Top Similarity Evaluation Results

5.1.5 Summary

In Table 5.11 the results of the different evaluations are shown. The best resulting weighting schema is the a g. In the combination with the cosine similarity it produces the best results for the dataset c and achieves a better MAP value than the Microsoft Sharepoint Search. Different types of algorithms are evaluated with the help of the TF–IDF weighting schema and a cosine similarity. The 2-Grams method obtained the best result of this combination. By the evaluation of the dataset c it also achieves better results than the Microsoft Sharepoint Search result.

Dataset	algorithm	MAP
c	2–Grams, TF–IDF, cosine	0.61853
c	a g cosine	0.62756
c	a g weighting schema mean	0.6166
c	Microsoft Sharepoing Search 5	0.6125
a	Spelling Correction and Synonyms, TF–IDF, cosine	0.46759
a	TF–IDF , cosine	0.62248
a	2–Grams, TF–IDF, cosine	0.64979
a	LDA, TD–IDF, cosine	0.6340

Table 5.11: Top Similarity Evaluation Results

5.2 Classification Results

N- Grams, different weighting schemas and classification algorithms are evaluated. The percentage of the correct classified items of a 1-n cross validation is used to compare the results.

5.2.1 N-Grams

N- Grams are also evaluated for the classification. The used algorithms are the KNN– and the Rocchio Cosine. The t p weighting scheme is used to calculate the vector space model. In Table 5.12 the results of the evaluation for N one to seven are shown. For the KNN algorithm the best value of k is dynamically determined.

		KNN Cosine		Rocchio Cosine
N	Dictionary Size	k	correct	correct
1	59	24	0,392	0,044
2	803	16	0,580	0,564
3	3434	8	0,620	0,640
4	6921	7	0,628	0,640
5	9789	9	0,584	0,624
6	11911	13	0,600	0,608
7	13402	32	0,616	0,572

Table 5.12: N-Grams Classification Results of Dataset ***b***

5.2.2 Weighting Schemas

To evaluate the different weighting schemas the mean correct value over the following algorithms is calculated:

- Bayes
- 10NN (dice, cosine, tanimoto, overlap)
- Rocchio (dice, cosine, tanimoto, overlap)

The dataset ***c*** is used. In Table 5.13 the top 10 and the worst result of the mean correct values are shown.

rank	local weight	global weight	correct
1.	a	g	0,47502
2.	t	g	0,47437
3.	a	f	0,46740
4.	a	e	0,46691
5.	t	f	0,46691
6.	l	g	0,47072
7.	l	f	0,46286
8.	l	e	0,46204
9.	a	x	0,46569
10.	l	x	0,46423
30.	b	p	0,40227

Table 5.13: Best Classification Weighting Schemas Results

In Table 5.15 the mean correct values for the local and in Table 5.14 for the global weighting schemas are shown.

global weighting	correct
t	0,45311
b	0,43889
l	0,45077
a	0,45362
n	0,44611

Table 5.14: Global Weighting Evaluation of the Classification

local weighting	correct
x	0,45992
f	0,46107
g	0,46621
n	0,43366
p	0,40928
e	0,46086

Table 5.15: Local Weighting Evaluation of the Classification

5.2.3 Dimension Reduction

In Figure 5.3 the different dimension reduction methods are plotted and in Table 5.16 the best results are shown. The Rocchio Cosine classification algorithm, the t p weighting schema and the dataset ***b*** are used for the evaluation.

Method	Dimension	correct
LDA	77	0.5320
PCA	112	0.3840
SVD	129	0.6080
	1201	0.6080

Table 5.16: Dimension Reduction Evaluation of Rocchio Cosine Classification

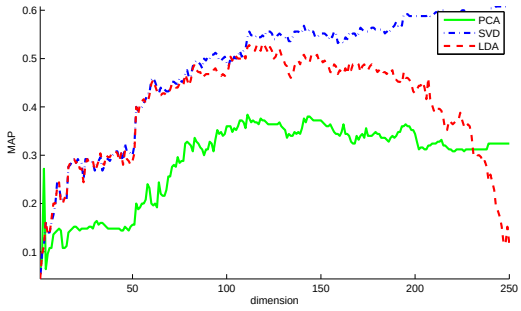

Figure 5.3: Dimension Reduction Graph of Rocchio Classification

5.2.4 Algorithms

In Table 5.17 the mean correct values of the different algorithms of all weighting schemas are shown of the dataset c. In Table 5.18 the best ten correct classified values and the

algorithm	correct
Bayes	0,00219
10 NN dice	0,58759
10 NN cosine	0,58774
10 NN tanimoto	0,58681
10 NN overlap	0,58122
Rocchio dice	0,34010
Rocchio cosine	0,58397
Rocchio tanimoto	0,43092
Rocchio overlap	0,33297

Table 5.17: Classification Algorithms Results

result of the rocchio cosine with a TF– IDF weighting schema are shown of the dataset c.

Support Vector Machines

The evaluation of the support vector machines is very time expensive. The used data set is ***b*** and weighting schema is t p. The vector space computation of the TMG toolbox is used. The data are normalised before. That means that a uniform distribution of the assigned classes is aimed. The mean count of the items per class is calculated. This mean value defines the max items count of each class, the remaining items are deleted. In Table 5.19 the results of the SVM and of the other classification methods are shown. The sigma for

rank	local weighting	global weighting	algorithm	correct
1.	t	g	10 NN cosine	0,64745
2.	a	g	10 NN cosine	0,64599
3.	a	g	10 NN tanimoto	0,64161
4.	l	g	10 NN cosine	0,64161
5.	l	g	10 NN tanimoto	,63942
6.	t	g	10 NN tanimoto	0,63796
7.	l	g	10 NN dice	0,63723
8.	a	x	10 NN cosine	0,63577
9.	a	x	10 NN tanimoto	0,63504
10.	t	x	10 NN cosine	0,63358
31.	a	e	Rocchio cosine	0,62190
41.	t	f	Rocchio cosine	0,61752

Table 5.18: Best Classification Results

the rbf kernel is set between 0,5 and 0,9 with a step of 0,1. The order of the polynomial kernel is set between one to five.

5.2.5 Summary

In Table 5.20 the best evaluation results are shown. The SVM classification algorithmn achieved bad results on the dataset b. The best results are obtained by 4–Grams with and a Rocchio Consine classification. The best weighting schema is also the a g, same as in the similarity evaluation, see Table 5.5.

5.3 Summary

The results fall off in quality by applying different text prepossessing methods with Hunspell. Contrary to that the N–Grams calculation with N = 2 improved the similarity calculation results. An advantage of this approach is also that the dictionary size is smaller. The best performing self implemented similarity algorithm is the cosine similarity. The Microsoft SharePoint Search achieves better results than the cosine similarity but are interior to the N–Grams results.

The a g (alternate log – GfIdf) weighting schema established the best results for the classification and similarity evaluation. In Figure 5.4 the TF–IDF and a g weighting calculations is shown. The alternate log weakens high local term frequencies. The GfIdf is high if a word occurs often in few documents. If a word occurs often overall and is contained in almost documents, the GfIdf is small (for instance stopwords). The IDF does not consider the global word frequencies. That means that words are equally important if they occur in few documents, even if one of them is contained more often.

5.4 Time Evaluation

Following attributes are influencing the execution times:

Algorithm	correct
Bayes	0,206
20NN dice	0,578
20NN cosine	0,561
20NN dice	0,578
20NN tanimoto	0,583
20NN overlap	0,544
Rocchio dice	0,594
Rocchio cosine	0,628
Rocchio dice	0,594
Rocchio tanimoto	0,600
Rocchio overlap	0,550
pairwise SVM linear	0,478
pairwise SVM Polynomial order = 1	0,478
pairwise SVM quadratic	0,328
oneAgainstAll SVM Polynomial order = 1	0,233
oneAgainstAll SVM linear	0,233
oneAgainstAll SVM quadratic	0,056
pairwise SVM Rbf sigma = 0.5	0,500
oneAgainstAll SVM Rbf sigma = 0.5	0,500

Table 5.19: SVM Classification Results

dataset	algorithm	correct
b	4–Grams, t p weighting schema, KNN Cosine	0.628
b	4– and 5–Grams, t p weighting schema, Rocchio Cosine	0.640
b	t p weigthing schema, Rocchio Cosine	0.6080
b	SVD, t p weigthing schema, Rocchio Cosine	0.6808
b	t p weigthing schema, TMG toolbox, Rocchio Cosine	0.628
b	t p weigthing schema, TMG toolbox, SVM Rbf sigma = 0.5	0.5
c	a g weighting schema mean	0.47502
c	t g weigthing schema, 10NN cosine	0.64745

Table 5.20: Dimension Reduction Evaluation of Rocchio Cosine Classification

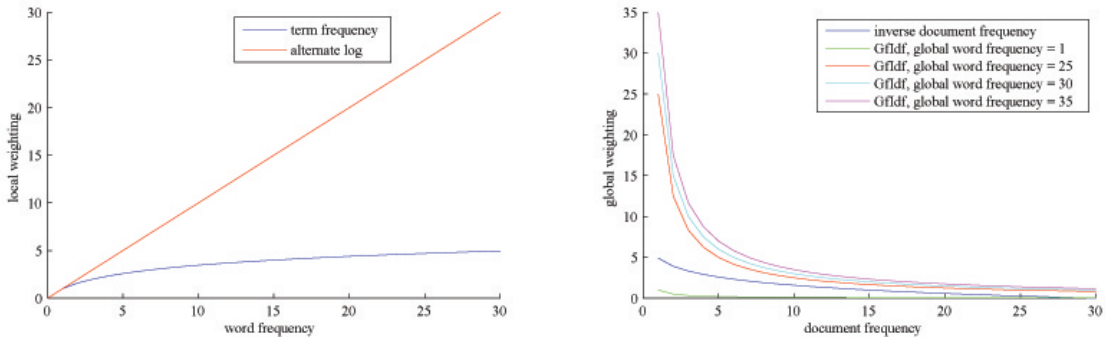

Figure 5.4: Weighting Calculation Comparison

- Eord count (n)
- Ticket count (m)
- Class count (k)
- Dictionary size (l)
- Feedback vector counts (p)

To add or update a text entry, following steps are executed:

- Delete existing entry
- Create new content entry
- Create vector entry

Delete existing entries

If a ticket entry has to be deleted, at first the entries of the feedback and vector table are determined. After that the corresponding inverted index entries are removed. Then the content, feedback vector and normal vector entries are deleted. Following SQL executions and time complexities (worst case) occur:

1. Select WHERE ... (Feedback table) $O(p)$
2. Delete from INDEX
3. Select WHERE ... (Vector table) $O(m)$
4. Delete from INDEX
5. Delete FROM feedback table WHERE ... $O(p)$
6. Delete FROM content table WHERE ... $O(m)$
7. Delete FROM vector table WHERE ... $O(m)$

The *delete from INDEX* contains for each word of the text entry following operations:

```
for each word of the text item
ids = SELECT WHERE ... (Index table)
        if {contains own id}
                remove own id from ids
                if {ids count == 0}
                        DELETE from index table
                else
                        UPDATE index
                end
        end
end
```

The SELECT and DELETE operations within the index table for a specific word need in the worst case $O(l)$. Because this has to be done for each word that occurs in the ticket, the complexity of deleting a vector from the index needs $O(l * n)$. Because of that the time complexity to delete a text entry from the database needs $O(p + m + l * n)$. p maximal can be equal to m and so the complexity is $O(m + l * n)$.

Create new content entry

Before a text entry is saved the text is filtered. At first a regular expression is used to remove HTML syntax. To filter signs, the text is spitted to words with the help of given signs like !?() and concatenated again ($O(n)$). After that the system tries to update the text entry ($O(m)$). If the entry does not exist($O(m)$), a new one is created. So the overall time complexity of creating a new content entry is $O(m + n)$.

Create vector entry

For each word of the the text item following steps are executed:

1. SELECT COUNT where = word (search if word exists) $O(l)$
2. INSERT word $O(1)$ (if it not exist)
3. SELECT word $O(l)$
4. Weighting.getWordCount() (calculate TF) $O(n)$
5. Set word count vector $O(1)$
6. Set global word count vector $O(1)$
7. Add entry to Index Table $O(l)$

After that the global and local word count vectors are calculated. These two vectors are used to determine the feature vector:

1. Weighting.calculateTFIDF $O(n)$
2. UPDATE vector $O(m)$
3. INSERT vector $O(1)$

Weighting.calculateTFIDF iterates the global and local word count vectors and calculates the TF– IDF value.

To add an entry to an index following steps are executed:

1. SELECT Index Table where $O(l)$
2. Check if item is already added $O(l)$
3. If the entry not exists, add it (UPDATE Index Table) $O(l)$

The time complexity of adding one entry to the index is $O(l)$. So the overall time complexity is $O((l + n) * n + n + m)$.

5.4.1 Search

The search has to select the feature vector of the query, that was saved before. If feedback information is variable it is possible to select it from the vector or feedback table. After that the inverted index is used to get item IDs for the comparison. Therefore each word of the source text is selected and the inverted index IDs are collected and the distinct ones are returned. For each of these IDs the corresponding feature vector is selected from the database. If a feedback vector is available, it is selected, otherwise the normal vector is chosen. For each of these vectors the cosine similarity with the source vector is calculated and saved in a list. At last this list is sorted and the top n IDs are returned. So following steps are executed:

1. Get feedback or normal vector $O(p + m)$
2. Get IDs form inverted index $O(n + m)$
3. For each id (max n):
   (a) Get feedback or normal vector for id $O(p + m)$
   (b) Calculate cosine similarity $O(l)$
4. Sort similarity list $O(m * log(m))$
5. Return a amount of the most important ids

To calculate the cosine similarity of two vectors at first the scalar product has to be calculated. This value is divided by the product of the distances of the vectors. To calculate the scalar product and the distances the overall time complexity is $O(l)$.

The hole complexity is $O(n * (p + m + l) + m * log(m))$.

5.4.2 Classification

To get the to assigning class, at first, the Rocchio centroids are selected from the database. In the worst case $O(kAll)$, if $kAll$ is the amount of all classes saved in the table, is needed. After that for each class the cosine similarity ($O(l)$) is calculated. The ID of the highest most similar class is saved ($O(k * l)$).

Chapter 6

Conclusion

A system to evaluate different information retrieval algorithms was implemented in Matlab. Also a solution for the *SharePoint Customer Service Desk* to retrieve similar problems respectively their answers and to assign classes to a ticket was created.

The implemented *SharePoint Customer Service Desk* system is adaptive because of the feedback mechanism and the training for the classification. An implicit and explicit feedback mechanism is implemented. The implicit feedback has the advantage, that the users have no additional effort.

The system can be used for different kind of Microsoft SharePoint lists. Therefore only a new content processor has to be created. The presentation of the search results can be changed via the XSLT file and the configuration settings. If a completely other representation should be implemented it is also possible because the communication with the information retrieval system can be established via the webservice. The classification can also be applied on different lists by setting the appropriate classier settings.

The TF–IDF algorithm is implemented because it is common. The a g (Alternate log–GfIdf) weighting achieved the best evaluation results, see 5.5 and 5.10.

The Rocchio cosine algorithm was chosen for the classification, it achieves good results and is more time efficient than the NN neighbour approach. For the nearest neighbour algorithm the distance to each item has to be calculated. The Rocchio classifier only requires a computation of the distances to the class centroids. This can be computed during the training phase by iterating the list once. A possible disadvantage of the classification is that data have to be available for the training. If a new class is created and no training data are available for this class, this class will not be assigned to incoming tickets. Also as more items are assigned to a certain class correctly, as better this one is described and a better classification is possible. Also no similar text items for a new one can be found, if it only includes new words. That means that no intersection with other documents and

so no similarity measurement can be done.

The cosine similarity is used to search similar text items. This approach achieved the best test results and a second advantage is, that the same computation is also used for the Rocchio classification.

The text preparation mechanism N–Grams is also implemented. This mechanism achieved the overall best results and has also the advantage to control the size of the dictionary. With the help of a factory pattern it is possible to use the word based or N–Grams mechanism within the Content Processor. The N–Grams overcome also the well known problem of the similarity measurement, the synonymy, polysemy and inflection of words.

No dimension reduction mechanism is implemented because a computation of a SVD is time consuming and the results N–Grams results are better.

The efficiency of the DAL can be improved in future. It turned out, that SQL connection and queries to the MS SQL database, used in this implementation, did not work highly efficient. With the help of stored procedures the communication can be improved. The TF–IDF weighting schema is used to calculate the vector space model. To improve the results the a g weighting schema should be used.

Appendix A

List of Abbreviations

SMART System for the Mechanical Analysis and Retrieval of Text

TF–IDF Term Frequency – Inverse Document Frequency

Corpus Document Collection

Document frequency Count of Terms in a Document

MAP Mean Average Precision

TP True Positives

FN False Negatives

p@k Precision at k

SVM Support Vector Machines

SVD Single Vector Decomposition

PCA Prinicipal Compontent Analysis

KNN k Nearest Neighbour

GUID Globally Unique Identifier

Bibliography

[Ass07] Association of Computational Linguistics. *Towards an Iterative Reinforcement Approach for Simultaneous Document Summarization and Keyword Extraction*, 2007.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 17:734–749, 2005.

[BDH03] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web Search for a Planet: The Google Cluster Architecture. 2003.

[bDK05] Kai bo Duan and S. Sathiya Keerthi. Which is the best multiclass SVM method? An empirical study. In *Proceedings of the Sixth International Workshop on Multiple Classifier Systems*, pages 278–285, 2005.

[Bur02] Robin Burke. Hybrid Recommender Systems: Survey and Experiments. In *User Modeling and User-Adapted Interaction*, pages 331 – 370, California State University, Fullerton, USA 92834, 2002. Department of Information Systems and Decision Sciences, Kluwer Academic Publishers.

[CZR05] Nick Craswell, Hugo Zaragoza, and Stephen Robertson. Microsoft Cambridge at TREC-14: Enterprise Track. 2005.

[Dep] Department of Research and Development NACSIS. *Evaluation of keyword extraction task*. NTCIR Workshop TMREC Group.

[FHM08] Ingo Feinerer, Kurt Hornik, and David Meyer. Text Mining Infrastructure in R. *Journal of Statistical software*, March 2008.

[GLWW00] Christoph Goller, J. Löning, T. Will, and W. Wolff. Automatic Document Classification - A thorough Evaluation of various Methods. In *ISI*, pages 145 – 162, 2000.

[Got09] Dr. Thomas Gottron. Information Retrieval, 2009.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nedellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142, Heidelberg et al., 1998. Springer.

[Kür06] Jens Kürsten. Systematisierung und Evaluierung von Clustering-Verfahren im Information Retrieval. 2006.

[LJ98] Y. H. Li and A. K. Jain. Classification of text documents. *The Computer Journal*, 41:537 – 546, 1998.

[LMV08] Kevin Laahs, Emer McKenna, and Veli-Matti Vanamo. *Microsoft® SharePoint 2007 Technologies Planning, Design and Implementation.* Digital Press is an Imprint of Elsevier, 2008.

[LSY03] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[MB10] Marty Matthews and Nancy Buchanan. *Microsoft SharePoint 2010 QuickSteps.* cGraw-Hill Osborne Media, 2010.

[MI03] Y. Matsuo and M. Ishizuka. Keyword Extraction from a single document using Word Co-Occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1 (2004) 157169), 2003.

[MMC02] P Majumder, M Mitra, and B. B. Chaudhuri. N-gram: a language independent approach to IR and NLP, 2002.

[Mol09] Cleve Moler. *Experiments with MATLAB.* 2009.

[MS06] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of MLnetECML2000 Workshop (2000)*, volume 4203/2006, pages 312 – 321. Citeseer, 2006.

[MSD09] Relevance in SharePoint Search. http://blogs.msdn.com/b/kundut/archive/2009/10/15/relevance-in-sharepoint-search.aspx 14.09.2010, 2009.

[MW06] Philipp Mayr and Anne-Kathrin Walter. Abdeckung und Aktualität des Suchdienstes Google Scholar. . *Information - Wissenschaft & Praxis*, 57(3):133–136, 2006.

[oel09] *Automatic Keyword Extraction for Database Search.* Association for Computational Linguistics, 2009.

[PB07] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web: methods and strategies of web personalization*, pages 325 – 341. Springer-Verlag, 2007.

[PL10] Tommi Pirinen and Krister Lindén. Creating and Weighting Hunspell Dictionaries as Finite-State Automata. *Investigationes Linguisticae*, 21:1–16, 2010.

[PWWB09] Rhonda D. Phillipsa, Layne T. Watsona, Randolph H. Wynnec, and Christine E. Blinnc. Feature reduction using a singular value decomposition for the iterative guided spectral class rejection hybrid classifier. In *ISPRS Journal of Photogrammetry and Remote Sensing*, pages 107 – 116, 2009.

[R.86] Quinlan J. R. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.

[RAS+10] Tom Rizzo, Reza Alirezaei, Paul J. Swider, Scot Hillier, Jeff Fried, and Kenneth Schaefer. *Professional SharePoint® 2010 Development*. Wiley Publishing, 2010.

[Rij79] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

[RV97] Paul Resnick and Hal R. Varian. Recommender systems. In *Communications of the ACM*, volume 40, pages 56 – 58, 1997.

[SB90] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41:288–297, 1990.

[Seb05] Fabrizio Sebastiani. Text categorization. In *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*, pages 109–129. WIT Press, 2005.

[Sin01] Amit Singhal. Modern information retrieval: a brief overview. *BULLETIN OF THE IEEE COMPUTER SOCIETY TECHNICAL COMMITTEE ON DATA ENGINEERING*, 24:2001, 2001.

[SL98] Brin Sergey and Page Lawrence. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, pages 107–117. Elsevier Science Publishers B. V., 1998.

[Smi02] Lindsay I Smith. A Tutorial on Principal Component Analysis, February 2002.

[TH01] Loren Terveen and Will Hill. Beyond Recommender Systems: Helping People Help Each Other. In *HCI in the New Millennium*, pages 487 – 509. Addison-Wesley, 2001.

[vdPPRG04] Lonneke van der Plas, Vincenzo Pallotta, Martin Rajman, and Hatem Ghorbel. Automatic Keyword Extraction from Spoken Text. A Comparison of two Lexical Resources: the EDR and WordNet. *Procedings of the LREC 2004 international conference*, pages 2205 – 2208, 2004.

[ZG05] Dimitrios Zeimpekis and Efstratios Gallopoulos. TMG: A MATLAB Toolbox for Generating Term-Document Matrices from Text Collections, 2005.