Karlsruhe Institute of Technology

Methods for Automatic Keyword Extraction on Transcribed Texts: A Comparison

Diploma Thesis at the
Institute for Anthropomatics and Robotics
Interactive Systems Labs (ISL)
Prof. Dr. Alex Waibel
Faculty for Computer Science
Karlsruhe Institute of Technology

of

cand. inform.
Miriam Hundemer

Advisor:
Prof. Dr. Alex Waibel
Dipl.-Inform. Kay Rottmann

Date of Registration: 2014-01-01
Date of Submission: 2014-06-30

I declare that I have developed and written the enclosed Diploma Thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 30.06.2014

Zusammenfassung

Das Thema dieser Diplomarbeit ist der Vergleich verschiedener Methoden zur automatischen Extraktion von Schlüsselwörtern basierend auf transkribierten Texten aus der TED[1] Datenbank. Zu diesem Zweck wurde zunächst ein eigener Ansatz, basierend auf der Kombination verschiedener Eigenschaften potenzieller Schlüsselwortkandidaten mittels eines Log-linearen Modells entwickelt. Dieser wurde anschließend in einem System zur Extraktion von Schlüsselworten implementiert. Das System wurde mit englischer Sprache getestet und entwickelt, ließe sich jedoch nach einigen Anpassungen auch für andere Sprachen einsetzen.
Um eine Entscheidung darüber zu treffen, ob ein Schlüsselwortkandidat ein gutes oder schlechtes Schlüsselwort für den gegebenen Text darstellt, kommen verschiedene Eigenschaften vorheriger Schlüsselwörter oder N-Gramme des aktuell betrachteten Dokuments zum Einsatz. Einige dieser Eigenschaften sind die Position des ersten Auftretens eines Schlüsselwortes im Dokument, *Part of Speech Tags*, die *inverse Dokumentenhäufigkeit IDF*, die bekannte Metrik *Term Frequency-Inverse Document Frequency (TF-IDF)* soweit das Abstandsmaß χ^2. Als Parameter Tuning Algorithmen kommen für das entwickelte System der *Simplex* Algorithmus nach John Nelder und Roger Mead und der *Powell Search* Algorithmus zum Einsatz. Diese beiden Ansätze werden mit den Ergebnissen einer *Support Vector Machine (SVM)* verglichen.
In der Arbeit werden zunächst wichtige Vorverarbeitungsschritte beschrieben, die die zu verarbeitenden Daten in eine geeignete Form für das Extrahierungssystem bringen. Anschliessend beschreibe ich detailliert, wie während eines Trainingsprozesses Informationen über alle wichtigen Eigenschaften von Schlüsselwörtern vom System gesammelt werden. Weiterhin beschreibe ich die Klassifikation und Extraktion von Schlüsselwörtern mit Hilfe der gewählten Ansätze.
Im Auswertungskapitel dieser Arbeit beschäftige ich mich zunächst mit potentiellen Auswirkungen verschiedener POS Schwellwerte, dem Finden einer guten Metrik für die Dokumentenposition sowie unterschiedlichen Pruning Ansätzen zum vorfiltern der Schlüsselwortkandidaten. Anschliessend werden die Auswirkungen verschiedener Eigenschaftskombinationen, verschiedener Schwellwerte sowie der unterschiedlichen Pruning Ansätze auf die verschiedenen Ansätze untersucht. Abschließend werden Ergebnisse der Experimente diskutiert und ein Ausblick auf mögliche zukünftige Verbesserungen und Erweiterungen gegeben.

[1] http://www.ted.com/talks

Abstract

The subject of this thesis is the comparison of different methods for automatic keyword extraction on transcribed texts from the TED[2] database. For this purpose a keyword extraction system based on different features was developed and implemented. The system uses a log linear model to combine the different features and classify potential keyword candidates. It was tested and developed on English texts but is easily adaptable to other languages.
In order to discriminate between good and bad keyword candidates for the current document different features are used that rely on both the features of past keywords that are collected during a training procedure and features of N-Grams that are part of the current document. These include the position of first occurrence of a candidate in the document, Part of Speech (POS) tag sequences, the Inverse Document Frequency (IDF), the χ^2 measure as well as the well-known metric TF-IDF. For the parameter tuning of the keyword extraction system the simplex algorithm by John Nelder and Roger Mead as well as Powell search are used. The results of the system with both parameter tuning methods are compared to results achieved using an SVM on the same task.
The thesis starts with the description of preprocessing steps that are necessary to give the text the right format. The training process where most of the feature information is gathered as well as the keyword extraction process itself using the three different approaches are described in detail afterwards. The three different keyword extraction approaches are evaluated with different POS thresholds and cleaning methods to prune bad candidates. Further experiments analyse the impact of differing feature combinations on the resulting F-scores.
The thesis concludes with a discussion of the experimental results as well as an outlook on prospective future improvements and changes to the developed system.

[2]http://www.ted.com/talks

Contents

List of Figures

List of Tables

1. Introduction

> *"But do you know that, although I have kept the diary [on a phonograph] for months past, it never once struck me how I was going to find any particular part of it in case I wanted to look it up?"*
>
> – Dr Seward, *Bram Stoker's Dracula, 1897*

Keyword extraction poses an important mechanism for the retrieval of information from documents, especially on large collections. Some use cases are, for example, document retrieval, document clustering, summarization or web page retrieval. Today, large text collections are made available to us on the internet, where we can search for almost any topic that comes to mind. But also in academic or private environments, the need for an efficient method to master a collection of documents is evident.

In these cases keywords can help to give a concise description of a document's contents which makes it easier for a prospective reader to decide whether or not the document is relevant. Another application is the comparison of documents by comparing their assigned keywords which enables their clustering according to similar topics. In an academic context a student could, for example, be interested in further reading material on a certain topic, either to get additional or different explanations or to find out whether the topic is of relevance in other contexts.

Without assigned keywords, however, this project could prove to be rather cumbersome and most people would sooner give up than work through a huge amount of information. Unfortunately, there are still lots of documents in all kinds of areas without keywords assigned to them and the task of manually attaching keywords to existing documents is a very laborious one. Additionally, different people might even attach very different keywords to a document because of differing understandings of the actual essence of a document. This could negatively effect comparability of documents.

An additional source of texts that becomes more and more relevant with ongoing research and ever better results in the field of speech recognition are transcribed texts. Transcribed texts offer the possibility to read spoken language and there are

already a lot of such texts available on the internet like the transcripts of TED[1] talks or the transcribed speeches of the European Parliament[2]. Another example might be transcribed versions of lectures in a university, which could provide additional learning materials for students. All these texts usually do not come with keywords attached to them as they are either manually or automatically transcribed from audio recordings or from a speech recognition system. However for people who would like to study those texts, keywords could be a great assistance for finding texts that are related to each other or cover a specific topic.
Finding ways to assign keywords automatically is therefore a key challenge for future information retrieval research.

1.1 History of Information Retrieval

For centuries human civilizations have known the importance of storing and archiving knowledge and information and the necessity of efficient retrieval of that stored information. With the invention of paper and later computers, the amount of information that could be stored grew ever larger and the need for efficient storing and retrieval methods became increasingly important.

In 1945 Vannevar Bush published an article titled *"As We May Think"* [1] in which he voiced the idea of using computers (or more specifically a machine he called "memex") to gain access to large amounts of stored knowledge.
During the 1950s a lot of research was done in the field of information science due to a generally increased interest in research caused by the just ended World War II. In 1950 Calvin Mooers was probably the first one to use the term *"Information Retrieval"* in a paper he published [2]. One important development was the proposal of using keywords for the indexing of items by Taupe et al. [3]. It allowed for more descriptors per document than had previously been common. After the Sputnik shock in the USA in 1957, research in the field of Information science experienced another boost. A very influential approach was proposed at the time by Hans Peter Luhn [4] in which he suggested that *"the frequency of word occurrence in an article furnishes a useful measurement of word significance"*. This approach later became known as *term frequency weighting*.
In his report in 1963 Dr. Alvin Weinberg [5] described a *"crisis of scientific information"* and how experts would be needed to master an *"information explosion"*. One of the major figures in the 1960s was Gerald Salton who worked at Harvard University and later Cornell University. Together with his research group he developed the **SMART** [6] system which allowed researchers to experiment with ideas to improve search quality.
In the 1970s the first online, interactive information retrieval systems like MEDLINE, Dialog, ORBIT and the European Space Agency's Information Retrieval Service (ESA-IRS) emerged. Furthermore, Karen Spärck Jones published a paper about "The significance of the frequency of word occurrence across the documents of a collection", the *inverse document frequency* [7].
During the 1980s, the developments of the 1960s and 1970s were improved and extended to fit the growing size of document collections. However, there was still a lack of availability of really large text corpora so it remained unproven whether

[1] http://www.ted.com/talks
[2] http://www.statmt.org/europarl/

the approaches would scale well with large text collections. To remedy these conditions, Donna K. Harman and Ellen M. Voorhees incepted the Text REtrieval Conference (TREC) [8] in 1992 which aims at encouraging research in Information Retrieval on large text collections.
With the rise of the *World Wide Web*, which was created by Tim Berners-Lee in 1989 [9], the interaction between researchers and commercial communities became stronger in the mid-1990s to develop new, efficient search engines to master the persistent growth of documents available in the web.
Until today there is still a lot of research going on in the broad field of Information Retrieval.

1.2 Related Work

The previous section already summarized that information retrieval in general and keyword extraction in particular have been important for human civilizations from the early centuries and that this importance grew ever larger with the invention of paper and later the computer.
For this reason there have been many different approaches to computer-based automatic keyword extraction since Bush's fundamental article in 1945 [1]. Some of these approaches will be presented in the following section.

Some approaches this thesis is also based on are the approaches of Y. Matsuo and M. Ishizuka [10], Frank et al. [11] and Anette Hulth [12].
In [10] the authors Y. Matsuo and M. Ishizuka present a keyword extraction method that is supposed to work on a single document without the need for additional information from a background corpus or other lexical features. The approach uses information about co-occurrences between frequent terms of a document and the other terms in the same document. A "term" by their definition is a single word or a word sequence. If the probability distribution of co-occurrence between a certain term a and the frequent terms is biased to a particular subset of frequent terms then the author's conclusion is that a is a keyword. The method of measuring the degree of bias is the χ^2-measure. According to the authors this approach performs comparable to the TF-IDF-measure but without using a corpus.
The approach from Frank et al. [11] introduces the Keyphrase Extraction Algorithm (KEA)[3] which is based on the naive Bayes machine learning technique. Furthermore it is shown that the extraction performance can be boosted by using collection-specific information about keyphrases. The first step of the KEA algorithm is pre-processing, basically consisting of splitting up the text, deleting unnecessary information (like numbers, stopwords and non-alphanumeric characters), case-folding of all words and stemming. All phrases of up to a length of three are then considered candidate phrases. The Bayes classification model of [11]'s approach consists of two features, the TF-IDF-score of a phrase and the distance into the document of the phrase's first appearance in a discretised form to fit Bayes' formula.
As opposed to the first two approaches Anette Hulth [12] only uses the abstracts of journal papers as source of information, arguing that for many of them no full-length copy is available. In her work she experiments with three different ways to select potential keyword phrases. One way is to select n-grams up to length three, the second is using noun phrase chunks and the third way consists of extracting

[3] http://www.nzdl.org/Kea/index.html

words or sequences of words that match a pre-defined set of POS tag patterns. As features for classification Hulth uses the within-document frequency, the collection frequency, the relative position of first occurrence and POS tags. The classification itself is done via a rule induction machine learning approach.
The overall conclusion of Hulth's approach is that each of the proposed keyphrase selection methods had its strength regarding *precision* and *recall* or an overall *F-score* and that POS information can be a valuable asset.

In 2002 James W. Cooper et al. [13] published an approach that represents another use case of keyword extraction - document comparison. Their aim is to distinguish between documents, extracted from the web, that are merely edited versions of each other or occur in different forms (such as PDF and HTML). The **Talent** suite [14] serves as instrument for document analysis with **Textract** for extracting named entities, multi-word terms etc. Each term is given an *Information Quotient (IQ)* that measures document selectivity of that term and it is categorized in one of nine pre-defined categories. All collected term information is later used to construct a database that can be queried for similar documents according to differing rules.

That the problem of automatic keyword extraction is still important is proven by several very recent approaches from 2013. In [15] Bidyut Das, Subhajit Pal and Suman Kr. Mondal propose an algorithm that focuses on n-gram rigid collocations as potential keywords which is similar to Matsuo's and Ishizuka's approach [10]. Unlike them the former assign a so called *Fuzzy Bigram Index (FBI)* or *Fuzzy N-Gram Index (FNI)* based on mutual information as a score to classify their terms. Like in Matsuo and Ishizuka's approach a term is either a mono-, bi-, or tri-gram. The resulting set of keywords contains the 20 highest ranked bi- and tri-grams according to the applied evaluation metrics *precison*, *recall* and *F-score*. Monograms are only included in the result if their frequency is higher than eight times, the highest frequency of a bi-gram possibly adding an additional five terms to the result set.
The approach of Weidong Jiang and Xiaofeng Hui [16] deals with the problem of imbalanced classes. In their research they illuminate this problem from two different perspectives: the algorithm and the data perspective. One possibility to approach the problem from the algorithm perspective is the *ensembling*, or more specifically, the Boosting method [17], which has a good generalization because multiple learners are trained instead of only one like with other machine learning approaches. To solve the problem from the data perspective Jiang and Hui apply selective over- or under-sampling of the data to "fill up" minority classes or reduce majority classes. The over-sampling is done using the SMOTE [18] method. For the task of classification an SVM is used [19].
Based on the TF-IDF score comes the approach from Rahki Chakaraborty [20]. The angle of her research is that conventional TF-IDF has the problem that it only considers keywords of single documents and not whole topics. Therefore it makes the assumption that a good keyword has a high frequency in one document (the one it is a keyword for) and a low frequency in the rest of the collection thereby neglecting the fact that a term that has a low frequency may still be a good representation of a certain domain. To correct this, Chakaraborty adds an additional weight to the TF-IDF score which also considers the frequency of a term which is part of a particular domain of the whole text collection. This additional weight is called the *common word possession rate*.
The last approach I am going to introduce is the one from Ludovic Jean-Louis et

al. [21] which can be divided into two steps, identification and ordering of keyword candidates. At first candidate keywords have to be extracted, the extraction process basically consisting of the application of specific rules defining the features of good keywords. One source of additional information is Wikipedia[4] which provides a large collection of documents on various domains divided into specific categories that are used to build an additional feature for keywords. All candidate keywords are then ranked using a learning-to-rank algorithm which uses a user defined keyword profile that monitors the users favourite domain knowledge. The actual ranking and classification of the candidate terms is done by a passive-aggressive perceptron using the features obtained from the keyword profile and some additional ones. The top 15 candidates compose the output of the extraction algorithm.

1.3 Goals

In my introductory motivation I already outlined how important keyword extraction is today and will probably be even more so in the future. This thesis focuses on the extraction of keywords from transcripts of spoken language which may differ from written texts. The goal of this thesis is to apply different approaches to the keyword extraction task on transcripts and evaluate them with respect to their performance on the input data and their respective features. For each individual approach the same questions have to be answered concerning the data input format, the representation of features and the evaluation of results that are suited best for the respective approach. The questions are the following:

- **What is a keyword?**
  What a good keyword should look like is an important question to solve for each approach that tries to extract keywords. Examples might be "single word units", "multi word units", "nouns" etc.

- **What is the input of the system?**
  This question deals with the problem of what kind of data an extraction system works with in the end. Should the data be pre processed and how? What kind of documents are we dealing with? Is there domain specific information that can be taken advantage of?

- **What features are discriminative?**
  If there is the possibility to retrieve information from existing texts with pre-assigned keywords, what kind of information could be the most relevant to determine keywords in future texts? What is the best evidence to determine if a candidate term is a good keyword or not? Or if there are no pre-labelled documents what evidence can be gathered from a previously unseen document?

- **How should keywords be classified?**
  As has been outlined in the previous section there are several different approaches for keyword classification. Different keyword extraction systems may choose to use different classifiers that probably are a better fit for the systems' respective needs.

[4] http://en.wikipedia.org/wiki/Main_Page

- **How can results be evaluated?**
  As with the classification, there are also several different methods to evaluate the results of a keyword extraction system. Depending on the use case or other features of the system one might choose to use an evaluation based on human opinions. In case of an automatic evaluation there is the possibility of different metrics or parameter settings like the weight of precision over recall in the F-measure as is described in section 2.7.

I my thesis, I will provide answers to each of the above questions for every approach that was tested, to find the best possible method of keyword extraction for the given input data.

1.4 Outline of the Thesis

The goal of this first chapter was to outline the importance of information retrieval in general and keyword extraction in particular. There has been a brief overview about the history of information retrieval and about some approaches to keyword extractions that have been taken in the last twenty years. The past paragraph outlined the main goal of this thesis.

The **second chapter** is intended to give a brief overview of the most important principles this thesis is based on. There will also be a brief description of different classification, parameter tuning and evaluation methods that are available in natural language processing and relevant for this thesis.
In **chapter three** I will give a detailed overview of the keyword extraction systems that were developed and tested in the context of my thesis, describing the pre-processing-, training-, classification- and evaluation stages of each algorithm.
chapter four outlines experiments that were performed to test the extraction systems under different circumstances. For each experiment the chapter contains a presentation and discussion of the results that were obtained with the respective approaches.
The **last chapter** gives a resume about the results of this thesis and will further provide a brief outlook as to what could be done to further improve the results of keyword extraction on transcript data.

2. Fundamentals

This chapter provides some necessary fundamentals this thesis is based on. I first describe classification of text in general and keywords in particular, which is followed by a description of two distance measures that are later used for clustering of keyword candidates. Section 2.4 provides a short description of a third distance measure that provides a basis for one of the features that is used in this thesis. In section 2.5 I describe in general the importance of parameter tuning and introduce two prominent approaches. The chapter concludes with a section about the evaluation of information retrieval tasks where some metrics for evaluation are described.

2.1 Classification

The task of text classification is a common problem in the field of natural language processing. Some examples for text classification tasks are text categorization (for example whether a text is spam or no spam), authorship attribution (who wrote this text?) or gender identification (was the author of the text male or female?).
The classification task of this thesis is to decide whether a certain candidate term is a keyword for the current text or not. As can probably be derived from the related work section 1.2, there exist many different ways to accomplish the task of classifying possible candidate terms either as being a keyword or not. Two popular classification methods I want to introduce here which are also prominent in the field of machine translation are the **naive Bayes model** and the **log linear model**.

The naive Bayes model is simply the application of Bayes' formula which describes the relationship between two independent probabilities $P(A)$ and $P(B)$ and the corresponding *conditional probabilities* $P(A|B)$ and $P(B|A)$ [22, p. 43]. It is defined as

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.1}$$

Although it is based on the assumption that the events A and B are independent, naive Bayes has been shown to produce good classification results even if this independence assumption is not correct [23].

The second classification method I would like to describe in this paragraph is the method of using log linear models. Log linear models belong to the group of discriminative models as opposed to the Bayes model which is a generative model. They are very flexible and allow for the use of rich feature sets in a model. One possible definition of a feature based log linear classifier in natural language processing is

$$P(c|d,\vec{\lambda}) = \frac{\exp\left(\sum_i \lambda_i \cdot h_i(c,d)\right)}{\sum_{c\in C} \exp\left(\sum_i \lambda_i \cdot h_i(c,d)\right)} \tag{2.2}$$

In the above formula $P(c|d,\vec{\lambda})$ is the probability of a particular candidate being a keyword c of the set of all possible candidates, given a document d and given the set of parameters $\vec{\lambda}$. The function $h_i(c,d)$ is the *feature function* which describes a particular feature of a candidate. The λ_i serve as weights for each feature which allows to discriminate between more and less important features. The decision rule in this thesis is

$$\hat{c} = \arg\max_c \{P(c|d,\vec{\lambda})\} \tag{2.3}$$

$$= \arg\max_c \left\{\sum_i \lambda_i \cdot h_i(c,d)\right\} \tag{2.4}$$

which makes the re-normalization of equation 2.2 unnecessary. In equation 2.4 $\hat{c}$ denotes the best candidate keyword. It is the candidate that achieved the best score with its feature values. This makes it the most likely keyword candidate for the given document. In a keyword extraction process, however, the classification task is not only to find the best keyword for a document, but a set of keywords that provide a good of summary of the documents contents. This leads to the definition of a set B which is a set of the b best keyword candidates.

All feature functions h_i have the form $h_i = log(f)$ where f is one of the features used for classification. If the values of f are normalized to lie between zero and one, the values of $h_i = log(f)$ will be close to zero for good candidates with high feature values and will strictly decrease the closer the feature values f get to zero because of the monotonous nature of the logarithm. As many optimization algorithms are defined to search for a minimum, it is convenient to re-write formula 2.4 as

$$\hat{c} = \arg\min_c \left\{-\sum_i \lambda_i \cdot h_i(c,d)\right\} \tag{2.5}$$

to get positive results and look for the candidate $\hat{c}$ that minimizes the function.

2.2 Jensen-Shannon Divergence

There are several possible methods to measure the similarity and differences between two probability distributions. A very popular one in probability theory and statistics that will also be deployed in this thesis is the Jensen-Shannon Divergence (JSD).
The JSD is based on the Kullback-Leibler Divergence (KLD) but differs from the latter in that it is symmetric, meaning that for two probability distributions P and Q, $JSD(P \parallel Q)$ is always equal to $JSD(Q \parallel P)$, and it is always a finite value. The JSD is defined as

$$JSD(P \parallel Q) = \frac{1}{2} \cdot D(P \parallel M) + \frac{1}{2} \cdot D(Q \parallel M) \tag{2.6}$$

where

$$M = \frac{1}{2}(P + Q)$$

and $D(\cdot \parallel \cdot)$ is the Kullback-Leibler divergence [24], [22, p. 304]

$$D(P \parallel Q) = \sum_i log\left(\frac{P(i)}{Q(i)}\right) \cdot P(i) \tag{2.7}$$

2.3 Mutual Information

Mutual Information (MI) can be described as a measurement of the shared information of two discrete random variables X and Y [25, pp. 13-22].
It measures the extent to which knowledge about one of the variables impacts certainty over the outcome of the other variable. A measure of certainty that is widely used in scientific disciplines is **entropy** which is defined as

$$H(X) = -\sum_{x \in X} p(x) \cdot log\ (p(x)) \tag{2.8}$$

The relation of two random variables can now be described with the **joint entropy** which is

$$H(X, Y) = -\sum_{x \in X, y \in Y} p(x, y) \cdot log\ (p(x, y)) \tag{2.9}$$

And with this, the **conditional entropy** can be defined as

$$H(X|Y) = H(X, Y) - H(X) \tag{2.10}$$

With this equation it is now possible to model how much uncertainty is removed from the joint entropy of two random variables when one of the random variables is known. However it might not necessarily be the case that $H(X|Y) = H(Y|X)$.

This problem is solved introducing a symmetric measure called mutual information which is defined as

$$I(X;Y) = \sum_{x \in X, y \in Y} p(x,y) \cdot log\left(\frac{p(x,y)}{p(x) \cdot p(y)}\right) \tag{2.11}$$

If the two random variables X and Y are independent, then having information about X does not give any information about Y and vice versa, so $I(X;Y)$ is zero. If X is a deterministic of Y and vice versa, knowledge about X perfectly predicts Y which makes $p(x,y) = p(x)$ and $I(X;Y) = H(Y)$.
A special case of mutual information is the Pointwise Mutual Information (PMI) M, which describes the same correlations for a pair of particular outcomes $x \in X$ and $y \in Y$:

$$M(x;y) = log\left(\frac{p(x,y)}{p(x) \cdot p(y)}\right) \tag{2.12}$$

2.4 χ^2-Measure

The χ^2-measure can be used to measure the independence of two events A and B or the difference between two frequency distributions. The χ^2 test checks if the frequency distribution of events observed in a sample is consistent with a particular theoretical distribution. This statement is referred to as the *null hypothesis*. The chi-squared test statistic is defined as

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{2.13}$$

where O_i denotes an observed frequency, E_i denotes the expected (theoretical) frequency and n is the number of samples. All events considered in the χ^2-test have to be mutually exclusive and their total probability must add up to one [26].

2.5 Parameter Tuning

In order to get the best possible results from a classification model that uses a classifier as described in the previous section it is vital to choose optimal weights λ_i for each feature function. The λ_i values serve as weights representing the influence of each feature function on the final score of a keyword. The feature with the most discriminative characteristic should get a higher weight than one that adds only little information to the overall score.
Parameter tuning is always done with a specific metric in mind. This metric may be one of those described in the following section for an information retrieval task. The goal of parameter tuning is to achieve an optimal score according to that metric. To achieve this all possible settings of λ_i weights have to be studied. The problem with different features is that they create a space of possible feature values that is too large to be exhaustively searched within a reasonable time, even for a small set of feature functions. Good heuristic methods provide the only means for exploring this space.

2.5.1 Powell Search

One prominent method for parameter tuning is **Powell search** [27] which was proposed by Micheal J. D. Powell in 1964. Powell's algorithm aims to optimize one parameter out of the high-dimensional parameter space at a time, thereby simplifying the optimization task. One important aspect of the algorithm is that it does not require a derivation of the function it is supposed to optimize, like other optimizing algorithms such as gradient descent. Computing the derivation of the classification function in information retrieval as well as machine translation tasks is often not feasible and sometimes even impossible. Additionally, the gradient descent method can only find a global optimum if the target function is convex which is also not the case for typical error metrics [28].

In the case of a log linear classifier described by a formula like 2.5 Powell's approach of optimizing one parameter at a time leads to the following representation of the target function:

$$\hat{c} = \arg\min_{c} \left\{ -\lambda_a \cdot h_a(c,d) - \sum_{i \neq a} \lambda_i \cdot h_i(c,d) \right\} \tag{2.14}$$

$$= \arg\min_{c} \left\{ -\lambda_a \cdot h_a(c,d) + u(c,d) \right\} \tag{2.15}$$

where $-\lambda_a \cdot h_a(c,d)$ represents the feature function whose parameter λ_a is currently to be optimized and $u(c,d)$ defines a constant value that is independent of the changing parameter λ_a.
Despite looking at only one parameter at a time the optimization problem might still be quite expensive, for some parameters have real-numbered values that provide for an infinite number of possible values. Formula 2.15 provides a representation of the probability score of a keyword candidate as a linear function. Now the only changes of the ranking of keyword candidates occur at intersection points of two or more of those linear functions. This insight allows for a reduction of λ_a values from all possible ones to only the ones at intersection points which is visualized in figure 2.1 . Because Powell Search is usually prone to getting stuck in local minima the parameter tuning process should be started several times with random initial values for the parameters λ_a. Typical values for the number of restarts are between five and ten.

2.5.2 Downhill Simplex

Another important method, the **downhill simplex method**, was described by John Nelder and Roger Mead in 1965 [29]. The downhill simplex method can be used to minimize mathematical functions that have more than one variable.
Initially a simplex is defined by a set of $n+1$ points for a function with n variables and three special values are computed: The maximum function value at a point P_i is denoted as h, the minimum function value at a point is denoted as l and the centroid of all points with $i \neq h$ is denoted as $\overline{P}$. Additionally $[P_i P_j]$ is denoted as the distance between P_i and P_j. The downhill simplex algorithm consists of three main operations, the *reflection*, *contraction* and the *expansion* operation. It works as described by algorithm 1.

Algorithm 1 Downhill Simplex

1: $Simplex = \{P_i\} i \in [0, n]$, with n = number of variables
2: $h = \max_i(f(P_i))$
3: $l = \min_i(f(P_i))$
4: $reflection(h) := P^* = (1 + \alpha) \cdot \overline{P} - \alpha \cdot P_h$
5: **if** $f(l) \leq f(P^*) \leq f(h)$ **then**
6: $h = P^*$
7: GOTO 1
8: **end if**
9: **if** $f(P^*) < f(l)$ **then**
10: $expansion(P^*) := P^{**} = \gamma \cdot P^* + (1 - \gamma) \cdot \overline{P}$, with $\gamma = [P^{**}\overline{P}] : [P^*\overline{P}]$
11: **if** $f(P^{**} < f(l)$ **then**
12: $h = P^{**}$
13: GOTO 1
14: **end if**
15: **if** $f(P^{**}) > f(l)$ **then**
16: $h = P^*$
17: GOTO 1
18: **end if**
19: **if** $f(P^*) > y_i \forall i \neq h$ **then**
20: **if** $\gamma(P_{h\_old}) < \gamma(P^*)$ **then**
21: $P_h = P_{h\_old}$
22: **else**
23: $P_h = P^*$
24: **end if**
25: $contraction(h) := P^{**} = \beta \cdot P_h + (1 - \beta) \cdot \overline{P}$, with $\beta = [P^{**}\overline{P}] : [P\overline{P}]$
26: $P_h = P^{**}$
27: **if** $f(P^{**}) > \min(f(h), f(P^*))$ **then**
28: $P_i = \frac{(P_i + P_l)}{2} \forall P_i$
29: GOTO 1
30: **else**
31: GOTO 1
32: **end if**
33: **end if**
34: **end if**

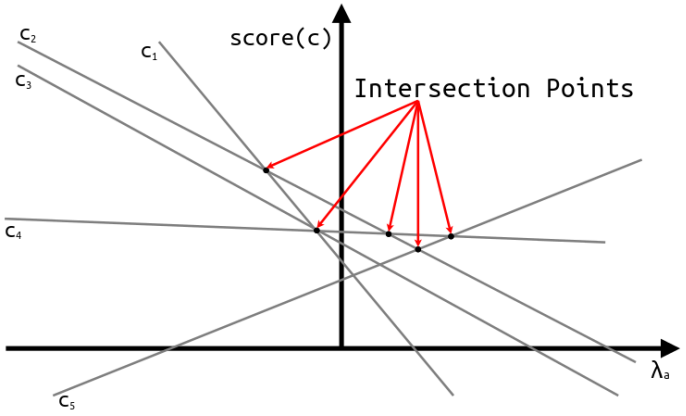


Figure 2.1: Linear Functions and their Intersection Points

The algorithm starts with an initial simplex as described above. First the reflection of h is calculated and creates a new point P^* which lies on the line joining h and $P\overline{P}$. If the new value $f(P^*)$ lies between $f(l)$ and $f(h)$ then P_h is replaced by P^* and the process is repeated with the newly created simplex. If the reflection has produced a new minimum, P^* is expanded to P^{**} as defined in the algorithm. If $f(P^{**})$ is smaller than $f(l)$, P_h is replaced by P^{**} and the process is restarted. If $f(P^{**})$ is greater than $f(l)$, the expansion failed and P_h is replaced by P^* before restarting. A failed expansion can occur if the expansion overshoots the "valley" of a function (P^*) and thus ends up on the opposite slope. If $f(P^*)$ turns out to be the new maximum after a reflection operation, h is replaced by either the old h or P^*, whichever has the lower $f(x)$ value and a contraction is performed. Afterwards h is replaced by P^{**} which was the result of the contraction and the process is again restarted. If, however, $f(P^{**})$ is greater than the minimum of $f(h)$ and $f(P^*)$ all P_i are replaced by $\frac{(P_i+P_l)}{2}$ before restarting and the contraction is considered a failure. The goal of the algorithm is to contract the initial simplex towards the lowest point of a function, bringing all points into the valley as a result of the process.

This process of creating new simplexes to find the minimum of a given function is repeated until the value of $\sqrt{\left\{\sum \frac{(f(P_i)-f(\overline{P}))^2}{n}\right\}}$ becomes less than a pre-defined value, a standard value for which is 10^{-8}. The initial simplex is important and should not be too small because this could lead to the algorithm getting stuck on local minima. Additionally the size and orientation of the initial simplex and the step length (defined by the values of α, β and γ) also has an effect on the speed of convergence of the algorithm. Standard values are $\alpha = 1$, $\beta = \frac{1}{2}$ and $\gamma = 2$.
Compared to Powell's Method 2.5.1, the downhill simplex algorithm usually requires less function evaluations until it converges.

2.6 Support Vector Machines

Support Vector Machines (SVMs) are supervised learning models with associated learning algorithms that can be used for classification in a machine learning task such as keyword extraction. Given a set of training examples $(x_i, y_i), i = 1, \ldots, n$ where $x_i \in R^{n \times m}$ are the n training examples with m features and $y \in \{1, 0\}$ are the class labels assigned to each x_i, SVMs can be used to separate that set of examples (which is not linearly separable in the original feature space) by mapping it to a higher dimensional space where separation might be easier.
The current standard SVM model was published by Corinna Cortes and Vladimir N. Vapnik in 1995 [19]. It allows for some mislabelled examples $\xi_i, i = 1, \ldots, l$ after separation when it is not possible to cleanly split the training examples into two classes. The method is called the *Soft Margin*. It solves the optimization problem that can be described by the following formula

$$\arg\min_{w,\xi,b} \left\{ \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{l} \xi_i \right\} \tag{2.16}$$

$$\text{subject to } y_i(w \cdot \phi(x_i) + b) \geq 1 - \xi_i, \tag{2.17}$$

$$\xi_i \geq 0 \tag{2.18}$$

where w denotes the normal vector to the separating hyperplane described by $w \cdot x_i + b$ and C denotes the penalty of the error term. $\phi(x)$ is the function that maps the training vectors x_i into a higher dimensional space.
In general a SVM constructs a hyperplane or a set of hyperplanes in a high dimensional space to separate a set of training examples that is not linearly separable in the original feature space. The hyperplanes that are chosen to separate the data are the ones that have the largest distance to the nearest training data point of any class. Figure 2.2 depicts an example of three hyperplanes separating two different classes. As can be seen, hyperplane H_1 does not separate the classes, H_2 separates them but with very small distances, while H_3 maximizes the distance between the two separated classes.

Support Vector Machines are also capable of solving non-linear classification problems with the help of so-called kernel functions [30]. Instead of calculating the dot product for measuring distances, non-linear kernel functions are used to allow for non-linear transformations to fit the maximum margin hyperplane in the transformed feature space. In this thesis the library LIBSVM A.2 was used to experiment with Support Vector Machines.

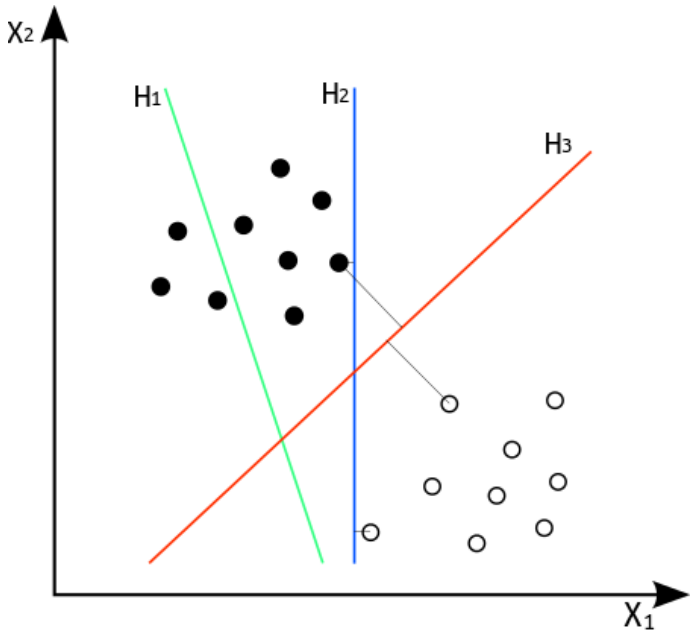


Figure 2.2: Hyperplanes Separating two Classes

2.7 Evaluation

Evaluating the results of information retrieval or natural language processing tasks is required to provide a basis for comparing different approaches. Here automatic methods are preferable to human aided ones because they provide an unbiased view on the data.
Two measures that are frequently used in information retrieval are **precision** and **recall**, which are both based on an understanding and measure of relevance. Precision can be described as the fraction of retrieved instances that are relevant while recall is the fraction of relevant instances that are retrieved. In a more formal context they are defined as

$$precision = \frac{tp}{tp + fp} \tag{2.19}$$

$$recall = \frac{tp}{tp + fn} \tag{2.20}$$

where *tp (true positives)* denotes the instances that are relevant to the task and were retrieved from the system, *fp (false positives)* denotes instances that were retrieved from the system but are not relevant and *fn (false negatives)* denotes the instances that would have been relevant but were not retrieved by the system. This relationship is illustrated in Figure 2.3. Here the rectangle plane depicts the space of all possible instances. The right circle represents the set of target instances that are relevant to the current task and the right circle is the set of instances the system extracted. The two circles overlap, creating a green area which depicts the true positives. The yellow area represents the false negatives, the blue one the false positives and finally the red one depicts the true negatives.
In some information retrieval task it makes perfect sense to trade of precision and recall, choosing either very high recall and getting a low precision or vice versa. In

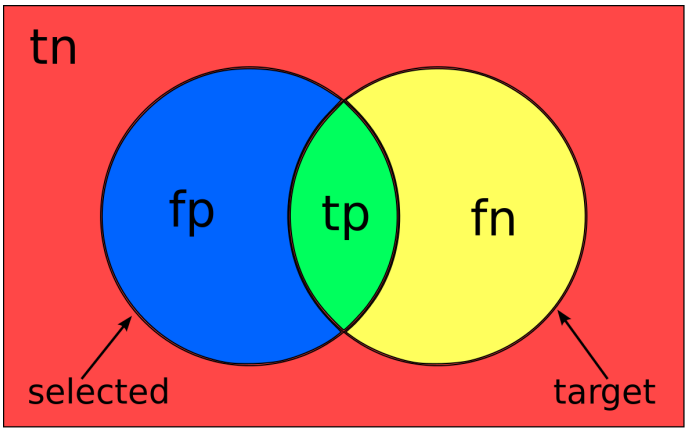


Figure 2.3: Precision and Recall

some tasks, however, this effect is not wanted in which cases another measure is utilised, the **F-measure** which denotes an harmonic mean of precision and recall:

$$F = \frac{1}{\alpha \cdot \frac{1}{P} + (1-\alpha) \cdot \frac{1}{R}} \tag{2.21}$$

where P is precision, R is recall and α is a scaling factor that allows for a weighting of P and R. The most common value for α is 0.5 which leads to the so-called *balanced F-score* that gives equal weights to P and R [22, pp. 267-269].
In information retrieval it is also quite common to evaluate rankings of relevant instances. **Uninterpolated average precision**, **interpolated average precision levels of recall** and precision at particular **cut-off points** are specifically designed to satisfy that requirement [22, pp. 534-538].

2.8 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical method that uses orthogonal transformation to convert a set of redundant, correlated variables into a set of linearly uncorrelated variables. In machine learning it can be used to reduce the dimensionality of problems with a huge number of variables. The reduction of dimensionality can either enable a machine learning process to run faster or need less disk or memory space or it can be used to make high dimensional data plottable for visualization if it is projected down to 2D or 3D.
If PCA is applied to a set of data X it finds a lower dimensional surface, onto which to project the data, which minimizes the sum of squared distances of all examples $x_i \in X$ to that surface. The sum of squared distances is also called the "projection error" and can be described by the following formula

$$\frac{1}{m} \cdot \sum_{i=1}^{m} \| x_i - x_i' \|^2 \tag{2.22}$$

where m is the number of examples in the data set X and x_i' is the projection of the original example x_i. Before applying PCA, the data is usually preprocessed performing mean normalization and feature scaling so that every variable or every

feature has zero mean and a comparable range of values.
Mean normalization is done by first computing the mean μ_j of every feature j with

$$\mu_j = \frac{1}{m} \cdot \sum_{i=1}^{m} x_{i,j} \tag{2.23}$$

and then replacing each $x_{i,j}$ with $x_{i,j} - \mu_j$. Because it can easily be the case that the different features have very different value scales it is also necessary to perform feature scaling to provide a comparable range of values for each of the features.
To apply PCA to the data, at first the covariance matrix Σ of the data is created with

$$\Sigma = \frac{1}{m} \cdot \sum_{j=1}^{n} x_j x_j^T \tag{2.24}$$

Then the eigenvectors of this matrix are calculated using for example singular value decomposition. To reduce the data from n to k dimensions the first k eigenvectors are used and multiplied with the data matrix X to provide a k-dimensional version of the original data.
If the target dimension is not 2D or 3D which can be used for data visualization, k is usually chosen to be the smallest value that still retains a certain percentage of the data variance, for example 99%. The data variance is defined as

$$\frac{1}{m} \cdot \sum_{i=1}^{m} \parallel x_i \parallel^2 \tag{2.25}$$

and so, if 99% of the variance should be retained, k is the smallest value with

$$\frac{\frac{1}{m} \cdot \sum_{i=1}^{m} \parallel x_i - x_i' \parallel^2}{\frac{1}{m} \cdot \sum_{i=1}^{m} \parallel x_i \parallel^2} \leq 0.01 \tag{2.26}$$

3. Keyword Extraction

This chapter is going to introduce and explain in detail the keyword extraction system that was developed during this thesis. First I would like to give a rough description of the developed system and provide an overview of its structure. Sections 3.2, 3.3, 3.3.5 and 3.4 correspond to the most important steps that are part of the keyword extraction process, describing the data processing and retrieval of decisive information of each step.
Furthermore, I will describe the Support Vector Machine-based approach that was followed in this thesis. In section 3.3 the training for the SVM will be described and the following section discusses the set up of the SVM-system that has been used for keyword classification.

3.1 System Overview

Figure 3.1 depicts an overview of the extraction systems structure. The keyword extraction process can be divided into two phases, the first of which includes a training process and tuning of feature parameters, the second representing the keyword extraction process itself.
The training process will be described in detail in section 3.3, the tuning of feature parameters will be covered in section 3.3.5. Both phases require their input data to be pre-processed, therefore a pre-processing step proceeds each phase. All necessary pre-processing steps will be described in the next section.
As has been noted before, the second phase is where the actual keyword extraction occurs. In this phase one or more documents for which keywords are to be extracted are pre-processed and keywords for each document are computed according to discriminative features that will be described later. The necessary information is obtained during training and the tuned feature parameters. The result of the extraction process are a maximum of 20 keywords that are returned at the end of the computation. The process of keyword extraction as conducted in this thesis will be described in section 3.4.

A keyword in this thesis is defined to be an N-Gram up to a length of $N = 4$. The decision to allow whole phrases of a length greater than one was made because a

lot of keywords that describe a text are actually phrases consisting of more than one word. Allowing only one-word phrases would have forced those phrases to be split up. Furthermore, having phrases of up to length four allows for including the information provided, for example by adjectives or verbs immediately preceding or following a noun, making it possible to give a more detailed description of a document's contents.
As the keyword extraction decision is based on different features that are considered discriminative for keywords sections, 3.3 and 3.4 will be subdivided into smaller sections that are congruent with the relevant features for their parent section.

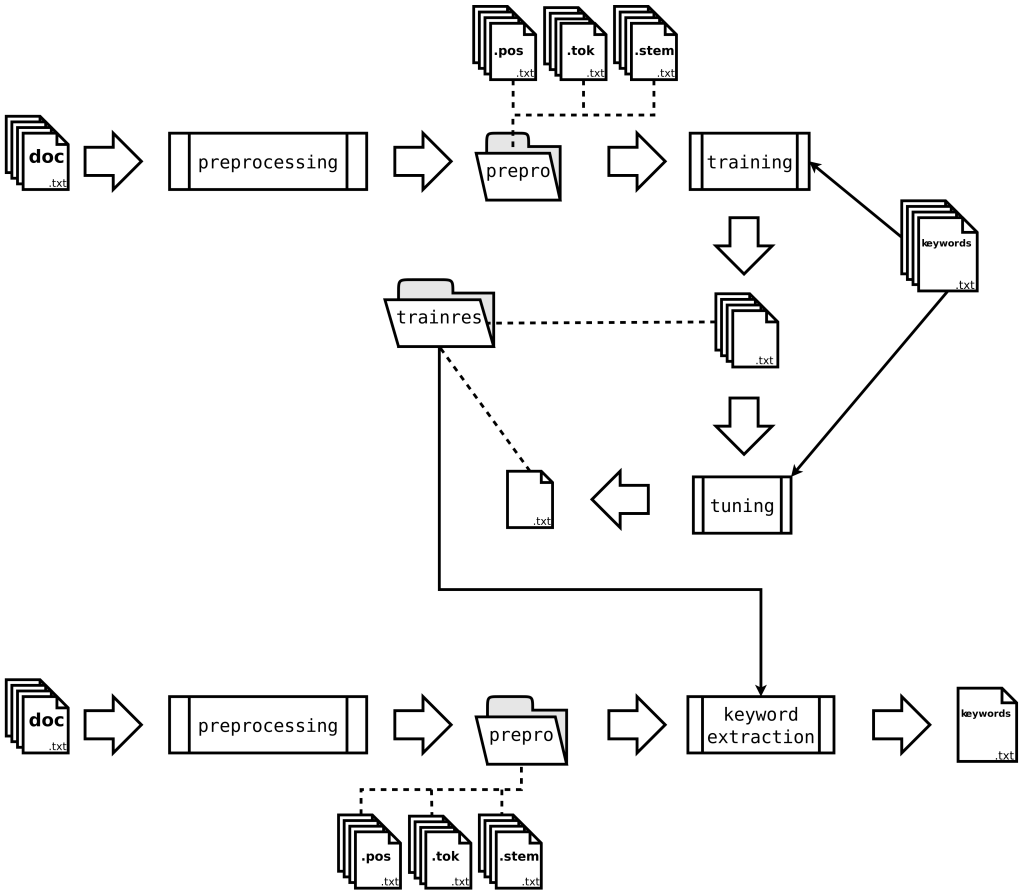


Figure 3.1: Keyword Extraction System Overview

3.2 Pre-processing

The first issue to address when extracting keywords from documents is the data itself. For the keyword extraction system developed in this thesis it is essential to have only data for processing that is comparable, otherwise counts could be messed up and conclusion drawn from misleading evidence. Additionally, results can only be reasonably compared with each other if the input of the system does not differ beyond a certain level. A differing length of two documents might for example be not as important as the fact if the words in the documents are lower cased or not which could lead to differing frequency counts.
In this thesis the corpus that was used for training, tuning and testing was composed of transcribed English talks from the Technology, Entertainment, Design (TED)[1]

[1] http://www.ted.com/talks

website. As these transcripts contain numbers for each transcribed line as well as timestamps and blank lines that do not contain any information concerning potential keywords all of the files have to be pre-processed to suit the needs of the keyword extraction system.
Therefore several pre-processing steps are executed for each new document:

- **Text Cleaning**
  In this step the talks are re-formatted in order to create a document with only lines containing text in it. For this purpose all blank lines are removed from the document as well as lines containing only a line number or a time stamp. Additionally in this step the documents get stripped of lines consisting only of response tokens like *(Applause)* or *(Laughter)*.

- **Tokenizing**
  The tokenizing step actually consists of several individual steps.

  - Sentence fragments, that were originally created so the transcripts fit into the video screen, are merged together again which leads to each line containing a single sentence in the resulting document. This is done using the `nltk.PunktSentenceTokenizer` which uses an unsupervised algorithm to build a model that is used to find sentence boundaries. Typical punctuation marks for determining sentence boundaries are `[';', '.', ':', '!', '?', ',']`.
  - Splitting contractions like `they'll`, splitting of commas and single quotes when followed by whitespaces and separating periods at the end of lines. For this the `nltk.TreebankWordTokenizer` is used.
  - Lowercasing all upper cased words in each line.
  - The splitted parts of the contractions get substituted with their long form, for example `they'll -> they 'll -> they will`.

- **Part-of-Speech Tagging**
  In this step all words are assigned part-of-speech tags according to the *Penn Treebank tagset* as described in [31]. The result is a file equivalent to the original document but consisting only of part-of-speech tags.

- **Stemming**
  The stemming step finally removes morphological affixes to leave only the stem of a word. Using only the word stem helps to get more stabilized counts for individual words than in the case of treating each form with a different ending as a different word. The stemming is done with the `nltk.SnowballStemmer` which is a port of the *Snowball stemmer* (an implementation of the *Porter stemming algorithm* [32]) developed by Martin Porter. *Snowball*[2] is a language especially invented to express the rules of stemming algorithms in a natural way.

As is indicated in figure 3.1, different output files form the result of the pre-processing step. The tokenizing, part-of-speech tagging and stemming step each produce their own set of output files that are later required as input for the training of different feature counts. The pre-processing is the same for the SVM-system and the keyword extraction system developed in the thesis.

[2] http://snowball.tartarus.org/

3.3 Training

Some of the features used for discriminating keywords from non-keywords in this thesis are based upon unsupervised algorithms, others require supervision. In the training process the keyword extraction system accumulates knowledge about all supervised features in order to learn how certain pieces of information such as frequencies can be used to determine if a particular term is a good keyword candidate for the current document or not. Figure 3.2 depicts the training process in detail.

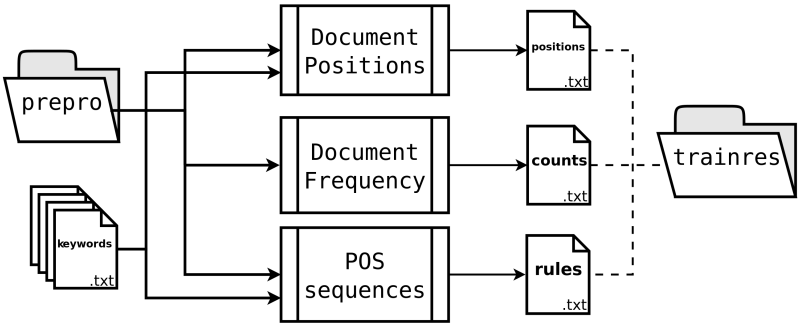


Figure 3.2: Training Process Overview

After the pre-processing step, which is performed as described in the previous section, the resulting files are used to learn about the different features. Additionally, the pre-labelled keywords for each of the training documents are taken into account by some of the training algorithms. Others only need the text documents themselves to compute frequencies. All the training algorithms will be described in the following subsections. All training results are finally stored in a dedicated results directory, the "trainres" directory.

3.3.1 POS Sequences

This feature makes use of the POS tags that are assigned to each of the extracted N-grams. Using the POS tag feature the system learns which part-of-speech tags are usually correlated with a keyword and which are not. As manually assigned keywords are often composed of nouns, verbs, adjectives or adverbs because they provide the most context information, an automatic keyword extraction system should focus on these word classes as well during the extraction process.
In order to achieve this, the extraction system developed in this thesis attempts to learn part-of-speech patterns or sequences that are most common for good keywords (hereby assuming that manually assigned keywords form a good representation of the corresponding document). Unlike Hulth's [12] approach, the POS tag patterns are automatically learned from the pre-assigned keywords of the training texts and are not derived from manually constructed rules.
The learning process for the POS tag sequences is described by the algorithm "Extraction of POS tag Sequences" 2. For the extraction of N-Grams as well as the assignment of POS tags special Natural Language Toolkit (NLTK) functions are used.

The resulting output file contains all the gathered POS tag sequences. During classification a filtering is applied to the POS sequences to obtain those sequences that are considered "part-of-speech tag rules" for the future keyword extraction process.

Algorithm 2 Extraction of POS tag Sequences

```
1:  postaglist := {}
2:  for all keyword files k ∈ K do
3:      KW :=list of manually assigned keywords
4:      if ∃ document d corresponding to k then
5:          open d
6:          ngrams := all N-Grams ∈ d corresponding to kw ∈ KW
7:          postagseq := {}
8:          for all ngram ∈ ngrams do
9:              postagseq.append(ngram.assign_postag)
10:         end for
11:         for all tag sequence ∈ postagseq do
12:             counts := # occurences of tag sequence
13:         end for
14:         postaglist.append(counts)
15:     else
16:         try with next keyword file
17:     end if
18: end for
```

Keeping all extracted POS sequences from the training process allows for including knowledge of future keyword extractions by simply updating existing counts or adding new entries to the list of POS sequences.

3.3.2 Document Position

The relative position of a word or phrase within its document is denoted in this thesis simply as the document position. Based upon this feature the keyword extraction system is supposed to learn making predictions about a phrase being a keyword or not from the document positions of past keywords. “Document position” here means the position of the first occurrence of a phrase in the text, later occurrences are not relevant for this feature.
If the distribution of keywords contained within a document concerning the position of their first occurrence is biased to a certain area of that document, this information can be used to predict the likeliness of a candidate phrase to be a keyword. If, for example, keywords tended to occur more often at the beginning and the end of a document, it could be concluded that a candidate phrase making its first appearance at the beginning is more likely to be a keyword than if it occurred somewhere in the middle.

The learning task for the document position feature is first to compute the position of the first occurrence of each pre-labelled keyword. If the keyword is composed of more than one word the first occurrence of its first term is defined to be the position of the whole keyword. This computation is conducted for each pre-labelled keyword in every file of the training corpus. However, if every possible position of a keyword in a document would be considered and treated individually, the counts would very likely have only a limited informative value as they would likely be rather small. Therefore each training document gets separated into position clusters and all the document position counts in the area of a cluster are accumulated to a single cluster

count.
The number of clusters has to be sufficiently large to provide a fine grained structure for the documents but without having again a problem of sparse counts. In this thesis, counts of keywords in a cluster are considered to be too sparse if there is one cluster with no keyword in it at all. This cluster would later get an assigned probability of zero which would exclude the possibility of a potential keyword being in this part of the document in the future. The clusters were chosen to be all equal in size but also other approaches are possible as will be described in section 5. The last step of training the document position feature is to compute a percentage distribution of the document positions regarding the clusters.

As a result the upper boundary of each cluster is stored in an output file together with the number of keywords contained in the cluster, as well as the percentage of keywords for each cluster. Later the percentage values are used to assign a feature score for each candidate keyword according to the position of its first occurrence in the document. The number of keywords in each cluster is kept to enable learning from the current and all future keyword extractions. After each keyword extraction the counts as well as the percentage values get updated to monitor the most recent extraction results.

3.3.3 Length

Although the length of keywords to be extracted from a document is restricted to a maximum of $N = 4$ in this thesis this should not bias the keyword extraction system's decision of extracting only longer phrases instead of short ones. If that would be the case the system might miss a lot of good short keywords.
To prevent the extraction system from being biased, the length feature is added to the list of features used to find good candidates for keywords. It learns from the length of formerly extracted keywords to make better predictions about the best length of unseen candidates. To achieve this the absolute frequency of each possible N-Gram length is computed and stored in an output file.

3.3.4 Document Frequency

The last piece of information gathered in the training process is the "Document Frequency (DF)" feature. This is not a feature itself but the document frequency is later used to compute the Inverse Document Frequency (IDF) [7] which is quite a common feature for distinguishing the significance of a particular term for a document. I will describe the IDF feature in more detail in the corresponding section of the "Classification" chapter 3.4.4.
For the training of the corresponding feature the number of documents in the training collection as well as N-Gram counts are collected as outlined in algorithm 3.

As with the other two features the training results are stored in an output file in the directory "trainres", the dedicated results directory for the training process.

Algorithm 3 Collection of document frequency

```
1: doccount := 0
2: for all files f ∈ training corpus do
3:     doccount++
4:     for all n-gram ∈ f with n ∈ [1, N] do
5:         if n-gram has been seen before then
6:             count_{n-gram}++
7:         else
8:             count_{n-gram} := 1
9:         end if
10:    end for
11: end for
```

3.3.5 Parameter Tuning

Recall that the classifier used for keyword extraction in this thesis is a log linear model as described in section 2.1. The last thing, the keyword extraction system requires, for filtering out good keywords from a document after collecting all important information from the training data, is to find optimal values for the feature weights λ_i. As has already been outlined in section 2.5 the process of finding optimal lambda weights is called parameter tuning.

Figure 3.3 depicts the parameter tuning process of the keyword extraction system developed in this thesis. The tuning process starts with an initial set of lambda weights which are in this thesis defined to be evenly distributed so each λ_i has a value of $\frac{1}{\#\lambda_i}$. At first a set of candidate keywords is extracted from each file of the tuning set just as it would be done in a usual keyword extraction. This leads to a set of feature scores that will be described in more detail in the following section.
The feature scores together with the initial values for the λ_i are used by a classifier that implements the log linear model to compute a total feature score for each of the potential candidates. To limit the amount of negative examples but still provide a good cross section of all files in the training set, the feature scores are shuffled and the number of negative samples is restricted to the number of positive ones. The remaining total feature scores are then used to compute a sum over all candidates that correspond to pre-labelled keywords denoted as $score_{kw}$ and a sum over all scores of candidates that do not correspond to pre-labelled keywords denoted as $score_{nkw}$.
The goal of the tuning process is to find the optimal parameter values λ_i to minimize the following term

$$\frac{1}{\sum_i \lambda_i \cdot score_{kw_i} - \sum_j \lambda_j \cdot score_{nkw_j}} \tag{3.1}$$

With the new parameter values the classifier is run again on all candidate keywords to compute new scores for each of them, compute the sums of keywords and non keywords like before and again find optimal values for all λ_i. This process is repeated until it converges or for a fixed number of iterations and the final parameter values

are returned. Two different parameter tuning methods were tested in this thesis; both of which were described in section 2.5. The first one is Powell search, the second method is the downhill simplex algorithm.

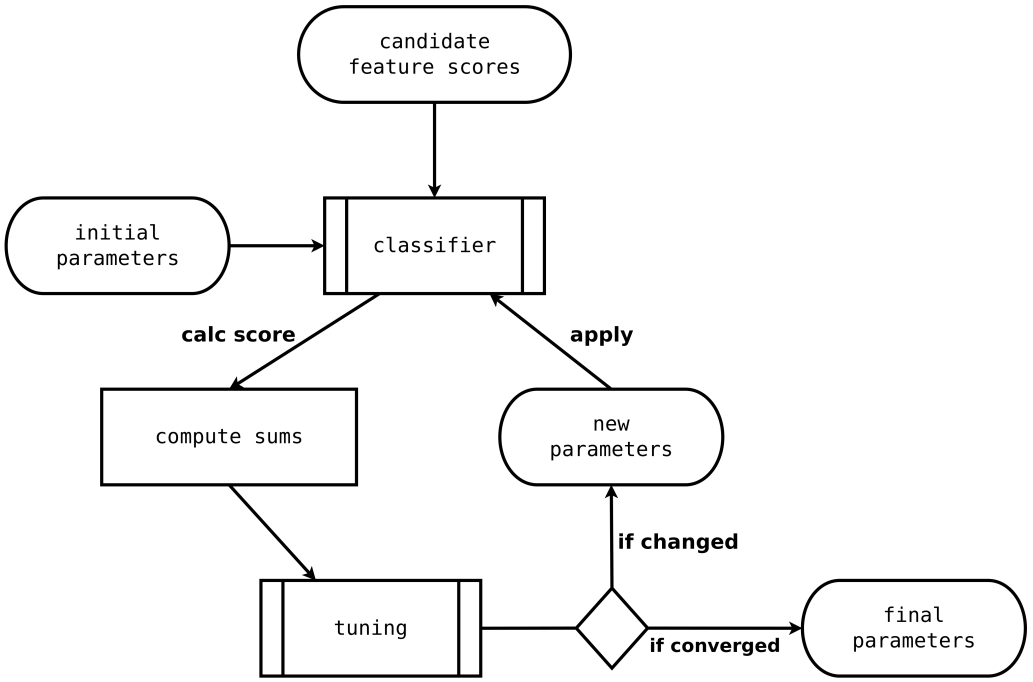


Figure 3.3: Tuning Process

3.3.6 SVM Training

For the Support Vector Machine approach the data used for training had to be of a slightly different form than for the other extraction system. Each extracted N-Gram of the training data represents an individual training sample which is written in one line of the training file that is then given to the SVM. Each training sample consists of a class label it gets assigned depending on whether it corresponds to a pre-labelled keyword or not. Additionally, each sample has a set of feature values assigned to them, which are numbered from one to the total number of features. Each line of the training file is therefore constructed as follows:

<classlabel> 1:<featurevale$_1$> 2:<featurevalue $_2$> ... n:<featurevalue$_n$>

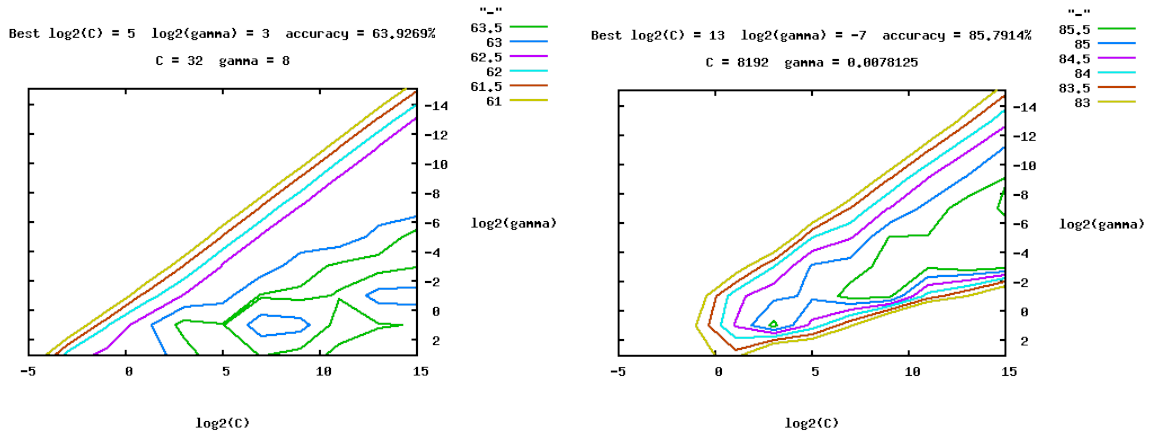
Before the training of a model each feature value is scaled to lie between zero and one to align the potentially different value ranges of the different features. Because the SVM requires each element of the feature vector to be a real number, categorial attributes such as the POS tags have to be converted into a numeric format. As the largest possible N-Gram to be extracted as a candidate is defined to have a length of $N = 4$, each candidate's POS sequence is converted into four numbers. Each of these numbers corresponds to one specific part-of-speech tag, the number zero meaning the current candidate has no part-of speech tag at that position because it has a length shorter than four. A monogram with the POS sequence NN, for example, might be encoded with (12,0,0,0).
Because there are only 20 pre-labelled keywords for each training document, there are much more bad candidates which are no keywords than good ones which correspond to keywords. To attenuate the effect of these skewed classes only every 200th

candidate with a label corresponding to "no keyword" is used for training. This leads to a number of "bad" candidates that is about as big as the number of "good" candidates. Furthermore a limited number of training examples speeds up the training procedure.

The training file of the form just described is fed into an algorithm, the goal of which is to produce a model which is supposed to predict the target values of the test data given only the test data attributes. The test data attributes are of the same format as the training samples. Because the training samples are not likely to be linearly separable the Gaussian Radial Basis Function (RBF) kernel is chosen as the kernel function for the SVM in this thesis. The RBF is one of four basic kernels available with the LIBSVM and also a common kernel used for non-linear classification problems with SVMs. It is described by the following formula

$$K(x_i, x_j) = \exp(-\gamma \parallel x_i - x_j \parallel^2) \tag{3.2}$$

where $\gamma \geq 0$ is a parameter that adapts the smoothness of the similarity function. The kernel is related to the function $\phi(x)$ 2.18 by $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. The first step of the trainings procedure for the RBF kernel is to compute values for its parameters C and γ which are best for the given problem. This is done using a "grid-search" on C and γ with cross-validation on several subsets of the training data to prevent an over-fitting problem. The cross-validation is done by dividing the training set into v subsets of equal size and sequentially testing one subset using a classifier that was trained on the remaining $v - 1$ subsets. The grid-search tests various pairs of C and γ values at the end taking the one with the best cross-validation accuracy. The search takes exponentially growing sequences of C and γ values (e.g. $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$) starting with a coarse grid to identify a better region on the grid, that is then searched with a finer grid. After the best values for C and γ have been found, the whole training set is trained again to generate the final classifier. During the training process the grid-search script plots a contour of the obtained cross-validation accuracy. Example plots, created during the training of the SVM, can be seen in figure 3.4.



(a) Contour 1 (b) Contour 2

Figure 3.4: Example Contour Plots for Cross-Validation Accuracy

3.4 Classification

The classification step is the main step where the actual keyword extraction takes place. As in the training step each file keywords are to be extracted from is first preprocessed as described in section 3.2. After that potential keyword candidates are extracted from each document. As has been mentioned in section 3.1, every N-Gram up to a length of $N = 4$ is considered a potential candidate for a keyword. To reduce the amount of possible candidates, a filtering step is applied to filter out candidates that are a priori not very likely to be good keywords. Such candidates are defined to be the ones that consist mostly or completely of stopwords. For each candidate the number of stopwords is computed and only the ones with a stopword ratio below 65% are kept in the set of possible candidates. The 65%-rule results for example in discarding all 3-Grams that consist of two or more stopwords or 4-Grams that consist of more than two stopwords which seems reasonable. Uni-Grams that are stopwords are not considered candidates at all. The stopword list used in this thesis was the one provided by the NLTK for English language. An additional filtering step is to strip all candidates of punctuation marks. Punctuation marks contain a lot of information in a whole text but for N-Gram chunks of a maximum size of only four they are considered irrelevant in this thesis.

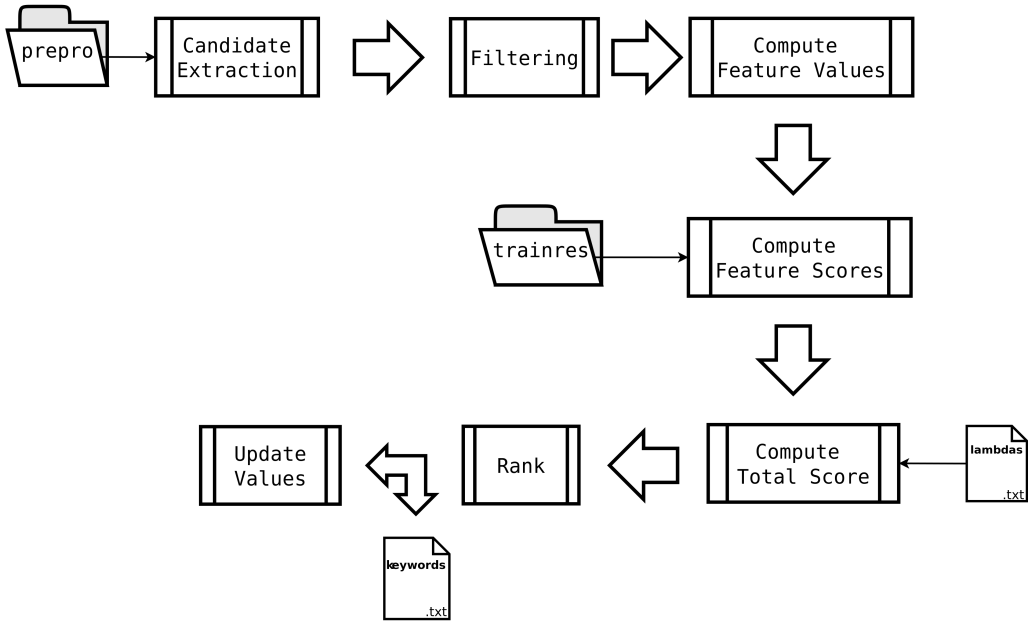


Figure 3.5: Classification Process Overview

Figure 3.5 depicts the classification step and its sub steps. After the filtering steps, feature values are computed for every remaining keyword candidate. From these feature values a feature score is computed for every feature of the candidate. The computing steps for each value will be described in the following sub sections. The feature scores and λ values from the parameter tuning are combined to compute an overall score for each candidate that is later used to rank the candidate and only output the best ones for each document. As with the other keyword extraction steps, the result of the extraction process is stored in an output file.
After the keyword extraction process, the information obtained from the new keywords is fed back into the system, to enable a learning process of the system for future keyword extractions.

3.4.1 POS Sequences

In chapter 3.3 I already noted that in order to get the so called part-of-speech rules from the part-of-speech sequences extracted during training, a filter has to be applied. After applying this filter there are only some of the original POS sequences left that are denoted part-of-speech rules. As these rules are considered to represent characteristics of good keywords they are used to assess new potential candidates.

In the classification step the POS rules are used to predict the probability if a particular candidate is a keyword or not. For that purpose every potential keyword candidate gets assigned its corresponding POS tags. This is done using the function `nltk.pos_tag()` which receives a list of words and returns their corresponding part-of-speech tags according to the Penn Treebank tagset [31]. The assigned part-of-speech tags of a candidate represent its feature value for the POS feature.
In the next step the feature value of each candidate keyword is compared to the POS rules extracted using the filter function. If a feature value matches one of the rules, the corresponding keyword candidate gets assigned the feature score that is associated with the matching rule. The feature scores represent the relative frequencies of the corresponding part-of-speech rule and can therefore be used to predict the probability of a keyword candidate matching that particular rule being a keyword as well.

Feature values that do not match any of the POS rules get assigned a special score for unseen sequences. If a feature value does not match any of the POS rules, it means that, according to the relative frequencies of the other rules, it should be assigned a probability of zero for being a keyword. That however, seems much too restrictive, for it may be possible to encounter a good keyword for a document with a part-of-speech sequence that has not frequently or not at all occurred in the POS sequences of the training keywords. Therefore a special score gets introduced for part-of-speech sequences unseen so far. This score is very small and close to zero to account for the fact that the part-of-speech sequence it gets assigned to did not occur in the POS rule. But it is also greater than zero to allow for the keyword candidate to still be selected as a keyword for the corresponding document.

3.4.2 Document Position

For this feature the system computes a document position distribution over all training documents after dividing each document into a number of clusters. The distribution values from the training are used in this step to compute a document position score for each keyword candidate.
During the extraction of potential keyword candidates from the document, the position of each candidate's first occurrence in the document is computed. As in the training process the document position of a candidate phrase consisting of more than one word is defined to be the position of the first word of that phrase. All candidates that remain after the filtering steps described at the beginning of this section 3.4 are then compared to the position distribution calculated during the training process. Each candidate's position value matches one of the clusters and gets assigned the matching clusters' corresponding score.

3.4.3 Length

As this feature is supposed to prevent the keyword extraction system from being biased to a specific length for keywords, each potential keyword candidate is evaluated according to its length.
The length of each candidate is compared to the lengths observed in the training set or previously extracted keywords. The score that gets assigned to each candidate after this comparison represents the relative frequency of the matching N-Gram length. If the length of the currently examined candidate has not been seen in any of the training keywords or previously extracted keywords, its relative frequency would be equal to zero. To prevent such candidates from getting assigned a zero feature value a small special feature score is used just like in the POS sequences feature 3.4.1.

3.4.4 Inverse Document Frequency

The Inverse Document Frequency (IDF) is a frequently used feature in natural language processing. In 1972 it was first described as a relevant feature for information retrieval by Karen Spärk Jones [7]. The IDF means to assign a weight for each term of a document that depends on the number of occurrences of that term in the document.

A very simple approach to do this would be just to take the number of occurrences of the term in the document, the **Term Frequency (TF)**, and use this as the desired weight. However, this suffers from the problem that all terms in a document are considered equally important if their number of occurrences are equally high thus loosing their discriminating power. If all documents in a collection come from the same domain, it is very likely that certain terms occur frequently in each of them. This makes it impossible to predict if the frequently occurring terms are good keywords for each of the documents or if they just happen to appear very frequently in the corpus (like stopwords).
The IDF comes with two improvements of the above approach: First it uses a document-level statistic for the terms for better discrimination between documents, and second it reduces the effect of terms occurring too often in a collection of documents by scaling down the weight of a term by a factor called the **Document Frequency (DF)**. The IDF of a term w is defined as

$$IDF_w = log\ \frac{N}{DF_w} \tag{3.3}$$

where N is the number of documents in the collection. This equation leads to a high value for terms that only occur in a few of the documents in a collection and a low value for terms that appear in many documents.

During the training process the keyword extraction system gathers the DF for every N-Gram in the collection of documents that is not filtered and counts the number of documents in the collection. Now in the classifying step each candidate phrase of the current document is compared to the N-Grams retrieved of the training documents and an IDF-score is calculated. Because the IDF-scores of all candidate keywords have to pass through an additional processing step, the scores have to be modified a little. In the case that a term occurs in each of the documents of a collection, its IDF would be $log\ \frac{N}{N}$ which is zero. As zero is not a valid value for the next processing step, the IDF has to be modified in that special case to avoid zero values:

$$IDF_w = log\ \frac{N}{DF_w - 1} \tag{3.4}$$

3.4.5 χ^2-Value

The χ^2-Feature is based on the approach of Y. Matsuo and M. Ishizuka [10] and is the only feature used in this thesis that does not require a trainings corpus because it draws all the information it needs from the current document itself.
The feature is based on a co-occurrence distribution of terms in a document with so-called "frequent terms" of that same document. If this distribution is biased to a particular subset of frequent terms Matsuo and Ishizuka conclude that this is evidence for a term being a keyword.

To compute feature scores for keyword candidates a first step is to extract frequent terms from the document. A term in case of this feature is the same as in the rest of the thesis, which means it can be an N-Gram of up to length four. Frequent terms are obtained by counting term frequencies during the extraction of potential keyword candidates and keeping all term with frequencies greater then the highest count times a pre defined threshold.
The next step is to construct a so called co-occurrence matrix. Each entry in this matrix represents the number of sentences in which a term and a frequent term co-occur as shown in table 3.1. In this example co-occurrence matrix term D, and term F co-occur in 18 sentences, term E and term D co-occur in only three sentences. If a term w occurs independently from frequent terms the co-occurrence distribution resembles an unconditional distribution, if the occurrences of w are not independent, the co-occurrence of w with some frequent terms is going to be greater than expected and the distribution gets biased. The method for measuring the degree of bias of a co-occurrence distribution is the χ^2-measure which has been introduced in section 2.4. However, the χ^2-measure gets a little adapted in this feature to also consider varying sentence lengths and make it more robust. The new definition of the χ^2 value of a term w is

$$(\chi^2)'(w) = \chi^2(w) - \max_{g \in G} \left\{ \frac{(freq(w,g) - n_w p_g)^2}{n_w p_g} \right\} \tag{3.5}$$

where G is the set of frequent terms, $freq(w,g)$ is the frequency of co-occurrence of a term w and a frequent term g, n_w is the total number of sentences in the document in which w appears, and p_g is the sum of the total number of terms in sentences in which g appears divided by the total number of terms in the document. This means

	A	B	C	D	E	F	G
A	-	10	2	4	15	11	11
B	3	-	12	12	16	7	3
C	4	5	-	0	13	6	23
D	11	15	5	-	8	18	2
E	1	4	42	3	-	6	3
F	5	3	8	1	2	-	9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
X	9	0	24	3	4	4	9
Y	2	1	0	4	12	12	11
Z	18	1	23	1	3	9	4

Table 3.1: Example Co-Occurrence Matrix

that $n_w p_g$ is the expected frequency of co-occurrence of a term w and a frequent term g.

To prevent the co-occurrence matrix from becoming too sparse, terms corresponding to columns of the matrix are clustered according to two metrics. The first one is similarity-based clustering which clusters terms that have a similar distribution of co-occurrence with other terms. This metric is implemented using the Jensen-Shannon Divergence (JSD) which was introduced in section 2.2. The second clustering metric is pairwise clustering. This metric clusters two terms w_1, w_2 if they co-occur frequently. It is implemented using Mutual Information (MI), see 2.3.
Applying this clustering formula 3.5 changes to its final form for calculating the score for each potential keyword candidate

$$\chi'^2(w) = \sum_{c \in G} \left\{ \frac{(freq(w,c) - n_w p_c)^2}{n_w p_c} \right\} - \max_{c \in G} \left\{ \frac{(freq(w,c) - n_w p_c)^2}{n_w p_c} \right\} \tag{3.6}$$

where c is a cluster.

3.4.6 Extraction of Keywords

When all feature scores are calculated the keyword extraction system uses the lambda values λ_i that where obtained during parameter tuning to compute a total score for each keyword candidate according to the log linear model represented by the following formula

$$score(w) = \sum_i \lambda_i * h_i(w,d) \tag{3.7}$$

Ranking all candidates w according to their obtained scores, the keyword extraction system then returns candidates w with

$$score(w) \leq \min\left(scores(w)\right) * kw\_thr \tag{3.8}$$

where *kw_thr* is a pre-defined threshold that represents a constrained for the quality of the returned candidates. If the threshold is set to be very high, only the candidates with scores close to the best obtained score are returned; if it is low the extraction system also considers candidates with worse scores. An additional constraint for the set of returned keywords is that it does not contain more than 20 elements. This was done to restrict the number of returned keywords in order to give a broad enough overview of what the current document is about but still be concise to not overload the set of topics.
As all documents are preprocessed before keywords are extracted, the final keyword candidates also come in their preprocessed form which might be lower cased, normalized or even stemmed. Because this is no format that reveals what the extracted keywords really are, the original terms have to be extracted from the un-preprocessed original document. To achieve this the keyword extraction system re-uses the document position feature values of the extracted candidates. It simply looks at the position in the original document that corresponds to the position value of a candidate and retrieves the term at that position. For N-Grams with lengths greater than one the $N - 1$ following words are extracted as well. The terms obtained by this procedure are then returned to the user as the final keywords for the current document.

SVM classification

The keyword extraction or classification process of the SVM works a little different from the process of the keyword extraction system developed in this thesis. The model which was trained before is now used to predict a class label depending on the attributes of the test data. Each sample of the test data has to be formatted like the trainings samples as described in section 3.3.6. Class labels may be omitted for the test samples but the SVM model can use them to calculate the accuracy of its prediction if they are provided.
The output of the SVM model is a set of class labels, one for each test sample, which corresponds to the class with the highest probability for a test candidate according to the model. This probability is derived from the distances to the separating hyperplane. In the case of one hyperplane, all test samples lying on one side of the hyperplane are assigned a label for class one and all test samples lying on the other side of the hyperplane are assigned the label of the second class. The SVM does not "rank" the candidates according to scores but predicts the most likely class for each one so it may well be that the classifier predicts more than the original amount of keywords.

4. Experiments and Evaluation

4.1 Experimental Setup

As has already been stated in chapter 3.2, all the data for training, tuning and testing was obtained from the TED[1] website. The TED website offers free videos and transcripts from the best talks of the corresponding conferences in more than 100 languages. All documents used in this thesis were English transcripts with keywords already assigned to them.
Unfortunately the pre-assigned keywords turned out to be rather generic ones corresponding to topics every talk gets assigned by the TED organizers creating the website. As these generic topics rarely match words actually occurring in the transcripts, they could not be used for the automatic keyword extraction system developed in this thesis.
The transcripts used in this thesis therefore were documents with manually assigned keywords that were extracted directly from the text. That way it was ensured that each keyword extracted by the keyword extraction system would have a possibility to be found in the set of keywords that were manually assigned. The actual corpus that was used in this thesis is described in table 4.1. All word and sentence counts are the counts of the partly pre-processed files. This means that they show only the amount of pure text in the respective files, blank lines, timestamps and other overhead that is not considered in this thesis already having been removed from the raw transcripts. Other pre-processing steps such as tokenizing, lower casing or the re-expansion of short forms had not been applied at that time.

Corpus	**# Documents**	**# Sentences**	**# Words**
Training	31	10687	101162
Tuning	5	1532	13547
Testing	5	1681	14770

Table 4.1: Corpus information

[1] http://www.ted.com/talks

4.2 Results

In this section I am going to describe experiments that were performed to receive the optimal threshold for feature values. Additionally I am going to describe results obtained by experiments with different tuning functions, feature settings and threshold values.

4.2.1 Obtaining POS Rules

In chapter 3.4.1 I mentioned a filter function for POS tags. This filter function gets applied to the part-of-speech tag sequences that were extracted during the training procedure in the classification step to obtain the so-called part-of-speech rules. In order to find a good filter function that is not too restrictive but still informative enough to give accurate predictions about good keywords, some experiments were performed on the POS sequences extracted during training.
Intuitively one would want only those sequences extracted for POS rules that have a maximum count of occurrence among the manually labelled keywords. Sequences with very low counts are not very likely to be particularly meaningful compared to the ones with large counts.

One possibility to filter out part-of-speech sequences with high counts might be to apply a relative threshold *thr* to the highest occurring count of all sequences. Applying this threshold retrieves all POS sequences with

$$count_{POS\_seq} \geq highest\_count \cdot thr \tag{4.1}$$

Table 4.2 shows the POS sequences that were extracted using several different values for *thr* and their respective counts. There are only a few POS sequences that occur quite frequently; this requires a rather low threshold in order to have more than one POS rule for keyword extraction. As would be expected, the most frequently occurring POS sequences contain nouns in either their singular or plural form. This is fine as nouns can generally be expected to be good keywords. But in order to allow other POS tags in the rules that not only contain noun tags and determiner tags, a different filter method is probably better because a threshold that is too low looses its discriminative power.

Another method could therefore be to simply take the n most frequently occurring POS sequences as rules. This method however is not at all adaptive to changes in the counts of the part-of-speech sequences. If the counts got more equal over time due to information gathered through new keywords, this change of mass distribution would not be reflected in the extracted POS rules. The extracted amount would not change even if in the new situation a not extracted POS sequence would be just as likely to represent a good keyword as an extracted one.
A further possibility that does, however, not have this issue could be to apply an absolute threshold to the scores each POS sequence gets. This score represents the frequency of a part-of-speech sequence relative to the total amount of POS sequences. It could be argued that sequences with a score smaller than 0.01 do not have enough discriminative power because their score is less than one percent of

Value of *thr*	POS Sequences	Count
0.75	NN	151
0.4	NN	151
	NNS	62
0.3	NN	151
	NNS	62
0.25	NN	151
	NNS	62
	NNP	44
	NNP NNP	40
0.2	NN	151
	NNS	62
	NNP	44
	NNP NNP	40
	JJ NN	35
	NN NN	32

Table 4.2: POS rules and their counts at different values of *thr* and 31 training documents

the total probability mass of all POS sequences. Therefore the third possibility of applying a threshold for extracting POS rules is to take only POS sequences with

$$score_{POS\_seq} \geq 0.01 \tag{4.2}$$

This threshold is not as restrictive as the first or second method and results in more extracted POS rules which promotes a greater variance among keywords. Table 4.3 shows the extracted rules using this threshold approach with 31 training documents. It also allows for adapting over time as the score of each POS sequence gets calculated again every time new keywords are extracted from a document.

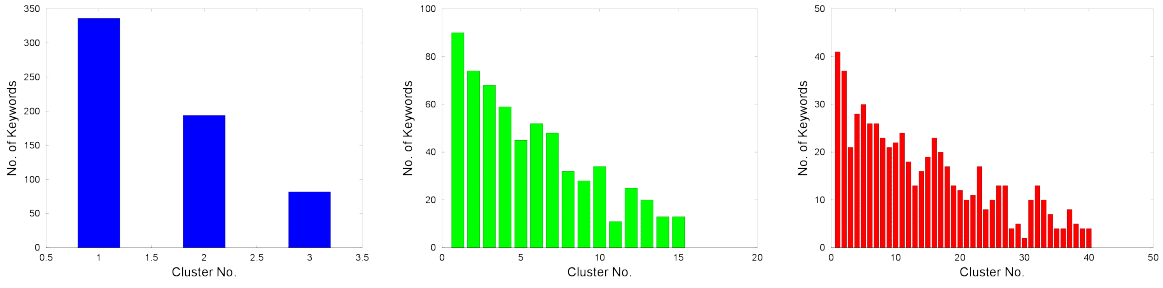
4.2.2 Document Position Clusters

The document position feature is supposed to represent a distribution of the position of keywords in a document that can later be mapped to an unseen document. This mapping supports the prediction whether a candidate term is a keyword for the current document or not. But in order to be a good support and provide valuable information for the classifier, the clusters the document is divided into have to be fine-grained enough to allow for precise conclusions.
If the document was subdivided into only three clusters for example, the distribution of keywords in the clusters would only allow for rather rough predictions. On the other hand, if the clusters get too small only a very small fraction of keywords would be left in each cluster, which would make predictions for individual clusters very weak. Therefore it is desirable to find a number of clusters that provides a fine grained enough subdivision of the document while having large enough clusters to make meaningful predictions.
Figure 4.2 shows the distribution of keywords in the training documents at different

POS Sequences	Score
NN	0.243
NNS	0.1
NNP	0.071
NNP NNP	0.065
JJ NN	0.056
NN NN	0.052
NN NNS	0.037
JJ NNS	0.026
NNP NNP NNP	0.015
VBG	0.015
JJ NN NN	0.013

Table 4.3: POS rules with score greater than 0.01

numbers of clusters ranging from one to 53. At a number of 54 clusters some clusters started to have no keywords in them, which is defined to be the cluster threshold as described in section 3.3.2.
As can be seen in figure 4.2, the keywords tend to concentrate roughly in the first half of the document. Although the number of keywords per cluster decreases with increasing number of clusters as expected, the number of keywords in the last quarter of the document decreases much more dramatically as in the first quarter of the document. Furthermore at a number of about 15 clusters it can be seen that the number of keywords in the last third of the document begins to decrease significantly, producing the dark band in the upper third of the figure. Because of this change in decrease, the number of 15 clusters is considered a good threshold and so the number of clusters each document gets subdivided into is set to this value. Figure 4.1 gives a more detailed overview about the distribution of keywords with 3, 15 and 40 clusters.



(a) Three Clusters (b) 15 Clusters (c) 40 Clusters

Figure 4.1: Keyword distribution over clusters with different numbers of clusters

4.2.3 Filtering candidates

Discarding N-Grams that are stopwords or mainly consist of stopwords was one first method to limit the search space for potential keyword candidates. Other methods were included to also get rid of those N-Grams that end with a stop word, because it did not seem likely that those would be good candidates for keywords. A third step was to get rid of candidates that only appear once in the document, because

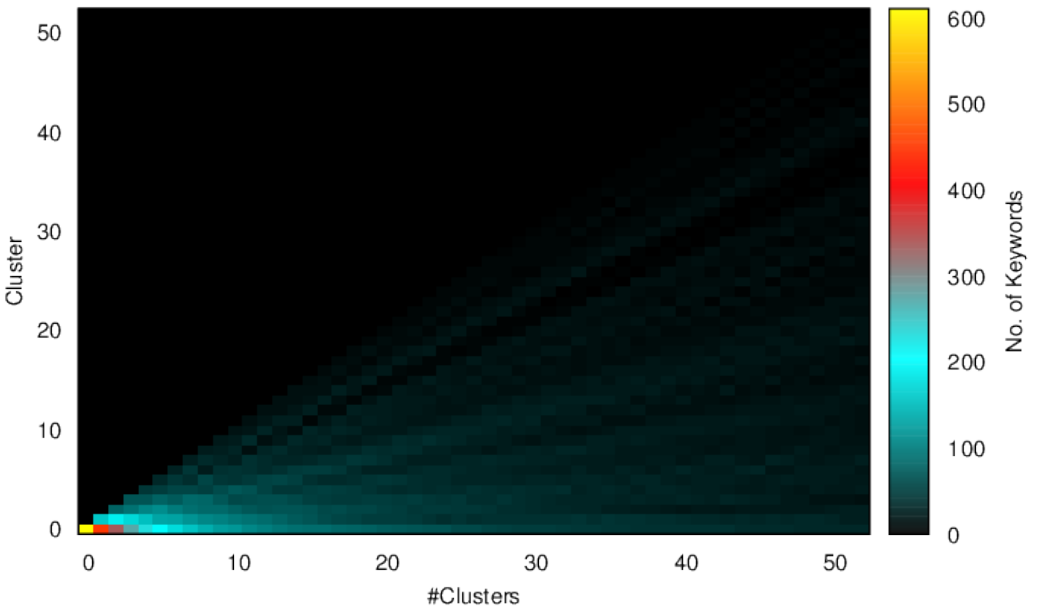


Figure 4.2: Number of keyword per cluster with different numbers of clusters

keywords that represent the content of the document and that are important are likely to appear rather frequently. The last cleaning method that was tried was not only to discard candidates that consisted of mainly stopwords or punctuation marks but to completely get rid of N-Grams that contained any punctuation. This proved to limit the amount of potential candidates, thus reducing the search space but without effecting the oracle score as can be seen in tables 4.4 and 4.5. The oracle score is a measure for the number of target keywords that can be found in the extracted candidates.

Steps two and three proved to be too restrictive for the TED data that was used in this thesis, as the oracle scores the system was able to achieve dropped after applying them. If the reduction of the search space is too restrictive, good candidates might

	Number of Candidates
STEM, with punctuation	21636
STEM, no punctuation	20272
TOKLOW, with punctuation	21711
TOKLOW, no punctuation	20429

Table 4.4: Number of candidates with and without punctuation

be thrown away even before classification, preventing the keyword extraction system from classifying them as good candidates.
I tested the filter methods with stemmed data, denoted as STEM, and N-Grams that were only tokenized and lower cased, which are denoted as TOKLOW. As can be seen in table 4.5 the method of discarding N-Grams that occurred less than two times in the document proved to be much too restrictive, producing an average oracle score of 0.58. The average oracle score is to be interpreted as the sum of the oracle scores for all test documents divided by the number of test documents. An oracle score of 0.55 would mean that only 11 out of 20 keywords remain in the candidate set after applying the filter rule. So a score of 0.58 over five documents means in all five test document only about half of the target keywords could be found by the keyword extraction system. Skipping only those candidates that ended with a stopword proved to have no effect at all on the stemmed data and only a small one on the TOKLOW data.
The best results, however, were obtained by not reducing the search space apart from discarding stopwords themselves and N-Grams that mainly consist of them, so this was the approach that was followed for the rest of the experiments. Although omitting N-Grams that had stopwords at the end did neither prove to improve the oracle score nor reduce it significantly, the method was not pursued during the rest of the experiments because the stopword filter was not very reliable. This is why I decided to rather leave the candidates it would have filtered out in the candidate set.
The fact that discarding rare terms had such a severe impact on the oracle scores of the system can maybe be explained by the transcript nature of the documents used in this thesis. In spoken language people often tend to use synonyms instead of always repeating the same term over and over. This usually makes a speech more diversified and sophisticated but as the keyword extraction system does not know about the shared context and synonymous character of different terms it treats them as different candidates, each of those getting a lower count than the accumulated synonyms.

4.2.4 Baseline

Of all the features that were proposed for keyword extraction in this thesis the TF-IDF and χ^2 feature are the most discriminative ones, when considered individually. All the other features mainly provide additional evidence for the nature of a keyword candidate. The χ^2 feature was already used as a sole indicator for keyword extraction in the work of Y. Matsuo and M. Ishizuka [10] and the TF-IDF measure is a very common method to find words that are discriminative for one document as compared to others. Therefore experiments using only the χ^2 feature and only the TF-IDF feature are defined as the baseline for other experiments. As with the filtering of potential candidates that was described in the previous section all experiments were conducted with stemmed data as well as texts that were only tokenized and lower cased.
The average results for precision, recall and F-score of the two baseline approaches can be seen in tables 4.6 and 4.7. Both the χ^2 and the TF-IDF baseline approach achieved rather low values in all three categories. In the χ^2 experiments both the extraction system developed in this thesis and the SVM achieved slightly higher precision and F-score values for the data that was only tokenized and lower-cased

Filter method	Oracle Scores
STEM, with cleancounts	0.58
STEM, no stopwords at end, no cleancounts	0.93
STEM, no cleancounts	0.93
STEM, no punctuation	0.93
TOKLOW, no cleancounts no stopwords at end	0.97
TOKLOW, no cleancounts	0.98
TOKLOW, no punctuation	0.98

Table 4.5: Filter methods and achieved oracle scores

than for the stemmed texts. Neither the SVM approach nor the one developed in this thesis, however, were able to achieve results that could reach those Matsuo and Ishizuka reported in their work, although they only reported about precision. The reason for this lies most probably in the different nature of the input data. While Matsuo and Ishizuka worked with scientific papers the TED talks used in this thesis are transcripts of spoken language so the rules Matsuo and Ishizuka applied successfully to their data may not apply as well to the TED talks.
The results for the baseline TF-IDF approach behave quite similar to the ones obtained with the χ^2 feature although here the stemmed data achieved better results on both systems than the lower-cased. This is most probably due to more stable counts for individual terms caused by stemming. When terms get stemmed the term frequency counts from every surface form of a term get accumulated to the stem of the term. These accumulated counts allow the TF-IDF measure a better discrimination between terms than was possible with the separate individual term frequencies. The fact that precision, recall and F-score always have the same value in all experiments conducted with my own system is due to the keyword extraction system always extracting the best 20 candidates as keywords and discarding the rest. In this set of 20 keywords every hit is a true positive, the rest of the extracted candidates are false positives but automatically every target keyword that was not hit becomes a false negative. Due to the target keyword set and the set of extracted keywords being equal the number of false positives is always equal to the number of false negatives thus making precision and recall equal. Because precision and recall are weighted evenly the F-score is then equal to precision and recall as well.

One additional improvement was made to the extraction process of the system developed in this thesis. With this improvement the system now only extracts a candidate as a keyword if it is not contained in a candidate extracted before. If one candidate is completely contained in another only the one with the higher score is used as a

	SVM, stemmed	**Powell/ Simplex, stemmed**	**SVM, tokenized, lower-cased**	**Powell/ Simplex, tokenized, lower-cased**	**M & I**
Precision	0.0349	0.09	0.0411	0.1	0.51
Recall	0.3717	0.09	0.1521	0.1	-
F-score	0.0635	0.09	0.0641	0.1	-

Table 4.6: Average precision, recall and F-score for baseline χ^2 approach

	SVM, stemmed	**Powell/ Simplex, stemmed**	**SVM, tokenized, lower-cased**	**Powell/ Simplex, tokenized, lower-cased**
Precision	0.0971	0.19	0.0877	0.14
Recall	0.4550	0.19	0.4600	0.14
F-score	0.1184	0.19	0.1156	0.14

Table 4.7: Average precision, recall and F-score for baseline TF-IDF approach

keyword, the one with the lower score is discarded and the next one in the list of ranked candidates is considered. Table 4.8 shows the baseline results obtained from keyword extraction with this approach. Compared to the results obtained without the improvement it can be seen that the score got better with TF-IDF as a feature but did not change at all or got even a little worse for χ^2. The increase in the scores of the TF-IDF feature can be explained by candidates being very similar to each other or even being part of each other having similar TF-IDF scores. If containments are omitted due to the new improvement new candidates with lower TF-IDF scores can get extracted as keywords which would have been discarded before. As keywords are not likely to contain each other, throwing away candidates that are contained in others makes the extracted keywords more divers and thus raises the probability to hit a good candidate.

A similar improvement to the one explained above was also added to the SVM approach. Originally, the SVM would apply one class label, in this thesis namely either "keyword" or "no keyword", to each considered candidate. Particularly, each candidate whose feature scores exceed the threshold which is defined by the hyperplane constructed during training, is classified to be a keyword. This extraction method is very likely to produce a high number of false positives which is most likely the reason for the inferior results of the SVM as compared to the approach developed in this thesis.

To reduce the number of false positives produced by the SVM, two additional filtering steps are applied to all candidates that are classified as a keyword by the original classification process. At first the candidates are ordered according to their distance to the hyperplane, assuming that the candidates with the largest distances are the best keyword candidates. Based on this ordering, the extraction system is now able to extract only the top 20 candidates with the greatest distances to the

hyperplane separating the two classes "keyword" and "no keyword". Additionally, it is possible to consider the containment of one candidate in another, and extract only those candidates that are not part of each other, just as described for the log-linear approach. As can be seen in table 4.9, the results of the SVM are now equal to those of the extraction system developed in this thesis for the χ^2 feature. The F-scores for the TF-IDF feature are significantly higher, compared to those in table 4.7 but are still lower than the results using the raw TF-IDF values. This is most likely due to transformations of the data caused by the use of RBF-kernel.

	χ^2, stemmed	**TF-IDF, stemmed**	**χ^2, tokenized, lower-cased**	**TF-IDF, tokenized, lower-cased**
Precision	0.09	0.22	0.08	0.16
Recall	0.09	0.22	0.08	0.16
F-score	0.09	0.22	0.08	0.16

Table 4.8: Precision, recall and F-score for improved baseline TFIDF and χ^2 approach on system developed in this thesis

	χ^2, stemmed	**TF-IDF, stemmed**	**χ^2, tokenized, lower-cased**	**TF-IDF, tokenized, lower-cased**
Precision	0.09	0.24	0.08	0.17
Recall	0.09	0.16	0.08	0.11
F-score	0.09	0.18	0.08	0.13

Table 4.9: Precision, recall and F-score for improved baseline TFIDF and χ^2 approach for the SVM

4.2.5 Experiments

Based on the baseline experiments several experiments were conducted at first on the SVM because support vector machines are supposed to be very robust even on complex data. As in the baseline, all experiments were conducted on stemmed data as well as only lower-cased data. Furthermore, two different methods of pruning candidates before classification were used. The first one was the one already described in section 3.4. Only those candidates are considered in the final classification step that have a stopword ratio smaller than 65%. The set of words considered to compute the stopword ratio consists of stopwords contained in a special stopword list, as well as punctuation marks. The second method consists of two parts, the first of which is almost identical to the first pruning method although this time punctuation marks are not considered in the calculation of the stopword ratio. Candidates containing punctuation marks are filtered out separately in a second step in addition to the calculation of the stopword ratio. The first and second cleaning method are referred to as "clean 1" and "clean 2" for the rest of this thesis.

Table 4.10 shows the results of two experiments conducted on the SVM. In both experiments all available features were used and the number of document position

clusters was 15 as defined in section 4.2.2. Experiment **A** was conducted using the method *clean 1* for candidate cleaning while in experiment **K** the method *clean 2* was used. The columns labelled "All terms", show the results considering all candidates that are classified to be a keyword by the SVM, while the columns labelled "Top 20" show the results using the improvement described in the previous section. Again, it can be seen that the SVM was able to achieve significantly higher results using the improvement than considering all positive candidates. In both experiments the SVM achieved higher scores with stemmed data than with the lower-cased candidates but the system set-up using the *clean 2* method was always inferior to the *clean 1* method. Although discarding candidates containing punctuation marks seems like a rational and useful approach, the alteration of the calculation of the stopword ratio seems to have a negative effect. The system set-up using all available features achieved the best results of all experiments beating the baseline results of both the SVM and the extraction system using a log linear approach for stemmed as well as only tokenized and lower-cased data. This shows that adding additional features to the baseline TF-IDF and χ^2 features provided valuable information for the SVM that allowed for a better separation of the test data.

	A: All features, clean1		**K: All features, clean2**	
	All terms	**Top 20**	**All terms**	**Top 20**
stemmed				
Precision	0.1501	0.2629	0.1731	0.2538
Recall	0.4359	0.2300	0.3384	0.17
F-score	0.1778	**0.2428**	0.1607	0.1904
tokenized, lower-cased				
Precision	0.0931	0.16	0.0851	0.1586
Recall	0.4342	0.16	0.2647	0.14
F-score	0.1383	0.16	0.1022	0.1478

Table 4.10: Average precision, recall and F-score on stemmed and lower-cased data using a SVM

Because the baseline experiments with the χ^2 and TF-IDF features gave rather opposing results on the individual test documents additional experiments were conducted omitting the χ^2 and TF-IDF feature respectively. The results of these experiments are shown in table 4.11. As can be seen the test results without using the χ^2 feature are the same as the results of experiment **A** for stemmed data. This indicates that the χ^2 feature does not provide any additional information for the keyword extraction system and may be omitted without reducing the F-score. This is also congruent with the low baseline results of the χ^2 value using both the SVM and the log-linear approach developed in this thesis. Omitting the TF-IDF feature, however, leads to a significant reduction of the F-score for both stemmed and tokenized data. This indicates that the TF-IDF feature is probably one of the most discriminative features for the input data used in this thesis. The experiment denoted with **O** also omitted the χ^2 feature but used the candidate cleaning method *clean 2*. Just as in experiments **A** and **K** scores were reduced using this method of candidate cleaning. Table 4.12 depicts the results of SVM experiments with additional feature settings. Omitting each of the other features leads to a decrease

in the resulting F-score which indicates that each feature provides some additional information that helps the SVM to identify good keyword candidates. The TF seems to provide the least helpful information on its own which is probably due to it being already contained in the TF-IDF feature. This is also congruent with the results omitting the IDF.

	No χ^2, clean 1	**No TF-IDF, clean 1**	**No χ^2, clean 2**
stemmed			
Precision	0.2629	0.04	0.03
Recall	0.23	0.04	0.03
F-score	0.2428	0.04	0.03
tokenized, lower-cased			
Precision	0.05	0.01	0.01
Recall	0.05	0.01	0.01
F-score	0.05	0.01	0.01

Table 4.11: SVM experiments omitting the χ^2 and TF-IDF feature

Table 4.14 shows the pre-labelled keywords of one example test text which is denoted as *Text 3* and depicts which of the pre-labelled candidates have been classified correctly as keywords by the SVM in two different experiments. These are both experiments considering all candidates that had been labelled a keyword by the SVM and not taking only the ones with the greatest distance to the separating hyperplane. As can be seen, the SVM was able to classify more than half of the pre-labelled candidates correctly in experiment **A** and extracted at least parts of some other candidates, for example "Boris Nikolayevich" which is part of the complete name "Boris Nikolayevich Kirshin" and "financial" which is part of the keyword "financial police". Table 4.13 depicts the total number of candidates the SVM classified as keywords for each test document in those experiments. The total number of extracted keywords for Text 3 was higher in the experiment using the *clean 1* approach than in the experiment using the *clean 2* approach. This could suggest a larger number of false positives and thus smaller scores but the corresponding F-scores show that the *clean 2* approach did not have a positive effect on the SVM's performance. A look at

	No POS	**No DOC-pos**	**No Len**	**No TF**	**No IDF**
stemmed					
Precision	0.19	0.19	0.14	0.26	0.21
Recall	0.16	0.16	0.14	0.23	0.20
F-score	0.17	0.17	0.14	0.24	0.20
tokenized, lower-cased					
Precision	0.09	0.12	0.13	0.14	0.14
Recall	0.09	0.12	0.13	0.13	0.14
F-score	0.09	0.12	0.13	0.13	0.14

Table 4.12: SVM experiments with different feature settings

table 4.14 shows that indeed the number of correctly classified candidates decreased in experiment **K**, suggesting that either good candidates got pruned away by the different cleaning method or that this cleaning method had a negative effect on the training process of the SVM where a separating hyperplane for the training data is chosen. Choosing a different hyperplane than in experiment **A** most likely caused the misclassifications in experiment **K** that led to the inferior results. The low precision and F-score values of experiment **K** with lower cased data can be explained looking at table 4.13 which shows quite a high number of candidates labelled as keywords in this approach. This high number of candidates however also produces a high number of false positives which results in a lower precision an thus lower F-score. Additionally the SVM seemed to have problems extracting candidates with lengths greater than $N = 2$. Of the keywords with lengths three or four, only parts were extracted. Considering only the top 20 candidates results, of cause, in a lower number of false positives. But the average top score of only 0.2428 shows that still only every fourth reference candidate is matched by the candidates extracted from the SVM.

	Clean 1, stemmed	**Clean 2, stemmed**	**Clean 1, tokenized, lower-cased**	**Clean 2, tokenized, lower-cased**
Text 1	16	7	71	18
Text 2	149	208	198	548
Text 3	166	96	260	127
Text 4	12	7	30	20
Text 5	35	33	104	39
SUM	378	351	663	752

Table 4.13: Number of extracted keywords for stemmed and lower-cased data using all available features and different cleaning methods

The high number of false positives considering all candidates, and a top average score of 0.2428 considering the top 20 candidates, indicate that the features selected in this thesis do not have enough discriminative power to clearly separate the good candidates of the input data from the bad ones. To corroborate these suspicions about the nature of the input data I analysed the feature scores individually as well as in combination.
Figure 4.4, for example, depicts the distribution of feature scores for the features χ^2, TF-IDF and document position for good candidates that should be classified as keywords and bad candidates that should not be classified as such. As the TF-IDF and the χ^2 are considered the two most discriminative features in this thesis, a scatter plot of the combination of these two features was made as well. Figure 4.5 shows this scatter plot of good and bad candidates and their corresponding TF-IDF and χ^2 scores. As can clearly been seen, in both figures, unfortunately, the majority of all distributions covers the same value range. This could indicate that the majority of good candidates have the same feature scores as bad candidates, which would make it very hard for the SVM to separate them correctly. The distributions of the other features look about the same as those depicted in figure 4.4. As the TF-IDF and the χ^2 are considered the two most discriminative features in this thesis the SVM

would be confronted with a very complex problem trying to separate good and bad candidates based on the given features.
As the analyses described above only focus on only one or two features it may well be that the the input data is much more clearly separable once the other features are considered as well. To analyse the behaviour of the feature scores when all available features are considered, a PCA was conducted. Unlike the results depicted in figure 4.5 or figure 4.4 the PCA takes into account all available features. The features are projected into a lower dimensional space, 2D in this case, to make them plottable. This is done by projecting the features into the two most discriminating dimensions. Figure 4.3 depicts the resulting distribution of feature scores for good and bad candidates after the PCA. Unfortunately the good and bad candidates can be seen to still be concentrated in the same value ranges apart from some few exceptions.

The results of these analyses, together with the test results of the keyword extraction indicate that the provided features do have enough discriminative power to enable a clear separation of good and bad candidates on the input data. Therefore, the SVM is only able to correctly extract some candidates and produces a lot of false positives if not restricted to the top 20 candidates. For the keyword extraction system developed in this thesis this problem would be even more severe. As has been described in section 2.1 the individual features are combined using a log-linear model that weights each feature with a value λ and produces an overall score for each possible candidate. In order to find the optimal λ values, a parameter tuning has to be performed. Possible methods for parameter tuning have been described in section 2.5. With the optimal λ values the keyword extraction system is supposed to assign high scores to good candidates and lower ones to bad candidates, thereby separating the possible candidates into two different sets. However, if both good and bad keyword candidates show similar distributions of their feature values as is indicated in figure 4.3 it is very difficult to find values for the λ_a that boost good candidates while lowering the scores of bad candidates at the same time. Because it has not been possible in the context of this thesis to find a function that could be optimised by either powell search or the simplex algorithm to achieve reasonable results on the provided test set given the features, no experiments could be conducted to test the selected features on the extraction system developed in this thesis. The baseline results, however, show results similar to those of the SVM, which indicates that the classification approach suggested in this thesis could provide reasonable results if connected with either a different parameter tuning approach or a good target function for parameter tuning for Powell search or the simplex algorithm.

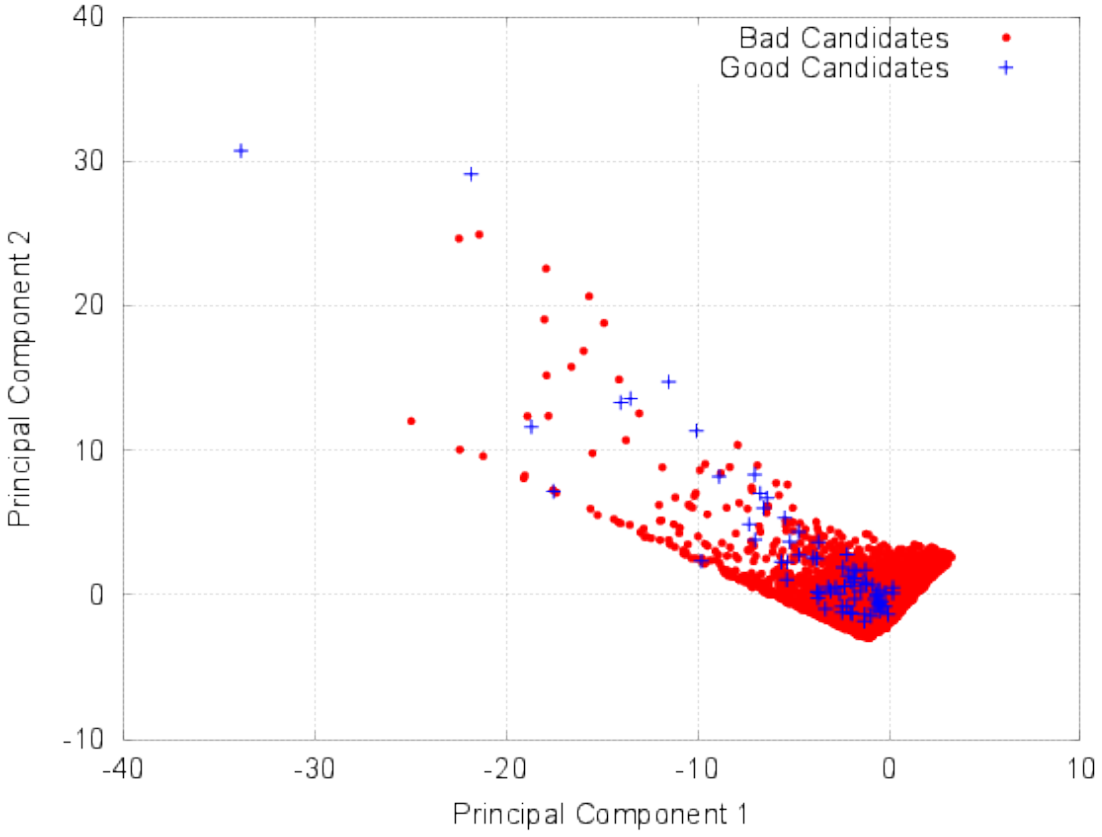
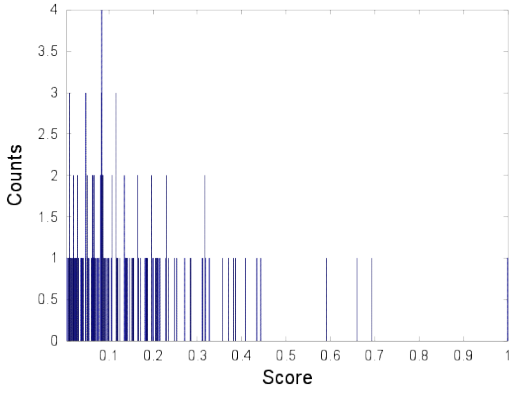


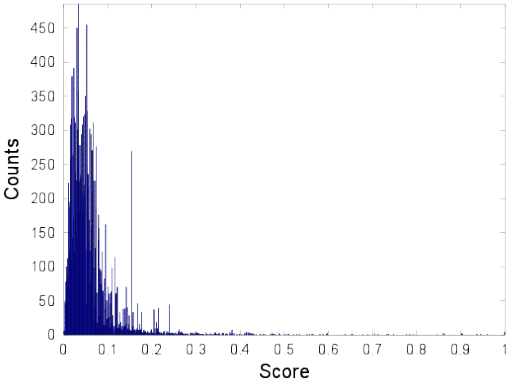
Figure 4.3: Distribution of good candidates and bad candidates after PCA

Reference	**All features, clean 1, stemmed**	**All features, clean 2, stemmed**
independent media	✓	✓
information	-	-
press freedom	✓	✓
B92	✓	✓
text control	-	-
financial police	financial	financial
advertisers	✓	✓
media companies	✓	✓
George Soros	-	-
Slovakia	✓	-
management systems	-	-
financing	✓	✓
loans	-	-
equities	✓	-
lease	✓	-
Boris Nikolayevich Kirshin	Boris Nikolayevich	Boris Nikolayevich
media management center	media	media
fundraising	✓	✓
bonds	✓	✓
marketplace	-	-

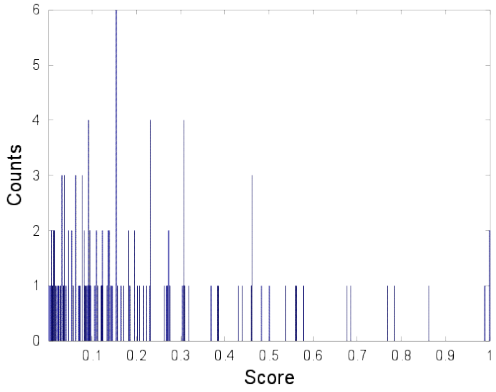
Table 4.14: Reference and hypothesis keywords for two experiments, considering all candidates labelled as keywords



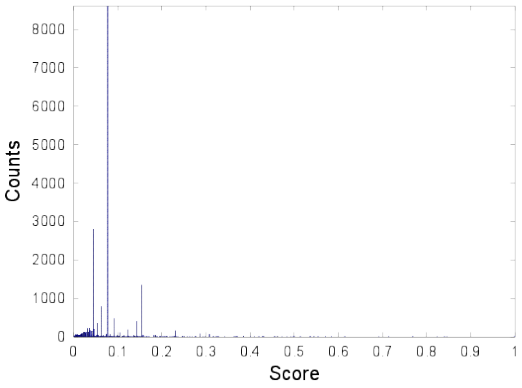
(a) χ^2 Scores



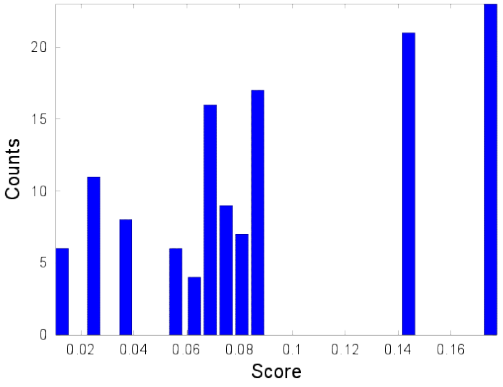
(b) χ^2 Scores



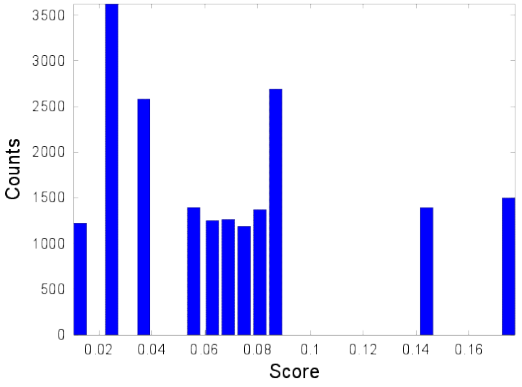
(c) TF-IDF



(d) TF-IDF Scores



(e) Document Position



(f) Document Position

Figure 4.4: Distribution of feature scores for good (left figures) and bad (right figures) candidates for different features.

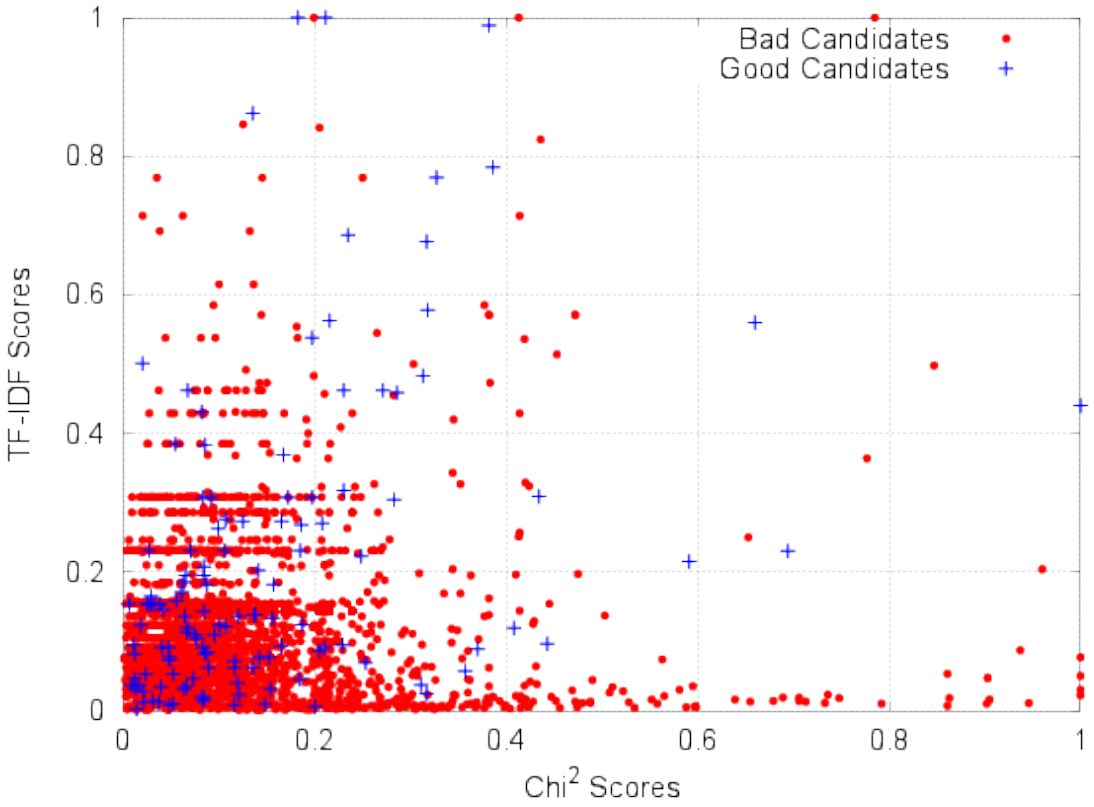


Figure 4.5: Distribution of good candidates and bad candidates with different TF-IDF and χ^2 scores

5. Conclusions and Future Work

In this thesis I presented my work on the task of comparing different keyword extraction approaches on transcribed texts based on TED talks. One of the presented approaches was a keyword extraction system that was developed in the context of this thesis which is based on a log-linear model. The approach was supposed to be tested with two different algorithms for parameter tuning applied. The other approach was the use of an SVM with a RBF kernel to classify keywords in a set of candidates.

Both keyword extraction approaches introduced in this thesis use a combination of different features as a source of information about potential candidates. The extraction system developed in this thesis weight each feature, computes an overall feature score for each candidate and extracts the candidates with the highest scores as keywords for the corresponding document. Candidates with scores lower than a certain threshold are not considered good keywords. The features that were used for keyword extraction were the TF-IDF, the position of candidate terms within the corresponding document, the POS sequence of each candidate term, the IDF, their χ^2 values, the TF and the length of the candidates. Comparison values and other important values for each feature except the χ^2 were gathered during a training step. The document position of the candidate terms was not used directly as a feature for the classification process but clusters were computed to provide more stable counts for the document positions than the individual values would have provided. The POS sequences gathered during the training process were used to generate the so called POS rules which were composed of those POS sequences that were seen in the training examples and that exceeded a certain threshold. Although no experiments with all available features could be conducted on the extraction system developed in this thesis the baseline experiments provided promising results which indicate that good results could be achieved with different parameter tuning algorithms or a good target function for powell search and the simplex algorithm.
Different cleaning methods have also been tested on the candidates, some of them proving to be too restrictive like the one discarding candidates with term frequencies smaller than three. The method discarding candidates containing punctuation marks proved to reasonably reduce the search space without effecting the oracle score.

However experiments later proved for this cleaning method to have a negative effect on the F-scores of the test set.

The SVM was trained with the same features as input, although the POS sequences were specifically encoded to fit the data format of the SVM. Furthermore no thresholds were used for the POS sequences in the SVM experiments, therefore experiments including changes in the POS threshold were not conducted on the SVM.
In a first set of experiments all candidates that were classified to be a keyword by the SVM were also considered as keywords. Naturally this produced a high number of false positives as the SVM was not restricted to an upper bound for keyword extraction. Therefore, an improvement was applied that, in a first step, sorted the positive candidates according to their distance to the separating hyperplane and in a second step filtered out candidates that were contained in other candidates. Using this improvement, the SVM was able to achieve significantly higher results that outperformed the baseline results of both main approaches used in this thesis. Unfortunately, the SVM still had problems with correctly classifying longer keywords (with length $N \geq 2$). Of many of those candidates only parts were classified as keywords. Comparing the distributions of scores of good possible candidates with the distributions of scores of bad possible candidates for different features showed that the majority of scores have a very similar range of values. Scatter plots depicting the combination of the two baseline features as well as the results of a PCA of the training data also supported the hypothesis that the input data consisting of transcribed TED talks is very hard to separate given the features selected in this thesis.

5.1 Future Work

The results of the experiments conducted during this thesis indicate that the features used to discriminated between keywords and non keywords were not discriminative enough to allow for an accurate separation of good and bad candidates. The use of only structural features may not have been sufficient for the data that was used in this thesis.

One step of future work should therefore include finding more discriminative features to aid the classification of keywords. These features should probably also contain semantic information apart from only structural information. To compensate the high use of synonyms in spoken language, lexical databases like *WordNet*[1] could be used to enable the keyword extraction system to treat synonyms as one word instead of individual ones.
An additional source of information could possibly also be provided by *Wikipedia*[2]. Comparing possible candidates to Wikipedia topics could be one way distinguish important terms in a document from the ones that are not as important. Furthermore cross links between Wikipedia topics could provide information about terms that are somehow related to one another. This could help to facilitate the keyword extraction systems understanding of synonyms similar to the approach with WordNet. It could also help to build clusters of important terms that are related to each other thereby pushing one term of the cluster if the score of another member of the

[1] http://wordnet.princeton.edu
[2] http://en.wikipedia.org/wiki/Main_Page

cluster gets boosted by some feature that might not effect the first term as much as the second. Another additional feature could probably be created by gathering information about the casing of a term before applying the lower-casing. If a term is only contained in a capitalised form in a document, it might be a proper noun or the name of a person which could be an indicator for it being a keyword.
Using POS sequences in a different way than the one in this thesis could also help to increase the value of this feature. One different way to use POS tags might be to not only use the part of speech sequences that were seen in a training set as POS rules, but to make them a basis for manually creating POS rules. These rules could then also get individual weights based on the preferences of either the person who designs the rules or a specific context, keywords are to be extracted for. Such a context might be preferring a focus on noun phrases with a minimum of two terms or verb phrases. These could be explicitly preferred with custom made POS rules. Another approach would be to assign equal weights to all rules as opposed to using the relative frequency as it was done in this thesis. More general improvements could include additional pruning steps of possible candidates to further reduce the search space. Creating a more sophisticated, custom-made stopword list could help to filter out candidates containing stopwords. Additionally, creating "negative" POS rules could help to prune the list of possible candidate more generally than looking for specific stopwords. Another possible pruning step could include discarding candidates that contain numbers, although one could argue that important dates might represent good keywords in certain contexts.
Apart from adding additional features, gathering additional training data might also facilitate a more clearly separation of the data. With more training examples, features like the document position or the Part of Speech could gather sufficient counts to make more accurate predictions about the qualities of a good keyword. The length feature, the IDF and TF-IDF feature could also benefit from additional training examples.

Another possible future feature, that could help to make the keyword extraction system more adaptable, is to use the newly extracted keywords of a document as an additional source of information. If all information that could be retrieved from those keywords such as their POS tags, their lengths, document positions and frequencies would be fed back into the system, it could probably learn from past keywords and gather additional counts to make future predictions more reliable. Updating all values corresponding to the feature values of the extracted keywords and the respective distributions could possibly make the system adaptive to new situations as well. Values or POS sequences that may not have been seen before or had been rare could then be able to become more significant values for distinguishing good keyword candidates from the ones that are not as descriptive.

A. Appendix

A.1 Natural Language Toolkit - NLTK

The Natural Language Toolkit is an open source suite of libraries and programs for the Python programming language[1]. It is intended to support research and teaching in Natural Language Processing (NLP) and other closely related areas. The NLTK provides interfaces to a great number of corpora and lexical resources and a suite of text processing libraries for tokenization, stemming and tagging; some of which are utilised in this thesis. It is available for Microsoft Windows, Mac OS X and Linux.

A.2 LIBSVM

LIBSVM is a library for Support Vector Machines written by Chih-Chung Chang and Chih-Jen Lin. It provides support for classification (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM) and also supports multi-class classification. It provides interfaces for several other programming languages such as the Python programming language, Perl, MATLAB/OCTAVE and Java and comes with a simple applet for demonstrating SVM classification and regression. Additionally many extensions are available to provide even more functionality.

[1] http://www.python.org

B. Abbreviations

ESA-IRS European Space Agency's Information Retrieval Service

IDF Inverse Document Frequency

TREC Text REtrieval Conference

TF Term Frequency

POS Part of Speech

IQ Information Quotient

FBI Fuzzy Bigram Index

FNI Fuzzy N-Gram Index

SVM Support Vector Machine

JSD Jensen-Shannon Divergence

MI Mutual Information

PMI Pointwise Mutual Information

NLTK Natural Language Toolkit

NLP Natural Language Processing

TED Technology, Entertainment, Design

TALENT Text Analysis and Language ENgineering Tools

KEA Keyphrase Extraction Algorithm

SMOTE Synthetic Minority Over-Sampling Technique

DF Document Frequency

RBF Radial Basis Function

TF-IDF Term Frequency-Inverse Document Frequency

KLD Kullback-Leibler Divergence

PCA Principal Component Analysis

Bibliography

[1] Vannevar Bush. As we may think. *The Atlantic*, July 1945.

[2] C.N. Mooers. *The theory of digital handling of non-numerical information and its implications to machine economics*. Zator technical bulletin. Zator Co., 1950.

[3] Mortimer Taube, C.D. Gull, and I.S. Wachtel. Unit terms in coordinate indexing. *American Documentation*, 3:213–218, 1952.

[4] H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM J. Res. Dev.*, 1(4):309–317, October 1957.

[5] Alvin M Weinberg et al. Science, government, and information: The responsibilities of the technical community and the government in the transfer of information. 1963.

[6] G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[7] Karen Spärk Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[8] Ellen Voorhees, Donna K Harman, et al. *TREC: Experiment and evaluation in information retrieval*, volume 63. MIT press Cambridge, 2005.

[9] Tim Berners-Lee. Information management: A proposal, March 1989.

[10] Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(01):157–169, 2004.

[11] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. Domain-specific keyphrase extraction. In *In Proceeding of 16th International Joint Conference on Artificial Intelligence*, pages 668–673, San Francisco, CA, USA, 1999. USA: Morgan Kaufmann Publishers.

[12] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, EMNLP '03, pages 216–223, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[13] James W. Cooper, Anni R. Coden, and Eric W. Brown. Detecting similar documents using salient terms. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 245–251, New York, NY, USA, 2002. ACM.

[14] Mary S. Neff, Roy J. Byrd, and Branimir K. Boguraev. The talent system: Textract architecture and data model. *Nat. Lang. Eng.*, 10(3-4):307–326, September 2004.

[15] Bidyut Das, Subhajit Pal, Suman Kr. Mondal, Dipankar Dalui, and Saikat Kumar Shome. Automatic keyword extraction from any text document using n-gram rigid collocation. 3(2):238–242, 2013.

[16] Weidong Jiang and Xiaofeng Hui. Automatic keyword extraction based on imbalanced classification methods. *Journal of Computational Information Systems*, 9(21):8483–8490, November 2013.

[17] ThomasG. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

[18] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[19] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[20] Rakhi Chakraborty. Domain keyword extraction technique: A new weighting method based on frequency analysis. *Computer Science & Information Technology (CS & IT)*, 3(2):109–118, 2013.

[21] Ludovic Jean-Louis, Michel Gagnon, and Eric Charton. A knowledge-base oriented approach for automatic keyword extraction. *Computación y Sistemas*, 17(2):187–196, 2013.

[22] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[23] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[24] Ido Dagan, Lillian Lee, and Fernando Pereira. Similarity-based methods for word sense disambiguation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, ACL '98, pages 56–63, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics.

[25] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. A Wiley-Interscience publication. Wiley-Interscience, Hoboken, NJ, 2. ed. edition, 2006.

[26] Karl Pearson. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine Series 5*, 50(302):157–175, 1900.

[27] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964.

[28] P. Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

[29] John A Nelder and Roger Mead. A simplex method for function minimization. *Computer journal*, 7(4):308–313, 1965.

[30] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

[31] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.

[32] M. F. Porter. Readings in information retrieval. chapter An Algorithm for Suffix Stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[33] K.A. Papineni, S. Roukos, and R. T. Ward. Maximum likelihood and discriminative training of direct translation models. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 1, pages 189–192 vol.1, 1998.

[34] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python* -. O'Reilly Media, Inc., Sebastopol, CA, 2009.