CHALMERS

Keyword Extraction using Machine Learning

Master of Science Thesis

Computer Science: Algorithms, Languages and Logic

MARTIN JOHANSSON
PONTUS LINDSTRÖM

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

Keyword Extraction using Machine Learning

MARTIN JOHANSSON
PONTUS LINDSTRÖM

Examiner: PETER DAMASCHKE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

A key inserted into a lock.

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

Abstract

This master thesis evaluates different approaches of keyword extraction. Natural language processing methods such as N-grams, Part-Of-Speech and Noun Phrase-Chunking are used to extract keyword candidates. Machine learning algorithms are used to determine whether a candidate is a keyword or not. Unsupervised state-of-the-art algorithms are implemented and compared to the machine learning classifiers. A number of keyword features, their representation and their impact on the results are investigated. The results show that combining natural language processing and machine learning algorithms can improve keyword quality, compared to other methods such as TFIDF.

Keywords

Keywords extracted from the title and abstract for this thesis.

Keyword Extraction, Unsupervised state-of-the-art algorithms, machine learning, keyword features, keyword candidates, Natural language processing methods, Noun Phrase-Chunking, keyword, master thesis, keyword quality.

Acknowledgments

We would like to thank our supervisor at Findwise AB, Svetoslav Marinov for providing several resources vital to the success of the project, for proof reading, ideas and most of all for his enthusiasm.

We would like to thank Peter Damaschke, our supervisor at Chalmers University of Technology, for proof-reading, support with machine learning algorithms and for swift responses when help was requested.

Contents

Terminology

Stopwords Frequently occurring words that do not add any context information.

F-Measure Performance measure.

Corpus A large set of documents.

Token A single word from a phrase or sentence.

Keyword A word or a phrase that represent the content of a document.

Classifier A function that map sets of input attributes to tagged classes.

PoS Part-of-Speech. The lexical class of a word.

Lemmatization The process of bringing inflected words to their morphological root.

Stemming The process of removing affixes from a word.

NP-Chunking Noun Phrase Chunking, the process of extracting nouns and proper nouns with modifiers from a sentence.

PSO Particle Swarm Optimization, a stochastic optimization algorithm.

KBANN Knowledge-Based Artificial Neural Network, a neural network implementation using domain knowledge to set initial weights.

TFIDF Term Frequency Inverse Document Frequency.

NBC Naive Bayesian Classifier, a machine learning classifier.

SVM Support Vector Machine, a machine learning classifier.

Chapter 1

Introduction

This master thesis was carried out at Chalmers University of Technology and Findwise AB[1] in Gothenburg, Sweden. The goal was to create a module for automatically extracting keywords from documents in English and Swedish.

1.1 Background

Keyword extraction is the task of extracting a small set of words and phrases from a document which describe its content. This is different from keyword assignment, where a set of predefined keywords are used. Regardless of the document content, keywords can only be selected from the predefined set.

Manual keyword extraction is performed by professional human annotators and is a very time-consuming task. The purpose of using an automated process is to increase speed, without sacrificing too much quality of the result.

In semi-automatic keyword extraction an automated process extracts a large number of keywords and a human has the final word. The quality might not be as high as the manual process, but is much faster and the quality should be significantly higher than full automation.

What constitute a good keyword? Studies have shown the disagreement rate between two human indexers to be 20-80%[22]. Using an automatic or semi-automatic process for keyword extraction will bring more consistency, which can be an advantage. The manual keywords are regarded as *gold standard* and the machine learning classifiers are trained to conform themselves to this standard. As a consequence, a classifier can only be as good as the keywords it is trained on.

What the keywords of a document should be is highly subjective and depend on a number of factors, e.g. the purpose of the keywords, the information that is sought after and who is looking for it.

[1]http://www.findwise.se/

1.2 Motivation

Keywords serve several purposes to a reader, who can quickly determine if a document is in their field of interest or not. When browsing a large index of documents and their corresponding keywords, the reader can decide which documents are worth looking into further.

When making a search query using keywords, the search engine will yield fewer and more specific results[17].

Not all documents have attached keywords when put into databases and the process of extracting (or assigning) keywords manually is time consuming. When indexing large sets of documents for search engines, a manual approach is unreasonably slow. The process can be sped up using the automatic keyword extraction methods described in this thesis.

1.3 Earlier work

In the 1950s scientists elaborated on the idea of searching texts with computers and a method was proposed that suggested to use words for indexing documents in information retrieval [15].

Many different approaches to keyword extraction has been tested since, some of the major landmarks are listed below:

TFIDF 1972. An improvement over computing word frequencies is introduced by Spärck[8]. An inverse document frequency, computed using a corpus, is weighed with the word frequency to improve performance.

GenEx 1999. A solution based on a genetic algorithm that tunes the parameters for a keyword extractor is developed by Turney[17].

KEA 1999. Frank[12] introduced a keyword extraction algorithm based on the Naive Bayesian learning scheme. The performance was comparable to GenEx.

HITS 1999. Kleinberg[9] introduces a graph-based ranking algorithm similar to PageRank. Can be converted into a keyword extractor using the TextRank approach.

Hulth 2004. Using both Natural Language Processing and Machine Learning, Hulth[7] developed a keyword extractor that has been considered to be state-of-the-art[16].

TextRank 2004. Based on Google's PageRank algorithm[11], Mihalcea and Tarau[16] introduce an unsupervised keyword extractor that utilizes a word co-occurrence graph.

1.4 Project goals and delimitations

The goal of this project is to implement a module written in Java for automatically extracting keywords from English and Swedish documents. The module is supposed to function as stand alone application or integrated into OpenPipeline[2], a framework for crawling, parsing, analyzing and routing documents.

A small number of selected machine learning algorithms will be implemented and improved, rather than making "shallow" implementations of many. Variants of existing algorithms will be used, instead of developing new ones.

The algorithms themselves will be the focus of the project. Document parsing and input retrieval will be given little or no attention. Words are assumed to be spelled correctly.

Code will be written for as much of the content as possible, with the exception of linguistic functionality.

Classification and training speed is to be given a significant amount of attention. This will enable more efficient testing and faster extraction of keywords in live situations.

The goal with regards to the quality of the result is an F-measure above 40%, similar to what Hulth reported in [7].

1.5 Method

The task was first defined and split up into smaller segments for a literature study, implementation, testing and verification.

In order to get acquainted with the field of keyword extraction a literature study was carried out. Large paper databases (e.g. IEEE and ACM) were searched and by reading frequently referenced papers, the most common and effective algorithms were discovered.

After the literature study, a number of algorithms were selected to be implemented. The implementation process was coupled with debugging and minor algorithm parameter tweaking. Alongside the implementation of the classification algorithms, the supporting structure was built. This included document representation, lemmatization and stemming, pre- and post processing of documents and a test suite for evaluation of results.

[2]`http://www.openpipeline.org/`

The following software and external tools were used:

- **SPARK** - Swedish NP-chunker.
  `http://stp.lingfil.uu.se/~bea/resources/spark/`
- **EngChunker** - English NP-chunker.
  `http://www.dcs.shef.ac.uk/~mark/phd/software/chunker.html`
- **Hunpos** - Part-of-Speech tagger for English and Swedish.
  `http://code.google.com/p/hunpos/`
- **Weka** - Machine learning package written in Java.
  `http://www.cs.waikato.ac.nz/ml/weka/`
- **Findwise lemmatization and stemming.**
  Proprietary software.

The project was wrapped up by generating result tables for the final version of each algorithm and comparing the results to those presented in other papers.

1.6 Precision, Recall and F-Measure

F-measure is one of the standard methods in information retrieval for evaluating results. It is most commonly used to evaluate search engine results, but can also be used in keyword extraction by replacing web sites for keywords.

The F-measure is a function of *precision* and *recall*. To compute these three values, two sets of words are needed for some document: a set of manual keywords M and a set of automatically extracted keywords A.

Precision, recall and the F-measure are computed according to formulae 1.1-1.3.

$$\text{Precision} = \frac{|M \cap A|}{|A|} \tag{1.1}$$

$$\text{Recall} = \frac{|M \cap A|}{|M|} \tag{1.2}$$

$$\text{F-measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{1.3}$$

The F-measure has the property that it leans toward the smallest parameter value, whether it be precision or recall. If there is a big difference between precision and recall, the F-measure will be approximately equal to double the smallest of them. This behavior can observed in appendix B.

1.7 Outline

The structure of this report follow the structure of implemented module, which is shown in figure 1.1.

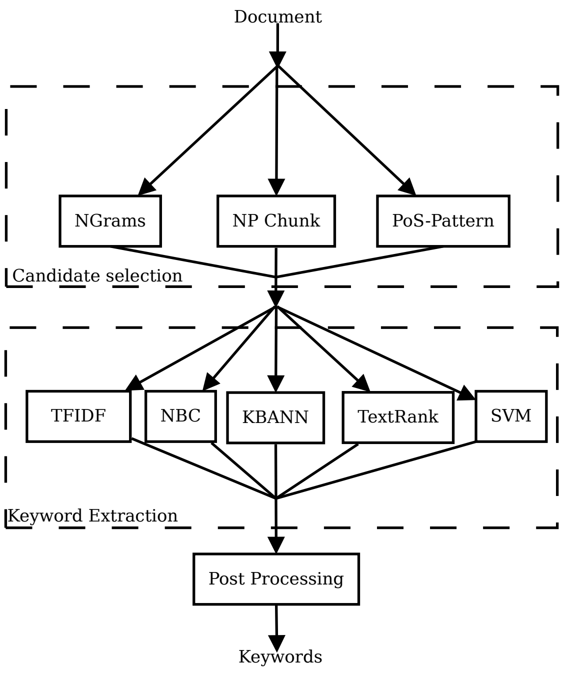


Figure 1.1: *Program flow.*

Chapter 2 describes the corpora and preprocessing methods used in this project. The corpora are processed to generate training- and validation data. Preprocessing steps include parsing, lemmatization, stemming, candidate selection and candidate abstraction.

Chapter 3 presents the keyword classification algorithms and post processing. Extracted candidates are classified to be *keyword* or *not keyword*. The post processing attempts to remove redundant keywords, those that are subsumed by others.

Chapter 4 presents the brief version of the results from this project and compare them to others. The extensive list of results can be found in appendix B.

Chapter 5 contains discussion of the results, evaluation methods and things that could have been done differently in order to achieve better results.

Chapter 6 conclude the report with final remarks and ideas for future work.

Chapter 2

Text Processing

This chapter describe the corpora used in this project which is used to generate training- and validation data sets. The preprocessing stages that documents undergo before extracted candidates are classified are presented.

2.1 Corpora

The corpora used in this report were all formatted before used. All documents were enhanced with meta information, in the form of a title, in order for the parser to recognize the title of each document. Any manually assigned keywords that did not explicitly appear in the associated document were removed. As a result, some documents lacked keywords all together, these were removed.

Information about the corpora used in this report is shown in table 2.1, all reported numbers are post formatting.

Table 2.1: *Corpora information. (*Automatically extracted from)*

Name	Language	Source	In papers
Inspec	English	Hulth[7]	[7, 16]
NUS	English	National University of Singapore[a]	[2]
Medicin	Swedish	* `http://www.internetmedicin.se`	
Socialstyrelsen	Swedish	* `http://www.socialstyrelsen.se`	

Name	Nr of docs.	Nr of words	Avg. doc. size	Nr of keywords	Avg. nr of keywords
Inspec	1 988	264 541	133	14 555	7.3
NUS	152	1 071 026	7 046	1 239	8.1
Medicin	671	893 834	1 332	3 587	5.3
Socialstyrelsen	1 473	746 704	507	3 550	2.4

[a] `http://wing.comp.nus.edu.sg/downloads/keyphraseCorpus/corpus.tgz`

Inspec is the main corpus in this report. It has been used in two of the referenced papers and is a collection of abstracts from the Inspec database. The texts are short, clean[1] and contain a lot of information. The keywords are set by authors or professional annotators.

NUS consist of scientific publications from the National University of Singapore. The documents are long and contain much noise, in the form of remnant tokens from the pdf to text conversion.

Internetmedicin is a web site with the purpose of supplying doctors and nurses with information about treatments and other information useful in their daily work. The texts are reviewed and maintained by professionals, the keywords are assumed to be set by the same people.

Socialstyrelsen is the Swedish National Board of Health and Welfare. The board establishes norms and general advices for municipalities, county councils and local authorities. The keywords are assumed to be set by the same people that work for previously mentioned authorities.

2.2 Lemmatization and stemming

Lemmatization and stemming are used to reduce an inflected word to its base form or stem. This serves two purposes by reducing the number of unique keyword candidates and also allows more accurate computations of word frequencies. The words "pirate", "pirates" and "pirate's" are different but have the same base form. It is redundant to classify all three as keywords since they all refer to the same object.

Lemmatization finds the base form, or lemma, of words. Stemming use an algorithm to iteratively remove word affixes until some termination criteria is met. This means that a stem is not necessarily equal to the morphological root of the word (table 2.2).

Table 2.2: *Stemming and lemmatization examples.*

Word	Stem	Lemma
Recharging	Recharg	Recharge
Recharged	Recharg	Recharge
Cats	Cat	Cat
Ponies	Poni	Pony
Ran	Ran	Run

There are both advantages and disadvantages of stemming. Morphologically similar words may be reduced to the same stem, but if they have no semantic relation this is unwanted behavior. On the other hand, if two morphologically similar words have a strong semantic relation are reduced to the same stem, but does not share the same lemma, this is acceptable behavior.

[1]Texts that contain few tokens that are not considered to be proper words, e.g. formulas.

This project used lemmatization with Porter's stemmer[14] as a fallback. If a word was not found in the lemmatization lexicon it was stemmed instead. It is believed that this combination has not been used before in the area of keyword extraction.

2.3 Candidate selection

The candidate selection step serve the purposes of limiting the number of potential keywords and guarantees a certain quality of the selected candidates.

If few poor candidates are extracted, the classifiers are less likely to select poor keywords. A decreased number of candidates also mean less work for the classifiers, which increase speed.

The text in the box below will be used throughout this section to show how the different candidate selection methods work.

A new survey carried out by the film companies indicates that only seven of the 109 torrent files named in the case had been taken off The Pirate Bay, all of which remained available via other sites.

2.3.1 N-grams

The simplest candidate selection method used in this report is based on N-grams and is also used in [12, 7, 17]. It extracts sequences of up to n tokens. In this report $n = 3$, since investigations on the corpora show that keyword of four or more tokens are rare (table 2.3). Also, the number of candidates extracted by N-grams increases dramatically with large values of n.

Table 2.3: *The number of keywords consisting of a certain number of tokens (columns) for each corpus.*

	1	2	3	4	5	6	7	8
Inspec	1 937	6 065	2 652	650	135	31	9	2
NUS	379	623	168	50	17	3		
Medicin	3 072	381	102	22	2			
Socialstyrelsen	3 345	222	18	2	1			

The only candidates that get filtered, are those that start or stop with a stopword (including single token sequences).

An example of what the N-grams method extracts is shown in the box below:

new	survey carried	film companies indicates
new survey	carried	companies
new survey carried	film	companies indicates
survey	film companies	indicates

2.3.2 Part-of-Speech

A PoS tag is a label assigned to a word that contain information of which lexical class the word belong to and its inflection, for example a singular or plural noun.

The box below show a sentence tagged with PoS tags:

a$_{DT}$ new$_{JJ}$ survey$_{NN}$ carried$_{VBD}$ out$_{RP}$ by$_{IN}$ the$_{DT}$ film$_{NN}$ companies$_{NNS}$ indicates$_{VBZ}$ that$_{IN}$ only$_{RB}$ seven$_{CD}$ of$_{IN}$ the$_{DT}$ 109$_{CD}$ torrent$_{NN}$ files$_{VBZ}$ named$_{VBN}$ in$_{IN}$ the$_{DT}$ case$_{NN}$ had$_{VBD}$ been$_{VBN}$ taken$_{VBN}$ of$_{IN}$ The$_{DT}$ Pirate$_{NNP}$ bay,$_{NNP}$ all$_{DT}$ of$_{IN}$ which$_{WDT}$ remained$_{VBD}$ available$_{JJ}$ via$_{IN}$ other$_{JJ}$ sites.$_{NNS}$

The same word can have different meanings depending on the context, therefore the tagger take the surrounding words into account when assigning the tag. An example is the word "play" that can be either a noun or verb depending on the context. In the sentence "I want to play" it is a verb and in the sentence "I went to a play" it is a noun.

The PoS tags are used in two aspects in this report: to extract candidates that match certain PoS patterns [7, 16, 21] and as keyword features [7] (see chapter 2.4).

The patterns used to extract candidates where found by examining the patterns of the keywords in the training data. If a pattern occurred ten times or more in the training data, it was included in the candidate selection.

The most common patterns for English and Swedish and their explanations are shown in table 2.4. A complete list of the PoS tags used to extract candidates can be found in appendix A.

Table 2.4: *The English and Swedish PoS tags are shown in the left table. PoS tag descriptions are shown in the right table.*

English	Swedish
jj nn	nn
nn nn	nnp
jj nns	jj nn
nn	jj
nn nns	nn kn nn
jj nn	vb
nnp	pc
nns	pc nn
nnp nn	nn kn jj
jj nn nns	nnp nnp
nn nn nn	nn pp nn

Tag	Description
nn	Noun, Singular
nns	Noun, Plural
jj	Adjective
vb	Verb, Base Form
nnp	Proper Noun, Singular
kn	Coordinating Conjunction
pc	Particle
pp	Preposition

An example of what the PoS candidate selection method extract is shown in the box below:

new survey	film	torrent	bay
survey	film companies	case	pirate bay
film	companies	pirate	other sites

2.3.3 Noun Phrase-Chunking

NP-chunking is the process of finding phrases whose head is a noun or proper noun, optionally accompanied by a set of modifiers. Similar to the PoS candidate selection method, it searches for patterns in the PoS tags of the words in a sentence. The major differences are that NP-chunking use a grammar instead of predefined patterns and that the candidates are restricted to noun phrases.

The grammar is used to reduce a sentence into a minimum number of phrases, the goal is to find nouns and proper nouns with all their respective modifiers. Longer phrases are preferred which is consistent with a minimum number of phrases.

In the implementation of NP-chunking in this project, a post processing stage is included that removes opening determinants[7], such as “the”, “a” and “an”, and filters candidates that are stopwords. The sentence “A new survey” will be reduced to “new survey”. This is motivated by that the determinant does not contain any relevant information.

An example of what the NP-chunking candidate selection method extract is shown in the box below:

new survey	torrent	other sites	only seven
film companies	case	pirate bay	

2.4 Keyword features and candidate abstraction

The Naive Bayesian Classifier, Artificial Neural Networks and Support Vector Machines are similar, they attempt to create an optimal separation of data.

In order to create separable data from a set of candidate keywords, each candidate keyword is abstracted into a numerical vector where a dimension represents a certain *feature*. A *feature* is a property or attribute of a candidate that is used as a keyword indicator (derived through empirical studies), e.g. if the candidate is in the title of a document or if it has a high TFIDF score. Using the vector representation for candidates, the classifiers can create a separation between those that are keywords and those that are not.

This project use boolean features, values of either 0 or 1. It is also possible to use real values but this has both advantages and disadvantages. It can help the classifiers create a more accurate separation of data, but at the same time they will be more prone to overfitting and training the classifiers take longer time.

Certain numerical measures (e.g. TFIDF) are converted into a set of binary features by applying a number of intervals. The feature "TFIDF High" is 1 of the normalized TFIDF value is in the interval: $0.9 \leq TFIDF \leq 1.0$ and 0 otherwise.

The TFIDF score and first- and last occurrence values are normalized, mainly due to the reason that documents can vary heavily in length.

A list of the features used in this project is presented in table 2.5.

Table 2.5: *Keyword features. All mentions of TFIDF scores are normalized within every document. Tags such as (jj/nn) means that an adjective or a noun is acceptable at the given position.*

Feature	Additional info	Description
TFIDF High	$0.9 \leq TFIDF \leq 1.0$	
TFIDF Mid/High	$0.7 \leq TFIDF < 0.9$	
TFIDF Low/Mid	$0.3 \leq TFIDF < 0.7$	
TFIDF Low	$0.1 \leq TFIDF < 0.3$	
Relative First Occurrence	$0 \leq F.O \leq 0.1$	First occurrence of candidate is within first 10% of the document
Relative Last Occurrence	$0.9 \leq L.O \leq 1.0$	Last occurrence of candidate is within last 10% of the document
In Title		If the candidate can be found in the title
Starts Sentence		If the candidate at any location in the document starts a sentence
Acronym		If the candidate contains an acronym or a sequence of words that is acronymized in the text
PoS: pm	Swedish	Tags are explained in table 2.4
PoS: nn	Swedish	
PoS: jj (jj/nn) (jj/nn) nn	English	
PoS: nn jj nn nn	English	
PoS: (jj/nn) vb nn	English	
PoS: (jj/nn) (jj/nn) nn	English	
PoS: (jj/nn) nn	English	

The TFIDF measure is a good enough feature to be used on its own when extracting keywords (see appendix B.3). It is used by Hulth[7] but as two separate features: TF and IDF.

The first- and last occurrence features are based on the layout of academic papers, where the beginning and end are usually dense in information. Keywords are likely to appear in abstracts and introduction sections, as well as in summaries. These sections tend to summarize the content of a paper in a condensed manner.

Titles of papers and sections give a short and information dense description of what the upcoming text will deal with. A good example is the title of this Master Thesis: “Keyword extraction using Machine Learning”. In this report only document titles were taken into account, the notation for chapter and section titles are too specific for each individual text to be considered.

Tests showed that if a word starts a sentence it is likely to be a keyword. When starting a new section or chapter, a keyword often starts the first sentence. An example from this report is: “Word frequency is the baseline ...”.

Experiments also indicated that abbreviated phrases were likely to be keywords.

Chapter 3

Keyword Classification

This chapter describes the keyword classifiers used in this project. Their task is to classify a set of candidates, each being either *keyword* or *not keyword*. The exceptions are the TextRank and ExpandRank classifiers, that extract their own keyword candidates.

The classifiers can be divided into two groups depending on what type of result they produce. Binary classifiers divide the candidates into two groups: *keywords* and *not keywords*. Other classifiers compute a score for each candidate and classify the candidates with the highest scores to be keywords.

The post processing that is performed after classification on a set of extracted keywords is also presented in this chapter. It can remove keywords if certain conditions are met, such as if one keyword subsumes another.

3.1 Word Frequency

Word frequency[1] is the baseline algorithm in this report with regard to the F-measure performance. It counts the number of occurrences of each candidate and classifies the most frequent to be keywords.

Certain words in domain-specific documents, that are not stopwords, may appear more frequently than others. It is not likely that they add any information that is unique to their documents and are therefore not likely to be keywords. This is a shortcoming of the word frequency approach, that will disregard this fact.

[1]Synonym to "term frequency"

3.2 TFIDF

The measure was introduced Karen Spärck Jones[8] in 1972 and is very common in keyword extraction and information retrieval in general.

It weighs the term frequency (TF) of a candidate keyword in a given document with its inverse document frequency (IDF), that requires a reference corpus to compute.

Formula 3.1 show how to compute the TFIDF score, where tf is the term frequency, D is the total number of reference documents and d is the number of reference documents that contain the current candidate.

$$\text{TFIDF} = tf \cdot log\left(\frac{D+1}{d+1}\right) \tag{3.1}$$

Adding 1 in the denominator avoid division by zero and the logarithm of the quotient is motivated for two reasons. The first is that differences in d becomes less significant, especially when d is small. The second and most important reason is that terms that exist in nearly all reference documents will be given a low score or even a score of zero if they exist in all of them.

The TFIDF measure weigh candidates that are unique for the current document heavier than others and avoid the problem of word frequency by using the IDF of a candidate. By using a domain-specific reference corpus, the quality of the keywords can be improved even more due to domain-frequent words being filtered.

3.3 TextRank

Based on Google's PageRank algorithm[11, 16], TextRank is a state-of-the-art unsupervised keyword extraction algorithm[5].

PageRank construct a co-occurrence graph where nodes represent web sites and edges are links. A formula to compute the score of each node is applied iteratively until they have all stabilized. A high score can be achieved by linking and being linked to by other sites with a high score.

TextRank construct a co-occurrence graph where the nodes are the unique nouns, proper nouns and adjectives of a document. The edges are created by moving a fixed size window, of size 2 to 20 words, over the sentences of the document. An edge of weight zero is put between every pair of nodes whose words are within the window at any point and its weight is incremented by one for each co-occurrence.

Formally, let $G = (V, E)$ be an undirected graph with a set of vertices V and set of edges E, where E is a subset of $V \times V$. Each vertex represent a word and hold a positive score and every edge hold a positive weight. For a given vertex V_i, let $Adj(V_i)$ be the set of adjacent vertices to V_i.

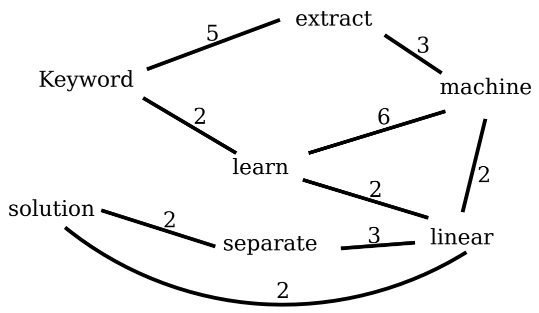


Figure 3.1: *Example of a co-occurrence graph.*

Experiments using a directed graph were performed in [16], where the direction of an edge was decided by the sequence of the words. If w_1 precede w_2 there will be an edge from w_1 to w_2. The results did now show any significant difference, therefore the undirected approach was used in this report.

The word score is calculated using formula 3.2, where d is a damping factor and is set between 0 and 1 and w_{ij} is the weight of the edge between nodes V_i and V_j. The recommended value for d is 0.85 [11, 16] but in this project the value is set to 0.01, which gave better results. This value was derived from empirical studies.

$$S(V_i) = (1-d) + d \cdot \sum_{V_j \in Adj(V_i)} \frac{w_{ij}}{\sum\limits_{V_k \in Adj(V_j)} w_{jk}} S(V_j) \tag{3.2}$$

Formula 3.2 is applied iteratively on the nodes of the graph until the scores have all stabilized. The words with the highest score are selected for the next stage of the process, which is to merge them into the phrases that will be the keywords of the current document.

A word can be merged with another word or phrase if they are adjacent at any place in the document and if the difference in word score does not exceed a set threshold value. When a merging occur, the score of the new phrase is computed according to formula 3.3, where w_1, w_2 are the words/phrases to merge, size return the number of tokens of a phrase and termSize is defined as in formula 3.4.

$$\text{Score} = 2 \cdot \frac{w_1 \cdot w_2}{w_1 + w_2} \cdot \text{termSize}(\text{size}(w_1) + \text{size}(w_2)) \tag{3.3}$$

$$\text{termSize}(k) = \begin{cases} 2 & = 3 \\ 3 & = 2 \\ \text{else} & = \text{small number} \end{cases} \tag{3.4}$$

Formula 3.3 is inspired by the properties of the F-measure, the lowest parameter value dominates the output. This is useful when merging high- and low score words, the word with high score might be more worth extracting as a single token keyphrase.

The termSize function is language dependent and conform to the number of tokens in the keywords for each language (table 2.3). The English corpora tend to have keywords of two or three tokens and a vast majority of the keywords in the Swedish corpora are single tokens. Formula 3.4 show the English version.

3.3.1 ExpandRank

An extension of TextRank is the ExpandRank algorithm[21], that use neighborhood knowledge to extract keywords. The knowledge is represented by the k most similar documents from a reference corpus and is used to construct an extended graph.

Formally, let d_0 be the current document, let $d_1, ..., d_k$ be the k most similar document with regard to d_0 and let $D = \{d_0, d_1, .., d_k\}$ be the expanded document set used to build the extended graph.

The k most similar documents are retrieved by computing the cosine similarity (formula 3.5) on vector representations $\vec{d_i}$ of the current document and the documents of a corpus. Each dimension in the vector contain the TFIDF score for a certain word in the document.

$$sim_{doc}(\vec{d_i}, \vec{d_j}) = \frac{\vec{d_i} \cdot \vec{d_j}}{||\vec{d_i}||\,||\vec{d_j}||} \tag{3.5}$$

The contribution of a neighborhood is altered edge weights. The weight of the edge between nodes v_i, v_j is given by formula 3.6, where $count(v_i, v_j)$ is the number of co-occurrences of words v_i, v_j in neighborhood document d_p. The weight contribution is regulated by the cosine similarity where similar documents will add more weight for every co-occurrence than unalike documents.

$$w_{ij} = \sum_{d_p \in D} sim_{doc}(\vec{d_0}, \vec{d_p}) \cdot count(v_i, v_j) \tag{3.6}$$

In [21] edge weights are normalized for each vertex. Tests performed for this report showed that it resulted in a lower F-measure.

3.4 Naive Bayesian Classifier

The purpose of implementing the Naive Bayesian Classifier (NBC) was mainly to provide a knowledge representation[4] for the KBANN classifier (see section 3.5.1). When the necessary functionality was written, it was easy converting it to a standalone classifier.

This classifier use probabilistic models built from training data to classify candidate keywords. Such a model contain the probabilities of each *feature* (presented in section 2.4) being either *present* or *not present* for any candidate. The probabilities are weighed together to create a total probability of a candidate being a keyword. The candidates with the highest total probability are classified as keywords.

To understand the computations of the NBC, one need to understand Bayes theorem (formula 3.7). Given two independent events A and B, the posterior probability $P(A|B)$ of A can be computed from the prior probability $P(A)$, the evidence $P(B)$ and the distribution function $P(B|A)$ (the probability of B given A).

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{3.7}$$

Suppose that there are several given events $B_1, ..., B_n$ instead of just one. To compute this posterior probability would be very complex and therefore an assumption is made that the events $B_1, ..., B_n$ are all conditionally independent. The posterior probability of A can now be rephrased in formula 3.8. Note that the denominator does not depend on A and can therefore be removed since it is constant.

$$P(A|B_1, ..., B_n) = P(A) \prod_{i=1}^{n} P(B_i|A) \tag{3.8}$$

In the case of the NBC there are several given boolean features (events) $F_1, ..., F_n$, where $F_i \in \{0, 1\}$ for $i \in \{1, ..., n\}$ and C is the event that the current candidate is a keyword. This is depicted in formula 3.9 which is a direct translation of formula 3.8.

$$P(C|F_1, ..., F_n) = P(C) \prod_{i=1}^{n} P(F_i|C) \tag{3.9}$$

Formula 3.9 is incorrect because the denominator from formula 3.7 is not constant and cannot be removed. A feature can either be present $F_i = 1$ or not present $F_i = 0$ and the factor within the product symbol need to be changed accordingly. This is shown in formula 3.10.

$$P(C|F_1, ..., F_n) = P(C) \prod_{i=1}^{n} \frac{F_i \cdot P(F_i = 1|C) + (1 - F_i) \cdot P(F_i = 0|C)}{F_i \cdot P(F_i = 1) + (1 - F_i) \cdot P(F_i = 0)} \tag{3.10}$$

Each factor within the product symbol in formula 3.10 now states how strong of an indicator the current feature is. A factor of 2 doubles the probability of a candidate being a keyword, while a factor of $\frac{1}{2}$ halves it.

For the NBC to function it is vital to use discretized features. If real value features would have are used instead, such as TFIDF score, the hypothesis space will grow infinitely large. In theory, all possible candidates can have different TFIDF scores, which will lead to a serious overfitting issue.

3.5 Artificial Neural Network

The purpose of this classifier was to experiment with the connection between domain knowledge and reduced training times. It is based on an ANN and use the the candidate vector representation as input to the network who classifies the candidate.

The network used in this report is a multilayered perceptron network. A perceptron is a simple mathematical model of the neuron devised by McCulloch and Pitts[13]. It holds a hyperplane represented by a weight vector and can be trained or adjusted to separate data effectively.

Formally, let $x \in \mathbb{R}^n$ be the input vector and $w \in \mathbb{R}^n$ be the weight vector. The output of a perceptron is given by $\vec{x} \cdot \vec{w} + b$, where b is the bias weight of the perceptron. It can also be expressed as a sum (formula 3.11).

$$b + \sum_{i=0}^{n} x_i \cdot w_i \tag{3.11}$$

The perceptron output is passed to an activation function that squashes it to a boolean value, active (1) or inactive (-1).

A multilayered perceptron network is a directed acyclic graph where perceptrons are divided into layers of three categories: the input layer whose output is the given input vector, an undefined number of hidden layers that each may vary in number of perceptrons and the output layer that calculate the output for the network. If the network output is 1, the current candidate is classified as a keyword.

An example architecture for an ANN can be seen in figure 3.2, where the input nodes are represented by squares. Normally the input layer is fully connected with the hidden layer, just as the hidden- and output layer are.

The ANN training methods are described in sections 3.5.2 and 3.5.3.

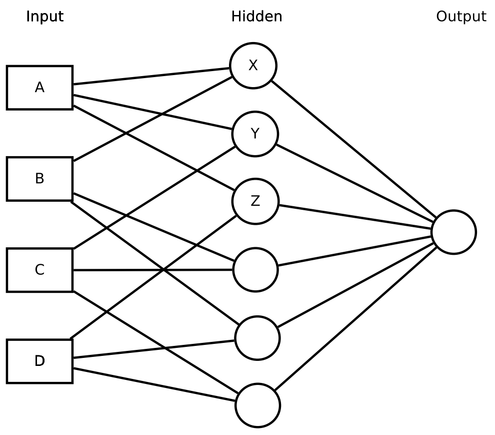


Figure 3.2: *The KBANN architecture used in this project. Input node A is paired up with the other input nodes B, C and D with the hidden nodes X, Y and Z.*

3.5.1 Knowledge Based Artificial Neural Network

A KBANN is an ANN where domain knowledge is used to initialize the weights of the neurons in the network, instead of using random values.

The purpose of using domain knowledge is to reduce the training time, which has been shown to be effective in [4]. It can also avoid local optima simply because the starting point is likely to be a shorter distance from the target hypothesis than a random starting point (figure 3.3).

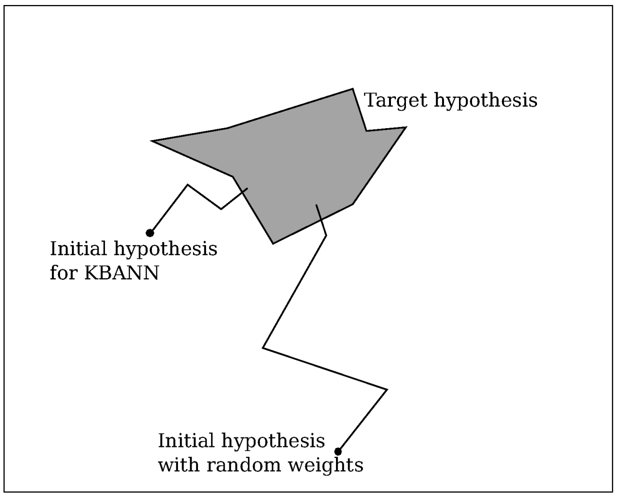


Figure 3.3: *The picture represents the hypothesis space. The shaded area is the target hypothesis. By using domain knowledge the starting hypothesis is likely to lie closer than the hypothesis that is initialized randomly. The training time can be reduced significantly using this method.*

The structure of the network used in this report is inspired by [4]. The connections from the input layer to the hidden layer are made up of every pair of nodes in the input layer such that no node is paired up with itself and no node is paired up with another node representing a feature of the same category. This is motivated since at most one TFIDF feature and at most one PoS feature can be present for any candidate. The goal is to find strong combinations of features.

$$w_{oj} = P(C|F_j = 1) \tag{3.12}$$

The weights of the connections to the hidden layer is retrieved from the NBC and is the posterior probability of the current candidate being a keyword given the presence of the associated feature. This is shown in formula 3.12, where C is the event that the current candidate is a keyword and F_j is the j:th feature. The rest of the weights in the network are initialized to a random value in the interval $[-1, 1]$.

3.5.2 Backpropagation

Backpropagation is a commonly used optimization method for multilayer perceptron networks.

Given a set of training data the algorithm employs gradient descent to minimize the error between the classification- and target values in the training data. The error is calculated on the output nodes and is propagated backwards to the nodes connected to the corresponding output node.

In this report the weight decay and weight momentum extensions were implemented.

Algorithm 3.1 *Backpropagation*[10]

Input: A multilayer feed-forward network, a set of training data.
Output: An optimized feed-forward network based on the set of training data.

Each training example $(\vec{x}, \vec{t}\,)$ consist of the input vector $\vec{x}$ and the output vector $\vec{t}$. Network weights are represented with w_{ij}, where i is the layer index and j is the node index. $\vec{o}$ is the output vector for the network.
Errors are denoted by δ_k and δ_h, where k represents a node in the output layer and h represents a node in the hidden layer. The input from node i into node j is denoted by x_{ji}.
Constants: η is the learning rate, α is the weight momentum, ϵ is the weight decay.

```
• Loop until some termination criteria is met
    – For each training example $(\vec{x}, \vec{t}\,)$ do
        1. Calculate the output $\vec{o}$ for the input vector $\vec{x}$
        2. For each network output unit $k$, calculate its error term $\delta_k$
               $\delta_k \leftarrow t_k - o_k$
        3. For each hidden unit h, calculate its error term $\delta_h$
               $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$
        4. Update the network weights
               $w_{ji} \leftarrow w_{ji} + (1 - \epsilon)\Delta w_{ji}$
           where
               $\Delta w_{ji} = \eta\delta_j x_{ji} + \alpha\Delta w_{ji}(n-1)$
           $\Delta w_{ji}(n-1)$ is the update on the previous iteration update.
```

Weight decay[6] is used to neutralize large weight updates. The formula $(1-\epsilon)$, where $0 \leq \epsilon < 1$ is a constant, is added to the weight update statement. ϵ decides how much of the update to remove. Small updates will not be affected by the weight decay.

Weight momentum[10] is used to avoid the algorithm getting stuck in local optima and is based on the physical idea of inertia. When calculating the weight update, the update value in the previous iteration is taken into account. This makes the weight continue to head towards the direction it was heading in the past iteration. The formula $\alpha\Delta w_{ji}(n-1)$ is added to the weight update statement, where $0 < \alpha < 1$ is a constant and $\Delta w_{ji}(n-1)$ is the update value from the previous iteration.

3.5.3 Particle Swarm Optimization

The original algorithm was discovered while attempting to simulate social behavior for individuals in a group and was simplified after the particles were observed to be performing optimization[3].

The Particle Swarm Optimization (PSO) algorithm holds a populations of particles, each of which have a position and a velocity in a search space. Both the positions and velocities of the particles are initialized to a random value and are subsequently updated in a randomized manner. The updated velocity of a particle depends on three factors: the previous velocity and the cognitive- and social factors.

The previous velocity is weighed with an *inertia weight*[20], which usually decreased from about 1.4 to about 0.4 over the course of an optimization. This favors exploration in the early stages and exploitation in the later stages of an optimization.

The cognitive factor steer each particle towards its own best position, while the social factor steer them toward the swarm best position. Each of these factors are weighed by a random number in the interval *[0,1]* to make the behavior of the particles more stochastic.

The implementation of PSO in this project has been extended with *function stretching*[19] in order to avoid local optima. When the algorithm is stuck in a local optima, the fitness function undergoes a two stage transformation without destroying any optima that is better than current best and the algorithm is restarted. The algorithm is considered to be in a local optima if the best solution is not improved in a certain number of iterations (default is 300). If no better solution is found in another set of iterations, it is believed that the global optima has been found.

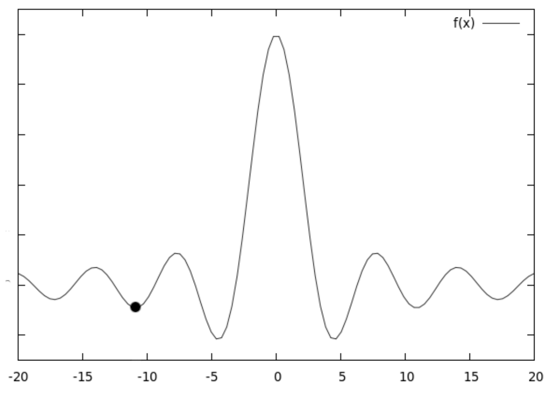
Algorithm 3.2 *Basic Particle Swarm Optimization*[20]

Input: A multilayer feed-forward network, a set of training data.
Output: An optimized feed-forward network based on the set of training data.

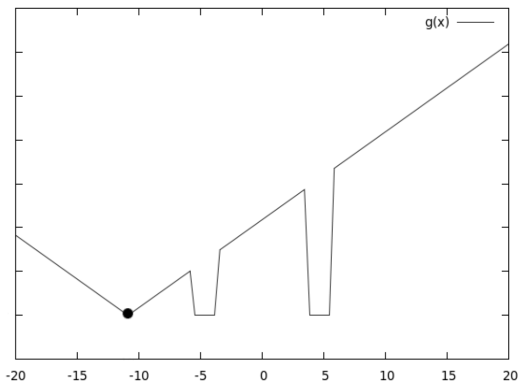
The function to optimize is denoted by $f : \mathbb{R}^n \to \mathbb{R}$. P is the set of particles and $N = |P|$ is the number of particles. $x_{ij} \in \mathbb{R}$ is the position of particle i in the j:th dimension and $v_{ij} \in \mathbb{R}$ is the velocity of particle i in the j:th dimension. The best position for particle i is denoted pb_i and sb is the swarm best. r, r_1 and r_2 are uniform random number in the interval $[0, 1]$ and c_1, c_2 are constants such that $c_1 + c_2 < 4$. w is the inertia weight.

```
• For each particle $i \in P$
    1. Initialize particle positions, velocities and personal bests
        (a) $x_{ij} \leftarrow x_{min} + r(x_{max} - x_{min})$, $i \in \{1, .., N\}$, $j \in \{1, .., n\}$
        (b) $v_{ij} \leftarrow -\frac{1}{2}(x_{max} - x_{min}) + r(x_{max} - x_{min})$, $i \in \{1, .., N\}$, $j \in \{1, .., n\}$
        (c) $pb_i \leftarrow x_i$
    2. Initialize swarm best: $sb \leftarrow arg\,min\,(\forall p \in P.f(p))$
    3. Loop until some termination criteria is met
        – For each particle $i \in P$ and dimension $j \in [1, n]$
            (a) Update velocity, $v_{ij} \leftarrow wv_{ij} + c_1r_1(pb_{ij} - x_{ij}) + c_2r_2(sb_j - x_{ij})$
            (b) Restrict velocities such that $|v_{ij}| \leq v_{max}$
            (c) Update position, $x_{ij} \leftarrow x_{ij} + v_{ij}$
        (a) Update particle best: if $f(x_i) < f(pb_i)$ then $pb_i \leftarrow x_i$
        (b) Update swarm best: if $f(pb_i) < f(sb)$ then $sb \leftarrow pb_i$
        (c) Update inertia weight: if $w > w_{min}$ then $w \leftarrow 0.99w$
```

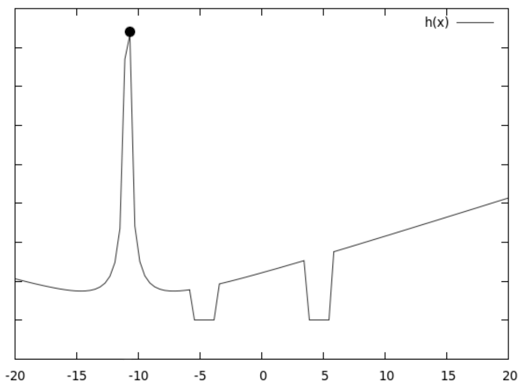
Figure 3.4 shows an attempt at minimizing a function where the algorithm has gotten stuck in a local minimum. It also shows the two transformation stages where the local minimum is transformed into a global maximum, without altering the two global minima.



(a) *Unmodified function:* $f(x) = sinc(x)$.



(b) *Stage 1*



(c) *Stage 2*

Figure 3.4: *The unmodified sinc function and the two stages of the function stretching transformation. The algorithm is stuck in local optima at the dot, the global optima can be seen to the right of the dot. Note that they remain unaltered, the scale on the y-axis has changed from figure 3.4a.*

The extended PSO algorithm has been tested against several functions with a high number of local optima with very good results ($> 99.9\%$ accuracy), in this project and in [19]. One such function is *Levy No 5* with 760 local and only one global optima in the $[-10, 10]^2$ interval.

The ANN is converted into a vector using its weights and then optimized by the PSO algorithm. The F-measure is used as the fitness function and is maximized.

3.6 Support Vector Machine

Introduced in 1995 by Vapnik [18], Support Vector Machines (SVMs) has made a big impact on the academic world and the number of papers on the subject exploded around 2000 [1].

The SVM is a binary classifier, it sorts each candidate into one of two categories: *keyword* or *not keyword*. Given a set of training data, the SVM builds a model consisting of a hyperplane that can separate unseen data and that also maximizes the margin between the two classes.

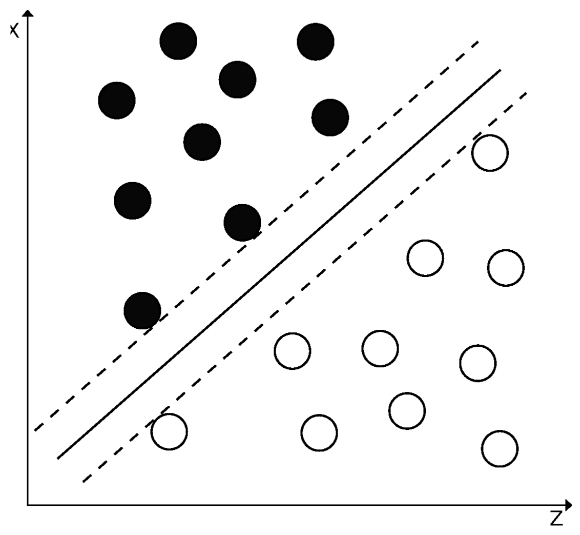


Figure 3.5: *An illustrative example of a separating plane.*

Not all sets of training data are separable in the input space, the SVM solves this problem by mapping the data into a higher dimension called a feature space. Once the data is mapped to the feature space, the SVM can use a hyperplane to perform a linear separation of data that was not linearly separable in the input space, but is in the feature space.

The function used to map the data into a feature space is called a kernel function. There are several types of kernel functions who perform differently on different sets of data.

The SVM used in this report was implemented using Weka[2], due to complexity and time constraints. It uses the candidate vector representation described in section 2.4 and the kernel function used is the Radial Basis Function (RBF). The only parameter optimization was which weight to assign the instances of training data classified as keywords, which utilizes the PSO algorithm (see section 3.5.3).

3.7 Ensembles

The concept of ensemble learning is to select a collection of classifiers and combine their predictions, hopefully improving the result compared to using a single classifier.

Each individual classifier makes a prediction about the classification of an instance. The predictions can be combined in different ways to produce a result. Using majority vote, at least half of the classifiers need to classify an instance as positive in order for the ensemble classification to be positive. Two more examples is if any (union) or all (intersection) of the classifiers predict an instance to be positive, the ensemble classification will also be positive.

In order for ensemble methods to be effective, each classifier either need to be trained on different subsets of training data or on the same training data but with different representations. An example is to train three classifiers on different sets of candidate keywords: N-grams, PoS and NP-chunks.

3.7.1 Regression

This is an ensemble method consisting of non-binary classifiers. Each candidate receive a score from each classifier, which are combined to a total score for each candidate.

This classifier ensemble was an attempt to mimic Hulth's impressive results in [7], that reached an F-measure of 45.5.

The regression ensemble was created using three different NBC classifiers, one for each candidate selection method. The NBC classifier was selected because it is the best regressive classifier, better than word frequency and TFIDF.

When classifying documents, three sets of candidates are extracted. Each classifier give a score to each candidate in their candidate set and then the scores are added for each unique candidate. If a candidate is selected by several classifiers, it has a higher chance of being classified as a keyword.

[2]http://www.cs.waikato.ac.nz/~ml/weka/

3.8 Post Processing

The post processing takes place after a set of candidates have been classified. The purpose is to eliminate redundant keywords if certain conditions are met.

Keywords can be removed if they are completely subsumed by another and if the subsumed keyword does not occur enough times on its own.

For example, two of the keywords extracted from a document are: “advice” and “scientific advice”. The first occur six times while the second occur five times. Not only does the first keyword only occur once by itself but the second contain more information, which makes it more specific to the current document. The assumption is made that the first keyword can be removed safely, without lowering the quality of the result.

This process is also performed by Hulth[7], but without considering the number of stand alone occurrences.

Chapter 4

Results

In this chapter the results obtained in this project are presented, they are also compared to results published in other papers. All precision-, recall- and F-measure values throughout this report are shown in percent.

4.1 Performance evaluation

The best result for each classifier for the main English corpus and the main Swedish corpus is presented in this section.

Extensive result tables can be found in appendix B. They contain results for the candidate selection methods without attached classifiers, all combinations of candidates and classifiers, results for the two remaining corpora (Socialstyrelsen and NUS) and approximate running times for classification and training.

The Word frequency, TFIDF, TextRank, ExpandRank and NBC classifiers were instructed to extract as many keywords as each document has been manually assigned during these experiments. The KBANN and SVM classifiers conform themselves to the number of manually assigned keywords in the training data. These approaches will be discussed in section 5.4.3.

The experiments with the Word frequency, TFIDF, TextRank and ExpandRank classifiers are classification of full corpora since they are unsupervised methods. They do not utilize the manual keywords of the training data in classification of documents. The TFIDF classifier needs a reference corpus to calculate the IDF, but is independent of the corpora used in this report.

The results presented for the NBC, KBANN and SVM classifiers are the average over ten runs of 10-fold cross validation in order to avoid overfitting. The results presented for the ensemble classifiers are from single runs of 10-fold cross validation.

Table 4.1 and 4.2 show the results for the main English and main Swedish corpora respectively.

Table 4.1: *Results for the Inspec corpus. The best result are shown in bold.*

Classifier	Assign. tot.	Assign. mean	Corr. tot.	Corr. mean	P	R	F
Word frequency	14 555	7.32	3 703	1.86	25.45	25.45	25.45
TFIDF	14 555	7.32	3 966	1.99	27.25	27.25	27.25
TextRank	14 555	7.32	4 627	2.33	31.84	31.84	31.84
ExpandRank	14 555	7.32	4 423	2.22	30.41	30.41	30.41
NBC	14 555	7.32	5 101	2.57	**35.05**	35.05	35.05
KBANN-BP	49 405	24.85	8 919	4.49	18.06	**61.28**	27.89
KBANN-PSO	23 384	11.76	7 341	3.69	31.39	50.44	38.70
SVM	23 869	12.01	7 452	3.75	31.22	51.20	38.79
Regression	27 353	13.76	8 136	4.09	29.74	55.90	**38.82**
KBANN ensemble	23 503	11.82	7 380	3.71	31.40	50.70	38.78
KBANN-TextRank	29 668	14.92	8 371	4.21	28.22	57.51	37.86

Table 4.2: *Results for the Medicin corpus. The best result are shown in bold.*

Classifier	Assign. tot.	Assign. mean	Corr. tot.	Corr. mean	P	R	F
Word frequency	3 587	5.35	753	1.13	21.00	21.00	21.00
TFIDF	3 587	5.35	1 047	1.57	29.19	29.19	29.19
TextRank	3 587	5.35	693	1.04	19.32	19.32	19.32
ExpandRank	3 587	5.35	318	0.48	8.87	8.87	8.87
NBC	3 587	5.35	1 046	1.56	29.17	29.17	29.17
KBANN-BP	51 775	77.16	1 292	1.93	2.86	36.02	5.30
KBANN-PSO	2 760	4.11	9 90	1.48	35.87	27.60	31.20
SVM	2 667	3.97	9 76	1.45	36.60	27.21	**31.21**
Regression	3 278	4.89	1 016	1.51	31.00	28.33	29.60
KBANN ensemble	2 292	3.42	884	1.32	**38.57**	24.65	30.08
KBANN-TextRank	8 207	12.23	1 297	1.93	15.81	**36.16**	22.00

4.2 Comparative evaluation

Table 4.3 show results reported in other papers that influenced this project and that is (or was) state-of-the-art.

Table 4.3: *Results from other sources. The best achieved result in this project for the Inspec corpus is also presented at the top of the table. Note that the Inspec corpus was not split in constant sets of training and validation data in this project, cross validation was used instead. This accounts for the larger numbers of assigned keywords and number of correct keywords.*
(Uses the same corpus, five keywords extracted per document.)*

Source	Assign. Tot.	Assign. Mean	Corr. Tot.	Corr. Mean	P	R	F	Corpus
This project	23 869	12.0	7 452	3.8	31.2	51.2	38.8	Inspec
	2 667	4.0	976	1.5	36.6	27.2	31.2	Medicin
Hulth[7]	5 380	10.8	2 093	4.2	**38.9**	**54.8**	**45.5**	Inspec
TextRank[16]	6 784	13.7	2 116	4.2	31.2	43.1	36.2	Inspec
ExpandRank[21]					28.8	35.4	31.7	DUC2001
GenEx[17]					29.0			Custom
KEA[12]					27.0			Custom
Fast ESP	15 509	8.0	3 090	1.6	19.9	26.9	22.9	Inspec
Fast ESP	10 067	15.0	446	0.7	4.4	12.4	6.5	Medicin

Source	Settings
This project	NP-chunk candidates, SVM classifier
	PoS candidates, SVM classifier
Hulth[7]	Regression
TextRank[16]	Undirected, Co-occ,window=2
ExpandRank[21]	Neighborhood of 5 documents
GenEx[17]	Journal, Experiment 2 *
KEA[12]	Journal *
Fast ESP	Semantic Pipeline
Fast ESP	Semantic Pipeline

Chapter 5

Discussion

This chapter discuss the corpora and components of the project, the results and different evaluation methods.

5.1 Corpora

In this section the properties of the corpora used in this report are discussed and also the implications that follow.

Due to the high density of keywords and low noise level in the Inspec corpus, better results should be obtained compared to the other corpora. It is also likely that the quality of the manual keywords, possibly the highest amongst the used corpora, influenced the result.

There are suspicions that the low keyword density the high noise level in NUS was the reason for the significantly worse results compared to the other corpora. The low number of documents also caused the biggest drop in F-measure when comparing results from the cross validation tests and results obtained when using the full corpus as both training- and validation data (from an F-Measure of 26-27 to 23).

Both of the Swedish corpora have a noise level higher than Inspec but lower than NUS, which is likely to affect the results negatively.

Using a general corpus to compute IDF scores can improve performance significantly[7]. The purpose is that the language cover a wider spectrum than a domain-specific corpus and it would be interesting to experiment with a corpus containing e.g. newspaper articles. It is also possible to combine a general- and a domain-specific corpus to calculate IDF scores, this could possibly improve results even more.

5.1.1 Manual keywords

Due to the statistics of PoS patterns (appendix A) and the number of tokens (table 2.3) of the keywords of the corpora used in this project, the quality of the manual keywords can be questioned.

Some manual keywords for Inspec have eight tokens, which is more of a sentence than a phrase. N-grams miss about 7% of the Inspec keywords since it only extracts candidates of up to three tokens. This property is also observable in the English PoS patterns where several patterns are four tokens long.

The PoS patterns also indicate a poor quality of the manually assigned keywords. The lists for both languages contain single adjective and verb. These word classes can be questioned as keywords, since they do not contain any relevant information on their own. There are also patterns that end with an adjective or verb. From a linguistic point of view such sequences are not legitimate (nominal) phrases and an NP-Chunker will fail to identify them.

There is the possibility that tokens are tagged erroneously, a possible cause for some of the odd patterns.

5.2 Candidate representation

The binary vector candidate representation turned out to be crucial for the results obtained from the NBC, KBANN and SVM classifiers. The choice to use binary features, instead of numerical, made a big impact on both the result and training times.

The results obtained from the cross validation experiments differed very little compared to when using the same data for both training and validation, the only noticeable difference was for the NUS corpus (see section 5.1). This points to low overfitting which is good.

No overfitting can also be a sign of too low dimensionality in the data representation. If the dimensionality is too low, no classifier will be able to separate it effectively. The goal is to create a separation as good as possible without causing overfitting.

If numerical features would have been used the dimensionality would be infinite, since there would have been an infinite number of possible vectors. This might make a more complex and better separation of data possible, but the higher the dimension, the harder it is for a classifier to find a good separation. Experiments were conducted using numerical features, such as normalized TFIDF score and also separating TF and IDF. The results did not improve and testing became extremely time consuming. It is possible that further tests and deeper analysis of the issue would yield better results.

One of the biggest impacts of choosing binary features was that the training times for the KBANN and SVM classifier were dramatically decreased. The training time of these classifiers is approximately proportional to the number of training data. If using numerical features, the number of training data is roughly equal to the number of extracted candidates for the current full corpus, e.g. the number of NP-chunk candidates for the Medicin corpus is about 208000.

Using binary features enables merging of equal pieces of training data by setting the weight to the number of pieces of training data an instance actually represent. This also impose an upper bound on the number of training data for each language, which is dependent on the number of features. For English the boundary is 2240 and for Swedish it is 960. The actual number of training data is much lower, averaging about a third of the reported numbers. This is because there are combinations that are very unlikely, e.g. if TFIDF High, one of the PoS features and the rest of the miscellaneous features are active and the current candidate is not a keyword. This corresponds to a speed increase in the order of 100, compared to using numerical features.

The KBANN (using PSO) and SVM classifiers report similar results and because of this there are reasons to believe that the classifiers themselves is not the limiting factor, the candidate representation is. If this project was to be revamped, the candidate representation would be given much needed attention since it plays such an important role throughout the module. For example, the result effects and implicit dependencies between the features need to be investigated further. The number of features are probably too many than necessary to achieve the results presented in this report.

5.3 Individual components

In this section the interesting individual components used in this project and their properties are discussed.

5.3.1 Text Tokenization

This area received little attention from the very start of the project. It was decided early that it would not be a priority of the project to develop a sophisticated text parser and it was assumed that the input to the module would be pure Unicode text. Also, no time was spent searching an existing component that could be used in the project.

Changes made to the implemented parser turned out to have a great effect on the F-measure. An example is splitting sentences on commas, colons and semicolons in addition to punctuation, question mark and exclamation mark that increased the F-measure with an average of 2%. Spending time implementing or finding an existing more sophisticated parser would surely improve the results further. Tokens such as formulas and other “non-words” can most likely be removed without removing a keyword or a part of a keyword.

The only meta information considered by the parser is the title of each document. This could be extended with chapter titles, abstracts, and bold and italic text to enable more keyword features.

5.3.2 Part-of-Speech and Noun Phrase-Chunking

A possible source of negative influence on the results is the PoS tagger used in this project: Hunpos. The PoS and NP-chunk candidate selection methods depend on it as well as all classifiers except Word frequency and TFIDF. The accuracy has not been measured or been verified from other sources. It is also the second biggest bottleneck regarding classification speed.

A possible extension and improvement of the PoS candidate selection method is to consider context when extracting candidates. The context would be represented by a window, spanning a number of words before and after the phrase of interest. This could possibly create a more discriminating result without having to surrender too much recall.

The NP-chunk method produce the best results, both as stand-alone and when paired up with classifiers, with a few exceptions. As shown in table B.1 it is able to reach a high precision while maintaining an acceptable recall. But, the recall is still considerably lower than both the N-grams and PoS methods and the share of missed keywords range between 17-32%. It would therefore be very interesting to investigate if there are any patterns amongst the missed keywords and complement with these, if the precision does not take to much damage.

The NP-chunker is the biggest bottleneck with regards to classification speed. This is understandable when considering the amount of information it needs to process and the complexity of the task itself due to ambiguities in the written language and in the grammar it uses. The complexity of the English NP-chunker is not known, but the Swedish has a complexity of $\mathcal{O}(n^2)$. It is possible that it can be rewritten to be linear with a large hidden factor, but it would take a considerable amount of work.

5.3.3 TextRank and ExpandRank

The original TextRank and ExpandRank algorithms[16, 21] differ from the implementation used in this report, mainly in the keyword merging process. An attempt was made to implement the original TextRank, which gave similar F-measure results to the original but differed significantly in the balance of precision and recall. This version also yeilded much worse overall results, but was better on the Inspec corpus.

When the words with highest word score have been selected, they can be merged in many different ways. It is stated in [16] that only words that were taken out as keywords were merged if they were adjacent in the text. The results were hard to recreate using this method on the Inspec corpus.

When using other methods, such as the one mentioned in section 3.3, that favored keywords of token size 2 or 3 when extracting keywords from English documents, the F-measure increased. The damping factor is usually defined as $d = 0.85$, when using this setting the implementation gave lower test results compared to the final setting of $d = 0.01$.

The results in [21] (ExpandRank) show that using a neighborhood can increase the F-measure. It is hard to pinpoint why the implementation used in this report give lower results than the TextRank algorithm. They use a small, publicly available corpus consisting of news articles that they annotated themselves. The quality of the keywords can therefore be questioned and it would be interesting to run their implementation on Inspec and compare to the original TextRank algorithm.

Manual evaluation of the extracted keywords shows that the quality of the keywords are good even though both algorithms miss the manually assigned keywords.

5.3.4 Knowledge-Based Artificial Neural Network

The use of knowledge, in the form of feature probabilities, did give the expected results. There was no noticeable improvement in initial F-measure nor in training time. It is possible that a different type of representation would be more beneficial, but has not been tested in this project.

There is room for improving the structure used in the underlying ANN, especially in connection to the number of features. The goal would be to reduce the number of neurons as much as possible without comprising the results. This would speed up training time even more and it should be possible to at least halve the training time.

The quality of the data separation is believed to be close to optimal when optimizing the network with PSO. The reason for this hypothesis is that the SVM classifier show very similar results, which perform very complex separations compared to the other machine learning classifiers. The suspicions that the performance of the KBANN classifier would be poor, mainly due to the high dimensionality of the optimization, turned out to be unfounded.

Comparisons were performed between basic PSO and PSO enhanced with function stretching. There were clear differences when testing on the *Levy No 5* function[19], but would probably not have been needed when separating the data used in this project. However, the extra computations needed for function stretching are negligible and it adds a safety net to the PSO algorithm. No investigation of the complexity of the data used in this project was performed, simply because it is very hard to do.

One variant of the Backpropagation algorithm that would be interesting to test is "Batch-BP". The difference to the normal Backpropagation algorithm is that it evaluates the whole set of training data and then calculates an average error, which is propagated backward.

5.4 Evaluation approaches

In this section different evaluation methods are discussed. It is important to have a performance measure that is as good as possible in order to know which direction that will give better results. This is especially important when using such optimization algorithms such as Particle Swarm Optimization that, in this project, uses the F-measure as a fitness function.

5.4.1 The F-measure

In this project the F-measure is used to measure the performance of a keyword classifier. It is the standard method for this purpose, but is far from optimal. It only gives a guarantee that the measured classifier is "at least this good" and even then there is an assumption that the manual keywords are good.

Rather than a measure of performance, it can be seen as a combination of the level of consistency in manual keywords and to which degree some algorithm has succeeded in following the observed patterns in the manual keywords.

This introduces a theoretical upper limit, an algorithm can only attempt to conform itself to the observed patterns. If there are no clear patterns in the underlying model, the classifier will perform bad. There is a delicate balance between following the observed patterns and avoiding overfitting. The upper bound exist in the model, namely how keyword candidates are represented and what features are accounted for. If low quality results are obtained, it is likely that the model is flawed (too specific or too general) and/or the training data is noisy.

Natural language is ambiguous and written text is difficult to parse, this will introduce errors when translating the text. An algorithm can only be as good as its underlying model and a model can only be as good as the data it is built on.

Also, just because a low F-measure is acquired does not mean that the extracted keywords are bad. It just means that the classifier has not been able to follow the observed manual keyword patterns. The most accurate method to measure the performance of a classifier is probably to perform a manual survey, where people read documents and then categorize the automatically assigned keywords into "good", "bad" and "neither". The primary goal should be to reduce the number of "bad" keywords and the secondary to reduce the "neither" keywords. However, this process is very time consuming and therefore not interesting to use in iterative development, only to obtain a more precise measure of a final result.

5.4.2 What constitutes a keyword hit?

In the training data used for this project each document has a set of manual keywords. When an algorithm extracts a keyword that is in the set of the manual keywords for that document, it is called a *keyword hit*. However, only accounting for exact matches is a very pessimistic approach.

The most common method is to perform lemmatized (or stemmed) comparison, i.e. if lemmas of two phrases are equal they are considered to be equal. This means that the two phrases "brown dog" and "brown dogs" are equal, because the words *dog* and *dogs* have the same lemma.

A more optimistic approach is to also account for subsuming keywords, in addition to the lemmatized comparison. This means that if all tokens in the manually assigned keywords are present in the automatically extracted keyphrase and their order is intact, it will constitute as a keyword hit. The idea behind this approach is that the automatically extracted keyword is at least as specific as the manual keyword, information can only be added that makes it more specific to the current document. This means that the manual keyword “dog” will be considered equal to the automatically extracted keyword “the brown dog”. Unfortunately this means that unwanted information can be added in the automatically extracted keywords. However, this method is still believed, by the authors of this paper, to be a too pessimistic measure of the true performance of a keyword extraction algorithm.

To get the most accurate measure of an algorithms performance, one also need to account for synonyms and semantic relations between words. This is believed to be especially important when classifying technical papers where writers tend to vary their language in order to prevent repetition. The two words “dog” and “canine” would then be considered equal, since they refer to the same physical object. Semantic relations between words could also be used to enhance the evaluation accuracy. An automatically extracted keyword is only compared to the manual keyword it has the strongest semantic relation to. If the strength of the relation exceeds some predefined threshold, it constitutes a keyword hit. It is also possible to account for fractions of a hit, proportional to the strength of the semantic relation.

This would be a more accurate *automatic* evaluation method, but it still does not account for good keywords that just were not in the set of manual keywords. Therefore, to get the most accurate measure a manual study should be carried out (as stated in 5.4.1).

5.4.3 How many keywords to extract?

Turney [17] and Frank [12] performed experiments where they extracted five and fifteen keywords per document. This does not reflect the performance of the classifier since extremely poor keywords might get extracted due to a lack of proper candidates. A cutoff value can be used to prevent this from happening.

Zesch and Gurevych[5] has slightly different and better approach. They account for the result of a document if the number of automatically extracted keywords and manually assigned keywords are equal . The drawback of this method is that the number of documents that contribute to the final result may be small and thus the result may vary from corpus to corpus. To help prevent this one can use corpora with a high number of documents, such as Inspec ($\approx$2000 documents).

The classifiers in this project use two different approaches to solve this problem. The non-binary classifiers extract the same number of keyword each document in the training data has been manually assigned. Note that this is only during evaluation experiments, in live classification the desired number of keywords is passed as a parameter to the classifier.

The binary classifiers conform themselves during training to extract the same number of keywords each manual document has been assigned. They only extract keywords that fulfill their requirements. This is consistent with regard to the number of keywords and the keywords themselves assigned manually to the documents in the training data as the gold standard.

5.5 Result discussion

The overall results of this project is not be as good as initially hoped, but they are satisfactory.

Fast ESP, the search engine used by Findwise AB which has the possibility to extract keywords, has considerably worse performance than the best classifiers used in this project, especially for Swedish documents. The classifiers in this project are also faster, but they can only extract keywords. When classifying documents, Fast ESP performs additional operations such as entity extraction. Because of this the speed performance is hard to measure, but if keywords are the only subject of interest there is a clear winner with a factor of about three.

Hulth's results remain impressive and is in another league than those reported in this project. Even though the methods used in both projects may look similar, such as the features and candidate selection methods, there is still a big difference. The known differences are the PoS tagger, NP-chunker and the machine learning algorithm. Since the best classifiers show very good performance, it is believed that a big difference lies in the PoS tagger, NP-chunker and the candidate representation. It is also possible that Hulth's partitioning of the Inspec corpus influences the result, but should not account for a difference more than maybe a few percentages in F-measure.

The classifiers, especially KBANN and SVM, show impressive results and would be a strong foundation to base further work on. By optimizing the candidate representation and the features it is very likely that the results would improve. Another possible improvement could be to test other external tools, in addition to the PoS tagger and NP-chunker, though this would most likely limit the supported languages to English only.

The big disappointment of this project is the poor results obtained from the different ensemble methods that were tested. No combination of classifiers, representations and training data gave any noticeable improvement over single classifiers what so ever. It is believed that the reason is the simplicity of creating a close to optimal separation of the training data, every classifier in the ensembles simply classify nearly the same subset of candidates as keywords.

Chapter 6

Conclusion

There are no known distinct statistical or linguistic patterns for finding keywords. Human annotators disagree to a great extent[22]. If an obtained F-measure is too high, this is likely a result of overfitting.

By using linguistic information in candidate selection methods, such as PoS and NP-chunking, increases the performance significantly compared to N-grams.

Translating text into numeric representations is hard. Much information is lost when words are converted into a numerical feature vectors and the generated data is partly contradictory.

Boolean features limited the number of permutations of candidate vectors. This lowered training times drastically and prevented overfitting but may have narrowed the complexity of data separation.

Though simplistic in design, the NBC classifier is very powerful and achieves near similar results to the more sophisticated classifiers used in this report.

The TextRank classifier show great potential and can be compared to the best supervised algorithms. The low F-measures compared to [16] is believed to be caused by a lacking implementation. Manual inspection of extracted keywords indicate good quality despite lacking results.

The knowledge model used in the KBANN classifier did not meet the expectations and the result influence, if any, was negative. A basic ANN was sufficient.

Enhancing the PSO algorithm with function stretching was probably unnecessary, considering the simplicity of the feature vectors. The extra computations are negligible and improves accuracy significantly, which could prove useful in applications of greater complexity.

The manual keywords were regarded as gold standard, but turned out to be of questionable quality. The quality of keywords extracted from documents in a live situation will be lower than if the manual keywords would have been better.

This report shows that combining NLP and ML algorithms can improve keyword quality, compared to other methods such as TFIDF. The results are satisfactory and outperform the Fast ESP suite used at Findwise AB to date.

6.1 Future work

With the implemented module and report as a foundation, there several improvements and extensions that are possible to enhance the results:

General corpus Employ a bigger and more general document set, with regards to content, to calculate IDF scores.

Parsing More sophisticated parser with support for additional meta info.

Missed keywords Investigate possible patterns in keywords not extracted by the NP-chunker.

Additional candidate selection method Use of dependency grammar and hyperonyms.

TextRank Improved keyword merge method and NP-chunk extension.

PSO More extensive use for optimization of various parameters.

Backpropagation Batch BP extension.

Boosting Use the boosting ensemble method.

Semantic relations Use of WordNet and Saldo to measure the strength of semantic relations between pair of words.

Manual keyword quality surveys Human evaluation is necessary to find actual quality of extracted keywords.

Bibliography

[1] Kristin P. Bennett, Colin Campbell. Support vector machines: Hype or halleluja? 2000.

[2] Nguyen Thuy Dung. Automatic keyphrase generation. 2006.

[3] James Kennedy, Russell Eberhart. Particle swarm optimization. 1995.

[4] Ram Dayal Goyal. Knowledge based neural network for text classification. 2007.

[5] T. Zesch, I. Gurevych. Approximate matching for evaluating keyphrase extraction. 2009.

[6] Simon Haykin. *Neural Networks and Learning Machines*. Peason, 2009. ISBN 978-0-13-129376-2.

[7] Anette Hulth. Combining machine learning and natural language processing for automatic keyword extraction. 2004.

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

[9] Jon M. Kleinberg. Authoritative source in a hyperlinked environment. 1999.

[10] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. ISBN 0-07-115467-1.

[11] Larry Page, Sergey Brin, Rajeev Motwani. The pagerank citation ranking: Bringing order to the web. 1999.

[12] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, Craig C. Nevill-Manning. Domain-specific keyphrase extraction. 1999.

[13] Warren S. McCulloch, Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. 1943.

[14] M.F. Porter. An algorithm for suffix stripping. 1980.

[15] Amit Singhal. Modern information retrieval: A brief overview. 2001.

[16] Rada Mihalcea, Paul Tarau. Textrank: Bringing order into texts. 2004.

[17] Peter D. Turney. Learning algorithms for keyphrase extraction. 1999.

[18] Vladimir N. Vapnik. The nature of statistical learning theory. 1995.

[19] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas, M.N. Vrahatis. Stretching technique for obtaining global minimizers through particle swarm optimization. 2000.

[20] Mattias Wahde. *Biological Inspired Optimization Methods.* WIT Press, 2008. ISBN 978-1-84564-148-1.

[21] Xiaojun Wan, Juanguo Xiao. Single document keyphrase extraction using neighbourhood knowledge. 2008.

[22] M.E. Dexter, P Zunde. Indexing consistency and quality. *American Documentation*, 20(4):259–267, 1969.

Appendix A

PoS Patterns

The full list of tags, for each language, used by the PoS candidate selection method are presented in section. The patterns extracted are based on statistics taken from the attached keywords in the corpora, every keyphrase pattern occur at least 10 times.

Table A.1: *PoS patterns.*

English			
jj nn	vbg nn	vbn nn nn	nn nnp
nn nn	vbg	nnp nnp nn	jj nn jj nn
jj nns	vbg nns	nn jj nns	nnp nnp nns
nn	nnp nnp nnp	jj vbg nn	cd nn
nn nns	jj vbg	nn jj	nnp vbg
jj nn nn	nn jj nn	nn in nn	nn vbg nns
nnp	nn vbg	nn vbg nn	nn vb
nns	jj nnp nn	cd nnp	vbn jj nn
nnp nn	nnp jj nn	rb jj nn	nn jj nn nn
jj nn nns	vbn nn	jj nnp nns	vbn nn nns
nn nn nn	nnp nn nns	jj jj jj nn	nns in nn
jj jj nn	jj nnp	jj nn vbg	vbz
nnp nns	nns nn	jj nns nn	nns nns
jj jj nns	jj jj	vbg nn nn	
nn nn nns	jj jj nn nns	rb jj nns	
jj	jj nn nn nns	vb nn	
nnp nnp	nn nn nn nn	nnp nn nn nn	
jj nn nn nn	vbn nns	vbn jj nns	
nnp nn nn	nn vbz	nnp jj nns	
jj jj nn nn	jj vbg nns	nnp nnp nnp nnp	

Swedish	
nn	pc
pm	pc nn
jj nn	nn kn jj
jj	pm pm
nn kn nn	nn pp nn
vb	

Appendix B

Extensive result tables

The test results presented in this section are run on computers with 2.1-2.4 GHz processors. The runtimes and training times are not to be interpreted as 100% accurate, they are only presented to give an indication of the speed. Only when there is a full order of magnitude in difference between two runtimes, it is safe to assume that there is a significant difference in speed (e.g. 5s and 2m).

Training times for the supervised machine learning algorithms are not included in their individual results and are presented separately in section B.9.

Explanations of the table column names used in this section:

- **Runtime** - The time it takes to run the test.
- **Assign. tot.** - Total number of keywords extracted.
- **Assign. mean** - Mean number of keywords extracted for each document.
- **Corr. tot.** - Total number of correct keywords extracted.
- **Corr. mean** - Mean number of correct keywords per document.
- **P** - Precision.
- **R** - Recall.
- **F** - F-measure.

B.1 Candidate selection methods

The results below concern only candidate selection methods, i.e. no classification algorithms are applied on the extracted candidates.

The runtimes for PoS include the time it takes to tag the documents with PoS tags. The actual time it takes to find the pattern matches are comparable to the runtimes for N-grams.

The runtimes for NP-chunk does not include the time it takes to tag the documents with PoS tags.

Table B.1: *Statistics for the candidate selection methods used in this project. The best results for each corpus are shown in bold.*

	Runtime	Assign. tot.	Assign. mean	Corr. tot.	Corr. mean	P	R	F
Medicin								
N-grams	5s	798 753	1 190.39	3 586	5.35	0.45	**99.98**	0.90
PoS	1m 24s	430 693	641.87	3 478	5.19	0.81	96.97	1.61
NP-chunk	3m 17s	208 745	311.10	2 704	4.03	**1.30**	75.39	**2.55**
Socialstyrelsen								
N-grams	4s	491 077	333.39	3 445	2.35	0.71	**97.05**	1.40
PoS	1m 11s	286 992	198.84	3 404	2.32	1.19	95.89	2.35
NP-chunk	3m 10s	139 541	97.73	2 621	1.79	**1.88**	73.84	**3.67**
Inspec								
N-grams	1s	222 107	111.72	13 343	6.71	6.01	**91.68**	11.28
PoS	12s	152 712	76.82	13 251	6.67	8.68	91.05	15.85
NP-chunk	1m 1s	53 622	26.97	9 655	4.86	**18.01**	66.34	**28.33**
NUS								
N-grams	3s	499 147	3 283.86	1 163	7.66	0.24	93.87	0.47
PoS	1m 34s	294 716	1 938.92	1 164	7.66	0.40	**93.95**	0.79
NP-chunk	3m 55s	143 855	946.41	1 031	6.79	**0.72**	83.22	**1.43**

B.2 Word frequency

In the results presented for this classifier, the number of keywords extracted is the same as the number of manual keywords for each document. The runtimes does not include the time it takes to extract candidate keywords.

Table B.2: *Statistics for the Word frequency classifier. The best results for each corpus are shown in bold.*

	Runtime	Corr.	Corr.	F
Medicin		tot.	mean	
N-grams	34s	655	0.98	18.27
PoS	18s	550	0.82	15.34
NP-chunk	42s	753	1.13	**21.00**
Socialstyrelsen				
N-grams	15s	675	0.46	19.02
PoS	11s	661	0.45	18.62
NP-chunk	5s	842	0.58	**23.72**
Inspec				
N-grams	5s	2 207	1.11	15.17
PoS	5s	3 210	1.61	22.06
NP-chunk	20s	3 703	1.86	**25.45**
NUS				
N-grams	46s	140	0.93	11.30
PoS	24s	106	0.70	8.56
NP-chunk	1m 6s	153	1.01	**12.35**

B.3 TFIDF

In the results presented for this classifier, the number of keywords extracted is the same as the number of manual keywords for each document. The runtimes does not include the time it takes to extract candidate keywords.

Table B.3: *Statistics for the TFIDF classifier. The best results for each corpus are shown in bold.*

	Runtime	Corr.	Corr.	F
Medicin		tot.	mean	
N-grams	26s	1 018	1.52	28.39
PoS	14s	1 047	1.57	**29.19**
NP-chunk	12s	1 035	1.55	28.86
Socialstyrelsen				
N-grams	12s	621	0.43	17.50
PoS	9s	721	0.49	20.31
NP-chunk	3s	838	0.57	**23.61**
Inspec				
N-grams	3s	2 401	1.21	16.50
PoS	3s	3 715	1.87	25.53
NP-chunk	1s	3 966	1.99	**27.25**
NUS				
N-grams	39s	212	1.40	17.12
PoS	23s	184	1.22	14.86
NP-chunk	19s	223	1.47	**18.00**

B.4 TextRank

Both of the TextRank- and Expand classifiers are not dependent on having candidate keywords given to them when classifying a document.

B.4.1 TextRank

Table B.4: *Statistics for the TextRank classifier.*

Corpus	Runtime	Corr. tot.	Corr. mean	F
Medicin	2m 12s	693	1.04	19.32
Socialstyrelsen	1m 48s	710	0.49	20.01
Inspec	31s	4 627	2.33	31.84
NUS	5m 52s	83	0.55	6.87

B.4.2 ExpandRank

Table B.5: *Statistics for the ExpandRank classifier.*

Corpus	Runtime	Corr. tot.	Corr. mean	F
Medicin	7m 36s	318	0.48	8.87
Socialstyrelsen	11m 18s	342	0.24	9.64
Inspec	5m 31s	4 423	2.22	30.41
NUS	7m 15s	64	0.43	5.17

B.5 NBC

In the results presented for this classifier, the number of keywords extracted is the same as the number of manual keywords for each document. To avoid an overfitted result, the results presented for the NBC classifier is the average from 10 runs of 10-fold cross validation. The runtimes does not include the time it takes to extract candidates, only the classification of a full corpus.

Table B.6: *Statistics for the NBC classifier. The best results for each corpus are shown in bold.*

	Runtime	Corr.	Corr.	F
Medicin		tot.	mean	
N-grams	59s	1 008	1.51	28.10
PoS	28s	1 046	1.56	**29.17**
NP-chunk	32s	1 019	1.52	28.40
Socialstyrelsen				
N-grams	31s	791	0.54	22.29
PoS	19s	915	0.62	25.78
NP-chunk	24s	1 033	0.70	**29.09**
Inspec				
N-grams	10s	4 491	2.26	30.86
PoS	8s	5 038	2.53	34.61
NP-chunk	25s	5 101	2.57	**35.05**
NUS				
N-grams	1m 24s	272	1.79	21.97
PoS	46s	248	1.63	20.03
NP-chunk	2m 8s	293	1.93	**23.62**

B.6 KBANN

The KBANN classifier is based on a supervised machine learning algorithm, therefore 10-fold cross validation is used. The results presented are the average results in each category from 10 runs.

The runtimes does not include the time it takes to extract candidates, only the classification of a full corpus. The classification time is independent of which optimization algorithm that has been used, therefore the runtimes reported for BP and PSO equal.

B.6.1 Backpropagation

Table B.7: *Statistics for the KBANN classifier using a network trained by BP. The best results for each corpus are bold.*

	Runtime	Assign.	Assign.	Corr.	Corr.	P	R	F
Medicin		tot.	mean	tot.	mean			
N-grams	32s	89 650	133.61	1 264	1.88	1.51	35.24	2.90
PoS	20s	59 130	88.12	1 283	1.91	2.47	35.77	4.62
NP-chunk	28s	51 775	77.16	1 292	1.93	**2.86**	**36.02**	**5.30**
Socialstyrelsen								
N-grams	21s	62 078	42.14	1 131	0.77	1.98	31.87	3.73
PoS	15s	57 753	39.21	1 374	0.93	2.46	**38.69**	4.63
NP-chunk	27s	35 158	23.87	1 304	0.89	**3.71**	36.73	**6.74**
Inspec								
N-grams	9s	94 751	47.66	8 962	4.51	9.46	61.58	16.40
PoS	8s	68 999	34.71	10 545	5.30	15.29	**72.45**	25.25
NP-chunk	21s	49 405	24.85	8 919	4.49	**18.06**	61.28	**27.89**
NUS								
N-grams	1m 3s	3 878	25.51	308	2.03	7.94	24.86	12.04
PoS	37s	3 440	22.63	313	2.06	**9.10**	25.26	**13.38**
NP-chunk	1m 48s	5 097	33.53	372	2.45	7.30	**30.02**	11.74

B.6.2 Particle Swarm Optimization

Table B.8: *Statistics for the KBANN classifier using a network trained by PSO. The best results for each corpus are bold.*

	Runtime	Assign.	Assign.	Corr.	Corr.	P	R	F
Medicin		tot.	mean	tot.	mean			
N-grams	32s	2 753	4.10	973	1.45	35.34	27.12	30.69
PoS	20s	2 760	4.11	990	1.48	35.87	**27.60**	**31.20**
NP-chunk	28s	2 314	3.45	881	1.31	**38.07**	24.56	29.86
Socialstyrelsen								
N-grams	21s	3 996	2.71	1 053	0.71	26.35	**29.66**	27.91
PoS	15s	3 760	2.55	994	0.67	26.44	28.00	27.20
NP-chunk	27s	3 474	2.36	1 018	0.69	**29.30**	28.68	**28.99**
Inspec								
N-grams	9s	30 678	15.43	8 310	4.18	27.09	**57.09**	36.74
PoS	8s	21 334	10.73	6 270	3.15	29.39	43.08	34.94
NP-chunk	21s	23 384	11.76	7 341	3.69	**31.39**	50.44	**38.70**
NUS								
N-grams	1m 3s	1 395	9.18	297	1.95	21.29	23.97	22.25
PoS	37s	1 421	9.35	284	1.87	19.99	22.92	21.35
NP-chunk	1m 48s	1 357	8.93	298	1.96	**21.96**	**24.05**	**22.96**

B.7 SVM

Since this is a supervised machine learning algorithm, 10-fold cross validation is used. The results presented are the average results in each category from 10 runs. The runtime measures a single classification of a full corpus including the time it takes to extract the candidates.

Table B.9: *Statistics for the SVM classifier. The best results for each corpus are bold. An outlier with 15.27 precision caused the precision for NP-chunk on Socialstyrelsen to drop by about 1.5 points.*

Medicin	Runtime	Assign. tot.	Assign. mean	Corr. tot.	Corr. mean	P	R	F
N-grams	2m 5s	2 650	3.95	953	1.42	35.96	26.57	30.56
PoS	1m 11s	2 667	3.97	976	1.45	36.60	**27.21**	**31.21**
NP-chunk	56s	2 224	3.31	868	1.29	**39.03**	24.20	29.88
Socialstyrelsen								
N-grams	1m 16s	4 306	2.92	1 115	0.76	25.92	31.42	28.41
PoS	49s	4 339	2.95	1 117	0.76	25.75	**31.47**	28.32
NP-chunk	31s	3 851	2.61	1 029	0.70	**27.88**	28.98	**28.42**
Inspec								
N-grams	8s	22 667	11.40	5 894	2.96	26.00	40.49	31.67
PoS	7s	20 776	10.45	6 060	3.05	29.17	41.64	34.31
NP-chunk	24s	23 869	12.01	7 452	3.75	**31.22**	**51.20**	**38.79**
NUS								
N-grams	1m 54s	1 538	10.12	307	2.02	19.96	**24.78**	22.11
PoS	1m 10s	1 351	8.89	265	1.74	19.62	21.39	20.47
NP-chunk	2m 12s	1 278	8.41	298	1.96	**23.32**	24.05	**23.68**

B.8 Ensembles

In this section the results for some of the tested ensemble methods are presented. All ensembles contain at least one supervised classifier and 10-fold cross validation is therefore used. Training ensembles take considerable longer time than single classifiers, so due to time constraints the result are from a single run only.

Several other ensemble combinations were tested, but no result was noticeably better than the best single classifiers. Those result are therefore omitted from this report.

B.8.1 Regression

Table B.10: *Statistics for the Regression classifier.*

	Runtime	Assign.	Assign.	Corr.	Corr.	P	R	F
Corpus		tot.	mean	tot.	mean			
Medicin	41s	3 278	4.89	1 016	1.51	31.00	28.33	29.60
Socialstyrelsen	1m 16s	4 202	2.85	960	0.65	22.85	27.05	24.77
Inspec	52s	27 353	13.76	8 136	4.09	29.74	55.90	38.82
NUS	2m 22s	1 475	9.70	276	1.82	18.72	22.28	20.34

B.8.2 KBANN ensemble

This classifier consist of seven individual KBANN classifiers, each trained on a different subset of training data and use NP-chunk candidates.

Table B.11: *Statistics for the KBANN ensemble classifier using NP-chunk candidates.*

	Runtime	Assign.	Assign.	Corr.	Corr.	P	R	F
Corpus		tot.	mean	tot.	mean			
Medicin	46s	2 292	3.42	884	1.32	38.57	24.65	30.08
Socialstyrelsen	37s	3 432	2.33	1 018	0.69	29.67	28.68	29.17
Inspec	15s	23 503	11.82	7 380	3.71	31.40	50.70	38.78
NUS	1m 20s	1 336	8.79	309	2.03	23.13	24.94	24.00

B.8.3 KBANN and TextRank

This ensemble classifier consist of a single KBANN classifier using NP-chunk candidates and a single TextRank classifier. The final result is the union of keywords extracted by both classifiers.

Table B.12: *Statistics for the KBANN-TextRank ensemble classifier using NP-chunk candidates.*

Corpus	Runtime	Assign. tot.	Assign. mean	Corr. tot.	Corr. mean	P	R	F
Medicin	1m 13s	8 207	12.23	1 297	1.93	15.81	36.16	22.00
Socialstyrelsen	1m 23s	9 179	6.23	1 559	1.06	16.99	43.92	24.50
Inspec	10s	29 668	14.92	8 371	4.21	28.22	57.51	37.86
NUS	2m 59s	3 396	22.34	363	2.39	10.69	29.30	15.67

B.9 Training times

The times presented in table B.13 are the average of ten training sessions.

The training data is created from candidates extracted using the N-grams method, this is motivated by that it extracts the most candidates by far to create a worst case scenario. If training is performed on NP-chunk candidates, the time for extracting them needs to be included in the times presented in table B.13. The times for extracting the N-grams can then be ignored since they are so small (see table B.1).

The NBC classifier is the only trained classifier whose training times depend heavily on the number of extracted candidates. The size of the training data used for the rest of the trained classifiers is restricted by the number of features (see section 2.4). Only identical pieces of training data are kept and are given weights that correspond to the actual number of instances they correspond to.

Table B.13: *Training times for the supervised algorithms.*

Algorithm	Medicin	Socialstyrelsen	Inspec	NUS
NBC	34s	22s	10s	1m 4s
KBANN-BP	44s	34s	28s	1m 10s
KBANN-PSO	3m 17s	2m 45s	7m 4s	5m 35s
SVM	58s	65s	72s	63s

The training times for the Regression classifier is approximately equal to three times the training time for a single NBC classifier. The training time for the KBANN ensemble is approximately equal to the training time required for a single KBANN classifier multiplied with the number of classifiers used in the ensemble. The training time of the KBANN-TextRank classifier is equal to the training time of a single KBANN classifier.