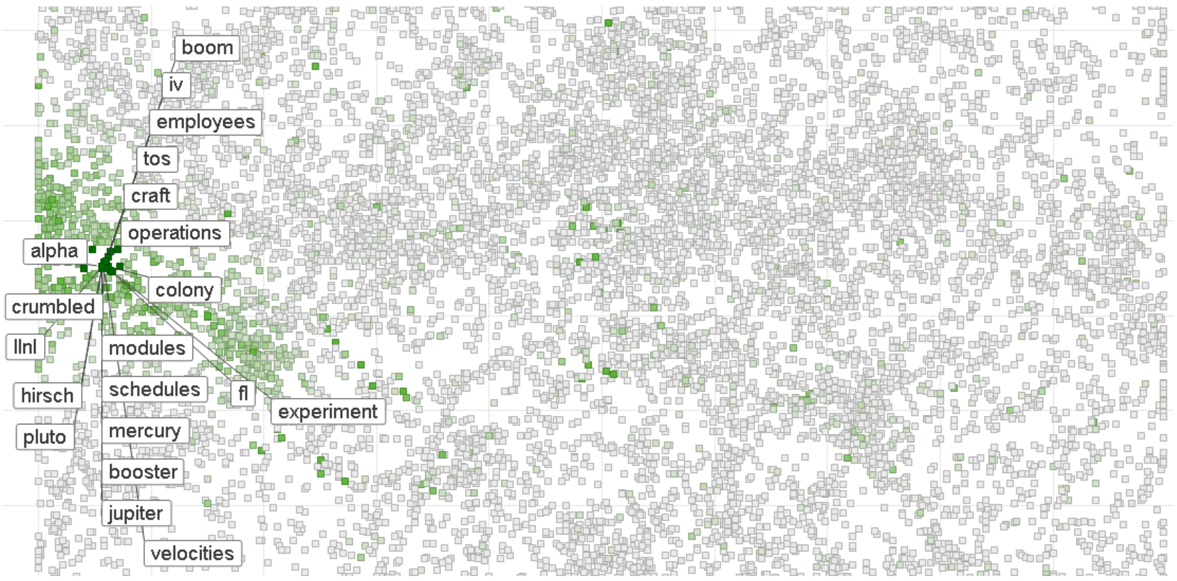


Text Analysis
Exploring latent semantic models for information retrieval, topic modeling and sentiment detection

Master of Science Thesis

ERIK JALSBORN
ADAM LUOTONEN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2011

Text Analysis
Exploring latent semantic models for information retrieval,
topic modeling and sentiment detection

ERIK JALSBORN
ADAM LUOTONEN

Examiner: DEVDATT DUBHASHI

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

Cover:
Words from the 20NewsGroups corpus projected to two dimensions using Latent Dirichlet Allocation followed by a self-organizing map, shown at page 66.

Department of Computer Science and Engineering
Göteborg, Sweden June 2011

Preface

We would like to express our gratitude to TIBCO Software AB and the people working there for making this thesis possible. Especially we would like to thank our supervisor Christoffer Stedt for general guidance, helping out with Spotfire specifics and for proof reading. We would also like to thank Leif Grönqvist for his input regarding text analysis techniques. Lastly, we would like to thank our examiner Devdatt Dubhashi and Vinay Jethava at Chalmers for helpful discussions.

Adam Luotonen and Erik Jalsborn, June 2011

Abstract

With the increasing use of the Internet and social media, the amount of available data has exploded. As most of this data is natural language text, there is a need for efficient text analysis techniques which enable extraction of useful data. This process is called text mining, and in this thesis some of these techniques are evaluated for the purpose of integrating them into the visual data mining software TIBCO Spotfire®.

In total, five analysis models with different running time, memory use and performance have been analyzed, implemented and evaluated. The tf-idf vector space model was used as a baseline. It can be extended using Latent Semantic Analysis and random projection to find latent semantic relationships between documents. Finally, Latent Dirichlet Allocation (LDA), Joint Sentiment/Topic model (JST) and Sentiment Latent Dirichlet Allocation (SLDA) are used to extract topics. The latter two are extensions to LDA which also detects positive and negative sentiment.

Evaluation was done using the perplexity measure for topic modeling, average precision for searching and classification accuracy of positive and negative reviews for the sentiment models. It was concluded that for searching, a vector space model with tf-idf weighting had similar performance compared to the latent semantic models for the test corpus used. Topic modeling showed to provide useful output, however at the expense of running time. The JST and SLDA sentiment detectors showed a small improvement compared to a baseline word counting classifier, especially for a multiple domain dataset. Finally it was shown that they had mixed sentiment classification accuracy from run to run, indicating that further investigation is motivated.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to a widely accepted rule [1, 2] about 80% of all business data is in unstructured form, for example e-mail messages, documents, journals, images, video, audio and so on. Clearly this unstructured data is a huge source of information that businesses and research can take advantage of to find new opportunities, trends and relationships. If we also take into consideration the current boom in social networking such as blogs and instant messaging, the importance of being able to assess unstructured data becomes even more evident. Most of this unstructured data is in text format, which is readable by humans but not easily interpretable by computers.

1.1 Problem

The TIBCO Spotfire®Platform (Spotfire) is a line of products for interactive visualization which includes both an engine for interactive graphics and an engine for high performance database queries. In combination, this makes it possible to interact graphically with large data sets. However, in its current state it offers no text analysis techniques.

1.2 Goal

With this thesis, we explore the possibilities of analyzing and visualizing larger amounts of texts in Spotfire. The aim is to evaluate some text analysis techniques in current research. Specifically, the following features are implemented and evaluated as they could be useful for both current and future Spotfire users.

Topic modeling Extracting topics from a collection of documents and representing the documents as mixtures of these topics, enabling the user to explore the collection and find relationships.

Information retrieval Finding documents relevant to an information need, often defined by a search string.

Sentiment detection Analyzing the sentiment of a text, thus extracting positive and negative aspects.

The purpose of the implementation is not to be a complete text analysis solution, but instead a proof of concept and possible foundation for future development.

1.3 Scope

Many text analysis techniques are language dependent. As most of Spotfire's customers use English data sources, this is the language which is used for evaluation in this thesis.

In the context of search, one often distinguishes between word and content based search. The former is a way of matching strings against strings, where one can accept typos using approximate string matching. This project will only focus on the latter, i.e. content based search.

All the techniques examined are unsupervised, which means that they can be directly applied to data without any prior training with annotated data. The reason for this choice is that Spotfire customers come from different areas and annotated training data may not be available.

1.4 Method

The project has been carried out in three different phases to fulfil the goal. In these phases, the following inquiry was undertaken:

In the first phase, a review of available text analysis techniques were conducted to get an overview of the field. Based on this review a subset of the found techniques was chosen for further investigation. An implementation was then made in the second phase, enabling evaluation of these techniques.

In the last phase the techniques were evaluated in a series of experiments in order to assess their current state, and for measuring future improvements. For topic modeling, experiments on the impact of number of topics were done. The information retrieval experiments were performed by measuring search precision. Finally, experiments were conducted to determine the classification accuracy of the implemented sentiment detectors. The running time and memory use for all implementations was also analyzed, to give a hint of their possible integration points in Spotfire.

1.5 Report outline

Chapter 2 describes the theory needed to understand the rest of the report. First a short introduction to the Spotfire Client is given, followed by a brief overview of text mining. Then all the techniques used are introduced in the order they are executed in the implemented pipeline: preprocessing, analysis and then visualization of the results.

Chapter 3 gives the details of how the techniques were implemented, along with their problems and considerations. A short overview of the testing tools implemented in Spotfire is given in appendix A.

Chapter 4 presents the results from the experiments performed. These results indicates how well the techniques work in their current state. This project evaluates three features: topic modeling, information retrieval (content based search), and sentiment detection. Also some output examples are given to show how the results can be presented to the user.

Chapter 5 discusses the implementation and its possible improvements. Also the results of the experiments are evaluated.

Chapter 6 presents the conclusions made during this thesis. It finally leads up to some recommendations regarding which of the techniques that could be integrated into Spotfire.

Chapter 2

Theory

Before starting the implementation, a literature study had to be done in order get an overview of the current research in the field. In this chapter the result of this study is presented. We first give an overview of what Spotfire is and how it is used. Then some basic concepts and techniques needed to understand the rest of this report are given, such as what a corpus and bag of words is. Finally the techniques for finding latent semantic relationships between documents are explained. The chapter ends with the theory behind self-organizing maps, a technique which can be used to visualize high dimensional data.

2.1 Spotfire

Spotfire is a data analysis and visualization platform [3]. In Spotfire, users can create their own analytic applications which can then be viewed by a large number of users on many different devices like personal computers, cell phones and tablets. See figure 2.1 for an example of an analysis. In this section, we describe briefly how Spotfire can be used.

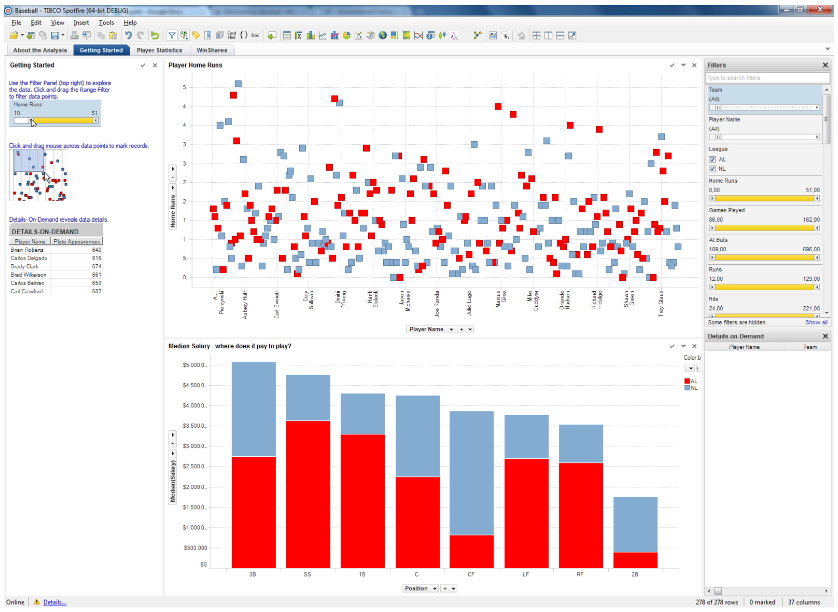
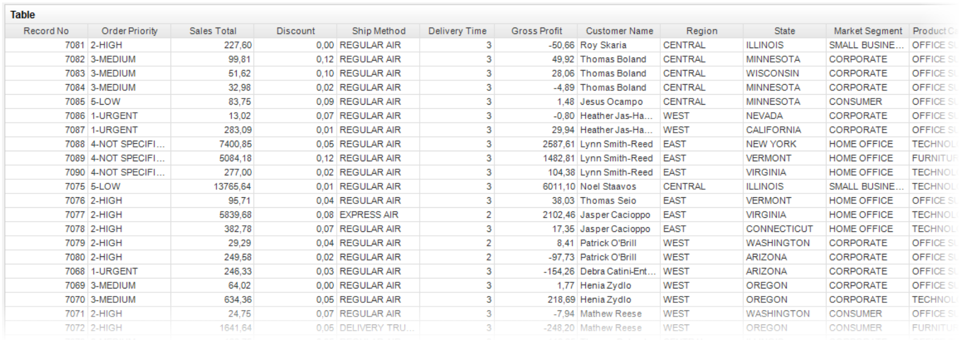


Figure 2.1: The Spotfire client user interface. An analysis consists of one or more pages which contain dynamic visualizations. The filters enable the users to analyze subsets of the data.

The first step of creating an analytic application is to identify the data. It can be data from experiments, web site usage statistics, gene data, sales numbers etc. These are imported into Spotfire as data tables, as can be seen in figure 2.2.

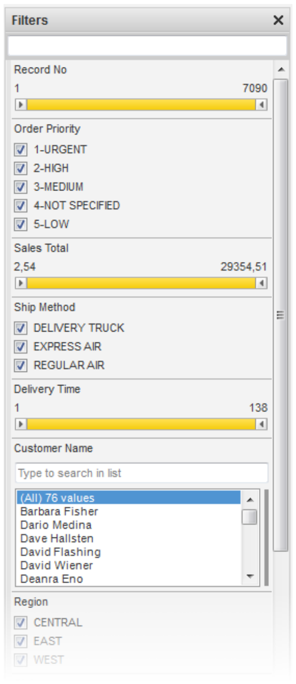


Table

Record No	Order Priority	Sales Total	Discount	Ship Method	Delivery Time	Gross Profit	Customer Name	Region	State	Market Segment	Product C
7081	2-HIGH	227,60	0,00	REGULAR AIR	3	-50,66	Roy Skaria	CENTRAL	ILLINOIS	SMALL BUSINE...	OFFICE S
7082	3-MEDIUM	99,81	0,12	REGULAR AIR	3	49,92	Thomas Boland	CENTRAL	MINNESOTA	CORPORATE	OFFICE S
7083	3-MEDIUM	51,62	0,10	REGULAR AIR	3	28,06	Thomas Boland	CENTRAL	WISCONSIN	CORPORATE	OFFICE S
7084	3-MEDIUM	32,98	0,02	REGULAR AIR	3	-4,89	Thomas Boland	CENTRAL	MINNESOTA	CORPORATE	OFFICE S
7085	5-LOW	83,75	0,09	REGULAR AIR	3	1,48	Jesus Ocampo	CENTRAL	MINNESOTA	CONSUMER	OFFICE S
7086	1-URGENT	13,02	0,07	REGULAR AIR	3	-0,80	Heather Jas-Ha...	WEST	NEVADA	CORPORATE	OFFICE S
7087	1-URGENT	283,09	0,01	REGULAR AIR	3	29,94	Heather Jas-Ha...	WEST	CALIFORNIA	CORPORATE	OFFICE S
7088	4-NOT SPECIFI...	7400,85	0,05	REGULAR AIR	3	2587,61	Lynn Smith-Reed	EAST	NEW YORK	HOME OFFICE	TECHNOL
7089	4-NOT SPECIFI...	5084,18	0,12	REGULAR AIR	3	1482,81	Lynn Smith-Reed	EAST	VERMONT	HOME OFFICE	FURNITUR
7090	4-NOT SPECIFI...	277,00	0,02	REGULAR AIR	3	104,38	Lynn Smith-Reed	EAST	VIRGINIA	HOME OFFICE	TECHNOL
7075	5-LOW	13765,64	0,01	REGULAR AIR	3	6011,10	Noel Staavos	CENTRAL	ILLINOIS	SMALL BUSINE...	TECHNOL
7076	2-HIGH	95,71	0,04	REGULAR AIR	3	38,03	Thomas Seio	EAST	VERMONT	HOME OFFICE	OFFICE S
7077	2-HIGH	5839,68	0,08	EXPRESS AIR	2	2102,46	Jasper Cacioppo	EAST	VIRGINIA	HOME OFFICE	TECHNOL
7078	2-HIGH	382,78	0,07	REGULAR AIR	3	17,35	Jasper Cacioppo	EAST	CONNECTICUT	HOME OFFICE	TECHNOL
7079	2-HIGH	29,29	0,04	REGULAR AIR	2	8,41	Patrick O'Brill	WEST	WASHINGTON	CORPORATE	OFFICE S
7080	2-HIGH	249,58	0,02	REGULAR AIR	2	-97,73	Patrick O'Brill	WEST	ARIZONA	CORPORATE	OFFICE S
7068	1-URGENT	246,33	0,03	REGULAR AIR	3	-154,26	Debra Catini-Ent...	WEST	ARIZONA	CORPORATE	OFFICE S
7069	3-MEDIUM	64,02	0,00	REGULAR AIR	3	1,77	Henia Zydlo	WEST	OREGON	CORPORATE	OFFICE S
7070	3-MEDIUM	634,36	0,05	REGULAR AIR	3	218,69	Henia Zydlo	WEST	OREGON	CORPORATE	TECHNOL
7071	2-HIGH	24,75	0,07	REGULAR AIR	3	-7,94	Mathew Reese	WEST	WASHINGTON	CONSUMER	OFFICE S
7072	2-HIGH	1641,64	0,05	DELIVERY TRU...	3	-248,20	Mathew Reese	WEST	OREGON	CONSUMER	FURNITUR

Figure 2.2: An example data table in Spotfire.

In Spotfire, an analysis of the data is created using standard visualizations like bar charts, scatter plots, tree maps, network graphs and so on. These are organized into one or more pages in the analysis in order to separate different aspects of the data. Finally, the application is published so that the end users can explore the analysis by interacting with it, for example by filtering out subsets of the data. Through these interactive analytic applications, the users can gain new knowledge.


Filters

Record No
1 7090

Order Priority
☑ 1-URGENT
☑ 2-HIGH
☑ 3-MEDIUM
☑ 4-NOT SPECIFIED
☑ 5-LOW

Sales Total
2,54 29354,51

Ship Method
☑ DELIVERY TRUCK
☑ EXPRESS AIR
☑ REGULAR AIR

Delivery Time
1 138

Customer Name
Type to search in list
(All) 76 values
Barbara Fisher
Dario Medina
Dave Hallsten
David Flashing
David Wiener
Deanna Eno

Region
☑ CENTRAL
☑ EAST
☑ WEST

Figure 2.3: The filters panel where the user can select which rows that should be shown in the visualizations.

The filters panel is shown in figure 2.3. In an analysis the user can filter out data using for example check box (Order Priority), range (Sales total) and list box (Customer name) filters on each column in the data table. Also a text filter is available which filters using pattern matching.

2.2 Text mining

Discovering new previously unknown facts by analyzing data is called data mining [4]. These new facts can be used as an inspiration for new theories and experiments. The difference between data mining and text mining is that in the latter, facts are extracted from natural language text. Normal data mining deals with numerical and discrete data types which are easily readable by a computer while natural language text is not. What is needed is a computer program which can read and fully understand text, which is not available in the foreseeable future [4].

A closely related area is information retrieval where the user specifies an information need defined as a search query, which the system then analyses to find a set of relevant documents. The key aspect of text mining as compared to searching, is that when searching you already know what you are looking for while in mining one explores the data to find previously unknown relationships. Yet, these areas have quite a lot in common.

To be able to perform text mining some text analysis have to be incorporated early in the process, in contrast to data mining where techniques for finding patterns can be employed directly on the data. Today there are two main approaches to text analysis, natural language processing and the statistical approach. The former is based on parsing text into parse trees using grammatical rules to extract information at sentence level. In this thesis however, we focus on the statistical approach which performs the analysis based on word frequencies and word co-occurrence.

2.3 Preprocessing

To perform analysis directly on a whole text is generally not applicable, hence the text needs to be broken down into smaller pieces, for example into sentences or words, in a preprocessing step. A preprocessing step may also contain some transformations applied to the text, depending on the problem at hand. Examples of such transformations include stop word removal and stemming. In this section we explain some of these concepts and how various techniques can be combined to preprocess text, guided by the following simple example of a text, fetched from the BBC News website[1].

[1]`http://www.bbc.co.uk/news/world-asia-pacific-13032122`, Accessed at 2011-05-23

A powerful earthquake has hit north-east Japan, exactly one month after the devastating earthquake and tsunami. The 7.1-magnitude tremor triggered a brief tsunami warning, and forced workers to evacuate the crippled Fukushima nuclear plant. The epicentre of the quake was in Fukushima prefecture, and struck at a depth of just 10km (six miles). It came as Japan said it was extending the evacuation zone around the nuclear plant because of radiation concerns.

Table 2.1: Example input text to be processed

The input data to a text mining application is a corpus (pl. corpora), which is a set of documents [5]. A document is a text, for example an email, web page, chapter in a book, answer to an open question etc. The example above could for instance be one of the documents in a huge collection of news articles to be analyzed. These documents are often stored as a sequence of bytes in a database or file system. When it comes to importing documents into an application for further processing, the documents may be stored in proprietary formats like Microsoft Word and Adobe PDF, so specialized import functions are needed for each format. Furthermore, the texts may use different encoding schemes like ASCII, UTF-8 or other nation specific standards. This information may be provided by meta data or be required by the user, but there are also heuristic methods for determining the encoding [6]. The bottom line is that the importer ensures that all the documents come in the same encoding.

2.3.1 Tokenization

The first step in analyzing the content of a document is to identify the elements it consists of. This can be done by a lexer. A lexer takes a sequence of characters as input and splits it into tokens which usually are the words, a process called tokenization. A simple approach to do this is to simply split on all non-alphanumeric characters. This can however be a problem in some cases. For example, how should the lexer treat the sequence *aren't*? With this approach the result would be two tokens, *aren* and *t*, which would be undesirable. Another problem is punctuation marks (are they the end of the sentence or just marking an abbreviation?). Even worse is Chinese where there are no white spaces.

Another way to perform the tokenization is to parse the text according to a predefined set of rules. A rule is basically a pattern to be matched given an input string. Such solutions can be realized with among other things regular expressions. There are applications for automatically generating lexers given a rule set. An example of such an application is flex[2] that generates source code for a lexer in C.

Referring to the example document given in the introduction of this chapter, using the simplest approach described above, the tokens shown in table 2.2 would be generated. Note that the word *north-east* is split into two tokens while it in this case would be correct to identify it as one token.

A, powerful, earthquake, has, hit, north, east, Japan, exactly, one, month, after, the, devastating, earthquake, and, tsunami, The, 7, 1, magnitude, tremor, triggered, a, brief, tsunami, warning, and, forced, workers, to, evacuate, the, crippled, Fukushima, nuclear, plant, The, epicentre, of, the, quake, was, in, Fukushima, prefecture, and, struck, at, a, depth, of, just, 10km, six, miles, It, came, as, Japan, said, it, was, extending, the, evacuation, zone, around, the, nuclear, plant, because, of, radiation, concerns

Table 2.2: Example document split into a comma separated list of tokens.

For more information about the difficulties in tokenization, see Manning et al. [5].

2.3.2 Bag of words

Given the documents as list of tokens, we could now start comparing them for equality by for example comparing each word position, i.e. compare word position 1 in document 1 with word position 1 in document 2 and so on for all the words. Clearly, the probability of the same word occurring at the same position in both documents is very low. Therefore the bag of words model is applied which makes the assumption that all words are conditionally independent of each other. This means that for example *John loves Mary* is considered equal to *Mary loves John*. It is a simplifying assumption which is commonly employed in information retrieval and statistical natural language processing [5].

Applying the bag of words model directly to the tokens often results in huge bags as a natural language has many words. Given a document there are only a few words in it which tell us what the document is about. This is a

[2]flex: The Fast Lexical Analyzer, `http://flex.sourceforge.net/`

problem when comparing texts, so a series of techniques have been invented to reduce the size of the vocabulary. In the next sections a short introduction to some of them are given.

2.3.3 Stop words

A stop word is a predefined word by the user that should be filtered out, i.e. removed from documents. A large proportion of the words in a document are function words such as articles and conjunctions. Examples of such words in English are *and*, *is*, *of* and so on. In the bag of words model, these words don't tell us anything about the content of a text, so they are usually added to the stop word list. Also numerics can be removed from the text in some applications.

The function words need to be manually identified for each language. Another way of deciding which stop words to use is to find them using word frequency when constructing the vocabulary list of the corpus. A word occurring in most of the documents in a corpus will not help discriminate the documents from each other, hence it may be removed from the corpus by adding it to the stop word list.

There are however cases where functional words are of importance, for example phrase search. Consider the phrase *to be or not to be*. By using functional words as stop words, this phrase would be completely removed.

Returning to the example document, the remaining words after performing stop word and numeric removal are shown in table 2.3. In this example the stop word list from the Snowball project [7] has been used.

powerful, earthquake, hit, north, east, Japan, exactly, month, devastating, earthquake, tsunami, magnitude, tremor, triggered, brief, tsunami, warning, forced, workers, evacuate, crippled, Fukushima, nuclear, plant, epicentre, quake, Fukushima, prefecture, struck, depth, miles, Japan, extending, evacuation, zone, around, nuclear, plant, radiation, concerns

Table 2.3: Example document with stop words and numerics removed.

2.3.4 Stemming

Many words in a text is not in its lemma form. For example, *squirrel* and *squirrels* refer to the same concept if you ask a human, but a computer treat them as completely different words. This means that a document containing only the form *squirrel* will not be seen as similar to a document containing *squirrels*. There are a couple of known ways to cope with this problem.

One technique is to use a dictionary which maps all known words and their inflections to their lemma form. This is called lemmatization. Although efficient, this technique requires access to such a dictionary which may not be available. It also requires that the dictionary always is up to date since new words are regularly added to languages.

Another approach is suffix stripping algorithms, which was first examined by Porter for English [8] but now exist for many languages [7]. These algorithms simply follow a small set of rules which removes the suffixes. This approach is called stemming since it leaves only the stem of the word, for example *brows* for *browse* and *browsing*. Although the results may not be real words, it maps words with standard inflections into the same stems, and thus reduces the number of word types. A potential problem with this approach is that words with different semantic meaning (which should be separate words in the analysis) can be stripped to the same stem. Below is the example document with Porters stemming algorithm applied to it.

power, earthquak, hit, north, east, Japan, exactli, month, devast, earthquak, tsunami, magnitud, tremor, trigger, brief, tsunami, warn, forc, worker, evacu, crippl, Fukushima, nuclear, plant, epicentr, quak, Fukushima, prefectur, struck, depth, mile, Japan, extend, evacu, zone, around, nuclear, plant, radiat, concern

Table 2.4: Example document with stemming applied.

2.3.5 Compound words and collocations

A weakness of the bag of words model is that it ignores word order. This can be a problem for languages which separates compound words, for example English. Consider the string *New York* which should be treated as one term instead of two, which would be the case if a simple space splitting tokenizer was used. To remedy this problem, experiments have been made where word n-grams are used as words together with single words [9]. A n-gram is a scientific term for n consecutive entities, in this case words (other text analysis techniques for example statistical language detection use character n-grams instead). By applying this technique we can for example find high frequencies of the 2-gram *New York* instead of *New* and *York*. Below is an example of the word bi-grams (2-grams) of a sentence.

This is an example.				
_ This	This is	is an	an example	example _

Table 2.5: The word bi-grams of a small sentence. Underscore denotes sentence boundaries.

In other languages like Swedish and German, compound words are not separated by spaces. In these cases it can instead be relevant to identify the words that the compound words includes. This can be achieved with the aid of a dictionary [5].

2.4 Analysis

With the preprocessing done, further analysis techniques can be applied to extract information. Typical tasks in text analysis include categorization, topic modeling, entity extraction, summarization, sentiment analysis, detecting trends and so on, however in this project only a subset of them are evaluated due to time restrictions. In this section some algebraic and statistical methods for this subset are introduced.

2.4.1 The Vector Space Model

One way of representing the bag of words model is the vector space model. In this model each document is represented as a vector of term frequencies. Terms are simply the word classes which remains after the preprocessing steps like stemming and stop word removal. Comparing two documents in this model is then a matter of using a distance measure (see section 2.4.1.5) between their term frequency vectors.

2.4.1.1 Occurrence matrix

Once the vocabulary is decided, an occurrence matrix A can be formed to represent the entire corpus. In this matrix the rows corresponds to terms and the columns to the documents. The entry A_{wd} contains the frequency of term w in document d. Column d in such a matrix is consequently the term frequency vector of document d. In table 2.6 below is an example occurrence matrix constructed from the three preprocessed documents: $\mathbf{d}_1$ = *cat feline paw cat fur*, $\mathbf{d}_2$ = *feline fur paw tail* and $\mathbf{d}_3$ = *dog tail paw smell drugs*.

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$
cat	2	0	0
feline	1	1	0
paw	1	1	1
fur	1	1	0
tail	0	1	1
dog	0	0	1
smell	0	0	1
drugs	0	0	1

Table 2.6: Example occurrence matrix

Here, we can see that the columns are the document vectors and the rows the term vectors. The matrix is usually very large and sparse, as a language has many words but a document contains only a small subset of them. Storing this entire matrix in computer memory can therefore be a problem. A sparse matrix data structure is often employed to solve this problem. There are several ways how to do this as every technique has its own benefits. When constructing the matrix, one often uses a simple structure, for example:

- A dictionary which maps row and column indices to values.
- A list of lists. Each row or column is stored in a list of non zero values along with their indices.
- Sorted coordinate list, which is a list of 3-tuples containing row index, column index and value.

All of the above have performance weaknesses when it comes to matrix operations, so the final matrix is then converted to a compressed sparse row or column format, depending on if accesses to entire rows or columns are likely [10]. In the compressed sparse column format, the matrix is stored using three arrays: A, JA and IA. A contains all non-zero entries in column major order, and JA contains their row indices. IA contains the indices in the other arrays where each column starts. (2.1) below is a small example of a matrix using the compressed sparse column format. Note that this example matrix is not sparse enough to take advantage of the format. In a corpus it is not uncommon that the matrix density is below 1%, so for these matrices large amounts of memory can be saved.

$$\begin{bmatrix} 1 & 0 & 4 \\ 0 & 0 & 5 \\ 2 & 3 & 6 \end{bmatrix} \Longleftrightarrow \begin{cases} A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \\ JA = \begin{bmatrix} 0 & 2 & 2 & 0 & 1 & 2 \end{bmatrix} \\ IA = \begin{bmatrix} 0 & 2 & 3 & 6 \end{bmatrix} \end{cases} \tag{2.1}$$

Looking up the value in cell (0,2) (assuming a zero based index) corresponds to first looking up the values at $IA[2]$ and $IA[3]$. These values indicate the index interval in JA where to look for the row number 0. The row number can be found using binary search. In this case we find it at $JA[3]$, and the value 4 is found in $A[3]$. In total, this is a $O(\log(R))$ function, where R is the number of rows.

2.4.1.2 Weighting schemes

In the original information retrieval systems, the document vectors were just the sets of terms they contained [11]. This means that either a term was associated with a document or not, and no other information was stored. A comparison between two documents were then just the number of terms they had in common, i.e. the size of the intersection of the corresponding sets. This was computed as the dot product between their vectors.

Research has shown that by assigning weights to the terms (the entries in the occurrence matrix), information retrieval performance can be improved [11]. These weights are often the product of three different measure components for the term as in equation (2.2).

$$A_{td} = tfc_{wd} \cdot cfc_w \cdot nc_d \tag{2.2}$$

The term frequency component tfc_{wd} represents the frequency of the term in the document. A document with higher frequencies of certain terms are more likely to treat the subject those terms are about. However, there are terms that occur with high frequency in all documents, for example function words or the word *computer* in a corpus about computers. These terms should have lower weight since they are not unique to any specific document or topic in the corpus. To solve this a collection (corpus) frequency component cfc_w can be used, which is a function of the number of documents a term w is associated with. Finally, the weights can be normalized using a normalization component nc_d so that longer documents get the same amount of weight as shorter documents. A list of weighting components can be found in table 2.7.

Term Frequency Component tfc_{wd}	
1.0	Binary, assigns weight 1 to terms present in the document, 0 otherwise.
n_{wd}	Term frequency, number of times the term w occurs in document d.
Corpus Frequency Component cfc_w	
1.0	No weighting. Words which occur in every document will get a high weight despite that they don't give much information.
$\log \frac{D}{d_w}$	Log inverse document frequency. D is total number of documents in the corpus, d_w is the number of documents the term w occurs in. Gives lower weight to words occurring in almost every document in the collection.
Normalization Component nc_d	
1.0	No normalization, long documents get high weights since they probably have more occurrences of the terms.
$1/n_d$	Normalize on number of terms in document n_d.

Table 2.7: Occurrence matrix weighting schemes

The combination of term frequency, log inverse document frequency and document length normalization is a common weighting function:

$$\frac{n_{wd}}{n_d} \times \log \frac{D}{d_w} \tag{2.3}$$

This combination is called tf-idf. The first approach similar to tf-idf was suggested by Salton et al. [11], with the only difference that they used Euclidean vector normalization instead of document length normalization.

2.4.1.3 Latent Semantic Analysis

The vector space model in its basic form has two problems, synonyms and homonyms. Synonyms are words that have the same meaning but are spelled differently. For example, if a user enters a query *cats* that query will not match a document which only contains the word *felines* even if they both refer to the same animal species. Homonyms are words that are spelled the same but mean different things depending on context, for example *bank* which can both refer to a financial institution and a sand mass in a river.

One way of attacking these problems is to reduce the noise in the matrix by applying dimension reduction [5]. Instead of having exact mappings between terms and documents (many dimensions), one wants to find a way to extract a lower dimensional concept space where *cat* and *feline* are not orthogonal to each other. Similarly the documents are represented in this low dimensional space. Deerwester et al. [12] proposed to perform singular value decomposition (SVD) on the occurrence matrix. SVD is an operation which factorizes a matrix A into three matrices:

$$A = U \cdot S \cdot V^T \tag{2.4}$$

If A is $W \times D$ (W is the vocabulary size, D the number of documents), then U is $W \times W$, S is $W \times D$ and V is $D \times D$. The columns of U are called the left singular vectors of A and are the eigenvectors of AA^T. Similarly, the columns of V are the right singular vectors which are the eigenvectors of A^TA. Finally, S is a diagonal matrix which contains the square roots of the eigenvalues of the corresponding vectors in U and V. These values are called the singular values. According to convention they are sorted in descending order along with their corresponding column vectors in U and V [12].

By only keeping the K largest singular values in S (and their corresponding columns in U and V), one creates a rank K approximation A':

$$A \approx A' = U_K \cdot S_K \cdot V_K^T \tag{2.5}$$

Here, U_k is $W \times K$, S_K is $K \times K$ and V_K is $D \times K$. See figure 2.4 for a visual interpretation. This way the information in A is transformed into a K-dimensional concept space. By this transformation the noise like redundant words are ignored and only the semantic meaning of the elements are kept. The actual decomposition ensures that the Frobenius norm of the difference between A and A' is minimized. The Frobenius norm is defined as the Euclidean length but for matrices. The rows of U_K are now the concept vectors of the terms and the rows of V_K are the concept vectors of the documents.

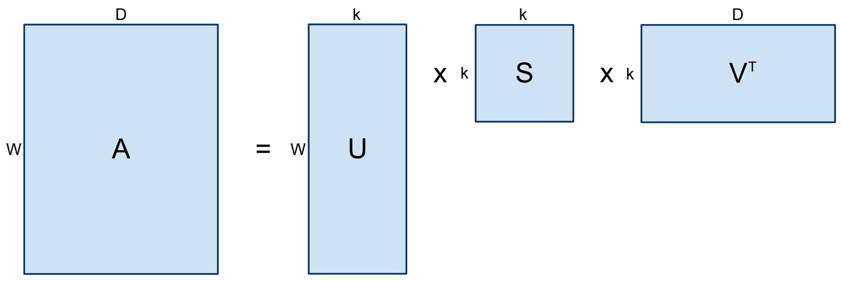


Figure 2.4: Visual interpretation of the singular value decomposition. Only the K most important singular vectors are kept, thus reducing the number of dimensions.

Choosing the number of dimensions K is a recurring issue in text modeling. Too many dimensions will result in no improvement over the default vector space model, while too few dimensions will be too coarse for information retrieval purposes. According to Bradford [13], moderate sized corpora (hundred of thousands of documents) need around 300 dimensions to give good results, while larger corpora (millions of documents) need around 400 dimensions. See chapter 4 for our own experiments.

In our previously mentioned example regarding the synonym problem with *cat* and *feline*, lets consider two documents $\mathbf{d}_1$ and $\mathbf{d}_2$. Both treat the subject of cats, however $\mathbf{d}_1$ only uses the term *cat* while $\mathbf{d}_2$ only uses the term *feline*. A user who enters the query *cat* will however probably get both documents returned if they share many other cat related terms, for example *paw*, *tail*, *fur* and so on. The singular value decomposition has detected that *cat* and *feline* are related, and thus their concept vectors are not orthogonal as their counterparts in the basic vector space model.

2.4.1.4 Random Projection

Although efficient in keeping the information of the occurrence matrix, the singular value decomposition is computationally expensive. See chapter 4 for running time experiments. It has been shown that a cheap random projection can successfully replace the SVD computation [14, 15]. In this technique, each document d is assigned an index vector i_d of length K. As LSA usually uses 300-400 dimensions, it is suggested that K should be set to a couple of thousands [15]. A small number c ($\sim 10 - 50$) of randomly chosen position pairs in the index vector are assigned 1 and -1. The full matrix A can then be approximated in a smaller matrix U by adding the generated index vector to all rows in U corresponding to the terms in the document. This is done for each document until the final A approximation U is computed. The full algorithm is shown in algorithm 1.

Algorithm 1 Random Projection pseudo code

```
$U \leftarrow newMatrix(W, K)$
**for all** $d \in D$ **do**
  $i_d \leftarrow$ new Vector($k$) with $c$ random positions set to 1 and -1
  **for all** $w \in d$ **do**
    $U(w) \leftarrow U(w) + i_d$
  **end for**
**end for**
**return** $U$
```

When the algorithm has finished, the matrix U is a word-concept matrix similar to the one produced by singular value decomposition. To produce the document-concept matrix V, each document vector in the original matrix A is multiplied with the random projection matrix U according to equation (2.6).

$$V = A^T \cdot U \tag{2.6}$$

2.4.1.5 Distance measures

Since both documents and terms are represented as vectors in the vector space model we can apply standard measures to assess the distances between them. A simple measure is the 1-norm of the difference, also known as the *Manhattan distance* (2.7). Another measure that can be used is the *Euclidean distance*, which is the 2-norm. Equations (2.7) and (2.8) below shows how to calculate these distances between two vectors $\mathbf{x}$ and $\mathbf{y}$ with K dimensions.

$$d_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{K} |\mathbf{x}_i - \mathbf{y}_i| \tag{2.7}$$

$$d_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{K} (\mathbf{x}_i - \mathbf{y}_i)^2} \tag{2.8}$$

There is however one problem by using these in our context. Consider a short document $\mathbf{d}_1$ containing the term *cat* 2 times and the term *paw* 1 time, and a long document $\mathbf{d}_2$ containing *cat* and *paw* 100 times each. The Euclidean distance between these two vectors would be very large even though the documents probably deal with the same topic. To solve this problem the *cosine similarity* (2.9) can be used instead.

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}||\mathbf{y}|} \tag{2.9}$$

The cosine similarity, which gives the cosine angle between two vectors, is defined as the dot product of the two vectors divided by the lengths of the vectors. Hence the lengths of the corresponding documents are no longer an issue. Note that the cosine gets larger for more similar documents, opposed to (2.7) and (2.8) which gets smaller.

2.4.1.6 Querying

When considering getting relevant documents given a user specified query, we would when using the basic vector space model just treat the query as a document vector (with the corresponding weighting scheme). A distance measure is then used to get a ranking of all document vectors based on how similar they are to the query vector.

If LSA has been applied to the original occur

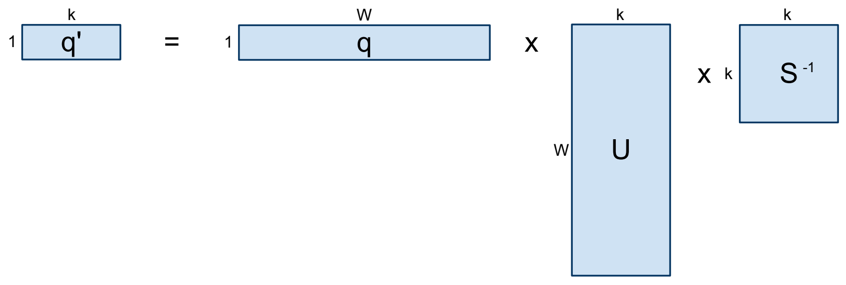


Figure 2.5: Transformation of a previously unseen query/document vector q into K-dimensional concept space.

Since the query vector now is in concept space, we can get a ranking for each document. This is done by using a distance measure on the transformed query vector and each document vector in the document-concept matrix V.

A similar procedure is done when random projection has been used. The query vector $\mathbf{q}$ is multiplied by the term-concept matrix U in order to get the low dimensional vector $\mathbf{q}_K$. This vector is then compared to each row in the document-concept matrix V, i.e. each low dimensional document vector.

2.4.1.7 Evaluation methods

To evaluate how well the models represent the data some kind of performance metric is needed. In the information retrieval domain two popular metrics for search engines are precision (2.10) and recall (2.11) [5]. These two metrics give indications of how well the search results match the information need. The information need is often defined by a search query consisting of a short phrase or some keywords. For each query, a subset of the documents in the evaluation corpus are labeled as relevant.

$$Precision = \frac{TP}{TP + FP} \tag{2.10}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.11}$$

TP indicates true positives, i.e. the number of documents correctly classified as relevant. Similarly, FP is the number of false positives (irrelevant documents classified as relevant) and FN is false negatives (relevant documents classified as irrelevant). The precision measure indicates the proportion of relevant documents in the returned set, given the query. Recall on the other hand is the proportion of how many of the relevant documents that are returned. These measures often show an antagonistic behaviour as increasing recall by considering larger search results often lowers precision. A perfect result would have both high precision (all the results were relevant) and high recall (all the relevant documents were returned).

When comparing two searching systems, their 11-point precision recall curves are often compared. These plot precision levels at 11 levels of recall (0.0, 0.1, ..., 1.0). They are often interpolated as well, which means that for each level of recall, the maximum precision for that or any higher level is plotted, see figure 2.6. If multiple queries are used, the average is computed for each recall level. If only a single performance measure is wanted, the mean average precision can be used, which is the average precision for all levels of recall.

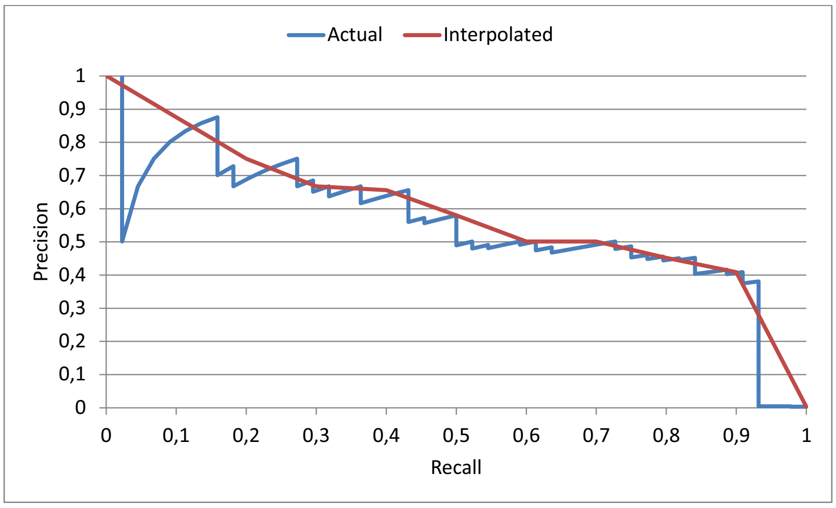


Figure 2.6: Example interpolated precision recall curve.

Precision and recall can also be combined into the F-score (see equation (2.12)) [9]. The parameter α indicates how important recall is compared to precision. For $\alpha = 0$, the F-score becomes precision, $\alpha = 1$ gives equal importance to both measures, while $\alpha > 1$ gives higher weight for recall. The F-score ranges from 0 to 1, just as precision and recall.

$$F_\alpha = \frac{(1+\alpha) \cdot Precision \cdot Recall}{\alpha \cdot Precision + Recall} \tag{2.12}$$

2.4.2 Probabilistic Topic Models

While LSA using SVD or random projection have been used successfully in many areas, for example indexing of documents [12] and images [16, 17], they are not statistically well founded for text as they are based on linear algebra. As an alternative to LSA, Hoffman introduced Probabilistic Latent Semantic Analysis (PLSA) [18] which is a probabilistic generative model. This model was later refined by Blei et al. in Latent Dirichlet Allocation (LDA) [19]. Apart from just modeling text, LDA has also been applied to for example images [20] and music [21]. In this section we give an introduction to how these models work and how they can be calculated.

2.4.2.1 Generative models

The idea behind a generative model is that it is assumed that the observed data has been generated by some generative process. For example, the data 4 heads and 6 tails could be the observed data from the generative process of flipping a coin 10 times. Using this observed data, the model parameter indicating the probability distribution of the coin (only one probability p here) is inferred by some inference method.

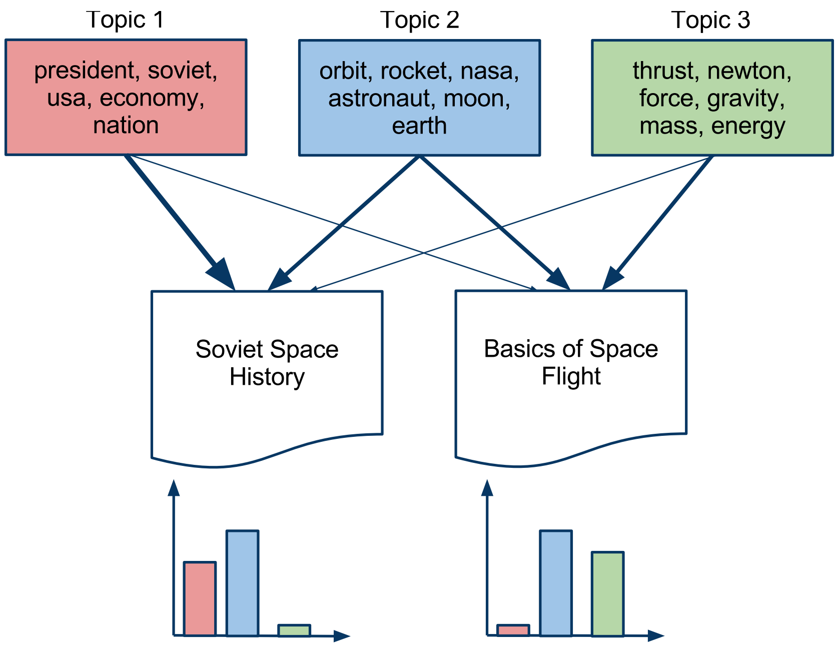

Topic 1
president, soviet, usa, economy, nation
Topic 2
orbit, rocket, nasa, astronaut, moon, earth
Topic 3
thrust, newton, force, gravity, mass, energy
Soviet Space History
Basics of Space Flight

Figure 2.7: In probabilistic topic models it is assumed that each document is a mixture of topics and that each topic is in turn a bag of words from which the words are drawn.

In probabilistic topic models, it is assumed that all the words in the corpus are observations from a generative process which uses some latent (hidden) variables. A topic z in this model is a multinomial distribution over words with probability mass function $P(w|z)$. We will call the distribution of topic z β_z. This means that given a topic, we can draw words which are common for this topic. For example, given the topic about space we may get the words *orbit*, *rocket* and *astronaut* with high probability while words like *computer* and *strawberry* belonging to other topics have low probability. This also allows the model to deal with homonyms as the same word can occur in different topics and therefore have multiple meanings.

Similar to the topics, each document is modeled a multinomial distribution over topics θ_d, with probability mass function $P(z|d)$. A document can therefore be a mixture of a couple of topics, for example the imaginary document *Soviet Space History* in figure 2.7 is a mixture of space science and politics. Similarly, the document *Basics of Space Flight* is a mixture of space science and physics, and not so much about politics. Putting it all together we get the total probability of a word w in a document d:

$$P(w, d) = \sum_{k=1}^{K} P(w|z = k)P(z = k|d) \tag{2.13}$$

Here we denote K to be the number of topics in the model, which is a user specified parameter. This is the PLSA model, often called the aspect model. If the parameters β and θ are known, we can generate a corpus by performing algorithm 2.

Algorithm 2 The generative process of the PLSA model

```
for all $d \in D$ do
  for all $i \in d$ do
    $z_{di} \sim Multinomial(\theta_d)$
    $w_{di} \sim Multinomial(\beta_{z_{di}})$
  end for
end for
```

Compare this to the process of flipping a coin, thus generating outcomes of heads and tails. The generative process of models like this is often described by a graphical model, which shows the variables and their dependencies. Figure 2.8 shows PLSA in this graphical notation.

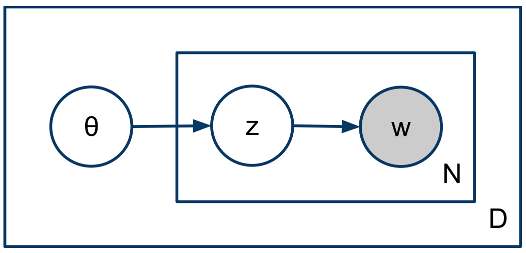


Figure 2.8: Graphical model of PLSA in plate notation. Each document has a topic distribution θ which the topic assignments are drawn from.

The circles in figure 2.8 represents random variables and the plates surrounding them shows repetition. For example, there is one distribution for each document, while there is a topic assignment z_{di} for each of the N words in the document. This way of illustrating repetition in graphical models is called plate notation. A shaded variable means that it is observed, in our case we can only observe the word assignments $\mathbf{w}$. The other variables are latent, which means that we assume that they are there and we want to infer them. For example, the topic assignment z_{di} to each word is a latent variable. Finally, the arcs between variables show their dependencies.

2.4.2.2 Latent Dirichlet Allocation

A drawback of the PLSA model is that it doesn't make any assumptions about how β and θ are generated. This makes it difficult to cope with new documents. In LDA [19] the model is extended by introducing Dirichlet priors on these distributions. This means that when generating the corpus, we also assume that each θ_d and β_k is drawn from Dirichlet distributions with parameters α and η, respectively. The generative process now follows algorithm 3 and is illustrated with the graphical model in figure 2.9.

Algorithm 3 The generative process of LDA

```
for all $k \in K$ do
  $\beta_k \sim Dirichlet(\eta)$
end for
for all $d \in D$ do
  $\theta_d \sim Dirichlet(\alpha)$
  for all $i \in d$ do
    $z_{di} \sim Multinomial(\theta_d)$
    $w_{di} \sim Multinomial(\beta_{z_{di}})$
  end for
end for
```

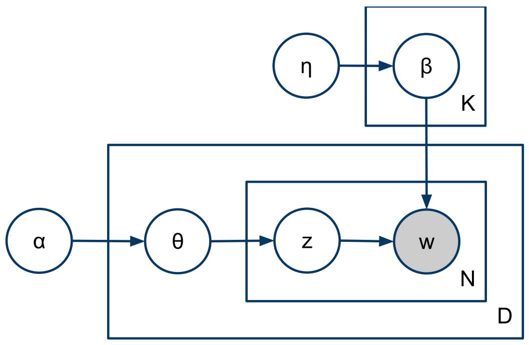


Figure 2.9: Graphical model for LDA in plate notation. The topic-word and document-topic distributions are sampled from Dirichlets parameterized by α and η.

The Dirichlet distribution is used since it is the conjugate prior to the multinomial distribution and this simplifies statistical inference of the model [19]. A sample from a K-dimensional Dirichlet is a point on the $K-1$ simplex. The simplex consists of all points whose components sum up to 1, so in 3 dimensions the 2-simplex is all points in the triangle with corners in $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$. This means that a draw from a Dirichlet can be used as the probability distribution of a multinomial.

The hyperparameters $\alpha_1, \alpha_2, ..., \alpha_K$ decide how the points on the simplex are distributed, i.e. how probable a certain document distribution is. They can be interpreted as pseudo counts, which in the case of topic distributions for documents can be seen as prior word assignments to each topic. For example, setting α_j to 5 would give higher probability of generating documents about topic j as it would be interpreted as every document has at least 5 words from that topic. For our purposes it is convenient to use a symmetric Dirichlet with a single parameter $\alpha = \alpha_1 = \alpha_2 = ... = \alpha_K$ as the topics are unknown. There have been some experiments which used asymmetric priors [22], however this is not used in this thesis. The parameter α is now a smoothing parameter. For $\alpha < 1$, the probability gets higher at the corners of the simplex which means that documents tend to focus on only a few topics, while higher α smoothes the distribution so that mixtures of all topics are more common. Setting α to 1 gives a uniform distribution over the simplex. See figure 2.10 for Dirichlet distributions with different parameters. η can be interpreted similarly as it is also a hyperparameter for a Dirichlet. For example, a high η gives a lot of smoothing, which means that words are allowed to occur in many topics.

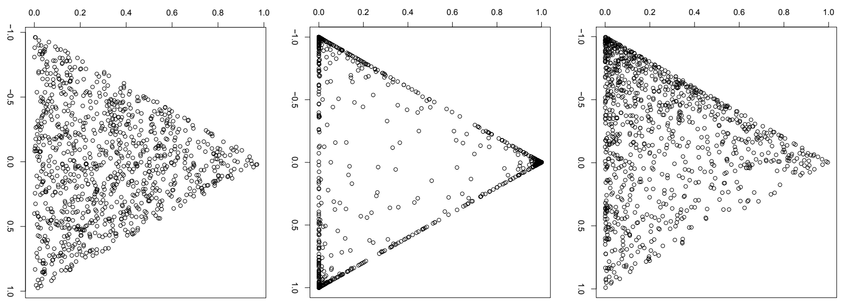


Figure 2.10: 1000 samples plotted on the 2-simplex for different Dirichlet distributions. From left: $\alpha = (1.0, 1.0, 1.0), (0.1, 0.1, 0.1), (1.0, 0.5, 0.5)$

2.4.2.3 Inference using Gibbs sampling

Given the observed data, the structure of the generative model and its hyperparameters α and η, it is possible to infer the model parameters β and θ. Figure 2.11 illustrates the problem of inference.

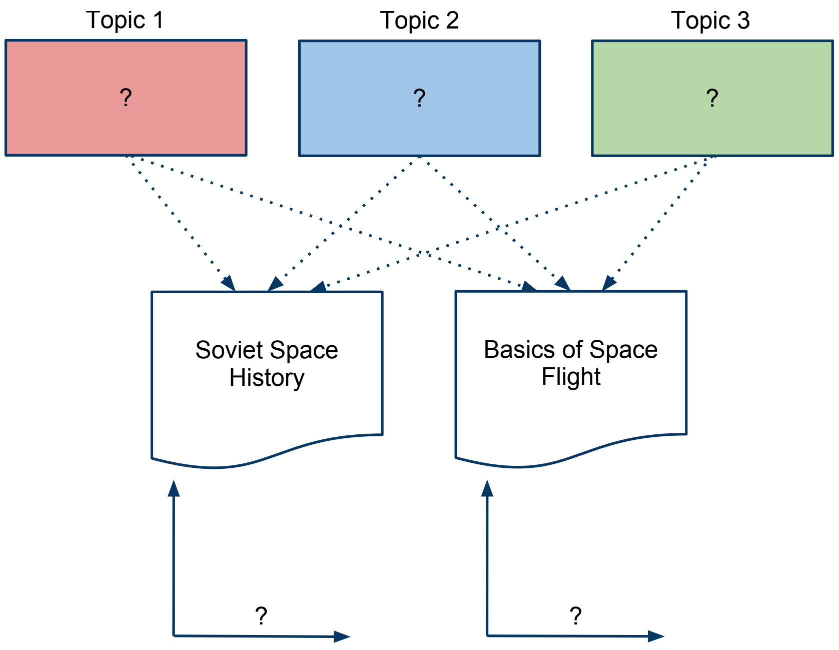


Figure 2.11: The task of the inference method is to estimate the words used in each topic and the topic associations for each document.

Exact inference of the parameters is intractable, so various approximation methods like variational Bayes [19], collapsed Gibbs sampling [23] and collapsed variational Bayes [24] can be used. In this thesis the collapsed Gibbs sampling method is used as it has a simple implementation and gives good results.

Gibbs sampling is a Markov Chain Monte Carlo (MCMC) algorithm, which means that it randomly walks through the solution space guided by the full conditional distribution (figure 2.12). This requires some further explanation. The solution space is in this case all possible distributions over words β and distributions over topics θ. These distributions can however be approximated if $\mathbf{z}$ is known, the topic assignment of each word in the corpus. Our solution space is therefore instead all possible topic assignments to all words. Here, θ and β are integrated out and this is why it is called collapsed Gibbs sampling. The MCMC algorithm then randomly transitions between states guided by a simple rule and picks samples (assignments of $\mathbf{z}$) along the way.

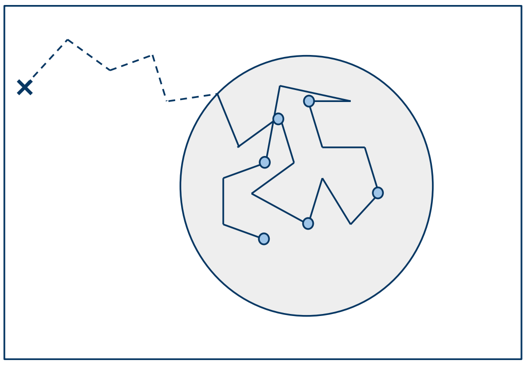


Figure 2.12: Visual interpretation of MCMC. The MCMC algorithm starts at a random point (x) in solution space. It then walks around guided by some simple rule until it converges (grey area), where samples (circles) are taken.

The simple rule is in our case defined by the conditional probability of a word being in a certain topic. Each word token in the entire corpus is considered in turn and a new topic is sampled for the word given all the other word topic assignments. This conditional distribution is written as $P(z_{di} = j|\mathbf{z}_{-di}, \mathbf{w}, \eta, \alpha)$, where $\mathbf{z}_{-di}$ denotes all topic assignments except the currently considered one. Griffiths and Steyvers [23] showed that this probability can be calculated by:

$$P(z_{di} = k|\mathbf{z}_{-di}, \eta, \alpha) \propto \frac{n_{dk} + \alpha}{n_d + K\alpha} \frac{n_{wk} + \eta}{n_k + W\eta} \tag{2.14}$$

Here, n_{wk} is the number of tokens of type w assigned to topic k, n_k the total number of tokens assigned to topic k, n_{dk} the number of tokens in document d assigned to topic k and n_d the total number of tokens in document d.

Observe that the probabilities are unnormalized so the sampling needs to be adapted for this. Note that the right part of the formula is the probability of word w under topic k, corresponding to β. Similarly the left part is the probability of a word being from topic k under document d, which is θ. From a full sample $\mathbf{z}$ we can estimate β and θ according to equations 2.15 and 2.16.

$$\beta_{kw} = \frac{n_{wk} + \eta}{n_k + W\eta} \tag{2.15}$$

$$\theta_{dk} = \frac{n_{dk} + \alpha}{n_d + K\alpha} \tag{2.16}$$

The full MCMC algorithm is visualized in figure 2.12. First, $\mathbf{z}$ is initiated to a random topic assignment, marked by $\times$ in the figure. Then the topic assignment of each word token is resampled according to the multinomial probability distribution given by equation 2.14. The count variables are updated after each token. This is performed for the entire corpus a couple of times, called the burn-in period (dashed line in the figure). The purpose of the burn-in period is to let the Markov chain stabilize to better estimations of the posterior (grey area). After the burn-in, samples of θ and β according to equations 2.15 and 2.16 are taken at regular intervals. The samples are taken with some spacing, called lag, to ensure that they are independent. When the algorithm is finished the samples are averaged to get the final posterior distributions. The full algorithm is described in pseudo code in algorithm 4.

Algorithm 4 LDA inference using collapsed Gibbs sampling

```
for $i = 0$ to Iterations do
  for all $d \in D$ do
    for all $i \in d$ do
      $z_{di} \sim Multinomial$(Probability according to equation (2.14))
      Update count variables $n$
    end for
  end for
  if $i > BurnIn$ and $i \mod lag = 0$ then
    Store $\beta$ and $\theta$ according to equations (2.15) and (2.16)
  end if
end for
return Average of stored $\beta$ and $\theta$
```

The problem is then to choose the hyperparameters α and η. Heuristic methods have shown that $\alpha = 50/K$ and $\eta = 0.01$ gives good model estimations [23]. Several estimation methods to learn them are known, however no exact closed form solution exists. The most exact method is using a iterative maximum likelihood estimation [25]. It uses the count variables available in the Gibbs sampler to update the α parameter each iteration according to equation (2.17). Ψ is the digamma function, the derivative of $\log \Gamma(x)$. The gamma function is an extension of the factorial function adapted to real numbers. The equation for estimation of η is similar to the one of α.

$$\alpha \leftarrow \frac{\alpha \cdot (\sum_{d=1}^{D} \sum_{k=1}^{K} \Psi(n_{dk} + \alpha) - DK\Psi(\alpha))}{K \cdot ((\sum_{d=1}^{D} \Psi(n_d + K\alpha)) - D\Psi(K\alpha))} \tag{2.17}$$

For a thorough explanation of probabilistic topic models and parameter estimation the reader is referred to the technical note by Gregor Heinrich [25].

2.4.2.4 Topic visualization

Extracting the most significant terms for each topic, i.e. labeling the topics, is useful for gaining insights what each topic roughly is about. As an example of use, by labeling the topics one could create a visualization that lets the user explore the whole corpus by drilling down into the individual topics. A user could then quickly locate documents of interest. See the Topic Model Visualization Engine [26] for an example of this.

Depending on the wanted result there are a couple of different ways to extract the terms. The most straight forward approach is to take the n number of terms that are most probable for each topic. These probability values are found in β_k for topic k.

2.4.2.5 Distance measures

Given the model distributions, for example the word distributions for two topics, it is possible to compute the distance between them. Measures such as those described in section 2.4.1.5 can be used if we consider the distributions as vectors in a geometric space [27], but other measures may be more appropriate for probability distributions.

When considering for example the distance between two documents d_a and d_b, we want to asses how dissimilar their corresponding distributions θ_a and θ_b are. To accomplish this, according to Steyvers and Griffiths [27], two distance measures that can be used and work well in practice are (2.18) and (2.19).

$$KL(\theta_a, \theta_b) = \frac{1}{2}(D(\theta_a, \theta_b) + D(\theta_b, \theta_a)) \tag{2.18}$$

Here $D(\theta_a, \theta_b) = \sum_{k=1}^{K} \theta_{ak} \log_2 \frac{\theta_{ak}}{\theta_{bk}}$ is the *Kullback Leibler divergence* which is 0 only when $\forall k : \theta_{ak} = \theta_{bk}$. It is an asymmetric divergence measure which means that $D(\theta_a, \theta_b) \neq D(\theta_b, \theta_a)$. Equation (2.18) above is a symmetric version of the Kullback-Leibler divergence, so that the order of the documents is ignored. Equation (2.19) below is a measure called the *symmetrized Jensen-Shannon divergence*. It is a measure of how dissimilar the distributions are to their average $(\theta_a + \theta_b)/2$.

$$JS(\theta_a, \theta_b) = \frac{1}{2}(D(\theta_a, (\theta_a + \theta_b)/2) + D(\theta_b, (\theta_a + \theta_b)/2)) \tag{2.19}$$

Another distance measure, the *Hellinger distance*, can also be used according to Blei et al. [28]. See equation (2.20) for its definition.

$$H(\theta_a, \theta_b) = \sum_{k=1}^{K} \sqrt{\theta_{ak}} - \sqrt{\theta_{bk}} \tag{2.20}$$

2.4.2.6 Querying

For performing queries and getting a ranked list of relevant documents, Steyvers and Griffiths [27] suggests to model it as a probabilistic query to the topic model. That is, for each document d calculating the conditional probability of the query q given the document, $P(q|d)$. The calculation is given in Equation (2.21) below.

$$\begin{aligned} P(q,d) &= \prod_{w \in q} P(w|d) \\ &= \prod_{w \in q} \sum_{k=1}^{K} P(w|z=k)P(z=k|d) \\ &= \prod_{w \in q} \sum_{k=1}^{K} \beta_{kw}\theta_{dk} \end{aligned} \tag{2.21}$$

This approach calculates the probability that the topic distribution of the document has generated the words associated with the query. The document which gives the maximum probability is the one which matches the query best.

Another approach is to infer the topic distribution of the query. This is done in a similar fashion to the original model inference, except that only the topic assignments for the query are resampled. Once the topic distribution of the query is inferred, it can be compared to all other documents using some similarity measure.

2.4.2.7 Evaluation methods

To compare different models, a metric indicating of how well the model fits the data is needed. Just as in the vector space model, the precision and recall measures can be used for evaluating the information retrieval performance.

A common evaluation method for probabilistic models is to measure the ability of the model to generalize to unseen data. First, a model is built on a subset of the corpus and the remaining documents are held-out. Then the total probability of the model generating the held-out data is computed. As the log probability becomes large negative numbers, one often uses the perplexity instead (see equation (2.22)) [19, 27]. It is defined as the exponent of the inverse of the average log probability of a word. The perplexity is monotonically decreasing as the model likelihood increases, so a lower perplexity indicates better generalization performance.

$$Perplexity(D_{test}) = \exp(-\frac{\sum_{d=1}^{D} \log P(\mathbf{w}_d)}{\sum_{d=1}^{D} n_d}) \tag{2.22}$$

The perplexity measure is however a purely mathematical evaluation method. The purpose of modeling topics is that the posterior distributions θ and β should be interpretable by humans. Large scale user studies show that models giving lower perplexity may give less semantically interpretable topics [29], indicating that it may not be the best evaluation method.

2.4.3 Sentiment detection

A popular application of text analysis is sentiment detection. An example is to detect positive and negative feedback about a product by analyzing online forums and blogs. This requires some kind of prior knowledge about sentiment (for example, what words are positive/negative/neutral?) to guide the algorithm. The supervised approach to this is to train a model with a corpus which is annotated with the sentiments of each document. In this thesis we focus on the unsupervised approach which uses a sentiment lexicon, i.e. a mapping from words to sentiments.

2.4.3.1 Combined sentiment topic models

There have been many approaches to sentiment detection on various levels (word/sentence/document). Commonly, these do not model topics which makes them less informative to the user as they only give an overall sentiment. A simple approach to modeling both sentiments and topics is to first find the topics using LDA and then perform sentiment detection on the topics [30]. There are however approaches which find sentiments and topics simultaneously [31, 32]. The advantage of this is that the same word can have different polarities in different domains. For example, the word *unpredictable* could have a positive meaning when talking about movies, while it can be negative in the context of car maneuvering.

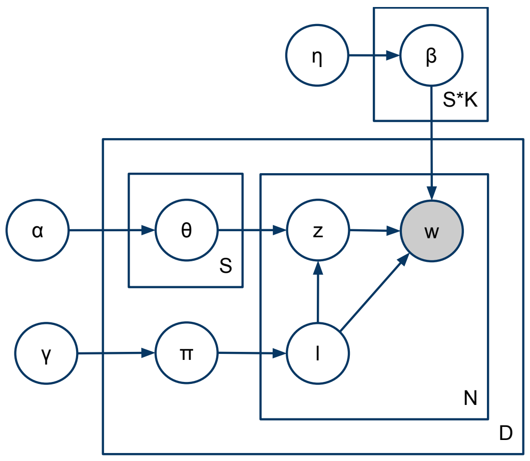


Figure 2.13: Joint Sentiment/Topic model

Lin and He [32] proposed a Joint Sentiment Topic (JST) model which assigns both a sentiment label l and a topic z to each word. As can be seen in figure 2.13, a document is modeled as a distribution over sentiments and a distribution over topics for each sentiment. This means that the generative process first draws a sentiment and then chooses a topic given that sentiment. The word is then generated from a distribution which depends on both sentiment and topic. A similar model called Sentiment-LDA (SLDA) was proposed by Li et al. [31]. It reverses the order of generation of sentiments and topics compared to JST, so a word is first assigned a topic and then a sentiment given that topic. A document is therefore a distribution over topics and a distribution over sentiment labels for each topic. See figure 2.14 for a graphical presentation of its generative model. Similar to LDA, the multinomial distributions θ, π and β are drawn from Dirichlet distributions. Experiments have shown that the JST model has better performance of the two in detecting sentiment at document level [33].

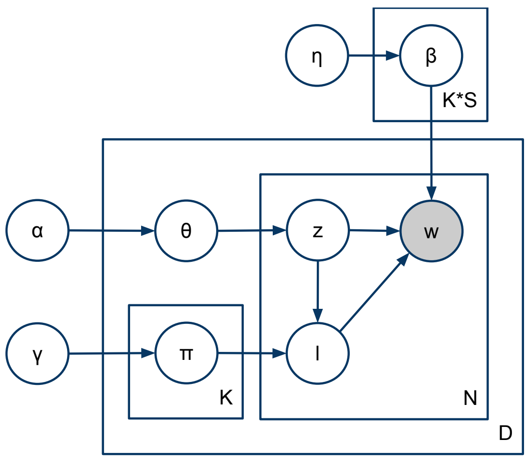


Figure 2.14: Sentiment-LDA

Both these models need a prior knowledge of known sentiment words, for example *good* is most likely a positive word. As previously mentioned, this knowledge is acquired from one or many sentiment lexicons which are available online [34, 35]. This prior knowledge is used during the inference by looking up each word in the corpus in the lexicon and assigning it the known sentiment if it is found. Other words are randomly assigned.

2.4.3.2 Inference using Gibbs sampling

Just as with LDA, the combined sentiment topic models can be inferred using Gibbs sampling. The only difference is that both the sentiment label l and the topic assignment z needs to be resampled for each word. The full conditional probability for a certain assignment is shown in equation (2.23) for JST and in equation (2.24) for SLDA.

$$P(l_{di} = s, z_{di} = k | \mathbf{l}_{-di}, \mathbf{z}_{-di}, \gamma, \alpha, \eta) \propto \frac{n_{ds} + \gamma}{n_d + S\gamma} \frac{n_{dsk} + \alpha}{n_{ds} + K\alpha} \frac{n_{wsk} + \eta}{n_{sk} + W\eta} \tag{2.23}$$

$$P(z_{di} = k, l_{di} = s | \mathbf{z}_{-di}, \mathbf{l}_{-di}, \alpha, \gamma, \eta) \propto \frac{n_{dk} + \alpha}{n_d + K\alpha} \frac{n_{dks} + \gamma}{n_{dk} + S\gamma} \frac{n_{wks} + \eta}{n_{ks} + W\eta} \tag{2.24}$$

Here the count variables n_x are defined the same way as before. For example, n_{dsk} denotes the number of words in document d assigned to sentiment s and topic k. From the full sample consisting of $\mathbf{z}$ and $\mathbf{l}$, estimation of β, θ and π can be done. The calculations are shown for JST in equation (2.25) and for SLDA in equation (2.26) below.

$$\beta_{ksw} = \frac{n_{wsk} + \eta}{n_{sk} + W\eta} \qquad \theta_{dsk} = \frac{n_{dsk} + \alpha}{n_{ds} + K\alpha} \qquad \pi_{ds} = \frac{n_{ds} + \gamma}{n_d + S\gamma} \tag{2.25}$$

$$\beta_{ksw} = \frac{n_{wks} + \eta}{n_{ks} + W\eta} \qquad \theta_{dk} = \frac{n_{dk} + \alpha}{n_d + K\alpha} \qquad \pi_{dks} = \frac{n_{dks} + \gamma}{n_{dk} + S\gamma} \tag{2.26}$$

2.4.3.3 Evaluation methods

Evaluating how well a model detects the correct sentiments is done on document level in this project. Evaluation on sentence level is not investigated but as an example it is useful for reviews where both positive and negative aspects are presented.

The requirement is that there exist labeled test data, i.e. documents that have already been classified as either positive or negative by for example a group of humans. When performing evaluation, the task is to first infer the sentiments for each document in the test data. The inferred sentiments are then compared to the ground truth labels.

In the proposed models in this section, the documents on a whole are not being directly assigned a sentiment label during inference. Instead, each word is either inferred as positive or negative (or neutral). Clearly, when performing evaluation, each of these words have to be taken into account. In [31] this is naturally achieved by calculating the probability of a sentiment s given a document d.

The calculations of these probabilities differ even though the models are very similar. Equations (2.27) and (2.28) shows how to do the calculations when using the JST and SLDA model, respectively.

$$P(s|d) = \pi_{ds} \tag{2.27}$$

$$P(s|d) = \sum_{k=1}^{K} \theta_{dk} \cdot \pi_{dks} \tag{2.28}$$

Here we can see that the probability for sentiment s in document d in JST is simply the probability found in π_{ds}. For the SLDA model we need to consider the distribution of topics θ_{dk} too, as π depends on it here.

2.5 Visualization

Given different distance measures it is possible to compute which documents and terms (and topics) that are similar to each other in the semantic space. In a text mining application these similarities can then be used when trying to discover previously unknown relationships. It is common to use some sort of visualization to aid with the text mining. The main problem with visualization in this context is that it is not possible to plot the vectors directly since they have too many dimensions, hence they must first be converted to a low dimensional space while still maintaining their characteristics.

One technique that can be used for getting a low dimensional view of high dimensional data is to employ a self-organizing map (SOM) [36], which is an artificial neural network used for unsupervised learning. A SOM consists of a number of neurons, each with an associated K-dimensional weight vector. The neurons are laid out in low dimensional space and together forms a map. For example, if a 2-dimensional space is used then each neuron gets an x and y-component specifying its position in the map. Figure 2.15 illustrates an example SOM.

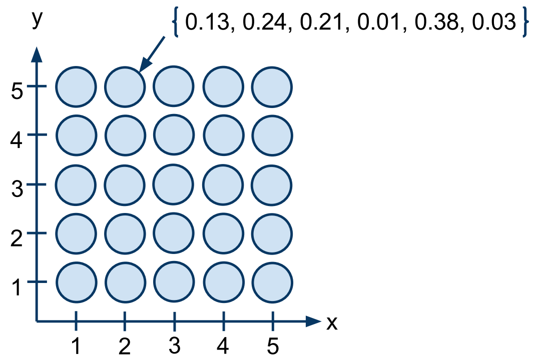


Figure 2.15: An example 2-dimensional SOM with 25 neurons. An example 6-dimensional weight vector is shown for the neuron at position (2,5).

Common for artificial neural networks is that they are trained with training data so that they later can be used in a classification process. In our case the training data are the document or term vectors (or topic vectors). When learning the network, the weight vectors are updated so that they characterize the training data vectors given some sort of distance measure. This usually implies that the weight vectors have to be of the same dimensionality as the vectors in the training data. The learning is done in a number of iterations as shown in algorithm 5 below.

Algorithm 5 Self-organizing map training

```
for $i = 0$ to Iterations do
    $t \leftarrow t + 1$
    $\mathbf{s} \leftarrow$ random sample from data
    $\mathbf{y} \leftarrow \arg\min_{\mathbf{n}}(sim(\mathbf{s}, \mathbf{n}))$
    for all $\mathbf{n} \in$ map do
        $\mathbf{n} \leftarrow \mathbf{n} + h(t, pos(\mathbf{n}), pos(\mathbf{y}))(\mathbf{s} - \mathbf{n})$
    end for
end for
```

For each iteration a random training sample **s** is picked from the training data. Then the neuron **y** with the most similar weight vector is selected according to the distance measure. Then each neuron **n** in the map is updated to be more similar to **s** according to an update function which depends on iteration number t and the map positions of the neurons. The update function is usually taken to be the Gaussian described in equation (2.29).

$$h(t, \mathbf{n}_{pos}, \mathbf{y}_{pos}) = \alpha(t) \exp(-\frac{|\mathbf{y}_{pos} - \mathbf{n}_{pos}|}{2\sigma(t)^2}) \tag{2.29}$$

Here, $\alpha(t)$ is called the learning rate and $\sigma(t)$ is the neighborhood function. They are both monotonically decreasing functions in the number of iterations t performed. Note also that $\mathbf{n}_{pos}$ and $\mathbf{y}_{pos}$ are the 2d positions of the neurons in the grid, and their computed distance is the Euclidean distance.

When the SOM has been trained it is used to place the documents or terms in a visualization. This is achieved by for each such sample (i.e. document or term), finding its most similar neuron in the SOM (like **y** in the training), and assigning that neurons position to the sample. All high-dimensional vectors will now have a low-dimensional representation.

One important issue is deciding the number of neurons to use in the SOM. To reduce the probability that two documents are mapped to the same neuron, a number larger than the number of documents should be used.

Chapter 3

Implementation

In this chapter, we describe the implemented text analysis pipeline. Note that what is presented here is not a final product but an example of how a text analysis toolkit can be implemented. As the Spotfire Platform API is implemented in Microsoft C# .NET, this language was used.

First an overview of the text analysis pipeline is presented, describing the recommended work flow. Then an explanation of each module in the pipeline is given, along with its implementation issues and considerations.

3.1 Overview

The text analysis is performed by a pipeline of functions which transform their input and passes it on to the next function. Which functions to use and how they are configured is decided by the user, see appendix A for the user interface in Spotfire.

The input is a collection of character strings, where each string is a document. The task of the preprocessing step is to split the strings into tokens using a tokenizer, followed by optional word filters and stemming. The resulting bag of words corpus has a fixed vocabulary, so the documents are represented by series of term identifiers instead of a list of tokens.

All models take a corpus as input, which is used for building the model. The vector space models output concept vectors and the probabilistic topic models output probability distributions, however these are all treated the same. The output can then be used for comparing documents using distance measures or for direct inspection using a visualization in Spotfire. An optional step is to project the model output into two dimensional space using a self-organizing map. A complete data flow diagram can be seen in figure 3.1.

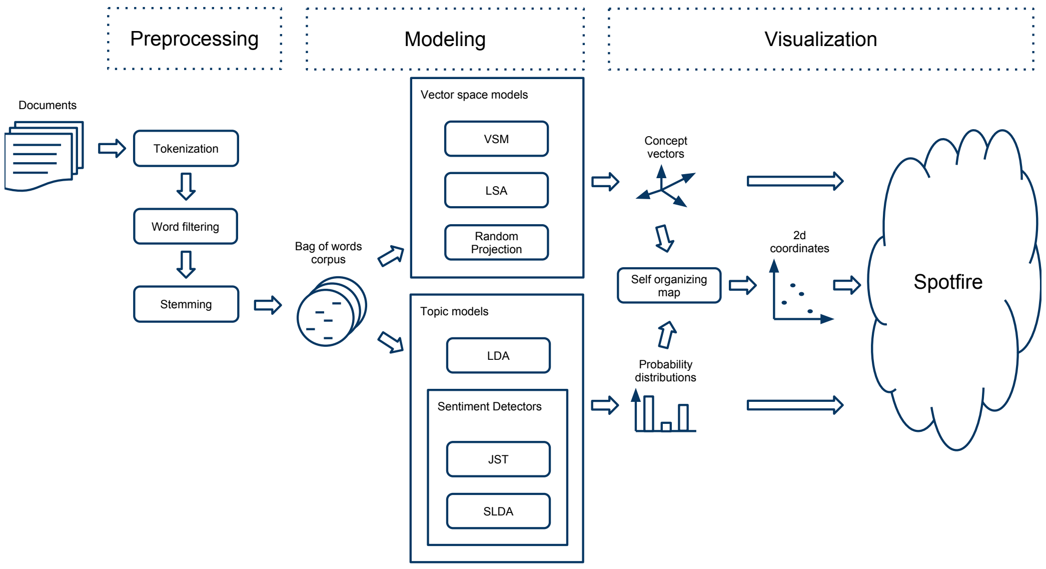


Figure 3.1: Data flow diagram over the text analysis framework.

An example use of the pipeline is shown in listing 3.1. The `Corpus` class handles all the preprocessing. It also takes a `ILanguageDetector` as some text filters need to know the language, however this component is not explored in detail in this thesis. It uses simple character n-gram statistics from texts in different languages for classification.

```
// Documents that are to be included in the corpus
List<String> docs = new List<String>();
docs.Add("This is an example document");
docs.Add("Another example document");

// Match [A-Za-z0-9_] as words using a regex
ITokenizer tok = new RegexTokenizer(@"\w+");

// Train a language detector using bi-grams
NGramLanguageDetector ld = new NGramLanguageDetector(2);
ld.AddLanguage(Language.en, "here is a long text");
ld.AddLanguage(Language.de, "hier ist ein langer text");

// Stop word filter by reading from comma-separated files
StopWordFilter swf = new StopWordFilter();
swf.AddStopWordsFromFile(Language.en, "enStopWords.txt");
swf.AddStopWordsFromFile(Language.de, "deStopWords.txt");

// The word filters to use
List<IWordFilter> wf = new List<IWordFilter>();
wf.Add(swf);
wf.Add(new MinWordLength(2));    // Words with length < 2
wf.Add(new MostCommonTerms(20));// 20 most common terms
wf.Add(new SnowballStemmer());  // Do stemming

// Create the corpus
Corpus c = new Corpus("MyCorpus", docs, tok, ld, wf);

// Create a LDAModel with 5 topics, alpha = 0.9, eta = 0.01
IModel lda = new LDAModel("MyLDAModel", c, 5, 0.9, 0.01);
while (lda.Iterate())
{
  // Report progress
  double p = (double)lda.CurrentIteration / lda.Iterations;
}
```

Listing 3.1: An example use of the text analysis toolkit.

In the following sections it is described how each step is implemented along with some considerations.

3.2 Preprocessing

The preprocessing step in this project is specific to text as this was the focus of this work. Other preprocessing pipelines could be implemented to generate bag of words representations of for example image or sound data. The implemented dimension reduction models could then be applied with little to no modification. In this section it is described how the text preprocessing pipeline can be implemented.

3.2.1 Corpus

The class `Corpus` represents a collection of documents, which is what all of the implemented models have as base input. Required input to a corpus, apart from a set of documents, is a tokenizer and a list of word filters. One problem with our implementation is that there is no support for incremental adding of documents to a corpus: when all documents have been added, the tokenizer and word filters are applied and the vocabulary is fixed. This is due to problems in the implemented models, see chapter 5.

The current implementation stores the corpus as two parts, the documents and their vocabulary. A vocabulary is a mapping from the original word string to an integer term identifier and the other way around. Using the vocabulary, the documents are simply lists of word indices stored as integers. This means that the memory use of a corpus is linear in the number of tokens and word types, i.e. $O(DN + W)$.

3.2.2 Tokenization

While natural language word tokenization is a scientific field on its own, the choice was made to not focus on this area. Tokenizers implement the interface `ITokenizer`. Only one tokenizer, `RegexTokenizer`, was implemented using the .NET regular expression class `Regex`. This allows the user to enter a regular expression for what is supposed to be parsed as words. This simple implementation was chosen to give more focus to the analysis techniques later in the pipeline, but still give the user the ability to specify his/her own parsing pattern.

The time complexity of this operation depends on the expression but often a simple one will do (only matching groups of alphanumerical characters), which gives a running time linear in the input data [37]. No part of speech tagging or other natural language processing is supported by the output, so the tokenizer only takes a string as input and returns a list of strings corresponding to the words found.

3.2.3 Word filtering and stemming

When the corpus has been tokenized, a series of word filters are applied to reduce the number of words. A word filter is like a transformation, so it can not only remove words but it can also generate new words or transform the words (for example stemming). Common for all word filters is that they take a list of strings as input and return a list of strings. Word filters implement the interface `IWordFilter` and the following word filters have been implemented:

`MinWordLength` removes words shorter than a specified number of characters, for example 2 or 3.

`MinWordFrequency` removes words which occur less than or equal a specified frequency in the entire corpus.

`MostCommonTerms` removes the n most frequent words.

`NGramGenerator` generates word n-grams from the token sequence in each document.

`NumericRemover` removes all numeric tokens, for example 123 and 3.14.

`StopWordFilter` filters out all words according to a user specified list. In our experiments the lists from the Snowball project [7] were used.

`TfIdfFilter` is a more advanced version of `MostCommonTerms`. It uses the term specific tf-idf formula (3.1) to decide which words to keep in the corpus.

$$n_w \times \log \frac{D}{d_w} \tag{3.1}$$

Here, n_w is the corpus wide frequency of word w, D is the total number of documents and d_w is the number of documents word w occurs in.

`SnowballStemmer` This filter does not explicitly remove words, but instead clip the word tokens to their stems using stemmers from the Snowball project [7]. This package was chosen as it is open source (BSD license) and a C# implementation was available[1]. The languages currently supported are Danish, Dutch, English, Finnish, French, German, Italian, Norwegian, Portuguese, Russian, Spanish and Swedish.

As can be seen, the `MinWordFrequency`, `MostCommonTerms` and `TfIdfFilter` filters require the entire corpus to be read in before they can be applied as they need corpus wide word statistics (n_w, d_w). This can be a problem in applications where new documents are presented continuously, and is one reason to why the corpus was decided to be fixed. Adding new documents to an already filtered corpus could potentially filter out words which should not be filtered out, as the new documents could change their statistics.

[1]`http://snowball.tartarus.org/archives/snowball-discuss/0943.html`

3.3 Analysis

In this section the implemented techniques for analyzing preprocessed text are outlined briefly. All the models described implement the `IModel` interface. What defines a model is that it has a reference to a `Corpus`, that it is possible to perform queries to find documents, that document and term similarities can be computed, and also that individual document and term vectors can be accessed. Example code for how a built model can be used follows in listing 3.2.

```
// Use the cosine distance measure
IDistanceMeasure dm = new Cosine();

// A query returns the similarity for each document
List<double> distQuery = m.PerformQuery("a query", dm);

// Get distance for the first document in the corpus
double distDoc1 = distQuery[0];

// Vector for the first document in the corpus
Vector vecDoc1 = model.GetDocumentVector(0);

// Gets distances to all documents from first document
List<double> distsDoc1 = model.GetDocumentDistances(0, dm);
```

Listing 3.2: An example use of a built model.

3.3.1 Math classes

As many models use sparse vectors and matrices, classes for these structures had to be implemented. The implemented classes can be seen in figure 3.2. The `SparseVector` is simply a sorted list of (index,value) pairs, which reduces the running time of a comparison between two vectors to linear in number of non zero values. The dense format uses a full array, which gives fast lookup but expensive comparisons.

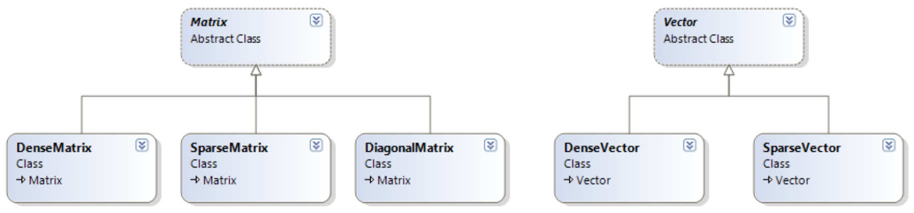


Figure 3.2: UML diagram for the vectors and matrices.

The matrix classes are correspondingly implemented. The `SparseMatrix` class uses the compressed sparse column format which was described in section 2.4.1.1. It provides low memory use at the cost of $O(\log N)$ lookup time for single values, however accessing whole columns is cheap (which is the case when accessing whole documents as in search). The diagonal matrix saves memory by only storing the values on the diagonal in an array.

3.3.2 Vector space models

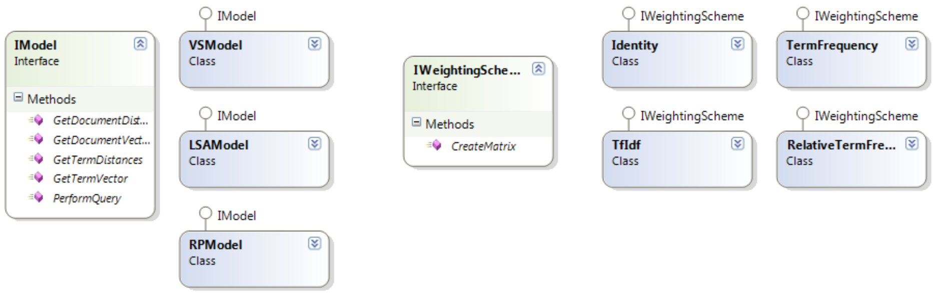


Figure 3.3: The vector space models and weighting schemes

The vector space models only implement the `IModel` interface as shown in figure 3.3. Below is a short description for each of those models.

`VSModel` is the basic vector space model as described in section 2.4.1. It utilizes a compressed sparse column matrix to store the occurrence matrix, thus it minimizes memory use to $O(DN)$, the number of non zero values in the occurrence matrix. When querying the model, the sparse vector format is used thus improving running time.

`LSAModel` implements Latent Semantic Analysis as described in section 2.4.1.3. The number of dimensions K is given as an input parameter. This model uses the *las2* algorithm in SVDLIBC to perform the singular value decomposition, as it seems to be the fastest and most widely used implementation [38, 39].

In the implementation the matrices U_K, V_K and $U_K \cdot S_K^{-1}$ are stored for computational efficiency. If memory usage is to be as low as possible, it is enough to store U_K and S_K while still being able to perform both queries and assessing document similarities. The documents can always be transformed into the semantic space by interpreting them as queries and using the query procedure as previously illustrated in Figure 2.5. The U_K and V_K matrices are dense, however S_K is stored as a diagonal matrix. Summing up, the memory use of the LSA model is $O(K(W + D))$. As the document vectors are not sparse, a full query to the model is $O(DK)$ in running time.

`RPModel` implements a simple random projection model according to section 2.4.1.4. The model currently uses a dense format for storing the U and V matrices, resulting in a memory use of $O(K(W + D))$ just as in the `LSAModel`. Similarly the querying is $O(DK)$ in running time since a distance measure is applied between each document concept vector and the query.

3.3.3 Weighting schemes

The `VSModel` uses an occurrence matrix to represent the corpus, and the `LSAModel` also uses such a matrix before the singular value decomposition. These matrices are calculated by a weighting scheme which implements the `IWeightingScheme` interface. A scheme takes a `Corpus` and creates a sparse matrix according to some weighting function. In the theory, these are described as a combination of three different components; term frequency, corpus frequency and normalization. As tf-idf is the most popular combination, it was chosen to be implemented directly together with some other simpler schemes for comparison. The interface also defines a function for properly weighting new documents, for example queries, so that they match the weighting of the occurrence matrix. The implemented schemes are described below.

`Identity` is the boolean approach which ignores the term frequencies, setting A_{wd} to 1 if w occurs in d, otherwise 0.

`TermFrequency` simply counts the term frequencies n_{wd} and store them as they are.

`RelativeTermFrequency` normalizes the term frequencies by document lengths, thus giving long and short documents equal importance.

`TfIdf` calculates the full tf-idf formula according to equation (2.3) in theory section 2.4.1.2.

3.3.4 Probabilistic topic models

The interface `ITopicModel` extends `IModel` and is used for all topic models. A topic model adds specific topic extensions such as extracting the most probable terms (thus getting a topic label) and getting topic distributions. See figure 3.4.

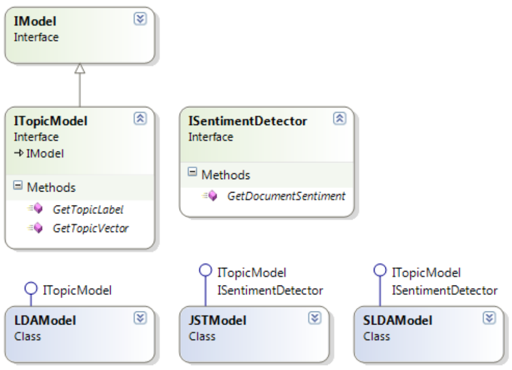


Figure 3.4: The probabilistic topic models

Apart from the implemented topic models which also models sentiment (described in the next section), one pure topic model has been implemented, namely `LDAModel`. This model implements Latent Dirichlet Allocation using collapsed Gibbs sampling as described in section 2.4.2.3. Using the approach described in pseudo code in algorithm 4 in that section, the running time complexity is $O(DNK)$ for each iteration. There are some optimizations available discussed in chapter 5.

Memory-wise, the algorithm needs the count arrays n_{wk}, n_{dk} and n_k (n_d is simply the document length, retrieved from the corpus) and the θ and β arrays. Just as the indices of the count arrays implies, their dimensions are $W \times K$ and $D \times K$, resulting in a memory use of $O(K(D+W))$. The θ and β arrays are used to store the posterior distributions are of equivalent size. Finally, all the topic assignments $\mathbf{z}$ of the words in the corpus needs to be stored as well, giving a final memory use of $O(K(D+W)+DN)$.

Specifying the number of burn-in iterations can be difficult since it varies with corpus and model size. The implemented sampler therefore checks if the model perplexity has converged after each sampling iteration according to a relative change threshold. Once it has converged, it starts taking samples with a specified interval and computes the probability distributions by averaging them.

Querying the model is implemented using equation (2.21) in the theory as proposed by Steyvers and Griffiths. As the probabilities become very small due to the multiplication, the log probability is used in order to avoid floating point precision errors.

3.3.5 Sentiment detection

The interface `ISentimentDetector` is implemented by all sentiment detectors. The currently implemented detectors all need a sentiment lexicon as input. It is simply an array which indicates prior known sentiments for each word. Words with unknown sentiment are set to -1.

Note that available lexicons offer positive, negative and neutral sentiments, hence mappings to these are given as input. However, the implementation supports any number and type of word labels. The lexicon could for example be eco related words, thus detecting the eco friendliness of a document.

`JSTModel` and `SLDAModel` are the Joint Sentiment Topic model and Sentiment-LDA model, as described in section 2.4.3.1. The sentiment lexicon given is used in both the initialization step and the iteration steps during Gibbs sampling, thus giving the words found in the lexicon a fixed label. Just as with LDA, the sampler checks for convergence using the perplexity measure.

As the sentiment detectors also models sentiment, their memory use is slightly higher than LDA. In the JST case, the count arrays are n_{ds}, n_{dsk}, n_{wsk} and n_{sk}. This results in a memory use of $O(SK(D+W))$, which shows that the sentiments adds another factor S compared to LDA. The probability distribution samples θ, β, and π have a corresponding memory use. Finally, besides the topic assignment $\mathbf{z}$, also a sentiment assignment $\mathbf{l}$ is stored.

3.3.6 Distance measures

The `IDistanceMeasure` is an interface for all functions used to compare two numeric vectors, returning a distance. The vectors can either be concept vectors or probability distributions depending on which model was used to generate them. As all functions should be directly comparable it was defined that a lower result means better match. Below is a short description on how they were implemented.

`Manhattan` *Manhattan distance*, defined in equation (2.7).

`Euclidean` *Euclidean distance*, defined in equation (2.8).

`Cosine` *Cosine measure*, defined in equation (2.9). Since the returned value needs to be lower when the vectors match, $1 - \cos(\mathbf{x}, \mathbf{y})$ is used instead.

`SymKullbackLeibler` The *symmetric Kullback-Leibler divergence* according to equation (2.18).

`JensenShannon` The *Jensen-Shannon divergence* according to equation (2.19).

`Hellinger` The *Hellinger distance* according to equation (2.20).

3.4 Visualization

The self-organizing map is implemented according to algorithm 5. The similarity function can be chosen from the standard measures depending on which type of input is chosen. For the update function, the learning rate is defined as the exponential function in equation (3.2). The neighborhood function σ is also an exponentially decreasing function, defined in (3.3).

$$\alpha(t) = \alpha_0 \exp(-\frac{t}{t_{max}}) \tag{3.2}$$

$$\sigma(t) = r_0 \exp(-\frac{t \log(r_0)}{t_{max}}) \tag{3.3}$$

Here, α_0 is the initial learning rate, set to a value between 0 and 1. r_0 is the initial influence radius, which should be set proportional to the map radius. The $\log(r_0)$ part in (3.3) is multiplied in so that the radius goes down to 1 in the last iteration.

The time complexity of finding the neuron with the most similar weight vector is linear in the number of neurons and number of dimensions. This can be quite time consuming when finding the position for each document or word, so the implementation also provides a sampling approximation for this. It's pseudo code is described in algorithm 6.

Algorithm 6 Self-organizing map sampling estimation

```
$\mathbf{n}_{best} \leftarrow \emptyset$
for $i = 1$ to Samples do
    $\mathbf{n} \leftarrow$ random neuron in map
    while $\exists \mathbf{n}_{new} \in Neighbors(\mathbf{n}); sim(\mathbf{n}_{new}, \mathbf{s}) < sim(\mathbf{n}, \mathbf{s})$ do
        $\mathbf{n} \leftarrow \mathbf{n}_{new}$
    end while
    if $sim(\mathbf{n}, \mathbf{s}) < sim(\mathbf{n}_{best}, \mathbf{s})$ then
        $\mathbf{n}_{best} \leftarrow \mathbf{n}$
    end if
end for
```

It picks a random neuron **n** in the map, and then checks if any of the neighbors are more similar to the target document/word **s**. If so, it takes the neighbor instead and repeats the process until no better neighbor has been found. In order to reduce the risk of ending up in local mimima, a series of such samples are taken and the best is returned. This is called a *random-restart hill climbing* algorithm.

Chapter 4

Experiments

In order to evaluate the techniques and find appropriate parameters, a series of experiments have been implemented. These experiments can also be used to evaluate future improvements to the implementations. This chapter gives the details and results of these experiments and the data sets that have been used. All experiments were run on a standard desktop computer with Intel Core 2 Duo 6400 processor and 4 GB RAM.

4.1 Preprocessing

For the experiments, four different corpora were used:

Medline The OHSUMED test collection contains 348566 abstracts from MEDLINE, an online medical information database. It was originally used by William Hersh [40] for information retrieval purposes. In our experiments we have used a subset of 20000 documents, along with 63 topics and their relevance judgments used in the TREC-9 filtering track [41].

20NewsGroups A collection of 20000 newsgroup documents, evenly partitioned across 20 newsgroups [42]. From this corpus 101 documents from each group was selected, resulting in a collection of 2020 documents with 730338 word tokens. This corpus is well suited for clustering and topic extraction experiments as it is naturally partitioned into groups. Some groups are more similar to each other, like com.sys.ibm.pc.hardware and com.sys.mac.hardware, while other are very unrelated like sci.electronics and soc.religion.christian.

PolarityDataset Version 2.0 of a movie review dataset used by Pang and Lee [43]. Contains 1000 positive and 1000 negative movie reviews, containing a total of 1336782 word tokens.

MultiDomain A collection of 8000 reviews from four different domains: books, dvds, kitchen and electronics. It was first used by Blitzer et al. [44].

In the experiments, different versions of the corpora have been used. For example, when generating topic labels in LDA it is preferred to have the words in their original form as they are more readable. Other applications like searching could however benefit from stemming. In table 4.1 below the corpus sizes given different word filtering schemes are given. For all experiments the `RegexTokenizer` was set to match words as $\backslash w+$. This regex matches one or more letters, digits and underscores as a word token.

	Medline		20NewsGroups		PolarityDataset		MultiDomain	
Filters	Terms	Tokens	Terms	Tokens	Terms	Tokens	Terms	Tokens
1. None	47110	2641399	42559	730338	39696	1336782	39179	1110961
2. L	47074	2505138	42522	675052	39659	1259869	39142	1033471
3. LS	46956	1648486	42400	421584	39537	718824	39020	565248
4. LSN	44880	1571566	38732	401851	39202	714912	38368	559646
5. LSNF	44850	1429899	38702	364210	39172	640000	38338	495555
6. LSNP	32349	1571566	29448	401851	25250	714912	25662	559646
7. LSNPF	32319	1381738	29418	359514	25220	625655	25632	484228
8. LSNPT	25000	1564217	25000	388039	25000	623715	25000	558984

Table 4.1: Corpus sizes with different word filters applied. L = MinWordLength(2), S = StopWordFilter, N = NumericRemover, F = MostCommonTerms(30), P = SnowballStemmer, T = TfIdfFilter(25000)

4.2 Topic modeling

In the topic modeling experiments, the 20NewsGroups corpus was modeled using LDA. The stop word list for English was used, together with a minimum word length filter set to 2 and a numeric filter. Also the 30 most common terms were removed. No stemming was performed in order to keep the words readable when extracting the most probable words for the topics. This corresponds to row 5 in table 4.1. As recommended by Steyvers and Griffiths [23], α was set to $50/K$ and η to 0.01 in all experiments.

4.2.1 Checking for convergence

In order to decide when to stop the burn-in and start taking samples, the model perplexity is used. After each iteration the current perplexity is compared to the previous, and once the relative change is below a certain threshold the model is assumed to have converged.

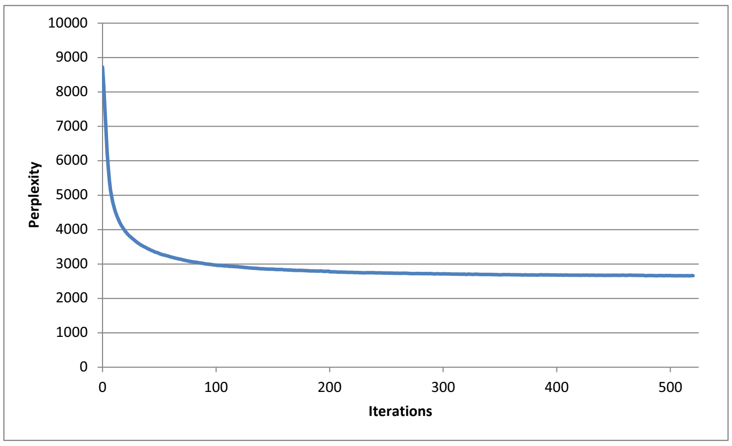


Figure 4.1: Perplexity for a LDA model with 50 topics, as a function of number of iterations.

In figure 4.1, the perplexity is plotted as a function of iterations performed. It starts to stabilize after about 200 iterations, which means that it can start to take samples. The convergence threshold was set to 0.1% for the following experiments, and this results in about 200 iterations. For each number of topics, the average perplexity of 5 models has been taken. Figure 4.2 plots the average number of iterations needed to converge as a function of number of topics used. As can be seen, it is relatively constant.

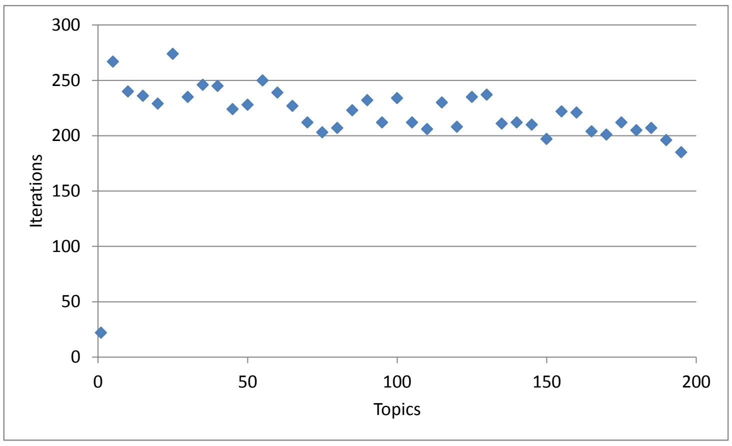


Figure 4.2: Average number of iterations needed for the perplexity to converge.

While the number of iterations needed is almost constant in topics, the whole estimation process requires linearly more time. See figure 4.3. This is no surprise as one Gibbs sampling iteration is $O(DNK)$, i.e. linear in number of topics. Still, a model with 200 topics completes in about 5 minutes for a corpus with 364210 word tokens.

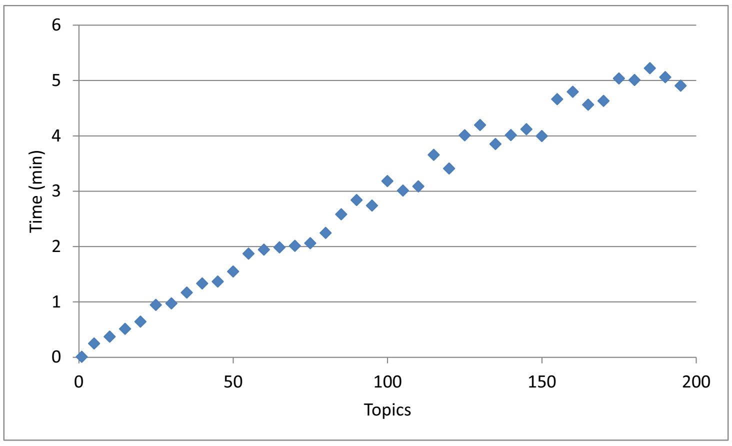


Figure 4.3: Time needed for the perplexity to converge.

4.2.2 Choosing number of topics

The number of topics used also affects the generalization performance, i.e. the perplexity. Figure 4.4 shows that the perplexity for this corpus decreases as the number of topics increase.

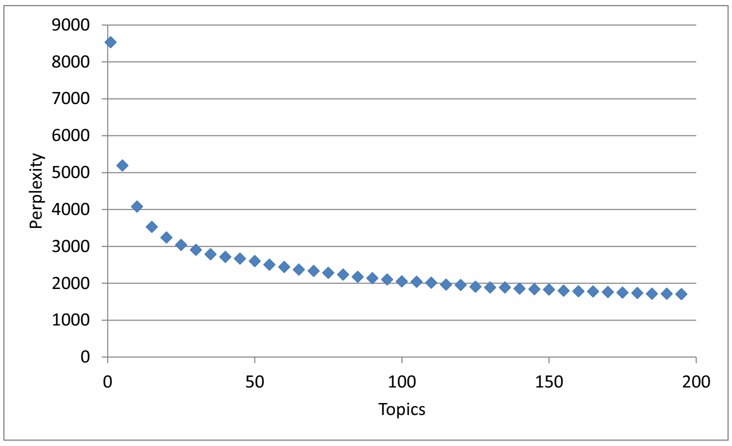


Figure 4.4: Model perplexity as a function of number of topics.

However, choosing too many results in uninterpretable topics due to overfitting while too few generate too broad topics. Instead a held-out perplexity measure can be used. In the next experiment a test set consisting of 20% of the corpus was held-out when building the model. The first half of these held-out documents were inferred by the model, and the rest was used for the perplexity calculation. The perplexity of this test set now settles at around 100 topics.

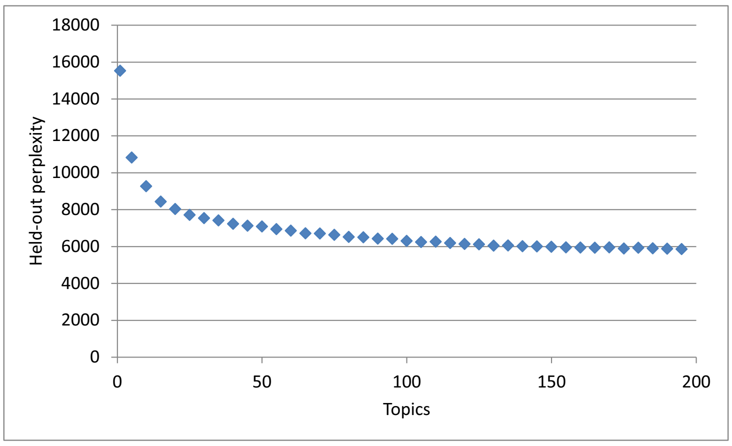


Figure 4.5: Perplexity of a 20% held-out test set as a function of number of topics.

4.2.3 Examples

By looking at the top 10 most probable words in each topic for the 50 topics model, one can get an idea of what they are about. In table 4.2 below three topics from the 20NewsGroups corpus are shown. Here, one can easily understand that topic 8 is about Christianity, probably identified from the posts in the soc.religion.christian, talk.religion.misc and alt.atheism news groups.

Topic 8	Topic 32	Topic 43
god	space	year
jesus	nasa	game
christian	earth	games
bible	shuttle	baseball
christ	orbit	team
church	mission	runs
christians	mars	players
religion	lunar	win
atheism	center	season
faith	data	player

Table 4.2: The most probable words for three topics in the 20NewsGroups corpus.

With the estimated distributions we can perform filtering on topics in Spotfire. This is achieved by importing the distributions as new columns in the data table containing the original documents. See appendix A for details on the Spotfire integration. As an example, consider the document in figure 4.3 below, found by filtering to documents mainly consisting of words assigned to topic 8 and 32. In this figure the words assigned to these two topics are colored, while non-colored words (except for those removed during preprocessing) belong to other topics. The full topic distribution for the document is shown in figure 4.6, visualized using the bar chart in Spotfire.

From: caldwell@facman.ohsu.edu (Larry Caldwell)
Subject: Re: SUNDAY! THE DAY OF OUR LORD!

pharvey@quack.kfu.com (Paul Harvey) writes:
>dlecoint@garnet.acns.fsu.edu (Darius_Lecointe) writes:
>>Exactly. Sunday worship is in honor or the *SUN*, not the *SON* of God.
>Same thing, isn't it? It's pronounced the same? What other heavenly
>beings are resurrected? The moon? That would by lunacy, at least to a
>sunday worshiper.
I have heard that the sabbath was originally determined by the phases of the moon, and had elements of moon worship. Early stuff, Egyptian in nature.

Table 4.3: An example document from the 20NewsGroups corpus colored by topic assignments. The following Color-Topic combinations are used: Blue-Topic 8, DarkRed-Topic 32.

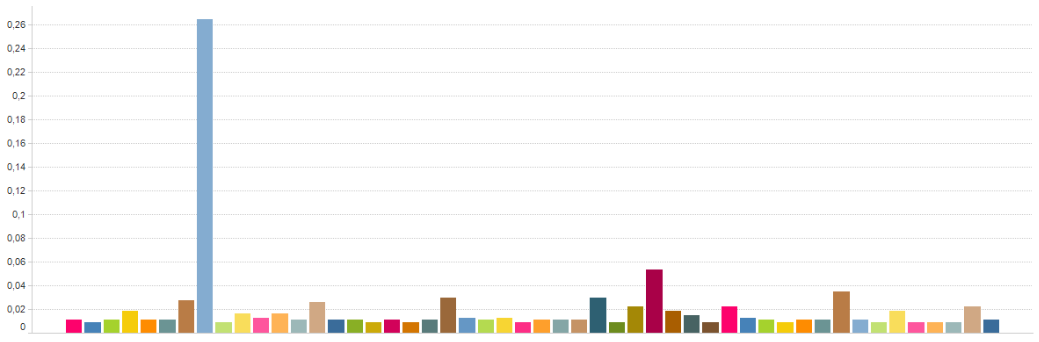


Figure 4.6: Topic distribution for the example document. As can be seen this document is mainly about topic 8.

4.3 Information retrieval

In the information retrieval experiments, a subset of the OHSUMED test collection was used. This subset consist of Medline documents from 1988 to 1991, together with 63 topics and the documents that are relevant for each topic. The relevance judgments are *definitely relevant* and *possibly relevant*. In our experiments these labels are merged into just one set of relevant documents. Furthermore, each topic has a title and description in the dataset. These are merged into a query in our experiments.

When creating the corpus the same filters as in the topic modeling experiments were used. This corresponds to row 5 in table 4.1.

4.3.1 Weighting schemes and distance measures

Both the VS and LSA models use occurrence matrices for internal representation. Using different weighting schemes can improve information retrieval performance. In the first experiment tf-idf weighting was compared to raw term frequencies for the VS model using the cosine distance measure. The results clearly indicate that tf-idf is beneficial, with an 11 point recall average precision of 0.442 compared to 0.359. See figure 4.7.

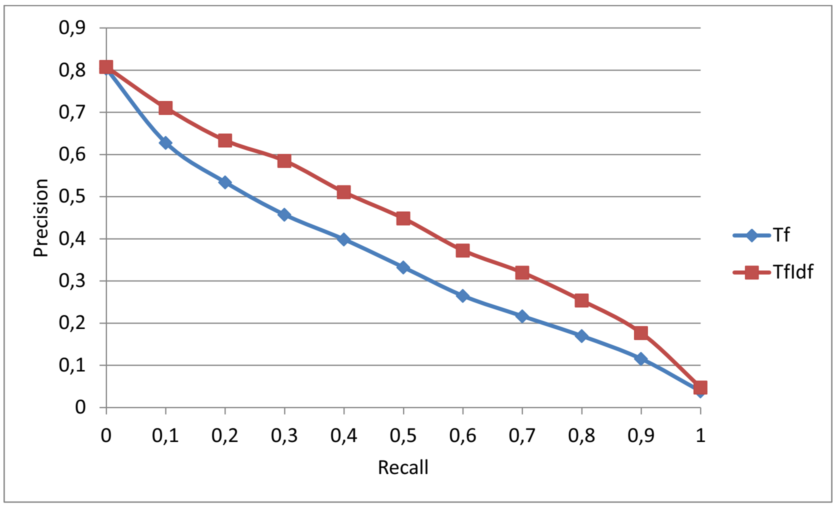


Figure 4.7: Precision for the basic vector space model with and without tf-idf weighting.

The normalized term frequency weighting scheme (without the idf-part) was not used as its performance is no different to raw term frequencies when using the cosine distance measure (cosine normalizes the vectors). In the following experiments, tf-idf was used for both VS and LSA.

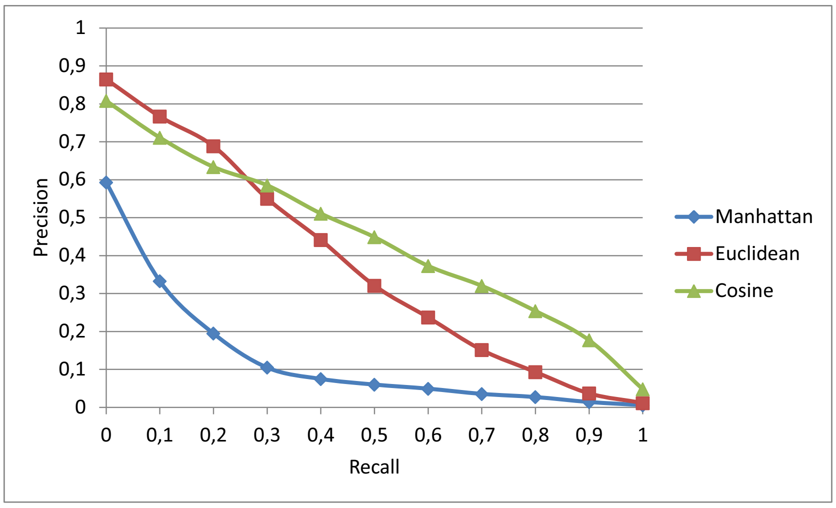


Figure 4.8: Precision for different distance measures at 11 different recall levels. The weighting scheme is tf-idf.

The cosine distance measure proved to give the best results compared to the other vector space distance measures. Euclidean distance was second best at 0.378 average precision, while the Manhattan distance falls far behind. See figure 4.8.

4.3.2 Choosing number of dimensions

All dimension reduction models require a parameter K which indicates the number of dimensions or topics to use. Each model has its own optimal K, so the next experiment shows how it should be set in order to optimize information retrieval performance for this particular corpus. In figure 4.9 the 11 point recall average precision is plotted as a function of the number of model dimensions. It shows that the LDA model outperforms the other models for 600 topics and higher for this corpus. As the VS model doesn't perform any dimension reduction, it is constant. Note that it performs well with an average precision of 0.442 in this benchmark.

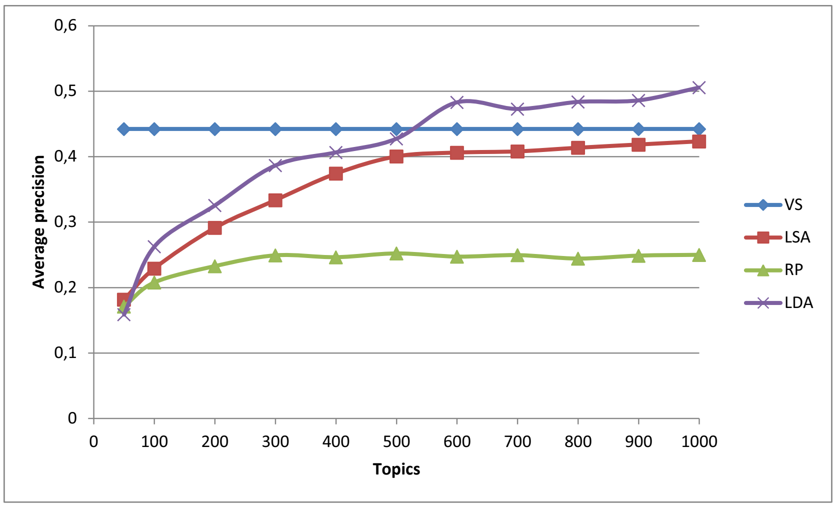


Figure 4.9: Average precision for increasing number of topics in the models.

As the LDA model gets increasingly better results the model perplexity for this corpus was inspected, shown in figure 4.10. It settles at around 600-800 topics, just as the average precision in figure 4.9.

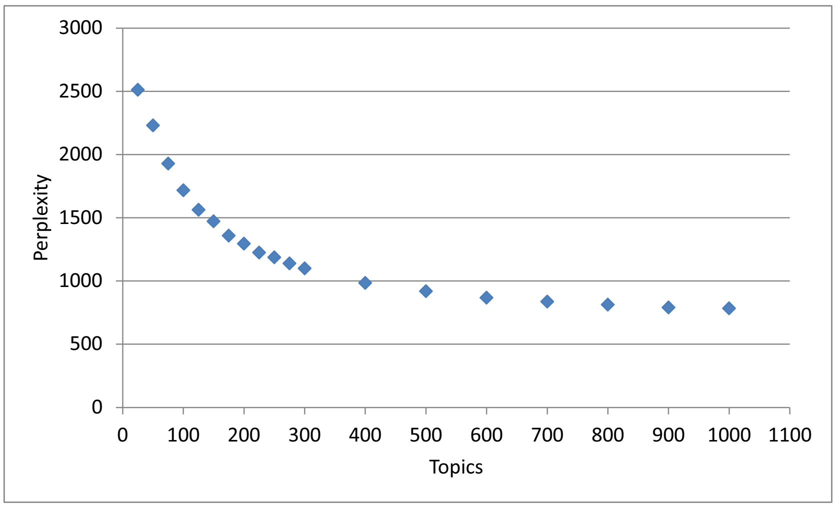


Figure 4.10: LDA perplexity for the Medline corpus for increasing number of topics.

4.3.3 Model comparison

Using the best combination for each model, their precision curves can be compared. Since the average precision doesn't increase much after 800 dimensions for any of the models, these models have been selected for the comparison, shown in figure 4.11 below. It shows that the LDA model with 800 topics performs best. Their building times are shown in table 4.4.

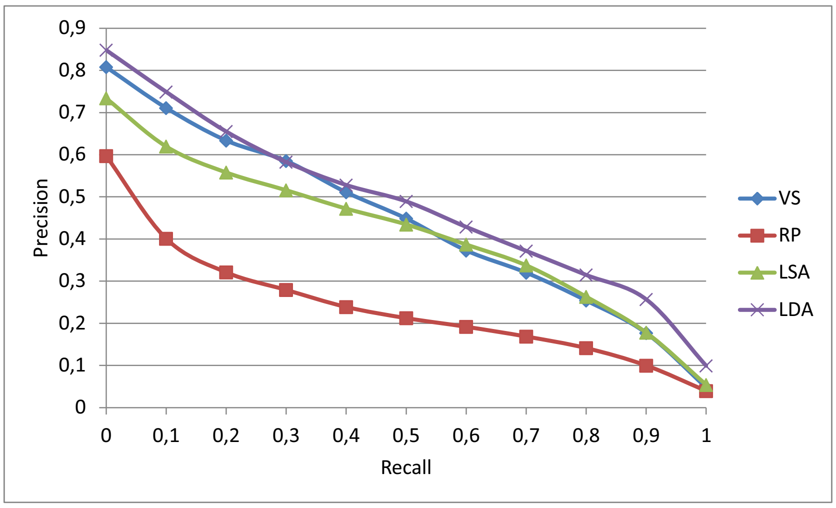


Figure 4.11: Final precision comparison for different models.

Model	Building time
VS	3.5s
RP	4.8s
LSA	20m 26s
LDA	1h 51m

Table 4.4: Building time for the models for the Medline corpus.

4.4 Sentiment detection

For the sentiment detection experiments, two corpora were used. The first corpus is the PolarityDataset, which contains 1000 positive and 1000 negative movie reviews. This is used for general sentiment detection in one domain. As the JST and SLDA models also detects topics in order to account for different words for expressing sentiment in different domains, a second dataset called

MultiDomain was used. It contains reviews from four different domains: books, dvds, kitchen and electronics.

The combined topic/sentiment models were compared to a baseline sentiment detector which simply counts the number of positive and negative words found in each review, and picks the label which had most words. For both corpora, the same filtering scheme as the previous experiments was used, i.e. row 5 in table 4.1. A symmetric γ set to 1 was used, as no recommendation was found in the literature.

4.4.1 Neutral sentiment

The first experiment to perform was to decide whether to only model words into positive or negative sentiment, or to also include a neutral label. The sentiment lexicon MPQA [35] was used in all experiments, and it contains neutral sentiments as well. As most words in a review belong to the neutral label, the final classification only considered the positive and negative labels. In this experiment 1 topic was used and the accuracies were averaged over 5 models.

	Baseline	JST		SLDA	
Dataset	PN	PN	PNN	PN	PNN
PolarityDataset	65.3%	68.9%	68.1%	69.9%	69.3%
MultiDomain	60.6%	52.4%	62.3%	52.4%	62.2%

Table 4.5: Accuracy of the three sentiment detectors with and without neutral labels. PN = Positive and negative labels. PNN = Positive, negative and neutral labels

Table 4.5 above shows the average accuracy for the three detectors with and without neutral labels. Note that JST and SLDA get very similar results as they in practice become the same model when the number of topics is set to 1. For PolarityDataset the incorporation of neutral labels didn't result in any improvement, but for MultiDomain there were clearly a rather large improvement. Hence it was decided to use neutral labels (PNN) for the following experiments.

4.4.2 Choosing number of topics

Just as with LDA, choosing the right number of topics is of importance. While the PolarityDataset only contains reviews from one domain, there are several genres and/or aspects of movies which could have different word uses. This can be of importance when analyzing the results in detail. For example, one might get a topic which treats acting performance. Also, positive words for a scary movie might not be positive for a movie for children, suggesting that also the PolarityDataset may benefit from more topics. This leads us to examine how the number of topics affects the results. The accuracy of the models for PolarityDataset using different number of topics were tested and the results are shown in figure 4.12 below. For each number of topics, the results of 5 models were averaged.

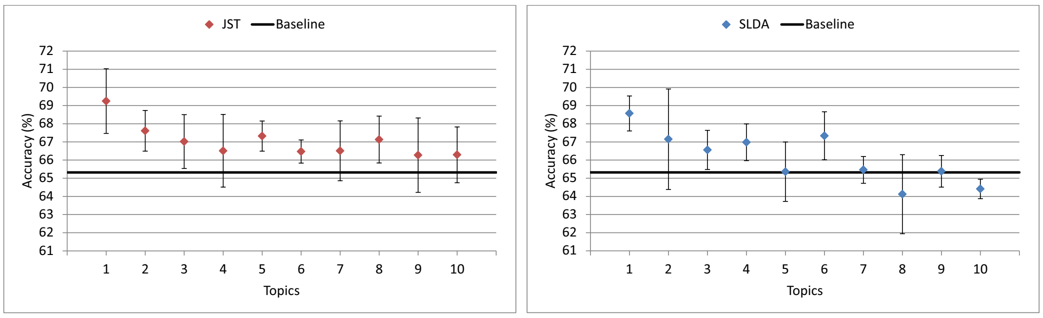


Figure 4.12: Accuracy plotted as a function of number of topics for Polarity-Dataset. The baseline sentiment detector is constant as it doesn't model topics.

The results shows that the classification accuracy for both models decreases as more topics are used. SLDA even falls below the baseline classifier for higher number of topics. Also, the results vary heavily for different model instances as shown by the standard deviation error bars.

Another experiment was performed on the MultiDomain dataset, in order to see if there is any difference in performance when the reviews are from different domains. The results are shown in figure 4.13. Here we can see that the accuracy increases slightly from having only one topic to a couple more. Still the quality of the results vary heavily shown by the error bars.

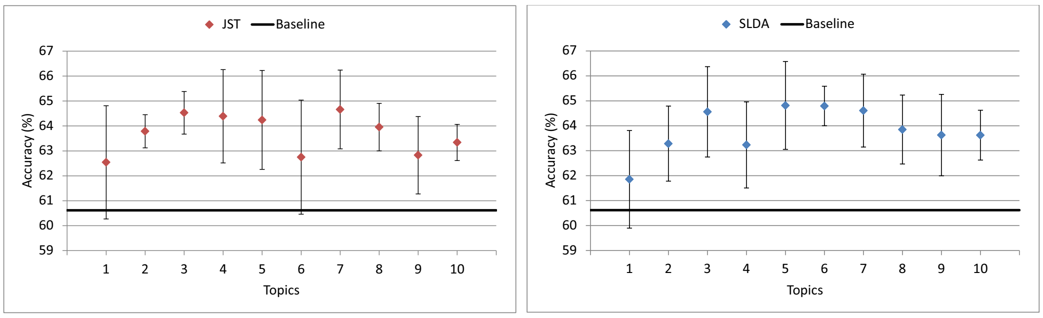


Figure 4.13: Accuracy for the MultiDomain corpus.

4.4.3 Examples

The pure classification performance of the combined sentiment/topic models is interesting as an evaluation measure, however it is most likely the extracted topics that are of interest for the user. As in LDA, the most probable words for each combined topic and sentiment can be extracted from β to get an idea of what the topics represent. In table 4.6 below are the 10 most probable words for two topics extracted by a 5 topic SLDA model for the Polarity-Dataset corpus. It can be seen that topic 4 is about animated movies and topic 5 is about romantic comedies.

	Topic 4			Topic 5	
Positive	Negative	Neutral	Positive	Negative	Neutral
star	horror	know	funny	bad	new
disney	ship	show	love	comedy	city
young	scream	re	humor	guy	james
joe	killer	big	best	script	york
voice	effects	think	romantic	least	big
new	last	ve	gets	doesn	enough
animated	deep	now	role	nothing	bond
murphy	movies	don	friends	isn	lee
wars	summer	something	town	goes	godzilla
original	scary	still	kids	re	take

Table 4.6: The most probable words for two sentiment topics in the Polarity-Dataset corpus.

4.5 Visualization

In the final experiments, the self-organizing map algorithm was evaluated. As the *random-restart hill climbing* algorithm described in section 3.4 is an approximation algorithm, the first experiment examined the number of random restarts needed for a good result, to give indications of how many to use for other data. The learning rate parameter α was set to 0.4, map size to 250×250 and it was run for 1000 iterations [1]. In this experiment synthetic data in form of 5 unit vectors in 5 dimensions was used for easy evaluation. Figure 4.14 below show final error maps for different number of random restarts.

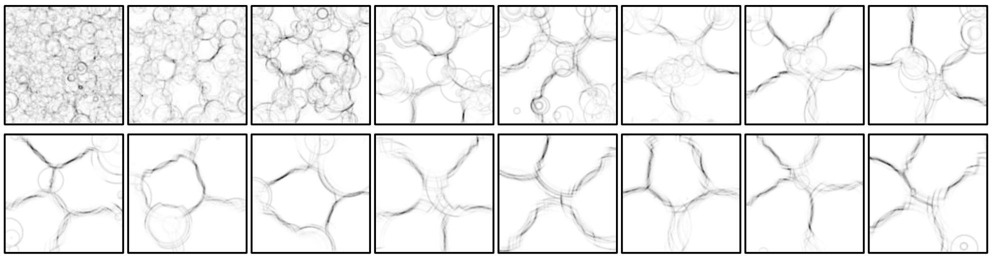


Figure 4.14: Error maps for increasing number of restarts. The top left map has used no restart (only one sample), up to 16 samples for the bottom right.

An error map (also called U-matrix, unified distance matrix) is a bitmap where the brightness of each pixel is scaled by how similar its weight vector is to its neighbors. Dark areas indicate large changes in the weight vectors between neighboring nodes, i.e. borders between clusters. In the above example a good result is 5 clearly distinguishable clusters, which is achieved already at 6-7 random restarts. See figure 4.15 for a solution from the exact algorithm where all neurons are inspected to find the best matching. For a further discussion about the random restart algorithm, see section 5.5.

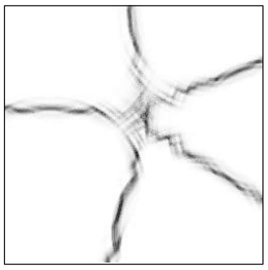


Figure 4.15: Error map from a run with exact best matching.

[1] α and iterations recommended by `http://www.multid.se/genex/hs530.htm`

In the next diagram, the random-restart algorithm is compared to the standard solution in running time. This time the 20NewsGroups corpus was used, fitted by an LDA model with 20 topics. The time measured is the total building time, i.e. first building the map using 10000 iterations, then finding the best matching neuron for each of the 2020 documents in the dataset. It can be seen that the approximation algorithm is about 10 times faster, with a linear increase in running time for number of restarts.

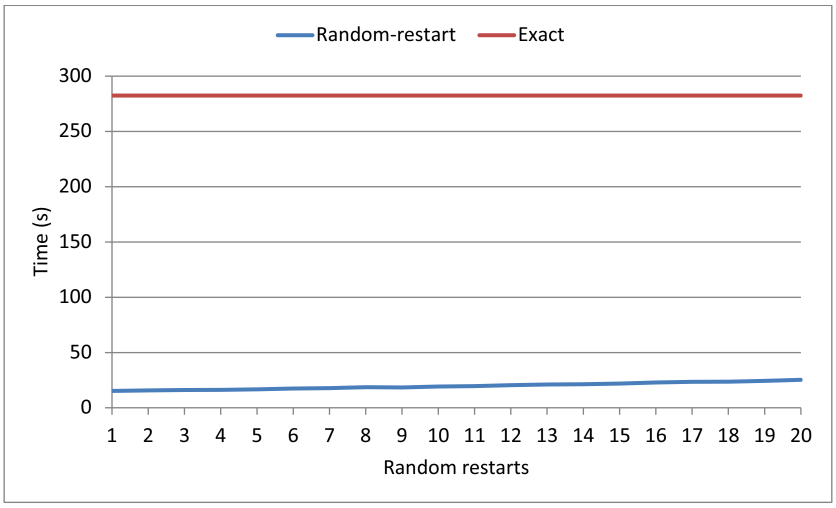


Figure 4.16: Running time for self-organizing map

As an example use, the words for the 20NewsGroups corpus was plotted onto the self-organizing map and was visualized using the scatter plot in Spotfire. The words was then colored by their probability of being generated by the topic about space, see figure 4.17. As we can see, the words *jupiter*, *mercury* and *pluto* are all plotted near each other.

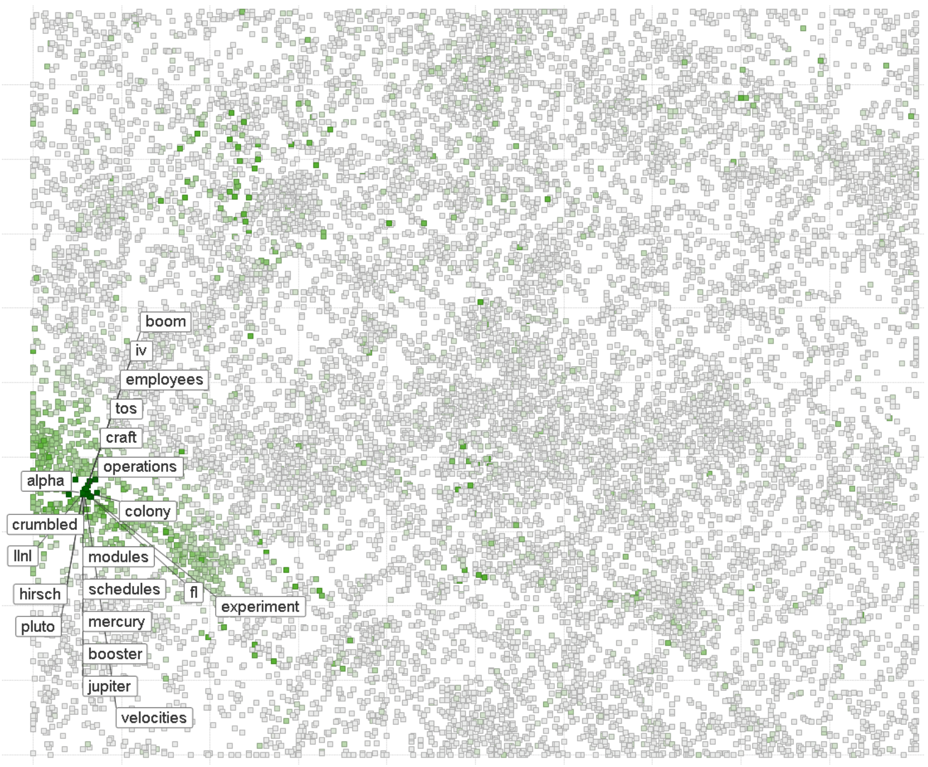


Figure 4.17: Words from 20NewsGroups in a self-organizing map.

Chapter 5

Discussion

This project has been a combination of research in the text analysis area and production of a prototype text analysis pipeline. The primary focus has been to find and evaluate techniques which could be interesting for a future integration into Spotfire. In this chapter we discuss the techniques that have been implemented and the results from the experiments. Future possible enhancements are also discussed.

5.1 Preprocessing

The tokenizer and word filters implemented in this project is a compilation of ideas presented in the literature. As the implementation allows the filters to work on the entire corpus at once, collection wide statistics can be used for filtering. An example of this is the filter which removes the n most common terms. The downside with this approach is that documents cannot be added once the corpus has been built (i.e. the filters have been applied). Adding new documents could for example introduce new concepts to the corpus, but those words could be removed by the `MinWordFrequency` filter as they had a low frequency in the original documents. The cost induced by this restriction is that the corpus needs to be rebuilt whenever new documents are added. As the tokenization and filters are linear in running time, this was not considered a problem compared to the running time of building the latent semantic models later in the pipeline.

A document is in our implementation represented as a list of term indices. As a term may occur several times in the same document, memory could be saved by instead saving the document as a list of (term,frequency) pairs as in the bag of words model. It was however decided to use the former representation as it allows future models to use the word order information.

As mentioned in the theory, the tokenization and preprocessing steps used in this thesis are quite simple. The regular expression used for tokenization in the experiments was naïve and didn't even produce correct words in some cases. This was due to the decision to put more focus on the analysis. Still, there are several opportunities for implementing more features here:

Part-of-speech tagging adds a label for each word, indicating its word category [45] based on surrounding words. Common word categories are nouns, verbs, adjectives, adverbs etc. This could disambiguate homonyms, which would improve the techniques presented in this thesis.

Named Entity Recognition is the task of finding named entities in text and sorting them into categories, for example people, organizations, dates, locations and so on [46]. This technique could prove useful when examining for example which companies that tend to co-occur.

Natural language parsing determines the grammatical parse tree of a sentence. Some experiments with combining this with LDA have been done with lower perplexity as a result [47].

5.2 Topic modeling

Early in the project, it was concluded that LDA and generative models are a hot topic in the area. Therefore, the main focus has been on this probabilistic approach. The advantage of the probabilistic topic models is that they not only give a low dimensional representation of the documents (like LSA) for finding latent semantic relationships, but they also provide meaningful interpretations of each latent dimension by extracting the most probable words. Choosing the number of such topics to use is however a problem, as choosing too many gives uninterpretable topics while too few gives too broad. The held-out perplexity measure can be a good indicator, however in the end it is up to the user.

Although creating a model usually takes more time than a user can wait when using Spotfire, some optimizations can be made to lower the LDA model building time. FastLDA [48] by Porteous et al. provides a fast sampling step for the basic LDA model, which gives a lower expected complexity of the factor K in $O(DNK)$. Experiments shows that this improves performance by 3-8 times, increasing in the number of topics used.

The Gibbs sampler is currently single threaded, although it can be multithreaded as in MALLET [49] and Newman et al. [50]. This approach basically splits the corpus between the threads in the sampling process, ignoring update conflicts to the count arrays. The conflicts are eliminated by simply recalculating the count arrays every ith iteration. This gives an almost perfect parallelization with neglectable sampling errors [50]. Finally, the entire sampler is now implemented in C#, so performance could be improved by moving the code to for example C++.

Also memory use can be of importance when working with large corpora. The count arrays n_x used in the topic models are usually sparse, which indicates that a sparse data structure could be used. The sparsity of these arrays is actually controlled by the hyperparameters α and η.

One problem with the implemented LDA model is that there is no support for incremental adding of documents after the model has been built. Current research now focuses on such incremental models [51], however due to time restrictions these were not examined. Another benefit of such techniques is that an LDA model then can be trained with only a small subset of the users corpus. Hence the user wouldn't have to wait hours for moderate sized corpora. Instead a smaller model would be built, and the rest of the documents added later on.

Even though in text mining one tries to discover previously unknown relationships, it could be of interest for a user to be able to specify something about what the topics should contain. This would mean that a way of assigning priors for topics have to be incorporated. One way would be to let the user choose some small number of words that a topic should be about. How to accomplish this sort of behaviour have to be investigated. Two approaches we believe would work is to either always sample these particular words into the corresponding topic, or increasing the count variables during initialization so that these words get a high probability for these topics. By implementing user specified topics, the user can set up topics that he or she already knows are present in the corpus.

5.3 Information retrieval

In the experiments it can be seen that many dimensions have to be used to beat the vector space model. If information retrieval alone is to be used, i.e. being able to filter on topics is not of importance, then the simple vector space model with tf-idf applied might be used instead of for example LDA and LSA. This can be seen in the experiments where precision for the vector space model is comparable to these models. The important thing here is that the time for building a vector space model is magnitudes faster than for the other two. It should be noted that the Medline corpus which was used in the experiments uses large amounts of medical terms. These are often very specific, which we believe could reduce the need for latent semantic models. To examine if this is the case, another evaluation corpus could be used.

In the experiments, the average precision measure was used to evaluate performance. It assumes that there is an unsorted set of documents in the corpus that are relevant for a certain query. In reality, a user wants to get the 10 most relevant documents presented and disregards the rest of the results. This calls for another evaluation method which uses a corpus with sorted relevance sets for the queries, and puts more weight to the first hits returned by the system.

For pure search engines there are other scoring functions which doesn't simply compare two (weighted) term frequency vectors like in our implementation. The most popular of these functions is the Okapi BM-25[1], which could be implemented and evaluated if only a search engine is to be integrated into Spotfire.

It would also be interesting to combine approximate string matching with the latent semantic space. An example would be to incorporate Netrics which is a fast approximate string matching technology [2], and then assign a weight for the importance of query-string and query-semantic similarity. It could also be used as a preprocessing step to detect and correct misspellings.

5.4 Sentiment detection

The result of the sentiment detection experiments was not always as expected. Increasing the number of topics gave better results for the MultiDo-

[1]`http://en.wikipedia.org/wiki/Okapi_BM25`
[2]`http://www.netrics.com/`

main corpus while it degraded performance for the PolarityDataset, indicating that combined sentiment/topic models perform well for mixed domain data. Common for both evaluation corpora is that the classification accuracy for JST and SLDA vary remarkably. This particular problem has not been presented in previous work, which raises the question whether these two models would be appropriate to use. In a realistic setting only one model is estimated, hence the ability to detect correct sentiment for a model might be worse than expected. We believe this could be improved by incorporating a more extensive sentiment prior, since there are a huge amount of random assignments made for each document. Further experiments have to be made to evalute how much impact the prior sentiment assignments have.

A better sentiment prior is only one of many improvements that could be made to the sentiment detection models. Below are two examples:

Negation An aspect that is overlooked for prior sentiment assignments is negated terms. Consider the sentence *This is not a good movie.* Since the term *good* is positive, a positive label will always be assigned to it using the current approach. In this case however *good* should be assigned a negative label, since the word *not* negates it. More or less sophisticated techniques could be incorporated to find these negated words, for example NegEx [3].

Conjunctions There is also a potential to increase accuracy by modeling conjunctions in sentences. Consider the sentence *The story is good but the actors are not.* This sentence clearly expresses a positive opinion on the story but a negative one for the actors. Here, *but* is changing the polarity. There exists such a model which extends SLDA by modeling conjunctions, called Dependency-SLDA [31].

For the experiments done it should be noted that the corpora have documents that on a whole are classified as positive or negative. However, we are in fact trying to detect positive and negative topics, hence these corpora might not be optimal for evaluation. It is usually the case that documents such as reviews include both positive and negative aspects, and also neutral statements. So a corpus annotated on either word or sentence level would be interesting to perform experiments on.

The running time for the sentiment detectors was not measured in the experiments, however their running time and memory use is clearly higher than LDA. This is due to the extra factor added by the sentiment labels. As they

[3]`http://code.google.com/p/negex/`

are so similar to LDA, the same optimizations as those mentioned in section 5.2 should apply here as well.

5.5 Visualization

For the self-organizing map implementation, the original incremental building method was used. Other approaches like the batch method [4] can give even lower running times. The batch method basically considers batches of data samples at a time and does an update of the map based on all of them at once. Still, the random-restart method proposed in this thesis should be possible to combine with the batch method.

The number of random restarts needed for the synthetic data in the experiments was as expected as the data could be clearly separated into these clusters. Based on this, we believe that the number of restarts should be chosen corresponding to the diversity of the input data. However, this has to be investigated further as time restrictions prevented more experiments.

Finally, for a text mining application the map could be made even more informative to the user. As both documents and words are projected down to K dimensions by the dimension reduction models in this thesis these could be placed in the map together, giving guidance to the user how the corpus is organized. The user could then drill down by examining documents plotted close to words of interest.

[4]`http://www.cis.hut.fi/somtoolbox/theory/somalgorithm.shtml`

Chapter 6

Conclusions

The goal of this thesis was to explore and analyze different text analysis techniques for topic modeling, information retrieval and sentiment detection, to give hints of possible integration in Spotfire. A restriction made was that all the techniques should be unsupervised, i.e. they should not need annotated training data. The implementation described in this project is an example of how a text analysis solution could be implemented. It allows for easily implementing new filtering schemes or other preprocessing steps. It also makes it easy to extend with more models. Based on this implementation, experiments were made for evaluating the techniques.

It has been shown that for information retrieval, i.e. finding relevant documents given a query, all the implemented models perform relatively equally on the specific test corpus, with LDA slightly outperforming the other. One big issue however is that an LDA model takes considerably more time to build than a vector space model which performs well in our experiments. Hence for implementing information retrieval in Spotfire, if preprocessing time is of importance, it is suggested that the vector space model should be used. The experiments have also shown that different measures for assessing document similarities give different information retrieval results, but the cosine distance measure worked well for all models. Regarding the weighting schemes, tf-idf proved to be the best choice.

With topic modeling implemented in Spotfire, a user would be able to perform filtering on topics to discover related documents. However, due to the time needed for inference, either the model building should be done separately or a faster sampler would have to be used. If user waiting times are critical, building a model in the client using the current implementation is not a viable solution for large corpora. Still, a model could be built on a static data set and saved along with the analytic application, and then be efficiently used by the end users. Extracting topic distributions and performing search on a built model are efficient operations and can be done in real time.

We have also shown that combined topic/sentiment detection using JST or SLDA might not be good solutions, since the accuracy from the experiments vary remarkably for different models. However, this accuracy is based on documents annotated as positive or negative, while these models find positive and negative topics. Hence a different annotated corpus should be used for evaluation. Still, the topics given by the sentiment models are informative and could be used for user guidance.

By using a self-organizing map, documents in concept space can be projected to two (or three) dimensions for visualization. However, for better text mining capabilities, a text oriented visualization could be implemented. For example the documents could be plotted together with their most probable words, so that the user directly in the visualization can see what documents are about and where they are in the semantic space.

To conclude, using text analysis could be beneficial since approximately 80% of all data is in unstructured form. The main trade-off to consider is speed and efficiency versus the intricacy of the technique(s) used. The time needed for building a model such as LDA would be a big bottleneck for software like Spotfire where the key is real time data processing, but as discussed improvements can be made.

Bibliography

[1] Christopher C. Shilakes and Julie Tylman. Enterprise Information Portals, 1998. Accessible at: `http://ikt.hia.no/perep/eip_ind.pdf`.

[2] Seth Grimes. Unstructed Data and the 80 Percent Rule, 2008. Accessible at: `http://clarabridge.com/default.aspx?tabid=137&ModuleID=635&ArticleID=551`.

[3] TIBCO Spotfire AB. Spotfire Technology Network. `http://stn.spotfire.com/stn/Default.aspx`.

[4] Marti A. Hearst. What is Text Mining? Accessible at: `http://people.ischool.berkeley.edu/~hearst/text-mining.html`, 2003.

[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2009. Online edition, Draft, Accessible at: `http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf`.

[6] Shanjian Li and Katsuhiko Momoi. A composite approach to language/encoding detection, 2002. Accessible at: `http://www.mozilla.org/projects/intl/UniversalCharsetDetection.html`.

[7] The Snowball Project. `http://snowball.tartarus.org`.

[8] Martin Porter. An algorithm for suffix stripping. In *Workshop on Multimedia Information Systems*, 1980. Accessible at: `http://www.cs.odu.edu/~jbollen/IR04/readings/readings5.pdf`.

[9] Leif Grönqvist. *Exploring Latent Semantic Vector Models Enriched With N-grams*. Växjö University Press, 2006. Accessible at: `http://www2.gslt.hum.gu.se/dissertations/gronqvist.pdf`.

[10] I. S. Duff, Roger G. Grimes, and John G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15:1–14, 1989.

[11] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. In *INFORMATION PROCESSING AND MANAGEMENT*, pages 513–523, 1988. Accessible at: `http://www.apl.jhu.edu/~paulmac/744/papers/SaltonBuckley.pdf`.

[12] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407, 1990. Accessible at: `http://lsi.research.telcordia.com/lsi/papers/JASIS90.pdf`.

[13] Roger B. Bradford. An empirical study of required dimensionality for large-scale latent semantic indexing applications. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 153–162, New York, NY, USA, 2008. ACM.

[14] Pentti Kanerva, Jan Kristoferson, and Anders Holst. Random Indexing of Text Samples for Latent Semantic Analysis. In *In Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, pages 103–6. Erlbaum, 2000. Accessible at: `http://www.rni.org/kanerva/cogsci2k-poster.txt`.

[15] Jussi Karlgren and Magnus Sahlgren. *From Words to Understanding*, pages 294–308. Stanford: CSLI Publications, 2001. Accessible at: `http://www.sics.se/~mange/papers/KarlgrenSahlgren2001.pdf`.

[16] Zoran Pecenovic, Hochschule Lausanne, Politecnico Federale Losanna, F Dra, Le D E Lau, S Anne, Advisors Dr, Serge Ayer, and Prof Martin Vetterli. Image retrieval using latent semantic indexing, 1997.

[17] Trong ton Pham, Nicolas Eric Maillot, Joo hwee Lim, and Jean pierre Chevallet. Latent semantic fusion model for image retrieval and annotation, 2007.

[18] Thomas Hofmann. Probabilistic Latent Semantic Analysis. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 289–296, 1999. Accessible at: `http://www.cs.brown.edu/~th/papers/Hofmann-UAI99.pdf`.

[19] David M. Blei, Andrew Ng, and Michael Jordan. Latent Dirichlet Allocation. *JMLR*, 3:993–1022, 2003. Accessible at: `http://www.cs.princeton.edu/~blei/papers/BleiNgJordan2003.pdf`.

[20] Xiaogang Wang and Eric Grimson. Spatial Latent Dirichlet Allocation. In *Neural Information Processing Systems*, 2007. Accessible at: `http://groups.csail.mit.edu/vision/app/pubs/wangEnips07.pdf`.

[21] Diane J. Hu and Lawrence K. Saul. A PROBABILISTIC TOPIC MODEL FOR UNSUPERVISED LEARNING OF MUSICAL KEY-PROFILES, 2009. Accessible at: `http://cseweb.ucsd.edu/~dhu/docs/ismir09.pdf`.

[22] Hanna M. Wallach, David Mimno, and Andrew Mccallum. Rethinking lda: Why priors matter, 2009.

[23] Thomas L. Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of The National Academy of Sciences*, 101:5228–5235, 2004. Accessible at: `http://psiexp.ss.uci.edu/research/papers/sciencetopics.pdf`.

[24] Yee Whye Teh, David Newman, and Max Welling. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. In *Neural Information Processing Systems*, 2006. Accessible at: `http://www.gatsby.ucl.ac.uk/~ywteh/research/inference/nips2006.pdf`.

[25] Gregor Heinrich. Parameter estimation for text analysis. Technical report, Fraunhofer IGD, 2009. Accessible at: `http://www.arbylon.net/publications/text-est2.pdf`.

[26] Allison Chaney. Topic Model Visualization Engine demo - Wikipedia Topics. `http://www.sccs.swarthmore.edu/users/08/ajb/tmve/wiki100k/browse/topic-list.html`, Accessed at 2011-04-26.

[27] Mark Steyvers and Tom Griffiths. Probabilistic Topic Models. In T. Landauer, D. Mcnamara, S. Dennis, and W. Kintsch, editors, *Latent Semantic Analysis: A Road to Meaning.* Laurence Erlbaum, 2006. Accessible at: `http://psiexp.ss.uci.edu/research/papers/SteyversGriffithsLSABookFormatted.pdf`.

[28] D. Blei and J. Lafferty. Topic models. In *Text Mining: Classification, Clustering, and Applications*, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, 2009.

[29] Jonathan Chang, Jordan Boyd-Graber, Sean Gerrish, Chong Wang, and David M. Blei. Reading Tea Leaves: How Humans Interpret Topic Models. In *Neural Information Processing Systems*, 2009. Accessible at: `http://www.umiacs.umd.edu/~jbg/docs/nips2009-rtl.pdf`.

[30] Qiaozhu Mei, Xu Ling, Matthew Wondra, Hang Su, and ChengXiang Zhai. Topic sentiment mixture: modeling facets and opinions in weblogs. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 171–180, New York, NY, USA, 2007. ACM.

[31] Fangtao Li, Minlie Huang, and Xiaoyan Zhu. Sentiment Analysis with Global Topics and Local Dependency. In *AAAI*, 2010. Accessible at: `http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1913`.

[32] Chenghua Lin and Yulan He. Joint sentiment/topic model for sentiment analysis. In *International Conference on Information and Knowledge Management*, pages 375–384, 2009. Available at: `http://portal.acm.org/citation.cfm?id=1646003`.

[33] Chenghua Lin, Yulan He, and Richard Everson. A comparative study of Bayesian models for unsupervised sentiment detection. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 144–152. Association for Computational Linguistics, 2010. Accessible at: `http://portal.acm.org/citation.cfm?id=1870586`.

[34] SentiWordNet. `http://sentiwordnet.isti.cnr.it/`.

[35] MPQA. `http://www.cs.pitt.edu/mpqa/`.

[36] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982. 10.1007/BF00337288.

[37] Regular expressions - matching behavior. `http://msdn.microsoft.com/en-us/library/0yzc2yb0(v=VS.71).aspx`, Accessed at 2011-05-28.

[38] Doug Rohde. SVDLIBC. `http://tedlab.mit.edu/~dr/SVDLIBC/`, Accessed at 2011-05-03.

[39] Radim Řehůřek. Subspace tracking for latent semantic analysis. In *Proceedings of the 33rd European conference on Advances in information retrieval*, ECIR'11, pages 289–300, Berlin, Heidelberg, 2011. Springer-Verlag.

[40] William Hersh, Chris Buckley, T. J. Leone, and David Hickam. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 192–201, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[41] TREC-9 Filtering track collections. `http://trec.nist.gov/data/t9_filtering.html`, Accessed at 2011-05-17.

[42] The 20 newsgroups data set. `http://people.csail.mit.edu/jrennie/20Newsgroups/`, Accessed at 2011-04-26.

[43] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*, 2004.

[44] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Association for Computational Linguistics*, Prague, Czech Republic, 2007. Accessible at: `http://www.cs.jhu.edu/~mdredze/publications/sentiment_acl07.pdf`. Dataset available at: `http://www.cs.jhu.edu/~mdredze/datasets/sentiment/index2.html`.

[45] Wikipedia - part-of-speech tagging. `http://en.wikipedia.org/wiki/Part-of-speech_tagging`, Accessed at 2011-05-25.

[46] Wikipedia - named entity recognition. `http://en.wikipedia.org/wiki/Named_entity_recognition`, Accessed at 2011-05-25.

[47] Jordan Boyd-Graber and David Blei. Syntactic topic models. 2008.

[48] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 569–577, New York, NY, USA, 2008. ACM.

[49] Andrew Kachites McCallum. MALLET: A Machine Learning for Language Toolkit, 2002. `http://mallet.cs.umass.edu`.

[50] Newman D., Smyth P., and Steyvers M. Scalable parallel topic models, 2006. `http://psiexp.ss.uci.edu/research/papers/parallel_topic_model_newman_20060721.pdf`.

[51] Matthew D Hoffman, David M Blei, and Francis Bach. Online learning for latent dirichlet allocation. *Nature*, 23:1–9, 2010.

Appendix A

Spotfire integration

The user interface for the text analysis toolkit is organized as a set of tools. The general workflow is illustrated in figure A.1. It is assumed that the user has started Spotfire and loaded some data which contains one or more text columns. These are parsed and filtered into a `Corpus` using the corpus manager. The model manager can then be used to create a model from the corpus. Finally, the output tools uses corpora and models to create new tables or columns in Spotfire, where they can be visualized.

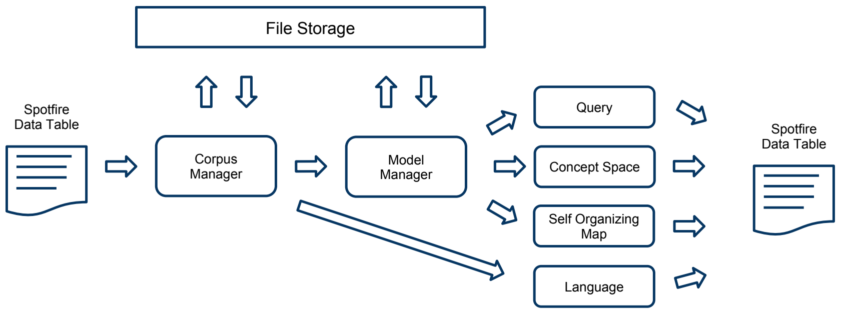
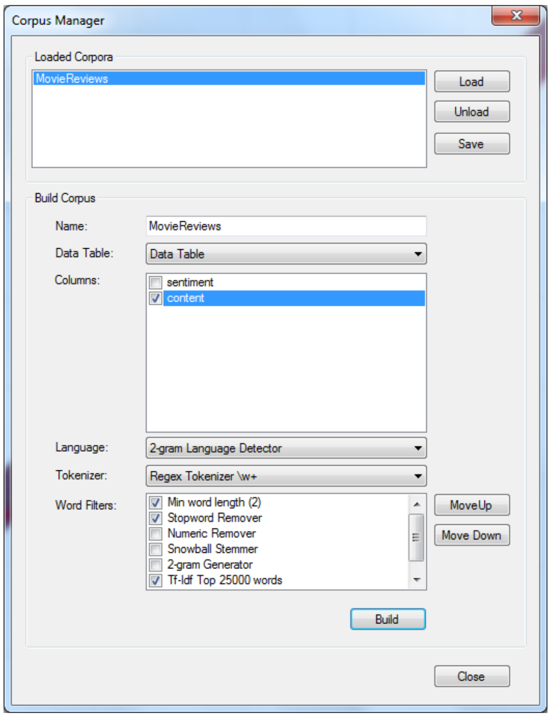


Figure A.1: Spotfire text analysis work flow.

Corpus manager This tool is used for constructing a `Corpus`. The user specifies the data table and the string column(s) that all text should be taken from. The user also choses a tokenizer, a language detector, and which filters to use and in what order they should be applied. When everything has been set-up the corpus is built by treating each row in the selected column(s) as a document. If multiple columns are selected, the content in these will be merged into one document. A reference to the corpus is then saved for further use in the other tools.

There is also support for loading and saving corpora.



Loaded Corpora: MovieReviews

Name: MovieReviews

Data Table: Data Table

Columns:
- [ ] sentiment
- [✓] content

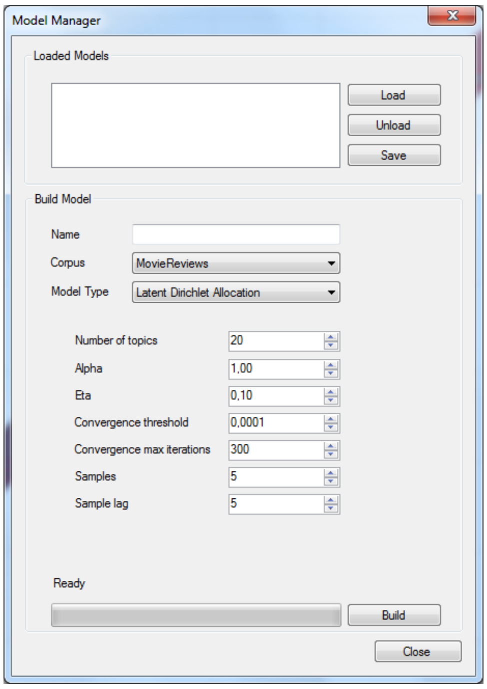
Language: 2-gram Language Detector

Tokenizer: Regex Tokenizer \w+

Word Filters:
- [✓] Min word length (2)
- [✓] Stopword Remover
- [ ] Numeric Remover
- [ ] Snowball Stemmer
- [ ] 2-gram Generator
- [✓] Tf-Idf Top 25000 words

Figure A.2: Corpus manager dialog

Model manager The model manager provides an interface for the user to create a model from a corpus. Each model has it's own properties, for example the maximum number of Gibbs sampling iterations allowed to do for the LDA model. A reference to each model is saved just as in the corpus manager. As a model takes some time to compute, the dialog also allows the user to save an estimated model to file for later use.


Model Manager

Loaded Models

Build Model

Name

Corpus: MovieReviews

Model Type: Latent Dirichlet Allocation

Number of topics: 20

Alpha: 1,00

Eta: 0,10

Convergence threshold: 0,0001

Convergence max iterations: 300

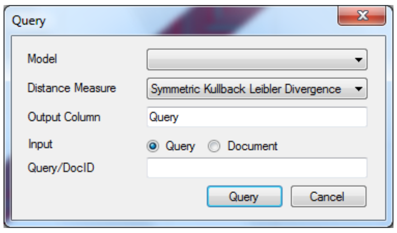
Samples: 5

Sample lag: 5

Ready

Figure A.3: Model manager dialog

Query The query tool is as the name implies used for performing queries. The user selects which model and distance measure to use. Either the user can enter a search string, or the id (row number, zero based) of a document to use as a query. The resulting distance for each document is then outputted to a column chosen by the user.


Query

Model

Distance Measure: Symmetric Kullback Leibler Divergence

Output Column: Query

Input: [✓ Query] Document

Query/DocID

Figure A.4: Query dialog

Language The language dialog lets the user output the estimated language of each document as a new column. This is only meaningful if a language detector other than a `ConstantLanguageDetector` was used.

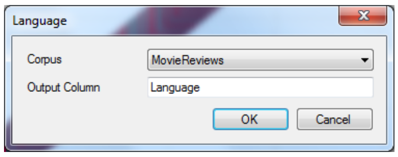


Figure A.5: Language dialog

Concept space The concept space dialog is a simple dialog which lets the user output the document concept vectors (for vector space models) and probability distributions (for topic models) as new columns. It can also output the used vocabulary along with their low dimension representation.

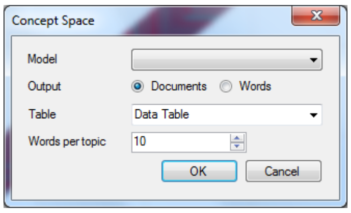
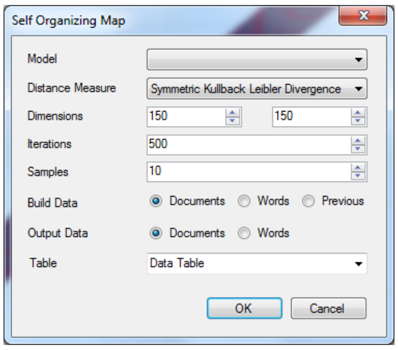


Figure A.6: Concept space dialog

Self-organizing map This tool uses the implemented SOM to output document and term positions in two dimensions. There are a couple of parameters to be set by the user such as the number of iterations and the size of the map. The algorithm to use (approximation or exact) must be also be set by either setting samples to -1 (exact) or a positive value for approximation. When the SOM has been built, the x and y positions for the documents (or terms) are outputted in a table specified by the user. It is also possible to use the previously trained SOM for outputting positions. Hence by training the SOM with for example documents, it is possible to output term positions with respect to document positions.



Self Organizing Map

Model	
Distance Measure	Symmetric Kullback Leibler Divergence
Dimensions	150 150
Iterations	500
Samples	10
Build Data	[✓ Documents] Words Previous
Output Data	[✓ Documents] Words
Table	Data Table

Figure A.7: Self-organizing map dialog