A comparison of algorithms for automated keyword generation

Ein Vergleich von Algorithmen zur automatischen Schlagwort-Generierung

Bachelor-Thesis von Viktor Seifert aus Karaganda (ehem. UdSSR)

Mai 2011

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Computer Science
Knowledge Engineering

A comparison of algorithms for automated keyword generation
Ein Vergleich von Algorithmen zur automatischen Schlagwort-Generierung

Vorgelegte Bachelor-Thesis von Viktor Seifert aus Karaganda (ehem. UdSSR)

1. Gutachten: Johannes Fürnkranz
2. Gutachten:

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den May 2, 2011

(V. Seifert)

Abstract

In automated keyword generation, or topic indexing, an algorithm has to generated topics for a given document or document set. This thesis compares three different algorithms: TF $\times$ IDF, a decision tree learner and the Maui indexer. Three datasets are used to compare the performance of the algorithm which have manually assigned topics. The algorithms are compared using criteria presented in this thesis.

Acknowledgments

Here I would like to thank all the people that helped me prepare this thesis, either by word or deed. The list is not sorted in any way, especially not by importance.

An dieser Stelle würde ich mich gerne bei allen Leuten bedanken die mir bei der Erstellung dieser Arbeit geholfen haben. Die Liste ist nicht sortiert, schon gar nicht nach der Bedeutsamkeit.

Many thanks to / Vielen Dank an:

- Professor Johannes Fürnkranz (Kowledge Engineering, TUD)
- Eneldo Loza Mencía (Kowledge Engineering, TUD)
- Frederik Janssen (Kowledge Engineering, TUD)
- Alyona Medelyan
- Richard Stenzel (ConWeaver GmbH)
- Georg Fette (ConWeaver GmbH)
- Andreas Leder (ConWeaver GmbH)
- Markus Schmitter (ConWeaver GmbH)
- Stefan Garhammer (ConWeaver GmbH)
- Tim (ConWeaver GmbH)

Contents

List of Tables

1 Introduction

To motivate this thesis imagine a company that has been in business for a couple of years. Over this time a lot of documents are created; documentation of projects, contracts and so on. This body of documents may grow to encompass millions of documents and can reach a size of thousands of gigabytes (terabytes) of data. Searching for a particular document in such a large collection may take a substantial amount of time, which in turn has a great impact on an employees productivity.

A search engine can drastically reduce the time spent on searching for particular documents. An employee can query the engine, say through a website form, and the engine, as a result, presents him with a page which lists the documents sorted by relevance. Usually the engine does not present the documents themselves but rather links to the locations of the documents, their titles and a short passage of the documents content in which the user-submitted query is highlighted.

To further enhance the search result the user user could be presented with keywords for each document. These briefly summarize the content of the document and show if it belongs to some category. The keywords could also be used as queries to the engine.
As an example say an employee is looking for documentation regarding a specific project. He would enter the projects name as query to the engine and further tell it to only look for documents which have "documentation" as a keyword.

Manually creating keywords for a body of documents which contains millions of documents could take months or even years to complete, which is infeasible for most companies. Instead an automated extraction technique (algorithm) could be used to generate keywords (topics) for a given document body. This algorithm is run at the companies side, as opposed to the search engine developers side, since a lot of the documents are not to be used outside the company.

This implies that the algorithm should perform **fast** and produce "**good keywords**".

As part of her PhD-thesis[1] Olena Medelyan developed Maui, a new algorithm for automated keyword generation (topic indexing). The goal of this thesis is to compare Maui to other existing techniques.

The next two sections will define some terms used in this thesis and highlight the task of keyword generation.

1.1 Topic Indexing

The previous section used the term "keyword" when referring to elements of a documents topic set. This is actually imprecise since the term "keyword" implies that a topic could only be represented by a single word. But we do not want to restrict the task of topic indexing to only encompass single word topics, we also want to be able to extract topics which consist of multiple words. As an example, we want "foreign politics" to be valid topic as opposed to only "politics" being valid. Hence the more general term "keyphrase" is actually correct.
The term "topic" encompasses both keywords and keyphrases. All these terms will be used where appropriate.

The terms and information used in the following part of this section(1.1) are taken from [Mede09, section 2.1]. This section will reproduce them for clarity and they will be used throughout this paper.

The general term for all document indexing tasks is "topic indexing". These tasks can be broadly categorized by two criteria:

- source of terminology used to generate a documents topics
- number of topics assigned per document

The tasks relevant to this thesis do not differ according to the second criterion. I.e. only a few main topics are extracted per document. They do however differ according the first criterion, namely the source that is used to generate topics. Table 1.1 on page 2 explains the three tasks relevant to this thesis. For more information please refer to [Mede09, section 2.1].
Motivated by the description in section 1 we do not want to restrict the source of the topics. This means that the main interest of this thesis are "keyphrase indexing"-algorithms. The next section will explain the task of "kephrase indexing" in more detail.

[1] See references[Mede09]

Table 1.1: Topic indexing tasks

Task name	Description
term assignment	The topics may only be taken from a large controlled vocabulary.
keyphrase extraction	The topics may be any phrase or word in the document.
keyphrase indexing	A topic may be any word or phrase. It is not necessary that it be contained in the document.

1.2 Keyphrase Indexing

Given are two sets of documents:

- A training set T of documents. For each $t \in T$ there is an associated set K_t of keyphrases.
- A set S of documents which do not have any keyphrases associated.

The goal is to generate for each $s \in S$ a set K_s of keyphrases using the information in the training set. As mentioned in 1.1 it not necessary that a keyphrase be contained in the document itself.

The part of using the training set and/or its associated keyphrase sets is purely optional for any algorithm. In fact any algorithm is not required to look at T or any of the sets K_i at all. The only requirement is that the algorithm generates "good" topics. What a "good" topic is will be explained in section 4.

1.3 Scope of this thesis & Outline

The goal of this thesis, like already metioned on page 1, is to compare different algorithms for automated topic extraction. The conception of the algorithms is not part of this thesis. Existing software libraries were used where possible. Modifications made to the algorithms will be noted in their corresponding sections.

Part of this thesis is finding datasets which can be used to test the algorithms. **Section 2** will describe the datasets which were used evaluate the algorithms. It will also show the preprocessing steps applied to each dataset individually.

Section 3 will present the algorithms which were evaluated in this thesis and describe how these algorithms have been chosen. It will also show the preprocessing steps which were applied equally to each dataset before it was used as input to the algorithms.

Section 4 will show and explain the criteria which were used to evaluate the algorithms. It will also present the summarized experiment results for each algorithm and evaluate each algorithm according to the criteria given in this section.

This section will not contain the detailed experiment results, i.e. the tables that contain the name/id of each document and its extracted and original keyphrase set. These can be found on the authors website at `http://viktor-s.net/BachelorThesis`. The web-page also contains results of some experiments that are not presented in this section.

Finally **Section 5** will summarize the results of the evaluation.

2 Datasets

To evaluate the algorithms suitable datasets are needed. Following the motivation from section 1 it is desirable that such a datasets have a few properties presented in this section. Table 2.1 lists theses properties.

Table 2.1: Dataset properties

1. The dataset should be sufficiently large to emulate the task from section 1. Desirable would be a size of several million documents.
2. The dataset should already have topics associated with each document. This can be used to compare a documents existing topic set to one produced by an algorithm.
3. The dataset should contain a wide variety of documents. Examples are: documentation, news articles, literary texts & reviews.
4. The dataset should be freely available in machine readable form.
5. The dataset should be in German language.

Of these properties number 2 is critical, since it would be very difficult to evaluate an algorithm fully without any examples. But it is impossible to find a dataset that has all the desired properties. Document collections usually contain only homogeneous documents (e.g. only news articles). Combined with the fact that all documents should have topic set, no dataset could be found that has all the desired properties.

To still have a good base for evaluation datasets were chosen that meet the properties as close as possible. Differences to these properties can have a significant impact on the task of "keyphrase indexing". Therefore it is important to note how any given dataset differs from the desired properties and how this influences the performance of the algorithms.

Each subsection corresponding to a dataset will list theses differences and explain the impact the have. Also each subsection will present properties that are not mentioned in table 2.1.

The subsections will also describe the preprocessing steps applied to each dataset. These are individual and vary from dataset to dataset. The intent and purpose of the preprocessing will also be explained in each subsection.

The preprocessing steps applied to each dataset equally will be shown in section 3.

2.1 Zeit.de

The Internet presence[1] of the German newspaper "Die Zeit" has an archive of the newspapers articles[2]. The archive dates back to the year 1946. It contains all articles written for "Die Zeit" or "ZEITmagazin". All articles are available online in XML-format. To obtain an article in XML-format one only needs to replace the "`www`"-part of its url by "`xml`".

XML can easily be parsed by a computer. Almost any programming language, that is in broader use today, has a library that can process XML-input.

Beginning with the year 2008 each article also has a set of keyphrases. These can be used to filter articles by topics. By clicking on a keyphrase the user only gets presented with the articles from the archive which contain that keyphrase in its topic sets. Articles without a topic set (2007 and earlier) are excluded from this filtering mechanism.

The articles were downloaded to a computer with a web-crawler[3] configured for the task. This was done in February 2010 and yielded a collection of approximately 2600 news-articles. Only the articles which had a keyphrase-set were downloaded.

The XML-version of each article also contains the keyphrase-set. Additionally an article may have a category and sub category. These category were only used as additional keyphrases. This could lead to duplicates in the keyphrase-set of an article. These duplicates were removed during the preprocessing of the articles.

Each keyphrase-element of an article also has a XML-attribute named "source". On all articles this was set to "manual". It is therefore assumed that the keyphrases were assigned by humans, most probably by the author of each article. This leads to to unwanted properties on the side of the keyphrase-sets. Most of the keyphrases were assigned very seldom to

[1] `http://www.zeit.de`

[2] `http://www.zeit.de/2011/index`

[3] `http://en.wikipedia.org/wiki/Webcrawler`

an article. This is most likely due to inconsistencies in the way that humans assign keyphrases[4]. The result is that about 90% of the keyphrases were assigned less than 10 times. And of these approximately 40% were only assigned once or twice.

A keyphrase that is assigned only once or twice could be interpreted as a document id. It was assumed that low-frequency keyphrases would cause the indexing algorithms to behave in undesired ways. Therefore, during the preprocessing of the articles, all the keyphrases, that were assigned less than 10 times, were removed from the keyphrase-sets.

As further preprocessing all elements of the XML-input were removed except the textual content of the articles. Headings of parts of the articles were not marked in the output of the preprocessing, i.e. they appeared as plain text in the articles.

Point number 1 of table 2.1 says that a dataset should have size of few million documents if possible. The "Zeit.de"-dataset only contains only ca. 2600 documents, thus missing the desired number by far. Although it cannot emulate the task from section 1, the number of document is deemed sufficient to evaluate the algorithms from section 3.

On the positive side this dataset meets all the other properties from table 2.1. Except that it only consists of news articles, thus slightly violating property number 3. It is also the only dataset that has topics that could be freely chosen. Even after removing low-frequency keyphrases a lot of the documents had keyphrases associated which could not be found in the document itself.

All these properties combined make the "Zeit.de"-dataset a reasonable choice as testing dataset.

2.2 Reuters

The Reuters-dataset[5] is a collection of short news texts from Reuters[6]. Only the German documents were used in this thesis which results in a dataset of ca. 210,000 documents. These are not full news articles but short texts, each only a few sentences long. The collection is available in machine readable form on CD from NIST[7]. It contains all the documents in XML-format.

Each document also has an associated topic set. The list of topics does not contain the keyphrases themselves but is a list of topics codes. The collection contains a file which lists all these topic codes and a description, which is short phrase in English language. Through this file it is possible to translate the topic codes to plain-text phrases thus creating keyphrase sets that are usable by the indexing algorithms. This was done in a preprocessing step.

But not all topic codes had a corresponding description. Since the descriptions could not be guessed from the topic codes, all topic codes that did not have a description were removed from the topic sets as part of the preprocessing.

Before that the topic descriptions had to be translated to German. As a translation the closest translation which appears as the title of an article in the German-Wikipedia was used. This was done to ensure better compatibility with the Maui-Indexer[8]. The translation was done manually by the author of this thesis, before applying any preprocessing to the dataset.

Although the Reuters-dataset is considerably larger than the "Zeit.de"-dataset it still falls short of the goal of having a dataset with a few million documents. But since it contains far more documents than the "Zeit.de"-dataset it also is sufficiently large to provide a good base for evaluation.

In section 1.1 we stated that we do not want to restrict the choice of keywords. But on the "Reuters"-dataset the choice of keywords was restricted to a controlled vocabulary, i.e. the available topic codes. This actually changes the task, as described in section 1.1, from "keyphrase indexing" to "term assignment". And it also will have an impact on the performance of the algorithms.

The restriction of the topics to fixed set has the effect that the do not overlap thematically. In other words, it always clear which topic should be assigned to a document. This greatly reduces the number of erroneously assigned topics and makes the frequencies of the topics evenly spaced.

[4] See [Mede09, Section 4.1.2]

[5] `http://trec.nist.gov/data/reuters/reuters.html`, RCV2

[6] Reuters Group Limited, a news agency, `http://en.wikipedia.org/wiki/Reuters`

[7] National Institute of Standards and Technology, `http://www.nist.gov/index.html`

[8] See section 3.3

2.3 Wikipedia

The Wikipedia[9] is a "free, web-based, collaborative, multilingual encyclopedia project. Its ... articles ... have been written collaboratively by volunteers around the world, and almost all of its articles can be edited by anyone with access to the site"[10]. Its whole content is available for download as a huge XML-file-dumps[11].

The version used in this thesis contains about 1.1 million articles in German language. These articles however do not have topic sets. Each Wikipedia-article has a short introductory section which summarizes the articles contents. By looking at the phrases which are used as links to other Wikipedia-articles one can see that they give a good overview of what the article is about.

Thus the following was chosen as a method of generating topics sets for the articles.

- Let A be an Wikipedia-article.
- Let $\mathscr{A}(A) := \{A_i : i \in \mathbb{N}\}$ be the set of Wikipedia articles pointed to by hyper-links from A.
- Let t_i be the title of the article A_i.
- Then the set $T(A) := \bigcup t_i$ becomes the topic set of A.

This leads to a collection of documents whose topic-sets vary strongly by their size, ranging from 4 topics to more than 150. The high end of this range encompasses pages from Wikipedia which merely list other articles by a category [12] and long articles that contain a lot of outgoing links [13]. The low end of the range mostly contains so called article stubs, articles that were created by one person but cannot be considered complete. It does not contain disambiguation pages, which are filtered during preprocessing.

As we will see in section 4 the Maui-Indexer cannot handle large amounts of documents. So we need to reduce the number of documents in this dataset. As criterion the number of topics was chosen. Only documents which a have a number of topics in the range $[8, 12]$ were kept in the dataset. The resulting dataset consists of ca. 10,000 documents.

Using the Wikipedia as dataset is not optimal. The XML-dump itself does not contain any topics sets. But freely available document collections, which contain a large number of documents, are rare. The Wikipedia was the largest collection that could be found. It was the only one that could be compared to property number 1 from table 2.1. The reduction to 10,000 documents only came after one experiment, namely running the Maui-Indexer on the "Reuters"-dataset.

[9] http://en.wikipedia.org/, http://de.wikipedia.org/

[10] http://en.wikipedia.org/wiki/Wikipedia

[11] http://en.wikipedia.org/wiki/Wikipedia:Download

[12] Examples: http://en.wikipedia.org/wiki/1946 & http://de.wikipedia.org/wiki/1946, http://en.wikipedia.org/wiki/Classical_composer & http://de.wikipedia.org/wiki/Komponist

[13] Example: http://en.wikipedia.org/wiki/Europe & http://de.wikipedia.org/wiki/Europa

3 Algorithms

From the motivation in section 1 we can derive guidelines for choosing the algorithms to be evaluated. They are shown in table 3.1.

Table 3.1: Algorithm choice guidelines

1. An algorithm that works as fast as possible. It will serve as a runtime-benchmark, possibly providing a lower bound to which the other algorithms can be compared.

2. An established algorithm from machine-learning/data-mining. This algorithm serves as a comparison for the first algorithms quality. It is compromise between the first and second algorithms.

3. The Maui-Indexer[1]. This is a new topic indexing algorithm. It promises very good results.

As we will see in section 4, for evaluation purposes, the datasets have been split into two subsets, a training and a testing set. In this section these will be referred to as D_{train} and D_{test} respectively.

The following subsections will in turn explain the algorithms chosen, in the order of the guidelines from table 3.1.

3.1 TF × IDF

TF × IDF is a measure from information retrieval[2]. It is used to measure the importance of a term to a document in a document collection. The formula consists of two parts:

- TF stands for *term frequency*.
  Informally, the term frequency will be higher the more often a term appears in a given document.
  Formally it is defined as follows: For a document d_j and a term t_i the term frequency is

$$\mathrm{TF_{i,j}} = \frac{n_{i,j}}{\sum_k n_{i,k}}$$

  where $n_{i,j}$ is the number of occurrences of term t_i in the document d_j.

- IDF stands for *inverse document frequency*.
  Informally, the inverse document frequency will be higher the less frequent the term appears in the document collection.
  Formally it can be defined as follows: For a term t_i

$$\mathrm{IDF_i} = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

  $|D|$ is the number of document in the document collection and $|\{d : t_i \in d\}|$ is the number of documents which contain t_i (i.e. $n_{i,j} > 0$). The denominator may be changed to $1 + |\{d : t_i \in d\}|$ to avoid division by zero.

The rationale behind this measure is that if a term t_i is important to a document d (high $\mathrm{TF_i}$) and the term is not very important to the document collection ($\mathrm{DF_i}$ is low and $\mathrm{IDF_i}$ is high) then the term is of meaningful importance to the document. The use of term t_i sets the document d apart from the other documents in the document collection D. If this is the case then t_i is hopefully a good key-word/-phrase for the document.

The context of the document collection D is very important. Take the term "movie" as an example. If the collection D is a collection of movie reviews then the $\mathrm{IDF_i}$ will be low since the word "movie" will be used in a lot, if not all, documents. If on the other hand the collection consists of news articles from different areas of interest then the frequent use of the word "movie" will set a document apart from other documents in the collection.

To preprocess the documents the following step were applied to each document:

1. A part-of-speech-tagger was used to determine the part-of-speech of every word in the document. The part-of-speech of a word is its category or "word class" in grammar [3]. "Noun" and "verb" are examples of such categories. The category of a word may change with the way it is used in a given context. As an example, the word "fast" may be an adjective as in "a fast car" or an "adverb" as in "New ideas followed fast"[4].

[2] Description taken from `http://en.wikipedia.org/wiki/Tf-idf`

[3] `http://en.wikipedia.org/wiki/Part_of_speech`

[4] `http://www.thefreedictionary.com/fast`

2. The part-of-speech-tagger was also used to determine the lemma of each word. In morphology and lexicography in linguistics the lemma of a word is its canonical form[5]. As an example both *going* and *went* have the same lemma *go*; *car, cars* have the lemma *car* [6].

3. All the words that are not of the categories "Noun" or "Name" were filtered from the documents. "Names" are names of cities, countries, days and the like. "Africa", "Germany", "Paris" and "Tuesday" are all examples of "Names".

4. Only the lemma of each word was used as input to the topic generation method described below.

This removes all the words like articles [7] pronouns [8], i.e. words that appear in every text but whose appearance does not distinguish one text from another. It can be easily seen that such words do not provide good topics.

It also has the following benefit. In some cases a specific form of a word may only be used in a document collection very rarely. As an example will use "Tuesday" and "Tuesdays"[9]. To the calculation of TF × IDF these two words are entirely different terms. If "Tuesdays" is only used very rarely, opposed to "Tuesday", its IDF will be high. This in turn will make it TF × IDF-score higher. The score may be so high that "Tuesdays" may be selected as a topic. The lemmatization of each word counter-effects this.

Lemmatization also has the effect that only the lemma of a word can be extracted as a topic. This is neither good nor bad, but it will have an effect on the evaluation.

After the preprocessing the following method was used to generate keywords with TF × IDF:

1. For each term t_i that appears in the training set D_{train} calculate the inverse document frequency IDF_i of the term.

2. For each document $d \in D_{test}$ calculate the term frequency TF_j for each term t_j that appears in the document.

3. For each document $d \in D_{test}$ and each term t_i that appears in d calculate $\text{TF}_\text{i} \times \text{IDF}_\text{i}$ using the values from the previous steps.

4. For each document d use the n terms t_i with the highest $\text{TF}_\text{i} \times \text{IDF}_\text{i}$-score as the documents topic set K_d.

In this thesis 10 was used as the number of topics (n above).

This method deviates from the usual way of calculating the TF×IDF in the way that it calculates the IDF on the training set but calculates the TF on the testing set. Usually the TF × IDF is calculated using the values from the whole document collection, not splitting it in two sets. This change was introduced to match the behavior of the other two algorithms which both operate in two phases: a training and a testing phase.

Section 1.1 states that the source of the topics should not be restricted. The topic-generation method presented in this sub-section can only draw topics from document. Therefore it actually performs "keyphrase extraction". It also only uses single words as topics, thus only generating keywords not keyphrases.

But it is a very simple and efficient method for generating topics. It is performed by counting the different words of a document and doing a few simple calculations with the resulting values. It would be very hard to develop a method that is faster or simpler but still has the promise of a meaningful result. Therefore it fills the role of the lower-runtime-bound as described in table 3.1.

3.2 Machine Learning: Decision Tree Learner

Artificial Intelligence concerns itself with the design and construction of agent programs (algorithms) that *act rationally* [10]. Machine Learning is a branch of Artificial Intelligence where the behavior of the agent program evolves over time. The agent receives perceptions of its world and adjusts its behavior according to some goal. Its future decisions are influenced by the examples of the world it has seen so far. The agent is said to *learn* from the examples.

This is a very brief description of Machine Learning, but to give a comprehensive introduction to Machine Learning, not to speak of Artificial Intelligence, would be far outside the scope of this thesis. Of main interest to this sub-section is the task of *classification*.

[5] `http://en.wikipedia.org/wiki/Lemma_(linguistics)`
[6] Although the examples are in English this also applies to the German language as well.
[7] Examples: "*the* car", "*an* airplane, "*das* Auto", "*ein* Flugzeug"
[8] Examples: "*That* reminds *me* of *something*.", "*Das* erinnert *mich* an *etwas*."
[9] Example: "On *Tuesdays* he usually goes to the movies."
[10] [SRPN03, Chapter 1]

In classification the agent program gets an object, described by a set of attributes, as input. The agent then returns an output value for the object[11]. The domain of possible values is a discrete set[12]. This output value is often called the *class* of the object in literature.

The agent learns the output function from previously seen examples, i.e. objects for which the output value is known.

Recall the definitions from section 1.2. Let $K := \bigcup K_i$, i.e. K is the set of all topics in the training set T. The topic generation algorithm then trains one binary classifier C_k for each $k \in K$. A binary classifier is a classifier whose set of possible output values has exactly two elements. In out case this will be the set $\{\mathsf{true}, \mathsf{false}\}$. During the training of the classifier C_k the class of a document $t \in T$ will be true if the topic k is in the topic set K_t of the document and false otherwise.

To generate topics for a document $s \in S$ the generation algorithm lets each classifier C_k assign a class to the document s. If the class is true then the algorithm adds the topic k to the topic set K_s. Initially every set K_s is empty for each $s \in S$.

This method of generating topics is highly dependent on the training set T. It can only assign phrases that appear as topics on the training data to new documents. In the worst case, where no training data is available, this method cannot generate topics at all. This is no problem however if the training set contains a sufficient amount of data and if it is not expected that new document will require new keyphrases. On the other hand this method cannot handle the situation where new documents require the assignment of keyphrases which have not been used as topics on the training data.

The description of classification in this sub-section is a general one. Numerous algorithms exist that solve the task of classification. Different algorithms were tried in this thesis to generate topics in the fashion described further above. These algorithms vary vastly in the way they store their theories. A theory, in this context, is a set of rules that an algorithm learns. This theory can bes used to make inferences about objects in the algorithms world.

The implementation of the algorithms used in this thesis was taken from the "*Minor Third*" software library[13].

The first algorithm tried was the Naïve-Bayes classifier. Naïve-Bayes is a statistical model for predicting object class values[14]. Naïve-Bayes assumes that the object attributes are conditionally independent given the class value [15]. It predicts the class value of an object by calculating probability estimates for each class values. The most likely class value then is the class value of the object. To calculate the probability estimate for each class Naïve-Bayes uses the probability distributions of the attribute values on the training data.

In the experiment of trying to generate topics on the "Zeit.de"-dataset Naïve-Bayes did not generate any topics because the predicted class for each topic was always false. This has two possible causes:

1. The documents are objects with large sets of attributes[16]. To calculate probability estimates Naïve-Bayes has to multiply a lot of numbers which are all less than 0. This may lead to numerical instabilities in the form that the probability-score of the class true becomes 0 due to rounding in floating point arithmetics. The reason is that modern computers use floating point arithmetic which cannot store arbitrary precision numbers [17].

2. The estimated probability-score of the class true is always lower than that of the class false. This may be due to the fact that each topic only appears in the topic sets of a small fraction of documents.

Both of the causes make Naïve-Bayes not suitable for generating topics. Therefore it is irrelevant which of them is actually true.

The second algorithm tried was a Support-Vector-Machine[18]. However the experiment could not be finished. The program used did not terminate and it did not seem to make any progress, because its cpu-load was 0 for a long period of time.

The third algorithm tried was a decision tree learner. In a decision tree each node is a test on one of the objects attributes. Each outgoing edge of a node is labeled with one possible outcome of the test. The leafs of the tree are labeled with class values for the objects. Predicting the class of an object works as follows:

[11] [SRPN03, section 18.3]
[12] Learning a continuous function is called *regression*, [SRPN03, page 653]
[13] `http://minorthird.sourceforge.net/`
[14] [SRPN03, Section 13.6 & Chapter 20]
[15] For *conditional independence* see [SRPN03, page 482]
[16] Called *features* in *Minor Third*
[17] See `http://en.wikipedia.org/wiki/IEEE_754-2008` for more details
[18] See [SRPN03, Section 20.6] for more

1. Start at the root node of the tree.
2. Apply the test of the current node to the object, or more precise to the attribute named in the test of the current node.
3. Examine the outgoing edge of the current node which is labeled with result of the test. Its endpoint becomes the current node.
4. If the current node is not a leaf go to step 2.
5. If the current node is a leaf then return the label of the leaf as the predicted class value.

Decision trees represent a very efficient way of predicting class labels for objects. Only the tests on one path from the root to a leaf get executed. The other tests, which may represent a large portion of the tree are not executed at all. At worst the classification performs *n* tests, with *n* being the number of attributes on an object. It is up to the individual learning algorithms to learn trees which are smaller than the largest possible but still yield good classifiers.

Numerous algorithms exist for learning decision trees. The one used in this thesis is provided by the "`DecisionTreeLearner`"-class of the "*Minor Third*"-library. It was chosen as the Machine-Learning-algorithm to be compared to the other algorithms in this thesis. The results of the experiments on this algorithm are provided in section 4.

3.3 Maui-Indexer

The details of the Maui algorithm are very complex. Only a brief description will be given here to provide a better base for viewing the evaluation results. For more information please refer to [Mede09].

The Maui-Indexer was developed by Olena Medelyan as part of her PhD-thesis[19]. "Maui builds on the keyphrase extraction algorithm Kea (Witten *et al.*, 1999) by adopting its two-stage indexing process and inheriting some of its components."[20]. The Maui algorithm consists of four main steps[21]:

1. Generating candidate topics *(training and extraction)*
2. Computing features for the candidates *(training and extraction)*
3. Building the topic indexing model *(training only)*
4. Applying the topic indexing model *(extraction only)*

This general process can be configured to perform different tasks from section 1.1. This changes the behavior of some steps of the algorithm. In this thesis Maui was configured to perform *Indexing with Wikipedia*. This means that Maui performs *keyphrase extraction* using the Wikipedia to enhance some steps of the algorithm. The content of this subsection only describes Maui's behavior when it is configured this way.

The first step of Maui is split into the following phases:

- **Phase A** receives the document text as input. Its output is "a set of textual segments (full sentences or their parts), each being a sequence of word tokens containing at least one letter" [22].
- In **Phase B** "Maui extracts all subsequences of tokens of length n (*n*-grams)" ... "The value of n ranges between lower and upper limits defined by the user. For each n-gram Maui then determines whether it is a suitable candidate" [22]. If Maui is configured to perform *Indexing with Wikipedia*, as is the case in this thesis, candidates are identified with "Wikipedia keyphraseness value over a given threshold ..., i.e. those that are likely to appear as anchors in Wikipedia" [22].
- In **Phase C** "the n-grams are conflated to a set of candidate topics"[23]. In the case of *Indexing with Wikipedia* topics are Wikipedia articles. "Here Maui uses the *Wikipedia Miner*[24] to retrieve matching Wikipedia articles and disambiguate them to those articles that are most similar to the unambiguous context"[23].

[19] [Mede09]
[20] [Mede09, page 107]
[21] [Mede09, page 109]
[22] [Mede09, page 111]
[23] [Mede09, page 113]
[24] `http://wikipedia-miner.sourceforge.net/`

- **Phase D** normalizes the occurrence positions of the candidates by the document length and the occurrence frequencies by the number of candidates.

It is important to note that in the case of *Indexing with Wikipedia* the candidates are Wikipedia articles, or more precise their titles. This does not pose a big restriction on the possible topics. With a size of 1.1 million articles the Wikipedia covers a huge set possible topics. Further the Wikipedia has a large number of pages which only redirect to, i.e. point to, other articles which allows it to cover a lot of synonyms [25]. *Phase C* of the candidate generation step also allows Maui to deal with ambiguous candidates. If a candidate has multiple meanings only the most likely meaning, given its context in the document, will be used as topic. This may enhance the quality of the topic sets generated by Maui but poses problems to the evaluation presented in section 4.

From the description of the candidate generation it can be seen that the candidate topics are extracted from the document text. This means that Maui performs "keyphrase extraction", but enhances it with disambiguation. Maui cannot generate topics that do not appear in the document text. Due to the disambiguation the topic may not be contained in the document in the exact same form, but at least one of its synonyms or meanings has to appear in the document text for Maui to consider it as candidate.

After the candidate generation Maui computes features for each of the candidates. The following features are computed during the candidate generation step:

- *term frequency* - See section 3.1
- *first occurrence* – "the position of the first occurrence for each candidate relative to the number of words in the document" [26]
- *last occurrence* – "the position of the last occurrence for each candidate relative to the number of words in the document"[26]

In the training stage Maui also creates tables with the following values[27]:

- n_t is how many documents contain each candidate topic
- N is the total number of documents
- m_t is the frequency of a topic that appears in the manually assigned topic sets

These features and values are then used in the computation of the following features [27]:

- *inverse document frequency* - See section 3.1 of this thesis
- TF × IDF - See section 3.1 of this thesis
- *spread = last occurrence - first occurrence*
- *domain keyphraseness* is 0 if the candidate topic never appears in a manually assigned topic set and m_t otherwise
- *Wikipedia keyphraseness* involves matching candidate title against anchors appearing in the Wikipedia corpus. Values are pre-computed during candidate generation.
- *inverse Wikipedia frequency* is computed by retrieving the most likely Wikipedia article for the current candidate (unless the candidate is a Wikipedia article itself) and counting the number of its incoming links.
- *total Wikipedia keyphraseness* is the sum of *Wikipedia keyphraseness* values over all *n*-grams that were mapped to the Wikipedia article corresponding to the given candidate
- *semantic relatedness* is the total relatedness of the Wikipedia article representing the candidate to Wikipedia articles identified for all other candidates computed using Wikipedia Miner
- *term length* is the number of words in the candidate topic's name

[25] "A word having the same or nearly the same meaning as another word or other words in a language" `http://www.thefreedictionary.com/synonym`

[26] [Mede09, section 6.2]

[27] descriptions taken from [Mede09, section 6.2]

- *generality* is computed for candidates that were mapped to Wikipedia articles, and is the distance between the category corresponding to the article and the root of the category tree, normalized by the tree depth
- *class value*, which is only known for training documents, is 1 if the candidate has been assigned manually, and 0 otherwise

In the third step *Building the topic indexing model* Maui uses Machine Learning to build an indexing model. Recall that Maui can be configured to perform different indexing tasks. [Mede09, Section 5.2.6] "shows that the performance of features depends on the indexing task. Machine learning allows the algorithm to capture such dependencies."[28]

In its fourth step Maui determines for each candidate a probability of it being a candidate according to the previously built indexing model. The k candidates with highest probabilities are chosen as topics. In this thesis 10 was chosen as the number of candidates.

Table 3.2 and table 3.3 give an overview of Maui during training and extraction respectively.

Table 3.2: Maui training

1. Split the text into tokens.
2. Extract candidates from the document text.
   a) Conflate the candidates to Wikipedia articles.
   b) Disambiguate the candidates with Wikipedia miner.
3. Use Machine Learning to build a topic indexing model.

Table 3.3: Maui extraction

1. Split the text into tokens.
2. Extract candidates from the document text.
   a) Conflate the candidates to Wikipedia articles.
   b) Disambiguate the candidates with Wikipedia miner.
3. Compute probability of being a topic for each candidate according to the topic indexing model.
4. Return the 10 candidates with the highest probabilities as the topics for each document.

[28] [Mede09, page 119]

4 Evaluation

To evaluate the different topic indexing algorithms we need two things. First we need good datasets on which experiments can be run. These have been presented in section 2. For the evaluation the datasets were split into two subsets: one used for training and one used for testing the algorithm. The ratio of training to testing data was 9:1, i.e. 90% of the data were used for training and 10% were used for training. The assignment of a document to either the testing or training set was done randomly. But this assignment was not changed for the different algorithms; every algorithm should receive the same training and testing datasets. This assignment scheme could not be upheld for all the experiments. The Maui indexer cannot handle large amounts of documents. It had to be run only with a fraction of the documents from the "Reuters"-dataset. This will be noted in the appropriate section.

To reduce the variability of the experiment results a technique like *cross-validation*[1] could be used. This would mean repeating the each experiment multiple times using different random assignments of training and testing data each time. Due to the long runtime of some of the experiments (up to 24 hours) this could not be done in the scope of this bachelor thesis.

Second we need criteria on which the experiment results can be compared. These can be derived from the introduction in section 1 and can found in literature *information retrieval* and *machine learning*.

This section will first present and explain the different evaluation criteria used in this thesis and then will present and evaluate the experiment results.

4.1 Evaluation Criteria

4.1.1 Precision & Recall

Precision & Recall are two metrics which can be used to measure the performance of *information retrieval* or *pattern recognition* algorithms. In *information retrieval* the algorithm returns a result set as a response to a user supplied query. Internet search engines, like `http://www.google.com` and `http://us.yahoo.com/`, perform *information retrieval* tasks. As input they receive an user supplied query (a string of text). Given this query there are Internet web pages which *relevant* and *not relevant* to the user. The search engine is expected to return a result set which contains as much as possible of the *relevant* pages and as little as possible of the *not relevant* pages. To a search engine also the *ordering* of the result set is important, i.e. the most relevant pages should be presented first to the user. But to the definition of *precision & recall* the ordering of the result set is irrelevant.

The above is only an example of an *information retrieval* task. To define *precision & recall* we need to formally define the task of *information retrieval*:

- Q is the set of all possible queries.
- I is a set of possible items which can be returned by the *information retrieval algorithm*.
- The function A represents the *information retrieval algorithm*

$$\begin{aligned} A &: Q \to \mathscr{P}(I) \\ A &: q \mapsto R_q \end{aligned}$$

  The function returns subsets of possible items.

- For every query $q \in Q$ there are sets:
  - $I_q \subseteq I$ is the set of all items that are *relevant* to the query.
  - $\tilde{I}_q \subseteq I$ is the set of all items that are *not relevant* to the query.
  - $R_q \subseteq I$ is the set returned by the *information retrieval algorithm A*, as defined above.
- For each query $q \in Q$ and each item $i \in I$ only one of the following can be true:

$$i \in I_q \tag{1}$$
$$i \in \tilde{I}_q \tag{2}$$

  i.e. an item can only be *relevant* or *not relevant* but not both at the same time.

[1] `http://en.wikipedia.org/wiki/Cross-validation_(statistics)`

To measure the performance of the *information retrieval algorithm A* we can now use *precision & recall*. For a query q *precision & recall* are defined as follows:

- Let $C_q := \{i \in I : i \in I_q \cap R_q\}$. i.e. C_q is the set of all *relevant* items in the result set of the query q.
- The precision p is the ratio of the *number of returned relevant items* to the *total number of* ***returned*** *items*:

$$p = \frac{|C_q|}{|R_q|}$$

- The recall r is the ratio of the *number of returned relevant items* to the *total number of* ***relevant*** *items*:

$$r = \frac{|C_q|}{|I_q|}$$

- $|S|$ denotes the size of the set S.

Both measures (*precision & recall*) should always be considered together when judging the performance of an algorithm since it is easy to maximize one of them. To demonstrate this consider the two following extreme cases:

1. The results R_q returned by the algorithm A are one element sets. If $R_q = \{i\}$ and $i \in I_q$ then *precision* would be at its maximum of 1.0, because the algorithm does not return any items which are *not relevant*. But this would only yield a low *recall* with $r = 1/|I_q|$, because the algorithm only returns one relevant item.

2. The results R_q returned by the algorithm are always $R_q = I$. This would bring the *recall* to its maximum of 1.0 since all the *relevant* items are returned. But it would bring the *precision* to a value of $r = |I_q|/|I|$, because all the items which are *not relevant* are also returned.

Both of the above cases are undesirable. In the first case the algorithm can fail to return a large number of relevant information given a real world task. In the second case the algorithm does not filter out any of irrelevant information, so a user would need to perform the search himself.

The definition above defines *precision & recall* only on one query. To judge the performance of an algorithm on a set of queries we can record the *precision & recall* of each query and then calculate the average and standard deviation of theses values.

There is also a measure that combines *precision & recall* into one[2]. The Maui indexer computes this *F-measure* during its extraction phase. These values can be found in the log-files on the authors web-site[3].

4.1.2 Runtime

Runtime is a critical criterion when comparing the algorithms. It determines to which problem instances the algorithms can be applied. An algorithm that takes a long time to execute is less useful in a practical application. If a document collection changes frequently the algorithm would also have to be rerun often. So if an algorithms has a long completion time the topics for new document would not be available for a long time to users that work with the document collection.

To better evaluate the algorithms the runtime of the training and extraction phases will both be measured separately. Of these two phases the runtime of the extraction phase is the more important one. If an algorithm provides good results the training phase would not have to be rerun every time a new document enters the document collection. On the other hand the extraction phase has to be repeated for each new document to provide topics for it.

Also the runtime of the algorithms will show how they can be integrated in a more complex system. If an algorithm takes a long time to complete it can be used as a preprocessor, e.g. in a search engine. On the other hand if it only takes a very short time to complete it could also be used when a user interacts with the system.

The time measured is the actual runtime not the asymptotic time complexity $O(\cdot)$ of the algorithms. A theoretical analysis of such a complex algorithm as Maui could take a long time to complete. Also the time complexity would not answer the question if an algorithm can be used in an interactive system. The experiments were conducted on the same computer. Therefore they can be compared to one another.

The different sizes of the datasets used in this thesis also provide a clue to how each of the algorithms scales with the size of the document collection.

[2] [Mede09, section 2.2.2]

[3] `http://viktor-s.net/BachelorThesis`

4.1.3 Memory use

This criterion will measure how much memory each algorithms uses. The measure will be split in two:

1. **Runtime** memory use is how much working memory an algorithm uses during its execution.
2. **Persistent** memory use is how much persistent storage memory, like that of a hard drive, the algorithm uses between executions. This is needed to store the results of the training phase from each algorithm.

This criterion is interesting because it shows which hardware is needed to run the algorithms. Also if an algorithm using a lot of working memory may have a significant impact on the algorithms runtime. A large memory consumption increases the amount of transfer done from disk to memory done by the operating system. This in turn means that the executing process of the algorithm will be idle for long periods of time waiting for the hard drive to finish its operations.

4.1.4 Quality of Topics

This criterion is the most important one and at the same time it is the most hard to measure of all the ones presented in this section. It is clear that we would like an algorithm to generate "good" topics, topics that provide a good summary of a documents contents. However it is hard to describe what constitutes a "good" topic. We can use theoretical measures to describe the quality of topics or topic sets. But these do not always reflect on the "goodness" of a topic. The *precision & recall* measure provided in this section tries to measure the quality of topic set by comparing the extracted set to the manually assigned one. But TF × IDF and Maui do not perform *keyphrase indexing* but *keyphrase extraction*. Hence it can be expected that their *precision & recall* values will lower because they cannot generate all the topics from the manually assigned topic sets. But this may not mean that they produce poor results.

However [Mede09, chapter 4] shows that the results of human indexers can be inconsistent. This means that what constitutes a good topics may vary from person to person. It would be therefore advisable to conduct a test asking people to rate the results of the algorithms and evaluate the ratings using statistics. But to have a statistically significant result one would have to ask many people to partake in the test. This however would be outside the scope of this thesis. In turn this means that only the authors opinion can be provided in this thesis about the quality of the generated topics.

4.2 Experiment Results

4.2.1 Precision & Recall

Table 4.1: Experiment results: Precision & Recall

	Zeit.de	**Reuters**	**Wikipedia**
TF × IDF	0/0	0/0	0/0
	0.04/0.19	0/0	0/0
Decision Tree	0.23/0.24	0.89/0.32	0.27/0.44
	0.04/0.21	0.77/0.42	0/0
Maui	0.07/0.08	0/0	-
	0.18/0.20	0/0	-

The upper value is *precision*, the lower is *recall*
value/standard deviance

Table 4.1 shows the results of running the algorithms on the various datasets. For each algorithm each the table contains two rows: the upper one shows *precision*, the lower shows *recall*. The first value is the averaged value over all documents. The second value is the standard deviation. The values of the Maui indexer were taken directly from its output.

Values of 0 in the table do not mean that the actual values are really zero, only that they were rounded down to zero by the averaging. But for this to happen a lot of the *precision & recall* values for the documents had to be 0.

This happened a lot with TF × IDF. On a majority of document the topic set generated by TF × IDF the *precision & recall* values were both 0. Only on a few documents did TF × IDF generate a topic set that had *precision & recall* of about 0.1. From these values it is clear that performance of TF × IDF, regarding *precision & recall*, is the poorest of the 3 algorithms. Not only that but also the values are only slightly above the minimum possible.

The values of the decision tree learner are clearly the best in the field. Its *precision* values are about 25% on the "Zeit.de" and "Wikipedia" datasets, which would be sufficient for a real world application, although its *recall* values are

relatively low. On the "Reuters" dataset the decision tree learner shows the highest values of all the experiments. They are well above the 75% mark. This is due to the structure of the dataset. The documents in the dataset are very short resulting in examples that have fewer features than documents from the other datasets. Also the dataset only contains relatively few topics which do not overlap thematically. This means that the algorithms has to learn clear cut associations from a few features to one class label, a task for which it has been designed.

All this makes the decision tree learner a good choice for document collections with short documents or when the number of topics is limited.

From table 4.1 we can see that Maui's *precision & recall* values are low. On the "Reuters" dataset they get rounded down to 0. Only on the "Zeit.de" dataset are they above zero. Also they do not reach the values of the decision tree algorithm. With a precision of only 7% Maui would also not be suited for a real world application.

But these values do not give appropriate credit to Maui's results. From samples of the extracted topic sets it can be seen that Maui generates good topics. More on this will be said in section 4.2.4.

4.2.2 Runtime

Table 4.2: Experiment results: Runtime

	Zeit.de	Reuters	Wikipedia
TF × IDF	00:00:07	00:00:17	00:00:10
	00:00:14	00:00:16	00:00:29
Decision Tree	01:53:19	02:05:27	21:04:47
	00:00:06	00:00:14	00:00:38
Maui	01:37:21	02:07:26	24:24:36
	00:14:03	00:15:14	00:23:42

Times are in *hh:mm:ss*.

Table 4.2 shows the runtimes of running the 3 algorithms on the different datasets. For each algorithms the table contains a row: the upper one shows the runtime of the training phase, the lower shows the runtime of the extraction phase. The times are given in hours, minutes and seconds.

From the table we can see that TF × IDF is the fastest algorithm in the field. Both of its phases only take a few seconds to complete. It fills the role of being a runtime-benchmark for the other two algorithms perfectly.

The training times of both Maui and the decision tree learner look like they are close together. But for Maui the amount of documents in the "Reuters" dataset had to be reduced because the indexer was taking too long to complete on the full dataset. After more than two hundred hours only about a quarter of the documents were processed in the training phase. Maui was rerun on ten thousand documents from the "Reuters" dataset. Therefore the runtime of Maui on the "Reuters" is highlighted in the table.

The high times of the training phases make both Maui and the decision tree learner unsuitable in an interactive application. Their training phases at least take one and a half hours to complete; and in the case of the "Wikipedia" dataset they both take about a day to complete. But they could be used in a preprocessing stage of a system.

Looking at the extraction times we see that the decision tree learner is the fastest in the field. It even outperforms TF × IDF. This is due to the fact that decision trees are very fast to evaluate. In any given evaluation only one path from the tree's root to one of its leaves has to be evaluated. This yields in a very short extraction time for the algorithm and also makes its extraction phase suitable for usage in an interactive application.

Maui's extraction phase takes a much longer time to complete. The fastest completion time was about 14 minutes. But is still reasonably fast to be used as preprocessor in a more complex system.

From the above one could deduce that Maui is not suited to be used on large document sets. The experiment of running Maui on the full "Reuters" dataset had to be aborted due to the long runtime. But Maui has one advantage: it only needs a small number of documents in the training phase. One experiment was conducted where the amount of training to testing data was reversed. I.e. 10% of the data from the "Zeit.de" dataset were used for training and 90% were used for testing. This did not deteriorate the results of Maui's extraction; neither the *precision & recall* values nor the quality keywords. This means if Maui were to be in a real world application it should only be trained on small portion of the available documents, considerably reducing the runtime of the training phase. Or it could be used in scenario where only a small number of indexed documents exist.

4.2.3 Memory use

The persistent memory use of all algorithms is only a few megabytes. Any modern computer should be able to store that much data without any problems. Also the differences between the algorithms do not result in any bottlenecks when reading the data from the persistent storage. Therefore the actual numbers are not relevant for the evaluation, since the persistent storage of the algorithms does not pose any problems.

On the hand the runtime memory consumption is worth mentioning. All the algorithms load the entire dataset into memory. This is not a problem since the datasets are only collections of text files. In the case of "Wikipedia" this would be 800 megabytes of memory. Again any modern computer should be able to handle this.

Beyond that TF × IDF only stores a table which contains a mapping from word to their IDF, i.e. a mapping from strings to numbers. This uses up a few kilobytes at most. It also is smaller than the dataset itself and does not pose any problems.

The decision tree learner uses up to 150 megabytes, additionally to the dataset, of memory during its execution. This is significantly larger than the memory consumption of TF × IDF but still is small enough to not pose any problems for modern computers.

Opposed to that Maui has an immense memory consumption. After processing about 23 thousand document on the "Wikipedia" dataset it used up about 22 gigabytes of memory. This is a multiple of the dataset itself. Also this may result in a lot of memory operations being performed making the executing process idle for long periods of time. All this makes Maui unsuitable for large amount of training data. Page 15 shows a way how this problem may be circumvented in an application.

4.2.4 Quality of Topics

Of all the evaluation criteria this one is the hardest to judge. [Mede09, chapter 4] shows that if humans are tasked with indexing documents the results can be highly inconsistent. The causes for this lie in the field of psychology. Different humans to tend think on different levels. This highly influences the way that humans perform association tasks, i.e. associating different objects to one another. Assigning topics to a document can be seen as such a task. But if assigning topics to a document is inconsistent because subjective, than judging the quality already assigned topics is too.

Nevertheless the quality of the topics an algorithm generates is an important criterion when judging the performance of an indexing algorithm. Low values in *precision & recall* may only mean that an algorithm is only inconsistent with the human indexer who manually assigned topics to the document. But it may not mean that the quality of the topics is bad, i.e. that they are not usable in a complex system.

Therefore this section provides the authors opinion on the quality of the topics generated by the algorithms. The author tried to be objectively as possible, but being a human, the opinion can be highly inconsistent with that of other humans. To actually provide an objective result would have meant to conduct a test with many participating people and evaluate it using statistical methods; which would have been to time consuming for a bachelor thesis.

The worst performance with respect to quality of topics is that of TF × IDF. On all of the datasets the topics generated by the algorithm are mostly unusable. This is mainly due to the fact that a lot of the time the algorithm extract a lot of word that are too general to be a good topic. "Tuesday"[4] is a prime example of such a topic. Especially in the light that it was extracted on a news article. The name of a weekday does not provide any meaningful information about a document to reader. This is symptomatic for the topics generated by the algorithm: words that are used fairly often in a document but are to general to have a meaning. This happened more often on the "Reuters" dataset because the documents are very short and words may get a high TF.

The quality of the topics generated by the decision tree learner depends on the dataset. The algorithm achieved high values of *precision & recall* on that dataset. Since the choice of keywords on that dataset is highly limited and the manually assigned topics are themselves of good quality, this in turn means that the quality of the topics the algorithm generated for that dataset is also good. On the other datasets the algorithm did not perform that good. Although the decision tree learner did not perform as poor as TF × IDF it sometimes produced poor topic sets.

The quality of the topics generated by Maui for the "Reuters" dataset was not good. On almost all document the indexer generated topic set that contained only 3 or less topics. Sometimes the indexer did not generate any keyword for a document. One observation was made during the execution of Maui: when using machine learning as described in 3.3, almost all class labels were set to "false". This may have resulted in the generation of the small topic sets. The most likely cause for this is the small size of the documents in the "Reuters" dataset.

On the other two datasets, which contain longer documents, the quality of Maui's topics was excellent. All the topics generated were meaningful to the document and the topic sets gave a good overview about the documents contents.

[4] The actual word was "Dienstag" since all the documents were in German

Only a few of topics were not fitting, most likely the result of an disambiguation mistake made by the algorithm. But this was rather an exception than a rule; in most instances Maui's disambiguation found the correct meaning of a keyphrase for a given document.

One example of the quality is an article from the "Zeit.de" dataset. The article was about a movie premiere. The movie was made by a German director and was shot in Asia. From the topic set extracted by Maui it was easy to see that:

- the article was about a movie
- the article was about a premiere (disambiguating it from the "Premiere AG"[5])
- the article was about an Asian country
- the article was about a city
- the city was a the capitol of that country, a true fact
- that the movie by German director

The topic set of the other documents were of similar quality.

This shows that the low *precision & recall values* of Maui do correctly reflect on the algorithms performance. Also the quality of the topics generated by Maui did not diminish when the number of training documents was reduced from 90% to 10% of the total dataset.

[5] `http://de.wikipedia.org/wiki/Sky_Deutschland`

5 Summary - Conclusion

The introduction of this thesis shows what the task of topic indexing means: Assigning topic set to set of documents, optionally using training information from manually assigned topic sets. It also shows an example how topic indexing algorithms can be used in a more complex system: enhancing the search results of a search engine.

To compare the topic indexing algorithms datasets are needed. Datasets that are document collections with manually assigned topic sets. Such dataset are rare. Three datasets were used to test the algorithms:

- The "Zeit.de" dataset, a collection of news articles from `www.zeit.de` compiled by the author.
- The "Reuters" dataset, a collection of short news items from the "Reuters" news agency.
- The "Wikipedia" dataset, which is subset of the on-line encyclopedia "Wikipedia"

The properties of these datasets have a high impact on the task of topic indexing. The "Reuters" dataset poses many problems to the algorithms because it only contains short documents. But it makes it easier for the decision tree learner to learn the assignments from the dataset because it restricts the choice of topics to well defined set. The "Zeit.de" brings problems of its own. The choice of topics was not restricted which resulted in noise on the data which had to be reduced in a preprocessing step.

This thesis presents different criteria which are used to compare the different indexing algorithms:

- *Precision & Recall* is a measure from *Information Retrieval*. It is an objective measure which can be used to compare the algorithms.
- The runtime and memory use are important criteria which determine how or if an algorithm may be used in a complex system.
- The quality of the keywords is the most important but also the hardest to judge criterion presented in this thesis. It is important because theoretical measures do not alway correctly reflect on the performance of an algorithm. It is also hard to judge because it is highly subjective and results between humans may be inconsistent.

All these criteria factor into the evaluation of the three algorithms:

- TF × IDF is measure which can be used to extract topics from a document. It is the fastest algorithm presented in this thesis. The combined execution time of both its phases is only a few seconds; far below the execution time of the other algorithms. But the quality of the topics generated by TF × IDF is poor. The topic sets are not usable in other applications. Therefore the algorithm can only server as lower bound benchmark to which other algorithms can be compared.
- A decision tree learner can be used to learn decision trees which can be used to assign a topic to a document. By using all learned trees to assign or not assign individual topics the algorithm can used to generate topic sets. The training phase of this algorithm is far slower than that of TF × IDF, taking hours to run. But its extraction phase is the fastest in the field, due to the nature of the decision trees, even surpassing the fast runtime of the TF × IDF extraction phase. The quality of its topics is dependent on the dataset. With only a well defined set of possible topics, as is the case with the "Reuters" dataset, the results of this algorithm are excellent.
- The Maui Indexer is the slowest algorithm presented in this thesis. This is probably due to the complexity of the algorithm. Also the indexer cannot be used to train on large document collections, because of its high memory consumption and long runtime. But Maui outweighs this by generating topics sets of excellent quality. Although the "Reuters" dataset with its short documents poses a problem for Maui it produces topic sets that could be used in other applications on the other two datasets. The Maui Indexer also only needs a small amount of documents to train on. This makes it usable in cases where only a small number of documents with manually assigned topics exist. Also this can be used to reduce the runtime of the algorithm by only training on a subset of the available documents.

References

[Mede09] Olena Medelyan, *Human-competitive automated topic indexing*, The University of Waikato, Department of Computer Science, July 2009

[SRPN03] Stuart Russel and Peter Norvig, *Artificial Intelligence A Modern Approach* Second Edition, Prentice Hall, Pearson Education Inc., 2003

[Witt99] Witten, I. H., G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. (1999). Kea: Practical automatic keyphrase extraction. In Proc. ACM Conf. on Digital Libraries, Berkeley, CA, US. New York, NY: ACM Press, pp. 254–255.