
ALWIS

A Visualization Tool for Concept Based Retrieval Schemes
Theoretical Foundations and Models

Bachelor thesis

Benedikt Grundmann

10th March 2006



max planck institut
informatik

Max Planck Institut Informatik, Saarbrücken
Abteilung 1, Algorithms and Complexity

Betreuer: Holger Bast

Abstract

Comparing and evaluating the performance of concept based retrieval schemes is notoriously difficult and of increasing practical importance. To aid the evaluation process we developed a visualization tool which puts particular emphasis on the comparison of single queries. This tool directly visualizes a generic theoretical model for concept based retrieval schemes which is presented in this paper. We show how concept based retrieval schemes and even clustering algorithms can be cast into it.

Contents

1	Introduction	1
2	Results	3
3	The vector space model	5
4	The Generic Model (GM)	7
4.1	Overview	7
4.2	The generated clustering	8
4.3	Queries in the GM	8
4.3.1	Notation	8
4.3.2	Queries as documents	9
4.3.3	The queries	9
5	Analysis of existing retrieval schemes	11
5.1	LSI	11
5.1.1	LSI and the GM	12
5.2	PLSI	12
5.2.1	PLSI and the GM	13
5.2.2	Generalization of folding in	14
5.2.3	Combined Queries	14
5.3	NMF	15
5.3.1	NMF and the GM	16
5.4	Spectral Clustering	16
5.4.1	Spectral clustering and the GM	17
5.4.2	Deriving pwz from pzd	17
5.4.3	Executing queries	18
6	Measurement in the GM	19
7	On the creation of the DTM	21
7.1	Stemming	21
7.2	Stop word elimination	22
8	Conclusion and future work	23
	Acknowledgments	25
	List of figures, tables and algorithms	27

Chapter 1

Introduction

A standard problem in computer science is the so called retrieval problem: How to search for a set of relevant documents given a query (that is a set of terms). Most of the time one is not actually interested in all relevant documents but just the “most relevant” document(s). Therefore the result should also include a number denoting its relevance. Which can then be used to sort the resulting set. This resulting sequence is also known as a ranking.

The most obvious approach to generate such a ranking is just to search for documents containing the terms mentioned in the query. The relevance of a document is then directly proportional to the number of query terms which are actually contained in the document.

This can be implemented very easily by an index, denoting which documents a term is contained in. Unfortunately this approach cannot cope with two phenomena, which occur very often in natural language:

- Polysems, that is words which have more than one meaning. For instance the word trunk has several meanings: a suitcase, the trunk of an elephant, the torso as well as a bole.
- Synonyms, that is several words which have the same meaning. For example the words: performer, entertainer, artist and player can all be used to denote the concept of an actor.

These phenomena can have a severe impact on the performance of a retrieval algorithm. The users of such an algorithm are normally not interested in specific words but in their meaning. But as seen above a word might actually have several meanings and there might also be several words for the same meaning. Therefore we need a way to identify or at least guess / approximate the meaning of both queries and documents.

One popular class of algorithms used to solve this problem are the so called concept based retrieval schemes. They introduce a new way to describe documents, in which documents are no longer described by the terms they contain, but rather by the concepts they belong to.

In general these algorithms consist of two phases. In the first phase the concepts contained in the corpus (the set of known documents) are identified. Intuitively a *concept* can be understood as a topic common to several documents in the corpus. The documents are then recast into a so called *concept space* representation. In the common *term space* representation a document is equal to its terms – each term weighted by its number of occurrences. In concept space each document is equal to the concepts “contained” in it, each concept weighted according to its relevance. Thereby we have now created a fuzzy clustering of the documents into their concepts. Note that these algorithms perform unsupervised clustering, in particular they do *not* need a labeled corpus.

In the second (query) phase the query is cast into concept space as well. The ranking can then be created by a simple comparison in concept space of the “query document” with all other documents.

Typically the performance of a retrieval scheme is measured by comparing the results of several queries with predetermined optimal results. The resulting average performance gives a good hint on the overall performance of an algorithm. But to understand why an algorithm performs well (or not) for certain queries it is necessary to actually look at the result of the query itself and compare it to the results of other algorithms on the same query. Ideally one would be able to compare the query and its results in term as well as in concept space.

Structure of the paper

First we will give a short overview of the results of our work. After a short overview of the purpose of clustering algorithms and retrieval schemes and the introduction of the basic concepts we present our generic model. We then present several existing retrieval schemes and show how they can be recast into the generic model.

Chapter 2

Results

Therefore we developed a visualization tool which allows us to compare different concept based retrieval schemes. Unlike previous attempts we turned our attention on comparing the results of a *single, specific* query on several different clusterings of the same corpus directly.

The notion of a concept is made explicit in our visualization tool. Therefore we can display an approximation of the concept space introduced by a concept based retrieval scheme in our visualization tool.

In preparation of our visualization tool we introduced the following theoretical foundations:

- We developed a generic model which unifies the different concept based retrieval schemes. Furthermore we generalized queries. Thereby we allow *nine* kinds of queries instead of only the classic one which retrieves the most relevant documents for a set of terms.
- We showed how to map concept based retrieval schemes into our generic model. We provide mappings for LSI, PLSI and NMF. Similar algorithms can easily be mapped in an analogous way.
- By the example of Spectral Clustering we showed that it is even possible to use plain clustering algorithms – which do not allow querying by themselves – for information retrieval.

We developed a suite of applications which is called ALWIS. As mentioned before concept based retrieval schemes are two phased algorithms. Our visualization tool naturally only visualizes the query phase. Nevertheless we also provided all necessary tools needed in the first phase. In particular we provide tools to prepare a corpus. All in all it consists of the following components:

- We provide five tools for corpus processing. This includes e.g. document filtering and stemming. Stream processing via Unix pipes is supported where appropriate. Altogether these tools consist of roughly 1900 lines of code.
- Three tools for calculating models on the basis of corpora are provided as well – for LSI, PLSI and Spectral Clustering. In total there are roughly 3100 lines of code for this tools.
- Query visualization is done in an intuitive graphical user interface which allows mouse control as well as keyboard control. It took four prototypes to reach this level of usability. The GUI consists of roughly 19300 lines of code.
- The graphical user engine uses two supplementary tools: the query engine and the precision recall measurement device. Both together consist of roughly 1700 lines of code.
- The usage of the applications as well as technical details like file formats and protocols are described in the reference manual. The manual is provided in HTML and PDF format and consists of roughly 55 pages.

One important issue in the development of the ALWIS suite was making the applications cross-platform. Especially process creation and inter-process communication with Unix pipes in the presence of a graphical user interface were problematic in this context. So far the applications have been tested on Linux and on Windows. Porting to other POSIX compliant platforms supported by the `wxWidgets` GUI library should be very easy. For Windows we provide a precompiled binary package coming with a full featured installer.

Due to the size of this project the work was split between my colleague Daniel Fischer [4] and myself. In this paper I present the results of my part which focuses on the theoretical generic model (GM).

Chapter 3

The vector space model

Before we present our generic model let us first introduce the notion of a document term matrix (DTM) and the underlying vector space model.

Each document is represented by a vector. The vector has the same number of elements as there are distinct terms in the corpus. Therefore a term can be represented by a number $t \in T$ denoting the corresponding element in such a vector. T is the set of all term ids ranging from 1 to the number of distinct terms in the corpus.

In an analogous way the complete document collection (the corpus) can be represented by a matrix. This matrix is known as the document term matrix (DTM) and can be created by stacking the document vectors row by row. This also defines a document id $d \in D$ as the number of the corresponding row in the DTM. Whereby D is the set of all document ids ranging from 1 to the number of documents in the corpus.

It is sometimes necessary to view the document term matrix as a probability distribution over the terms. In that case we use the following notation $P(d, t) = \frac{dtm_{dt}}{\sum_{t \in T} dtm_{dt}}$.

Note that in this representation the positions of terms within a document is not modeled. Actually the position of a term in a document is never needed in any of the algorithms we studied so this is not a loss.

In analogous way we represent each concept by a number $c \in C$ ranging from 1 to the number of concepts.

Chapter 4

The Generic Model (GM)

4.1 Overview

Our approach to a generic model for comparing concept based retrieval schemes can be summarized as follows:

- We operate on the vector space model as outlined above. The corpus itself is represented by a document term matrix. The position of the terms in the documents is not taken into account!
- Concept based retrieval schemes are two phased algorithms:
 1. A clustering phase which introduces a new concept space. It computes a matrix representing the transformation from document space to concept space and vice versa. This matrix is known as the **pzd** matrix. It also provides a transformation from term space to concept space and vice versa in the **pwz** matrix.
 2. A query phase, which operates on the matrices generated above or some matrix derived from them. Also depending on the retrieval scheme being modeled performing a query might involve performing more than one query in the generic model.
- Queries are generalized. We do not only allow queries from a set of terms to a document ranking, but also the inverse query and any other query from either documents, terms or concepts to one of the two remaining entities.

4.2 The generated clustering

Multiplication of the matrices yields a new matrix $\mathbf{pwd} = \mathbf{pwz} \cdot \mathbf{pzd}$. This matrix has the same dimensions as our document term matrix. In fact it should approximate the document term matrix. An intuitive way to think of this matrix is as an idealized corpus where gaps are filled out and outliers are removed. That is terms which are “missing” in certain documents are added and terms which “should not be there” are removed.

In section 4.3.3 we will explain how queries can be performed by comparing each row of one of the matrices $\mathbf{pwd}, \mathbf{pwz}, \mathbf{pzd}$ or their transposed with a query. This only works if both the query (considered as a document – see section 4.3.2) and the rows are normalized by the same norm. We normalize each vector to element sum 1. This gives us a probability distribution over the elements. Of course this only works if all elements are non-negative prior to the normalization.

Therefore we require that all elements of the matrices $\mathbf{pzd} \in \mathbb{R}^{\#C \times \#D}$ and $\mathbf{pwz} \in \mathbb{R}^{\#T \times \#C}$ are non-negative.

Note that only the three matrices $\mathbf{pzd}, \mathbf{pwz}$ and \mathbf{dtm} have to be stored on disk to store a generic model. Thereby the computation of a model and its use can be separated.

We also provide a generic way to compute a \mathbf{pwz} matrix given the document term matrix and the \mathbf{pzd} matrix (see section 5.4.2).

4.3 Queries in the GM

4.3.1 Notation

To clarify the different kinds of queries we are talking about we introduce some further notation and terminology:

- $q \in Q(X)$ is a query over X . $Q(X) \subset \mathcal{P}(X \times [0, 1])$ A query is a set of tuples of an id and a associated weight.
- $r \in R(X) = Q(X)$ is the type of a query result. A query result obviously has the same type as a query.
- $f \in F(X, Y) = Q(T) \rightarrow R(C)$ is a query function. When a query is executed to obtain a query result of appropriate type, a query function has to be applied to the query. This query function determines the type of the result.

As an clarifying example note that in above notation if we ask for “the most relevant documents r for a given set of terms q ” then

- $r \in R(D)$ is the query result we are searching for.
- $q \in Q(T)$ is the query itself.
- to actually execute the query we need a query function of type $F(T, D)$, which we can apply to q to yield r .

In this document we will talk about the different query functions a lot more than we will talk about queries. Therefore – if it can be done without arising confusion – we abbreviate “query function” by “query”.

4.3.2 Queries as documents

Notice that both the queries and the query results can also be represented by a vector of size $\#X$. The vector $v \in [0, 1]^{\#X}$ represents the query $q \in Q(X)$ iff $\forall i \in X$:

$$\{v_i\} = \begin{cases} \{w|(i, w) \in q\} & \text{if } i \in \{j|(j, w') \in q\} \\ \{0\} & \text{otherwise} \end{cases}$$

This implies that a term query of type $Q(T)$ can also be interpreted as a document; and a query function $F(T, D)$ can be realized by searching for similar documents.

4.3.3 The queries

Besides the obvious query function $F(T, D)$ several others are of interest, when trying to gauge the performance of a given clustering:

- $F(T, C)$: Obviously a very important query as it gives us a hint why a given retrieval scheme ranked a document as important even if the search term(s) are not contained in the document.
- $F(C, T)$. The inverse query is the best tool in understanding what a given concept is about. It intuitively answers the question “What does a document representing this concept look like?”. And therefore the best way to visualize a concept is displaying the first k terms returned when such a query function is applied.
- $F(C, D)$. This query complements the one above by answering the question “What *existing* documents in our corpus represent the combination of these concepts?”. Again its probably most useful when used with just

query function	matrix
$F(T, D)$	pwd
$F(T, C)$	pwz
$F(D, C)$	pzd^T
$F(D, T)$	pwd^T
$F(C, D)$	pzd
$F(C, T)$	pwz^T

Table 4.1: Matrix-query relationship

a single concept: “What *existing* documents in our corpus represent this concept?”.

- $F(D, C)$ “What concepts are common to these documents?”
- $F(D, T)$. When used with just a single document this query answers the question “How does this document should look like?” Interestingly this seems to be the least useful query. It would in theory be interesting to compare this idealized document to its original form in the DTM but due to the size of a document this is not really feasible in general.

Each of these query functions can easily be done on our GM. Taking the result of section 4.3.2 into account, we just need to compare the “query document” with the rows of the corresponding matrix.

For example to find the most relevant documents for a set of terms ($F(T, D)$) we compare the query with each row of the matrix **pwd**. The inverse query function $F(D, T)$ can be implemented by comparison with each row of the transposed matrix **pwd^T**. In an analogous way to find the most relevant concepts for a set of weighted terms ($F(T, C)$), we need a matrix which expresses the relationship between the documents and the terms, in a similar way the **pwd** does for terms and documents. We already have such a matrix! It is the matrix **pwz**.

In general the algorithm to express a query in the GM is listed as algorithm 4.2. Which matrix denotes which query function is listed in table 4.1.

```

M the matrix according to table 4.1
q ∈ Q(X)(normalized) ∧ r ∈ R(Y)
r ← ∅
for x ∈ X do
  v ← row x of M(normalized)
  r ← r ∪ {(d, compare(q, v))}
end for
return r

```

Algorithm 4.2: Generic query $F(X, Y)$ in the GM

Chapter 5

Analysis of existing retrieval schemes

In order to create unified model of concept based retrieval schemes we first analyzed several, namely:

- Latent Semantic Indexing (LSI)
- Probabilistic Latent Semantic Indexing (PLSI)
- Spectral Clustering
- Non-Negative Matrix Factorization (NMF)

5.1 LSI

Latent Semantic Indexing as described in the paper [3] performs a singular value decomposition (SVD) on the document term matrix. The SVD computes a factorization $\text{dtm} = U \cdot \Sigma \cdot V^T$. Σ is a diagonal matrix of singular values. This matrix and the matrices U and V are now reduced in size by removing all but the k largest singular values and the corresponding rows and columns of U and V . The multiplication of the resulting matrices U' and V' yields a approximation of the original DTM.

An appropriate way to realize the query function $F(T, D)$ is to map the query into feature space using the matrix U and perform the comparison in concept space, i.e. the documents have to be mapped as well. But as noted in [1] it is also possible to do the cosine comparison of the query document on the approximated DTM $U \cdot V$, i.e in term space.

5.1.1 LSI and the GM

The matrices U and V resemble the matrices \mathbf{pwz} and \mathbf{pzd} with the exception that they might contain negative values. Therefore the matrices have to be postprocessed. Several different postprocessing strategies are possible:

truncating Every negative element of the matrices is set to zero.

$$m_{ij} = \begin{cases} 0 & \text{if } m_{ij} < 0 \\ m_{ij} & \text{otherwise} \end{cases}$$

absolute value The absolute value of every element is used.

$$m_{ij} = |m_{ij}|$$

shifting If there is at least one negative element the absolute value of the smallest element is added to each element of the matrix.

$$m_{ij} = |\min m_{ij}| + m_{ij}$$

Following the line of reasoning in section 4.2, clamping seems to be the most reasonable strategy to use. Shifting would change the meaning of zero elements, which is in general not a desirable property. The absolute value strategy on the other hand would make negative values important and hence seems not to be the correct strategy.

5.2 PLSI

Probabilistic Semantic Indexing as described in the paper [6] is an iterative algorithm and was designed as an improved successor of LSI. The values in the model computed by PLSI can be interpreted as probability distributions. To compute the clustering the two matrices \mathbf{pwz} and \mathbf{pzd} are randomly initialized. The formulae (5.1), (5.2) and (5.3) are then repeatedly applied to the matrices.

$$P(c|d, t) = \frac{\mathbf{pwz}_{tc} \mathbf{pzd}_{cd}}{\sum_{l=1}^K \mathbf{pwz}_{tl} \mathbf{pzd}_{ld}} \quad (5.1)$$

$$\mathbf{pzd}_{cd} = \frac{\sum_{t \in T} \mathbf{dtm}_{dt} P(c|d, t)}{\sum_{t \in T} \mathbf{dtm}_{dt}} \quad (5.2)$$

$$\mathbf{pwz}_{tc} = \frac{\sum_{d \in D} \mathbf{dtm}_{dt} P(c|d, t)}{\sum_{t \in T} \sum_{d \in D} \mathbf{dtm}_{dt} P(c|d, t)} \quad (5.3)$$

The resulting matrices do represent the transformations as required by the generic model.

In the search phase the query is mapped into concept space using a variation of the iterative algorithm seen above and the query is compared to the documents in concept space.

5.2.1 PLSI and the GM

It is immediately obvious that the result of the clustering part of PLSI has a direct representation in our generic model.

The two steps implicitly done by the query $F(T, D)$ in PLSI are actually made explicit in our generic model.

1. The term vector has to be transformed into concept space. This process is also known as folding in. The actual algorithm to do so is listed in algorithm 5.1.
2. The resulting vector has to be compared to each row of the matrix \mathbf{pzd} using the standard cosine comparison method as listed in algorithm 4.2.

```

 $\mathbf{pzq} \in [0, 1]^C$ 
 $\mathbf{pzqw} \in [0, 1]^{C \times T}$ 
for  $n \leftarrow 1 \dots N$  do
  for  $c \in C$  do
    for  $t \in T$  do
       $\mathbf{pzqw}_{c,t} \leftarrow \frac{(\mathbf{pwz}_{t,c} \mathbf{pzq}_c)^\beta}{\sum_{l \in C} (\mathbf{pwz}_{t,l} \mathbf{pzq}_l)^\beta}$ 
    end for
  end for
  for  $c \in C$  do
     $\mathbf{pzq}_c \leftarrow \sum_{t \in T} q_t \mathbf{pzqw}_{c,t}$ 
  end for
end for
return  $\mathbf{pzq}$ 

```

Algorithm 5.1: Folding a term query $Q(T)$ into concept space in PLSI

As step two is already provided by the GM, all we need is a way to incorporate algorithm 5.1 in the GM. The key observation to do so is the fact that this folding-in operation has the same type as a query $F(T, C)$. By extending the set of supported query functions, we can actually provide the query function $F(T, D)$ as required by PLSI.

5.2.2 Generalization of folding in

Now that we model folding in as a query of its own kind why should we not generalize it, in the same way we generalized the cosine based query functions in section 4.3.3? In studying algorithm 5.1 we realized that its functionality depends on a single matrix \mathbf{pwz} . And coincidentally we can replace this matrix by any of the other matrices with the expected semantics. That is if we replace the matrix \mathbf{pwz} by the matrix \mathbf{pwd} we get a mapping from term space into document space. Besides the matrix replacement we have to adapt the dimensions of the temporary matrices and a few running counters – for details refer to algorithm 5.2.

```

pyq  $\in [0, 1]^Y$ 
pyqx  $\in [0, 1]^{Y \times X}$ 
for  $n \leftarrow 1 \dots N$  do
  for  $y \in Y$  do
    for  $x \in X$  do
      pyqx $_{y,x} \leftarrow \frac{(M_{xy} \mathbf{pyq}_y)^\beta}{\sum_{t \in Y} (M_{xt} \mathbf{pyq}_t)^\beta}$ 
    end for
  end for
  for  $y \in Y$  do
    pyq $_y \leftarrow \sum_{x \in X} q_x \mathbf{pyq}_{y,x}$ 
  end for
end for
return pyq

```

Algorithm 5.2: Folding a query $Q(X)$ into another space Y in PLSI

As shown in [2] it can be advantageous to combine advanced folding in mechanism with simple clustering methods. Our GM might be helpful for future research in that area. One can easily use PLSI style query functions on matrices generated by a another clustering algorithm such as LSI. This might give an further insights whether this method or the clustering itself or both are the strength of PLSI. With the same intent one can use only the cosine query functions on the matrices created by the PLSI clustering algorithm. And whenever a new algorithm is introduced to our GM the same reasoning can be applied. We believe that this might turn out to be a very helpful contributions of the GM in understanding and evaluating the performance of concept based retrieval schemes.

5.2.3 Combined Queries

PLSI allows one additional query function. Given a term id t and a document id d it computes a concept ranking. Unlike the usual queries $F(T, C)$ and $F(T, D)$ this query computes a combined ranking, where each concept is weighted by $P(c|d, w)$.

This query can be used to differentiate between the meanings of polysems. Intuitively it answers the question: “Which concepts represent the meaning of the term t in document d ?”.

Unfortunately many algorithms do not provide a specific way to do this query. Therefore we do not support this specific algorithm in the GM. Nevertheless we believe that the algorithm 5.3 can be used to express those kinds of queries in a general way.

$$\begin{aligned} f_d &\in F(D, C) \\ f_t &\in F(T, C) \\ r_d &\leftarrow f_d(\{(d, 1)\}) \\ r_t &\leftarrow f_t(\{(t, 1)\}) \\ r &\leftarrow \text{merge}(r_d, r_t) \end{aligned}$$

Algorithm 5.3: Query for concepts given document and term

The basic idea behind this algorithm is very simple. We first obtain the most relevant concepts for the given term(s) and then we also obtain the most relevant concepts for the document. Both rankings are normalized to sum 1 and can as such be interpreted as probability distributions. Therefore multiplying both weights of a concept id essentially states that we require the corresponding concept to be relevant for term and document simultaneously.

$$\begin{aligned} &\text{argument } (r_1, r_2) \in R(X) \times R(X) \\ v_1 &\leftarrow r_1 \text{ as a vector (normalized to sum 1)} \\ v_2 &\leftarrow r_2 \text{ as a vector (normalized to sum 1)} \\ &\text{return } v_1 \text{ element-wise multiplied with } v_2 \end{aligned}$$

Algorithm 5.4: Merging the results of two queries

5.3 NMF

Non-negative Matrix Factorization is another approach to cope with the problems of LSI. As described in the paper [7] it is also an iterative algorithm calculating an approximated decomposition $\text{dtm}^T \approx W \cdot H$ of the document term matrix. A key feature of this decomposition is that all elements of the matrices W and H are non-negative.

In general Non-negative Matrix Factorization repeatedly applies an update function to the matrices W and H to minimize some cost function. Therefore a particular variant of NMF can be described by two update rules u_h and u_w and a cost function c .

In the paper cited above two variants are described. The first one is based on the euclidean distance between the approximated and the original document

term matrix:

$$\begin{aligned} u_h &= H_{cd} \leftarrow H_{cd} \frac{(W^T \mathbf{dtm}^T)_{cd}}{(W^T W H)_{cd}} \\ u_w &= W_{tc} \leftarrow W_{tc} \frac{(\mathbf{dtm}^T H^T)_{tc}}{(W H H^T)_{tc}} \\ c &= \sum_{d \in D} \sum_{t \in T} (\mathbf{dtm}_{dt}^T - (W H)_{dt})^2 \end{aligned}$$

The other on the relative entropy between the approximated and the original document term matrix:

$$\begin{aligned} u_h &= H_{cd} \leftarrow H_{cd} \frac{\sum_t W_{tc} \mathbf{dtm}_{td} / (W H)_{tc}}{\sum_t W_{tc}} \\ w_h &= W_{tc} \leftarrow W_{tc} \frac{\sum_d H_{cd} \mathbf{dtm}_{td}^T / (W H)_{td}}{\sum_d H_{cd}} \\ c &= \sum_{d \in D} \sum_{t \in T} \left(\mathbf{dtm}_{dt}^T \log \frac{\mathbf{dtm}_{dt}}{(W H)_{dt}} - \right) \end{aligned}$$

5.3.1 NMF and the GM

Similar to PLSI the matrices W and H already fulfill all our requirements on the result of the clustering phase. Therefore we can use W as the pwz and H as the pzd matrix.

Likewise the paper [7] does not explain how to perform queries. A suitable way seems to be cosine comparison of the query with the documents, concepts or terms as appropriate (already described in section 4.3.3).

5.4 Spectral Clustering

Spectral Clustering as described in the paper [8] is just a clustering method. Unlike LSI, NMF, and PLSI it is not an retrieval scheme. Nevertheless we will show how Spectral Clustering can be used to do retrieval.

Standard Spectral Clustering performs a strict clustering of documents. The algorithm itself is listed in algorithm 5.5. In principle it is based upon it affinity matrix which denotes the distance between pairs of documents. The important step of Spectral Clustering is the calculation of the k largest eigenvectors of a matrix L which is derived from the affinity matrix. After stacking these eigenvectors in columns one can treat each row in this matrix as a point in \mathbb{R}^k . These points are clustered into k clusters using K-means or any other standard clustering algorithm.

affinity matrix: $A \in \mathbb{R}^{\#D \times \#D}$
 $A_{ij} \leftarrow \begin{cases} \exp(-\frac{\|s_i - s_j\|^2}{2\sigma^2}) & i \neq j \\ 0 & i = j \end{cases}$
 $D \leftarrow$ diagonal matrix with $D_{ii} = \sum_{d \in D} A_{id}$
 $L \leftarrow D^{-1/2} \cdot A \cdot D^{-1/2}$
 $x_1, x_2, \dots, x_k \leftarrow k$ largest eigenvectors of L
 $X \leftarrow [x_1 x_2 \dots x_k] \in \mathbb{R}^{\#D \times k}$
 normalize: $Y \in \mathbb{R}^{\#D \times k}, Y_{ij} \leftarrow X_{ij} / (\sum_j X_{ij}^2)^{1/2}$
 Cluster the rows of Y by K-means

Algorithm 5.5: Spectral Clustering

5.4.1 Spectral clustering and the GM

As mentioned above we will use this section to show how a clustering algorithm can be used to do retrieval. To do so we define a mapping from the result of a clustering algorithm to the matrices needed by the generic model.

One can treat the calculated k clusters as the set of k new features. In that case the pzd matrix is a boolean matrix. For each document cluster pair the corresponding element in the matrix is 1 if and only if the document is contained in the cluster and 0 otherwise.

In general, terms are not needed to do spectral clustering and therefore spectral clustering makes no statement on the cluster and term relationship. However we realized that there is a simple and meaningful way to create a pwz matrix given a pzd matrix and the original DTM.

5.4.2 Deriving pwz from pzd

An entry in the pwz can be calculated by the following formula:

$$pwz_{tc} = \sum_{d \in D} pzd_{cd} \frac{dtm_{dw}}{\sum_{t \in T} dtm_{dt}}$$

Above formula expresses that a term which occurs very often in documents of high importance to a given concept should be marked as important with respect to this concept.

Notice that this derivation depends in no way on the nature of spectral clustering! In particular one could use it with any non strict clustering algorithm as well.

5.4.3 Executing queries

As noted before there are no query functions defined in the spectral clustering paper [8]. Nevertheless we use the approach outlined in section 4.3.3. Preliminary tests seem to provide good results but a detailed study is the subject of future work.

Chapter 6

Measurement in the GM

To aid humans in the evaluation of the performance of clustering algorithms several different algorithms have been developed. These algorithms have in common that they create either some graph or a special number which is supposed to express the performance of a clustering. Let us have a closer look at two different techniques.

precision recall In this method the results of several $F(T, D)$ queries on a fixed collection are compared to predetermined optimal result. The results of this comparison are then summed up in two values precision and recall. It requires extensive preparation to determine this optimal result, which so far can only be done manually.

relative entropy This technique calculates the similarity between the DTM and the approximated DTM by the formula

$$\sum_{d \in D, t \in T} \text{dtm}_n(d, t) \cdot \log \frac{\text{dtm}_n(d, t)}{\text{pwd}_{n_{d,t}}}$$

where dtm_n and pwd_n are normalized variants of the matrices dtm and pwd , that is normalized to element sum 1.

The key observation here is that there are measurement techniques like precision recall which need to perform queries and on the other hand there are techniques like relative entropy which only need access to the matrices of the GM.

In the implementation of ALWIS we distinguish between both kinds of measurement techniques. To allow the user to add more measurement techniques of the same kinds we modeled both of them as external programs, please refer to [4] for details.

Chapter 7

On the creation of the DTM

Creation of the document term matrix from a given collection of texts involves several steps and is in general independent from the clustering algorithm used. Nevertheless as part of our work we developed a very flexible way to create a document term matrix based on the UNIX stream concept. An overview is displayed in figure 7.1.

The starting point of every clustering algorithm is the DTM. Before the actual creation of this matrix the documents themselves have to be preprocessed to remove unwanted terms, characters and reduce the terms to their stems. For a more detailed explanation of these operations see [9]. These operations have all to be done in the same way for each document of a given corpus. Otherwise one might introduce unwanted (dis-)similarities between the documents. Therefore we bundle all documents of a given corpus into a single stream before applying these operations to it. Each operation can now easily be implemented as a stream transformer. Again refer to [4] for details.

The importance of careful creation of the DTM can not be stressed enough. In particular stemming and stop word elimination are very important.

7.1 Stemming

A stemmer is an algorithm which determines a stem form of a given inflected or derived word form. This stem is not guaranteed to be identical to the morphological root of the word, but related word forms are supposed to be mapped to the same stem. The process of applying a stemmer to a given text is known as stemming.

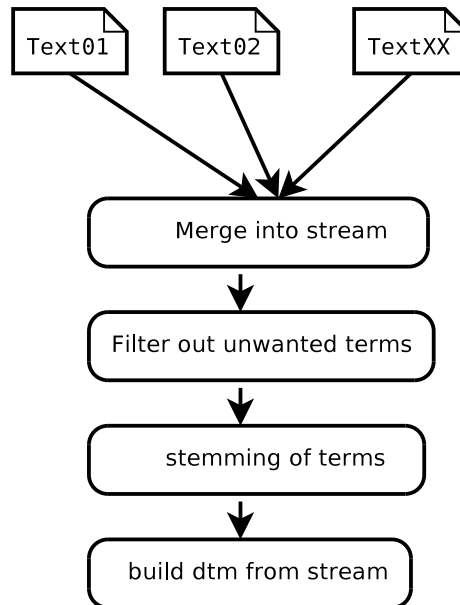


Figure 7.1: Creation of a DTM

In general the semantic value of a word does not change when it is inflected. As we are interested in the semantic value of a document when determining the concepts of a corpus stemming obviously improves the performance of a clustering.

Note however that stemming is highly language dependent. For an overview over existing stemming algorithms see [5].

7.2 Stop word elimination

A stop word is a term that appears frequently within a text, but has very little informational content. Therefore stop words only increase the size of the DTM and add noise to the model e.g. by introducing artificial concepts. Luckily its quite easy to create a list of stop words. For most languages there are already standard stop word lists. Also it is quite easy to create a corpus specific stop word list by just adding all terms to the list which occur with a certain minimum frequency.

Chapter 8

Conclusion and future work

Together with my colleague Daniel Fischer I implemented a visualization tool which puts emphasis on the comparison of single specific queries across several concept based retrieval schemes.

To facilitate that we have shown how to map different concept based retrieval schemes into the generic model. This model introduces the notion of a generalized query. The generalized query has proven itself to be helpful in understanding the performance of a concept based retrieval scheme on a single query. For certain applications it might be advantageous to provide the power of the generic model to the end user. For example a movie database might want to provide means to list related movies.

We are convinced of the usefulness of our tool, nevertheless we are interested in reports of using ALWIS. In particular we would like to hear about additional mappings of other concept based retrieval schemes into the generic model.

We hope that our tool will help in understanding the performance of concept based retrieval schemes. It might be interesting to compare the performance of the same clustering using different kinds of query functions.

Acknowledgments

Our supervisors Holger Bast and his colleague Ingmar Weber were very helpful during the development of ALWIS. They tested the different prototypes of the graphical user interface. Both are actually working in the field of concept based retrieval schemes and as such they are future users of our program. This tight feedback loop resulted in a lot of constructive criticism and useful suggestions.

As mentioned before we developed ALWIS in a team of two. I wish to thank my colleague Daniel Fischer whose perfectionism and thoroughness combined with his clever mind were one of the driving forces in this project. I also want to thank one of my fellow students, who was always ready to provide distractions when we needed them. She will know who we are talking about.

Last but not least I want to thank my family. They are just great.

List of figures, tables and algorithms

4.1	Matrix-query relationship	10
4.2	Generic query $F(X, Y)$ in the GM	10
5.1	Folding a term query $Q(T)$ into concept space in PLSI	13
5.2	Folding a query $Q(X)$ into another space Y in PLSI	14
5.3	Query for concepts given document and term	15
5.4	Merging the results of two queries	15
5.5	Spectral Clustering	17
7.1	Creation of a DTM	22

Bibliography

- [1] Holger Bast and Debapriyo Majumdar. Why spectral retrieval works. In *28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*, pages 11–18, Salvador, Brazil, August 2005. ACM. cited on page(s) **11**
- [2] Holger Bast and Ingmar Weber. Insights from viewing ranked retrieval as rank aggregation. In Jun Adachi, Wang Shan, and Athena Vakali, editors, *Workshop on Challenges in Web Information Retrieval and Integration (WIRI'05)*, pages 243–248, Tokyo, Japan, April 2005. IEEE. cited on page(s) **14**
- [3] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990. cited on page(s) **11**
- [4] Daniel Fischer. Alwis – a visualisation tool for concept based retrieval schemes – design and implementation, 2006. This is the Bachelor thesis of my colleague. cited on page(s) **4, 19, 21**
- [5] W. B. Frakes. Stemming algorithms. pages 131–160, 1992. cited on page(s) **22**
- [6] Thomas Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd Annual ACM Conference on Research and Development in Information Retrieval*, pages 50–57, Berkeley, California, August 1999. cited on page(s) **12**
- [7] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2000. cited on page(s) **15, 16**
- [8] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm, 2001. cited on page(s) **16, 18**
- [9] Ian H. Witten, Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, May 1999. cited on page(s) **21**

Eidesstattliche Erklärung

Hiermit erkläre ich, Benedikt Grundmann, an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Benedikt Grundmann

Saarbrücken, den 10. März 2006