AI 2006, Group 11

# Information retrieval system

Fortunato Flores Ando Vasiliki Georgiadou Andreas Hultberg
Karl Jansson Olivier Mehani

Winter 2006

# Contents

# Introduction

In a time when gathering and exchange of information is growing more important every day, it is becoming very important to find automated ways of filtering out the relevant information. Databases of documents and articles are growing to enormous sizes and are often very dynamical too.

Just think about the newspapers, how many articles do they publish everyday? Some of these databases can also have various authors which could make two articles that are basically about the same topic appear very different depending on the style of writing and the words usage of each author. In this case it would be preferable to search in the database on the context of a query rather than on exact words used.

This report aims to describe different approaches to information retrieval using mainly three paradigms, all of which are designed for retrieving documents with the same context as the query.

# Chapter 1

# Indexing methods

## 1.1 Indexing

To be able to search in the set of articles there has to be some information about which words appear where. The process of finding out this information is called indexing. When the index was created a system called inverted index was used. In the inverted file index each word is associated with a list of articles in which it occurs. So the whole collection of articles is read through and indexed according to this method and the data is stored in a martix where the columns represent the words and the rows the articles. The nature of this problem is such that the index martix is going to be very sparse. (Because we need a row to be as long as the total number of words in the dataset). One way to reduce the amount of data is to use stemming which is explained below.

## 1.2 Stemming

Stemming is the procedure of trying to reduce a word back to its root meaning. For example the words “run”, “ran”, “running”, “runner” etc.. all refer to basically the same concept. Observing that one may wish to represent that concept with a single term, for example “run”, called the stem.

A stemmer is an algorithm performing that task. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. A stemmer for English, for example, should identify the string “cats” (and possibly “catlike”, “catty”, etc.) as based on the root “cat”, and “stemmer”, “stemming”, “stemmed” as based on “stem”.

English stemmers are fairly trivial (with only occasional problems, such as “dries” being the third-person singular present form of the verb “dry”, “axes” being the plural of “axe” as well as “axis”); but stemmers become harder to design as the morphology, orthography, and character encoding of the target language becomes more complex.

For example, an Italian stemmer is more complex than an English one (because of more possible verb inflections), a Russian one are more complexes (more possible noun declensions) and Polish stemmers are known to be one of the hardest of all to construct.

The advantage of stemming is that the number of unique search terms is reduced considerably making indexing easier and less memory consuming. However the effectiveness of stemming for English query systems is still questioned by lots of people and we have not found any real proofs of its importance. Despite that we still chose to use stemming with no other motivation then the reduced memory usage and a possibly better performance. We were quite confident though that the stemmer worked as it should, because we used a java implementation of the widely spread Porter stemmer, originally constructed by the computer linguistic Martin Porter, which is commonly known to be the best English stemmer up to date.

# Chapter 2

# Vector space models and tf-idf Search

## 2.1 Boolean Search

The Boolean engine we implemented permits searches using And, Or and Not operators. The default search is one that makes an Or search over all words of the query retrieving all articles that contain at least one word of the query. The relevance is calculated according to this simple rule: the relevance is the fraction of words of the query the document contains. Thus a document containing all the words of the query will be given qualification 1.0 and a document with occurences of two of the words of the query will have $\frac{2.0}{n_{\text{words in query}}}$.

The hash tables we used to keep track of the relations of words and articles made it a straight forward matter to program the boolean search engine actually being quiet time-wise efficient.

## 2.2 Vector space models for information retrieval

The vector space model for information retrieval [1] is an algebraic model that tries to represent natural language in a formal manner. The central concept is that a text document is viewed as a vector in a multi dimensional space, where every unique word corresponds to its own dimension. All these documents can be put into a matrix as columns and thereby constructing the so called document-term matrix, often denoted A. This model was used for the first time by the SMART Information Retrieval System but has been extended in a large number of ways since then. One of the most significant improvements is the term weighting schemes.

## 2.3 Term weighting background

Different terms have different importance in a particular text document. Very common words like (the, a, and, in, ...) are most likely to show up in almost every document written in English. So those words give little or none information on what the document is really about. On the other hand if you encounter the word 'electromagnetic' you can be quite sure that you have encountered some sort of physical or technical text that deals with electromagnetism implying that this word holds a lot more information.However, if you are searching in a database only covering different aspects of electromagnetism, the term most likely will show up in every document and will provide you little information this time. This means that the amount of information a particular word gives you is dependent on the word itself but also on what document collection you are searching in.

## 2.4 tf-idf weighting

The most common weighting scheme is called term frequency- inverse document frequency, tf-idf for short. It is an entropy measure and can be said to capture the information you gain when you encounter a certain word i in a certain document j. The formula looks like this:

$$tf = \frac{n_{ij}}{\sum_k n_{kj}} \tag{2.1}$$

where $n_{ij}$ stands for the number of times the word i appears in a certain document j and that is then divided by the total number of words in that document. This measures the importance of all the words in a single document. A word that is found many times in a document is probably more important to that document.

$$idf = \frac{D}{d \supset t_i} \tag{2.2}$$

Here D stands for the total number of documents in the collection and the denominator is a measure of how many documents contain the term i.

Finally the combined measure is created by

$$tfidf = tf \cdot \log(idf) = \frac{n_{ij}}{\sum_k n_{kj}} \cdot \log\left(\frac{D}{d \supset t_i}\right) \tag{2.3}$$

This procedure is then repeated for all terms and all documents in the collection .

The resulting weighted document-term matrix will still be as sparse as before but the integer entries representing raw term frequencies have been replaced by weighted number instead.

## 2.5 The search procedure

When the indexing part is done, the actual searching can begin. What the user does is that he or she types in a query, which is then weighted in exactly the same way as before, with the query viewed as a document of its own, although usually a very short one. The resulting vector $\vec{q}$ (existing in the high dimensional term-space) can then be compared to one column j of A [1] at a time. The comparison is done by a given similarity measure. The most commonly used is called the cosine measure, and that is also what was used in our implementation. The formula looks like this:

$$r_i = \cos\,\theta_i = \frac{< q', v_k^i >}{|q'||v_k^i|} \tag{2.4}$$

where $v_k^i$ stands for one row in the document matrix $V_k$ and the scalar product is the usual vector product. The highest ranked articles are then returned to the user in descending order.

## 2.6 User Feedback

With this vector representation of documents there exist a very easy and straight forward way of implementing user feedback. Let's assume that the user first searches with an ordinary query as usual. Then he or she quickly screens through the highest ranked documents and mark them as relevant or irrelevant. Finally the articles marked as relevant can then form a new query vector, by for example take the normalized vector sum of the relevant article vectors. And when the search is done again, with this new vector, it usually performs better than before mainly because this new vector now is much less sparse than the original one which usually only consists of one or a few words.

[1] recall that a certain column j in the A matrix corresponds to the article j in the collection

## 2.7 A simple example

The document collection:
d1: Small insects hate flying
d2: Insects are small annoying creatures
d3: Flying bees are bees
d4: I hate bees

## 2.8 Raw Frequency Matrix

No pre-processing has been performed for this example. But normally the most frequent words, called stop words, are removed and most often the words are also stemmed , as described above. But all this is left out right now for simplicity.

The A matrix:

| | d1 | d2 | d3 | d4 |
|---|---|---|---|---|
| small | 1 | 1 | 0 | 0 |
| bees | 0 | 0 | 2 | 1 |
| are | 0 | 1 | 1 | 0 |
| flying | 1 | 0 | 1 | 0 |
| insects | 1 | 1 | 0 | 0 |
| annoying | 0 | 1 | 0 | 0 |
| creatures | 0 | 1 | 0 | 0 |
| i | 0 | 0 | 0 | 1 |
| hate | 1 | 0 | 0 | 1 |

The weighted matrix:

$$A' = \begin{pmatrix} 0.1733 & 0.1386 & 0 & 0 \\ 0 & 0 & 0.3466 & 0.2310 \\ 0 & 0.1386 & 0.1733 & 0 \\ 0.1733 & 0 & 0.1733 & 0 \\ 0.1733 & 0.1386 & 0 & 0 \\ 0 & 0.1386 & 0 & 0 \\ 0 & 0.1386 & 0 & 0 \\ 0 & 0 & 0 & 0.2310 \\ 0.1733 & 0 & 0 & 0.2310 \end{pmatrix} \tag{2.5}$$

If the user does a search for q='bees' her query vector q will look like this: $q = (0, 1 * \log(4/2), 0, 0, 0, 0, 0, 0)^T$ and she will get out the articles 3 (rank=0.82) and 4 (rank=0.58).

# Chapter 3

# A Neural Network approach to Information Retrieval

## 3.1 Introduction

It is obvious that the retrieval of relevant documents to a user query is a highly demanding procedure, which mainly tries to approximate the vague cognitive process followed by humans when seeking for relevant information. Neural networks seem to be quite promising for a variety of tasks where a more vague human inspired process should be considered, for example pattern recognition or classification. Therefore, it does not come as a surprise that there has been an increasing interest of neural network integration in information retrieval models.

One of the most common approaches to informationretrieval based on neural networks is the spreading activation network. Theabove mentioned model is a simple Hopfield style network combined with aninteractive activation and competition learning rule. However, a more attentivestudying of the model shows that it resembles the traditional vector space model mentioned in previous section.[2].

A different approach that employs the backpropagation learning rule in the information retrieval process is the so-called COSIMIR model (*CO*gnitive *SIM*ilarity Learning in *I*nformation *R*etrieval) which is discussed in the following sections.

## 3.2 The COSIMIR Model

The COSIMIR model is simply a feed-forward neural network which learns through backpropagation algorithm to calculate the similarity measure between a query and a document representation. It is claimed to be able for integrating human knowledge into the core of the retrieval process [3] [4].

### 3.2.1 Description

The network is consisted of three layers: input, hidden and output layer. The query and document representation obtained during the indexing phase are fed into the input layer in parallel. The activation then spreads through the unique hidden layer into the output layer consisting of one unit which denotes the similarity measure between the query and document representation. Therefore, using the backpropagation learning rule is possible to approximate a function for calculating the similarity coefficient that may approach the cognitive procedure of human similarity judgments.

The documents in the database are ranked according to the similarity measure resulting from the above mentioned procedure. The most relevant documents are then to be presented to the user.

It is obvious that in order to train the network beforehand, a suitable database of documents is required along with their relevance to potential queries. When the network has approximated the desired similarity function, is ready to be used in the searching procedure described above.

### 3.2.2 Relevance Feedback

Relevance feedback can be easily integrated into the COSIMIR model simply by requiring from the user to rank some of the resulted documents according to their similarity to the initial query. The network is trained once again using the updated data. The searching procedure is then executed denoting the new ranked documents. In that way, it is possible to implement a user personalized system if needed.

### 3.2.3 Advantages

It is quite clear that one of the main advantages of the COSIMIR model is its ability to provide a cognitive function for calculating the similarity measure, in contrast to the mathematical functions that are currently used by most of the information retrieval models, such as the cosine and the dice coefficient.

Furthermore, even though the user could personalize such an information retrieval system, COSIMIR maintain the general attribute of neural networks being a tolerant processing method. This means that different opinions of different users will not dramatically change its performance. In addition, COSIMIR tolerates and integrates potentially diverse definitions of similarity provided by the users.

Last but not least, COSIMIR could be used in a heterogeneous environment where the query and the document representation are formulated using different techniques.

## 3.3 LSI as Preprocessing Technique

As mentioned in previous sections, documents and queries are processed in order to be represented as vectors in a multidimensional space. However, using a rough representation could lead to very large sparse vectors which may not be dealt easily with neural networks. Hence, a preprocessing method of such a representation is essential for the success of the discussed model.

One of the most promising techniques, which was also used in order to develop an efficient enough search engine, takes into account the representation resulted using Latent Semantic Indexing, abbreviated as LSI [5]. In that way the number of input units can be easily reduced to 200-600 from twice the number of words in the document collection (see section 4 on page 12).

## 3.4 Evaluation

Unfortunately, training the neural network was proven to be a quite difficult and demanding task both on memory and time requirements. Therefore, appropriate evaluation of the COSIMIR model is not available.

This may indicate a potential drawback of the model, since neural networks, usually described as black boxes, cannot be easily debugged.

Furthermore, to our knowledge, the COSIMIR model has been tested for small experimental databases. Further experiments with real data could lead to useful conclusions and improvements.

## 3.5 Generic perceptron and supervised learning implementation

For the neural network based approach, a basic implementation had to be written.

Several reasons motivated this choice rather than using a pre-existing package. First, it had to be compatible with the COLT library, in order to avoid manipulations of the input and output data as much as possible. Moreover, whereas providing a good abstraction to the underlying neural process, it was necessary to be able to add or modify basic functionalities of the network.

Both IR-techniques that have been considered were making use of interconnected layers of neurons only[1]. It has then be decided to start by implementing a single layer, which would then be stacked into a complete perceptron when needed. A generic supervised learning rule model has then been added to teach the network.

### 3.5.1 Generic layer

A generic layer represents a number $M$ of neurons connected to $N$ input units via $N \times M$ weights. Each of these neurons has a bias which serves as a threshold for the activation function. The possibility to define a specific activation function for each node in the layer was not necessary for this project, therefore all the neurons in one generic layer share the same activation function.

In prevision of the activation spreading model, where the layer can be reversed (input and output reversed), a slightly more complex model has been coded. This model allows for propagation in both directions, thus considering the former inputs as actual neurons with biases.

Finally, the model can be graphically represented as on Fig. 3.1.

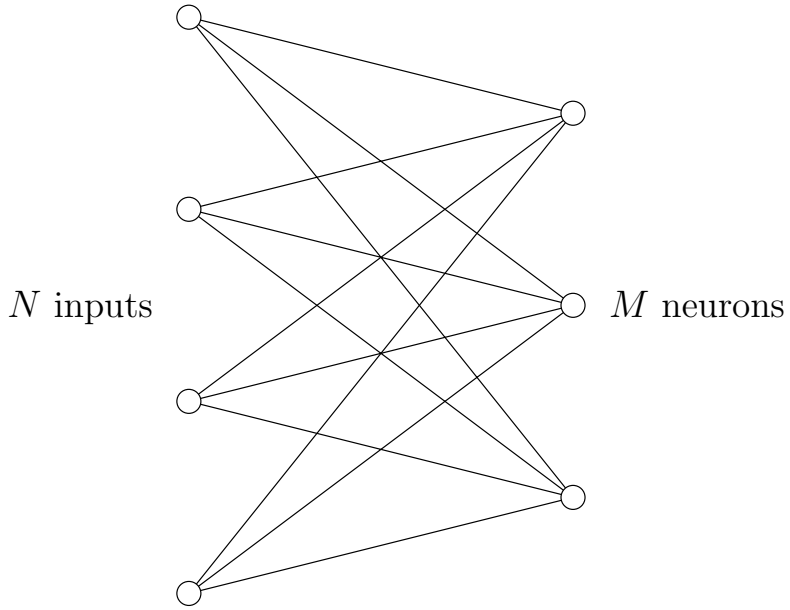


Figure 3.1: A graphical representation of a generic layer.

#### Activation functions

Each neuron in an artificial neural network applies a non-linear function to the sum of its inputs. It is this non-linearity that provides a big part of the interesting behavior (and inherent complexity) of neural networks. This function is usually referred to as the *activation function*.

Several activation functions can be used. Actually, every non-linear function might be a good choice. One observes, however, that a few well known functions are used in most of the cases. The

[1] the activation spreading optionnaly used in-layer neurons connections which were planned to be added later, depending on the need

value they take usually drastically vary around a specific value of the input, which is called the threshold. One can list some of these functions

**threshold function** the simplest activation function which is 0 when the sum of the inputs is below a given threshold, or 1 otherwise;

**sigmoid function** a smoothed version of the threshold function, which has the main interest of being derivable;

**gaussian function** the output is high when the input is close to the threshold, and low when it is too high or too low.

It has been noted, for the sigmoid function, that it was possible to compute its derivative. This is a very interesting feature as some learning methods use the derivative of the learning rule in order to compute the updates to the network.

The implementation relies on the interface of an activation function, which can be configured to return either the function itself (applied to a given input) or its derivative.

Three of these functions have been ported to this interface, namely the threshold function (for tests purposes) and the sigmoid and tangent ones, which are very similar to one another. These functions have been subsequently used in the layers and networks.

Whereas it should be necessary to allow the specification of an activation function for each node in the layer, the choice has been made to allow only for one activation function for a whole layer. This made the generic layer smaller in memory and easier to use, moreover, more flexibility was not needed.

### 3.5.2 Generic neural network

A generic neural network can simply be built out of the already defined basic blocks presented above. The first generic layer serves as the input layer of the network, then the subsequent layers are connected to the outputs of the preceding one. The only obvious limitation is that the number of output nodes of the former matches the number of input terminals of the latter.

Propagation in the network is then simply handled by giving a matrix of the values to apply to the input terminals. After going through the propagation process for all layers, a matrix of the values output by the network is obtained.

### 3.5.3 Backpropagation Learning rule

As discussed above, the COSIMIR model uses backpropagation in order to learn the similarity features for different documents. A version of this classic neural network learning rule has been developed for the generic neural network implementation.

A learning function “teaches” the network to better rate the similarities of two documents (or a document and a query) by taking the matrix representation of both documents, and the expected similarity measure (either from the test set or as modified by user feedback).

# Chapter 4

# Improvement of the matrix-modelled indices: Latent Semantic Indexing

## 4.1 Background

The biggest problem with systems for information retrieval is the enormous number of words used in natural language. In extent some of the words have different meanings in different contexts (polysyms) and other words have more or less the same meaning (synonyms), or are at least strongly related to each other.

You can describe a certain subject in an almost infinitely variety of ways and using different word combinations, depending on the style of writing and word usage. Although in a more abstract, semantic level, the texts are basically describing the same thing which a human reader probably will notice quite easy but for a machine it can be a very difficult task to see that the texts are related.

In the Latent Semantic Indexing approach (LSI) to this problem mathematics and in particular linear algebra is used to handle the problem of dimensionality. Again we look on the document-term matrix $A$ described above, preferably normalized by for example the tf-idf method. But now we notice that $A$ usually is very sparse, in fact sometimes the load factor can be as low as 0.001-0.002%. That is because most words in the dictionary are not present in a typical text snippet. So the idea is to compress $A$ to a smaller and more compact matrix.

## 4.2 SVD

In order to compress $A$, first an Singular Value Decomposition (SVD) of it is performed.Without going into any deeper mathematical details, the basics of SVD are:

We decompose $A$ to three different matrices

$$A = U\Sigma V^T \tag{4.1}$$

Exactly how this decomposition is done is not important for our case. But it is worth to mention that the procedure can be very time and space consuming, especially for large matrices A, which we in fact are dealing with here. There have been lots of research made on this topic alone and the problem is not at all trivial. But for the moment we assume that the decomposition is already made and we will come back to address the problems involved of doing it later.

The properties of the matrices are:

$$U^T U = V^T V = I_n \tag{4.2}$$

naming them the right and left normalized singular vectors respectively
and

$$\Sigma = \mathbf{diag}(\sigma_1, ..., \sigma_n),\ \sigma_1 > \sigma_2 > ... > \sigma_n \geq 0 \tag{4.3}$$

which contains the ordered set of singular values for $A$. It is important that the singular values are ordered, because then you can very easily construct an approximation of $A$ called $A_k$. You only have to take out the k largest (most important) singular values and the corresponding left and right singular vectors from U and V (see figure 4.1) and multiply them together to form $A_k$. It can then be proved that the matrix:

$$A_k = U_k \Sigma_k V_k^T \tag{4.4}$$

in fact is the closest rank-k approximation to $A$ that exists for any unitarily invariant norm.

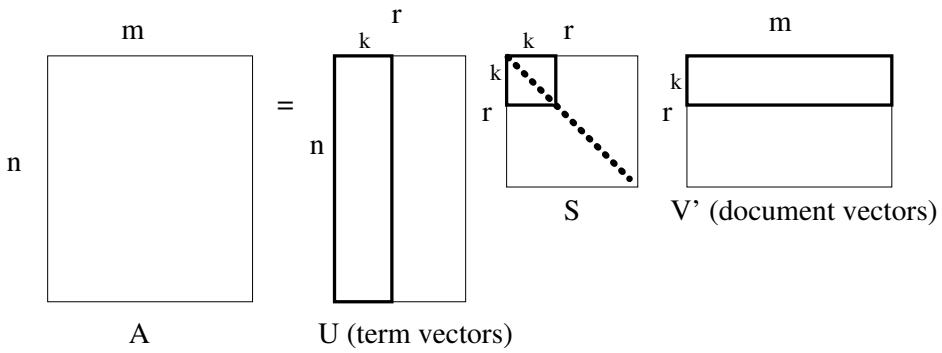


Figure 4.1: n=number of terms, m=number of documents, r=rank of the A matrix

## 4.3 Preparing the query

When the indexing part is completed, and we have our term matrix $U$ and document matrix $V$ we want to use them for comparison with the query vector $\vec{q}$ prepared exactly as before in the tf-idf search.In order to do so $\vec{q}$ is projected onto the right space by the following matrix multiplication:

$$q' = q^T U_k \Sigma_K^{-1} \tag{4.5}$$

Finally $\vec{q'}$ can be compared to each document one by one with one of several similarity measures.

## 4.4 The reduced document-term space

This reduced matrix $A_k$ has some very nice features. First of all, it is much smaller than the original $A$ thereby reducing memory usage which almost always is an important issue. Secondly, it will be significantly faster to compare a query vector $\vec{q}$ and a document vector $\vec{d}$ in this new space because they are much shorter now, since they have only k-elements instead of n (and it is usually the case that $k << n$). Thirdly and most importantly is that, we have also improved the IR system by this; 'noise' due to word usage and different styles of writing have been smoothed out and the underlying semantic structure has been better relived. Unfortunately there are no or very few mathematical proofs on how and why this actually works. People have tried to analyze this and maybe give certain meanings to the different dimensions but have not had any great success on more than very small toy examples. However, an enormous amount of empirically tests of this method has shown that it works and the performance is improved, sometimes dramatically.

## 4.5 A Simple Example

Here follows an example with a very small document collection:
c1: *Human* machine *interface* for Lab ABC *computer* applications
c2: A *survey* of *user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user*-perceived *response time* to error measurement
m1: The generation of random, binary, unordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors*: A *survey*

Documents c1-c5 deals with computer human interfaces and m1-m4 is about graph theory.

### 4.5.1 Raw frequency matrix

Note this time the articles have been pre-processed a bit, stop words like (a,the,and,for,of) has been removed and words that only shows up in one article have been left out, leaving the *italicized* words. The dataset is also deliberately chosen so that a k=2 is sufficient. The term weighting part is also left out for simplicity but it should of course be included in real applications, due to the significantly increased performance by doing it. The **A** matrix will look like this:

| | c1 | c2 | c3 | c4 | c5 | m1 | m2 | m3 | m4 |
|---|---|---|---|---|---|---|---|---|---|
| human | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| interface | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| computer | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| user | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| system | 0 | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| response | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| time | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| EPS | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| survey | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| trees | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| graph | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| minors | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

### 4.5.2 Decomposition

After decomposition and compression we are left with this:

$$A_k = U_k * S_k * V_k^T =$$

$$\begin{pmatrix} 0.22 & -0.11 \\ 0.20 & 0.62 \\ 0.24 & 0.49 \\ 0.40 & 0.27 \\ 0.64 & -0.14 \\ 0.27 & 0.11 \\ 0.27 & 0.11 \\ 0.30 & -0.14 \\ 0.21 & 0.27 \\ 0.01 & 0.49 \\ 0.04 & 0.62 \\ 0.03 & 0.45 \end{pmatrix} \cdot \begin{pmatrix} 3.34 & 0 \\ 0 & 2.54 \end{pmatrix} \cdot \begin{pmatrix} 0.20 & 0.61 & 0.46 & 0.54 & 0.28 & 0.00 & 0.02 & 0.02 & 0.08 \\ -0.06 & 0.17 & -0.13 & -0.23 & 0.10 & 0.10 & 0.44 & 0.62 & 0.53 \end{pmatrix} \tag{4.6}$$

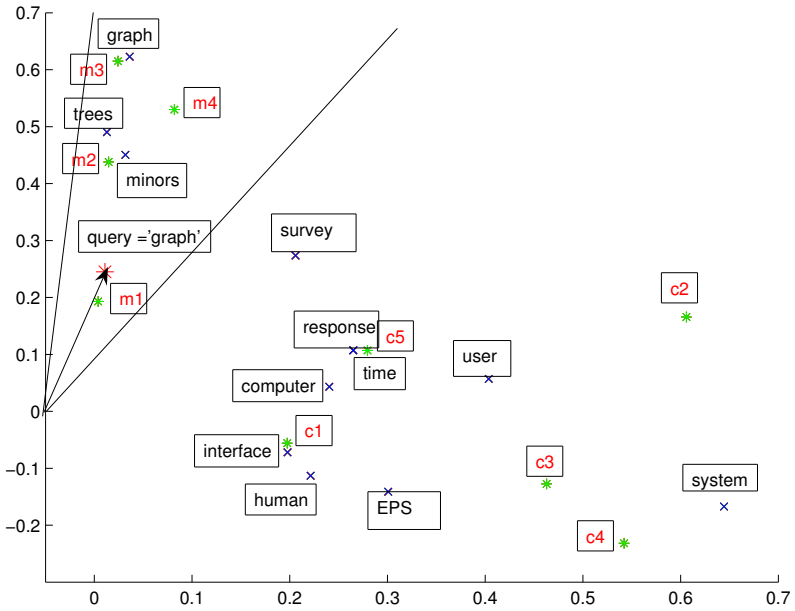


Figure 4.2: Graphical representation of terms and articles. '*'=articles and 'X'=terms. The arrow shows the $\vec{q}'$ vector and the lines to the right and left of it represent a particular $cos\,\theta$ threshold. If better precision is wanted the angle can be narrowed, and if better recall is wanted the angle should be increased.

Note if the SVD is done in matlab the signs may be switched on some of the columns in U and V.

### 4.5.3 Vector representation

When k=2 it is possible to represent all terms and documents in a 2D figure to see how they are geometrically positioned. That is shown in figure 4.2. In the plot a query vector is also shown which in this case only concists of one word namely 'graph'. Note that 'response' and 'time' have exactly the same position in space which also can be observed in the U matrix. That is natural because in the collection those two word only shows up together, meaning that their correlation is 1. And it is exactly these kinds of relations one wants to capture in this model.

## 4.6 Problems and limitations

Although the LSI model has lots of interesting properties it is not perfect. The biggest problem is the construction of the SVD matrices which is both time and memory consuming. In our case we had a collection of about 1400 articles with about 7000 uniqe stemmed terms and the SVD took 3 hours to compute[1]. However, in order to decompose the swedish IR collection, given on the course homepage (150 000 terms and 17 000 documents) we estimated that it would take about 4-7 day. Apparently a considerably long time but still not the biggest issue,since we ran out of memory and therefore we could not even try to run it.

Bearing in mind the above mentioned limitations, the conclusion is that for really large datasets it is almost impossible to do the SVD in the usual way, by solving equation systems,and thus

[1]The computers in the physics department where used: IntelP4 2800MHz processors and 2GB RAM

another approach is needed. An interesting thing to remember though is that it is only the $A_k$ matrix that is needed and not the entire decomposition. So if an algorithm finds an approximation to $A_k$ in a faster and less memory consuming way the problem could perhaps be solved.

### 4.6.1 Hebbian Learning

One approach to this is to use a generilized hebbian algorithm as described by [6]. For a detailed description of the algorithm and the mathematics look in the article. But here follows a brief description of the basics.

The important thing to remember is that U is a matrix of eigenvectors for $A * A^T$ and V of the eigenvectors for $A^T * A$ and that $\Sigma$ is a diagonal matrix of the square roots of the correspondning eigenvalues. Furthermore $V = A^T * U * \Sigma^{-1}$, so it suffices to find the eigenvectors and eigenvalues of $A * A^T$. The main idea is to first find one eigenvector of this matrix, the one with the largest eigenvalue, which

usually is called the principal direction. The method of finding it is the interesting thing, it is not calculated in the usually way but it is learned instead! Every document in the collection is presented to the algorithm one by one and the vector is updated every time by a fixed step in the same direction as the current document shown. That will make the vector to jump around rapidly in the begining but after a while settle in the right direction and then only grow longer and longer in that particular direction given by the largest eigenvalue. Then the first column of U is found and we can go back and try to learn the next one in almost the same way. The only difference is that when a document vector is presented to the algorithm, the part of the documemt vector that points in the direction of the first eigenvector has to be removed first.

Unfortunately, we did not have time to implement this in our program but it looks like a very interesting approach and there are currently a lot of research done in these fields at the moment with new results all the time. With more time we most definately would have included this.

# Chapter 5

# Results and comments

Note: The neural network we created trying to implement the COSIMIR model never worked since it never completed the learning phase. Training never really improved the performance of the engine to levels better than chance. Thus this method is not discussed in the following analysis.

## 5.1 Dataset

The four implementations of the text retrieval engine were tested using a dataset on cystic fibrosis [7]. It consists of 1239 documents and after stemming we obtained a set of 7484 different terms that could be searched for.

## 5.2 Accuracy measures

In order to evaluate the performance of our different methods we used three common measures in the field of information retrieval: *precision*, *recall* and *F-measure*.

**precision** is the fraction of relevant documents out of the total number of documents retrieved by the searh method,

$$P = \frac{n_{\text{relevantretrieved}}}{n_{\text{totalretrieved}}}; \tag{5.1}$$

**recall** is the fraction of the relevant documents retrieved

$$R = \frac{n_{\text{relevantretrieved}}}{n_{relevant}}; \tag{5.2}$$

**F-measure** is the harmonic mean of the recall and the precision.

$$F = 2\frac{RP}{R+P}. \tag{5.3}$$

The two first measures usually represent a tradeoff. When one goes up the other goes down. To have a unique measurement of the overall performance the last measure is introduced. It weights as much as both the previous ones and can simply be seen as their average.

## 5.3 Analysis

The cystic dataset came with one hundred queries and their relevant results. The engines have been tested with such queries. The results are shown in Table 5.1 on the following page.

| Method | Precision | Recall | F-Measure |
|---|---|---|---|
| boolean search | 0.1261 | 0.5096 | 0.1650 |
| tf-idf | 0.6265 | 0.0905 | 0.2044 |
| tf-idf w. feedback | 0.7454 | 0.1322 | 0.2713 |
| LSI ($k = 150$) | 0.2943 | 0.2764 | 0.2486 |
| LSI with feedback ($k = 150$) | 0.3908 | 0.3097 | 0.2998 |

Table 5.1: The accuracy measures for the search methods implemented in the application.

Important things come out right away. First, the boolean search has the lowest precision and the highest recall, second the tradeoff between both measures is readily noticeable especially in the first three rows (and actually a balance found in the last two).

The fact that the boolean method has the largest recall and smallest precision can be understood by the fact that usually with this method bigger sets of documents were retrieved (since only looking for occurrences of words of the query) than with tf-idf or LSI. Boolean search has more recall because in a bigger set of retrieved documents, even by chance, it is more probable to find the relevant ones than in a more reduced one. But this, of course, came with the reduced precision, meaning that just a low fraction of such retrieved documents were actually relevant.

Tf-idf, with and without feedback, had the biggest precision and lowest recall. This can be understood in an analogous way to the boolean scenario, but in this case with smaller sets of retrieved documents a big majority of which were relevant to the actual query.

It can be seen from the table that the best scenario in any case was achieved with the Latent Semantic Indexing method. The case included in Table 5.1, is the one with $k = 150$ which gives the biggest F-measure of all. With this method, after choosing a suitable size for the Singular Value Decomposition of the tf-idf matrix, something that has to be tuned empirically (see below), the best balance between precision and recall was obtained. The operations performed on the original matrix did not only keep the metasymbolic, semantic relationships but also made the search a lot easier and faster.

The simple feedback mechanism which has been implemented, making a new query using the most relevant articles in the retrieved set, worked very well and did manage to improve the search. Of course it has not been used with the boolean search. This would certainly have retrieved a very big set, thus shooting recall way up and precision way down, finally making it impractical from the user perspective. In both tf-idf and LSI, however, feedback improved all of the performance measures meaning more relevant documents and in a higher density were achieved.

To tune the size of the SVD matrix used in LSI the 100 queries were run in order to obtain measures for sizes going from 50 to 300. Fig. 5.1 on the following page shows precision, recall and F-measure for the LSI without feedback and Fig. 5.2 on page 20 shows the same in the case with feedback. It can be seen that the point in which the engine works the best, given that both precision and recall are of the same importance, is for $k = 150$, close to where both curves cross. This is a kind of "equilibrium" and certainly we have not found a way to calculate such an optimal $k$ thus a drawback is that it has to be obtained empirically. By comparing Fig. 5.1 and 5.2 one can also notice that the system consistently performed better with feedback thus validating the simple feedback scheme.

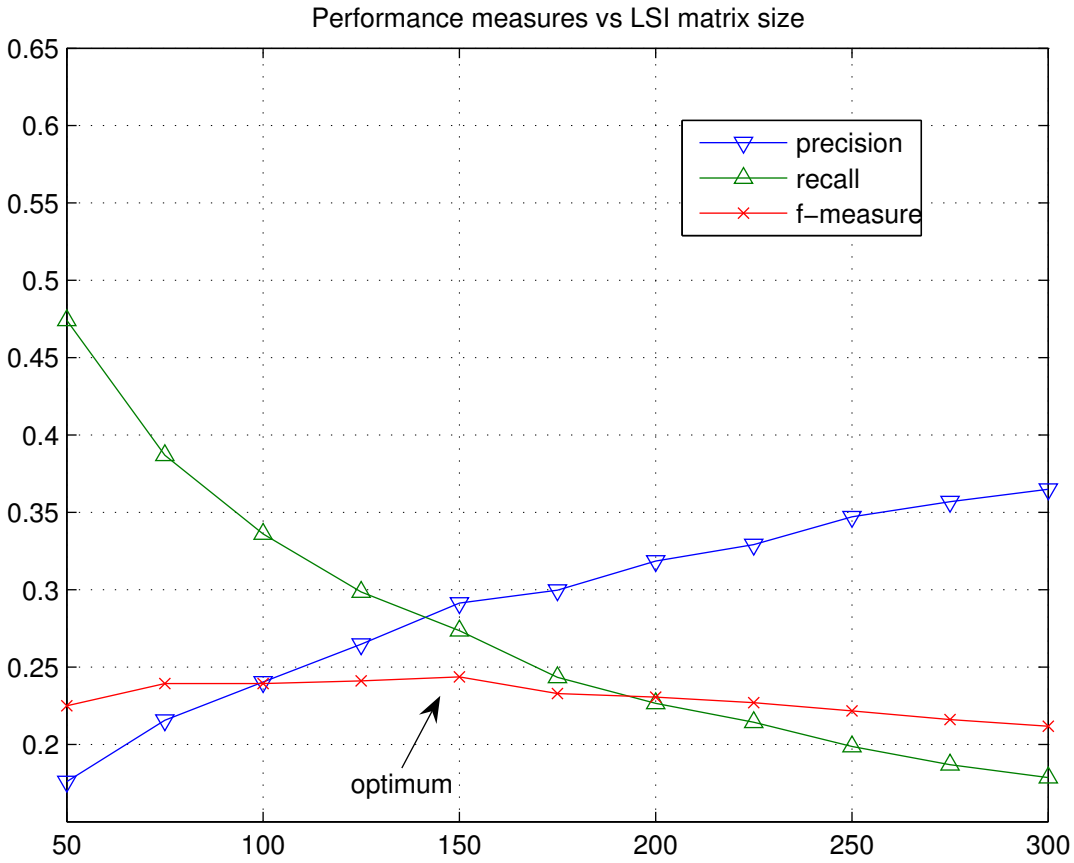


Figure 5.1: Accuracy measures for the LSI method without feedback.

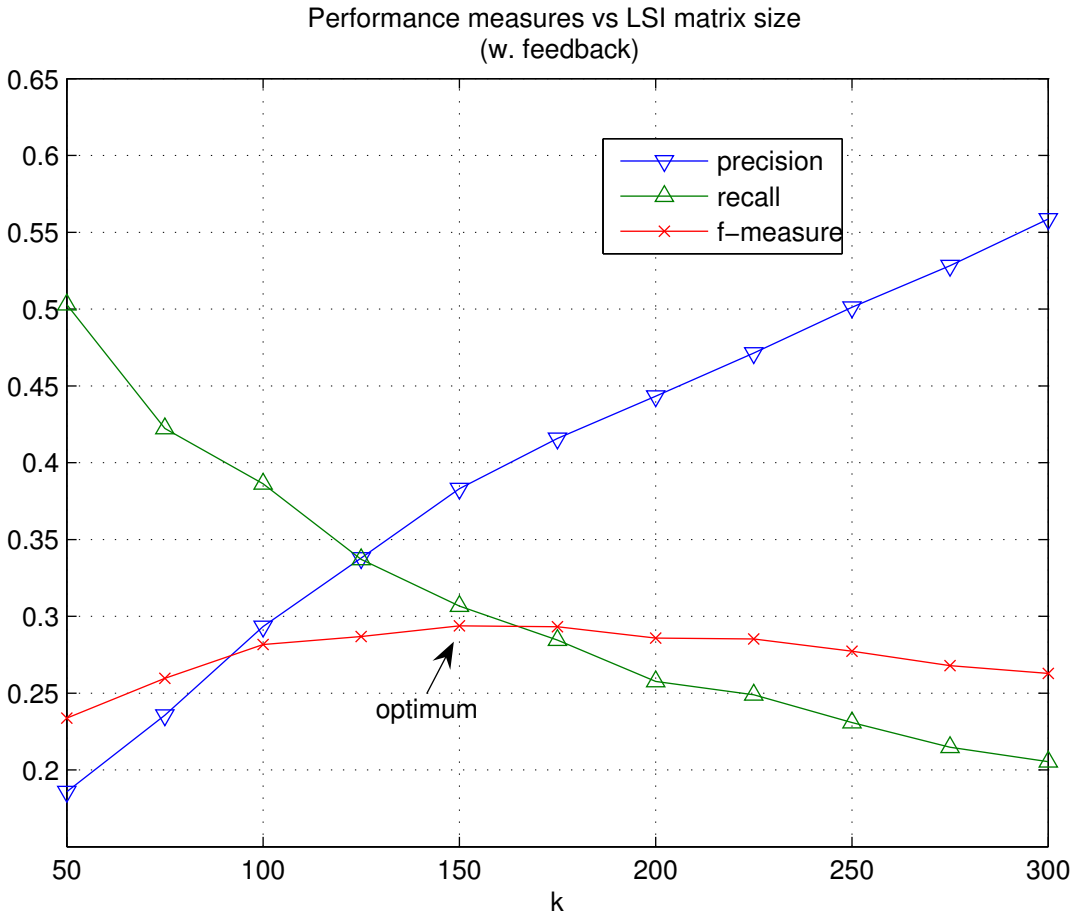


Figure 5.2: Accuracy measures for the LSI method with feedback.

# Conclusion

In this project different approaches on Information Retrieval systems were introduced and evaluated.

The vector space models actually performed quite well, especially the Latent Semantic Indexing model. However, as commented before, a serious drawback with the latter was that calculating the SVD was proved to be an extremely time and memory consuming task for large document collections. An incremental approach to the SVD problem should be used instead.One of the most promising seems to be the hebbian learning algorithm, which unfortunately we were not able to impement on the grounds of time constraints.

The tf-idf search without compression also performed well, though the results were not as satisfactory as in the case of LSI model but still acceptable enough. However, it was a considerably slower algorithm as far as search time concerns, since scalar products between very long vectors should be computed.

The performance was also improved considerably with feedback search,which was quite expected, since that model exploits less sparse query vector enhancing in that way the already existed algorithm. For LSI the effects of feedback search were not as significant, though still an improvement. This implies that feedback search based on comparing documents with each other could be a helpful tool in IR systems. Moreover it can be very easily implemented in a vector space model.

A quite different approach in information retrieval was presented as well, based on a backpropagation network. The COSIMIR model, in theory, seems to have the potential to enhance the existing information retrieval models, though further evaluations are necessary in order to truly make use of its advantages.

The biggest disappointment was that, mainly because of time shortage, indexing the Swedish test collection was not possible. That would have been quite helpful in our survey since the corresponding perfomance on that set might have been even better.That is because the subjects in that set were much more spread and covering many different topics, whereas in the cystic data set all articles were about cystic fibrosis in some way or another possibly making it harder for the IR system.

# Appendix A

# Graphical Interfaces

To facilitate the usage of the information retrieval system two different graphical user interfaces where created. The evaluation GUI and the user GUI. The usage of these two will be explained along with some information on how they work behind the scenes. For now the interfaces are only able to search in the cystic dataset.

## A.1 Using the Evaluation GUI

The evaluation GUI was primarily created for an easy to understand representation of the performance by the different search algorithms. To use it run the evaluatorGUI class and the following interface will show up at you screen (see figure A.1).

All the pre specified search queries given from the dataset are loaded on start-up from the query file and the search strings are found in the drop-down-list. The other drop-down-list lets the user choose which of the four different search algorithms he or she wants to use. So when the query and searcher has been decided and the ”search” button is clicked the query is passed to the search algorithm that springs into action. The articles retrieved by the searcher is presented in the left list and the relevant articles for the present query, according to the document set, are presented in the right list. The documents that appear in both of these lists are highlighted blue to indicate that these are relevant according to the specification of the dataset. In the bottom of the window the three performance measures we used to evaluate our system are prompted when the search is performed.

### A.1.1 Feedback Search

The GUI has also an additional button called ”Search the List” which is used to simulate the feedback search. It gives the user the ability to perform an additional search using information given by the user. What happens in the real case is the following: The user reads through the highest ranked articles and marks them as relevant or not relevant. The user can press the ”Search the List” button. What happens then is that the three highest ranked articles marked relevant by the user are fed into the search algorithm again. Or more precise the vector sum of the three corresponding document vectors is used. The resulting vector is then used as a new query and a search is performed with that vector instead.

What happens in the evaluator though is that the program automatically chooses the three highest ranked articles that where relevant according to the query file in the feedback search.

This method turned out to be quite effective primarily improving the precision but also the recall to some extent, quantitatively results are shown and discussed more deeply in the results section.

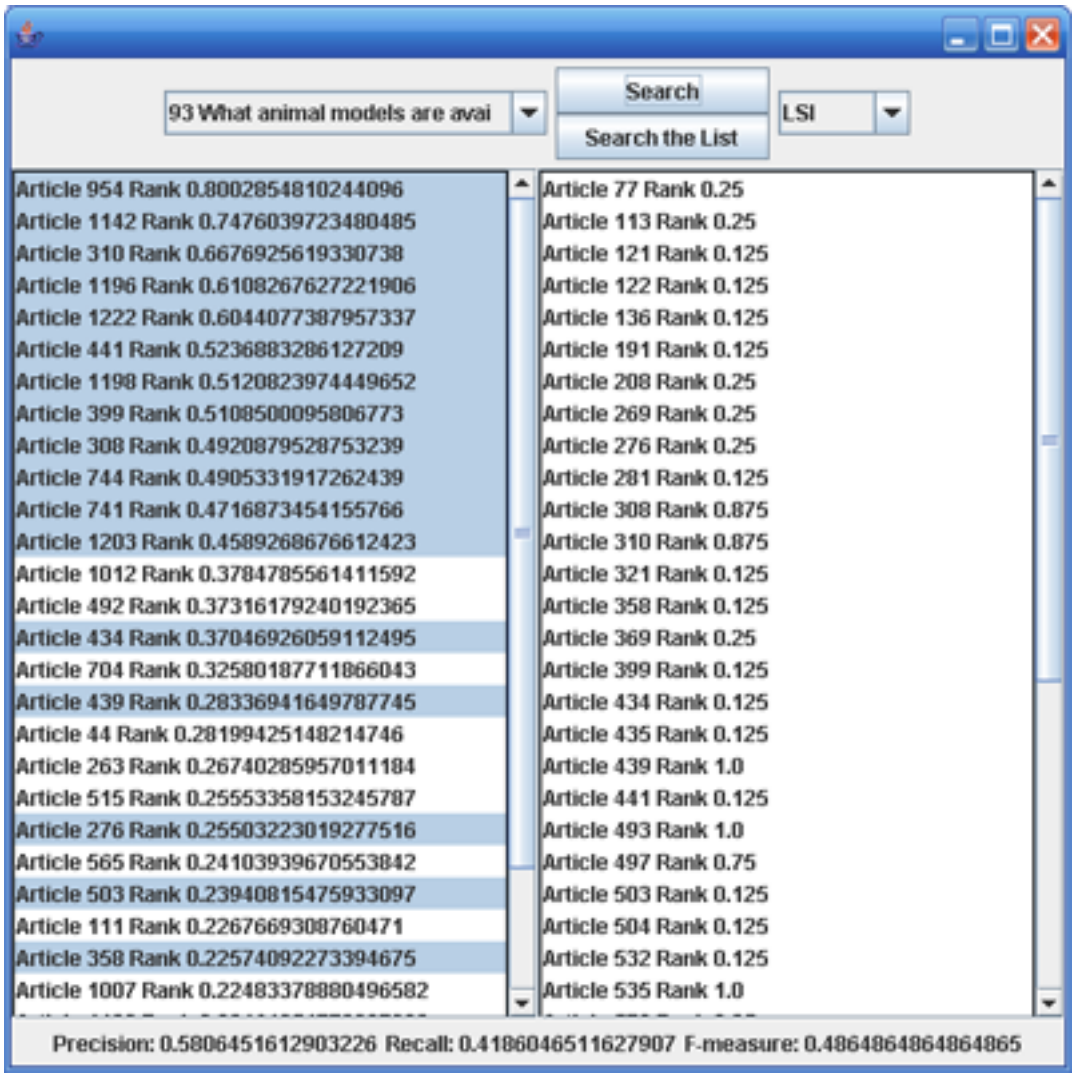


Figure A.1: A screen shot of the evaluation GUI

## A.2 User GUI

This is the program that should be executed by the real users. It has all the basic functions that an interface to a searcher should have. The search query is typed in the text field and the drop-down-list lets the user specify the preferred search algorithm. When the ”search” button is clicked the retrieved articles are presented in the list on the right hand side. To view the article simply double click on one of the elements in the list and the article will be revealed in the text area to the left. This program should be used when the IR system is complete, all the parameters are tuned and the indexing is done.

# Bibliography


[1] G. Salton A. Wong C. S. Yang. A vector space model for automatic indexing. volume 18, pages 613–620, 1975.

[2] Mothe J. Search mechanisms using a neural network model. In *Proceedings of the RIAO*, pages 275–2. Rockfeller University, 1994.

[3] Thomas Mandl. Tolerant information retrieval with backpropagation networks. volume 9, pages 280–289, 2000.

[4] Thomas Mandl. Learning similarity functions in information retrieval. In Hans-Jürgen Zimmermann, editor, *EUFIT '98. 6th European Congress on Intelligent Techniques and Soft Computing*, pages 771–775, Sept. 8-10 1998.

[5] Thomas Mandl. Efficient preprocessing for information retrieval with neural networks. In Hans-Jürgen Zimmermann, editor, *EUFIT '99. 7th European Congress on Intelligent Techniques and Soft Computing*, Sept. 13-16 1999.

[6] Genevieve Gorrell. Generalized hebbian algorithm for incremental singular value decomposition in natural language processing. In *11 th Conference of the European Chapter of the Association for Computational Linguistics: EACL 2006*, pages 97–104, 2006.

[7] J.B. Wood R.E. Tibbo H.R. Shaw, W.M. Wood. Cystic fibrosis reference collection.