Concept indexing of news articles with neural systems

Essink, R.

Award date:
2006

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of mathematics and computer Science

Concept Indexing of News Articles with Neural Systems

by
Rob Essink

Supervisors:
A.I. Cristea
N. Rooijmans

Eindhoven, 20th June 2006

Preface

For a long time self learning systems have caught my interest. A graduation in the area of Neural Networks would be only a logical step. As Alexandra Cristea put me on the idea of document searching with neural networks the following work is the result.

There are two reasons for writing this article in English. First I want all members of the Information Systems group to be able to read and possibly comment on this article.
Second it might be useful for future work (a student or an ilse employee) not to exclude people who are not familiar with the Dutch language.

The first part of this report is a literature study, from chapter 7 on analyses on the input data are turned into design features, from chapter 14 on the realization of the code and the results are presented.

In appendix A you will find an exercise to determine the data complexity growth with each new word occurrence (postings). In appendix B the design of a SOM neural network library is described. Appendix C is an exercise in creating a language detector.

I wish to thank my colleagues at ilsemedia for their support and enthusiasm during the process of this report.

Gratitude goes out to all authors who have published their papers on-line, this has made my task a lot easier. I wish to include the people at Citeseer for making it possible to find these articles. In all cases where the authors did not publish their work on-line my regards goes out to the university library.

I wish to thank the following software developers who have provided (free) software for development, benchmarking or provision of my ideas: Dimitri van Heesch (Doxygen) [43], Andrew McCallum (Bow) [33], Michael W. Berry (GTP) [13].

A final thank you I wish to address to all the people at the Eindhoven University. First of all the woman who talked me into this subject and provided me with (hopefully) enough wisdom for obtaining my master degree, dr. Alexandra Cristea. Of course this gratitude includes the rest of my graduation committee (Lora Aroyo, Rudolf Mak and Nils Rooijmans).
Further I note Jaap vd Woude and Henny van Keulen who made it possible to start my graduation.

Contents

Chapter 1

Introduction

1.1 Information Retrieval

For several decades people have made all kinds of attempts to extract information from natural text. It can be from abstracts, articles, literature, etc.
As the amount of digital data increases by the years the necessity to process this information by computer in a more accurate and faster manner grows. This assumption becomes plausible when looking at the position of search engines on the Internet, which over the years have taken a key role.

The field of finding documents or articles that match a certain topic is called *Information Retrieval*.

1.2 Issue

The main problem or issue with Information Retrieval systems is the difference between syntax and semantics. It is very hard to explain to people that a simple Information Retrieval system is incapable of interpreting full sentences ('I want to know the departure of the first train to Amsterdam tomorrow morning').

1.3 Goal

The goal of this research is to look for a system that in a certain sense is capable of analyzing natural text in a semantical fashion, that means that it exceeds simple string matching and/or indexing. The proposed system should be capable of grouping words to a single concept and one word to multiple concepts, resulting in a *concept hash*. A visualisation of a *concept hash* is in figure 1.1.

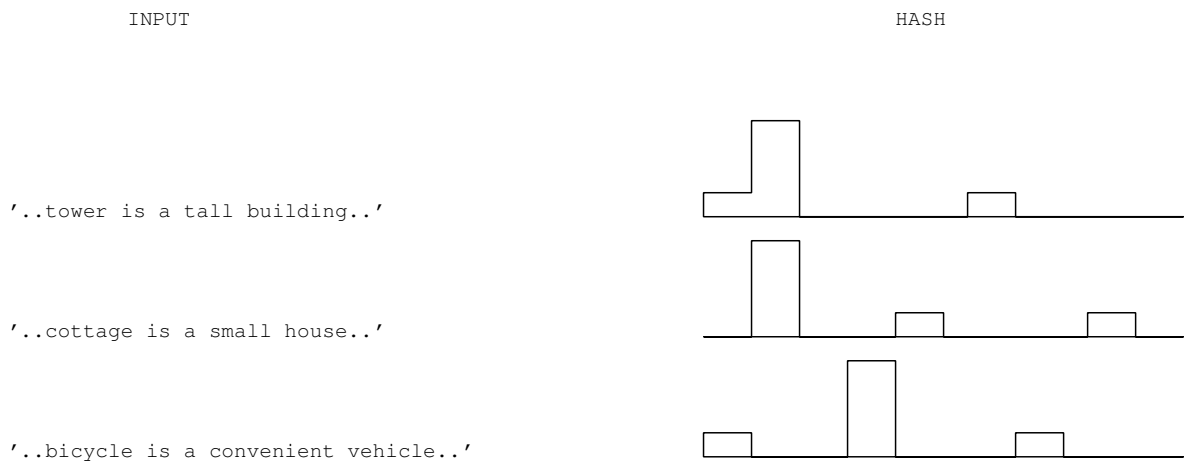

Figure 1.1: Visualisation of a concepthash

The advantage of such a system is that documents can be compared by meaning or that documents can be searched, that match a query more 'naturally'. I have been looking specifically at small and medium sized text corpora. This means that I do not need a supercomputer to handle the input data.

1.3.1 Research questions

- Is it possible to create a concept hash of a document, i.e. is it possible given two documents regarding the same concepts to generate two similar hashes, and if two documents do not have concepts in common to generate two dissimilar hashes (see figure 1.1)?
- Is it possible to create a concept hash of a query, i.e. if it is possible to map a document to a concept hash, is it possible to do the same for a small sequence of words?
- How can I compare two concept hashes?
  - Which comparison methods are there?
  - How can I use this comparison in practice? Which applications can be built with a concept hash?
- How well does the system answering to the above questions perform?

1.4 Method

To reach this goal a *concept hash* can be obtained by relating words that occur within the same context.

To this purpose, traditional methods, as well as the Self Organizing Map paradigm have been reviewed. Primarily because it is a prominent unsupervised neural network.

As input, the NU.nl news feed, an XML document containing articles written in HTML, was used. The language is Dutch and has several categories and subcategories: General news (foreign, internal, politics), Business (company news, macro economics, stock exchange), Computer & Internet, Sports, Gossip.
News documents are, opposed to web documents, well structured, and contain less noise. This makes the NU.nl news feed qualified for this research.

1.5 Location

This research was located at ilsemedia in Eindhoven under supervision of Nils Rooijmans. ilsemedia is a leading company in Internet business within the Netherlands, with news site NU.nl, directory startpagina.nl, and the search engine ilse.nl. The Eindhoven department of ilsemedia is specialized in search engine technology.

Chapter 2

Natural Language Parsing and Information Retrieval methods

Information Retrieval is about finding information within a set of documents which is called the *corpus*.

2.1 Natural Language Parsing

2.1.1 Tokenization

The first step in processing natural language is the *tokenization*, which separates the individual *term*s.

Tokenisation is usually done by assuming that space, newline and several punctuation marks are word separators. Some extra intelligence [14] gives a far better result on large corpora.

2.1.2 Stemming

stemming is the mapping from to word to its stem. In [32] it is shown that information retrieval algorithms perform better when words are substituted by their stem.

For Dutch, the snowball algorithm [37], derived from the Porter stemmer [38], is available.

2.2 Vector Space Model

To perform computations on a *corpus*, the most common way to encode documents is the *Vector Space Model* [36, 5], where each document is mapped to a vector of *term frequencies* (also called *posting*s). Each *term* in the document collection or *corpus* is represented by a dimension.

The collection of *document vector*s is called the *index* of the *corpus*.

2.2.1 Common words

Some words are extremely common, are therefore not distinctive and might even introduce noise when information retrieval algorithms are applied to the *document vector*.

To improve the quality of the *document vector* there are three solutions:

1. **Removing stopwords**
   A way of reducing the number of dimensions is removing (or ignoring) certain *stopword*s from the input. Words like 'a', 'about', 'of', 'in', which do not contribute to the semantics of a document, can be left out without negative consequences.

2. **Pruning**
   Pruning is used here as defined in the WEBSOM system [22], and means that *term*s with a *document frequency* (the number of documents the *term* occurs in) within a certain range are removed from the *corpus*. Words that bring little information are removed. It is mostly used for performance of the program (as the vector dimensionality is reduced).

3. **Term weighting**
   *term weight*ing functions do not remove any *term*s from the *corpus* but try to establish how important a *term* is.

2.3 Term Weighting Functions

Not all *term*s in the *index* are equally *important*. The challenge is to devise a weight $w_{t,d}$ for the presence of a *term* t in an document d within the *corpus* D, given the frequency of *term* t in document d is $f_{t,d}$ and the overall frequency of a term t in the corpus is $f_t = \sum_{d \in D} f_{t,d}$.

The best way is to have $w_{t,d}$ as the product of a *local term weight*, indication the relation between t and d, and a *global term weight*, representing the uniqueness of *term* t in the *corpus*.

2.3.1 Local weighting

A *local term weight* $lw_{t,d}$ only depends on the *term frequency* $f_{t,d}$. The two most common formulas for *local term weight*ing are the *term frequency* itself

$$lw_{t,d} = f_{t,d} \tag{2.1}$$

and *binary weighting*

$$lw_{t,d} = \chi(f_{t,d}), \quad \chi(x) = \begin{cases} 0 & x = 0 \\ 1 & x \neq 0 \end{cases} \tag{2.2}$$

Some others are (see [29]):

- *alternate log* $lw_{t,d} = \chi(f_{t,d})(1 + log(f_{t,d}))$
- *inverse frequency* $lw_{t,d} = 1 - \frac{1}{1+f_{t,d}}$
- *log term frequency* $lw_{t,d} = log(1 + f_{t,d})$

2.3.2 Global weighting

By introducing a global factor based on f_t, one can take into account whether a *term* is unique for a document, or common within the *corpus*. To get the weight for a certain *term* in a certain document ($w_{t,d}$) the *local term weight* is multiplied with the *global term weight* to determine the weight for a certain *term* t in a specific document d: $w_{t,d} = lw_{t,d} \cdot g_{t,d}$

A simple example of such a weight is:

$$g_{t,d} = \frac{1}{f_t} \tag{2.3}$$

There are two commonly used *global term weighting schemes*, of which *idf* (Inverse Document Frequency) [36, 48] is used most.

$$idf_t = log\left(\frac{|D|}{\sum_{d \in D} \chi(f_{t,d})}\right) \tag{2.4}$$

The alternative to *idf* is *entropy* [29, 31, 48].

2.3.3 TF-IDF

The most common *term weight* is *TF-IDF* (so it is mentioned in wikipedia [47]). It is the product of *local term weight* $f_{t,d}$ and *global term weight* IDF_t, so $w_t, d = f_{t,d} \cdot idf_t$.

There are more exotic configurations for *global term weight*ing and *local term weight*ing ([29],[41]). In [48] many possible configurations of weighting schemes and similarity functions have been tested, concluding that there is no absolute winner.

2.4 Vector Similarity

It is convenient to have a function that can produce compact meaningful vectors as describe above. The next requirement is a function to compare these vectors with a reference vector (or with each other).

Commonly, two functions are used for vector comparison: the *Euclidean distance*, the 'distance' between the vector ends, and the *cosine similarity*, the cosine of the angle between the two vectors.

2.4.1 Euclid

The *Euclidean distance*[46] between two vectors is traditionally used to compare image vectors.

$$eucl(x, y) = ||x - y|| = \sqrt{\sum_i (x_i - y_i)^2}$$

2.4.2 Normalizing

A vector can be normalized [31] by dividing each element by the (Euclidean) length of that vector.

$$x' = \frac{x}{||x||} \qquad ||x'|| = \left\| \frac{x}{||x||} \right\| = 1$$

Normalized vectors are easier to use with cosine measurement and random mappings (chapter 10).

2.4.3 Cosine

The cosine similarity function, usually called $sim(x, y)$, is very popular for comparing TF-IDF weight vectors [31]. It has a good reputation with term-searches.

$$sim(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||} = \frac{x}{||x||} \cdot \frac{y}{||y||}$$

If $||x|| = ||y|| = 1$ then the cosine measurement is equal to the inner product of the two vectors.

The cosine similarity is 1 when the two vectors converge, 0 when they are orthogonal and -1 when opposites.

2.4.4 Correlation Coefficient

The *correlation coefficient* could also be used to determine the relationship between two vectors of equal dimensionality. *Correlation coefficient* [45] usually is applied on statistical data to determine the linear relationship between two stochastic variables.
If the correlation coefficient is +1 the two vectors have a stong positive relationship, -1 means a strong negative relationship and values that near or equal 0 indicate a weak or non-existent relationship. The *correlation coefficient* of stochastic variables X and Y, $\rho_{X,Y}$, is defined as:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E((X-\mu_X)(Y-\mu_Y))}{\sigma_X \sigma_Y}$$

Where $cov(X,Y)$ is the covariance of X and Y, σ_X is the standard deviation of X, $E(X)$ the mean of X just as μ_X.

2.5 Measuring Relevance with Recall & Precision

The performance of an information retrieval system can be expressed by the values *recall* and *precision*. These values can be measured for any *query* ([5], [44]). In this context, a *query* may also be a document, depending on the information retrieval system.

Given a set of documents U and a *query* Q there is a set $B \subseteq U$ of documents that are retrieved by the information retrieval system using Q and a set $A \subseteq U$ of documents that are truly relevant to Q.

- **Recall**
  the relative amount of relevant documents retrieved.

$$\frac{|A \cap B|}{|A|} = P(B|A)$$

- **Precision**
  the relative amount of retrieved documents being relevant.

$$\frac{|A \cap B|}{|B|} = P(A|B)$$

An information retrieval system is considered better than another when it can retrieve more documents (and have a larger *recall*) with the same *precision*.

In retrieval systems which measure a distance instead of returning a boolean value (for instance a search engine, the document is part of the result set or not), a parameter is used.
If, for example, Q is a TF-IDF weighted document vector, U a set of document vectors and sim the cosine similarity function. Let $B_\lambda = \{d \in U | sim(d,Q) > \lambda\}$. For each λ a *recall-precision* pair is found. While iterating lambda over [0..1], a *recall-precision* curve is formed.

Chapter 3

Self Organizing Feature Maps

3.1 Concept

The *Self Organizing Feature Map* (*SOM*) or *Kohonen Map* is a *Vector Quantization* algorithm, which means that every output neuron consists of a reference vector, which has the same dimension as the input. Reference vectors are compared to input vectors, and the output of a neuron is determined by the distance between the input and the reference vector.

Neurons in the Kohonen map are connected to their neighbours, so if a neuron is updated, than so will its neighbours (as defined by the algorith).

Figure 3.1 shows the grid of neurons which for each neuron the respective reference vector visualised with arrows. Each intersection represents a neuron and the reference vectors are indicated by a sequence of arrows

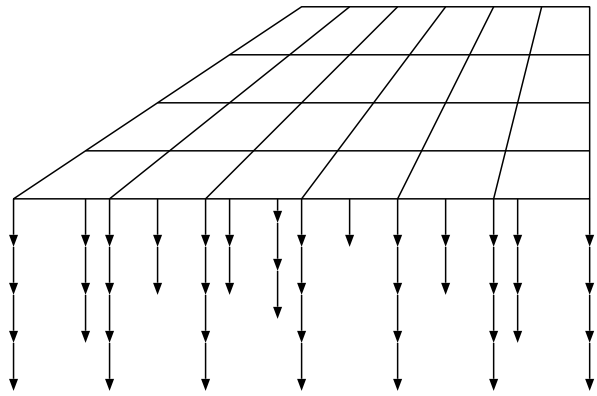

Figure 3.1: Visualisation of a som

3.2 Usage

The Kohonen map is intended for vector clustering. However the concept is being used for all kind of unsupervised purposes, like dimensionality reduction, feature extraction, etc.

3.3 Training

The structure of a Kohonen map is a (usually) 2 dimensional grid of neurons. Each neuron is in effect a reference vector which has the same dimension as the input vector.

Training the grid with a certain input means finding the nearest reference vector in the grid and updating this vector and its neighbourhood vectors.

3.3.1 Distance

The neuron that needs to be updated is determined with a distance function usually noted with $d(x, y)$.

By default $d()$ is the Euclidean distance, but it may also be a cosine function ($d(x, y) = 1 - sim(x, y)$), see 2.4.3.

3.3.2 Neighbourhood

During training, each neuron has a neighbourhood of neurons, which depends on shape of the grid and the radius. In my implementation only the square shaped neighbourhood will be used, but it is also possible to have a hexagonal or diamond shaped neighbourhood. The radius is the maximal distance between two neurons to be considered 'neighbours'.

3.3.3 Update

The update should be minimizing the distance function. For two vectors x and y, in the wordspace $d(x, update(x, y)) \leq d(x, y)$ and $d(y, update(x, y)) \leq d(x, y)$.

The default update method for a reference vector $w_r(t)$ with input x and learning rate α is ([35]):

$$w_r(t+1) = (1 - \alpha)w_r(t) + \alpha x$$

The neighbourhood vector y with distance k from r is updated with:

$$w_y(t+1) = (1 - \alpha \cdot h_{ry})w_y(t) + \alpha \cdot h_{ry} \cdot x$$

where $h_{ry} = e^{-\frac{d(r,y)^2}{2rad^2}}$ and the distance between neurons $d(r, y)$ is the Hamming distance between edges. This function h is a Gaussian function which graduately decreases the impact of the input vector x on the neighbourhood vector y as its position on the map is further away from the winning vector r.

3.4 Output

After a SOM has been trained, it can be queried, by determining the distance from the input vector to all reference vectors. In this case, the SOM functions as feature extraction.

It is also possible to determine the nearest reference vector; in that case the SOM functions like a clustering algorithm.

Chapter 4

Related work

There are many methods and research area's which can contribute to mapping documents to concept hashes. The three area's mentioned below form a large part of those techniques and methods.

4.1 Systems based on Neural Techniques

4.1.1 WEBSOM

In the '80s Kohonen worked on a *Semantic Map* [40] which was able to cluster and/or classify single words using the context of surrounding words in the text.

In the '90s this system (then called a *Word Category Map* [28, 21]) was extended, to cluster complete documents. The information extracted from the word category map was used to create vectors of categories instead vectors of words. These vectors are used as input for a second SOM which clusters the documents.

4.1.2 SOMLib

An excellent related project is SOMLib [3].

Due to the hierarchical architecture of the underlying Growing Hierarchical SOM [1] it is very effective where the size of clusters are of different order of magnitude (like with news articles).

There are several experiments in clustering German news articles.

4.1.3 liGHt SOM

liGHt SOM is a project within the PRIS program. It implements the WEBSOM concept using the hierarchical GHSOM architecture.

Unfortunately there is very little published about the project itself, there is only a website [2].

4.2 Clustering

A method which is somewhat comparable to *Self Organizing Feature Map* (and *LVQ*) is clustering. K-means, discussed in Manning/Schütze [31], is one of the most popular clustering algorithms.

The centroid of a cluster (the average of the weight vectors in the cluster) can be used as substitute for the reference vectors in the SOM.

The difference between K-means and the SOM is that a SOM has a neighbourhood, so it 'learns' from adjoining neurons, also the SOM does not assign input vectors to neurons (or clusters) like K-means does (K-means is a so called hard clustering algorithm).

4.3 Other techniques

4.3.1 SVD

There are a number of matrix techniques for information retrieval. The basis of these techniques is Singular Value Decomposition [20].

A very popular SVD based system is Latent Semantic Indexing (LSI), also called Latent Semantic Analysis ([12],[30]). There are very little details known about LSI, except that it uses the Vector Space Model and SVD.

Principle Component Analysis (PCA) is a singular value decomposition on the covariance matrix instead of the original.

The advantage of these methods is that they are faster than the learning algorithm. The disadvantage is that it is harder to scale (you are confronted with computations on large matrices).

Independent Component Analysis (ICA [23]) is a rediscovered dimensionality reduction algorithm which has become increasingly popular in the last decade. This algorithm expects nongaussian input and therefore requires preprocessing with an algorithm like LSI or PCA (see [7, 25]). Although this makes it seem a less obvious algorithm for text analysis there are a few text analysis applications where this algorithm will excel, like topic identification [24].

Chapter 5

Problem Design

5.1 Introduction

Basically I am looking for a system that has a scheme like picture 5.1, where $M << N$, assuming that the number of concepts is far less than the number of words in the *corpus*.

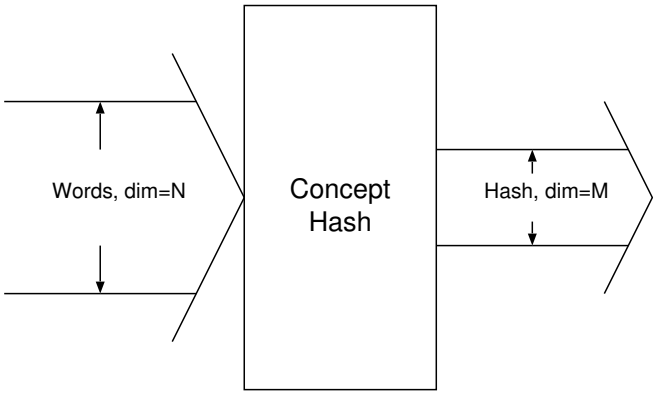

Figure 5.1: Basic Concept Hash Function

The hashing function does not have to be reversible (therefore its dimensionality can be smaller). Most important is that the concept hash is a 'digest' of the document and not a function that removes noise (see section 2.2.1). My hypothesis is that a neural network like the SOM would be capable of achieving that, which I will clarify in chapter 6.

5.2 Supervised/unsupervised

As this system is being trained, I needed to decide what method of training is to be used.

Choosing a supervised method would mean that I have knowledge of the relations between words an concepts. In effect, I would be building classifiers mapping the *corpus* to $\{S_i, \overline{S_i}\}$ where S_i is the set of documents matching concept i ($\overline{S_i}$ the documents in the *corpus* not matching concept i) .
The construction of a classifier is discussed in Appendix C.

However I did not have sufficient information about the *corpus*; I only have a newspaper categorization mentioned in 1.4.

Therefore the SOM is indeed a good candidate.

Chapter 6

Stereotypes

In the previous chapter I promised to clarify why the SOM is such a useful technique. I will explain this with a small metaphor, from which the concept *stereotype* is directly linked to the reference vectors in a SOM.

6.1 Introduction, a metaphor

Imagine a dating service which has a database with profiles of all its members. The company wishes to review couples on features in the profile. One of these features is 'favorite means of transport'.

The company converted the profiles to Boolean feature tables like table 6.1. With vectors like {'transport = public transport', 'transport = bike', 'transport = BMW X5', 'transport = 2CV', ...} keeping other profile features out of this scope. If a couple has a match in the favorite means of transport, one of the Booleans matches.

	Alice	Bert	Christine	Dan
transport				
public transport	✓			
on foot				✓
bike				✓
2CV	✓			
Land Rover		✓		
BMW X5			✓	
sports				
none	✓			
hockey		✓		
tennis			✓	
soccer				✓

Figure 6.1: Personal Profile Table

The only problem is that a 'LandRover man' might prefer a 'BMW X5 woman' over a '2CV woman', but the distances to either women are equal.

To solve this problem, the company wishes to generalize. This can be done by creating classes of means of transport (no-car, cheap-car, SUV, executive-car). Then an 'executive-car' is a better match for a 'SUV' than a 'cheap-car'.

In my opinion, a more effective way, is to generate classes of people. One fictive person represents a whole class of people. These fictive persons are called stereotypes.

A stereotype can be a 'yuppie', 'perfect son/daughter-in-law', 'athlete', etc. The vector representing a stereotype will not have Boolean values, but averages of the features that match that stereotype. An athlete might have 0.7 for bike, 0.2 for public transport and 0.1 divided over the rest of the transport features. The yuppie will favor the expensive cars, but also use public transport (required in the city). The perfect son-in-law will have a preference for SUVs (for the kids) but does not spend too much money (cheaper cars).

Matching a person's profile with a set of stereotypes results in a vector of distances to all stereotypes. Two people with a strong correlation of these distances are likely to have the same kind of taste, also two people with a poor correlation will not share much interest.

The question that remains is: How can you create/find those stereotypes?

6.2 Applying Stereotypes

The traditional information retrieval system only compares the two weight vectors using the *cosine similarity* or *Euclidean distance*; in figure 6.2 this is demonstrated by the upward flow from the two weight vectors.

The stereotype method assumes a number of vectors $\{r_n\}$ with the same dimensionality as the document weight vectors. Each of these vectors represents a cluster of documents in the corpus, and is called a stereotype.

By computing the cosine or Euclid distance between a document vector j and each stereotype r_i, an array of distances $d[j,i]$, is determined for document j ($d[j] = [d[j,0], d[j,1], \dots]$).

By comparing vectors $d[l]$ and $d[m]$, the correlation between document l and m within the context of the stereotypes $\{r_n\}$ is computed.

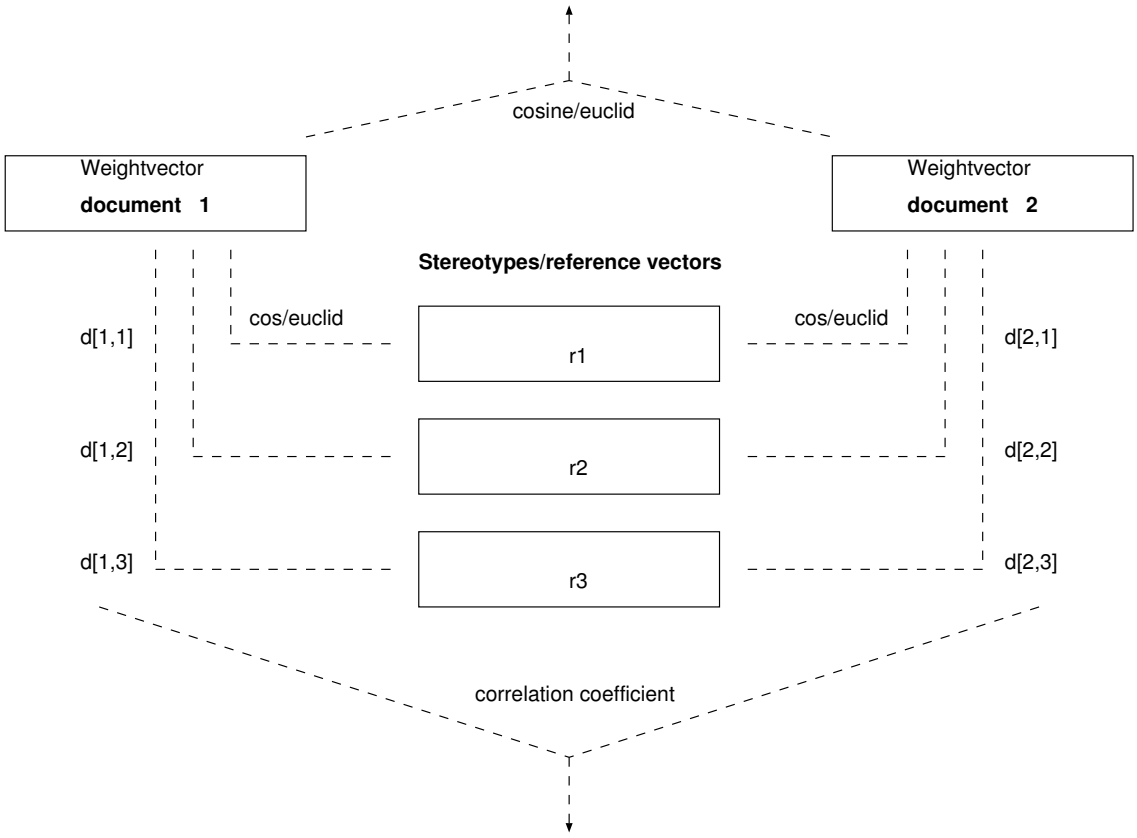

Figure 6.2: Implementation of the stereotype concept

6.3 (Dis-)Advantages

An obvious disadvantage of the method describe above, is that you have to do a lot of computations before being able to determine the distance between two documents. The advantage is that if $\{d\}$ already has been computed, it is a lot faster, as the dimensionality of $d[l]$ is smaller than that of the weight vector of l.

6.4 Kohonen, proof of concept

The example of the dating service can also be used to show the effectiveness of the Kohonen network to generate stereotypes.

In essence, the Kohonen algorithm does not classify, but attempts to create stereotypes and neighborhoods of stereotypes. So this algorithm is the number one candidate for searching for stereotypes in our feature space.

The dating service allowed only one entry in the field 'favorite form of transport'. So, to relate a BMW X5 together with the Landrover to the same stereotype (outdoor rich), we need another profile feature to link these two cars.

Now we will look at sports: assumed is that people who own a Landrover, BMW or Mercedes merely play tennis. Also that people who drive cheaper cars, all play soccer (we do not want to make things too complicated).

The distance between a Landrover tennis player and a BMW tennis player is smaller than between either and a Volkswagen soccer player. The consequence is that neurons with a strong sentiment for tennis will 'attract' LandRover, BMW and Mercedes drivers, resulting in neurons with a strong sentiment for all those three brands (and tennis).

The Kohonen Map fits perfectly within the stereotypes concept, and as the stereotypes concept fits the requirements for the concept hash, so the Kohonen Map is my primary choice. The reference vectors of a trained SOM now form the set of stereotypes.

Chapter 7

Setup

This chapter shows step by step the process from the NU.nl corpus input file to an ordered list of most related documents.

7.1 Preprocessing

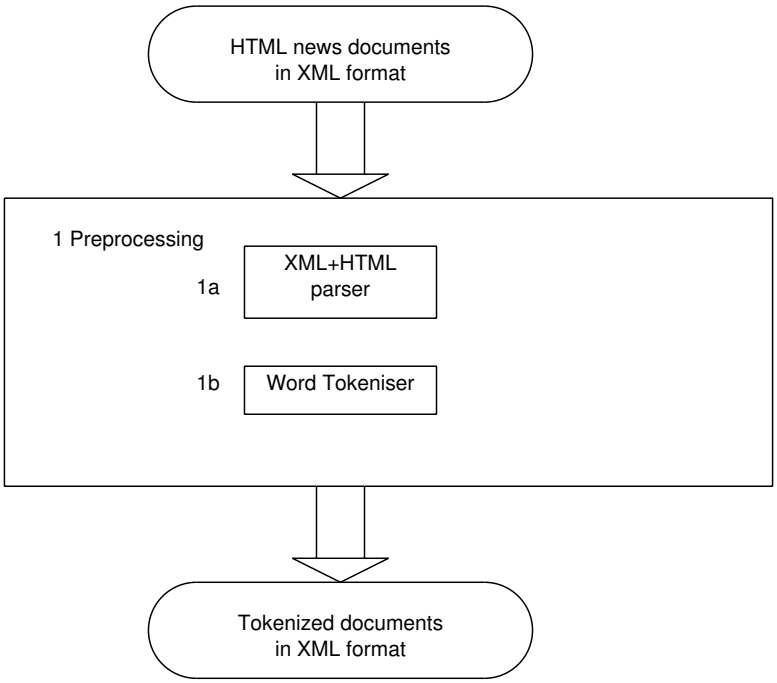

Figure 7.1: Preprocessing

The first step of the computational process (figure 7.1) is the preprocessing of the input.
The input (in XML embedded HTML) is dissected into words (or stems of words).

1a The XML+HTML parser removes all XML and HTML tags with a XML parser and a simple finite state automaton (FSA).

1b The *tokenizer* is a more complex *FSA* described in chapter 8.

```
<nu>
 <item>
  <number>186734</number>
  <title><![CDATA[Shell verhoogt weer prijs autobrandstof]]></title>
  <url><![CDATA[http://nu.nl/news.jsp?n=186734&amp;c=32]]></url>
  <excerpt>
<![CDATA[DEN HAAG - Marktleider Shell heeft donderdag voor de tweede
keer binnen een week de prijzen van autobrandstof verhoogd.
Benzines zijn een cent duurder geworden en diesel 0,6 cent per
liter. De adviesprijs van super plus komt bij Shell nu op 1,223 per
liter. Euro ongelood kost 1,169 en diesel komt op 78,5 eurocent per
liter. ]]>
  </exerpt>
  <created>1060255327</created>
  <update>-1</update>
  <copyright>ANP</copyright>
  <body>
<![CDATA[<P>Oliemaatschappij Shell zegt de prijzen van autobrandstof te
baseren op internationale productnoteringen. De prijs van ruwe olie
is de afgelopen periode gestegen.
</P>]]>
  </body>
 </item>
</nu>
```

Figure 7.2: Input Example

Figure 7.2 shows a part of the input (from the NU.nl news feed). The result of preprocessing can be viewed in figure 7.3.

```
<token_document version="0.20">
  <item cat_id="32" seq_id="15631">
    <part descr="title">
      <word>Shell</word>
      <word>verhoogt</word>
      <word>weer</word>
      <word>prijs</word>
      <word>autobrandstof</word>
    </part>
    <part descr="exerpt">
      <name>DEN</name>
      <name>HAAG</name>
      <dash/>
      <word>Marktleider</word>
      ....
    </part>
    <part descr="body">
      ....
   </part>
  </item>
</token_document>
```

Figure 7.3: Preprocessed Example

7.2 Indexing

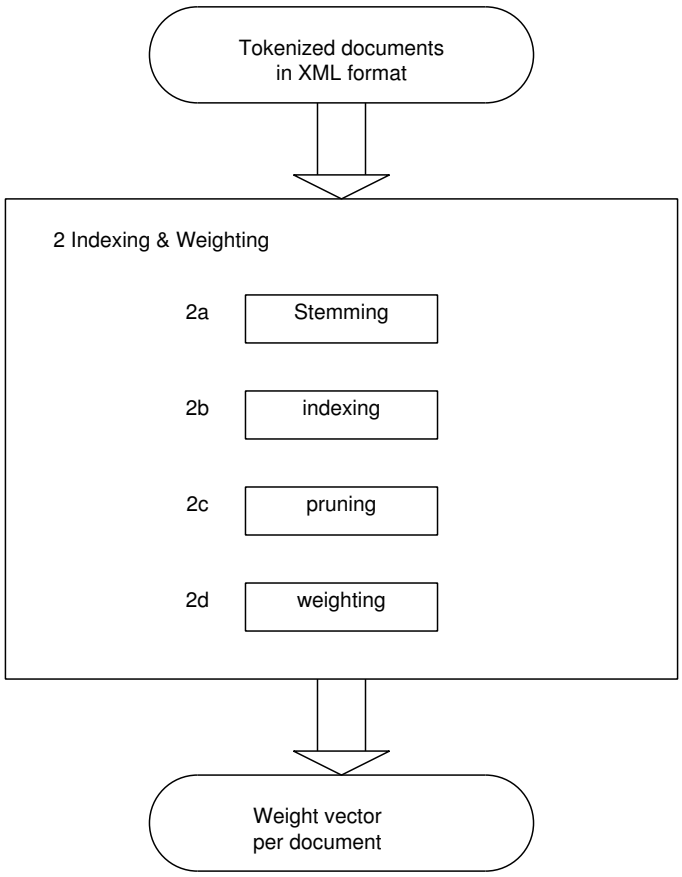

Figure 7.4: Index & Weighting

The second step (figure 7.4) is the creation of vectors, according to the Vector Space Model.

2a *stemming* is optional. This is done with the snowball stemmer [37] which is derived from the Porter stemmer [38].

2b *indexing*, simply counts words in articles in such a way that there is a mapping from word to word-ID to a vector of frequencies per document. Figure 7.5 gives an impression of the resulting matrix.

2c *pruning*, removes all words which do not have a document frequency within a certain interval (see section 2.2.1). This way all irrelevant words are removed.

2d *term weighting*, converts word frequencies into weights with *TF-IDF* (see section 2.3.3, [47]). This results in a matrix like figure 7.6. Weight vectors can be normalized afterwards.

word (wordid)	document #15631	document #15632
shell(0)	1	..
verhoogt(1)	1	..
weer(2)	1	..
prijs(3)	1	..
autobrandstof(4)	1	..
den(5)	1	..
haag(6)	1	..
marktleider(7)	1	..

Figure 7.5: Word Index

wordid	#15631	#15632
0	4.6051	..
1	3.6888	..
2	2.9957	..
3	3.6888	..
4	5.2983	..
5	3.6888	..
6	4.6051	..
7	4.6051	..

Figure 7.6: TF-IDF weight matrix

7.2.1 Random Mapping

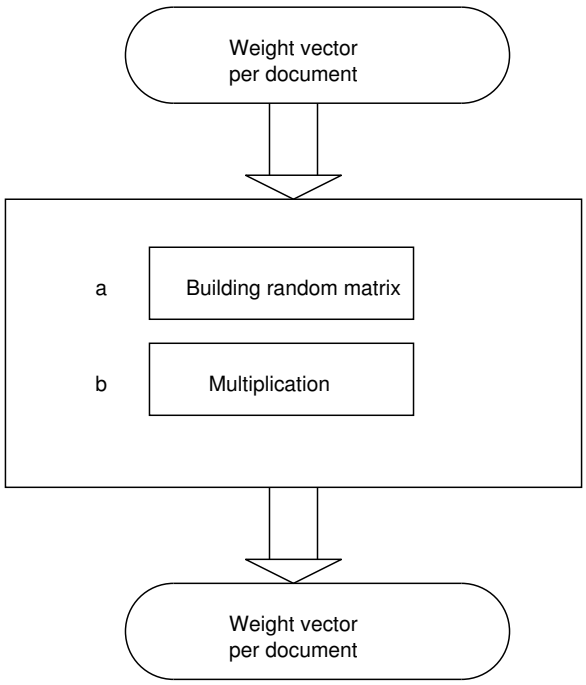

Figure 7.7: SOM usage

To reduce the dimensionality of the weight vector it can be processed with a random mapping or *random matrix* (see chapter 10).
The characteristics of the weight vectors (like length and mutual distances) are preserved, while the dimensionality, memory usage and CPU usage are reduced.

7.3 Training

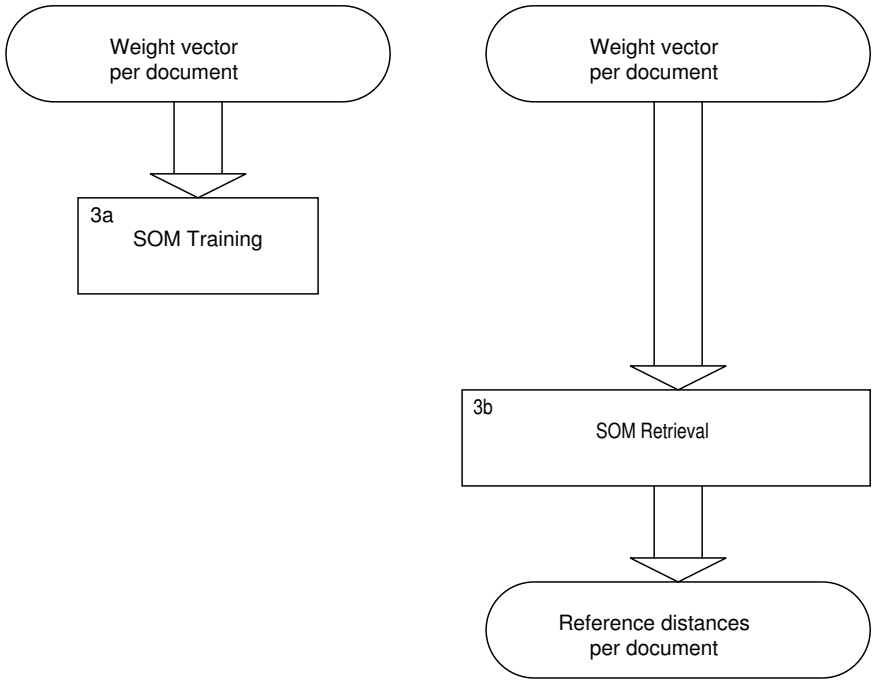

Figure 7.8: SOM usage

The SOM is trained, step 3a in figure 7.8, with the weight vectors from the random map (or directly from step 2 (section 7.2) if no random map is used). The training set contains the weight vectors of all documents. Training sets the final values for the reference vectors in the SOM.

7.4 SOM retrieval

After training the same set of document weight vectors can be used to generate a vector of distances (step 3b) to all reference vectors, as described in section 6.2. The resulting vectors ($d[l]$) will be called distance vectors.

7.5 Ranking

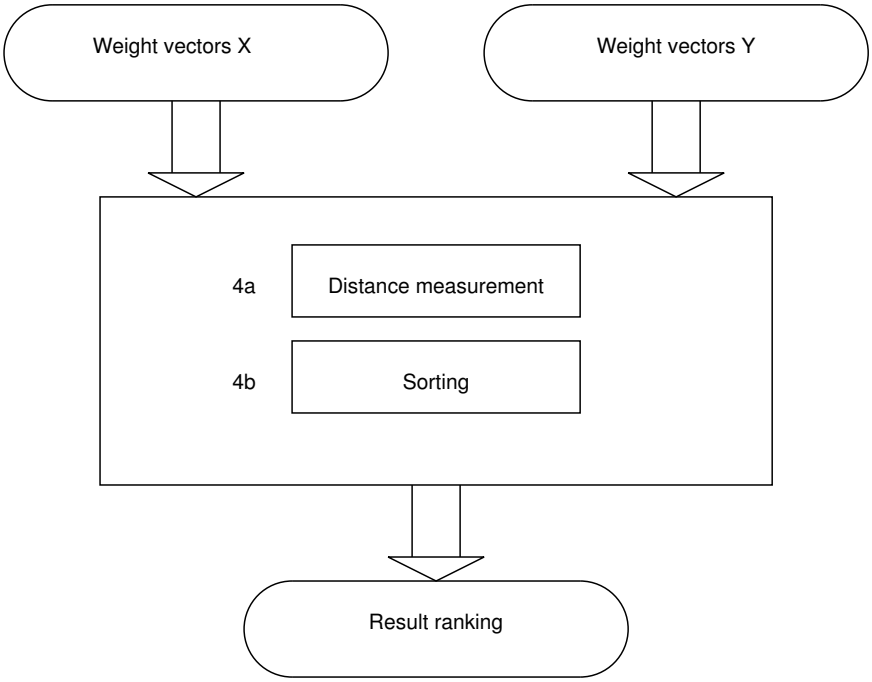

Figure 7.9: Ranking

The ranking part (figure 7.9) can be considered part of the testbed: it tests pairs of distance vectors (if the input is from a SOM) or pairs of document weight vectors (if the input is directly from the weighting step).

Step 4a, determines the distance between all pairs of the input (using euclidean distance, cosine similarity or correlation coefficient, as in section 2.4).

Step 4b sorts these pairs by similarity value, so it can be used for *precision-recall* measurement according to section 2.5.

Chapter 8

Tokenization

The first step in NLP is converting characters into symbols of word order. The process to do this is called *tokenization* [14]. The main questions which have to be answered are 'What is a (Dutch) word?' and 'What is a (Dutch) sentence?' Manning and Schütze [31] questioned this in general and for English in particular.

To get as much information as possible from a natural text I will need to use all possible syntactical information. This not only includes the sequence of the words, as trivial tokenizers do, but also punctuation marks. This information can be used for word relevance evaluation and can be used to produce better weight vectors for whatever further processing is performed.

A good source of information can be punctuation marks. Together with words, numbers, partial numbers, alphanumerical references (like A50) these marks can be used to determine whether a word starting with a capital is a (proper) name or just the start of a new sentence.

My first attempt was to create a tokenizer for the Dutch language. The final goal of this tokenizer is to detect all possible syntactic features of a general Dutch text on a character level. Features that have to be examined are summing up, detecting date and time.

The reason why I do not use a self-learning system for this task, is that the knowledge about the input can be modeled fairly straightforward with a finite-state-machine, as I will point out in this chapter.
The reason why I do not use the algorithms provided by others like Grefenstette [14], is that it seems like the topic of tokenization does not have a high priority within their work and it can be significantly improved.

8.1 Input Alphabet

The input alphabet is the ISO-latin1 character set.

The first thing I want to do is to partition it in separation characters (space, tab), lowercase and uppercase characters (these include accented characters), digits and illegal characters. The question mark and exclamation mark form one class. Other characters have a singleton class.

- *Dot, Question mark, Exclamation mark*
  Together with the question mark and exclamation mark the dot indicates the end of a sentence.

  The dot can also be used to separate numeric values in time and numerals ('10.000' and '12.45').

  The dot is also used for web addresses, titles and abbreviations ('www.ilse.nl', 'ir.' and

'ilsemedia b.v.'). A dot embodied in a word (like web addresses) can be easily detected (and registered as a name or proper name). Abbreviations and certainly titles can trick the tokenizer into entering an end-of-sentence state, 'ir. R. J. H. M. Essink' could be parsed as 4 full and 2 half sentences instead of 1 name.

- *Comma*
  The comma plays an important role in Dutch. It is used as separator of the decimal part in numbers. Also it is used more frequently for separating clauses.

  The comma can be used within enumerations, which could be detected, but would require a special processing tool (certainly a *FSA* does not have that capability).

- *Colon*
  A colon can be used as a terminating character, sometimes it may not. The presence of a colon with a slash might indicate a URL.

- *Semi-colon*
  The semicolon can be used as a comma, but also for special HTML characters like & In this case it is combined with the ampersand. As the HTML is already parsed, the & should not occur in the input.

- *Ampersand*
  The ampersand can be used as a substitute for 'en', but it is mostly used within proper names like 'V&D', 'AT&T' or 'Deloitte & Touche'.

  I will assume that an ampersand (possibly surrounded by spaces) is part of a proper name.

- *Quote/Apostrophe*
  The fact that one character has two names indicates that there should be a discrimination between quote and apostrophe.

  Detecting whether this character is a apostrophe or not is in theory not that difficult. In practice sometimes a space is missing between and ending quote and the next word, what makes it look like a apostrophe. The Dutch " 's morgens " can be considered as one token which makes this word more difficult to tokenize.

- *Hyphen*
  There are a lot of situations where a hyphen is used:

  - **for splitting syllables**
    A single word is divided over two lines using the hyphen.
    A different issue [42] arises when hyphenating Dutch words: the word 'autootje' is hyphenated to 'au-to-tje' which makes it very difficult to used standard hyphenation rules. The construction 'auto-tje' should be tokenized as 'autootje' and not 'autotje'. The same applies to 'zo-even', which becomes 'zoëven' and not 'zoeven', 'zoeven' has a totally different meaning. For these situations a special postprocessing function has to be built.

    Because of hyphenation the newline should be handled separately from space and tab (and therefore have its own character class).

  - **dash**
    The hyphen can be used to indicate a subsentence. In these cases it is pre- and succeeded by a spatial character.

  - **referring hyphen**
    When the hyphen is used to refer to a following word, for example 'pre- and succeeded'. It is very hard to process this to 'preceded and succeeded'. If the hyphen is succeeded by a newline it could be interpreted by 'preand succeeded'.

The only way to solve this issue is to get a feedback from the part of speech tagger in combination with a lexicon indicating that 'preceded' and 'succeeded' both exist and 'preand' does not.

For now I suggest this construction should be ignored and just parsed as the singleton token 'hyphen'.

- **minus sign**
  The hyphen is also used as the minus sign (but not too much in natural text).
- **names and numbers**
  The hyphen is quite often used as a separator for names or scores 'Floris-Jan schoot de 4-1 binnen'. In both cases the construction should be parsed as one single token 'Floris-Jan' remains a name, '4-1' becomes a alphanumeric reference.

  The problem is that the match 'Oranje Zwart-den Bosch' should be parsed as `(Oranje Zwart)- (den Bosch)` and not `(Oranje) (Zwart-den) (Bosch)`. This mix-up can be prevented; if a good notion of proper names is implemented, it could be parsed as a whole, i.e. `(Oranje Zwart-den Bosch)` .
- **hyphen composed expressions**
  Some words are composed by hyphens to a indivisible expression, like 'e-mail' or 'data-base'. Splitting the expressions to its atoms will ruin the expression.

  Complete sentences or expressions that are grouped to one word, like 'duizend-en-één', 'weg-van-de-snelweg'. The problem is that these expressions look a lot like the earlier examples from a syntactic point of view. But if they are split their meaning is not completely lost.

One serious question is, whether two hyphen-composed words should be split into three parts (two words and a hyphen) or the construction should be considered as a whole. This is up to the designer to decide. Splitting the construction into two word tokens and a hyphen-token will increase the recall of the system, keeping it together will improve precision.

A strategy where hyphens are only processed as a separate character if they are preceded or followed by a spatial character seems as the best solution to handle hyphens. It will cause expressions to be parsed as one token, but this can also be explained as an advantage.

- *Spatial characters*
  The space, tabular and newline character will only be used for the tokenizing process, but will rarely appear in the output. The exception on that are names ("ir. R. Essink") and special constructs like "'s morgens".

  The only need to explicitly note a spatial character is to express empty lines or sentences starting on a new line, which can indicate a new section or paragraph. With this, abstracts and excerpts could be marked. As this is author dependent, it is better to determine this from the text itself. Besides such a detection is (like enumeration detection) not feasible with an *FSA*.

- *Other special characters*
  The '<' less than and '>' greater than character can be used to discriminate HTML, SGML or XML tags.
  These can be excluded from the input, parsed as ordinary symbols (LT, content, GT) or mapped to one token (HTMLTAG).
  The question what to do with this is up the the programmer to decide. I don't need the HTML tags, so I will ignore them.

  All other characters can be ignored or represented as single tokens.

8.2 Abbreviations and titles

As already stated in 8.1, dots can be used for end-of-sentence, numerals, abbreviations or titles.

The last category is the hardest to detect. A good example of two abbreviations is 'Joh. Enschedé & Zn.' Therefore a list of Dutch titles and abbreviations should be used by the tokenizer. This can be extended with a few name abbreviation detections like the 'Joh.' for 'Johan'; of course plain 'J.' should also be identified as an abbreviation.

Another possibility is to detect the likeliness of being a normal (Dutch) word or a potential abbreviation. This can be done with a IDF table (see section 2.3.2), but if tokenization is a preprocessing step for creating such a table (as in this situation) using an IDF table is not a serious option. Terms like 'etc' are less likely to be a word than 'etcetera' although the latter is uncommon, so a check on the length is also required for such a detector. Anyhow, this cannot be implemented or is very hard to implement in the *FSA*.

8.3 Output Alphabet

It is also useful to determine what kind of information I wish to produce about the input, to put it in another way: 'What is my output alphabet going to be?'

- *Words*
  Words are a sequence of lowercase characters which may only start with an uppercase character at the start of a sentence. An exception on this rule is the 'IJ' combination in 'IJsje'.

  Words may contain a hyphen ("populair-wetenschappelijk"), an apostrophe ("auto's") or even a space ("'s morgens").
  The last exception is a typical Dutch construction. If "'s morgens" is put at the start of a new line it becomes "'s Morgens", which makes the tokenizing process even more complicated.

- *(Proper) Names*
  Proper names can be detected by their name or by the fact that they usually start with an uppercase character, or have one included, as with "Val d'Isere" or "eBay". If the position of the word is at the beginning of a sentence

  To increase the chance that a proper name is properly detected I will need a dictionary of proper names found within the context of the article, as the odds are high that a proper names has multiple occurrences within a text, book or corpus. Also, using a corpus/name lexicon will ensure that the name 'ilse', which doesn't start with a capital-i will be noted as a proper name. This issue is similar to the detection of abbreviations, and cannot be implemented with the *FSA*.

  Proper names can be categorized into first names, surnames, celebrity names, company names, etc.

- *Digit sequences*
  The set of terms with only digits, $\{'0', \ldots, '9'\}^*$, is called the digit sequence. Numbers can also contain punctuation marks like with '10.000', '3,141', etc.

- *Generic Numbers*
  Generic numbers are digit sequences polluted with punctuation marks. The class includes decimal numbers (3,141), time (15:51), time intervals (2:08,01), dates (10/12/2003) and telephone numbers (040-2473402).

  One could try to discriminate between these tokens and write them to a unified form; for example, time is always separated by two colons and without dots. This is a possible future feature.

The sentence 'met een aantal standaardonderdelen (zie figuur 8.8).' Is tokenized into

```
<word>met</word>
<word>een</word>
<word>aantal</word>
<word>standaardonderdelen</word>
<word>zie</word>
<word>figuur</word>
<number>8.8</number>
<eos/>
```

Figure 8.1: Tokenization Example

- *Alphanumeric Reference*
  A token from this class is not a true name, not a word, not a number and not a punctuation mark.

  Examples of alphanumerical references are numberplates ('44-BX-12'), road identifiers ('A50'), additive codes ('E-350'), ISBN numbers ('0-201-39829-X'), etc, etc.
  These examples show that these references are quite common, and even can play an important role within information retrieval.

- *End-Of-Sentence*
  A dot, question mark, exclamation mark or colon can indicate and end-of-sentence but that does not always need to be the case.

  To indicate that, an *EOS* token will be sent (after the specific symbol).

8.4 Tokenizer design

Looking at the amount of remarks pointed out earlier, the design of a tokenizer for Dutch is not a trivial task.

Each character is mapped to a $classid = \{$ **letter**, **digit**, **spatial**, ...$\}$, the set of states is { **start-of-word**, **in-name**, **in-number**, ...}.
Using a limited number of values enables me to create a finite state transition function $classid \times state \mapsto state$ implemented in a reasonably compact matrix (because the input alphabet is partitioned).

In some cases, extended checks have to be performed, this is also noted in the transition matrix by adding exceptions to the output value of the transition function ($classid \times state \mapsto state \times exception$).

The output tokens are created in XML format as described in figure 8.1.

8.5 Conclusion and Remarks

The finite state automata is a very effective tool for word tokenization. If it can be extended with exception handlers or postprocessing functions (like abbreviation detection); it shows very natural word decompositions.

Creating a natural language tokenizer is not fully independent of its language. Date and time can be stored differently; just as with numbers.

Chapter 9

Building Document Vectors

The second stage of document processing is converting words into *term weight* vectors (see section 7.2). First, all word-document occurrences (postings) are counted and stored in an *index* like in table 7.5. From this *index term weight* vectors (for example table 7.6) are computed.

Some words are less relevant. My choice is to use the *document frequency* to determine the word relevance (and not by using a table of *stopwords*). Stopwords tend to dominate the list of most frequent words in the *corpus*. The other words in that list are words that are specific for the *corpus*, but, like stopwords, do not contribute to the semantics of individual documents within the *corpus*. In the sports section, for example, the word 'sport' does not add anything to the meaning of documents, but it cannot be considered as a stopword in general.

Word frequency should be within a certain interval: frequent enough to bind a number of articles (at least more than one) but not too frequent to be regarded as common. These two values are the pruning thresholds.

Pruning, as described in section 2.2.1, is supposed to reduce noise while not removing relevant *terms*. To determine the optimal values for the NU.nl *corpus* I performed a few *recall-precision* tests (described in section 2.5 and chapter 12) on a set of 400 documents.
The results in figure 9.1 (which are a subset of the intervals tested) show that removing common words (lowering the upper threshold) generally has a positive effect (the *precision* increases for each *recall* value) with [2..50] as optimal setting. Removing rare words (raising the lower bound) has a negative effect, where [20..400] is the worst interval. The default test is the interval [0..400] which is the curve in the middle.

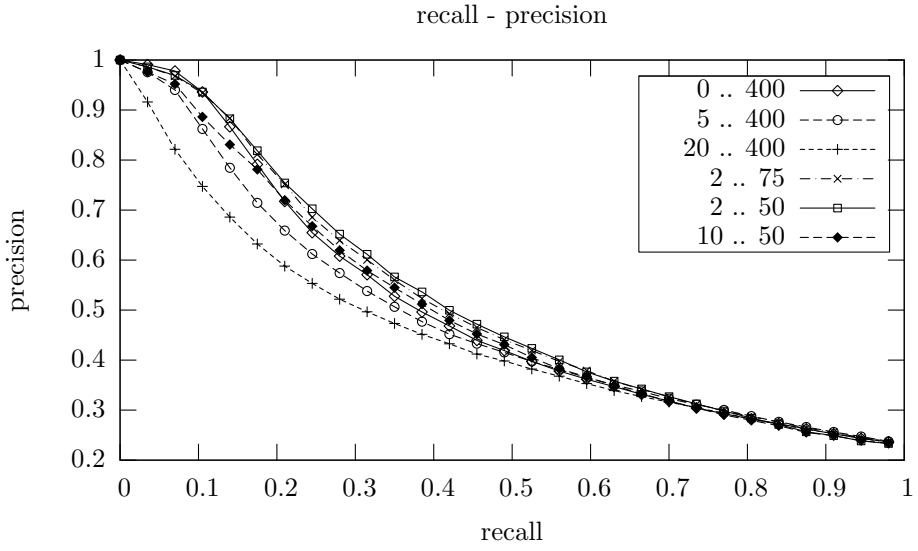

Figure 9.1: Effect of pruning on precision/recall

9.1 Optimal Weighting Function

In section 2.3 a number of possible configurations for *term weight*ing functions are listed. In section 2.3.1 five options for *local term weight* are mentioned:

a Alternate log

b Binary weighting

i Inverse frequency

l Log Term Frequency

t Term Frequency

To determine the (if there is one) best *local term weight* function these five methods were compared in figure 9.2. In this test the *global term weight* was IDF and the similarity was computed with the *cosine similarity*.

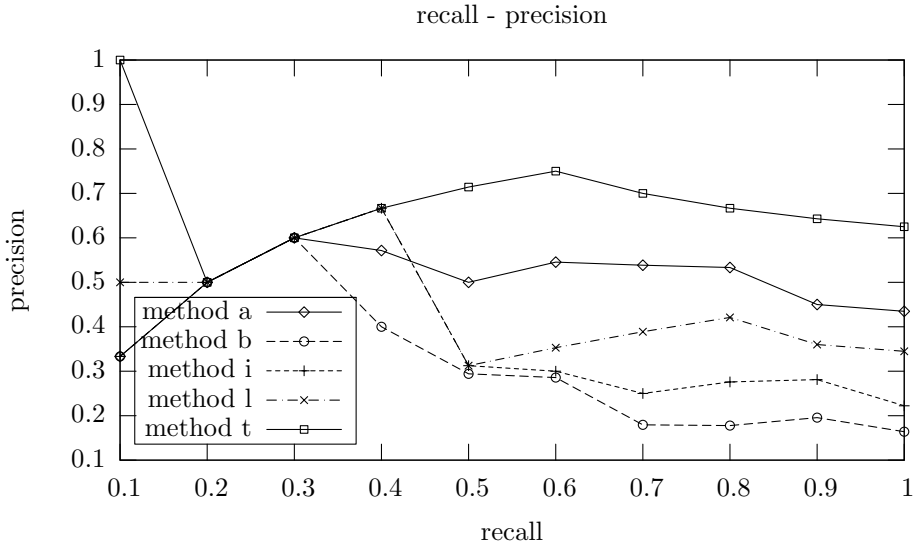

Figure 9.2: Query on the corpus using different local weights

Although the figure only shows one experiment, others gave similar results: a slight favor for the Term Frequency method.

As a rule of thumb we can conclude that Term Frequency is the best, whilst being also the easiest, but the differences are negligible.

Testing all possible local and global configurations will result in a research project by itself (like in [48]). Besides, I am convinced that a test with the global weighting functions will provide a similar result (meaning that differences are negligible), mostly because the *entropy global term weight* function was also implemented, but did not show surprisingly different graphs.

9.2 Optimal Similarity Function

To determine the distance (or similarity) between two vectors one has to choose between the *cosine similarity* and the Euclidean distance. The figures 9.3 and 9.4 compare recall-precision for the *Euclidean distance* and the *cosine similarity*. All articles in a corpus were tested against 1 specific article.

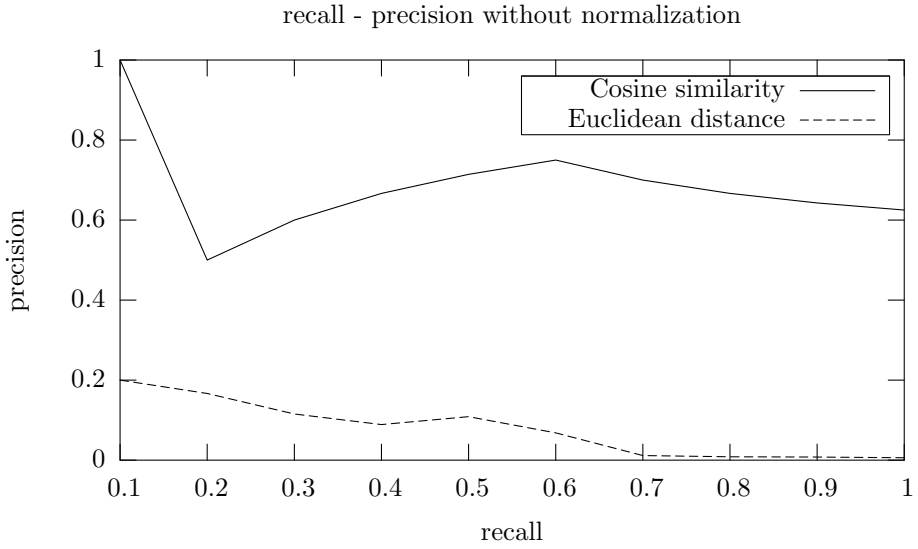

Figure 9.3: Recall-precision graph without normalized input vectors

The results of the second test (in figure 9.4) showed that on an index of normalized vectors, the *cosine similarity* and *Euclidean distance* perform equally well.

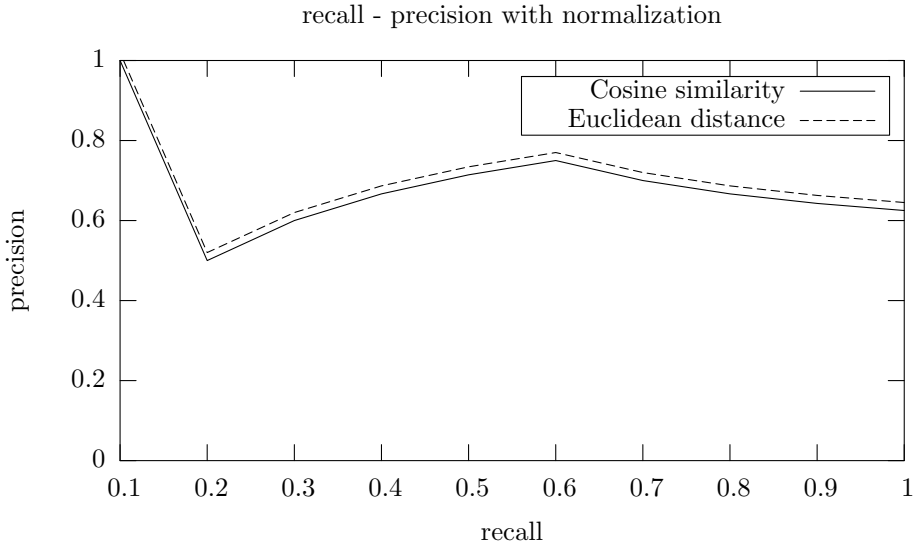

Figure 9.4: Recall-precision graph with normalized input vectors

9.3 Relation between similarity functions

So there is a relation between the cosine function and the Euclidean distance. If the two vectors x and y are normalized, i.e., $x' = \frac{x}{||x||}$ and $y' = \frac{y}{||y||}$ then

$$
\begin{aligned}
& \|x' - y'\| \\
= & \\
& \sqrt{\sum_i (x'_i - y'_i)^2} \\
= & \\
& \sqrt{\sum_i ({x'_i}^2 - 2x'_i y'_i + {y'_i}^2)} \\
= & \nleft \sum_i {x'_i}^2 = 1 \nright \\
& \sqrt{2 - 2\sum_i x'_i y'_i} \\
= & \\
& \sqrt{2}\sqrt{1 - sim(x, y)}
\end{aligned}
$$

So the reason why *cosine similarity* is recommended in articles ([29],[36]) about information retrieval is its normalizing effect (apart from the computational advantage on sparse vectors).

Without normalization, the *Euclidean distance* does not perform that well. Given that the frequencies are responsible for the length of a weight vector, figure 9.5 shows that the impact of frequencies is larger than the impact of word correspondences.

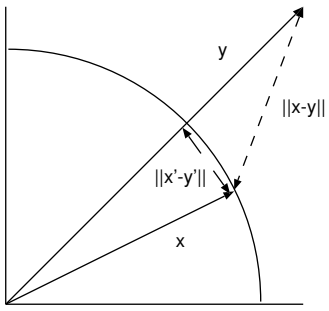

Figure 9.5: Euclidean distance between normalized and unnormalized vectors

A good question is: 'What is the meaning of the *Euclidean length* of a weight vector?'
What causes a vector to be longer than another? Does it depend on the corpus (the global weight might differ with the corpus size)? It surely depends on the size of the document represented by this query (more words means more non-zero elements in the vector).

My first thought would be that there is a direct connection between the length of a document weight vector and the number of words in the matching document. A second thought is that if a document would use a different, more uncommon vocabulary, it will have higher *idf* weights. But if that would be true, that document might not be considered part of the same corpus.

Figure 9.6 shows the relation between the Euclid length (the normalization factor) and the number of word instances (not the number of unique words) in a certain document.

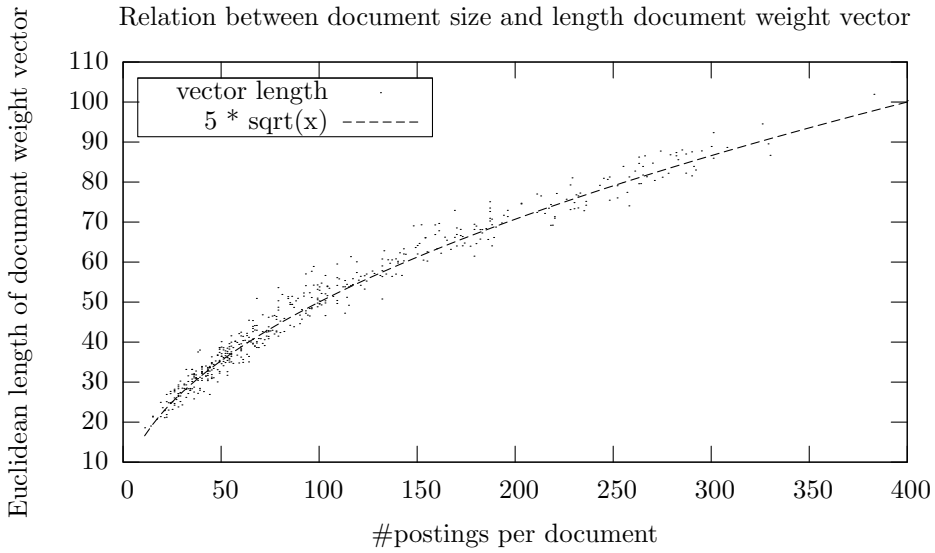

Figure 9.6: Document length vs. document vector length

This figure is not very hard to explain: suppose a document z is made of two chapters x and y with an exact equal amount of word instances. Therefore the assumption that $\|w_x\| = \|w_y\|$ is not unreasonable. Because of the way the weight vector is constructed one may also assume that $w_z = w_x + w_y$

$$
\begin{aligned}
& \|w_z\| \\
= & \\
& \|w_x + w_y\| \\
= & \nleftarrow cosinerule \nrightarrow \\
& \sqrt{\|w_x\|^2 + \|w_y\|^2 + 2\|w_x\|\|w_y\| cos\alpha} \\
= & \nleftarrow \|w_x\| = \|w_y\| \nrightarrow \\
& \|w_x\| \cdot \sqrt{2} \cdot \sqrt{1 + cos\alpha}
\end{aligned}
$$

$cos\alpha$ is the *cosine similarity* between chapter x and y. So this curve depends on the coherence between words, lines, sections, etc. within the document.

9.3.1 Cosine similarity and Correlation Coefficient

If there is very little difference between the *cosine similarity* and the Euclidean distance there might also be a relation between the correlation coefficient and the *cosine similarity*.

The manual of a clustering document *Cluster 3.0* [11] suggests a correlation coefficient where the mean of both stochats is assumed zero. A simple derivation shows that this is nothing less than the *cosine similarity*:

$$
\begin{aligned}
& \rho_{x,y} \\
= & \\
& \frac{E(x-\mu_x)(y-\mu_y)}{\sigma_x \sigma_y} \\
= & \\
& \frac{E(x-\mu_x)(y-\mu_y)}{\sqrt{E(x^2)-\mu_x^2}\sqrt{E(y^2)-\mu_y^2}} \\
= & \lessdot \mu_x = 0, \mu_y = 0 \gtrdot \\
& \frac{\frac{1}{N}\sum_i x_i \cdot y_i}{\sqrt{\frac{1}{N}\sum_j x_j^2}\sqrt{\frac{1}{N}\sum_j y_j^2}} \\
= & \\
& \frac{\frac{1}{N}\sum_i x_i \cdot y_i}{\frac{1}{N}\sqrt{\sum_j x_j^2}\sqrt{\sum_j y_j^2}} \\
= & \\
& \frac{x \cdot y}{||x|| \cdot ||y||}
\end{aligned}
$$

With sparse vectors (such as weight vectors!) $\mu_x = 0$ is a pretty safe assumption.

Chapter 10

Dimensionality Contemplations

One of the values that are configurable is the dimensionality (the size) of the output, the size of the grid.

As the goal is to perform some kind of generalization (assuming that synonyms are more common than polysemants) the number of outputs should be lower than the number of articles, or at least not much higher.

The pictures 10.1 and 10.2 illustrate that this might not be the case.

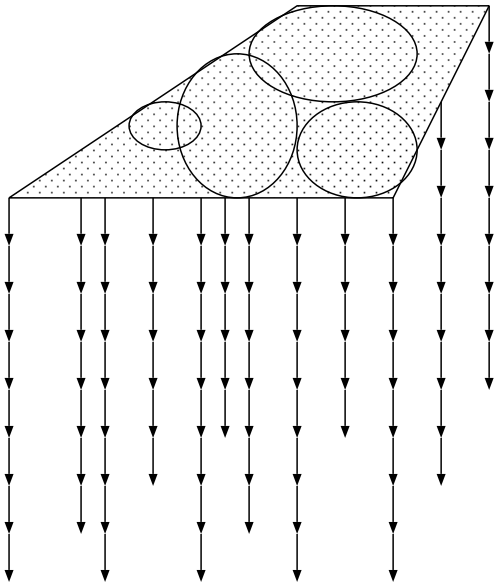

Figure 10.1: Small SOM with large inputs

It might be possible that our input is too diverse to form a nicely structured clustering. Figure 10.1 shows overlapping clusters, not necessarily by articles relating to both clusters, but because of insufficient space.

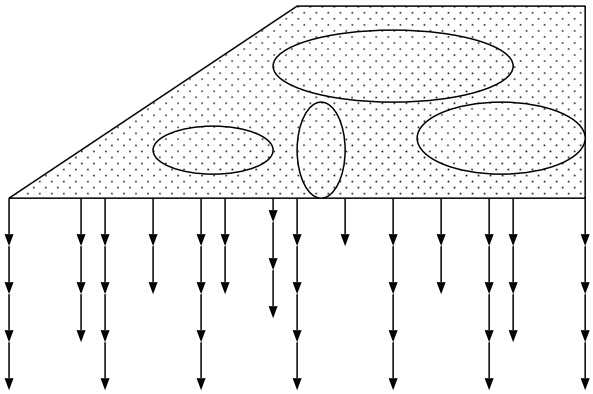

Figure 10.2: Large SOM with small inputs

By enlarging the grid (shown in figure 10.2) the risk of unrelated documents being clustered is much smaller.
So there has to be enough space for all articles to find a matching reference vector.

10.1 Random Matrix

Training several thousands of vectors with dimensionality O(10000) causes tremendous computational and memory requirements. If these vectors are entered raw into the neural net, time and memory consumption increases dramatically.

In [26] a solution is provided by the use of random matrices (also Glossary *random matrix*).

A vector n of dimensionality N can be reduced to a vector x with dimensionality $d << N$ by computing $x = Rn$. The elements R_{ij} of random matrix $R \in \mathbb{R}^{N \times d}$ are fully random over the range of $[-\alpha..\alpha]$ for any $\alpha > 0$, as long as

$$\forall_i \sum_j (R_{ij})^2 = 1$$

The value of d depends of how much computer memory and CPU time you wish to save and how much loss of signal quality you are willing to tolerate.

10.2 Segmented Random Matrices

The memory consumption of a random matrix is O(Nd) which can become quite large: with the NU.nl corpus N has order of $O(10^3)$ up to $O(10^6)$ and as a result d has a value of a smaller order (but still reasonably large).

The solution to this problem is to reduce the number of non-zero elements in R: for each vector R_i there is only a small segment or interval where $R_{i,j} \neq 0$ However, the precision can be reduced by this as well, this is also a matter of performance vs. quality.

Chapter 11

Applications

The concept hash is developed as a way of representing text into a vector format which is easy to compare, therefore it can be applied in a number of applications. I will mention two.

11.1 Related Documents Search

The easiest way to use this system is to search for documents within the corpus that are related to a query document. With news articles this can be used for suggesting which articles the reader also could visit for further information. Chapter 9 has already shown some tests with document similarity.

Another way to use related documents is to search for duplicates (or documents with a strongly related content). Duplicates (or near duplicates) already have a high similarity when the TF-IDF method is used, but the SOM may find the less trivial correspondence's.

11.2 Searching Words

The second question is whether it is possible to build a (small) search engine with this technique and how it will perform against a regular search engine.

An issue that has to be solved is how to create the query vector from a sequence of query words. This can be solved by using *idf*, in the same manner as in section 2.3.3, or by assigning weights to the query words, or both (the assigned query words are regarded as local weights and are multiplied with the idf global weight).

For example the query 'lecture:2 computer:5 engineering:1' searches for documents with an accent on computer. Let the dimensions for 'lecture', 'computer' and 'engineering' be i,j and k; then for the resulting query-vector q will yield $\forall_{l \notin \{i,j,k\}} q_l = 0$; $q_i = 0.36$; $q_j = 0.91$; $q_k = 0.18$.

Chapter 12

Testing & Results

The purpose of testing was to compare the SOM - approach with a traditional document similarity search. Given D, the set of vectors (produced by either TF-IDF or by the SOM) and $R \subseteq D \times D$, the set of all document combinations which are considered similar. Determining R is relatively easy, certainly if D is only the sports section of NU.nl (R is the set of all document pairs about the same sport). $|D| = 100$, $|R| = 998$.

A test means that each pair in $\{(d, d')|d, d' \in D, d \neq d'\}$ is compared with both methods and sorted by similarity. From this list the *precision-recall* graph is produced according to section 2.5.

In some tests, a random matrix is used, therefore the TF-IDF curve varies (the test and result set are constant in all tests), but this variation is negligible. Figure 12 is the precision-recall graph where the SOM is configured with radius=3; in figure 12.2 a radius of 6 is used. The size of the SOM grid is 9×9.

In the *precision-recall* graphs the highest scoring results, retrieved by one of the methods, are positioned in the upper left corner, where the least scoring results are positioned in the lower left corner. If a system performs well the precision remains relatively high.

The first two tests, where all vectors are compared with the cosine similarity, show that the SOM is generally better when finding the less obvious results. At a lower radius the SOM would perform generally poorer (it performs like a simple dimensionality reduction).

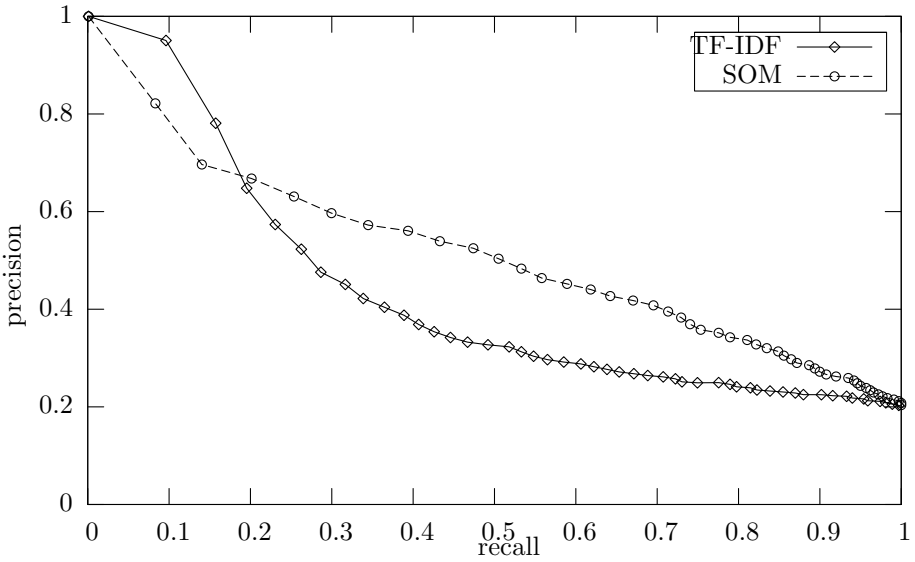

Figure 12.1: Sports Section radius 3

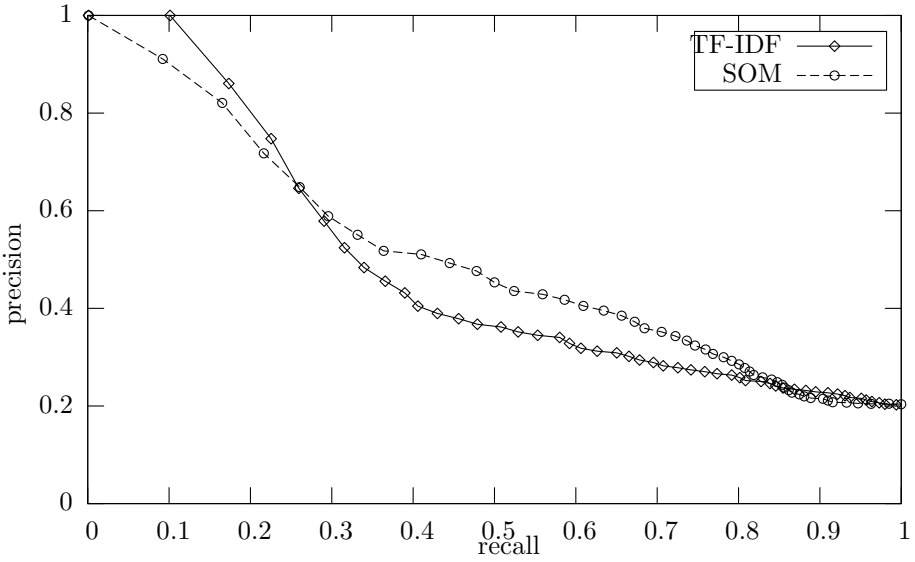

Figure 12.2: Sports Section radius 6

The third test in figure 12.3 is performed with both cosine similarity and correlation coefficient (abbreviated with R). This graph really made my day: it shows that the SOM keeps a high precision while recall increases, provided the correlation coefficient is used for the similarity function.
The lack of improvement of the TF-IDF method (called TERM in this graph) with the correlation coefficient can be explained by the average value of the vectors: the SOM output vectors have an average value close to 1, where the TF-IDF vectors are closer to zero. Section 9.3.1 explains the difference.
To verify that the advantage is not the effect of solely the correlation coefficient, random value vectors were used as reference vectors. Their results are called BEACON. They show a remarkable improvement between the cosine similarity and correlation coefficient, but cannot compete with the results of the SOM.

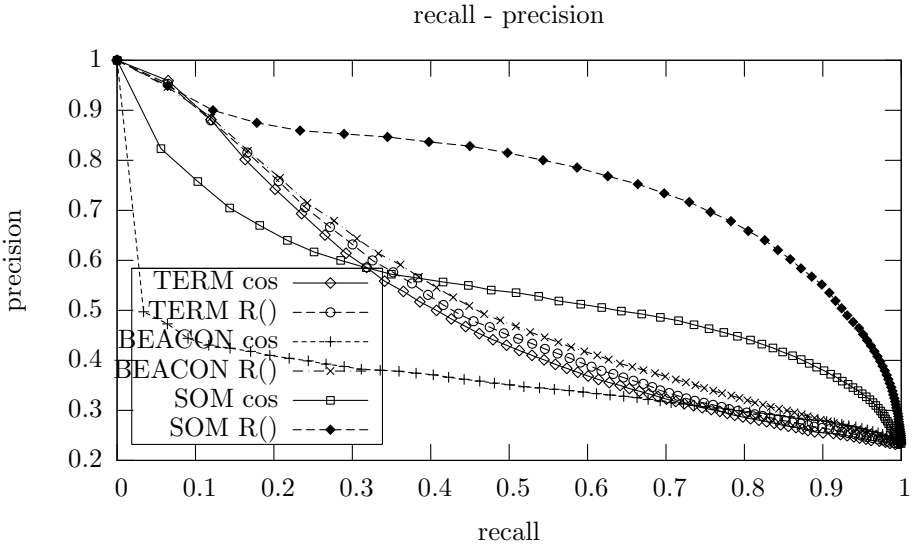

Figure 12.3: Sports TERM/BEACON/SOM method

Chapter 13

N-Gram Solution

During an assignment (Appendix C) to examine a language identification function I encountered the surprisingly strong effect of the Markov assumption [1].

The idea is that words are only a sequence of characters, and that it might even be better to use (small) fixed character sequences (N-grams) instead of words.

N-grams can also overlap, which makes it easier to relate derived words. For instance 'computer' and 'computers' which would not be considered equal words have (if limited to 3-grams) 'com', 'omp', 'mpu', 'ute' and 'ter' in common.

Another advantage to N-grams is that the dimensionality is limited by Σ^N where Σ is the accepted alphabet.

In a language, like Dutch or German, with a lot of composed words, this is an advantage, but other languages can also benefit from the fact that conjugations of the same word can be linked.

13.1 New experiment

The experiment is not really different from the test with words. Only the word tokenizer is changed for a ngram tokenizer.

The SOM has in these experiments a radius of 2 (which explains its relative poor performance) and a grid of 25×25 .

The results in the figures 13.1, 13.2, 13.3 and 13.4 are samples of a few tests. In figure 13.1 the 'old' words experiment is displayed. The bigram experiment (figure 13.2) shows a slightly better performance for the SOM. But the TF-IDF does worse. With the 4-grams (figure 13.4) TF-IDF shows a better curve. Both systems do poor on trigrams (figure 13.3).

13.2 Conclusion

Ngrams show a reasonable alternative for words, but it is not perfect.

Pruning of ngrams is something that needs to be looked into. So is the use of spatial characters in N-grams (to indicate the start and end of a word).

[1] The k^{th} order Markov approximation is $P(X_n = x_n|X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) = P(X_n = x_n|X_{n-1} = x_{n-1}, \ldots, X_{n-k} = x_{n-k})$: an element depends only on k of its predecessors

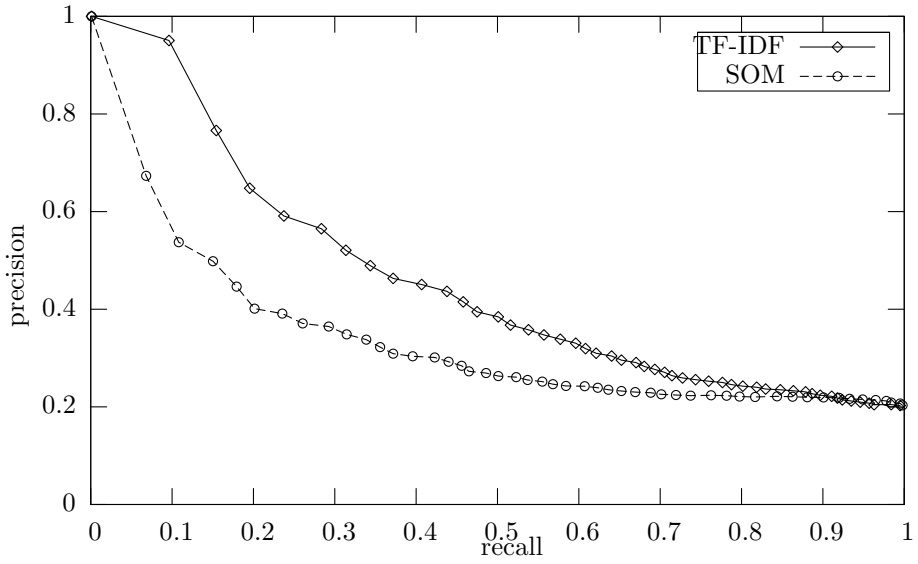

Figure 13.1: Precision-Recall graph with usage of words

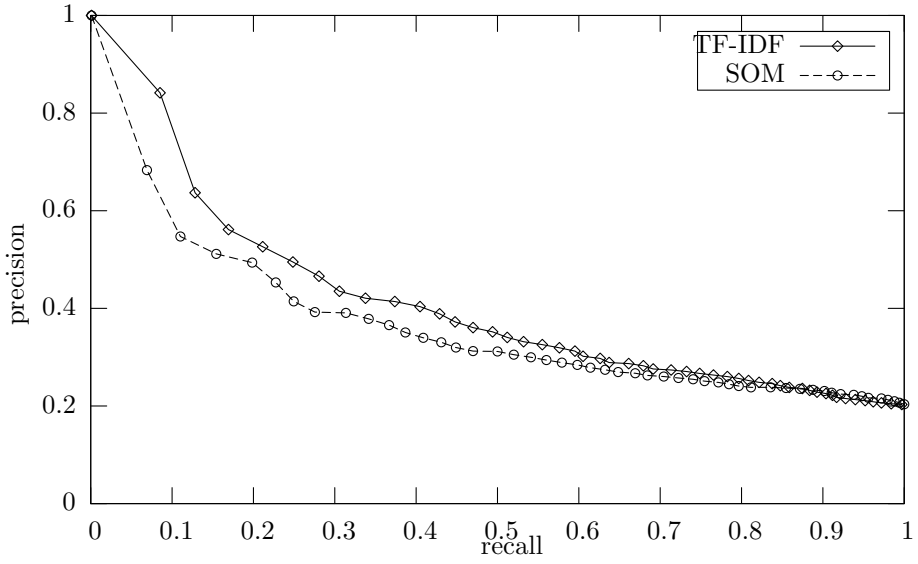

Figure 13.2: Precision-Recall graph with usage of 2grams

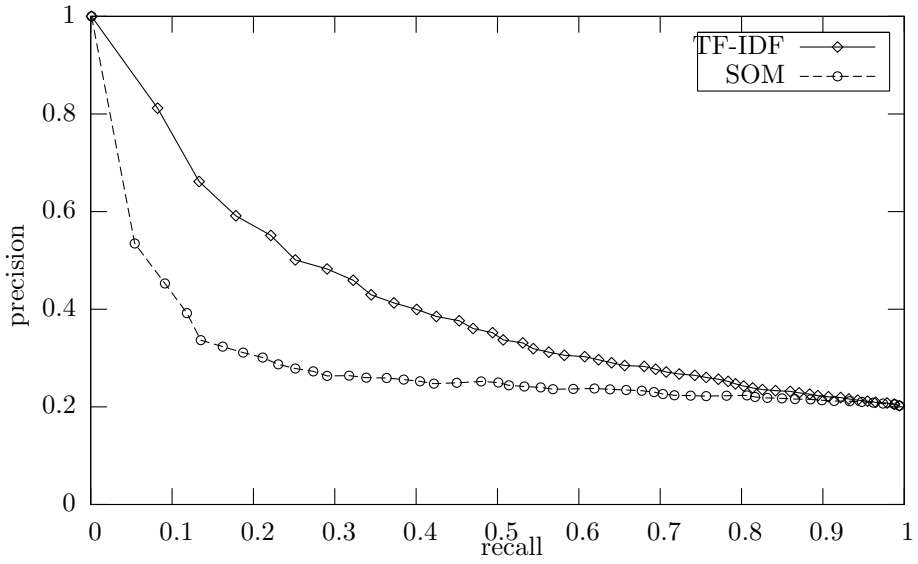

Figure 13.3: Precision-Recall graph with usage of 3grams

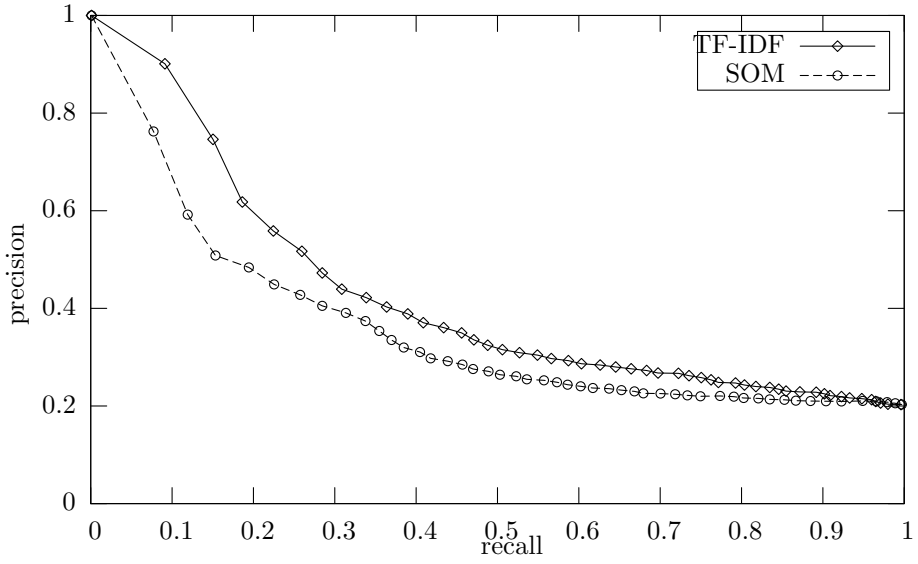

Figure 13.4: Precision-Recall graph with usage of 4grams

Chapter 14

Developed Software

On behalf of speed and memory consumption I chose C/C++ as programming language for developing my experiments.

14.1 Parsing

input xml parsing

The input is an XML stream (like in figure 7.2), where each element is a piece of HTML code. The XML is parsed with the GNOME libxml2 [39] library, the HTML tags are removed with a small finite state machine.

word tokenizer

The word tokenizer is an implementation of section 7.1 and chapter 8. Each word is converted into an XML token. So the output of the parsing section is again an XML file (as exemplified in figure 7.3).

14.2 Indexing & Weighting

token input

A small XML parser based on the expat library [9] is used to read the HTML/word parser output. Depending on a configuration flag the word is stemmed or not. Every word in the input is then stored into a word index.

word index

The word index stores words and their frequency in each article. Which equals the process described in section 7.2. After all words have been counted, pruning can be applied to remove hyper frequent and hypo frequent words. After pruning, the frequencies are converted into weights.

FIsparse

FIsparse is a hashtable for implementing a sparse word index ($\#words \times \#articles$ tends to become extremely large).

vector base

In the NeuroBor library (Appendix A) the NeuroVector, a vector object to compute with weights, is implemented. The vector base is an array of neurovectors.

queryBase

The queryBase loads and processes queries against a wordindex and creates a vector base of query vectors as specified in section 11.2. Unfortunately this code has never been used.

14.3 Filtering & Evaluating

weightfilter

A weightfilter takes a vector base or word index and converts it into a new vector base using a filter method. An example of a weight filter is the random mapping.

random map

A random map is a weightfilter implementation of a Random Mapping (chapter 10).

Kohonen implementation

The Kohonen implementation (koh_impl) is an interface layer between the main routine and the selected neuro library, in this case it is the NeuroBor library from Appendix A.

main program

A main routine and a configuration object class combine the classes above into a program.

14.4 SOM Library

There are multiple implementations of the SOM network. From them I chose the JNeural library [19] because it is programmed in C++, well documented and easy to incorporate. An alternative could have been the SOMPak software [27], which is less easy to incorporate or modify. Also SOMPak has dimensionality limitations.

JNeural did not turn out to be what I expected, some features of the SOM concept like neighbourhood were poorly implemented, so I decided to write my own SOM library (Appendix A). This library also has better memory usage, so more data can be processed, before the process runs out of memory. It is still better structured than SOMPak.

Chapter 15

Conclusions

15.1 Answers

Referring to the questions in the introduction:

- **Is it possible to create a concept hash of a document?**
  The answer is certainly yes. The developed system might not seem as powerful as hoped when I started this project, but it proves to be able to do some generalizations and relate documents by learning.

  The stereotypes concept (chapter 6) looks like a good framework to fit the SOM paradigm and leave some room for other algorithms (see next section).

- **Is it possible to create a concept hash of a query?**
  Yes. Although this is not tested, section 11.2 provides the recipe for creating a vector of *term weights*. This vector can be converted into a concept hash

- **How can I compare two concept hashes?** The *correlation coefficient* is the preferred way to measure the distance between to vectors.

- **How well does the system with the SOM perform?** It is not fast, but the results are significantly better than the normal *TF-IDF* scores.

  From that I conclude that the SOM is capable of exceeding the syntactic level of the documents and truly adds to the 'interpretation' of the documents.

A question that was not asked, but is interesting to answer, is 'Is it possible to extract the underlying concepts from a hash?'
The answer is yes, provided that the SOM was capable to generate useful clusters. Considering the NU.nl sports section, the SOM was capable of doing so. It can even be used to extract the keywords related to that concept, by comparing the values in the reference vectors to the reference vectors of other neurons.

15.2 Future Research

The details of the concept and keyword extraction mentioned above is something that can be considered future research.

It might be possible that using this system in combination with other (possibly supervised) techniques might provide better results.

From tests with the system in appendix C can be concluded that supervised training is a strong guarantee for reasonably good results. As unsupervised training is one of the conditions of this thesis this is an open issue. A combination of both supervised and unsupervised techniques might prove to be better than both systems separately.

The SOM could be substituted by faster and less memory exhausting (clustering) algorithms. Possibly such algorithms may provide comparable results, but in less time. PCA is also an algorithm what could be used for this purpose.

Chapter 16

Advised Reading

The research areas of natural language processing and information retrieval have produced a vast amount of literature worth reading.
Most of these books are rarely lent at our university because of the closure of the IPO a few years ago, so they're waiting for you.

16.1 Natural Language Processing

'Foundations of Statistical Natural Language Processing' [31] is an extensive book on natural language processing from a mathematical point of view.

'Natural Language Processing' [36] is a good book when you can't afford (or carry) [31]. It has a lot of bibliographical references, so a good book to start with.

'Computer-taalkunde' [10] and 'Rekenen op Taal' [4] are two pleasantly readable (dutch) books on the earlier methods of natural language processing.

16.2 Information Retrieval

'Understanding Search Engines' [6] is a short but still comprehensive book on processes that are performed by search engines. It is even smaller than [36] and very useful introduction to IR.

'Modern Information Retrieval' [5] is a more generic work on Information Retrieval. It contains few formula's and more practical implementations of IR.

'Information Retrieval' [44] is a real classic, but still very interesting reading.

'Statistical Language Learning' [8] is a good book on using Markov theory on natural language.

16.3 Neural Networks

'Artificial Neural Networks, Theory and Applications' [35] is an accessible book which covers most neural network types.

'An Introduction to Neural Networks', from Gurney [16] is also very accessible as it is downloadable.

Further you can read 'Fundamentals of Artificial Neural Networks' [17] or 'Neural Networks' by Haykin [18].

Appendix A

NeuroBor SOM library

The NeuroBor SOM library is an implementation of a Self-Organizing Feature Map (or Kohonen Map) in C++.

A.1 Criteria

Criteria for this library are CPU and memory efficiency, scalability and applicability.
The JNeural library [19] has the disadvantage that it uses too much memory overhead. The SOMPak library [27] is not capable of handling large vectors. And writing your own library means it connects very well with your existing software.

A.2 Design

The library has three classes (in bottom-up order):

A.2.1 NeuroVector

The NeuroVector is an implementation of a vector of fixed dimension. The type of its elements is NeuroVectorElem, which can be a float or double.
A vector can be updated with another vector by a certain scale (alpha). The distance between two NeuroVectors can be computed if dimension and distance function for both vectors are equal.

The NeuroVector class is an abstract for StaticVector and SparseVector. The first one is implemented with a real value for each element, the second with a coordinate and real value for each non-zero element.

A.2.2 NeuroGrid

A NeuroGrid is a matrix of NeuroVectors. The NeuroGrid has two iteration methods: to iterate over all nodes in the grid, or all nodes in a certain radius from a specific node. Where a node equals a NeuroVector.

A.2.3 SOM

The SOM is a NeuroGrid with training and querying methods.

Training is possible in diamond and square shaped neighbourhoods (and possibly hexagonal). The problem with the JNeural library was that it did not take the distance to the winning neuron into account.

A.3 Implementing

The NeuroBor library is implemented in C++, but with a high level of plain C code (no usage of standard C++ (STL) functions, like hashmaps, vectors, etc. , and no separate class for neurons). This static way of programming makes the NeuroBor library substantially faster and more memory efficient then the JNeural library.

The disadvantage of the NeuroBor library is that it implements the SOM concept (mostly the neighbourhood part is better implemented) a bit more accurately than JNeural and therefore loses some of its speed (but not all).

Appendix B

Word/postings ratio

When processing more and more articles, the size of the vocabulary (the number of unique words) increases. This implies an increase in vector dimensionality, and therefore in time complexity.
I wish to determine how the number of articles influences this time complexity.

First, there is a factor between articles and the number of postings (word instances): simply the average article size. In case of NU.nl news articles this proves to be 204.

There is also a function from the number of postings to the number of unique words.

Graph B.1 shows the number of unique words versus the number of postings in a NU.nl news feed. As a reference, the formula $14.2 \cdot x^{0.6}$ is printed.

The function from number of posting to unique words is of the order $\sqrt{x}$. This relation is based on Heaps' Law [5].
This means that an index of N documents uses N vectors of dimension $O(\sqrt{N})$.

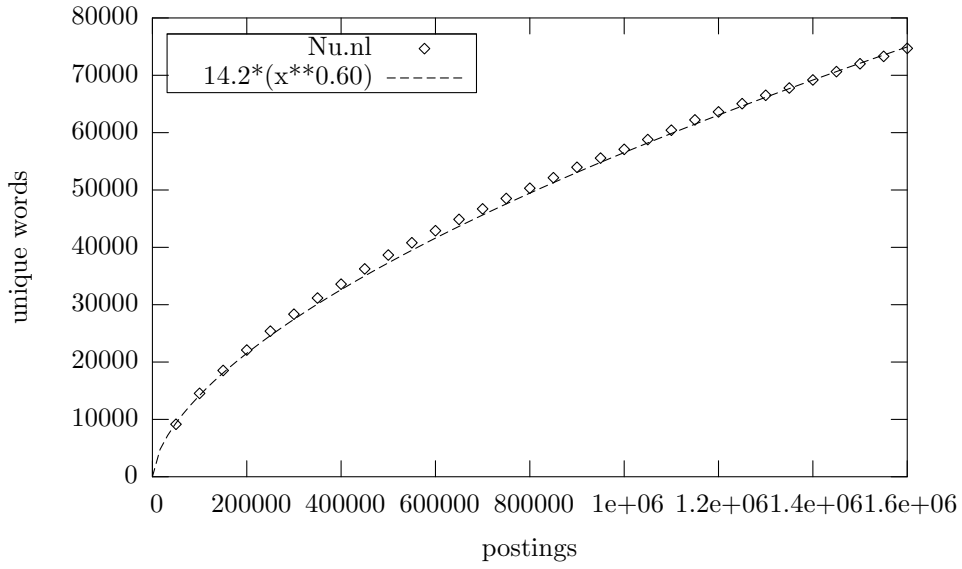

Figure B.1: Word/postings ratio in index

This graph will differ with other corpora. The difference with other news sites (like the news articles on the website of regular newspapers) tends to be minimal.

With literature (like 'Max Havelaar'[34], figure B.2), the somewhat larger difference can be explained by the lack of new names, cities, terms and definitions.

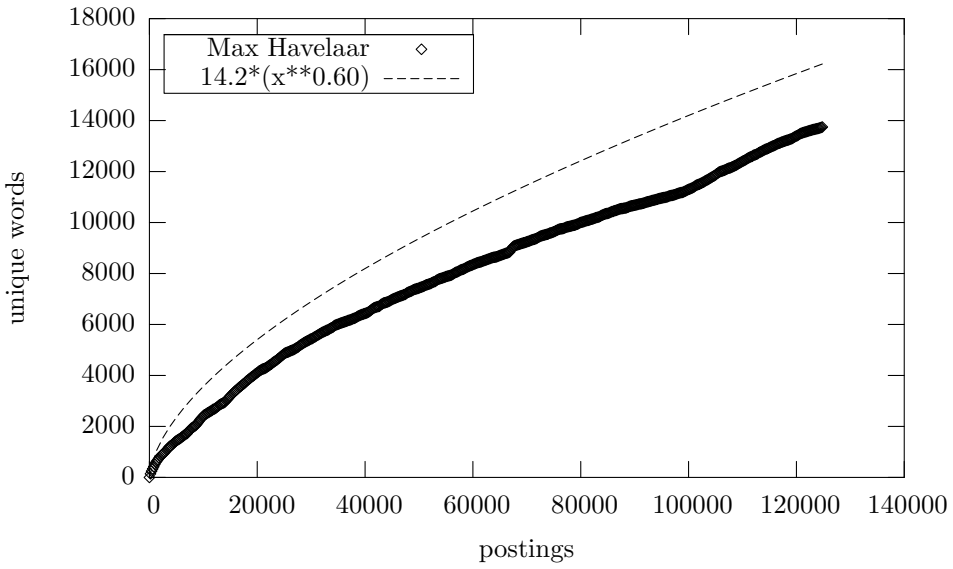

Figure B.2: Word/postings ratio in Max Havelaar B.2

Considering that similarity of chapters in Max Havelaar is stronger than in news articles I suspect that the Word/posting ratio is directly related to the ratio between the number of postings and the length of the weight vector in figure 9.6. Unfortunately the word probability distribution [15] is that hard to work with that I suspect a proof for this relation is a paper on its own.

Appendix C

Using N-grams for Language Detection within the ilse Parser

The ilse parser has a language detection function in case the language of a document is not provided by the HTTP header.

The language detection function uses all trigrams (see chapter 13) in the document to determine the probability that a document is of a certain language L.

Previous version

Unfortunately the former language detector has a number of deficiencies:

- Only the top 30 ranked trigrams (per language) are stored, all others are ignored.
- Trigrams that occur in multiple languages are only represented by the most frequent language.
- Each language has (initially) the same probability to be appointed as the winning language.

Suggestion

In my opinion, the trigram statistics will suffice for language detection, only the algorithm and the structure and amount of data has to be altered. Let

$$P(L = l|X_0 = x_0, X_1 = x_1, \ldots, X_{n-1} = x_{n-1})$$

be the probability that the language of a sequence of characters $x_0, \ldots, x_{n-1}$ is l.

From [8] the following is derived:

$$P(L = l|X_0 = x_0, X_1 = x_1, \ldots, X_{n-1} = x_{n-1})$$

$=\nleftarrow$ Product rule:$P(A|B) = \dfrac{P(A,B)}{P(B)} \nrightarrow$

$$\frac{P(L = l, X_0 = x_0, X_1 = x_1, \ldots, X_{n-1} = x_{n-1})}{P(X_0 = x_0, X_1 = x_1, \ldots, X_{n-1} = x_{n-1})}$$

$=\nleftarrow$ Inductive Bayes$\nrightarrow$

$$\frac{P(L = l)P(X_0 = x_0|L = l)P(X_1 = x_1|X_0 = x_0, L = l)\cdots P(X_{n-1} = x_{n-1}|X_{n-2} = x_{n-2}\ldots X_0 = x_0, L = l)}{P(X_0 = x_0)P(X_1 = x_1|X_0 = x_0)\cdots P(X_{n-1} = x_{n-1}|X_{n-2} = x_{n-2}\ldots X_0 = x_0)}$$

$=\nleftarrow$ Markov Assumption for k=2:$P(Y_i = y_i|Y_{i-1} = y_{i-1}\ldots Y_0 = y_0) = P(Y_i = y_i|Y_{i-1} = y_{i-1}, Y_{i-2} = y_{i-2}) \nrightarrow$

$$P(L = l)\frac{P(X_0 = x_0|L = l)}{P(X_0 = x_0)}\frac{P(X_1 = x_1|X_0 = x_0, L = l)}{P(X_1 = x_1|X_0 = x_0)}\prod_{i=3}^{n}\frac{P(X_i = x_i|X_{i-1} = x_{i-1}, X_{i-2} = x_{i-2}, L = l)}{P(X_i = x_i|X_{i-1} = x_{i-1}, X_{i-2} = x_{i-2})}$$

The probability on $\frac{P(X_0=x_0|L=l)}{P(X_0=x_0)}$ and $\frac{P(X_1=x_1|X_0=x_0,L=l)}{P(X_1=x_1|X_0=x_0)}$ are assumed almost equal to 1

$$P(L = l)\prod_{i=3}^{n}\frac{P(X_i = x_i|X_{i-1} = x_{i-1}, X_{i-2} = x_{i-2}, L = l)}{P(X_i = x_i|X_{i-1} = x_{i-1}, X_{i-2} = x_{i-2})}$$

Let $T_l(a,b,c) = log\frac{P(X_i=a|X_{i-1}=b,X_{i-2}=c,L=l)}{P(X_i=a|X_{i-1}=b,X_{i-2}=c)}$ then the logarithm of the language probability will be:

$$log(P(L = l|X_0 = x_0, X_1 = x_1, \ldots, X_{n-1} = x_{n-1}))$$
$$=$$
$$log(P(L = l)) + \sum_{i=3}^{n} T_l(x_i, x_{i-1}, x_{i-2})$$

Now it is just a matter of building the table T for all languages. Such a table (L[1] = English) will look like:

Trigram	L (id)	T()
$\_\_v$	001	-0.096021
$\_\_w$	001	-0.078285
$\_\_x$	001	-0.078553
$\_\_y$	001	-0.071473
$\_\_z$	001	-0.082266
$\_a\_$	001	0.505938
$\_aa$	001	-0.496125
$\_ab$	001	0.067126
$\_ac$	001	0.164828

Bibliography

[1] *The growing hierarchical self-organizing map*, http://www.ifs.tuwien.ac.at/∼andi/ghsom/.

[2] *light som project*, http://www.comp.nus.edu.sg/∼pris/liGHtSOMProject/liGHtSOMIndex.html.

[3] *The somlib digital library project*, http://www.ifs.tuwien.ac.at/∼andi/somlib/.

[4] Hugo Brandt Corstius alias Hugo Battus, *Rekenen op taal*, Querido, 1983.

[5] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, *Modern Information Retrieval*, Prentice Hall, 1999.

[6] Michael W. Berry and Murray Browne, *Understanding search engines*, Society for Industrial and Applied Mathematics, 1999.

[7] E. Bingham, J. Kuusisto, and K. Lagus, *Ica and som in text document analysis*, Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval (Tampere, Finland), august 2002, http://www.cis.hut.fi/ krista/papers/Bingham02.sigir.pdf, pp. 361 – 362.

[8] Eugene Charniak, *Statistical language learning*, Massachusetts Institute of Technology, 1993.

[9] James Clark, *Expat xml library*, http://expat.sourceforge.net.

[10] Hugo Brandt Corstius, *Computer-taalkunde*, Coutinho, 1978.

[11] Michiel de Hoon and Michael Eisen, *Cluster 3.0 manual*, November 2002, http://bonsai.ims.u-tokyo.ac.jp/˜mdehoon/software/cluster/manual/Distance.html.

[12] 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** (1990), no. 6, 391–407, http://lsi.research.telcordia.com/lsi/papers/JASIS90.pdf.

[13] Michael Berry et al., *General text parser*, http://www.cs.utk.edu/∼lsi/soft.html (software) and http://www.cs.utk.edu/∼berry/papers02/GTPchap.pdf (documentation).

[14] Gregory Grefenstette, *Tokenisation*, Syntactic Wordclass Tagging (H. van Halteren, ed.), Kluwer Academic Publishers, 1999, pp. 117 – 133.

[15] F.A. Grootjen, D.C. van Leijenhorst, and Th. P. van der Weide, *A formal derivation of heaps' law*, Tech. Report NIII-R0302, Radboud Universiteit, Nijmegen, 2003, http://www.cs.ru.nl/ bolke/heapsrapp.pdf.

[16] Kevin Gurney, *An introduction to neural networks*, UCL Press, 1996, http://www.shef.ac.uk/psychology/gurney/notes/index.html.

[17] M. H. Hassoun, *Fundamentals of artificial neural networks*, MIT Press, 1995.

[18] Simon Haykin, *Neural networks*, Prentice Hall, 1999.

[19] Jettero Heller, *Jet's Neural Library* , http://www.voltar-confed.org/jneural/.

[20] David R. Hill and Bernard Kolman, *Modern Matrix Algebra*, pp. 293–302, Prentice Hall, 2001.

[21] Timo Honkela, *Comparisons of self-organized word category maps*, WSOM'97: Workshop on Self-Organizing Maps (Finland), June 1997, http://www.cis.hut.fi/wsom97/progabstracts/ps/honkela_2.ps, pp. 298 – 303.

[22] Timo Honkela, Samuel Kaski, Krista Lagus, and Teuvo Kohonen, *Newsgroup Exploration with WEBSOM Method and Browsing Interface* , Tech. report, Helsinki University of Technology, 1996, http://websom.hut.fi/websom/doc/websom.ps.gz.

[23] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent component analysis*, Wiley Interscience, 2001.

[24] A Kabán and M. Girolami, *Fast extraction of semantic features from a latent semantic indexed corpus*, Neural Processing Letters **15** (2002), 31 – 43, http://cis.paisley.ac.uk/kaba-ci0/Kaban_NPL.zip.

[25] Ata Kabán and Mark Girolami, *Clustering of text documents by skewness maximazation*, International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000 (Helsinki, Finland), june 2000, http://www.cis.hut.fi/ica2000/proceedings/0435.pdf, pp. 435 – 440.

[26] Samuel Kaski, *Dimensionality reduction by random mapping: Fast similarity computation for clustering*, Proceedings of IJCNN'98, International Joint Conference on Neural Networks, vol. 1, IEEE Service Center, Piscataway, NJ, 1998, citeseer.nj.nec.com/kaski98dimensionality.html, pp. 413–418.

[27] Teuvo Kohonen, Jussi Hynninen, Jari Kangas, and Jorma Laaksonen, *Som_pak, the self-organizing map program package*, Rakentajanaukio 2 C, SF-021500 Espoo Finland, 3.1 ed., April 1995, http://ftp.funet.fi/pub/sci/neural/cochlea/som_pak/som_doc.ps.Z.

[28] Teuvo Kohonen, Samuel Kaski, Krista Lagus, and Timo Honkela, *Very large two-level SOM for the browsing of newsgroups*, Proceedings of ICANN96, International Conference on Artificial Neural Networks, Bochum, Germany, July 16-19, 1996 (C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, eds.), Lecture Notes in Computer Science, vol. 1112, Springer, Berlin, 1996, http://websom.hut.fi/websom/doc/ps/kohonen96icann.ps.gz, pp. 269–274.

[29] Tamara G. Kolda, *Limited-memory matrix methods with applications*, Ph.D. thesis, University of Maryland, College Park, Maryland, 1997, http://csmr.ca.sandia.gov/~tgkolda/pubs/umcp-cs-tr-3806.pdf.

[30] Thomas K. Landauer, Peter W. Foltz, and Darrel Laham, *An Introduction to Latent Semantic Analysis*, Discourse Processes **25** (1998), 259–284, http://lsa.colorado.edu/papers/dp1.LSAintro.pdf.

[31] Christopher D. Manning and Hinrich Schütze, *Foundation of statistical natural natural language processing*, MIT Press, 1999.

[32] Behrang Mohit Mark Kantrowitz and Vibhu Mittal, *Stemming and its effect on tfidf ranking*, 2000, http://www.sims.berkeley.edu/~behrangm/sigir2000.pdf.

[33] Andrew Kachites McCallum, *Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering*, http://www.cs.cmu.edu/~mccallum/bow, 1996.

[34] Eduard Douwes Dekker (Multatuli), *Max havelaar*, 1860, http://www.gutenberg.org/etext/11024.

[35] Dan W. Patterson, *Artificial Neural Networks*, Prentice Hall, 1996.

[36] Peter Jackson and Isabelle Moulinier, *Natural Language Processing for Online Applications*, John Benjamins, 2002.

[37] M.F. Porter, *Snowball: A language for stemming algorithms*, http://snowball.tartarus.org/texts/introduction.html.

[38] ———, *An algorithm for suffix stripping*, Program **14** (1980), no. 3, 130 – 137, http://telemat.det.unifi.it/book/2001/wchange/download/stem_porter.html.

[39] GNOME Project, *Gnome libxml2 library*, http://www.xmlsoft.org/.

[40] H. Ritter and T. Kohonen, *Self-organizing semantic maps*, Biological Cybernetics **61** (1989), 241 – 254.

[41] Gerard Salton and Christopher Buckley, *Term-Weighting Approches in Automatic Text Retrieval*, Information Processing & Management **24** (1988), no. 5, 513–523.

[42] Piet Tutelaers, *Herziene afbreekpatronen voor het nederlands*, 1993, http://www.ntg.nl/maps/pdf/11_35.pdf.

[43] Dimitri van Heesch, *Doxygen*, 1997 – 2003, http://www.doxygen.org/.

[44] C.J. van Rijsbergen, *Information retrieval*, 2 ed., Butterworths, 1979.

[45] Wikipedia, *Correlation coefficient*, http://en.wikipedia.org/wiki/Correlation_coefficient, http://nl.wikipedia.org/wiki/Correlatiecoëfficiënt.

[46] ———, *Euclidean distance*, http://en.wikipedia.org/wiki/Euclidean_distance, http://nl.wikipedia.org/wiki/Afstand.

[47] ———, *Tf-idf*, http://en.wikipedia.org/wiki/Tf-idf.

[48] Justin Zobel and Alistair Moffat, *Exploring the similarity space*, SIGIR Forum, vol. 32, Springer Verlag, 1998, http://www.cs.rmit.edu.au/~jz/fulltext/sigirforum98.pdf, pp. 18 – 32.

Glossary

A

alternate log a *local term weighting* scheme based on the log of the term frequency, p. 6.

B

binary weighting a *local term weighting* scheme where the weight is 0 if the *term frequency is 0*, 1 otherwise, p. 6.

C

corpus a (large) set of documents representing a category of documents, for example newsarticles from Reuters, p. 5.

correlation coefficient indicates the strength and direction of a linear relationship between two stochatic variables, p. 8.

cosine similarity the angle between two vectors of equal dimensionality. This can be computed by dividing the inner product by the lengths of the two vectors, p. 7.

D

document frequency the document frequency of a certain *term* is the number of documents that *term* occurs in, p. 6.

document vector a vector where each dimension is related to a specific term and the value of that dimension depends on the *term frequency* and the *term weighting* scheme that is used, p. 5.

E

entropy can also be used in Information Retrieval as a global weighting scheme, p. 7.

Euclidean distance (L2-distance) the distance of two (vector end) points in the Euclidean plain, p. 7.

Euclidean length the *Euclidean distance* between a (vector end) point and the origin, p. 35.

F

FSA see *Finite State Automaton*, p. 20.

G

global term weight The part of the term weight that depends on the likeliness of the term to occur in a document. The global term weight indicates that a term is uncommon and therefore more likely to be a *keyword* instead of a *stopword*, p. 6.

I

idf idf is a global term weighting scheme where the weight g_t for a certain term t is defined by the log of the number of documents divided by the number of documents t occurs in, p. 7.

index a reverse mapping, in the context of information retrieval this is usually a mapping from *term*s to documents, like with search indices, p. 5.

inverse frequency a *local term weighting* scheme based on the inverse of the termfrequency, p. 6.

K

Kohonen Map (see *Self Organizing Feature Map*), p. 9.

L

local term weight The part of the *term weight* that depends on the term frequency of a certain *term* in a certain document, p. 6.

log term frequency a *local term weighting* scheme based on the log of the term frequency, p. 6.

LVQ see *Learning Vector Quantization*, p. 11.

P

posting an instance of a *term*, as opposed to a unique *term*, p. 5.

precision the amount of relevant results, for a certain query q, retrieved by an information retrieval system divided by the total amount of results retrieved, p. 8.

pruning withing the *Vector Space Model* this means removing vector dimensions which do not contribute to the performance of the Information Retrieval system, p. 22.

Q

query a *term*, sequence of *terms* or even a complete document to be matched against a set of documents, p. 8.

R

random matrix a rectangular matrix M with random values in such a way that the result of a weight vector multiplied with M is 1) of lower dimensionality and 2) can be compared to other multiplied weightvectors using the cosine similarity, p. 24.

recall the amount of relevant results, for a certain query q, retrieved by an information retrieval system divided by the total amount of relevant results for q, p. 8.

S

Self Organizing Feature Map an unsupervised Neural Network that matches an input with a neuron in a two-dimensional grid, p. 9.

stemming mapping a word to its stem, i.e. $worked \mapsto work$, p. 5.

stopword a word that it assumed not to add much information to a document, as opposed to *keyword*, p. 6.

T

term a word, number, or alphanumeric expression, p. 5.

term frequency the amount of times a certain *term* occurs in a certain document usually formulated with $f_{t,d}$, p. 6.

term weight a value for a certain *term*/document combination based on the *term frequency* of the *term* in that specific document and the *document frequency* of that *term*. Usually the termweight is a product of a *localtermweight* and a *globaltermweight*, p. 6.

term weighting A mapping from a *term*-document frequency matrix into a matrix with *term weights*, p. 22.

TF-IDF is a term weighting scheme where the local term is equal to the term frequency in the document and the global term is equal to the idf of the term: $w_{t,d} = f_{t,d} \cdot idf_t$, p. 7.

tokenization splitting a (natural) text into separate *terms*, p. 5.

V

Vector Quantization maps vectors within a two-dimensional grid, p. 9.

Vector Space Model a model where documents are mapped to vectors of *term frequencies*, see also *document vector* , p. 5.

W

Word Category Map a system that clusters words into (unsupervised) categories, p. 11.