

Text Processing Study Project Report

Archil Avaliani

International University in Germany

April 2004

Table of Contents

1. Introduction
2. Information Retrieval Techniques
3. Porters Stemming Algorithm
4. Useful Libraries
5. Index Generation (Indexing)
6. Introduction to The Google PageRank Algorithm
7. Text Vector Comparison
8. Stemming and Synonyms
9. Spam Filter (Final Result Formulas)
10. Available Spam Filtering Programs
11. Learning based on available data corpus
12. Text Clustering
13. Naïve Bayesian Algorithm for Spam Filtering
14. Implementation Details of Spam Filter
15. Spam Filter Result Analyses and Statistics
16. Further Ideas

1. Introduction

The goal of this course was to introduce us to different text processing approaches. The course clearly discussed different information retrieval, text clustering, cluster comparison, and analyses methods. Finally we were introduced to Bayesian filters. To accomplish our goals we were allowed to do a research about different existing mechanisms during the first part of the course. The aim of this paper is to clearly discuss the text processing methods covered in the course, describe algorithm, provide analysis and statistics, and finally provide some further ideas.

2. Information Retrieval Techniques

This section of a paper contains different techniques of retrieving base information from the source documents.

The text may contain different irrelevant information that is not important to take into account. Such information sometimes needs to be removed. This irrelevant information is for example lower/upper case letters, plurals, different spellings, etc. In order to avoid this, several kinds of methods are available:

- Analysis start with word normalization. Normalization methods are the techniques used to reduce the vocabulary. They normalize textual forms of words into standard terms. It helps to get rid of different spellings, capital letters, plurals, etc. Basically it is a method that normalizes the text spelling and inflection wise.
 - Removing stop words.
 - Normalization of spelling.
 - Reduction of inflected words in the text
 - Excluding the words of the same grammatical meaning. (The hardest Step)

- Stemming. These are the disenchanting cut-off operations. Different approaches are used that attempt to group words together with the same stem. The vocabulary is reduced because some words can be tracked in others. These algorithms work using different morphological techniques. They find stems, decompress words and group them together.
 - Finding word stems within larger words.

- Syntactic means. Providing relationships between syntactic terms.
 - Analyzing text and mapping them to normalized words.
 - Attempts to find relationships between term concepts of words that are composed of several words.
 - Finding structural dependences among the words.

- Statistical techniques. These methods permit to use words other than the originals from the text. This is offered by finding semantically related words; depending on how often do they occur in a given file.

After this steps a list of words of the vocabulary is generated. The information may be separated in different files.

Sometimes it is necessary to analyze large amount of data. In this case a fast and powerful system is necessary that will operate successfully. The algorithms should not depend on the type of text also. In order to generate analysis, different learning techniques need to be applied. Especially for the word normalization step.

3. Porter's Stemming Algorithm

The goal of Porter's stemming algorithm is to remove inflexion and morphological endings from the words. This process is called stemming or term normalization. Porter, M.F, described the algorithm in 1980. The first original stemmer was written in BCLP programming language. There are many different implementations for this stemmer.

There have been some small changes that made the original algorithm more efficient.

The algorithm is nicely described on the following site: <http://snowball.tartarus.org/>

For more information about the algorithm and its implementation code in different languages see: <http://www.tartarus.org/~martin/PorterStemmer/>

4. Useful Libraries

i) WordNet

I did put this library on the first place because of the importance and variety of its functions. The library is nicely compatible with Java. WordNet is an online lexical system. It organizes English words according to their grammatical type into sets. Where each sets represents one lexical idea.

More information about WordNet library can be found at:

- <http://www.cogsci.princeton.edu/~wn/index.shtml>
- <http://www.dlib.org/dlib/october98/10bookreview.html>

ii) Glimpse

This is a word searching system that is based on very powerful indexing. It can be used for large data collections also. The system operates very fast and also is a standard search engine (in Harvest). For the matter of curiosity Harvest is a Distributed Search System. (<http://harvest.sourceforge.net/>)

Fore more information please visit: <http://freshmeat.net/projects/glimpse/>

iii) GROK

This was an interesting library/project to discuss because it provides large amount of various tools. GROK is special because its “natural language modules” follow the defined guidelines. This means that the modules can be exchanged freely and easily with other same type modules. The libraries of the GROK modules implement the interfaces particularized by OpenNPL.

Official Website: <http://grok.sourceforge.net/about.html>

5. Index Generation (Indexing)

Indexing is a process of identifying content terms of the text. Here I will discuss the simplest approach that is based on using words that occur in the individual documents.

The process consists of three steps:

1. For the specified document find the individual words.
2. Remove stop words. Using some existing dictionary. This is important because the high number of stop word occurrence in the documents can effect the result.

3. Remove Suffixes
4. Group the same words together and generate a term weight (depending on number of times a word occurs in text)
5. Put the words into stem with their weight accordingly

Index generation can be complex if multi term words are considered. In this case there are many different algorithms that deal with this special case and use rich vocabularies. One of these methods is Associative Indexing Methods.

Distinguishing allowable indexes from the rest of the vocabulary is done by term scheduling (also called thesaurus).

Generating the index for a document is an important part, because later it can be used by different kinds of algorithms. For this step a synonym database, phrase generation, hierarchical subject identifiers are useful for successful index generation.

6. Introduction to The Google PageRank Algorithm

Google uses a PageRank algorithm in order to determine the relevance level of the website. PageRank basically shows the importance of a page. Its value is calculated according to how many other sites have links to this site.

The rank formula is the following:

$$PR(\text{SiteA}) = (1-d) + d * (PR(T1)/C(T1) + \dots + PR(Tn)/C(Tn))$$

Where:

- | | |
|-------------------------------------|--|
| PR(S) | Is a PageRank of site S (shows the number of links coming to the site) |
| C(S) | Shows the number of links that are going out from site S. |
| T ₁ up to T _n | The pages that have links to the site we are ranking. In this case Site A. |
| Variable d | A dumping factor (mostly it is set to 0.85). |

This is a text processing paper, so I will not go into details into ranking and focus more about the indexing system that Google uses.

Google has a repository, which contains fully every web page and uses it as a corpus. According to Google and Text Retrieval Conference, 20GB data is considered as the “Very Large Corpus”. As one can see is really very small in comparison to 147GB (uncompressed) repository (of about 24 million web pages) that Google is using. But because of small corpus that other search engines use, standard vector space algorithms do not produce correct results. This is because web searching is more complex and the algorithm may return a result that is one or a few words bigger than the searching string and as a result have completely different link. for that reason standard information retrieval methods need to be improved in order to return precise results for web pages also.

The system that Google uses is very complex and consists of many different stages. The indexing algorithm, which I will discuss, parses a web page and converts every word into a wordID (this is done by using Lexicon and in-memory hash table). After the wordID generation, the occurrences of each word on a page is calculated and translated into Google’s hit list (hit list contains following information for the word: number of occurrences in a text, location, font and capitalization), which are later stored as forward barrels. Several indexers are running at the same time, and writing the data in a log file. After this step is completed, the final indexer processes the log file. The indexing process ends with sorting wordID objects; this is necessary for inverted index generation.

At the end of this section I would like to show some statistical storage data for Google's System:

Storage Statistics	
Total Size of Fetched Pages	147.8 GB
Compressed Repository	53.5 GB
Short Inverted Index	4.1 GB
Full Inverted Index	37.2 GB
Lexicon	293 MB
Temporary Anchor Data (not in total)	6.6 GB
Document Index Incl. Variable Width Data	9.7 GB
Links Database	3.9 GB
Total Without Repository	55.2 GB
Total With Repository	108.7 GB

(Retrieved from: <http://www-db.stanford.edu/~backrub/google.html> on April 9, 2004)

For more detailed information about search engine algorithms please check: <http://www-db.stanford.edu/~backrub/google.html>

7. Text Vector Comparison

After generating the index of a documents and normalizing it, one needs to create a vector that gives information about how many times each word appeared in a document. After generating this information for all documents we need to compare, all we need to do is to compare the cosine value between any two vectors. The vectors need to have equal size. For the word that appears in one document but not in the other we just assign the number 0.

8. Stemming and Synonyms

Stemming and finding synonyms can change the final result of vector comparison. One can check and replace a word with its synonym after stemming is done (in our case with porters algorithm). Sometimes this process leads to significant differences in results.

WordNet libraries provide both stemming and synonym finding functionalities. This step can easily be implemented if one uses WordNet. But it is important to consider if you really need this step for your program. For some programs like Spam filters it is better not to implement this. To use stemming and synonyms are hard steps. It needs to be done very carefully and very large dictionary is needed. In case of spam filters, it becomes even more sensitive. Just a slight change of words can show us that a spam mail is a ham. My testing results showed much better and precise results when I didn't use WordNet Libraries and made my stop word list small. (A spam can contain very large amount of stop words, so sometimes it is not a good idea to remove all or most of it)

9. Spam Filter (Final Result Formulas)

Recall and Precision are the evolution measures of spam filtering indexing. However the final result F depends on both of them. They are measured as follows (Fig 1):

$$R = \frac{\text{number of items retrieved and relevant}}{\text{total relevant in collection}}$$

$$P = \frac{\text{number of items retrieved and relevant}}{\text{total retrieved in collection}}$$

(Fig 1, retrieved from: <http://delivery.acm.org/10.1145/330000/321441/p8-salton.pdf>)

$$F = 2 / (1/p + 1/R)$$

Nc – number correct guesses

Nic – number of incorrect guesses

Nns – number of not selected

R - Recall

P - Precision

F - efficiency in terms of time

As one can see recall measures the programs ability to correctly identify relevant materials. And precision is a measurement of ability of rejecting irrelevant materials. Therefore the software result needs to depend on both of them.

10. Available Spam Filtering Programs

One can see many different spam filter implementations on the web. The most interesting ones I found were POSTARMOR, JSpamFilter and Secluda. They are both written in java.

- POSTARMOR is a free spam filter, implemented in java. The software can be installed on personal mail account.
- JSpamFilter is a server side cross-platform spam filter that provides different functionalities for mail servers.
- Secluda processes all the emails before they are delivered to the mail server.

I didn't go into details to see how this programs work. It was not the aim of the course. The source code of these programs is not available for free.

11. Learning based on available data corpus

In order to make the Spam filtering algorithm work faster, it is better sometimes to implement the learning step. This is especially important when having large corpus.

During this step one needs to generate a convenient text cluster object, that later can be efficiently used by the algorithm in order to work properly. The process is deeply described in Bayesian Algorithm section.

12. Text Clustering

Text Clustering is a process of dividing the text documents into different clusters. With the corpus data one can create text clusters that will later be used to group a new document correctly.

During clustering process the total set of data is called a Hypercentroid, which is divided in Supercentroids. Supercentroids contain Centroids and Centroids contain Documents. One can divide the data into many different levels, depending on how the data is going to be used later.

As one can see the structure of this data can be seen as a tree (especially used in searching algorithms) with the Hypercentroid as a root element and documents as the leaves. This is very useful for cluster-search process. One can use similarities and make the searching algorithm really efficient. The time consuming part, cluster-generation, needs to be done only once, while the cluster-search algorithm can be performed many times.

There are three main cluster generation algorithms that require pairwise item similarities to be computed: Single-link clustering, complete-link clustering, and group-average clustering.

Alternatively there exist heuristic methods for cluster generations. These does not require pairwise comparisons. The algorithm strategies are uses the data about the item pair wise similarities. This is the reason of cluster generation algorithms being expensive.

Cluster searching is a simpler than cluster-generation algorithm. It allows fast searches for files. This reduces the amount of data that needs to be searched.

13. Naïve Bayesian Algorithm for Spam Filtering

Bayesian Spam Filter provides the function of filtering spam emails automatically. Using a probability based approach it calculates the chance that an email is a spam. This calculation is done by looking what words does the message contain and compares it to the corpus data. The algorithm begins by learning the content of the spam and non-spam (ham) corpus. After it gets a new message, the information available from the

training/studying process is used to compute the probability whether a message is a spam or not. This computation is based on the content of the message.

Bayes Rule

$$P(\text{spam} | \text{content}) = P(\text{content} | \text{spam}) * P(\text{spam}) / P(\text{content})$$

In order to simplify the formula and computations we need to assume that a words probability of being contained in a text is independent from other words that of the same document. One can see that this assumption is not true, but because this efficiently leads to computation simplification and returns still a nice approximate result, it is a good idea to use it. This is called a Naïve Bayes Assumption.

$$P(\text{content} | \text{spam}) = (\# \text{Spam messages with word 1}) / (\# \text{ spam messages}) * \dots * (\# \text{Spam messages with word N}) / (\# \text{ spam messages})$$

$$P(\text{Spam}) = \# \text{Spam messages} / \# \text{ all messages}$$

$$P(\text{Ham}) = \# \text{Ham messages} / \# \text{ all messages}$$

$$\text{Ratio} = P(\text{spam} | \text{content}) / P(\text{ham} | \text{content}) = (P(\text{content} | \text{spam}) * P(\text{Spam})) / (P(\text{content} | \text{ham}) * P(\text{ham}))$$

This ratio shows whether a message is a spam or not. If it is greater than 1 it is more likely to be a spam, otherwise it is ham. If the chance is exactly one, or a very near to one it is better to classify it as a ham, in order not to delete necessary messages. It is always better to keep a message that one is not sure about rather than just classifying it as a spam and deleting it.

In order to make computation easier, and get nicely distributed results one uses logarithms. In this case the benefits are that the products become sums, and the Ratio changes from minus infinity to 0, instead of from 0 to 1.

However there is a slight difficult in computation. When a particular word appears only in spam (or only in ham), $P(\text{word} | \text{spam}) = 0$ ($P(\text{word} | \text{ham}) = 0$)

In which case we cannot log the value. To solve the problem one needs to use smoothing. During the calculation

$$(\text{\#Spam messages with word 1}) / (\text{\# spam messages})$$

Changes to

$$(\text{\#Spam messages with word 1} + 1/2) / (\text{\# spam messages} + 1/2)$$

Sometimes when we do not use logarithms and just have multiplication, the probability becomes to small and therefore computer assigns it value zero after some very small value. To avoid this one can multiply the probabilities during computation to some constant. Note that the same value is necessary to be used in both spam and ham calculation in order to cancel out in the final result.

If some values will still become zero, it is necessary to change them with some very small number. Otherwise one can get that the probability that a document is a spam or ham to be there. In which case the ratio result will return either infinity or NaN value.

14. Implementation Details of Spam Filter

First the program starts with training process. It takes the spam corpus and generates and creates a set of words with how many times each word occurred in the text. After this the program generates a Word object for this data and adds them into array list. It does the same thing for each file in the spam corpus and joins the results with the previous ArrayList of words. The final result is a set of Word objects with frequencies. After the program is done with all the files in spam, it saves the result as an object in a file. Exactly the same procedure applies when training with ham corpus.

After the training objects are generated one can run the Bayesian Filter of the program. Bayesian filter reads the objects from the files and applies its algorithm

The slight modification in the algorithm is that when calculating probability for each word, one divides the number of times a word occurred in a new message to the number of times it occurred in spam or ham. (Depending the probability that is calculated)

If the probability is zero, because it was too small and the computer made an approximation, it is changed to very small number. (In my special case 0.001)

After the probabilities of a message being a spam and ham is calculated, the result is returned by simply dividing them. The probability of a ham is multiplied by 1.01 in order to assign it a bigger weight.

15. Spam Filter Result Analyses and Statistics

The spam filter takes about 18 minutes for training and 15 minutes for filtering when each spam and ham corpus is about 1 MB and the data to be filtered is about 500K. It also indirectly depends on the number of files the data is distributed among. (Because it increases the number of I/O s to be done, and the number of times of putting the results together increases). Useful text processing software needs to be able to provide high recall and precision. The final result of the spam filter that I wrote is shown below (see Table 1).

P	1
R	0.81
F	0.895028

(Table 1)

Some of us have done the spam filter that didn't need any training and took about 3 seconds for calculations for the same data. Also the result of F was about 70%, the algorithm was very efficient. Results of the contest are the following (see Table 2).

Angel	Archil	Felix	Min	Raluca	Thade
P 0.786667	P 1	P 0.260163	P 1	P 0.323887	P 0.510989
R 0.59	R 0.81	R 0.32	R 0.38	R 0.8	R 0.93
F 0.674286	F 0.895028	F 0.286996	F 0.550725	F 0.461095	F 0.659574

(Table 2)

Specialties about programs:

- ✓ Angel - No corpus, very fast algorithm. Took about 3 seconds.
- ✓ Archil - Algorithm described above
- ✓ Felix - No Bayesian Algorithm. Normal vector comparison
- ✓ Min - Same algorithm as mine, just modified Bayesian formula
- ✓ Raluca - I don't have enough information for this program.
- ✓ Thade - Using WordNet extensions.

Unfortunately this is the only information I have about the other implementation of programs. It would be really nice to write more statistical/implementation information and provide later analysis.

However the algorithms that didn't use any Bayesian filters, and filtered the messages by simply using the vector comparison mechanisms showed F = 29 %. This shows that Bayesian filter is really a good approach for filtering the data.

I would like to add some more statistical analyses about comparing results of my program using different libraries and implementing Machine Learning algorithm for text categorization. But however there was not enough time to fit this in one course.

16. Further Ideas

Having an efficient information retrieval algorithms that give good performance are very important for many different serious projects.

It would be interesting to use many different implementations and also the libraries that are available on the net in order to test and analyze the results. There are many modified implementations of Bayesian algorithm that generate the results differently. These different implementations would allow to generate a lot of statistical data, analyze the performance and find the most suitable algorithms for different uses.

Conclusion

The objective of this course was to study different text processing and information retrieval methods. We also discussed how to classify the documents, compute the similarities and do the clustering in order to perform complex operations on them later. Finally we implemented the Bayesian Spam Filter and had a small competition. Please see the survey document in order to find more about each step the course contained.