Aalto University

T-75.4400 Information retrieval

Assignment 2

On the Efficacy of Alternate Similary Measures, Stemming and Stop Word Removal in Text-Based Information Retrieval Systems

Antti Partanen
antti.partanen@aalto.fi
295967

Vikram Kamath
vikram.kamath@aalto.fi
440819

Aalto University
School of Science

April 22, 2015

1 Introduction

The task of Information retrieval is to obtain information resources relevant to the imminent need from information resources. Resources include books, journals and other documents. Searches can be based on meta-data or on full-text indexing.

Automated information retrieval systems are used to reduce the so called *information overload*, i.e. the difficulty of a person to understand an issue due to overwhelming amounts of information. Most existing text retrieval techniques rely on indexing key-words. Unfortunately keywords or index terms alone cannot adequately capture the document contents, resulting in poor retrieval performance. Yet, keyword indexing is widely used in commercial systems because it is still the most viable way by far to process large amount of text.

The generic textual information retrieval process is show in Figure 1 and it has the following steps:

1. The system builds and index of the documents (*indexing*)

2. User describes the *information need* in a form of a query, which is parsed and transformed by the system with the same operations applied to the documents (*query formulation*)

3. The system retrieves, ranks and displays documents that are relevant to the query from the index

4. User may give relevance feedback to the system

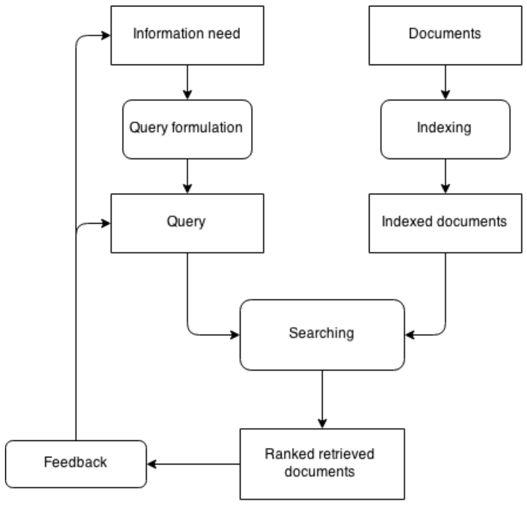

Figure 1: Information retrieval pipeline (Hiemstra, 2009)

Efficient and effective information retrieval techniques are critical in managing the increased amount of textual information available in electronic form. Therefore, it is no wonder that currently the most visible IR systems are web search engines.

Apache Lucene

In this paper, we compare the two information retrieval techniques: Vector Space Model (VSM) and BM25. Furthermore, we analyze what effects does the usage of stemmers (porter stemmer) and stopwords have with the retrieval result. The evaluation of our experiments is visualized with eleven point precision recall curves.

Draft version: aka needs modifying + references

2 Techniques

Ranking methods are used by search engines to rank matching documents according to their relevance to a given search query. Most common approaches to information retrieval are algebraic (e.g. VSM) and probabilistic (e.g. BM25)

2.1 VSM

VSM (Vector Space Model) is an algebraic representation model for objects as vectors of identifiers. Beyond information retrieval, VSM is also used in information filtering and indexing. It was first introduced by (**?**) and it was first utilized in SMART IR system (**?**).

The core idea of VSM is rather simple: represent documents (D_j) and queries (Q) as vectors in vector-space.

$$D_j = (w_{1,j}, w_{2,j}, ..., w_{t,j})$$
$$Q = (w_{1,q}, w_{2,q}, ..., w_{n,q})$$

There are several different ways of calculating these weights, but *tf-idf* is one on the best known. Documents are ranked according to their proximity to the query in this space, where proximity corresponds to the similarity of the vectors. Similarity is not calculated as an euclidean distance, but rather as an angle or cosine between the vectors

$$cos\theta = \frac{D_j \cdot Q}{\left\|D_j\right\| \|Q\|}$$

, where the norm or length $\|Q\|$ is calculated as

$$\|Q\| = \sqrt{\sum_{i=1}^{n} q_i} \ .$$

2.2 BM25

BM25, often referred to as Okapi BM25 (where BM stands for Best Matching), is a probabilistic ranking function. It is based on probabilistic relevance model, devised by Robertson et al. **??**. BM25 uses a bag-of-words representation of a document, a representation that ignores the relative ordering of words in the document and the query. It isn't one stand-alone method, but is actually a collection of functions, each of which uses different hyperparameters, the outputs of which are combined to give the final similarity score between a document and a query. BM25 uses both term frequencies and inverse document frequencies to compute the final similarity score. [TODO Insert Citation to Text book]

ADD REFERENCES

2.3 Stemmers

In most languages and predominantly so in English, words occur in dif- ferent forms even though they have the same meaning. For example ‘democracy’, ‘democratic’ , ‘democratize’ all have the same deriva- tion and although they’re used in different contexts, they have the same ‘semantic’ meaning. More often than not, it would be useful to search for using just one of the forms and have documents retrieved that con- tain all semantically similar forms. Lemmatization is a crude heuristic process that aims to achieve this by removing the ends of the words so that different forms of the same word, after processing, lead to the same, shortened form. So in the example above, all three words would map to ’democraci’ after processing. We used the porter stemmer [TODO: Insert Citation] for our experiments (although there exist many other different stemming algorithms that aim to achieve a good ’lemmatization’)

2.4 Stopwords

Most search engines preprocess both indexed text and search queries so as to remove commonly occurring words that are of little value to the context of the document. Words like ‘A’, ‘and’,‘the’, ‘of’ etc are found abundantly in most English documents but add no overall value to the indexing, ranking or retrieval process. The average size of stopword collections is between 200 - 300 terms [TODO: Insert Citation]. We experimented with using the Lucene in-built stopword list, which although not exhaustive (relative to some of the stop-word corpora available), is sufficiently good.

3 Evaluation

Add text and charts

Precision
Recall
Eleven point precision recall curve

4 Conclusions

Add conclusions

5 Contributions

In the end both of us put equal amounts of effort into the project. We used GitHub [TODO: Citation?] for version control and although we did some independent work at the comfort of our own homes, a bulk of the work was done in a series of 'hackathon' style, caffeine-fueled meetings, where we just sat for 4-6 hours straight at Maari and woked on it. Although this style might be deemed by some as a sloppy, we'd like to think of it as pair programming on speed - in that the fact that we sat together and worked on it enabled almost seamless debugging and ideation.
A diagrammatic (albeit) rough division of tasks is shown below [TODO: Should we add this?]

Mind map style of work division might look neat?

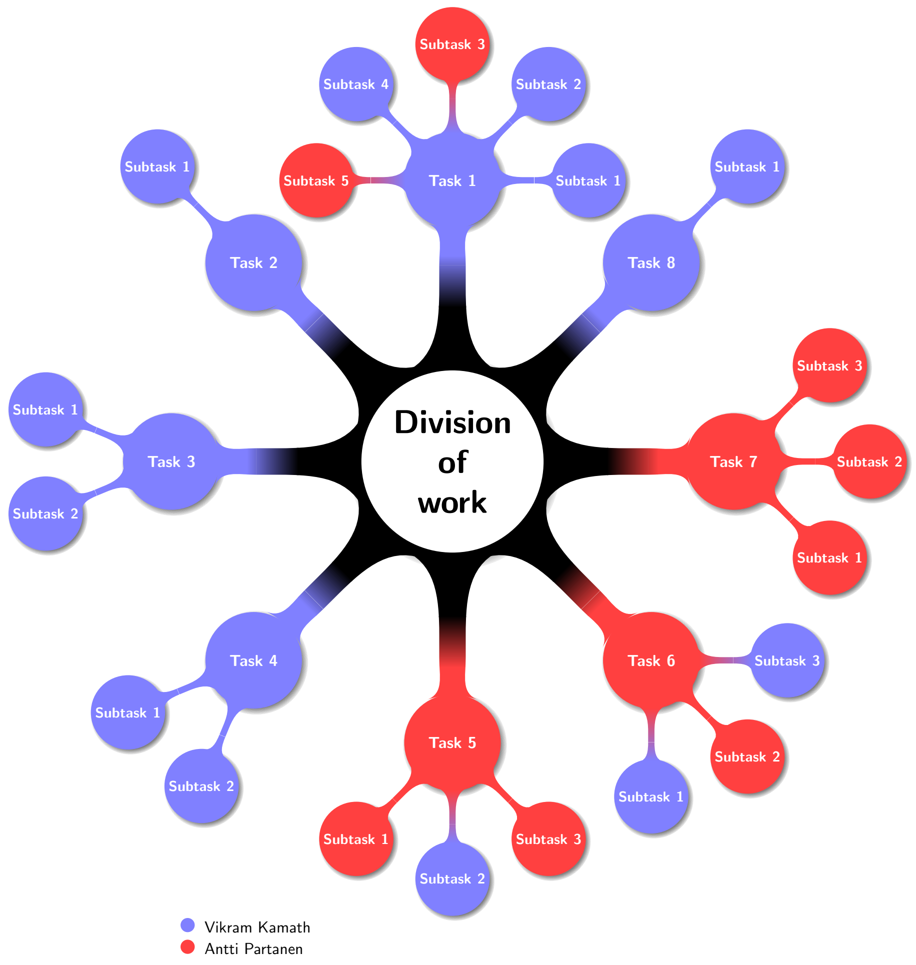

Figure 2: Contributions represented in a mindmap

References

Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

Djoerd Hiemstra. Information retrieval models. *Information Retrieval: searching in the 21st Century*, pages 2–19, 2009.

Add more references

A Installation instructions

Here is a brief systems installation and system configuration instructions.

1. **Preprocessing**
   We wrote a small script to parse the XML file and dump 'items' relevant to our search task in a document called 'data.xml'. It is assumed that this file will be used as a command line argument to the program. To create a fresh dump, run 'parseXML.py' and ensure that the corpus file is in the same directory as the script.

2. **Java Indexing, Search and Ranking Implementation**
   The whole Java project 'Assignment2' once loaded into your Eclipse workspace will make available the file 'assignment2.java', that contains our implementation. Make sure to use all our imports as we use the Java HashMap implementation in our code.

3. **Plotting, Metrics**
   We dumped the output of 'assignment2.java' into csv files, one for each query-parameter combination (24 in total - 8 per query). The csv files, ordered according to query are in a folder called 'PrecRec', which further contains 3 folders, one per query containing csv files relevant to the query. Each of the query folders also contains a python script called 'calcPrecRecall.py'that consumes the csv files in the folder and computes the 11-point non-interpolated precision recall curve, which is then saved into a 'plot.png' file in the same folder.