INFORMATION RETRIEVAL

(CS F469)

“A Text Based Search Engine”

Project Studied and Presented by:

- Abhimanyu Siwach (2013A7PS075P)
- Adnan Oquaish (2013A7PS174P)
- Deevankshu Garg (2013A7PS178P)
- Antil Singhal (2013A7PS159P)
- Sai Charan Agraharam (2013A7PS134P)

This project was designed to implement a Text based information retrieval system. For the training and testing of the retrieval system we used the dataset provided by Arizona State University: the TUAW dataset containing more than 10,000 documents which disseminates information on Apple products and services. (http://www.socialcomputing.asu.edu/datasets/TUAW)

We have chosen C++ as the design language because of the fluency of all the team members in it. It was also used because of the Standard Template Library (STL) which was heavily utilised in our project.

For the Phase – I: the building of the indexing component we tried both the Probabilistic Model and the Vector Space Model. However, we realized that the Probabilistic Model heavily depended on floating point computations whose accuracy in C++ was low. So, instead we settled on the Vector Space Model (VSM) whose accuracy was relatively high.

The data set to be processed was in the comma separated values (CSV) format.

(Note: - Our program considers each line, separated by the newline character (\n) as one document. Since the input provided to us was in this format.)

So we took the input and parsed it using the *strtok* function in C++. Then we normalised it and stemmed it. We tried to use the Porter Stemmer. However, it kept setting off the warning flags and was not compatible with our code. So instead we chose to use the Porter2 Stemming algorithm by smassung (http://www.bitbucket.org/smassung/porter2_stemmer/wiki/Home)

This gave us our documents and the Unique Document ID (docID) which was simply the document index.

The next step of our project design was to store the words that were read and parsed from our corpus. We decided to go with a string based 3D-Trie (A normal trie with Pointers/Arrays in the last node of every word). The last node of the word had a pointer to the Posting List of that word.

The Posting List was implemented as a STL Map. (Note :- The internal implementation of a STL Map is a Binary Search Tree which provides O(log(n)) Insertion and Search). The posting list of every word contained a map indexing that word to the Posting List Nodes and Also contained the Document Frequency (DF) and Inverse Document Frequency (IDF) container. The Posting List Node contained a map of the Unique Document ID to the Posting Node which contained the Term Frequency (TF) of that term in that particular document and also the Term Frequency Inverse Document Frequency (TFIDF).

(Note : - the Posting List was built after the parsing of the Input Files the first time. The second run on the Posting List, after the document frequency was computed, was done for the calculation of the TFIDF of each term-document pair).

The Final Step of the project was the Query Input and Processing. The Query was taken through the Standard Input stream and Normalised using the library function *strtok*. This was then run through the Porter2 Stemmer to get the Stemmed, Tokenised words of the query. The Document Frequency (DF) and Inverse Document Frequency (IDF) of the Word were found through the Trie Lookup and then the Term Frequency was calculated. This gave us the Term

Frequency Inverse Document Frequency (TFIDF) of each query term as well.

Now the most important part of the last step was carried out, the Cosine Similarity calculation. The Cosine Similarity was calculated and stored in the Map of Document ID and Cosine Similarity. And then the output was displayed.

Precision

During our Testing phase we used more than 100 queries which gave us a precision of almost 85%.

Recall

Since this was a pure text based search and retrieved all documents containing the term, so this brought our recall to nearly up to a 95%.