### [COMS E6111 Advanced Database Systems](http://www.cs.columbia.edu/~gravano/cs6111/index.html) Fall 2015

**Project 1**

1. **Name and UNI**

Bahul Jain – bkj2111

Parth Parekh – prp2121

1. **A list of all the files that you are submitting**

temp.py – Main Python Script which runs the project.

temp2.py – Python Script that incorporates additional fine tuning to query expansion

Account Key.txt – Contains the Bing Search Account Key and Customer ID.

stopwords.txt – Contains a list of stop words and other symbols we want to eliminate.

transcripts.txt – Log of the output on the basic 3 cases (gates, musk, taj mahal), with some additional test cases (apple, friends)

1. **A clear description of how to run your program** (note that your project must compile/run under Linux in your CS account)

To run the program, run the temp.py file using the command:

python temp.py <account key> <precision> <query>

For e.g. If we want to search for the animal ‘jaguar’ and we require a precision of 0.9, the command to run the project would be:

python temp.py 2dyKIv94jDETd7ClbVKoHvJSWFJ73ZvZRc7rjpBdkG8 0.9 jaguar

This should be run from the directory which consists the file temp.py

**Note:** To run program with fine tuning in query expansion the same commands as above need to executed, but this won’t run in CS account since certain python libraries required to run this program are missing.

Command:

python temp2.py 2dyKIv94jDETd7ClbVKoHvJSWFJ73ZvZRc7rjpBdkG8 0.9 jaguar

1. **A clear description of the internal design of your project**

The flow of the code is as follows:

1. Getting bing account key, precision and query as input from the user.
2. Storing the stop words in an array which are used to eliminate irrelevant highly frequent occurring words.
3. Connecting to Bing API and getting response of the query passed in the URL.
4. Storing the Titles, Descriptions and URL’s in their corresponding array.
5. Get relevance feedback for all the search results and calculate the new precision.
6. Check for the termination cases, if termination case not reached, continue.
7. Remove the punctuations and add the words from title and description of the result in the dictionary.
8. Design a Vector Space Model for the information retrieved
9. Reformulate the query (only query expansion by augmenting to the original query)
10. Repeat steps 3 through 10
11. **A detailed description of your query-modification method**

Once we have formed the vector space model of the information retrieved we have a vector corresponding to every relevant document, every non-relevant document, and the original query. Using these vectors we implement Rocchio’s algorithm. This algorithm computes a new query vector that is similar to the relevant documents vector and dissimilar from the non-relevant documents vector. The formula for Rocchio’s algorithm is as follows…

This algorithm increases the weight of the words that occur in the relevant documents and decreases the weight of the words that occur in the non-relevant documents in the revised query vector. Alpha, beta and gamma are constants that control the factor by which we are considering contributions from the original query, relevant documents and the non-relevant documents respectively. Since we are more concerned with minimizing the distance of the query vector and relevant documents than maximizing the distance of the query vector and the non-relevant documents we generally keep a high value of beta and a low value of beta. Hence we choose.

Now we choose the words that have maximum weight after applying Rocchio algorithm. Since we are can append at the most two words, we pick two words with maximum weights. The problem arises in deciding when to add the second word and when not to. Using some statistics and observation we noticed that if the word with maximum weight and the word with second maximum weight have close by weights compared to the word with third maximum weight and the following words then adding the second weight improves the search significantly. More precisely…

Let

We choose the second word along with the first word in the query if, otherwise we simply ignore the second word and only append the first word. Often adding two words can cause the query to lead in another direction if the second word is even slightly off-topic, hence we make a more stringent comparison between the differences

The above method will give inconsistent result if. This usually happens if the there is insufficient context available for every result or if there are multiple terms with roughly same occurrences in the relevant documents. In this situation we will need to crawl through the contents of the relevant documents to obtain more context information. We have implemented this fine tuning technique, in which we crawl through relevant documents and find the term frequency for each of these conflicting terms. Based on the term frequency we then choose words to be augmented in the new query. This fine tuning is performed when…

…which rarely happens, but when it does the result can go bad drastically.

Note: We have added the code for the above fine tuning technique (in comments), but due to restrictions on the CS Account we are not able to install and run the libraries (goose-extractor) that crawl through the contents of the relevant results.

1. **Bing Search Account Key** - 2dyKIv94jDETd7ClbVKoHvJSWFJ73ZvZRc7rjpBdkG8