ADVANCED DATABASE SYSTEMS
PROJECT 1

Rohan Kulkarni (rk2845)
Aishwarya Rajesh (ar3567)

PROJECT FILES

bing_api.py : Contains the Python source code.
README.pdf: Includes the required README information.
transcript.pdf: Contains the transcript of the runs for the given test cases.
requirements.txt: Contains the list of Python library dependencies.
run.sh: Driver file to run the program.

RUNNING THE PROGRAM

1)Use the '**pip install -r requirements.txt**' command to install the python dependencies listed in the *requirements.txt* file.

2) Navigate to the "bingapi" folder and Enter "**./run.sh**" command to run the *bing_api.py* file.

User inputs to the program:
i. Query as a list of words. Eg. *musk*
ii. Target precision value. Eg. 0.9
iii. *1* if a document presented by the search is relevant and *2* if it is non-relevant.

PROJECT DESCRIPTION AND INTERNAL DESIGN

The program accepts a query (as a list of words) and the target precision value (in this case 0.9) from the user and executes the following functions to perform the corresponding tasks until the target precision is achieved, once Bing returns an initial set of search results:

a. *stripPunctuation* and *hasNumber* are used in the *updateDictionary* function to eliminate the punctuation marks and numbers respectively from the words retrieved from the document descriptions presented in the search. *updateDictionary* further orders these retrieved words to form a word dictionary for use in subsequent operations.

b. *wordTaggerFunction* finds the 'noun' words out of these retrieved words for use in subsequent operations.

c. *findDocFrequency* and *findTermFrequency* calculate the number of documents in which these retrieved words occur and the number of occurrences of these words in the retrieved documents respectively.

d. *calculateDocVector* and *calculateQueryVector* calculate the document and query vectors respectively. that are dependent on the weights of the available words, where the weight of the *i*-th word in the *j*-th document as discussed in [1] is given by (*N*=total number of documents retrieved):

$$w_{ij} = (tf)_{ij} * log\left(\frac{N}{d_j}\right)$$

e. *getUserFeedback* displays the details [title, URL, description] of the retrieved documents and records the relevant and non-relevant documents as input by the user.

f. *roccioAlgorithm* applies the Rocchio algorithm to the query vector using this relevance feedback and computes a new query vector for the next iteration.

g. *selectNewWords* uses the result of the Rocchio algorithm to estimate the choice and order of words to be added to the query for the next iteration.

The program terminates if one of the following conditions occur:

i. An insufficient number of documents (i.e. less than 10) are retrieved by the search.
ii. No relevant documents are found in the first search result.
iii. The target precision is achieved.

The word selection and query modification method is discussed in detail in the next section.

QUERY MODIFICATION METHOD

We primarily employ Rocchio's algorithm using relevance feedback to create a new query vector for use in a subsequent iteration. This new query vector is given by

$$\overline{q}_{t+1} = \alpha\overline{q}_t + \beta\frac{\sum_{d_j \in R}\overline{d_j}}{|R|} - \gamma\frac{\sum_{d_j \in NR}\overline{d_j}}{|NR|}$$

Where,
$\overline{q}_{t+1}$: *New query vector*
$\overline{q}_t$: *Original query vector*

$\sum_{d_j \in R}\overline{d_j}$: *Set of documents marked relevant with respect to* $\overline{q}_t$

$\sum_{d_j \in NR} \overline{d_j}$: *Set of documents not marked relevant with respect to* $\overline{q}_t$

$|R|$: *number of relevant documents*

$|NR|$: *number of non − relevant documents*

The constants are chosen as follows: $\alpha = 1$, $\beta = 0.75$, $\gamma = 0.15$. [2]

The new query vector $\overline{q}_{t+1}$ thus obtained contains the modified weights of all terms occurring in the collection. The weights of terms from relevant documents in the query vector increase after the application of Rocchio's algorithm to the original query vector. We sort these newly calculated weights (non-zero weights indicate their importance/relevance in relevant documents) in decreasing order of their values, signifying their decreasing order of relevance as estimated by this algorithm based on the user relevance feedback. We then consider a subset of this set of weights that represents 'noun' words or terms for our purpose. Subsequently, we choose one or two words having the highest weight(s) in this subset according to this order. We choose one or two words for the subsequent query as follows:

i. If the ratio of the first weight value to the second weight value in the ordered set of weight values is approximately 1 or rounds off to 1, i.e. lies in the range $[1, 1.5)$, we estimate that the two highest weight values representing the top two words are equally relevant to the query and relevant documents. We append these two words to the original query in the order in which they appear in the ordered set of weight values.

ii. If the ratio calculated above does not approximate to 1 then we append the word represented by the the highest weight value in the ordered set of weights to the original query.

BING SEARCH ACCOUNT KEY:

"**JbyKIOD9ljIg7tO7i8C4PzOBmKzTuUOcjCIA9R53A8k**"
(is included in the bing_api.py file)

REFERENCES

[1] Prof. Gravano, Luis; Lecture Notes, COMS E6111 "*Advanced Database Systems*", Columbia University, New York, Fall 2015.

[2] Manning D., Christopher; Raghavan, Prabhakar; Schütze, Hinrich; Chapter 9, "*An Introduction to Information Retrieval*"; Accessed: September 2015; http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

[3] Singhal, Amit; "*Modern Information Retrieval: A Brief Overview*", IEEE Data Engineering Bulletin, 2001.

[4] Harman, Donna; "*Chapter 14: Ranking Algorithms*", National Institute of Standards and Technology; Accessed: September 2015. http://orion.lcg.ufrj.br/Dr.Dobbs/books/book5/chap14.htm