=================================================================

| Project 1 |

| |

|Team Members: Vibhor Gaur (vg2376) and Joaquín Ruales (jar2262)|

=================================================================

-------------------------------------

List of files that we are submitting:

-------------------------------------

- Code:

- Extractor.java

- FeedbackBing.java

- Query.java

- QueryAdjuster.java

- Result.java

- run.sh

- Stopwords.java

- Utils.java

- Assets:

- stopwords.txt

- Documentation:

- README.txt

- transcript\_musk.txt

- transcript\_gates.txt

- transcript\_taj\_mahal.txt

--------------------------------------------------------------------

A clear description of how to run our program (in the clic machines)

--------------------------------------------------------------------

To run, use:

> bash run.sh <bing key> <desired min precision> '<query>'

for example:

> bash run.sh ghTYY7wD6LpyxUO9VRR7e1f98WFhHWYERMcw87aQTqQ 0.9 'gates'

---------------------------------------------------------

A clear description of the internal design of our project

---------------------------------------------------------

Our project is designed as follows

>>>>Overview of Algorithm Design<<<<

1.We format the input query by the user so that it does not contain any spaces.

2.We extract the top 10 bing results for the adjusted input query and take relevance feedback from the user for each result as relevant or not relevant.Based on the the user feedback we put the result into the relevant or the irrelevant result.

3.We remove stop words from the documents and title.We use a specified stop word list rather than the tf-idf for terms because the document sizes are not enough to eliminate stop words through this technique.

4.After that we tokenize the string by keeping only spaces, apostrophes and letter characters and remove any s at the end of a word.Then we turn all strings to lowercase.Then we split into a list of tokens based on the spaces.

5.We implement our query modification method to reformulate the query and get bing results from bing for this new query.(the query modification method is described in detail in the next section).

6.Repeat steps 2 to 5 until 0.9 precision is achieved.

>>>>Code Design<<<<

FeedbackBing.Java

-----------------

It contains the main program which calls other functions in different classes to first format the input query,get top 10 bing results.It then takes input from user for relevance feedback for each result.If the precision is less than 0.9, the query is reformulated (using irrelevant and relevaant documents)and the process is iterated till the desired precision (0.9) is achieved which usually takes just one iteration.

Query.java

----------

Serializes the input query from the user (“UTF-8” format) and tokenizes it before sending it to the server for requesting results.

Extractor.java

--------------

Extracts the top 10 results from bing for the query. It parses the xml results obtained into Strings.

Stopwords.java

--------------

Remove stopwords from the documents and do tokenization of strings obtained after stop word removal.

stopwords.txt

-------------

Contains the list of stopwords to be removed from the document.

Result.java

-----------

This class contains the raw strings after xml parsing of the bing results.It then calls functions on these strings to tokenize them and remove stop words.The result class consists of the title and description.So, we have an arraylist one corresponding to each result obtained.

Utils.java

----------

Only keep spaces, apostrophes, and letter characters (including letter characters in languages other than English).Then turn into lowercase. Then discard any "'s" at the end of a word. Then split into a list of tokens based on the spaces.

QueryAdjuster.java

------------------

This file implements the actual algorithm used for reformulating the query to add new words and use in the next iteration to get results from bing.It is described in detail in the next section.

run.sh

------

Bash commands to run the project.

-----------------------------------------------------

Detailed description of our query-modification method

-----------------------------------------------------

1.Get the set of all words after stop word removal and tokenization.Create a vector with as many dimensions as the number of these words.

2.Map the tokenized words to indexes or integer values.

3.Calculate weights for the dimensions of this vector (each representing a token).

a.For each relevant result get each token from the title words of that relevant result.For each of these tokens add the following weight to the corresponding index in the vector

(TITLE\_WEIGHT \* RELEVANT\_WEIGHT)/ (TITLE\_WEIGHT \* count of title tokens for this result + DESCRIPTION\_WEIGHT \* count of description tokens for this result

b.Get all tokens from the title words of irrelevant results.For each of these tokens subtract this weight from the corresponding index in the vector

(TITLE\_WEIGHT \* IRRELEVANT\_WEIGHT)/ (TITLE\_WEIGHT \* count of title tokens for this result + DESCRIPTION\_WEIGHT \* count of description tokens for this result

c.Get all tokens from the description words of relevant results.For each of these tokens add this weight to the corresponding index in the vector

(DESCRIPTION\_WEIGHT \* RELEVANT\_WEIGHT)/ (TITLE\_WEIGHT \* count of title tokens for this result + DESCRIPTION\_WEIGHT \* count of description tokens for this result

d.Get all tokens from the description words of irrelevant results.For each of these tokens subtract this weight from the corresponding index in the vector

(DESCRIPTION\_WEIGHT \* IRRELEVANT\_WEIGHT)/ (TITLE\_WEIGHT \* count of title tokens for this result + DESCRIPTION\_WEIGHT \* count of description tokens for this result

RELEVANT\_WEIGHT:Weight given to tokens from relevant documents = 0.75

IRRELEVANT\_WEIGHT:Weight given to tokens from irrelevant documents = 0.15

TITLE\_WEIGHT:Weight given to tokens from title words = 1.1

DESCRIPTION\_WEIGHT:Weight given to tokens from description words = 1.0

4. Sort this vector and get the first two indexes to get the top two occuring words to be used for the next iteration

5. The ordering for the new words to be added to the original query words :

We use the ordering in which we see the first instance of each token in the relevant documents since that is more probable to imply the natural ordering for this query and expected to give better results.

-----------------------

Bing Search Account Key

-----------------------

ghTYY7wD6LpyxUO9VRR7e1f98WFhHWYERMcw87aQTqQ

----------------------------------------------------

Additional information that we consider significant.

----------------------------------------------------

N/A