**Group 5**

\* **Name and student number:**

|  |  |
| --- | --- |
| Student number | Student Name |
| 300049083 | Cara Yuejia Li |
| 300037013 | Tong Zhao |
| 300030558 | Daniel Xu |

\* **How were tasks divided?**

To create our information retrieval system, we implemented 5 steps in total, which are preprocessing, indexing, retrieving and ranking, getting a file with results, and evaluating.

In the beginning, Cara designed the algorithm for preprocessing, and Daniel was the coder who coded Step 1 (tweet\_tokenizer.py). After finishing Step 1, Cara gave an idea about algorithms for Step 2 (indexer.py) and Step 3 (retrieval\_ranking.py), and Tong was the coder who coded Step 2 and 3. In the end, all three team members talked about their ideas about Step 4 and 5, and they discussed them in detail and found the optimal solution for the IR system. Cara was the coder who wrote the test\_query file, and she compared their results with the expected results.

\* **A detailed note about the functionality of your programs?**

The Information Retrieval (IR) system is based on a collection of documents., and we will submit the results of our system on a set of 49 test queries.

* The IR system is implemented using the Python programming language.
* Preprocessing: tweet\_tokenizer.py will lowercase all the words and remove all invalid information, e.g., punctuations, stopwords, numbers, etc.
* Indexing: indexer.py will create a dictionary with terms as key and {documentID:term frequency} as value.
* Retrieving and ranking: retrieval\_ranking.py will compute the similarity scores between a query and each document.
* Results file: test\_query.py will return the top 1000 results for each query and write them into a text file called Results.txt.
* Evaluation: our results will be compared with the expected results using the trec\_eval script.

\* **Instructions on how to run them?**

**Step 1:**

If you don't have nltk, os, numpy, bs4 installed, please do the following steps:

Windows:

open <Command Prompt>

py -m pip install --upgrade pip

py -m pip install nltk

py -m pip install os

py -m pip install numpy

py -m pip install bs4

macOS:

open <Terminal>

python -m pip install --upgrade pip

pip install nltk

pip install os

pip install numpy

pip install bs4

Linux:

Open <terminal>

python3 -m pip install --upgrade pip

pip install nltk

pip install os

pip install numpy

pip install bs4

**Step 2:**

Make sure the following files are under the same directory as our python files:

1. Trec\_microblog11.txt
2. StopWords.txt
3. topics\_MB1-49.txt

**Step 3:**

Open test\_query.py. Compile test\_query.py, then type “test\_query()” in Python IDLE Shell. Wait for code execution. You will see a text file called “Results.txt” in the same directory, which contains the top 1000 results for 49 test queries.

**Step4:**

If you would like to run the final result, make sure to have “Trec\_microblog11-qrels.txt” “Results.txt” under the same folder as trec\_eval-9.0.7 . Use Cygwin terminal and enter the command line: ./trec\_eval Trec\_microblog11-qrels.txt Results.txt .

\* **Explain the algorithms, data structures, and optimizations that you used in each of the three steps. How big was the vocabulary? Include a sample of 100 tokens from your vocabulary. Include the first 10 answers to queries 1 and 25. Discuss your final results.**

Vocabulary size:

Our vocabulary has 93241 words.

Algorithms:

* Preprocessing: Open two files and read them line by line, which are Trec\_microblog11.txt and StopWords.txt. Trec\_microblog11.txt contains the Twitter messages, and StopWords.txt contains all the stopwords that needed to be removed. Firstly, we read the Trec\_microblog11.txt file and converted it to a word list, then we lowercase all the words using lower() and we removed all the punctuations, symbols, and spaces. After that, we removed all the numbers, and then we removed all the stopwords.
* Indexing: We built an empty dictionary for initialling and we went through the word list we have after tokenization. We added all word terms as keys in the dictionary and added a dictionary inside each term with all documents that contained that term. For the inside dictionary, we set related dictionary IDs as key and term frequency as value. We search from the first-word term in the first document. If a term is not one of the keys in the dictionary, then create a new term as a key and an empty sub-dictionary as a value, also add a new document ID as a key and add term frequency to 1 as a value. If both term and document ID exist, then the term frequency increases by 1. If only a term exists but document ID doesn't exist, then add a new document ID as key and the term frequency increases by 1.
* Retrieval and Ranking: Firstly, we log tf-idf weighting into the query matrix. Next, we log length normalization in the normalization matrix. Finally, we compute the similarity between each document and query.

Data structures:

* Arrays: We read the file Trec\_microblog1.txt, and add each word and document ID as string type to a 2D array. Also, we read the file StopWords.txt, and add each stop word to an array.
* Hash table/dictionary: We set the word terms as keys and another dictionary {document ID : term frequency} as values.
* Matrix: We used matrix as table and stored our data in the matrix while we computed the similarities. We set word terms as rows, query as the first column and documents as other columns. We first stored all tf-idf weight data in our matrix and created a similar matrix with the data after length normalization. Then we used these two matrices to compute the similarity between the query and each document.

Optimization:

* Removed punctuations and symbols as much as possible. Used lower() to replace all uppercase letters to lowercase letters.
* Used a hash table to store keys and values.
* Compared the similarity between two matrices which contain all tf-idf weight data and the data after length normalization, respectively, in order to get the optimal result.

Sample of 100 tokens:

'bbc', 'world', 'service', 'savage', 'cuts', 'lot', 'people', 'fun', 'question', 'rethink', 'group', 'positive', 'outlook', 'technology', 'staffing', 'specialist', 'expects', 'revenues', 'marg', 'zombie', 'fund', 'manager', 'phoenix', 'appoints', 'ceo', 'buys', 'funds', 'closed', 'business', 'latest', 'top', 'releases', 'cdt', 'presents', 'alice', 'wonderland', 'catonsville', 'dinner', 'posted', 'territory', 'location', 'calgary', 'alberta', 'canada', 'job', 'category', 'jobs', 'news', 'today', 'free', 'school', 'funding', 'plans', 'lack', 'transparency', 'manchester', 'city', 'council', 'details', 'saving', 'plan', 'depressing', 'deprived', 'hardest', 'hit', 'interested', 'professional', 'global', 'translation', 'services', 'fitness', 'float', 'full', 'model', 'dead', 'david', 'cook', 'mostest', 'beautiful', 'smile', 'piss', 'cnt', 'stand', 'lick', 'asses', 'beware', 'blue', 'meanies', 'thebluemeanies', 'como', 'perde', 'os', 'dentes', 'warcraft', 'alisson', 'exciting', 'hello', 'whats', 'happening', 'gearing'

First 10 answers to queries 1 and 25:

* Query 1:

MB001 Q0 30260724248870912 1 1.0 myRun

MB001 Q0 30198105513140224 2 0.993 myRun

MB001 Q0 30275282464153600 3 0.887 myRun

MB001 Q0 30167063326629888 4 0.887 myRun

MB001 Q0 30016851715031040 5 0.887 myRun

MB001 Q0 30016488928706560 6 0.887 myRun

MB001 Q0 34952194402811904 7 0.846 myRun

MB001 Q0 33823403328671744 8 0.846 myRun

MB001 Q0 32504175552102401 9 0.846 myRun

MB001 Q0 32415024995631105 10 0.846 myRun

* Query 25:

MB025 Q0 32609015158542336 1 0.999 myRun

MB025 Q0 31738694356434944 2 0.999 myRun

MB025 Q0 31550836899323904 3 0.999 myRun

MB025 Q0 31286354960715777 4 0.999 myRun

MB025 Q0 32685391781830656 5 0.901 myRun

MB025 Q0 32528974961713152 6 0.901 myRun

MB025 Q0 31320463862931456 7 0.901 myRun

MB025 Q0 30093525102108674 8 0.901 myRun

MB025 Q0 32541161675558912 9 0.893 myRun

MB025 Q0 29974357501550592 10 0.798 myRun

**Final results:**

This is the evaluation of our system using trec\_eval script by comparing our results (results.TREC) with the expected results.

runid all myRun

num\_q all 49

num\_ret all 36053

num\_rel all 2640

num\_rel\_ret all 2142

map all 0.2710

gm\_map all 0.1932

Rprec all 0.2931

bpref all 0.3040

recip\_rank all 0.5599

iprec\_at\_recall\_0.00 all 0.6505

iprec\_at\_recall\_0.10 all 0.5087

iprec\_at\_recall\_0.20 all 0.4361

iprec\_at\_recall\_0.30 all 0.3891

iprec\_at\_recall\_0.40 all 0.3603

iprec\_at\_recall\_0.50 all 0.3019

iprec\_at\_recall\_0.60 all 0.2182

iprec\_at\_recall\_0.70 all 0.1755

iprec\_at\_recall\_0.80 all 0.1371

iprec\_at\_recall\_0.90 all 0.0835

iprec\_at\_recall\_1.00 all 0.0323

P\_5 all 0.3755

P\_10 all 0.3306

P\_15 all 0.3197

P\_20 all 0.2990

P\_30 all 0.2762

P\_100 all 0.1788

P\_200 all 0.1317

P\_500 all 0.0763

P\_1000 all 0.0437

**Discussion**: From the final results, it shows that our “map”, which represents average precision, is 0.2710 (around 27%) for the whole test queries and “P\_10” (precision in the first 10 documents retrieved) is 0.3306. We believe that the result is not bad since we only have 45899 documents. Overall, our algorithms and data structure did a fairly good job, for example, we remove punctuation/symbols and stopwords. And we use matrices and formulas to compute similarity cosine, which can make our results more relevant and reliable.