Amit Nadam 205851819

Amit Vigdor Bart 314938333

Search Engine Report

**Part A**

1. **Software Design:**
2. **Classes:**

**Document Class** Holds the information of a tweet including:

* tweet\_id
* tweet\_date
* full\_text
* url
* retweet\_text
* retweet\_url
* quote\_text
* quote\_urls
* term\_doc\_dictionary
* doc\_length
* **max\_tf**
* **unique\_terms**

\*the fields colored in red are fields that we added to the class.

**Reader Class**

Read all the documents in the corpus.

**Parse Class**

Handles the parsing process for a single document.

For each document, we are tokenizing by these rules:

* Hashtags
* Urls
* Mentions/Tags
* Lower-Upper cases
* Percentages
* Numbers without units
* Names and Entities
* Currency cases
* Ignoring terms  
  \*the rules colored in red are rules that we added to the parser.

In Lower-Upper cases rule, we hold a dictionary of terms with Boolean values. Terms which start with a capital letter will be suspicious to be in capital letters in all the corpus. The first time we met a term with capital letter, we set a False value that means we didn't see the term in lower case and then when we meet at another time the same term in lower case we change the value into True. If we met first a term with lower case, we set a True value. Each time after finish parsing 500,000 documents we load the dictionary to the disk. At the end of parsing and indexing the corpus we will merge it with OR operation. If one is True, all of them will be True.

**Indexer Class**

Handles indexing the terms at each tweet. In this class, we hold two dictionaries, one for inverted index which use to hold about each term the amount of documents that contains the term (dt) and the total appearances of the terms in the corpus (sum\_fij), the other for the posting which contains 27 dictionaries, one for each letter and one for other characters. Each posting dictionary contains for each term the **tweet id, max\_tf, unique\_terms, fij and tf.**

This class is responsible to save the inverted index and the 27 dictionaries after indexing 500,000 documents by using the function named *write\_files*. At the end of the parsing and indexing all the corpus we will merge the posting files and inverted index into 28 files – one for the inverted index and the other 27 are for each letter's (a-z and specials) posting.

**Searcher Class**

Handles searching the documents that may be relevant for the specific query. In this class we implement the WordNet approach, where for each term in the query we expand the query with its synonyms and hyponyms. For each term in the expanded query we calculate the cosine-similarity formula for all the documents that are in the posting of the term.

**Ranker Class**

Handles ranking the documents by cosine similarity formula and returning the minimum between 2000 to the length of the list relevant documents.

**Stemmer Class**

In this class, we used the Porter's stemmer.

**Search\_engine file**

Responsible for running the engine.

In the file we parsing and indexing the documents in the corpus.

When finishing reading, parsing and indexing all the documents, we have 3 functions to merge each files in the disk into 3 files – capital letter, inverted index and posting.

In addition, after the first step in this class (reading, parsing, indexing and merging the files) we can search queries as a list or as a file text.

**Utils file**

This file contains all utility functions such as save object, load object, delete object index (we use delete in the merge functions) and load inverted.

1. **Limit Memory:**

In order to know how many documents we need to parse and index before writing the data structures of inverted index, posting and the capital letters, we made experiments. Each time while running the engine, we changed the documents interval size that will saved in the RAM without writing it to the disk.

1. **Posting Files:**We saved the posting file in '.pkl' format. We saved 27 pickle files for each letters and one for non-letter, the files contains the details of documents for each terms. We chose to use pickle format because pickle works good for Python objects.
2. **Partial tweet set and Inverted index (dictionary and posting file):**

The size we chose for the set is 500,000 tweets. We wanted to choose a number as high as possible to minimize the access to the disk and we made a lot of experiments in order to get the number that gives us the best trade-off between execution time and RAM usage.

The data structure holding the posting files is one dictionary that contains 27 sub-dictionaries, one for each letter in English and one for special cases (such as numbers and non-letter characters) which saved in 27 files of '.pkl' format, one for each sub-dictionaries. The sub-dictionaries contain the terms as a key and the posting as a value. The posting holds data about the document**: tweet id, max\_tf, unique\_terms, fij, tf.**

At the beginning, we chose to hold one dictionary that contains all the terms and posting together without separation by letters so that in the writing we need to separate them by a letter or non-letter – the operation takes a lot of time due to we chose to change this to one dictionary with 27 sub-dictionaries.

1. **The details of terms/documents:**

We chose to hold the tf in posting for each term-document, sum\_fij in terms dictionary for each term-document.

1. **The new rules for Parsing:**
2. **Currency rule:**

When we see a term from the following forms:  
Number[$,£,€,¢] / Number [$,£,€,¢]   
(the difference between the two is the space)

For example, (took from tweet with id: 1288849931545899009)  
$100 -> [$,100] -> $100  
(took from tweet with id: 1288828637639139328)  
$ 150 -> [$,150] -> $150  
Before tokenizing -> After tweet tokenize -> After rule.

1. **Ignoring terms rule:**

When we see terms like: rt, http, https, www, twitter, com, co, punctuators and some punctuators we added.

For example, (use the example from the assignment instructions)   
<https://www.instagram.com/p/CD7fAPWs3WM/?igshid=o9kf0ugp1l8x>

saved as:  
[Instagram.com, p, CD7fAPWs3WM, igshid, o9kf0ugp1l8x]

We chose to ignore them because they have no value in the retrieval and they might cause bad results due to their high frequency. These terms don't represent the corpus like stopword.

1. **Currency rule** – In order to give meaning to the number near the sign currency and can help us recognize tweets referring money.
2. **Ignoring terms rule** - We chose to ignore them because they have no value in the retrieval and they might cause bad results due to their high frequency. These terms don't represent the corpus like stopword.
3. **Algorithm of Ranker Class:**

In Ranker class we used the list which returned from the Searcher Class. We placed the algorithm of the ranking in the searcher in order to keep the Ranker class as simple as possible.

We moved on each term in the expanded query (we used WordNet). We hold a dictionary of terms and their fiq (number of repetitions of term x in the query). For each term, we read the posting file of it, calculate the wij, wij\*fiq and saved in a dictionary (term as keys and calculate wij, wij\*fiq as values) in order to calculate at the end the cosine similarity formula. When fiq = wiq.

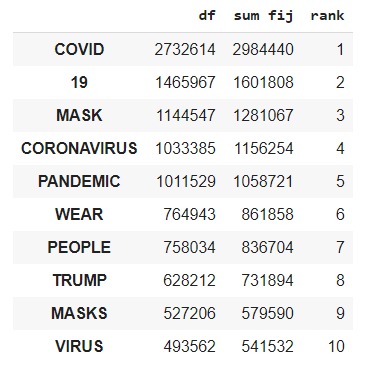
1. **Open Source:**

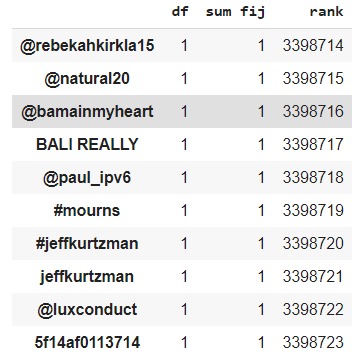
There is no use in open source code.

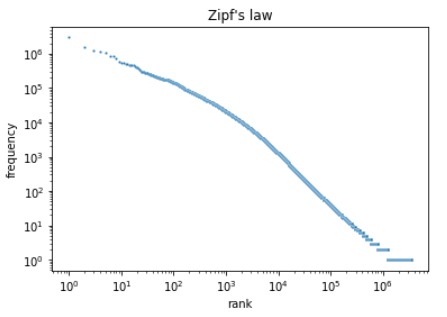
1. We ran the engine on hardware of 8 GB RAM and intel core i7 8th Gen.

|  |  |
| --- | --- |
| Advantages | Dis- Advantages |
| RAM frugal | The WordNet approach adds to the query many terms that aren't in the corpus |
| Decrease the amount of writing and reading from the disk | Bad design for the parser (a lot of if-else) |
| Separating for sub-functions | Can't parse acronym |

1. **After processing the tweets:**
2. How many different terms there are in the corpus with stemming? **3,398,723 terms**
3. How many different terms there are in the corpus with stemming? **3,361,150 terms**
4. How many different terms are numbers in the corpus? **266,345 numbers**
5. Top 10 most common terms in the corpus:



Top 10 least common terms in the corpus:



Yes, it looks like the Zipf's law graph shown in the lecture.

1. **Posting Size:**

Without stemming – 4.28 GB  
With stemming – 4.28 GB

1. **Time running parsing and Indexing** – 1:20 hours  
   In our engine, we do the parsing and indexing for documents together (we don't parse all of them and after index them)

|  |  |
| --- | --- |
| Tweet id | Explain |
| 1287376214492028928 | This tweet got a high rank although there isn't high correlation between the tweet to the query – because our approach – WordNet, where we expand our query with synonyms. |
| 1287423561645228032 |
| 1291785549846589440 | Rules - Capital letters, number. There were many RTs to the original tweet so all of them get high ranking. There is a lot of correlation between the tweet to the query. |
| 1291786635097657344 |
| 1291791311998029824 |

|  |  |
| --- | --- |
| Tweet id | Explain |
| 1291767378896920577 | Rules - Capital letters, hashtags, number. There were many RTs to the original tweet so all of them get high ranking. |
| 1284090741011165184 |
| 1284092724065574918 |
| 1284095044740358144 |
| 1284096710554206208 |

1. The results of the queries are based on parsing and indexing without stemming.  
   1. Dr. Anthony Fauci wrote in a 2005 paper published in Virology Journal that hydroxychloroquine was effective in treating SARS.  
     
   2. The seasonal flu kills more people every year in the U.S. than COVID-19 has to date.  
     
     
     
     
     
     
     
     
     
     
     
   4. The coronavirus pandemic is a cover for a plan to implant trackable microchips and that the Microsoft co-founder Bill Gates is behind it

|  |  |
| --- | --- |
| Tweet id | Explain |
| 1281761456170639361 | The tweet doesn't exist. We can't see the tweet. |
| 1285676416479133704 | Rules – Entities (Bill Gates) There were many RTs to the original tweet so all of them get high ranking. There is a lot of correlation between the tweet to the query. |
| 1285818210865078273 |
| 1286022794916925441 | The tweet doesn't exist. We can't see the tweet. |
| 1286022820145684480 |

7.Herd immunity has been reached.

|  |  |
| --- | --- |
| Tweet id | Explain |
| 1281709555420991490 | There is a lot of correlation between the tweet to the query. |
| 1282065712413122563 | There were many RTs to the original tweet so all of them get high ranking. There is a lot of correlation between the tweet to the query. |
| 1282152719990484992 |
| 1282152818799853568 |
| 1282159420215037953 |

|  |  |
| --- | --- |
| Tweet id | Explain |
| 1286645991593185280 | This tweet got a high rank although there isn't high correlation between the tweet to the query – because our approach – WordNet, where we expand our query with synonyms. |
| 1285276863527444480 |
| 1290528323961929729 |
| 1282478942167789568 |
| 1283811161981558788 |

8. Children are “almost immune from this disease.”