Assignment 1 - CSI4107

Students

Samuel Daly - 8173488
Ryan Matte - 300027432
Michel Moore - 300063096

Task Division

Step 1: Samuel Daly
Step 2: Michel Moore
Step 3: Ryan Matte
Step 4: Ryan Matte
Step 5: Samuel Daly

Folder Layout

```
├── Modules
│   ├── data
│   │   ├── document_word_dict.json
│   │   ├── frequency_dict.json
│   │   ├── stopwords.txt
│   │   ├── topics_MB1-49.xml
│   │   ├── trec-microblog11-qrels.txt
│   │   ├── trec-microblog11.txt
│   │   └── weighted_dict.json
│   ├── indexer.py
│   ├── preprocessor.py
│   └── query.py
├── requirements.txt
├── create_index.py
└── create_results.py
```

How to run the program

topics_MB1-49.txt must be converted to XML

1. Rename topics_MB1-49.txt to topics_MB1-49.xml
2. Add to first line: `<data>`
3. Add to last line: `</data>`

Windows

1. To install necessary packages, run ***pip install -r requirements.txt***
2. To create the inverted index, run ***python create_index.py***
3. To create the results, run ***python create_results.py***

MacOS

1. To install necessary packages, run ***pip3 install -r requirements.txt***
2. To create the inverted index, run ***python3 create_index.py***
3. To create the results, run ***python3 create_results.py***

Functionality of the program

For this assignment, we had to work in order of steps. The first step was to pre-process all of the tweets. The second step was to build the inverted index and produce a modified weighted dictionary using the modified tf-idf weighting scheme $w_{iq} = (0.5 + 0.5\, tf_{iq}) \bullet idf_i$. The third step was to rank the documents in decreasing order of similarity scores using the cosine method. Finally, upon running ***create_results.py***, the program called to ***query.py*** to perform document ranking for every given query in ***topics_MB1-49.xml*** and saved the results in ***results.txt***.

Preprocessing

All of the preprocessing for the tweets is done in the **processor.py** file.

We first created a function that takes the file path for the tweets and the file path for the stopwords. In this function we start off by reading the file of tweets and the goal is to process them to create a vocabulary that we will use to create the inverted index and then use for the future steps. We initially remove the links since we do

not want those in the vocabulary. Next, we stem the words to create a better vocabulary. To do this we use the *English Stemmer* from the NLTK library. This simplified the stemming for us. Next steps were to remove the punctuation, the stopwords and all the words that also contained numbers in them. We realized that the punctuation removal was leaving some weird unicode text behind, so we added more processing to get rid of those. This process was done for each tweet and therefore created a dictionary for each tweet that contained the tokenized, stemmed and processed words. The data structure used for this step is a dictionary which allows us to keep a list of words for each tweet id. It makes it easy to access in the future and it is very maintainable. This dictionary was saved under the file *document_word_dict.json* which we then use to do the in the future steps. Once the processing was done for each tweet and it was saved in a dictionary for us to refer, we went ahead and built the inverted index.

Indexing

All of the indexing is done in the **indexer.py** file. The indexing can be split into 2 parts:

1. Frequency dictionary (*frequency_dict.json*)

The first part involved creating an inverted index with frequency values in the form of a dictionary. This was done by transforming and inverting ***document_word_dict.json*** and saving this modified dictionary into ***frequency_dict.json***.

We first iterate through ***document_word_dict.json*** and append each word to a new array called **all_words** (no duplicates). Then, we transform this array into a dictionary called **frequency_dict**. From there, we iterate through **document_word_count_dict** (word count dictionary) to add the inner key values to **frequency_dict**. Finally, we save **frequency_dict** as a json file to the data folder.

2. Weighted dictionary (*weighted_dict.json*)

The second part involved adding weight values to each document for each word. This was done by modifying **frequency_dict** and saving this modified dictionary into ***weighted_dict.json***.

We first deep copy **frequency_dict** to a new dictionary called **weighted_dict**. For every word in this new dictionary, we find the document frequency, calculate the max frequency, and calculate the inverted document frequency. Now, for every document ID attributed to said word, we calculate the word frequency and use this value to calculate the weight value. Then, we set the weight value to its corresponding document ID. Once the initial loop has iterated through all words, we save **weighted_dict** as a json file to the data folder.

Retrieval and ranking

All of the query ranking is done in the **query.py** file.

The first step involves reading and parsing each query from ***topics_MB1-49.xml***. This is done by importing ElementTree and utilizing its parse method to retrieve each query. For each query, we perform preprocessing on all words in the query (remove links, remove punctuation, remove stopwords, etc.). Once the query words are processed, we then calculate and return both weighted and frequency vector arrays using the same methods used to create the weighted dictionary (indexing step).

For every word in the weighted vector array, we add this word to a new set called **unique_set_of_documents**. Then, for every document ID in **unique_set_of_documents**, we add the document ID to a new dictionary called **document_vect**. For every word in the weighted vector array, we iterate through each document in **unique_set_of_documents** and insert the counter variable as key and the corresponding weighted value from **weighted_dict** as value to the document key in **document_vect**.

For every document ID in **document_vect**, we calculate the cosine similarity value, then we append this value and its corresponding document ID to a new array called **document_ranking**. Next, we sort **document_ranking** by highest value first (decreasing order). For every document in this newly sorted dictionary, we write a line to ***results.txt***. This line contains the topic_id, Q0, docno, rank, score, and tag. Here is an example of the structure:

```
1 Q0 30260724248870912 1 0.9944520157716682 myRun
```

The loop breaks at the 1000th document to ensure there are no more than 1000 documents per query.

Future changes

Currently, a problem is that part of our preprocessing code can be found in 2 separate Python files (***preprocessor.py*** and ***query.py***). This means we have some duplicate code. In the future, we plan on storing this function in one place, and having both locations call to this one function. That way, future changes to our preprocessing would only require modification in one location.

Results

It is important to note that ***create_results.py*** creates different ***results.txt*** files every time it runs. Documents with identical scores are interchangeable.

For example, in query 1, the first 2 ranked documents are interchangeable:

Option 1:

```
1 Q0 30260724248870912 1 0.9944520157716682 myRun
1 Q0 30275282464153600 2 0.9944520157716682 myRun
```

Option 2:

```
1 Q0 30275282464153600 1 0.9944520157716682 myRun
1 Q0 30260724248870912 2 0.9944520157716682 myRun
```

Since both scores are identical, their ranks are interchangeable. Therefore, document 30260724248870912 is rank 1 and document 30275282464153600 is rank 2, or document 30275282464153600 is rank 1 and document 30260724248870912 is rank 2. This logic applies to all other documents with identical scores.

The following scores are the result of one specific ***results.txt*** file:

With everything done and working, we went ahead and ran the **trec_eval script** to test our results with the qrels file that we were supplied with. To run the script, we used this command: *./trec_eval -m map -m P.10 trec_microblog11-qrels.txt results.txt*

After running for the MAP (Mean Average Precision and the P10 (Precision in the first 10 documents retrieved), these were our results:

```
map                     all 0.2771
P_10                    all 0.3020
```

MAP represents an overall performance of our searching, and we managed to achieve a 27.7% Mean Average Precision. Considering we have a limited data set, and our data set is only considered of tweets, we think this is pretty good.

Looking at the precision in the first 10 documents, we achieved 30.2% in precision. We think we could have achieved a higher precision for the first 10 documents if we tweaked the pre-processing. With some better pre-processing, we believe the precision for the first 10 documents could have been slightly higher, but overall we are satisfied with our results.

Sample tokens from the vocabulary

Vocabulary size = 53310 words

```
ottawa
human
societi
adopt
chicacuppacupp
matter
rich
poor
life
money
call
refundcheck
name
class
oceanographi
multimedia
techniques
tvd
death
burnley
teen
controversi
control
contribut
dec
express
hurt
lol
woof
vintag
red
leather
sailor
nautic
jojoluvsjon
cesarmillan
savelennox
calm
influenc
comendo
cama
euri
nowplay
fear
ghost
```

```
pig
dont
minut
analys
chandeli
primeiro
ofici
novo
millan
hora
renata
peixoto
pedro
glad
enjoy
stuff
junebug
glori
snowdog
god
son
ya
rid
sooni
help
dude
worry
ella
pretti
green
nice
cuddl
buddi
handmad
esti
post
labrador
retriev
adult
eukanuba
food
gent
tenho
que
consola
prima
aqui
pq
sabe
mando
resposta
dizendo
se
era
naum
```

Sample queries

These queries are the result of one specific ***results.txt*** file.

Query 3

"Haiti Aristide return"

Results

1. 32384227123134464 Haiti allows ex-president's return: Jean-Bertrand Aristide, who was Haiti's first democratically elected leader,... http://aje.me/fQ4j4T
2. 32211683082502144 #int'l #news: Haiti opens door for return of ex-president Aristide: PORT-AU-PRINCE (Reuters) - Haiti'... http://bit.ly/gSIFwd #singapore
3. 29296574815272960 Haiti – Aristide : His return, an international affair… – Haitilibre.com http://bit.ly/gzyLXG #haiti
4. 32411439918489600 Haiti allows ex-president's return: Jean-Bertrand Aristide, who was Haiti's first democratically elected leader,... http://aje.me/fQ4j4T
5. 32387196078006272 Haiti allows ex-president's return: Jean-Bertrand Aristide, who was Haiti's first democratically elected leader,... http://aje.me/fQ4j4T
6. 29613127372898304 If Duvalier Can Return to Haiti, Why Can't Aristide? – New America Media http://bit.ly/eCWStk #haiti
7. 32080564265680898 Haitian Politics - Rev Jeremiah Wright Wants ARISTIDE To Return To Haiti: Fadel, you just tell the truth and the... http://bit.ly/hdEvaZ
8. 31034174752169984 Haitian Politics - Rev Jeremiah Wright Wants ARISTIDE To Return To Haiti: Hey. Non NOn NOn pou okin rezon aristi... http://bit.ly/ekgjqX
9. 33711164877701120 Haiti's former president Jean-Bertrand Aristide vows to return http://gu.com/p/2nvx3/tf
10. 34694060157435904 Former Haitian President Aristide has been issued with a new passport enabling him to end exile and return to Haiti, from AFP

Query 20

"Taco Bell filling lawsuit"

Results

1. 31043176684851200 Oxymoron alert: "The lawsuit is bogus & filled with completely inaccurate facts" Taco Bell President said. Inaccurate facts? Freudian slip?
2. 29906116062220290 Lawsuit: Taco Bell Ground Beef Is Really Just "Meat Filling" - @consumerist http://consumerist.com/2011/01/lawsuit-says-taco-bell-ground-beef-is-really-just-taco-meat-filling.html?utm_source=streamsend&utm_medium=email&utm_content=13297631&utm_campaign=Fo
3. 31082136219947008 Taco Bell Counters 'Meat Filling' Charges in Lawsuit With Print, Web Effort http://goo.gl/fb/H9wcB
4. 29853985930219520 That ain't necessarily "beef" in your Taco Bell burrito...a new lawsuit wants the chain to label it "taco meat filling" instead.
5. 32443636847218688 The Ins And Out Of Small Claims: Filling a civil lawsuit against an organization or person in hopes of col... http://tinyurl.com/4s8h474
6. 29158340424634368 Lawsuit Loans – Filling the Need For Funding http://bit.ly/hYuw1C
7. 30048091021246466 RT @bittman: Taco Bell's 'meat' filling is 36% meat. Mostly oats. Which isn't a bad thing, but let's be real: http://bit.ly/dSX8OF
8. 30700675742572545 I wonder what makes all the people who want taco bell to go vegetarian think that they will put real vegetables in the filling.
9. 30019159723089920 Taco Bell's 'meat' filling is 36% meat. Mostly oats, evidently. Which isn't a bad thing, but let's be real: http://bit.ly/dSX8OF
10. 29902271479287808 How Much Actual Beef Is in Taco Bell's 'Meat Filling'? | The Blaze http://goo.gl/jrXLK