Assignment 2 - Information Retrieval Systems

CSI 4107
Winter 2023
Due Date: Apr 2nd, 2023

Course Coordinator: Diana Inkpen

Group 33
Mai Lyn Puittinen 300124762
Shreya Langhe - 300109724
Joanna Wang 300119385

GitHub link: https://github.com/JjwangG/4107Assignment1

Task Delegation

Name	Task
Mai Lyn Puittinen	Implementation and running of Method 1: FastText/Expand Queries Improvement of Assignment 1 functionalities (indexing runtime, map score)
Shreya Langhe	Setting up and retrieving map score for both methods Improvement of Assignment 1 functionalities (indexing runtime, map score)
Joanna Wang	Implementation and running of Method 2: BERT

Program Functionalities

The previous files from assignment 1 still have the same initial functionalities. In addition to this, for each method, another file with its own functionalities was made. The chosen experiments for this assignment were option 1. BERT and option 2. Query Expansion

ExpandQuery.ipynb:

This file uses FastText to expand the query and then continues on in the same way that assignment 1 does. The program first loads the FastText model cc.en.300.bin that will be used to generate other query terms of relevance. Queries are represented in an array where each query is a string, read from cleaned_queries.txt (generated from Assignment 1 Retrieval). We then loop through each query, and for each word use the FastText model to add words of relevance. The final output is expanded_queries.txt which contains all original and extended query terms.

Running Instructions

- Files required: cc.en.300.bin, cleaned_queries.txt
- Model downloaded from: https://fasttext.cc/docs/en/crawl-vectors.html#models (English, bin)
- We ran ExpandQuery.ipynb in Jupyter Notebooks in order to be able to install fasttext and access cc.en.300.bin locally. Run all cells.

BERT.ipynb:

This file takes the results file from the original IR assignment 1 and takes all the documents that match each query to recalculate the cosine similarity score using the BERT built in methods. After completing the steps from assignment 1 it the program takes all the queries and the ranked docs and encodes them using sentence embedding (function imported from the HuggingFace sentence_transformers library using the all-mpnet-base-v2 model) to produce

vectors representing each query/doc. Using a function from the sklearn.metrics.pairwise module, the cosine similarity between each query and their original ranked docs is calculated, and the files are reranked. The final result is bert_results.txt which contains all of the final rankings.

Important .txt files
cleaned_queries.txt - query terms with removed stop words and punctuation
expanded_queries.txt - expanded query terms and original query terms generated in ExpandQuery.ipynb

Running Instructions

In order to more easily use the IR models, FastText and BERT are executed in python. As a result, there are multiple compilations and runs necessary to achieve the final results.

FastText - Expand Queries:

1. Open terminal to project directory
2. If Results.txt already exists in the project, delete the file
3. Ensure expanded_queries.txt exists. If not, see ExpandedQuery.ipynb explanation above and run.
4. Open Main.java and set the retrieval object as QERetrieval r = new QERetrieval
5. Compile javac Main.java
6. Run java Main
7. Sit and wait for the program to write the file Results.txt
8. Done

BERT:

1. Open terminal to project directory
2. If Results.txt already exists in the project, delete the file
3. Compile javac Main.java
4. Open Main.java and set the retrieval object as Retrieval_Bert r = new Retrieval_Bert
5. Run java Main - this will output the “input_bert.txt” file
6. Add all the documents, the input_bert.txt file, and the cleaned_queries.txt file to the same directory as the BERT.ipynb file.
7. If you wish to embed new queries, run the 7th cell from the top and add the outputted .npy files to the embed folder.
8. Run all the cells in BERT.ipynb and wait for the bert_results.txt file

To change the documents being read, place/remove them in the docs folder. To change the queries being used, change the queries.txt file.

Data Structures and Algorithms

FastText:

FastText expands the queries by using the cc.en.300.bin model which was trained on Common Crawl and Wikipedia using CBOW with position-weights.
Generally FastText uses a trained model based on texts that are associated with labels and vice versa. The model learns how to associate a certain word with other words of relevance. The relevant words are added to our original query terms and then compared to the documents. This way there is more material to compare to get more accurate results.

First 10 results of queries 3 and 20:

```
3 Q0  AP880323-0053  1 0.10548361812039325 my_search
3 Q0  AP880419-0133  2 0.10000934812633737 my_search
3 Q0  AP881031-0350  3 0.09658778163487232 my_search
3 Q0  AP880615-0319  4 0.09358749192260768 my_search
3 Q0  AP880423-0086  5 0.09315625796837922 my_search
3 Q0  AP880711-0238  6 0.09225263570197918 my_search
3 Q0  AP881101-0216  7 0.09187199708695809 my_search
3 Q0  AP880513-0106  8 0.09101089940288615 my_search
3 Q0  AP881102-0262  9 0.08919746242796939 my_search
3 Q0  AP880925-0076  10 0.08912420002334356 my_search
```

```
20 Q0  AP881111-0092  1 0.29721435141481595 my_search
20 Q0  AP881110-0035  2 0.26259124047376686 my_search
20 Q0  AP881122-0171  3 0.2513544536436202 my_search
20 Q0  AP881123-0037  4 0.22186140001112867 my_search
20 Q0  AP880504-0233  5 0.16158681444743458 my_search
20 Q0  AP880627-0239  6 0.15629803418328236 my_search
20 Q0  AP880616-0020  7 0.12085929611078104 my_search
20 Q0  AP880518-0359  8 0.097385152842866 my_search
20 Q0  AP880406-0143  9 0.08439230121650154 my_search
20 Q0  AP881111-0107  10 0.08406625729904675 my_search
```

BERT:

To do the BERT method portion we use the all-mpnet-base-v2 model which is a model that has been trained for an all around use. The original rankings from our Retrieval.java file are represented as arrays where each array has the query as the first element and the 1000 top ranked docs as rest of the elements. Since the encode function costs a lot of time, we used sentence embedding to make vectors of all the queries and documents ahead of time and saved these vectors in .npy files so we did not have to rerun the sentence embedding again.

To use the vectors we created with the sentence embedding, we were able to load each .npy file (containing vectors for each document in the corresponding file) and create a hashmap using

document number as the key and vector as the value. We use BERT to then calculate sentence cosine similarity between the queries and documents. We do this by taking the documents and queries and representing them as strings and then with BERT we can use sentence embedding to map those strings as vectors and calculate the cosine similarity.

First 10 results of queries 3 and 20:

```
3 Q0 AP881010-0256 1 0.40528757041067776 my_search
3 Q0 AP880613-0204 2 0.3234662331215233 my_search
3 Q0 AP880325-0234 3 0.30060260637076663 my_search
3 Q0 AP880420-0298 4 0.2937189627236611 my_search
3 Q0 AP880607-0043 5 0.29294130914735456 my_search
3 Q0 AP881128-0018 6 0.2910804754029194 my_search
3 Q0 AP880411-0283 7 0.284072652842541 my_search
3 Q0 AP880914-0156 8 0.2824191531880549 my_search
3 Q0 AP880929-0019 9 0.2772661449390252 my_search
3 Q0 AP881008-0019 10 0.27303199439719017 my_search
```

```
20 Q0 AP880709-0185 1 0.30683345391850436 my_search
20 Q0 AP881215-0243 2 0.3065858524520516 my_search
20 Q0 AP881012-0153 3 0.3056346298159459 my_search
20 Q0 AP881025-0138 4 0.2950909082481129 my_search
20 Q0 AP880722-0189 5 0.2796471380847652 my_search
20 Q0 AP880405-0272 6 0.27834725997896287 my_search
20 Q0 AP880614-0174 7 0.27610007024870764 my_search
20 Q0 AP880519-0324 8 0.27051465451605394 my_search
20 Q0 AP880729-0087 9 0.26986970031126484 my_search
20 Q0 AP880809-0154 10 0.26935949435758544 my_search
```

Inverted Indexing:

For the inverted index, a few things were changed to make it run faster and more efficiently. The main point that drastically lowered the time was changing some of the ArrayList types to HashMaps. For example the data structure used to store tokens was an ArrayList. During the process of indexing .contains was used to check if the tokens ArrayList contained any of the words being looped through. .contains has a time complexity of O(n), which means to loop through and check all words we would be doing O(n) for every word in every subdoc. Once we realized this we changed the tokens array to a HashMap that can then use .containsKey which has a time complexity of O(1). Obviously this was significantly better and more efficient of a data structure to use.

Mean Average Precision (MAP) and P@10

Changes and Improvements to Assignment 1

Our MAP score has greatly improved from Assignment 1 (0.0027) due to modifications in our code and methodologies. Initially, our program was unable to process all document files in a reasonable amount of time, resulting in our Assignment 1 inverted index based on only 10 of the 322 files. We were able to drastically improve the runtime of the inverted index by replacing ArrayList with HashMap and our results now reflect the entire document corpus. We were also able to improve the runtime of the preprocessing by reducing the number of regexes used (which was compiling each time it was used). Furthermore, a significant bug was detected regarding the value for the number of documents. In Assignment 1 we mistakenly used the number of document files (322) as opposed to the number of documents (79923). Changing this vastly improved our cosine similarity calculations and thus the MAP score.

Base Queries: 0.2414

runid	all	my_search
num_q	all	50
num_ret	all	2636982
num_rel	all	2099
num_rel_ret	all	2094
map	all	0.2414
gm_map	all	0.1335
Rprec	all	0.2513
bpref	all	0.2864
recip_rank	all	0.5374
iprec_at_recall_0.00	all	0.5950
iprec_at_recall_0.10	all	0.4523
iprec_at_recall_0.20	all	0.3637
iprec_at_recall_0.30	all	0.3150
iprec_at_recall_0.40	all	0.2731
iprec_at_recall_0.50	all	0.2430
iprec_at_recall_0.60	all	0.2059
iprec_at_recall_0.70	all	0.1632
iprec_at_recall_0.80	all	0.1208
iprec_at_recall_0.90	all	0.0765
iprec_at_recall_1.00	all	0.0338
P_5	all	0.3440
P_10	all	0.3100
P_15	all	0.2840
P_20	all	0.2720
P_30	all	0.2480
P_100	all	0.1518
P_200	all	0.1028
P_500	all	0.0547
P_1000	all	0.0325

Expanded Queries: 0.2496

runid	all	my_search
num_q	all	50
num_ret	all	50000
num_rel	all	2099
num_rel_ret	all	1697
map	all	0.2496
gm_map	all	0.1349
Rprec	all	0.2627
bpref	all	0.3022
recip_rank	all	0.5449
iprec_at_recall_0.00	all	0.5966
iprec_at_recall_0.10	all	0.4658
iprec_at_recall_0.20	all	0.3986
iprec_at_recall_0.30	all	0.3201
iprec_at_recall_0.40	all	0.2863
iprec_at_recall_0.50	all	0.2554
iprec_at_recall_0.60	all	0.2161
iprec_at_recall_0.70	all	0.1549
iprec_at_recall_0.80	all	0.1159
iprec_at_recall_0.90	all	0.0728
iprec_at_recall_1.00	all	0.0312
P_5	all	0.3600
P_10	all	0.3280
P_15	all	0.3227
P_20	all	0.3000
P_30	all	0.2747
P_100	all	0.1670
P_200	all	0.1084
P_500	all	0.0578
P_1000	all	0.0339

BERT: 0.0001

```
runid                   all     my_search
num_q                   all     50
num_ret                 all     49000
num_rel                 all     2099
num_rel_ret             all     20
map                     all     0.0001
gm_map                  all     0.0000
Rprec                   all     0.0008
bpref                   all     0.0087
recip_rank              all     0.0022
iprec_at_recall_0.00    all     0.0022
iprec_at_recall_0.10    all     0.0001
iprec_at_recall_0.20    all     0.0000
iprec_at_recall_0.30    all     0.0000
iprec_at_recall_0.40    all     0.0000
iprec_at_recall_0.50    all     0.0000
iprec_at_recall_0.60    all     0.0000
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.0000
P_10                    all     0.0000
P_15                    all     0.0000
P_20                    all     0.0000
P_30                    all     0.0000
P_100                   all     0.0008
P_200                   all     0.0005
P_500                   all     0.0005
P_1000                  all     0.0004
```

Discussion - FastText vs BERT :

In comparing the MAP score from ranking based on original query terms versus expanded query terms, there is only a slight increase. Intuitively, the increase makes sense as with an expanded vocabulary to search, there is a greater chance you will find documents of significance. However, we did find that the related query terms generated by the FastText model were not always helpful. Sometimes it was only variations of the capitalization of letters in the word which was a problem eliminated by setting the document corpus to lowercase before query term search. This may have skewed the query term weighting and detracted from possible increased accuracy.

```
In [6]: model.get_nearest_neighbors("Obama")

Out[6]: [(0.9087767601013184, 'Barack'),
         (0.7838232517242432, 'Obam'),
         (0.776511549949646, '0bama'),
         (0.7733059525489807, 'Obama.Obama'),
         (0.7692676782608032, 'Obama--'),
         (0.7583456039428711, 'OBama'),
         (0.7558523416519165, 'Obma'),
         (0.7557627558708191, '.Obama'),
         (0.7540246844291687, 'Obama-'),
         (0.7488862872123718, 'Obams')]
```

BERT should have resulted in a higher MAP score because BERT has been trained on a large corpus of text data and can capture the context and meaning of words and phrases, which is essential for understanding natural language.
On the other hand, fastText is a simple bag-of-words model that represents words as vectors and calculates the similarity between them using cosine similarity. While fastText is efficient and easy to use, it may not capture the nuances and complexities of language as well as BERT.

The BERT results we have included were generated by reading the document files by bit as opposed to line by line, resulting in faulty BERT vectors. We attempted to solve this problem, however after regenerating all the embeddings, we ran into more issues where documents were duplicated in query rankings. We were unable to ultimately solve this issue to find a possibly improved map score, however we believe the problem is related to our BERT program logic.