Alejandro Rocas
Lomis Chen
Nikhilesh Sivapatham
Bahaar Bhatia

CS 410 Documentation

The competition is based on the CORD-19 dataset for supporting research on COVID-19. The goal of the information retrieval project is to put together an IR/ranking system such that the results can be as accurate as possible against the test data using either supervised or unsupervised approaches.

The approach our group modeled our attempt after 2.2 and 2.4 respectively. The main portion of the code is located in two files named “search_eval_IR.py" and “extract_metadata.py.” Extract_metadata.py primarily serves two purposes: first it scrapes the sample XML search queries and extracts the query, questions, and narratives into text format to ease the overall processing; the second is to access the test_metadata.csv file and extract the relevant information it contains such as authors, abstract, title, and the uid to build up the content-base. Then, this data is formatted and outputted to two different files, namely output.dat and metadata.dat, respectively.

Since the core of the project is emulated from MP2, this means that search_eval_IR.py intrinsically has a similar structure to the MP2 code. In addition to that, the config_IR.toml file sets up the config for the overall project, this includes paths for search queries and contents, the default ranker, the language model to be used, and etc. What’s new in our project is i) implementing the extract_metadata.py to extract, clean, and simplify the original dataset -- the code is relatively easy to comprehend; ii) extending the original stopwords.txt file to create a more comprehensive version -- this was found on the internet which again was proven to be helpful in boosting the score; iii)scraping the additional *intro* content from original paper beyond the *abstract* section; and iv) attempting to tune the parameters through brute force and testing of different analyzers.

Some future improvements include utilizing different rankers or expanding past metapy. If possible, tuning without brute force can be a future improvement. Prior to using the default parameters for the okapi bm25 ranker the best tuned score that was produced was 0.6027417341816668, with other parameters for bm25 being with results as follows:

1.8 0.9 800
0.5933321073084418

1.8 0.74 800
0.6060514406728549

The PL2 ranker that we made in MP2 produced the worst result with 0.4504774849333016.

Testing with pivoted length resulted in scores around the mid .50s, same with the absoluteDiscount and the JelinkMercer. The DirichletPrior gave a score in the high .40s.

We've attempted many methods to improve our scoring given the realm of our knowledge. However, as next steps, there are also a few potential approaches we could explore as per our internal discussions and TA outreach. For one, we could leverage a better web-scraper to scrape more relevant content in the full papers to help supplement our content-base. Second we could also experiment with various other rankers or even create our own. Third, we could leverage other team's improvements on metapy to see if the additional improvements made would be beneficial to our ranking performance. And lastly we could leverage an additional filter such as logistic regression to help us prune and filter irrelevant search researches to further fine tune the scoring.

In order to run the code, you will need to follow the following set up instructions:

1) Please make sure you remove the idx_IR folder as you need to create that on your own machine: when you run search_eval_IR.py, it will recreate the folder.
2) It is recommended to use Python 3.5 to avoid version errors with code dependencies.
3) Prior to running the code, please make sure you have metapy installed as well. If this is not set up locally already, you may follow the tutorials found in MP2 to do this.

Once set-up is complete, please follow the remaining instructions to run the code:

1) Download the metadata.csv file from the test link
2) Grab the queries.xml file from the test link folder that can be accessed through this link
3) Rename metadata.csv to test_metadata.csv.
4) Rename queries.xml to test_queries.xml.
5) Run python extract_metadata.py ("python3 extract_metadata.py")
6) Run search_eval_IR.py and pass it the config_IR.toml ("python3 search_eval_IR.py config_IR.toml")

The code will then generate a predictions.txt file. To verify that the file is generated correctly it is recommended to delete the one in your local copy of the repository before running the sequence above. Once this step is complete, open the newly generated predictions.txt file and compare it to the one in the repository. The predictions file is what was sent into a live datalab to verify our results. The highest score we were able to achieve was 0.6155538743352519.

Of the four members of the group, Nikhil and Lomis were responsible for the early development including the set up, config, and adaptation of the code from MP2. Upon completion, Alejandro and Bahaar worked on testing different rankers, adding more stopwords, and parameter tuning of

the different rankers to see which yielded the highest score we have. From this process, we were able to determine that the OkapiBM25 ranker resulted in the highest accuracy. Alejandro and Nikhil both independently investigated different analyzers (unigram, bigram, combination of both etc. as well as different filters, stemming, POS tagging) but found that the default unigram chain alone was the most effective. Lomis and Bahaar reached out and connected with the TAs and learned more about how to leverage both logistic regression and neural networks to further improve our performance for which we're still at the stage of investigating. Lastly, Alejandro as the project lead kickstarted the doc and ppt and brought the team along to reach project completion.