Course Project

CS 242: Information Retrieval & Web Search

Winter 2024

**Build a Search Engine**

You must work in teams of 5. If you cannot find a partner, email the TA to connect you to other students who are looking for a partner.

*By the end of the second week, you must complete the formation of your team and email the instructor (and cc Shihab) with your project proposal (title and 1-2 paragraphs of description) for approval.*

Each project report must have a section called "Collaboration Details" where you should clearly specify the contributions of each member of the team.

No handwritten submissions allowed.

**Part A: Collect your data and Index with PyLucene or PyElasticSearch**

A1: You have the following options:

1. Crawl the Web to get Web pages using Scrapy (<https://scrapy.org/>) or jsoup (<http://jsoup.org/>). You may restrict pages to some category, e.g., edu pages, or pages with at least five images, etc.
2. Use PRAW to collect reddit posts.
3. (Twitter API may not be free anymore, check if interested) Use Twitter Filtered Stream API (<https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction>) to get Tweets. You can use Tweepy ([tweepy.org](file:///C:\Users\vagel\Downloads\tweepy.org)) if you use Python. (hint: Filter to only collect geotagged tweets, so you can then display them on a map in Part B.)
4. Your own ideas for a dataset are also acceptable, pending instructor approval.

Collect at least 500 MB of data.

We recommend using Python to make the integration with the Web app in Part B2 easier (as BERT uses Python), but you may also use Java (Lucene) if you prefer (but you have to figure how to make the Web app work if you choose to do a Web app).

A2: Index your data using Py[Lucene](http://lucene.apache.org/core/) (or PyElasticSearch but not Solr) or an equivalent from your language of preference (ask for approval from instructor).

You will be graded on the correctness and efficiency of your solution (e.g., how does the crawler handle duplicate pages? Is the crawler multi-threaded? How do you store the incoming tweets to maximize throughput?), and the design choices made when using PyLucene (e.g., did you remove stop words, and why? Or did you index hashtags separately from keywords and why?).

Deliverables:

1: Report (5-10 pages) in pdf that includes:

* 1. Collaboration Details: Description of contribution of each team member.
  2. Overview of the crawling system, including (but not limited to).
     1. Architecture.
     2. The Crawling Strategy.
  3. Overview of the Lucene indexing strategy, including (but not limited to).
     1. Fields in the PyLucene index, with justification (e.g., indexing hash tags separately due to their special meaning in Twitter).
     2. Text analyzer choices, with justification (e.g., removing stop words from web documents; using separate analyzers for hashtags and keywords).
     3. Report the run time of the Lucene index creation process. E.g., a graph with run time on the y axis and number of documents on the x axis.
  4. Limitations (if any) of system.
  5. Obstacles and solutions
  6. Instruction on how to deploy the crawler.
     + Ideally, you should include a crawler.bat (Windows) or crawler.sh (Unix/Linux) executable file that takes as input all necessary parameters.
     + *Example:* [user@server]./crawler.sh <seed-File:seed.txt> <num-pages: 10000> <hops-away: 6>  <output-dir>
  7. Instruction on how to build the Lucene index.
* Ideally you should include an executable file that takes as input all necessary parameters. *Example:* [user@serer] ./indexbuilder.sh <input\_dir> <Analyzer Options>

2: Zip file with your code

Submit your report and zip file through Canvas by **2/14**.

**Part B: Index your data using BERT and build Query Interface**

B1: Here, you will create an alternative index of your data, using BERT (dense representation) instead of Lucene (sparse, bag-of-words representation). You will use PyTorch (<https://pytorch.org/hub/huggingface_pytorch-transformers/>) to generate an embedding for each passage of your data and store them in a *faiss* index (<https://github.com/facebookresearch/faiss>). Passages should be up to 512 tokens (tokenized using the BERT parser) long for BERT to work. Hence, the output of the indexing will be a list of passages, along with their BERT embedding.

Note: You can use Tensorflow (<https://github.com/google-research/bert>) instead of PyTorch, but we will only support PyTorch in the class. Also, instead of using faiss, you can store the passage embeddings in a database like MySQL and write your own cosine similarity function.

Compute the indexing time for BERT and compare to the one for Lucene.

B2: Build command-prompt or Web application (extra credit) that inputs a query and an index choice (PyLucene or BERT) and outputs the top-k results.

For command-prompt, ideally a single program will run both indexes, but if not possible you can create two separate programs.

For the case of a Web app, it should have a textbox for the query and a radio button to pick the index. We recommend Flask or Django for the Web app, but you are free to use your own Web-based programming language (at your own risk).

Extra credit ideas:

1. (for Web app only) If your data has coordinates (e.g. geotagged tweets), show results on a map.
2. Show snippets for results. You can use ideas discussed in class or your own ideas to come up with a good snippet generation algorithm.

You will be graded on the correctness and interestingness of your solution; you will also be graded on the design choices made with respect to the indexes created (e.g., how do you split documents to passages, what schema you use in the database).

Deliverables:

1: Report (5-10 pages) in pdf that includes:

1. Collaboration Details: Description of contribution of each team member.
2. Overview of system, including (but not limited to):
3. Architecture
4. Details of how BERT was used (model choice, indexing schema).
5. Explain how you use the BERT index to do the ranking
6. Explain how you use the Lucene index to return results. E.g., what options did you use with the QueryParser? How were multiple fields in the Lucene index used with the QueryParser?
7. Report the run time of the BERT index construction. This should be comparable with the report on the run time of the Lucene index construction from report A. There should also be a discussion explaining the differences between the run times.
8. Compare the quality of the rankings of Lucene and BERT. Show examples where one is doing better than the other and explain.
9. Limitations of system.
10. Obstacles and solutions
11. Instructions on how to deploy the system.
12. Ideally, you should include an indexer.bat (Windows) or indexer.sh (Unix/Linux) executable file that takes as input all necessary parameters e.g.
    1. **Example:** [user@server] ./indexer.sh <input-dir> <output-dir>
13. Create a web-application (e.g. Web Archive) that can be deployed to a webserver like Tomcat.
14. Screenshots showing the system in action.

2: Zip file with your code

Submit your report and zip file through Canvas by **3/17**.

Each team will prepare a 5-min presentation video by the day before their scheduled presentation and email a link to the video to the TA and instructor. We will play the presentations in the last week. All members of each team should participate in the presentation. E.g., you could each present a part of the project and concatenate the presentation videos into a single video file.