Documentation for COSI132a Assignment 5: Elasticsearch

Date: 05/01/2021
Author: Daeja Showers daejashowers@brandeis.edu

Description: An Elasticsearch command line interface that allows the user to query a subset of documents from the 2018 TREC Washington Post Corpus

Build Instructions:

Install Python 3.7 locally or in your desired conda environment

Run <pip install –r requirements.txt> or <conda install --file requirements.txt> to install required python modules

Add the following code at the end of elasticsearch_dsl/query.py after installing the package:

```
class ScriptScore(Query):
   name = "script_score"
   _param_defs = {"query": {"type": "query"}}
```

I suggest searching 'elasticsearch_dsl' in your system search bar to find the folder.

This system requires multiple services to be running, so I suggest a tool like Windows Terminal to easily access several shells at once. The given commands were executed in Git Bash. Before following any of the instrutions below in a new shell, cd to the PA5 folder and activate any relevant conda environments.

Download the .jl file containing the data from here: https://drive.google.com/file/d/1Y03Cgf-84lua5cmBEt8-4JQKbdgvu3vG/view?usp=sharing and insert into the pa5_data directory.

Download the pretrained FastText embedding using this link: https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M-subword.vec.zip. Also insert it into the pa5_data directory.

Download Elasticsearch from https://www.elastic.co/downloads/past-releases#elasticsearch. Make sure you are choosing Elasticsearch 7.10.2.

To start the Elasicsearch engine run the following commands:

```
$ cd elasticsearch-7.10.2/
$ ./bin/elasticsearch
```

NOTE: You must call .bin/elasticsearch.bat on a Windows machine.
Leave this running. In a new shell, run the 3rd command in the provided scripts.sh file to to load the elasticsearch index (python load_es_index.py…).

After the index has finished loading, run the first command in the scripts.sh file to run the embedding service for FastText. Leave this running.

Open another shell. Run the second command in the scripts.sh file to run the embedding service for sBERT. There should be three shells total in the background.

Finally, open one last shell. You will interact with the search application here.

Run Instructions:

Run <python evaluate.py ----index_name INDEX-NAME --topic_id TOPIC_ID –query_type QUERY TYPE> within the PA5 directory in the command line to display results for the query specified. Use the index name given when running the load_es_index.py command (‘wapo_docs_50k’ is the given name in scripts.sh). The topic ids available can be found in the topics2018.xml file in the pa5_data directory. The query type specifies which topic field you would like to use as a query: ‘title’, ‘description’, or ‘narrative’. This runs a BM25 query with the standard text analyzer.

Other commands are possible for a more customized search.

By default the system will display the top 20 results, but adding the flag --top_k TOP_K will allow you to display TOP_K number of documents.

Adding the flag --u to your search will use a custom text analyzer that employs a snowball stemmer, an asciifolding token filter, a lowercase token filter and a standard tokenizer.

Adding the flag --vector_name VECTOR_NAME and giving either ‘sbert_vector’ or ‘ft_vector’ will use the sBERT or FastText vectors (respectively) instead of BM25. These both use standard text analyzers (the custom flag will be ignored, and display a message notifying you if you have attempted to use the custom analyzer with a vector).

Finally, running the command:

```
$ python evaluate.py --index_name wapo_docs_50k --produce_metrics
```

Will display tables for the first five topics in the topics2018.xml file that give normalized discounted cumulative gain at rank 20 scores for each combination of search method and query type.

Results:

Topic 321	Title	Description	Narrative
BM25+standard	0.81265	0.68446	0.62636
BM25+custom	0.47738	0.61541	0.56101
fastText	0.46762	0.51028	0.36749
sBERT	0.82318	0.37448	0.47048

Topic 336	Title	Description	Narrative
BM25+standard	0.82266	0.40198	0.43046
BM25+custom	0.33096	0.42904	0.35386
fastText	0.0	0.44523	0.36841
sBERT	0.33302	0.0	0.24465

Topic 341	Title	Description	Narrative
BM25+standard	0.76166	0.59263	0.896
BM25+custom	0.81394	0.81024	0.95092
fastText	0.63679	0.63066	0.80628
sBERT	0.82056	0.73713	0.84942

Topic 347	Title	Description	Narrative
BM25+standard	0.34976	0.46804	0.40892
BM25+custom	0.0	0.38545	0.57594
fastText	0.33333	0.25	0.22767
sBERT	0.22767	0.0	0.0

Topic 350	Title	Description	Narrative
BM25+standard	0.0	0.37169	0.344
BM25+custom	0.42986	0.79772	0.79772
fastText	0.0	0.29585	0.22767
sBERT	0.0	0.27024	0.35037

Analysis:

The BM25+standard search method was surprisingly effective for the first three queries, as I expected it to consistently perform the worst. There is a lot of variance in the results which I also did not expect. I realized that one query might be more effective than another based on how it is described, how ambiguous the search terms are, and how many relevant documents there are for a topic, but I didn't expect that within a topic some combinations of query types and search methods performed vastly better than others: for example for topis 366 BM25+standard when combined with the title had an NDCG@20 of 0.82266, but sBERT + narrative only 0.24465 and some other combinations didn't return any relevant documents. Some general trends:
sBERT does badly with the description category, titles produces heavily inconsistent search results while descriptions give the most consistent performance, and none of these search combinations easily outpace the others across the board (it really varies by query).

Additional Comments:

I have no idea how long I spent on this assignment. I am having an extraordinarily difficult semester and felt it was better to turn what I have in without integrating Elasticsearch into Flask so that I could focus on the group project. This took quite a while not due to reading APIs or overall complexity, but because the way the word document was put together made it very difficult to understand where to find what information and exactly what needed to be modified/implemented. I sat down for several days attempting to start this assignment before I finally successfully understood what it was asking for on the 5th or 6th readthrough. This could be a function of my ADHD, as I find it difficult to hold many pieces of information in my head at a time (I had to write down what variables were passed around to where to understand how the corpus was being accessed, how to access the document fields, how to access the vector embeddings, etc). However I have heard similar complaints from other classmates. I don't know if I would call this assignment difficult as much as I would just call it frustrating. The video Tu Jingxuan recorded was absolutely vital to completing this project.