HomeWork - 1

REPORT

Information Retrieval

Shri Datta Madhira

February 19, 2021

Softwares and Installation

The softwares needed to complete this assignment are,

1. PyCharm: To run python codes.
2. ElasticSearch: For indexing and retrieving the documents.
3. Kibana: It provides a beautiful interface to access all the indexes, and a developer tools page.
4. Docker Desktop: To run ElasticSearch and Kibana containers.

The installation of the softwares is straightforward.

PyCharm: For PyCharm, we need to download an executable compatible with your laptop.

Docker Desktop: The same procedure goes for Docker Desktop. If you are using Docker on Windows, there are some virtualization settings that need to be changed in the BIOS settings when booting up the laptop.

ElasticSearch & Kibana: There are elasticsearch.yml and kibana.yml files in the AP89_DATA folder which we have to run in PyCharm using *"docker-compose up"* command to download at first. Then we those containers can be accessed and started through Docker desktop.

Indexing

A document corpus containing 84,678 files is given in the AP89_DATA folder. A new index should be created in ElasticSearch using the ElastiSearch API for python. The command for it is ***es.indices.create()***.

Once the index is created, the documents in the corpus can be indexed into that index using a request body that determines the format in which the docs will be indexed. It is when we mention the removal stop words, stemming and other things that we want to do on the corpus.

Once indexing is done, we need to process the queries.

Query Processing

Next step is to process the queries so that they can retrieve as many relevant documents as possible. The ideal length for any query is to have 4-6 keywords that can fetch the most relevant documents. But, more often that not, we run into queries that are unusually long. They contain a lot of stop words and unnecessary words that can reduce the effectiveness of your retrieval models.

The first step is to remove the stop words from your query.

The second step is to remove the repeating and unnecessary words that can affect the query in a bad way.

The third step is to get document information for the docs relevant to your query words. For this, we need to send each word to ElasticSearch search API and retrieve the document IDs in which the word occurs. For retrieving all the possible documents for a word, we need to use scroll API. Else, we will only be getting 10 docs.

The fourth step is to get TermVectors for all these documents using the termvectors API. From this query, we will get to know the “term_freq”, “doc_freq”, and “total_term_freq”. Then store all these values in a file so that they can accessed quickly for retrieval models.

Retrieval Models

We have to run each query using different retrieval models.

1. ***Okapi TF***: This is a vector space model. It uses term frequency in each document to calculate the Okapi TF score that is ultimately used to rank the documents retrieved. The Okapi TF score formula is,

$$okapi\_tf(w,d) = \frac{tf_{w,d}}{tf_{w,d} + 0.5 + 1.5 \cdot (len(d)/avg(len(d)))}$$

tfw,d = term frequency of term ‘w’ in document ‘d’.
len(d) = Length of document ‘d’.
avg(len(d)) = Average length of all the documents in the corpus.

This value needs to be summed over all the words in that query and then you will get the score with which you can rank the document.

2. ***TF-IDF***: This is also a vector space model. But the scoring formula changes a little to get more accuracy in document retrieval. The formula is as follows,

$$tfidf(d,q) = \sum_{w \in q} okapi\_tf(w,d) \cdot \log \frac{D}{df_w}$$

okapi_tf(w,d): Okapi TF score for term ‘w’ and document ‘d’.
D = total number of documents in the corpus = 84,678.
dfw = no. of documents that contain the word ‘w’ (or) Document Frequency.

3. ***Okapi BM25***: This is a language based model. The scoring formula is,

$$bm25(d,q) = \sum_{w \in q} \left[\log\left(\frac{D + 0.5}{df_w + 0.5}\right) \cdot \frac{tf_{w,d} + k_1 \cdot tf_{w,d}}{tf_{w,d} + k_1\left((1-b) + b \cdot \frac{len(d)}{avg(len(d))}\right)} \cdot \frac{tf_{w,q} + k_2 \cdot tf_{w,q}}{tf_{w,q} + k_2} \right]$$

tfw,q = term frequency of term ‘w’ in query ‘q’.
k1, k2, and b are constants.

4. ***Unigram Language Model - Laplace Smoothing***: This is a language model that makes use of Laplace smoothing technique. The formula is,

$$\text{lm\_laplace}(d,q) = \sum_{w \in q} \log \text{p\_laplace}(w|d)$$

$$\text{p\_laplace}(w|d) = \frac{\text{tf}_{w,d} + 1}{\text{len}(d) + V}$$

V = Vocabulary Size, which depends on your indexing procedure.

5. ***Unigram Language Model - Jelinek Mercer Smoothing***: This is also a language model but it uses a different kind of smoothing technique called Jelinek Mercer Smoothing Technique. The formula is as follows,

$$\text{lm\_jm}(d,q) = \sum_{w \in q} \log \text{p\_jm}(w|d)$$

$$\text{p\_jm}(w|d) = \lambda \frac{\text{tf}_{w,d}}{\text{len}(d)} + (1-\lambda) \frac{\sum_{d'} \text{tf}_{w,d'}}{\sum_{d'} \text{len}(d')}$$

$\lambda \in (0,1)$ = is a smoothing parameter which specifies the mixture of the foreground and background distributions.

Evaluating the results

The results from these retrieval models are stored in “trec_eval” file. This file contains the code to evaluate your results.

For evaluating the results, we have to run trec_eval.pl file by inputting your results file from the terminal. The scores you get will displayed as output. This is done after every change to see if the scores have improved.
My results:

	Precision at 10 Documents	Precision at 30 documents	Uninterpolated mean average precision
Okapi TF	0.3880	0.3200	0.2227
TF-IDF	0.3840	0.3553	**0.2845**
Okapi BM25	0.3120	0.2612	0.2192
Unigram LM Laplace	0.4320	0.3267	**0.2240**
Unigram LM Jelinek Mercer	0.1320	0.1160	0.0781
ES Built-in	0.3240	0.2627	0.2124

Pseudo Relevance Feedback

We retrieve the top ‘k’ documents for each query from a results file. We will then find the significant terms (words with high score, high IDF, etc.). Then we will append those words to the original query to get more relevant documents for that query.

This relevance feedback should improve the results.