CE706 - Information Retrieval 2021

Assigment 1

2003258

Instructions for running your system (Engineering a Complete System)

Project is done in python using Jupyter notebook.

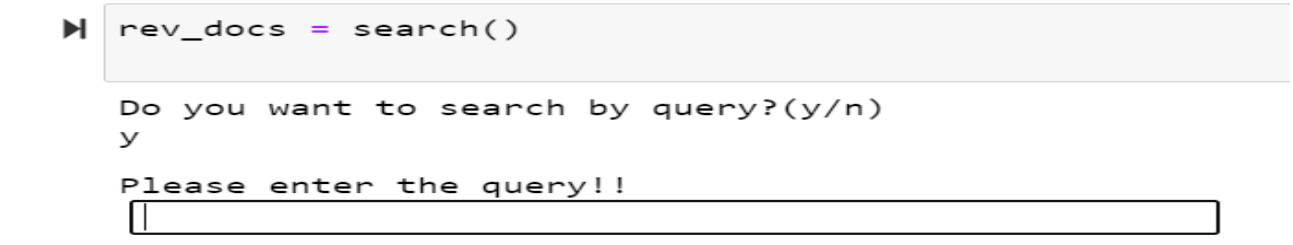
Project file is broadly divided into 5 sections:

I. Load data
II. Pre-Processing data
III. Indexing to Elastic search
IV. Retrieval models
V. Ask me

First 4 sections should be executed once before running “Ask me” which invokes user interface to accept queries from user.
Consecutively, searches can be done by just running the “Ask me” section (no need to run from the beginning of the Jupyter notebook file.

Follow the below instructions to perform a query search:
On running “Ask Me” section, it prompts “Do you want to search by query? (y/n) "
Please select "y", if you want to search by query or “n" to just filter documents by fields.

On selecting "y", you will be provided text box to enter query as shown below



And additionally, an option to filter documents by field is given, please select
1 for "None",
2 for "Published date",
3 for "Authors" and
4 for "Journal"

```
rev_docs = search()
```

```
Do you want to search by query?(y/n)
y
Please enter the query!!
Transfusing to normal haemoglobin levels

Want to add an additional filter? Please select any one of the option
1)None 2)publish date 3)Authors 4)Journal 1
```

On selecting "n", an option to select a field is provided. Please enter
1 for date range
2 for "Authors" and
3 for "Journal"

```
rev_docs = search()
```

```
Do you want to search by query?(y/n)
n


please select the filter 1)publish date 2)Authors 3)Journal
```

Indexing

Downloaded the *COVID-19 Open Research Dataset (CORD-19)* data set from Kaggle and metadata.csv
Which Has metadata for over 181000 scholarly articles is used as a source for search engine corpus.

Due to computational constraints, only first 1000 documents are considered for processing, and relevant fields like 'cord_uid','source_x','title','abstract','publish_time','authors','journal','url','license' are used for indexing.

Some of the values in the data are missing, and ,for simplicity , those rows are just dropped as a part of data imputation.

```
metadata = pd.read_csv('C:\\elastic_stack\\archive\\metadata.csv')
data = metadata[0:1000]
data = data[['cord_uid','source_x','title','abstract','publish_time','authors','journal','url','license']]
data = data.dropna()
```

Sentence Splitting, Tokenization and Normalization

Pre-processing majorly include tokenization and stop word removal and identification of sentences Most of the pre-processing has done by using python NLTK library .
On tokenizing raw data , it is seen punctuation , numbers and whitespace are also returned as tokens , so before tokenization ,those are removed from the raw data and another problem faced while tokenizing is “stopword” dataset needed to downloaded separately into the system.

```
import nltk
nltk.download('stopwords')
```

```
#lowercase
text = text.lower()

#remove numbers
text = text.translate(str.maketrans('', '', string.digits))
#remove punctauation
text= text.translate(str.maketrans("","", string.punctuation))

#white space removal
text = text.strip()
text = re.sub(r'[^a-zA-z.,!?/:;\"\'\s]', '', text)
#Tokenization
```

Selecting Keywords

Scikit-learn feature extraction module is used to extract the most informative terms in the corpus.
IDF scores of all the terms are calculated and terms with highest rank are selected which are used to get TF-IDF vectors of all the documents.

```
#Considers 2040 most informative terms based on IDF scores and buids TF-IDF vecotor for every document
#and fitted tfidf_vectorizer can be used in future to transform user query
tfidf_vectorizer=TfidfVectorizer(max_df=0.4,use_idf=True)
fitted_vectorizer=tfidf_vectorizer.fit(docs_1)
tfidf_vectorizer_vectors=fitted_vectorizer.transform(docs_1)

#tfidf_vector column contains TF-IDF vector of corresponding documents
#data['tfidf_vector'] = list(tfidf_vectorizer_vectors.toarray())
data['tfidf_vector'] = build_tfidf_vector(list(tfidf_vectorizer_vectors.toarray()))
data1 = data.to_dict("records")
```

Indexed documents have a field “tfidf_vector” which has its TF-IDF score . this fields can be leveraged to make elastic search complete back-end search engine instead a just database while implementing vector space model with cosine similarity metric. Implementation is shown below .
As can be seen above , max_df is used to remove terms which appear more than 40% of documents ,to avoid more frequent words specific to corpus
Here variable “Vec” in screen shot is TF-IDF vector of user query and “tfidf_vector” is a field that has TF-IDF vector Of corpus document.

```
def get_relevent_documents(vec,value,field):
    try:
        req_get_all = {
           "size":10,
           "query":{
              "script_score":{
                 "query":{
                   "wildcard":{
                       field:
                       {
                        "value": "*"+value+"*",
                        "case_insensitive":True
                       }
                   }
                },
                "script":{
                   "source":"cosineSimilaritySparse(params.query_vector, 'tfidf_vector') + 1.0",
                   "params":{
                       "query_vector":vec
                       }
                  }
               }
            }
        relevent_docs = es.search(index=indexName, body=req_get_all)['hits']['hits']
```

This can be achieved by using elastic search “script_score” with cosineSimilaritysparse capability require TF_IDF vector of type “sparse-vector”.

Stemming or Morphological Analysis

Stemming and lemmatization of the tokens are done using nltk porterstemmer and wordnetlemmatizer respectively.

Lemmatization ran into an error as it requires nltk “wordnet” dataset, which is solved by downloading it to a local system as shown below

```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
```

Another problem faced while stemming is, porterstemmer unable to stem the entire list of tokens but need each token a time to transform it to stem form. And some non-English root words can be seen during stemming.

On researching on that a bit, I have learnt porterstemmer library does not follow linguistic rules but a different set of stemming rules. But overall, the performance is reasonably good.

As can be seen in the below screenshot, some tokens like ‘troubl’ are not known root words in English

```
stemmer = PorterStemmer()
text_stemmed  = [stemmer.stem(word) for word in ["texting","troubling","cars","mouse"]]
print(text_stemmed)
```

```
['text', 'troubl', 'car', 'mous']
```

Searching

An intercative standard input is designed to accept user query various ways (Please refer instructions section)