Data mining Project

BY :
Ahmed Kahla 202202231

First of all, I started with importing the important libraries to be used during the project

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
import re
pd.set_option('display.max_colwidth', 150)
!pip install ir_datasets
import ir_datasets
!pip install python-terrier
import pyterrier as pt
if not pt.started():
  pt.init(boot_packages=["com.github.terrierteam:terrier-prf:-SNAPSHOT"])
```

Here's the link of the dataset used in the project:
https://ir-datasets.com/beir.html#beir/arguana

I started with reading the documents, queries, and the qrels of the dataset. Then I moved to the preprocessing step which involved removing the stopwords, steeming, tokenization, and lowercase.

Indexing :

The results of the indexing step are to get the collection statistics as shown below :
Number of documents: 8674
Number of terms: 22473
Number of postings: 559226
Number of tokens: 750403
The screenshot below shows the code of the indexing step

```
indexer = pt.DFIndexer("./myFirstIndex", overwrite=True)
index_ref = indexer.index(docs_df["processed_text"], docs_df["docno"])
```

```
19:34:45.443 [main] WARN org.terrier.structures.indexing.Indexer - Adding an empty document to
19:34:46.800 [main] WARN org.terrier.structures.indexing.Indexer - Indexed 1 empty documents
```

```
print(index_ref.toString())
index = pt.IndexFactory.of(index_ref)
print(index.getCollectionStatistics().toString())
```

```
./myFirstIndex/data.properties
Number of documents: 8674
Number of terms: 22473
Number of postings: 559226
Number of fields: 0
Number of tokens: 750403
Field names: []
Positions:   false
```

Query Processing:

I used tf_idf to apply a simple query on my dataset “ahmed” was the query. I applied the query preprocess before applying the model to the query, the screenshot below shows how was the results of the query

```
tfidf_retr = pt.BatchRetrieve(index, controls = {"wmodel": "TF_IDF"},num_results=10)
query="ahmed"
query = preprocess(query)
results=tfidf_retr.search(query)
if len(results)==0:
   print("There are no relevant documents for your selected query.")
else:
   print(results)
```

	qid	docid	docno	rank	score	query
0	1	2500	test-religion-msgfhwbamec-pro01a	0	7.566685	ahm
1	1	953	test-international-aglhrilhb-con02b	1	6.179755	ahm
2	1	6557	training-politics-oamepdgtwh-pro02a	2	5.431404	ahm
3	1	5249	training-international-aptwhbfri-pro02a	3	4.883798	ahm
4	1	3018	training-health-ssiahrgmhwc-con02a	4	4.806266	ahm
5	1	4759	training-international-meptwhbi22-con01b	5	4.205402	ahm

Query expansion:

First was to declare the bert model and display the full text on the notebook without truncation as shown in this code

```
pd.set_option('display.max_colwidth', 150)
from transformers import AutoTokenizer, AutoModel
model_name = "bert-base-uncased"
bert_tokenizer = AutoTokenizer.from_pretrained(model_name)
bert_model = AutoModel.from_pretrained(model_name)
```

Then i declared the model itself as shown in this line of code

```
model = AutoModel.load_from_hf_hub("xpmir/monot5", as_instance=True)
```

In this code

```
result_merged = results.merge(docs_df, on="docno")[["score",
"processed_text"]]
result_merged = result_merged.sort_values(by="score", ascending=False)
result_merged
output = model.rsv(query, result_merged["processed_text"].values)
data =[(list(obj.document.items.values())[0].text,obj.score) for obj in
output]
reviews_result_v2 = pd.DataFrame(data, columns=['document',
"score"]).sort_values(by="score", ascending=False)
```

The code merges search results with document data, selects only relevant columns, and sorts them by score. Then, it applies a model's function (model.rsv) to compute relevance scores for each document based on a given query and the processed text. Finally, it generates a dataframe containing documents and their corresponding scores, sorted by relevance.

Then begins by setting up a relevance feedback (RM3) model for query expansion, which expands the initial query using retrieved documents. It then retrieves relevant documents using a TF-IDF retrieval model and checks if any results are returned. If results exist, it merges them with a document dataframe based on document numbers, sorts them by relevance score, and computes scores using a specified model. Finally, it presents the documents along with their computed scores in a dataframe named reviews_result_v2 as shown in the code below

```
rm3_expander = pt.rewrite.RM3(index,fb_terms=10, fb_docs=100)
rm3_qe = tfidf_retr >> rm3_expander
expanded_query = rm3_qe.search(query).iloc[0]["query"]
tfidf_retr = pt.BatchRetrieve(index, controls = {"wmodel":
"TF_IDF"},num_results=10)
query=expanded_query
results=tfidf_retr.search(query)
if len(results)==0:
   print("There are no relevant documents for your selected query.")
```

```
else:
   print(results)
result_merged = results.merge(docs_df, on="docno")[["score",
"processed_text"]]
result_merged = result_merged.sort_values(by="score", ascending=False)
result_merged
output = model.rsv(query, result_merged["processed_text"].values)
data =[(list(obj.document.items.values())[0].text,obj.score) for obj in
output]
reviews_result_v2 = pd.DataFrame(data, columns=['document',
"score"]).sort_values(by="score", ascending=False)
```

Evaluation:

Then the final step is to evaluate the model using this line of code

```
nltk.download('wordnet')
qrels_df = qrels_df.rename(columns={'query_id':'qid'})
qrels_df = qrels_df.rename(columns={'doc_id':'docno'})
queries_df = queries_df.rename(columns={'text':'query'})
queries_df = queries_df.rename(columns={'query_id':'qid'})
queries_df['query'] = queries_df['query'].apply(preprocess)
res = tfidf_retr.transform(queries_df)
eval = pt.Evaluate(res,qrels_df)
```

And this is the results of the evaluation step

{'map': 0.22684611303032304, 'ndcg': 0.34234808901474206}