Assignment #2

Homework Submission Guidelines

1. **Due date: 26.12.24**
2. The assignment can be done in pairs
3. Answers can be submitted either in English or Hebrew
4. HW submission should be done via moodle in the corresponding area (by **only** one of the students)
5. Late submission penalty (**5% a day**) for submitting after the assignment's due date
6. Questions / clarifications and more in the dedicated discussion sub-forum in Piazza

Dry part (70%)

Vector space model (10%):

The following matrix represents the word frequencies of four documents d1, d2, d3, d4. Columns represent the documents in the above order; rows represent the vocabulary of six indexed terms a,b,c,d,e,f in that order. (Use **ln**.)

	d1	d2	d3	d4
a	0	1	1	1
b	1	2	0	1
c	2	0	0	0
d	0	0	0	0
e	1	0	1	1
f	7	5	7	2

Assume that the fraction of corpus documents in which each term appears is 10%, 10%, 20%, 5%, 50%, 90% for the terms a, b, c, d, e, and f, respectively.

1. Compute the cosine similarity between d1 and d2 where terms are represented by the tf-idf scheme. (Describe the tf-idf scheme you have used and provide details of the computation. Use **raw tf**.) (5%)

2. Rank the documents in response to the query "a b f". Use the vector space model where document terms are represented by tf and query terms by tf-idf. Provide details of your computations. (Use **raw tf**.) (5%)

Term Weighting (10%):

1. What causes the short-documents bias effect when using cosine similarity? (5%)

2. Given following weighted tf function:

$$W_{t,d} = \alpha + (1-\alpha)\frac{tf_{t,d}}{tf_{\max(d)}}$$

Where α is a value between 0 and 1; $tf_{\max(d)}$ is the raw tf of the most frequent term in the document d.

State **one** reason why this weighted tf function is useful and **one** issue that might arise from it. (5%)

Relevance feedback and evaluation (15%)

1. User 'A' submitted a query to a search engine and obtained an ordered result list. Then, the user provided feedback to the engine (4 – the document is highly relevant to the information need expressed by the query, 0 – the document is not relevant)

DocID	Relevance
5	4
2	1
1	1
3	3
4	0

The total number of relevant documents in the collection is 10.
Calculate the **AP**, **precision** and **recall** (at rank 5) (5%)

2. Suggest a version of Rocchio's model that utilizes graded relevance judgments. (5%)
3. Suggest a version of Rocchio's model that utilizes the rank of relevant documents in the list. (5%)

Evaluation (25%)

1. Explain the problem of using NDCG for evaluation when negative labels (negative relevance judgments) for documents are used (10%)
2. In what cases evaluation using MAP will yield the same results as evaluation using MRR? Mention at least 4 different cases (5%)
3. Name two different examples where:
    a. The removal of stopwords reduces the recall. (2%)
    b. The removal of stopwords reduces precision. (2%)
4. Given a query for which there are 3 relevant documents in the collection. The R-precision of the search engine for this query is $\frac{2}{3}$. Answer the following bullets based on this information.

4.1 What is the F1 score at rank 3 for this query? (3%)

4.2 What are the maximum and minimum **possible** AP (average precision) values at rank 3 for this query? (You should provide possible values and not general bounds.) (3%)

True/False questions (10%) :

Mark each of the following sentences as true or false and give a short **(but full)** explanation for why your answer is correct:

1. p@k is a monotonic non-decreasing function with respect to k. (2%)
2. Vector space-based retrieval is always more effective than Boolean retrieval. (1%)
3. In the vector space model, the higher the value of the normalization factor for a document is, the lower are the chances of retrieval for that document. (2%)
4. The stemming process increases the number of unique terms in the index (2%)
5. Values of beta>1 in F-measure emphasize precision. (1%)
6. In Rocchio's model, q_0 might be closer to the centroid of the relevant documents than q_m. (2%)

Wet part – Intro to Pyserini (30%)

Part A: (Assignment_2/data/WET_PART_A)

1. The documents to be indexed for Part A are located in the file **docs.txt**
2. Create a pyserini index using the following parameters:
   (change the boldfaced text accordingly)

   python -m pyserini.index.lucene --collection TrecCollection --input **The path to the WET_PART_A directory** --index ***path_to_write_the_created_index*** --keepStopwords --stemmer none --storePositions --storeDocvectors –storeRaw --optimize

   Do the following to check if the index is created correctly:

   Run the following code:

```
from pyserini.index.lucene import IndexReader
part_a_index = IndexReader('Path to your created index')
print(part_a_index.stats())
```

Desired output:

```
{'total_terms': 212, 'documents': 4, 'non_empty_documents': 4, 'unique_terms': 140}
```

Run retrieval using BM25 method using the following code:

```
# perform retrieval
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.search import LuceneSearcher
# Initialize the searcher
searcher = LuceneSearcher ('Path to your created index')
# specify custom analyzer for the query processing step to match the
way the index was built
analyzer = get_lucene_analyzer(stemmer='none', stopwords=False)  #
Ensure no stopwords are removed from the query
searcher.set_analyzer(analyzer)
# Set BM25 parameters
searcher.set_bm25(k1=0.9, b=0.4)
# perform a search
query = 'corporation'
hits = searcher.search(query, k =4)

#Display results
for i in range(len(hits)):
    print(f"Doc {hits[i].docid}, Score: {hits[i].score}")
```

1. Run a query "corporation" over the collection using the above parameter file
    a. How many documents did you retrieve?
    b. How many documents did you expect to retrieve? Perform and explain the change that is needed for getting the additional documents. (Examine the text of documents.)
2. Write a query that will return document D1 first; **use up to 2 words**; explain your choice.
3. By running the query: " Michael Jackson" you will retrieve document D4.
    a. Do you think D4 is relevant to the information need expressed by this query? Explain.
    b. Type a query for which D4 can be marked as relevant document; **use up to 2 words**; explain (refer to the ranking score assigned to D4 in response of the two queries)

Part B:

1. The files for PartB are located in **Assignment_2/data/WET_PART_B/**
2. In the PartB folder you will find the following files and directories:
    a. "AP_Coll.tgz" compress file contains AP documents ("database")
        i. Untar/Unzip the file before indexing
    b. "queries.txt" – query file with 150 queries (stopwords from the query terms were removed in advanced)
    c. "qrels_AP" file – the AP relevance judgments
    d. "StopWords.xml" – the INQUERY 418 stopwords list

3. Build 2 indexes using the given "database" directory
    a. Index 1: **With** stopwords removal and **with** stemming (use “Krovetz” stemmer)
    b. Index 2: **With** stopwords removal and **without** stemming.

Note: use the parameter **--stopwords “Path to stopwords.txt file”** to remove the terms in the file from the index.

4. Run retrieval over the 2 created indexes using the following code:

Load queries:

```
# Load TSV-format topics
topic_file_path =
r'C:\IDC\Semesters\Winter24\HWs\HW2\WET_PART_B\queries.txt'
topics
=get_topics_with_reader('io.anserini.search.topicreader.TsvIntTopicRea
der',topic_file_path)
#fix query ids
queries = {}
for topic_id, topic in topics.items():
    fixed_topic_id = str(topic_id)
    if len(fixed_topic_id) == 2:
        fixed_topic_id = '0'+str(topic_id)
    queries[fixed_topic_id] = topic['title']
assert len(queries) == 150, 'missing queries'
```

For the stemmed index use:

```
# Initialize the searcher with the path to your stemmed index
index_path = 'Path to your created index'
searcher = LuceneSearcher(index_path)
# specify custom analyzer for the query processing step to match the
way the index was built
analyzer = get_lucene_analyzer(stemmer='krovetz', stopwords=False)  #
Ensure no stopwords are removed from the query
searcher.set_analyzer(analyzer)
# Optionally, configure BM25 parameters (can be adjusted as needed)
searcher.set_bm25(k1=0.9, b=0.4)

#Loop through each query in the topics dictionary and retrieve
documents:
results = {}  # To store results for each query
for topic_id, topic in queries.items():
    hits = searcher.search(topic,k=1000) # k=1000 is the number of
retrieved documents
    # Store results in TREC format for each topic
    results[topic_id] = [(hit.docid, i+1, hit.score) for i, hit in
enumerate(hits)]


# Now you can save the results to a file in the TREC format:
output_file = 'Path to write results + file name'
sorted_results = dict(sorted(results.items()))
```

```
with open(output_file, 'w') as f:
    for topic_id, hits in sorted_results.items():
        for rank, (docid, _, score) in enumerate(hits, start=1):
            f.write(f"{topic_id} Q0 {docid} {rank} {score:.4f}
pyserini\n")
```

For the un-stemmed index use:

change the index path to the created unstemmed index. In addition change the analyzer to the following:

```
analyzer = get_lucene_analyzer(stemmer='none', stopwords=False)  #
Ensure no stopwords are removed from the query
```

5. Use the trec_eval application or any other evaluation toolkit (details can be found in the lecture of week 2) to evaluate the effectiveness of the 2 retrieved lists and complete the following table. Which retrieval result obtained the highest MAP value? Explain.

Stopword Removal	Krovetz Stemmer	MAP	P@5	P@10
With	**With**			
With	**Without**			

Submission Instructions:

1. A **PDF** file containing all answers to the questions (Dry and Wet parts).
2. The name of the file as follows:
   HW2_Student_1_EMAIL_Student_2_EMAIL.pdf