INFS7410 Project - Part 1

version 1.3

Assignment Preamble

Due Date

30 August 2024, 16:00 Eastern Australia Standard Time

Weight

This assignment (Project - Part 1) constitutes **20%** of the overall mark for INFS7410.
(Part 1 + Part 2 = 40% of the total course grade)

Completion Requirements

- You should complete it **individually**.

- You can check the detailed **marking sheet** provided alongside this notebook.
    - see `INFS7410-project-part-1-marking-sheet.pdf`

Prerequisites Checker

You should have already tackled the activities from pracs week1-5, including:

- *Indexing* corpus (`prac-week1`)
- Implementing *retrieval functions* (`prac-week3`)
- Implementing *rank fusion methods* (`prac-week4`)
- Implementing *query expansion and reduction* from *pseudo relevance feedback*(`prac-week5`)
- Performing *evaluation*, *visualisation*, and *statistical significance test* (`prac-week2`)

Tips

- Start early to allow ample time for completion.
- Proceed step-by-step through the assignment tasks.
- Most of the assignment relies on knowledge and code from your computer practicals. However, be prepared for challenges that may require additional time and effort to solve.

Aims

Project Aim:

The aim of the entire project is to implement a number of representiative information retrieval methods, evaluate and compare them in the context of real use-cases.

Part 1 Aim

The aim of Part 1 is to:

- Familiarise yourself with the basic retrieval workflow.

- Set up the infrastructure for indexing the corpus and evaluating with queries.
- Implement classical information retrieval methods coverd in the pracs and lectures.
- Tune your retrieval methods to improve their effectiveness.

The Information Retrieval Tasks: Fact Checking and Bio-Medical Retrieval

In this project, we will consider two tasks in IR:

- **Fact Checking** verifies a claim against a large collection of evidence. Here, we focus on the *scientific* domain, which ranges from *basic science* to *clinical medicine*. We verify scientific claims by retrieving evidence from a corpus of research literature containing scientific paper abstracts.
- **Bio-Medical Retrieval** involves searching for relevant scientific documents, such as research papers or blogs, in response to a specific query within the biomedical domain.

For these tasks, we will use selected datasets from the BEIR benchmark, specifically **SciFact** (Fact checking), **NFCorpus** (Biomedical),and **TREC-COVID** (Biomedical).

What we give you:

Files from Previous Practicals

You can freely re-use all the materials from `prac-week1` to `prac-week5`, e.g. your implemenation/codes.

Files for This Project

- `infs7410_project_collections.zip` (74.1 MB)
  - Click here to **download** and **unzip**.
- `INFS7410-project-part-1.ipynb` (This notebook)
- `INFS7410-project-part-1-marking-sheet.pdf`

We provide the following collections for the project:

- **NF Corpus**: (*training + test*)
  - `./nfcorpus_corpus.jsonl`
  - `./nfcorpus/nfcorpus_train_queries.tsv`
  - `./nfcorpus/nfcorpus_train_queries.tsv`
  - `./nfcorpus/nfcorpus_train_qrels.txt`
  - `./nfcorpus/nfcorpus_test_qrels.txt`
- **SciFact**: (*training + test*)
  - `./scifact_corpus.jsonl`
  - `./scifact/scifact_train_queries.tsv`
  - `./scifact/scifact_train_queries.tsv`
  - `./scifact/scifact_train_qrels.txt`
  - `./scifact/scifact_train_qrels.txt`
- **TREC-COVID**: (*test only*)

- `./trec-covid_corpus.jsonl`
- `./trec-covid/trec-covid_test_qrels.txt`
- `./trec-covid/trec-covid_test_queries.tsv`

Generally, each collection contains:

- `corpus.jsonl`: Containing the texts to be retrieved for each query.
- `queries.tsv`: Listing queries used for retrieval, each line containing a topic id and the query text.
- `qrels.txt`: Containing relevance judgements for your runs in TREC format <qid, Q0, doc_id, rank, score, tag>.

Additionally,

This Jupyter notebook is the workspace where you will implement your solutions and document your findings.

Put this notebook and the provided files under the same directory.

Overview of the IR Workflow

To conduct an experiment for the IR tasks in this project, we generally follow three key stages: `Indexing -> Retrieval -> Evaluation`. Each stage involves a corresponding portion of data from the collection. A collection typically comprises a corpus, queries, and a qrel. These are illustrated below:

What you need to do:

You are expected to deliver the following:

- **Correct implementations** and **evaluations** of **the methods** required by this project specifications.
- **Write-up** about the **retrieval methods** used, including:
  - **formula** that represent the method you implemented.
  - **code** that corresponds to the formula.
  - **evaluation settings** with explanation followed.
  - **discussion** of the findings.

Both the implementations and write-ups must be documented within this Jupyter notebook.

Required Methods to Implement

- Indexing
  - **Pyserini index command**: You first need to index the new datasets introduced in this project. Each dataset should be made into a separate index. Check how we use the command to build index for given collections from `week1-week3`.
- Ranking functions
  - **BM25**: Implemented by yourself, not using the one from Pyserini. Check `week3`.
- Query reformulation methods

  - **Pseudo-Relevance Feedback using BM25 for Query Expansion**: Implemented by yourself. Check `week5`.
  - **IDF-r Query Reduction**: Implemented by yourself. Check `week5`.
- Rank fusion methods
  - **Borda**: Implemented by yourself. Check `week4`.
  - **CombSUM**: Implemented by yourself. Check `week4`.
  - **CombMNZ**: Implemented by yourself. Check `week4`.

Parameter Tuning:

N.B. ONLY TUNE WITH TRAINING QUERIES.

- Tune the parameters in your **BM25**, **Query Expansion**, and **Query Reduction** implementations. Conduct a parameter search over **at least 5 carefully selected values** for each method, and **5 pairs** when the method involves **two parameters**.
- For **Rank fusion methods**, focus on fusing **the highest-performing tuned run** from each of the BM25, Query Expansion, and Query Reduction implementations.

Required Evaluations to Perform

In this project, we provide three datasets with sampled queries the original BEIR version. You first task is to check these datasets and get familiar with the size, content, format, and consider how to process them. Pay particular attention to the differences between these datasets and the MSMARCO collection we used in the pracs.

In Part 1 of the project, you are required to perform the following evaluations:

1. Run your **BM25** with `k=1.2, b=0.75` with the `test_queries` of `SciFact` , `NFCorpus`, and `TREC-COVID` as the **baselines**.
2. **Tune the parameters** for **BM25**, **Query Expansion**, and **Query Reduction** with the `train_queries` of `SciFact` and `NFCorpus`. Refer to the ***Parameter Tuning*** section outlined above.
3. **Report the results** of each method from tuning in a table. Perform **statistical significance analysis** across the results of the methods and report them in the table (e.g., comparing Method_A with para-setting_a on dataset_1 with baseline on the same dataset).
4. **Select the best parameter setting** of the methods from `Sci-Fact` and `NFCorpus` seperately, and run with `test_queries` of `TREC-COVID`. **Report the results** in the table, follow the same requirements listed above.
5. Create a **gain-loss plot** comparing **BM25vs. Pseudo-Relevance Feedback Query Expansion using BM25**, as well as plots comparing **BM25 vs. each rank fusion method** on `TREC-COVID`. Using the **baseline BM25** and the two best parameter settings from `SciFact` and `NFCorpus` for `Query Expansion`.

Discussions

1. Comment on **trends and differences** observed in your results. Do the methods work well on `SciFact` and `NFCorpus` also **generalise** to `TREC-COVID`? Is there a method that **consistently outperforms** others on all the datasets?
2. Provide insights into whether **rank fusion** works or not, considering factors like runs included in the fusion process and query characteristics.
3. Discuss the results obtained from `SciFact` and `NFCorpus` compared to those on `TREC-COVID`. Do ranking methods perform well? Why or why not? Include the **statistical significance analysis** results comparing the baseline with the tuned methods, and each method against another (including fusion, query expansion and reduction).

Evaluation Measures

Evaluate the retrieval methods using the following measures:

- **nDCG at 10** (`ndcg_cut_10`): Use this as the primary measure for tuning

All **gain-loss plots** should be produced with respect to **nDCG at 10**.

For **statistical significance analysis**, use the **paired t-test** and distinguish between **p < 0.05** and **p < 0.01**.

How to submit

Submit **one** `.zip` file containing:

1. this notebook in `.ipynb` format
2. this notebook saved as a `.pdf` by navigating to the menu and selecting `File -> Save and Export Notebook As -> HTML`, then open and `Print` it with your browser.

N.B.

- Ensure that the code is executable. Include all your discussions and analysis within this notebook, not as a separate file.

- Don't add any runs, indexes in the zip file! (If you do this, the file will be too big and you will encounter errors in submitting.)

- Submit the file via the link on the INFS7410 course site on BlackBoard by **30 August 2024, 16:00 Eastern Australia Standard Time**, unless you have received an extension according to UQ policy, which must be requested **before** the assignment due date.

## Check datasets and perform indexing

Note: Try out different indexes to find the most effective setup for your needs.

First, have a look at the datasets. Then, try to use the indexing command from pracs. Remember to add `-storeDocvectors`.

You may want to store your indexes under `./indexes/`.

You can check the pyserini guidance.

```
# Run your commands for indexing the corpus of each datasets
# Examining the TREC-COVID dataset
import json

with open('./infs7410_project_collections/trec-covid/trec-
covid_corpus.jsonl', 'r', encoding='utf-8') as f:
    for i, line in enumerate(f):
        if i < 5:  # Print first 5 documents
            doc = json.loads(line)
            print(f"Document ID: {doc['id']}")
            print(f"Contents: {doc['contents'][:100]}...")  # First
100 characters of contents
            print("---")
        else:
            break

# Count total documents
with open('./infs7410_project_collections/trec-covid/trec-
covid_corpus.jsonl', 'r', encoding='utf-8') as f:
    doc_count = sum(1 for _ in f)
print(f"Total documents in TREC-COVID: {doc_count}")
```

```
Document ID: ug7v899j
Contents: Clinical features of culture-proven Mycoplasma pneumoniae
infections at King Abdulaziz University Ho...
---
Document ID: 02tnwd4m
Contents: Nitric oxide: a pro-inflammatory mediator in lung disease?
Inflammatory diseases of the respiratory ...
---
Document ID: ejv2xln0
Contents: Surfactant protein-D and pulmonary host defense Surfactant
protein-D (SP-D) participates in the inna...
---
Document ID: 2b73a28n
Contents: Role of endothelin-1 in lung disease Endothelin-1 (ET-1) is
a 21 amino acid peptide with diverse bio...
---
Document ID: 9785vg6d
Contents: Gene expression in epithelial cells in response to
pneumovirus infection Respiratory syncytial virus...
---
Total documents in TREC-COVID: 171332
```

```
# Examining the nfcorpus dataset
import json
```

```
with
open('./infs7410_project_collections/nfcorpus/nfcorpus_corpus.jsonl',
'r', encoding='utf-8') as f:
    for i, line in enumerate(f):
        if i < 5:  # Print first 5 documents
            doc = json.loads(line)
            print(f"Document ID: {doc['id']}")
            print(f"Contents: {doc['contents'][:100]}...")  # First
100 characters of contents
            print("---")
        else:
            break

# Count total documents
with
open('./infs7410_project_collections/nfcorpus/nfcorpus_corpus.jsonl',
'r', encoding='utf-8') as f:
    doc_count = sum(1 for _ in f)
print(f"Total documents in nfcorpus: {doc_count}")
```

```
Document ID: MED-10
Contents: Statin Use and Breast Cancer Survival: A Nationwide Cohort
Study from Finland Recent studies have su...
---
Document ID: MED-14
Contents: Statin use after diagnosis of breast cancer and survival: a
population-based cohort study. BACKGROUN...
---
Document ID: MED-118
Contents: Alkylphenols in human milk and their relations to dietary
habits in central Taiwan. The aims of this...
---
Document ID: MED-301
Contents: Methylmercury: A Potential Environmental Risk Factor
Contributing to Epileptogenesis Epilepsy or sei...
---
Document ID: MED-306
Contents: Sensitivity of Continuous Performance Test (CPT) at Age 14
Years to Developmental Methylmercury Expo...
---
Total documents in nfcorpus: 3633
```

```
# Examining the scifact dataset
import json

with
open('./infs7410_project_collections/scifact/scifact_corpus.jsonl',
'r', encoding='utf-8') as f:
    for i, line in enumerate(f):
        if i < 5:  # Print first 5 documents
```

```
            doc = json.loads(line)
            print(f"Document ID: {doc['id']}")
            print(f"Contents: {doc['contents'][:100]}...")  # First
100 characters of contents
            print("---")
        else:
            break

# Count total documents
with
open('./infs7410_project_collections/scifact/scifact_corpus.jsonl',
'r', encoding='utf-8') as f:
    doc_count = sum(1 for _ in f)
print(f"Total documents in scifact: {doc_count}")
```

```
Document ID: 4983
Contents: Microstructural development of human newborn cerebral white
matter assessed in vivo by diffusion ten...
---
Document ID: 5836
Contents: Induction of myelodysplasia by myeloid-derived suppressor
cells. Myelodysplastic syndromes (MDS) are...
---
Document ID: 7912
Contents: BC1 RNA, the transcript from a master gene for ID element
amplification, is able to prime its own re...
---
Document ID: 18670
Contents: The DNA Methylome of Human Peripheral Blood Mononuclear
Cells DNA methylation plays an important rol...
---
Document ID: 19238
Contents: The human myelin basic protein gene is included within a
179-kilobase transcription unit: expression...
---
Total documents in scifact: 5183
```

2. Setup and Indexing

```
# Indexing NFCorpus
!python -m pyserini.index.lucene -collection JsonCollection -generator
DefaultLuceneDocumentGenerator \
 -threads 9 -input ./infs7410_project_collections/nfcorpus \
 -index ./indexes/nfcorpus-index \
 -optimize \
 -storeRaw \
 -stemmer none \
 -keepStopwords \
 -storeDocvectors

# Indexing SciFact
!python -m pyserini.index.lucene -collection JsonCollection -generator
DefaultLuceneDocumentGenerator \
 -threads 9 -input ./infs7410_project_collections/scifact \
 -index ./indexes/scifact-index \
 -optimize \
 -storeRaw \
 -stemmer none \
 -keepStopwords \
 -storeDocvectors

# Indexing TREC-COVID
!python -m pyserini.index.lucene -collection JsonCollection -generator
DefaultLuceneDocumentGenerator \
 -threads 9 -input ./infs7410_project_collections/trec-covid \
 -index ./indexes/trec-covid-index \
 -optimize \
 -storeRaw \
 -stemmer none \
 -keepStopwords \
 -storeDocvectors
```

```
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This
will impact performance.
2024-08-29 10:54:48,804 INFO  [main] index.IndexCollection
(IndexCollection.java:380) - Setting log level to INFO
2024-08-29 10:54:48,805 INFO  [main] index.IndexCollection
(IndexCollection.java:383) - Starting indexer...
2024-08-29 10:54:48,805 INFO  [main] index.IndexCollection
(IndexCollection.java:384) - ============ Loading Parameters
============
2024-08-29 10:54:48,805 INFO  [main] index.IndexCollection
(IndexCollection.java:385) - DocumentCollection path:
./infs7410_project_collections/nfcorpus
2024-08-29 10:54:48,805 INFO  [main] index.IndexCollection
(IndexCollection.java:386) - CollectionClass: JsonCollection
2024-08-29 10:54:48,805 INFO  [main] index.IndexCollection
(IndexCollection.java:387) - Generator: DefaultLuceneDocumentGenerator
2024-08-29 10:54:48,805 INFO  [main] index.IndexCollection
(IndexCollection.java:388) - Threads: 9
2024-08-29 10:54:48,805 INFO  [main] index.IndexCollection
(IndexCollection.java:389) - Language: en
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:390) - Stemmer: none
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:391) - Keep stopwords? true
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:392) - Stopwords: null
```

```
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:393) - Store positions? false
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:394) - Store docvectors? true
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:395) - Store document "contents" field? false
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:396) - Store document "raw" field? true
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:397) - Additional fields to index: []
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:398) - Optimize (merge segments)? true
2024-08-29 10:54:48,806 INFO  [main] index.IndexCollection
(IndexCollection.java:399) - Whitelist: null
2024-08-29 10:54:48,807 INFO  [main] index.IndexCollection
(IndexCollection.java:400) - Pretokenized?: false
2024-08-29 10:54:48,807 INFO  [main] index.IndexCollection
(IndexCollection.java:401) - Index path: ./indexes/nfcorpus-index
2024-08-29 10:54:48,809 INFO  [main] index.IndexCollection
(IndexCollection.java:481) - ============ Indexing Collection
============
2024-08-29 10:54:48,815 INFO  [main] index.IndexCollection
(IndexCollection.java:468) - Using DefaultEnglishAnalyzer
2024-08-29 10:54:48,815 INFO  [main] index.IndexCollection
(IndexCollection.java:469) - Stemmer: none
2024-08-29 10:54:48,815 INFO  [main] index.IndexCollection
(IndexCollection.java:470) - Keep stopwords? true
2024-08-29 10:54:48,815 INFO  [main] index.IndexCollection
(IndexCollection.java:471) - Stopwords file: null
2024-08-29 10:54:48,894 INFO  [main] index.IndexCollection
(IndexCollection.java:510) - Thread pool with 9 threads initialized.
2024-08-29 10:54:48,894 INFO  [main] index.IndexCollection
(IndexCollection.java:512) - Initializing collection in
./infs7410_project_collections/nfcorpus
2024-08-29 10:54:48,896 INFO  [main] index.IndexCollection
(IndexCollection.java:521) - 2 files found
2024-08-29 10:54:48,896 INFO  [main] index.IndexCollection
(IndexCollection.java:522) - Starting to index...
2024-08-29 10:54:49,677 DEBUG [pool-2-thread-2]
index.IndexCollection$LocalIndexerThread (IndexCollection.java:345)
- .ipynb_checkpoints/nfcorpus_corpus-checkpoint.jsonl: 3633 docs
added.
2024-08-29 10:54:49,677 DEBUG [pool-2-thread-1]
index.IndexCollection$LocalIndexerThread (IndexCollection.java:345) -
nfcorpus/nfcorpus_corpus.jsonl: 3633 docs added.
2024-08-29 10:54:50,117 INFO  [main] index.IndexCollection
(IndexCollection.java:578) - Indexing Complete! 7,266 documents
indexed
2024-08-29 10:54:50,117 INFO  [main] index.IndexCollection
```

```
(IndexCollection.java:579) - ============ Final Counter Values
============
2024-08-29 10:54:50,117 INFO  [main] index.IndexCollection
(IndexCollection.java:580) - indexed:            7,266
2024-08-29 10:54:50,117 INFO  [main] index.IndexCollection
(IndexCollection.java:581) - unindexable:            0
2024-08-29 10:54:50,117 INFO  [main] index.IndexCollection
(IndexCollection.java:582) - empty:                  0
2024-08-29 10:54:50,117 INFO  [main] index.IndexCollection
(IndexCollection.java:583) - skipped:                0
2024-08-29 10:54:50,118 INFO  [main] index.IndexCollection
(IndexCollection.java:584) - errors:                 0
2024-08-29 10:54:50,121 INFO  [main] index.IndexCollection
(IndexCollection.java:587) - Total 7,266 documents indexed in 00:00:01
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This
will impact performance.
2024-08-29 10:54:51,175 INFO  [main] index.IndexCollection
(IndexCollection.java:380) - Setting log level to INFO
2024-08-29 10:54:51,176 INFO  [main] index.IndexCollection
(IndexCollection.java:383) - Starting indexer...
2024-08-29 10:54:51,176 INFO  [main] index.IndexCollection
(IndexCollection.java:384) - ============ Loading Parameters
============
2024-08-29 10:54:51,176 INFO  [main] index.IndexCollection
(IndexCollection.java:385) - DocumentCollection path:
./infs7410_project_collections/scifact
2024-08-29 10:54:51,176 INFO  [main] index.IndexCollection
(IndexCollection.java:386) - CollectionClass: JsonCollection
2024-08-29 10:54:51,176 INFO  [main] index.IndexCollection
(IndexCollection.java:387) - Generator: DefaultLuceneDocumentGenerator
2024-08-29 10:54:51,177 INFO  [main] index.IndexCollection
(IndexCollection.java:388) - Threads: 9
2024-08-29 10:54:51,177 INFO  [main] index.IndexCollection
(IndexCollection.java:389) - Language: en
2024-08-29 10:54:51,177 INFO  [main] index.IndexCollection
(IndexCollection.java:390) - Stemmer: none
2024-08-29 10:54:51,177 INFO  [main] index.IndexCollection
(IndexCollection.java:391) - Keep stopwords? true
2024-08-29 10:54:51,177 INFO  [main] index.IndexCollection
(IndexCollection.java:392) - Stopwords: null
2024-08-29 10:54:51,177 INFO  [main] index.IndexCollection
(IndexCollection.java:393) - Store positions? false
2024-08-29 10:54:51,177 INFO  [main] index.IndexCollection
(IndexCollection.java:394) - Store docvectors? true
2024-08-29 10:54:51,178 INFO  [main] index.IndexCollection
(IndexCollection.java:395) - Store document "contents" field? false
2024-08-29 10:54:51,178 INFO  [main] index.IndexCollection
(IndexCollection.java:396) - Store document "raw" field? true
2024-08-29 10:54:51,178 INFO  [main] index.IndexCollection
```

```
(IndexCollection.java:397) - Additional fields to index: []
2024-08-29 10:54:51,178 INFO  [main] index.IndexCollection
(IndexCollection.java:398) - Optimize (merge segments)? true
2024-08-29 10:54:51,178 INFO  [main] index.IndexCollection
(IndexCollection.java:399) - Whitelist: null
2024-08-29 10:54:51,178 INFO  [main] index.IndexCollection
(IndexCollection.java:400) - Pretokenized?: false
2024-08-29 10:54:51,178 INFO  [main] index.IndexCollection
(IndexCollection.java:401) - Index path: ./indexes/scifact-index
2024-08-29 10:54:51,180 INFO  [main] index.IndexCollection
(IndexCollection.java:481) - ============ Indexing Collection
============
2024-08-29 10:54:51,185 INFO  [main] index.IndexCollection
(IndexCollection.java:468) - Using DefaultEnglishAnalyzer
2024-08-29 10:54:51,186 INFO  [main] index.IndexCollection
(IndexCollection.java:469) - Stemmer: none
2024-08-29 10:54:51,186 INFO  [main] index.IndexCollection
(IndexCollection.java:470) - Keep stopwords? true
2024-08-29 10:54:51,186 INFO  [main] index.IndexCollection
(IndexCollection.java:471) - Stopwords file: null
2024-08-29 10:54:51,254 INFO  [main] index.IndexCollection
(IndexCollection.java:510) - Thread pool with 9 threads initialized.
2024-08-29 10:54:51,254 INFO  [main] index.IndexCollection
(IndexCollection.java:512) - Initializing collection in
./infs7410_project_collections/scifact
2024-08-29 10:54:51,256 INFO  [main] index.IndexCollection
(IndexCollection.java:521) - 2 files found
2024-08-29 10:54:51,256 INFO  [main] index.IndexCollection
(IndexCollection.java:522) - Starting to index...
2024-08-29 10:54:52,261 DEBUG [pool-2-thread-1]
index.IndexCollection$LocalIndexerThread (IndexCollection.java:345) -
scifact/scifact_corpus.jsonl: 5183 docs added.
2024-08-29 10:54:52,261 DEBUG [pool-2-thread-2]
index.IndexCollection$LocalIndexerThread (IndexCollection.java:345)
- .ipynb_checkpoints/scifact_corpus-checkpoint.jsonl: 5183 docs added.
2024-08-29 10:54:52,917 INFO  [main] index.IndexCollection
(IndexCollection.java:578) - Indexing Complete! 10,366 documents
indexed
2024-08-29 10:54:52,917 INFO  [main] index.IndexCollection
(IndexCollection.java:579) - ============ Final Counter Values
============
2024-08-29 10:54:52,917 INFO  [main] index.IndexCollection
(IndexCollection.java:580) - indexed:           10,366
2024-08-29 10:54:52,918 INFO  [main] index.IndexCollection
(IndexCollection.java:581) - unindexable:            0
2024-08-29 10:54:52,918 INFO  [main] index.IndexCollection
(IndexCollection.java:582) - empty:                  0
2024-08-29 10:54:52,918 INFO  [main] index.IndexCollection
(IndexCollection.java:583) - skipped:                0
```

```
2024-08-29 10:54:52,918 INFO  [main] index.IndexCollection
(IndexCollection.java:584) - errors:                 0
2024-08-29 10:54:52,920 INFO  [main] index.IndexCollection
(IndexCollection.java:587) - Total 10,366 documents indexed in
00:00:01
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This
will impact performance.
2024-08-29 10:54:54,000 INFO  [main] index.IndexCollection
(IndexCollection.java:380) - Setting log level to INFO
2024-08-29 10:54:54,001 INFO  [main] index.IndexCollection
(IndexCollection.java:383) - Starting indexer...
2024-08-29 10:54:54,001 INFO  [main] index.IndexCollection
(IndexCollection.java:384) - ============ Loading Parameters
============
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:385) - DocumentCollection path:
./infs7410_project_collections/trec-covid
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:386) - CollectionClass: JsonCollection
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:387) - Generator: DefaultLuceneDocumentGenerator
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:388) - Threads: 9
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:389) - Language: en
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:390) - Stemmer: none
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:391) - Keep stopwords? true
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:392) - Stopwords: null
2024-08-29 10:54:54,002 INFO  [main] index.IndexCollection
(IndexCollection.java:393) - Store positions? false
2024-08-29 10:54:54,003 INFO  [main] index.IndexCollection
(IndexCollection.java:394) - Store docvectors? true
2024-08-29 10:54:54,003 INFO  [main] index.IndexCollection
(IndexCollection.java:395) - Store document "contents" field? false
2024-08-29 10:54:54,003 INFO  [main] index.IndexCollection
(IndexCollection.java:396) - Store document "raw" field? true
2024-08-29 10:54:54,003 INFO  [main] index.IndexCollection
(IndexCollection.java:397) - Additional fields to index: []
2024-08-29 10:54:54,003 INFO  [main] index.IndexCollection
(IndexCollection.java:398) - Optimize (merge segments)? true
2024-08-29 10:54:54,003 INFO  [main] index.IndexCollection
(IndexCollection.java:399) - Whitelist: null
2024-08-29 10:54:54,003 INFO  [main] index.IndexCollection
(IndexCollection.java:400) - Pretokenized?: false
2024-08-29 10:54:54,003 INFO  [main] index.IndexCollection
(IndexCollection.java:401) - Index path: ./indexes/trec-covid-index
```

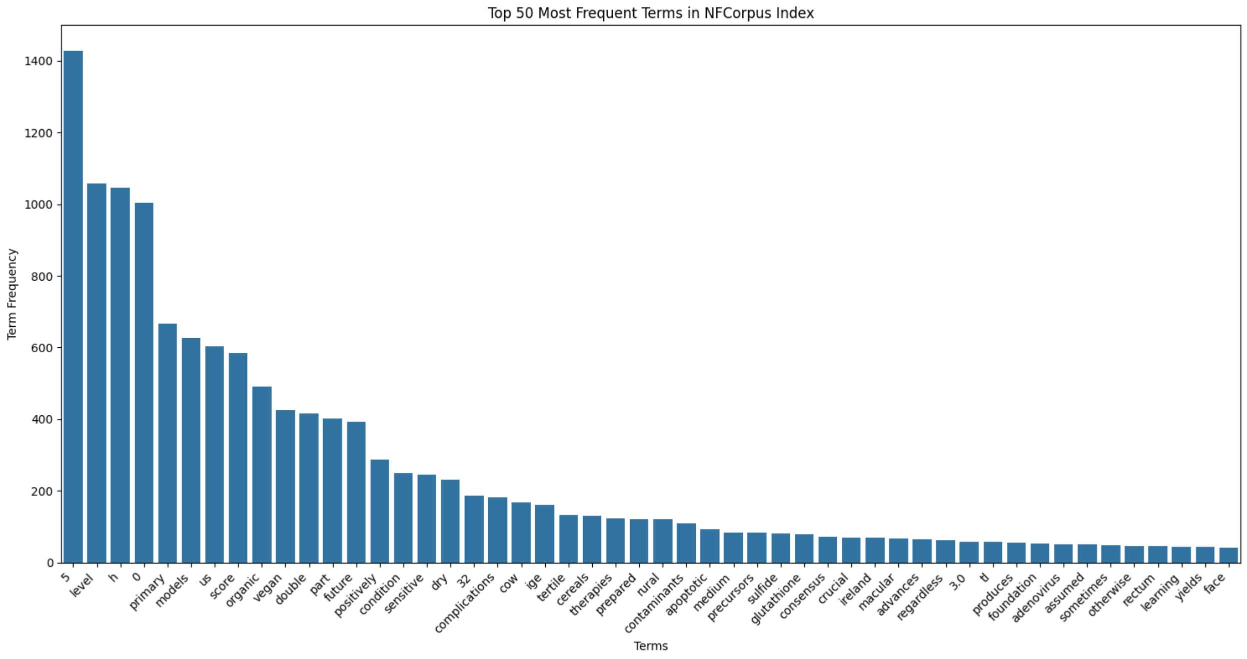
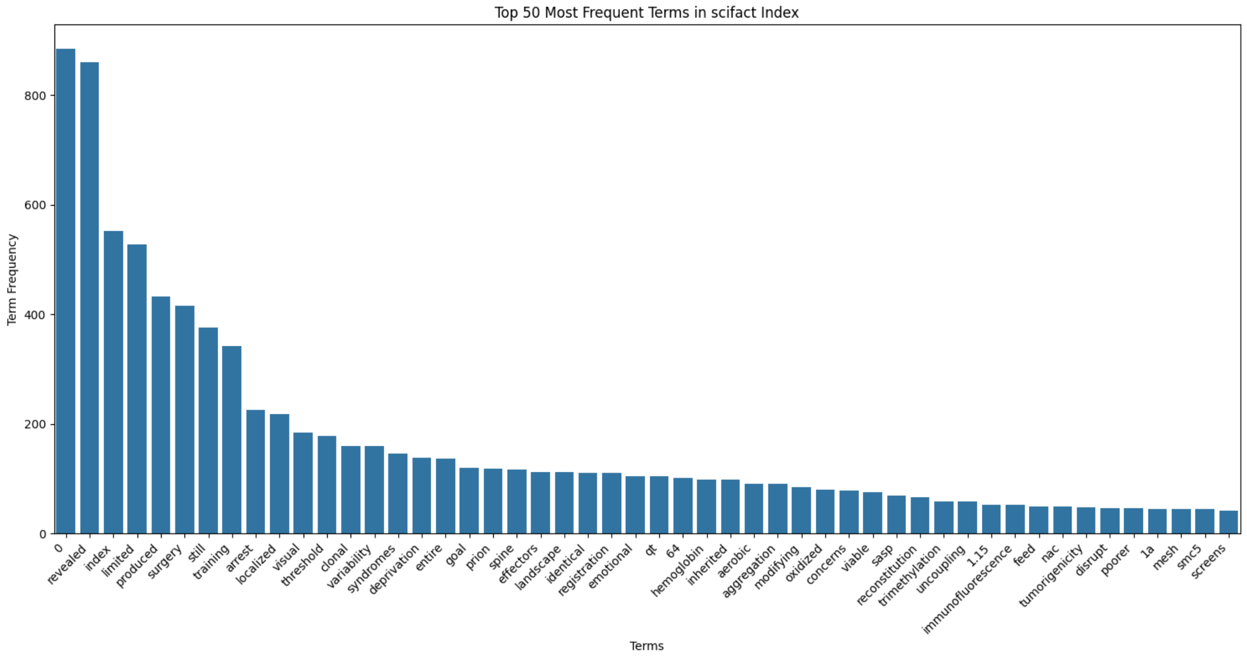
```
2024-08-29 10:54:54,005 INFO  [main] index.IndexCollection
(IndexCollection.java:481) - ============ Indexing Collection
============
2024-08-29 10:54:54,009 INFO  [main] index.IndexCollection
(IndexCollection.java:468) - Using DefaultEnglishAnalyzer
2024-08-29 10:54:54,009 INFO  [main] index.IndexCollection
(IndexCollection.java:469) - Stemmer: none
2024-08-29 10:54:54,009 INFO  [main] index.IndexCollection
(IndexCollection.java:470) - Keep stopwords? true
2024-08-29 10:54:54,009 INFO  [main] index.IndexCollection
(IndexCollection.java:471) - Stopwords file: null
2024-08-29 10:54:54,075 INFO  [main] index.IndexCollection
(IndexCollection.java:510) - Thread pool with 9 threads initialized.
2024-08-29 10:54:54,075 INFO  [main] index.IndexCollection
(IndexCollection.java:512) - Initializing collection in
./infs7410_project_collections/trec-covid
2024-08-29 10:54:54,076 INFO  [main] index.IndexCollection
(IndexCollection.java:521) - 1 file found
2024-08-29 10:54:54,077 INFO  [main] index.IndexCollection
(IndexCollection.java:522) - Starting to index...
2024-08-29 10:55:09,488 DEBUG [pool-2-thread-1]
index.IndexCollection$LocalIndexerThread (IndexCollection.java:345) -
trec-covid/trec-covid_corpus.jsonl: 171331 docs added.
2024-08-29 10:55:11,090 INFO  [main] index.IndexCollection
(IndexCollection.java:578) - Indexing Complete! 171,331 documents
indexed
2024-08-29 10:55:11,090 INFO  [main] index.IndexCollection
(IndexCollection.java:579) - ============ Final Counter Values
============
2024-08-29 10:55:11,091 INFO  [main] index.IndexCollection
(IndexCollection.java:580) - indexed:          171,331
2024-08-29 10:55:11,091 INFO  [main] index.IndexCollection
(IndexCollection.java:581) - unindexable:            0
2024-08-29 10:55:11,091 INFO  [main] index.IndexCollection
(IndexCollection.java:582) - empty:                  1
2024-08-29 10:55:11,091 INFO  [main] index.IndexCollection
(IndexCollection.java:583) - skipped:                0
2024-08-29 10:55:11,091 INFO  [main] index.IndexCollection
(IndexCollection.java:584) - errors:                 0
2024-08-29 10:55:11,094 INFO  [main] index.IndexCollection
(IndexCollection.java:587) - Total 171,331 documents indexed in
00:00:17
```

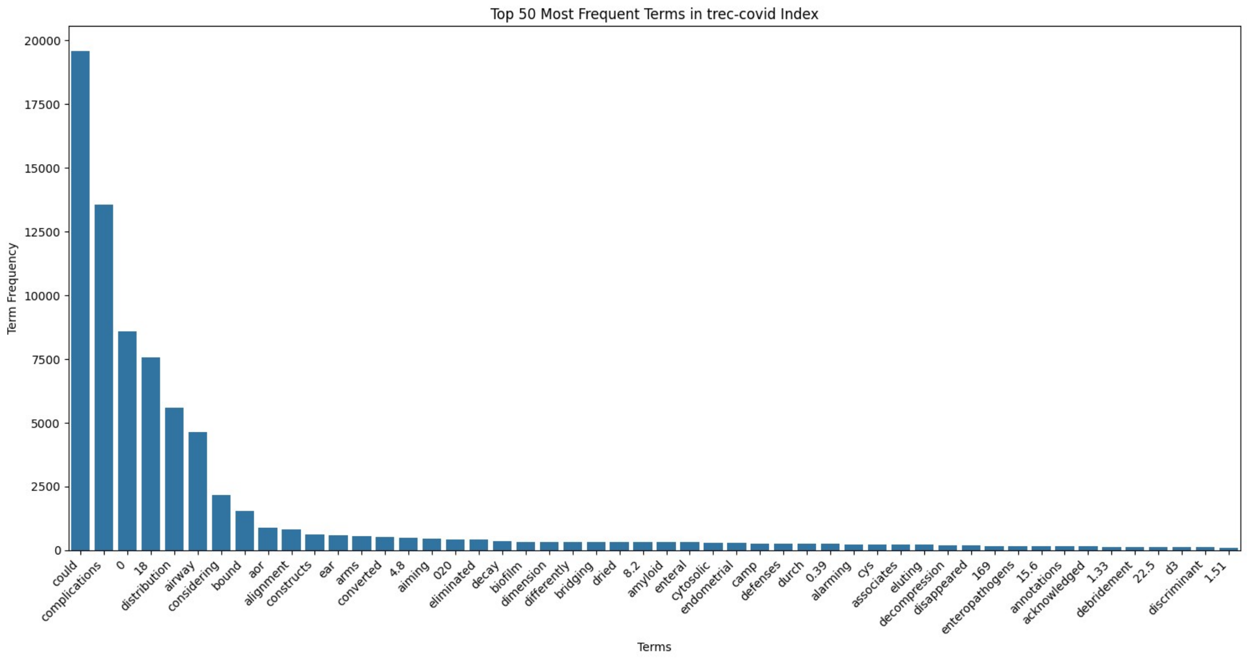
2.1 Index Statistics

```
from pyserini.index import IndexReader
import itertools
```

```python
index_reader1 = IndexReader('indexes/nfcorpus-index/')
print(index_reader1.stats())

index_reader2 = IndexReader('indexes/scifact-index/')
print(index_reader2.stats())

index_reader3 = IndexReader('indexes/trec-covid-index/')
print(index_reader3.stats())

import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter


# Collect term frequencies
term_freq = Counter()
for term in itertools.islice(index_reader1.terms(), 0, 100000, 100):
# Sample every 100th term up to 100,000
    term_freq[term.term] = term.cf

# Get the top N most frequent terms
N = 50
top_terms = dict(term_freq.most_common(N))

# Create the bar plot
plt.figure(figsize=(15, 8))
sns.barplot(x=list(top_terms.keys()), y=list(top_terms.values()))
plt.title(f'Top {N} Most Frequent Terms in NFCorpus Index')
plt.xlabel('Terms')
plt.ylabel('Term Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()


# Collect term frequencies
term_freq = Counter()
for term in itertools.islice(index_reader2.terms(), 0, 100000, 100):
# Sample every 100th term up to 100,000
    term_freq[term.term] = term.cf

# Get the top N most frequent terms
N = 50  #
top_terms = dict(term_freq.most_common(N))

# Create the bar plot
plt.figure(figsize=(15, 8))
sns.barplot(x=list(top_terms.keys()), y=list(top_terms.values()))
plt.title(f'Top {N} Most Frequent Terms in scifact Index')
plt.xlabel('Terms')
plt.ylabel('Term Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()


# Collect term frequencies
term_freq = Counter()
for term in itertools.islice(index_reader3.terms(), 0, 100000, 100):
# Sample every 100th term up to 100,000
    term_freq[term.term] = term.cf

# Get the top N most frequent terms
N = 50  #
top_terms = dict(term_freq.most_common(N))

# Create the bar plot
plt.figure(figsize=(15, 8))
sns.barplot(x=list(top_terms.keys()), y=list(top_terms.values()))
plt.title(f'Top {N} Most Frequent Terms in trec-covid Index')
plt.xlabel('Terms')
plt.ylabel('Term Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
{'total_terms': 1750458, 'documents': 7266, 'non_empty_documents':
7266, 'unique_terms': 29444}
{'total_terms': 2310514, 'documents': 10366, 'non_empty_documents':
10366, 'unique_terms': 38437}
{'total_terms': 28720002, 'documents': 171331, 'non_empty_documents':
171331, 'unique_terms': 244432}
```


Top 50 Most Frequent Terms in NFCorpus Index
Term Frequency
Terms
5
level
h
0
primary
models
us
score
organic
vegan
double
part
future
positively
condition
sensitive
dry
32
complications
cow
ige
tertile
cereals
therapies
prepared
rural
contaminants
apoptotic
medium
precursors
sulfide
glutathione
consensus
crucial
ireland
macular
advances
regardless
3.0
tl
produces
foundation
adenovirus
assumed
sometimes
otherwise
rectum
learning
yields
face

Top 50 Most Frequent Terms in scifact Index
Term Frequency
Terms
0
revealed
index
limited
produced
surgery
still
training
arrest
localized
visual
threshold
clonal
variability
syndromes
deprivation
entire
goal
prion
spine
effectors
landscape
identical
registration
emotional
qt
64
hemoglobin
inherited
aerobic
aggregation
modifying
oxidized
concerns
viable
sasp
reconstitution
trimethylation
uncoupling
1.15
immunofluorescence
feed
nac
tumorigenicity
disrupt
poorer
1a
mesh
smc5
screens



TREC-COVID emerges as the largest corpus with 171,331 documents, dwarfing SciFact (10,366 documents) and NFCorpus (7,266 documents). This substantial difference in corpus size is likely to influence retrieval performance and the effectiveness of various IR methods. TREC-COVID also boasts the most extensive vocabulary with 244,432 unique terms, compared to SciFact's 38,437 and NFCorpus's 29,444.

However, when considering the ratio of unique terms to total terms, NFCorpus and SciFact demonstrate relatively richer vocabularies (1.68% and 1.66% respectively) compared to TREC-COVID (0.85%). Interestingly, despite being the largest corpus, TREC-COVID has the shortest average document length at approximately 168 terms per document, while SciFact and NFCorpus have longer average lengths of 223 and 241 terms per document, respectively.

The term frequency plots for each dataset clearly illustrate Zipf's law, which states that the frequency of a word in a large corpus is inversely proportional to its rank in the frequency table. All three datasets exhibit Zipfian distributions, but with notable differences:

NFCorpus shows a classic Zipfian distribution with a steep initial decline and a long tail. SciFact's distribution is less steep initially and flattens out more quickly. TREC-COVID displays the most pronounced Zipfian distribution, with an extremely high frequency for COVID-related terms, reflecting its specialized nature.

Initialise packages and functions

You also need to decide which stemming algorithm and whether keep stopwords or not in the following cell.

```
!pip install matplotlib
!pip install seaborn
!pip install pytrec-eval-terrier
```

```
Requirement already satisfied: matplotlib in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (3.7.5)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (1.1.1)
Requirement already satisfied: cycler>=0.10 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (1.4.5)
Requirement already satisfied: numpy<2,>=1.20 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (1.24.4)
Requirement already satisfied: packaging>=20.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib) (6.4.0)
Requirement already satisfied: zipp>=3.1.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
importlib-resources>=3.2.0->matplotlib) (3.17.0)
Requirement already satisfied: six>=1.5 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from python-
dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: seaborn in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
seaborn) (1.24.4)
Requirement already satisfied: pandas>=1.2 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
seaborn) (2.0.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
seaborn) (3.7.5)
Requirement already satisfied: contourpy>=1.0.1 in
```

```
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.1.1)
Requirement already satisfied: cycler>=0.10 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=6.2.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (6.4.0)
Requirement already satisfied: pytz>=2020.1 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: zipp>=3.1.0 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from
importlib-resources>=3.2.0->matplotlib!=3.6.1,>=3.4->seaborn) (3.17.0)
Requirement already satisfied: six>=1.5 in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Requirement already satisfied: pytrec-eval-terrier in
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages (0.5.6)
```

```
stemming = None    # None or 'porter' or others
stopwords = False  # False or True
index = 'indexes/nfcorpus-index/' # Load the index of the dataset you
selected
```

Run the following cell to load and cache some useful packages and statistics that you will use later. We will alter adjust the caching to accomodate for vectorization

```python
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm

lucene_analyzer = get_lucene_analyzer(stemming=stemming,
stopwords=stopwords)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Get the total number of documents in the collection
total_doc_num = index_reader.stats()['documents']

# Get all document IDs of the collection
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in tqdm(range(total_doc_num))]

# Cache document vectors: dict{'doc_id': {'term_1': term_freq, ...}}
doc_vec_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vec:"):
        doc_vec_dict[docid] = index_reader.get_document_vector(docid)

# Cache document lengths for each document
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc len:"):
    doc_len_dict[docid] = sum(doc_vec_dict[docid].values())
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

100%|██████████| 7266/7266 [00:00<00:00, 50172.97it/s]
Caching doc vec:: 100%|██████████| 7266/7266 [00:05<00:00,
1343.47it/s]
Caching doc len:: 100%|██████████| 7266/7266 [00:00<00:00,
727624.22it/s]
```

3. Implementation of Retrieval Methods

The original version of the provided search function to be used in this experiment, we will later adjust this slightly to accomodate for vectorization of our bm25 funciton.

```python
def search(query: str, k: int=1000, scorer=None):
    """
    Inputs:
        query (str): the query string to perform the search.
        k (int): the number of documents to be returned.
        scorer: your implemented scoring function, such as bm25.

    Output:
        results (list): the sorted result list, a list of tuples.
                        The first element in the tuples is the docid,
                        the second is the doc score.
    """

    assert scorer is not None
    print("------------------------------------------------------")
    print("Current query:", query)

    # Get the analyzed term list
    q_terms = analyzer.analyze(query)
    doc_socres = {}
    for term in q_terms:
        # Get the posting list for the current term
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        # Get the document frequency of the current term
        df = index_reader.get_term_counts(term, analyzer=None)[0]
        if postings_list is not None:

            # Iterate the posting list
            for posting in tqdm(postings_list, desc=f"Iterate posting
for term '{term}'"):
                internal_id = posting.docid
                # Convert pyserini internal docid to the actual docid
                docid =
index_reader.convert_internal_docid_to_collection_docid(internal_id)
                tf = posting.tf
                # Use the cached dictionary.
                doc_len = doc_len_dict[docid]

                # Call the scoring function (you will implement these
below).
                score = scorer(tf, df, doc_len)
                if docid in doc_socres.keys():
                    doc_socres[docid] += score
                else:
                    doc_socres[docid] = score

    # Sort the results by the score.
    results = [(docid, doc_socre) for docid, doc_socre in
doc_socres.items()]
    results = sorted(results, key=lambda x: x[1], reverse=True)[:k]
    return results
    print("------------------------------------------------------")
```

As we progress, we will keep the search function fundamental at core, but will change it slightly to allow vectorization.

```python
# Import all your python libraries and put setup code here.
import math
from typing import Dict
from tqdm import tqdm
```

3.1 BM25 Baselines

BM25 (Best Matching 25) is a ranking function used in information retrieval to estimate the relevance of documents to a given search query. It extends the basic TF-IDF model by incorporating document length normalization and employing tunable parameters (k1 and b) to balance term frequency saturation and document length normalization, thus providing a more nuanced approach to document scoring.

In this section we are running the BM25 retrieval model with the test_queries of SciFact, NFCorpus, and TREC-COVID as the baselines with k=1.2, b=0.75.

```python

#Classic BM25 model
avg_dl = index_reader.stats()["total_terms"] / index_reader.stats()
["documents"]

# BM25 parameters
k_1 = 1.2 #1.8 for nfcorpus ish
b = 0.75

def bm25(tf, df, doc_len):
    # Calculate IDF
    N = index_reader.stats()["documents"]
    idf = math.log((N - df + 0.5) / (df + 0.5) + 1)

    # Calculate BM25 score
    numerator = idf * tf * (k_1 + 1)
    denominator = tf + k_1 * (1 - b + b * (doc_len / avg_dl))

    return numerator / denominator

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

```

Why vectorize the BM25 method?

Vectorization of the BM25 function offers several significant advantages over the non-vectorized approach, particularly in the context of large-scale information retrieval tasks. Three key reasons why we opted for vectorization in this experiment:

Computational Efficiency: The vectorized implementation leverages NumPy's highly optimized array operations, which are implemented in C under the hood. This allows for parallel processing of multiple elements simultaneously, as opposed to the sequential processing in the non-vectorized version. For large datasets like TREC-COVID with over 170,000 documents, this translates to a substantial reduction in computation time. The speedup can be orders of magnitude faster, especially when calculating scores for numerous query-document pairs.

Memory Efficiency: Vectorization allows us to work with entire arrays of term frequencies and document lengths at once, rather than processing each document individually. This approach is more memory-efficient, as it reduces the overhead of function calls and loop iterations. In Python, loops are relatively slow, and function calls have some overhead. By operating on entire arrays at once, we minimize these inefficiencies, leading to better memory utilization, particularly crucial when dealing with large-scale retrieval tasks.

Loop elimination: Vectorization eliminates loops, which reduces interpreter overhead, allows for more efficient memory access patterns, and enables better compiler and CPU-level optimizations, collectively leading to significant performance improvements in computational tasks.

3.1.1 Baseline BM25 with NFCorpus Test Queries

```python
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/nfcorpus-index/'  # Adjust this path as needed
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
# Adjust stemming and stopwords as needed
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
```

```
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score
    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def run_search(run_file_name, k1=1.2, b=0.75):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/nfcorpus/nfcorpus_test_queries.ts
v', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc="Processing
queries"):
            results = search(query_text, k=1000, k1=k1, b=b)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Run the search
run_search("nfcorpus-bm25-baseline.run", k1=1.2, b=0.75)

SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 7266/7266
[00:04<00:00, 1664.74it/s]
Processing queries: 100%|██████████| 323/323 [00:11<00:00, 27.09it/s]

Search completed. Results written to 'nfcorpus-bm25-baseline.run'



#Function to load qrels
def load_qrels(qrels_path):
    qrels = {}
    with open(qrels_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, relevance = line.strip().split()
            if query_id not in qrels:
                qrels[query_id] = {}
            qrels[query_id][doc_id] = int(relevance)
    return qrels

# Function to load the run file
```

```python
def load_run(run_file_path):
    run = defaultdict(dict)
    with open(run_file_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, rank, score, _ = line.strip().split()
            run[query_id][doc_id] = float(score)
    return run

#lets evaluate our baseline run
import pytrec_eval

def load_qrels(file_path):
    qrels = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rel = line.strip().split()
            if qid not in qrels:
                qrels[qid] = {}
            qrels[qid][docid] = int(rel)
    return qrels

def load_run(file_path):
    run = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rank, score, _ = line.strip().split()
            if qid not in run:
                run[qid] = {}
            run[qid][docid] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels, 
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    # Collect individual NDCG scores
    ndcg_scores = {qid: query_measures['ndcg_cut_10'] for qid, 
query_measures in results.items()}
    mean_ndcg = sum(ndcg_scores.values()) / len(ndcg_scores)

    return mean_ndcg, ndcg_scores

# Load qrels and run
qrels = 
load_qrels('./infs7410_project_collections/nfcorpus/nfcorpus_test_qrel
s.txt')
run_file_path = 'nfcorpus-bm25-baseline.run'
run = load_run(run_file_path)
```

```
# Evaluate run
mean_ndcg, ndcg_scores = evaluate_run(run, qrels)

# Print result
print(f"Mean nDCG@10 for the BM25 baseline run: {mean_ndcg:.4f}")

# Write results to file
output_file_path = 'nfcorpus_bm25_baseline_ndcg10_detailed.txt'
with open(output_file_path, 'w') as f:
    f.write(f"Mean nDCG@10 for the BM25 baseline run: {mean_ndcg:.4f}\
n\n")
    f.write("Individual query nDCG@10 scores:\n")
    for qid, score in sorted(ndcg_scores.items()):
        f.write(f"Query {qid}: {score:.4f}\n")

print(f"Detailed results have been written to {output_file_path}")

# Optionally, you can store all scores in a list if needed for further
analysis
baseline_scores = list(ndcg_scores.values())
print(f"Number of queries: {len(baseline_scores)}")
print(f"First few scores: {baseline_scores[:5]}")
```

```
Mean nDCG@10 for the BM25 baseline run: 0.3353
Detailed results have been written to
nfcorpus_bm25_baseline_ndcg10_detailed.txt
Number of queries: 298
First few scores: [0.6387878864795979, 0.0, 0.0, 0.30523488393970116,
0.7534494445524138]
```

3.1.2 Baseline BM25 with SciFact Test Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/scifact-index/'  # Adjust this path as needed
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
# Adjust stemming and stopwords as needed
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
```

```
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)
```

```
            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def run_search(run_file_name, k1=1.2, b=0.75):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/scifact/scifact_test_queries.tsv'
, 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc="Processing
queries"):
            results = search(query_text, k=1000, k1=k1, b=b)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Run the search
run_search("scifact-bm25-baseline.run", k1=1.2, b=0.75)
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 10366/10366
[00:05<00:00, 1758.00it/s]
Processing queries: 100%|██████████| 300/300 [00:59<00:00,  5.03it/s]

Search completed. Results written to 'scifact-bm25-baseline.run'
```

```
import pytrec_eval

def load_qrels(file_path):
    qrels = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rel = line.strip().split()
            if qid not in qrels:
                qrels[qid] = {}
            qrels[qid][docid] = int(rel)
    return qrels

def load_run(file_path):
    run = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rank, score, _ = line.strip().split()
            if qid not in run:
                run[qid] = {}
            run[qid][docid] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    # Collect individual NDCG scores
    ndcg_scores = {qid: query_measures['ndcg_cut_10'] for qid,
query_measures in results.items()}
    mean_ndcg = sum(ndcg_scores.values()) / len(ndcg_scores)

    return mean_ndcg, ndcg_scores

# Load qrels and run
qrels =
load_qrels('./infs7410_project_collections/scifact/scifact_test_qrels.
txt')
run_file_path = 'scifact-bm25-baseline.run'
run = load_run(run_file_path)

# Evaluate run
mean_ndcg, ndcg_scores = evaluate_run(run, qrels)

# Print result
print(f"Mean nDCG@10 for the BM25 baseline run: {mean_ndcg:.4f}")

# Write results to file
output_file_path = 'scifact_bm25_baseline_ndcg10_detailed.txt'
with open(output_file_path, 'w') as f:
    f.write(f"Mean nDCG@10 for the BM25 baseline run: {mean_ndcg:.4f}\
n\n")
    f.write("Individual query nDCG@10 scores:\n")
    for qid, score in sorted(ndcg_scores.items()):
        f.write(f"Query {qid}: {score:.4f}\n")

print(f"Detailed results have been written to {output_file_path}")
```

```
# Optionally, you can store all scores in a list if needed for further
analysis
baseline_scores = list(ndcg_scores.values())
print(f"Number of queries: {len(baseline_scores)}")
print(f"First few scores: {baseline_scores[:5]}")
```

```
Mean nDCG@10 for the BM25 baseline run: 0.6657
Detailed results have been written to
scifact_bm25_baseline_ndcg10_detailed.txt
Number of queries: 300
First few scores: [1.0, 0.2640681225725909, 0.20438239758848611, 1.0,
1.0]
```

3.1.3 Baseline BM25 with Trec-Covid Test Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/trec-covid-index/'  # Adjust this path as needed
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
# Adjust stemming and stopwords as needed
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
```

```python
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def run_search(run_file_name, k1=1.2, b=0.75):
    # Read queries from the TSV file
    with open('./infs7410_project_collections/trec-covid/trec-
covid_test_queries.tsv', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc="Processing
queries"):
            results = search(query_text, k=1000, k1=k1, b=b)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Run the search
run_search("trec-covid-bm25-baseline.run", k1=1.2, b=0.75)
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 171331/171331
[01:23<00:00, 2058.01it/s]
Processing queries: 100%|██████████| 50/50 [05:01<00:00,  6.02s/it]

Search completed. Results written to 'trec-covid-bm25-baseline.run'
```

```
#Lets evaluate our baseline bm25 run
qrels = load_qrels('./infs7410_project_collections/trec-covid/trec-
covid_test_qrels.txt')

# Function for Rank-based IR measure ndcg@10 to evaluate run
def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    # Calculate mean ndcg_cut_10
    ndcg_scores = [query_measures['ndcg_cut_10'] for query_measures in
results.values()]
    mean_ndcg = sum(ndcg_scores) / len(ndcg_scores)

    return mean_ndcg

run_file_path = 'trec-covid-bm25-baseline.run'  # Adjust this to your
run file path
run = load_run(run_file_path)

# Evaluate run
ndcg_10 = evaluate_run(run, qrels)

# Print result
```

```
print(f"nDCG@10 for the BM25 baseline run: {ndcg_10:.4f}")

# Write results to file
output_file_path = 'trec-covid_bm25_baseline_ndcg10.txt'
with open(output_file_path, 'w') as f:
    f.write(f"Mean nDCG@10 for the BM25 baseline run: 
{mean_ndcg_10:.4f}\n")
    f.write("Individual query nDCG@10 scores:\n")
    for query, score in individual_ndcg_scores.items():
        f.write(f"Query {query}: {score:.4f}\n")

print(f"Results have been written to {output_file_path}")
```

```
nDCG@10 for the BM25 baseline run: 0.5829
Results have been written to trec-covid_bm25_baseline_ndcg10.txt
```

Table 1: BM25 Baseline Performance on Test Queries

Dataset	Mean nDCG@10
NFCorpus	0.3353
SciFact	0.6657
TREC-COVID	0.5829

Table 1 presents the baseline performance of BM25 using default parameters (k1 = 1.2, b = 0.75) on the test queries for NFCorpus, SciFact, and TREC-COVID datasets. The performance is measured using mean nDCG@10 (normalized Discounted Cumulative Gain at rank 10).

These results provide a foundation for comparison in our study. SciFact shows the highest performance with an nDCG@10 of 0.6657, followed by TREC-COVID at 0.5829. NFCorpus demonstrates the lowest baseline performance at 0.3353.

3.2 Tuning Parameters

In this section we will tune the parameters of the BM25 model, as well as Query Expansion and Query Reductio on the training queries of the 2 source query datasets.

We will choose 10 pairs (or more accurately permutations) of k_1 and b which are the tuning parameters of the BM25 model and 5 pairs for query expansion parameters, and 5 values for query reduction

3.2.1 Tuning BM25 Parameters for nfCorpus Train Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict
```

```python
# Initialize components
index = 'indexes/nfcorpus-index/'
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def run_search(run_file_name, k1, b):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/nfcorpus/nfcorpus_train_queries.t
sv', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc=f"Processing
queries (k1={k1}, b={b})"):
            results = search(query_text, k=1000, k1=k1, b=b)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Define 10 reasonable pairs of k1 and b
parameter_pairs = [
    (0.6, 0.5),
    (0.8, 0.6),
    (1.0, 0.7),
    (1.2, 0.75),  # Default values
    (1.4, 0.8),
    (1.6, 0.85),
    (1.8, 0.9),
    (2.0, 0.95),
```

```python
    (2.2, 0.7),
    (2.4, 0.8)
]

# Run the search for each parameter pair
for k1, b in parameter_pairs:
    run_file_name = f"nfcorpus-bm25-k1_{k1:.1f}-b_{b:.2f}.run"
    run_search(run_file_name, k1, b)
    print(f"Completed run for k1={k1}, b={b}")

print("All parameter tuning runs completed.")
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 7266/7266
[00:04<00:00, 1647.68it/s]
Processing queries (k1=0.6, b=0.5): 100%|██████████| 324/324
[00:10<00:00, 29.55it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_0.6-b_0.50.run'
Completed run for k1=0.6, b=0.5

Processing queries (k1=0.8, b=0.6): 100%|██████████| 324/324
[00:10<00:00, 30.12it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_0.8-b_0.60.run'
Completed run for k1=0.8, b=0.6

Processing queries (k1=1.0, b=0.7): 100%|██████████| 324/324
[00:10<00:00, 32.08it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_1.0-b_0.70.run'
Completed run for k1=1.0, b=0.7

Processing queries (k1=1.2, b=0.75): 100%|██████████| 324/324
[00:12<00:00, 25.98it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_1.2-b_0.75.run'
Completed run for k1=1.2, b=0.75

Processing queries (k1=1.4, b=0.8): 100%|██████████| 324/324
[00:12<00:00, 26.19it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_1.4-b_0.80.run'
Completed run for k1=1.4, b=0.8

Processing queries (k1=1.6, b=0.85): 100%|██████████| 324/324
[00:11<00:00, 28.39it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_1.6-b_0.85.run'
Completed run for k1=1.6, b=0.85
```

```
Processing queries (k1=1.8, b=0.9): 100%|██████████| 324/324
[00:10<00:00, 31.05it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_1.8-b_0.90.run'
Completed run for k1=1.8, b=0.9

Processing queries (k1=2.0, b=0.95): 100%|██████████| 324/324
[00:10<00:00, 30.85it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_2.0-b_0.95.run'
Completed run for k1=2.0, b=0.95

Processing queries (k1=2.2, b=0.7): 100%|██████████| 324/324
[00:10<00:00, 31.75it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_2.2-b_0.70.run'
Completed run for k1=2.2, b=0.7

Processing queries (k1=2.4, b=0.8): 100%|██████████| 324/324
[00:09<00:00, 33.68it/s]

Search completed. Results written to 'nfcorpus-bm25-k1_2.4-b_0.80.run'
Completed run for k1=2.4, b=0.8
All parameter tuning runs completed.



# Lets evaluate our tuned run
import pytrec_eval
from collections import defaultdict

def load_qrels(qrels_path):
    qrels = {}
    with open(qrels_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, relevance = line.strip().split()
            if query_id not in qrels:
                qrels[query_id] = {}
            qrels[query_id][doc_id] = int(relevance)
    return qrels

def load_run(run_file_path):
    run = defaultdict(dict)
    with open(run_file_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, rank, score, _ = line.strip().split()
            run[query_id][doc_id] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
```

```
    results = evaluator.evaluate(run_results)

    ndcg_scores = [query_measures['ndcg_cut_10'] for query_measures in 
results.values()]
    mean_ndcg = sum(ndcg_scores) / len(ndcg_scores)

    return mean_ndcg

# Load qrels
qrels = 
load_qrels('./infs7410_project_collections/nfcorpus/nfcorpus_train_qre
ls.txt')

# Define parameter pairs
param_pairs = [
    (0.6, 0.5),
    (0.8, 0.6),
    (1.0, 0.7),
    (1.2, 0.75),  # Default values
    (1.4, 0.8),
    (1.6, 0.85),
    (1.8, 0.9),
    (2.0, 0.95),
    (2.2, 0.7),
    (2.4, 0.8)
]

results = []

# Evaluate each run
for k1, b in param_pairs:
    run_file_path = f'nfcorpus-bm25-k1_{k1:.1f}-b_{b:.2f}.run'
    run = load_run(run_file_path)
    ndcg_10 = evaluate_run(run, qrels)
    results.append((k1, b, ndcg_10))
    print(f"nDCG@10 for BM25 run (k1={k1:.1f}, b={b:.2f}): 
{ndcg_10:.4f}")

# Sort results by nDCG@10 score
results.sort(key=lambda x: x[2], reverse=True)

# Write results to file
output_file_path = 'nfcorpus_bm25_tuning_results.txt'
with open(output_file_path, 'w') as f:
    f.write("BM25 Parameter Tuning Results for nfcorpus\n")
    f.write("=========================================\n\n")
    f.write("k1\tb\tnDCG@10\n")
    for k1, b, ndcg_10 in results:
        f.write(f"{k1:.1f}\t{b:.2f}\t{ndcg_10:.4f}\n")
    f.write("\nBest performing parameters:\n")
    best_k1, best_b, best_ndcg = results[0]
    f.write(f"k1 = {best_k1:.1f}, b = {best_b:.2f}, nDCG@10 =
{best_ndcg:.4f}\n")

print(f"Results have been written to {output_file_path}")
```

```
nDCG@10 for BM25 run (k1=0.6, b=0.50): 0.2855
nDCG@10 for BM25 run (k1=0.8, b=0.60): 0.2872
nDCG@10 for BM25 run (k1=1.0, b=0.70): 0.2888
nDCG@10 for BM25 run (k1=1.2, b=0.75): 0.2889
nDCG@10 for BM25 run (k1=1.4, b=0.80): 0.2904
nDCG@10 for BM25 run (k1=1.6, b=0.85): 0.2896
nDCG@10 for BM25 run (k1=1.8, b=0.90): 0.2881
nDCG@10 for BM25 run (k1=2.0, b=0.95): 0.2871
nDCG@10 for BM25 run (k1=2.2, b=0.70): 0.2907
nDCG@10 for BM25 run (k1=2.4, b=0.80): 0.2889
Results have been written to nfcorpus_bm25_tuning_results.txt
```

3.2.2 Tuning BM25 Parameters for Scifact Train Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/scifact-index/'
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def run_search(run_file_name, k1, b):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/scifact/scifact_train_queries.tsv
', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc=f"Processing
queries (k1={k1}, b={b})"):
            results = search(query_text, k=1000, k1=k1, b=b)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Define 10 reasonable pairs of k1 and b
parameter_pairs = [
    (0.6, 0.5),
    (0.8, 0.6),
    (1.0, 0.7),
    (1.2, 0.75),  # Default values
    (1.4, 0.8),
    (1.6, 0.85),
    (1.8, 0.9),
    (2.0, 0.95),
    (2.2, 0.7),
    (2.4, 0.8)
]

# Run the search for each parameter pair
for k1, b in parameter_pairs:
    run_file_name = f"scifact-bm25-k1_{k1:.1f}-b_{b:.2f}.run"
    run_search(run_file_name, k1, b)
    print(f"Completed run for k1={k1}, b={b}")

print("All parameter tuning runs completed.")
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 10366/10366
[00:07<00:00, 1453.65it/s]
Processing queries (k1=0.6, b=0.5): 100%|██████████| 809/809
[02:33<00:00,  5.26it/s]

Search completed. Results written to 'scifact-bm25-k1_0.6-b_0.50.run'
Completed run for k1=0.6, b=0.5
```

```
Processing queries (k1=0.8, b=0.6): 100%|██████████| 809/809
[02:29<00:00,  5.42it/s]

Search completed. Results written to 'scifact-bm25-k1_0.8-b_0.60.run'
Completed run for k1=0.8, b=0.6

Processing queries (k1=1.0, b=0.7): 100%|██████████| 809/809
[02:47<00:00,  4.84it/s]

Search completed. Results written to 'scifact-bm25-k1_1.0-b_0.70.run'
Completed run for k1=1.0, b=0.7

Processing queries (k1=1.2, b=0.75): 100%|██████████| 809/809
[02:44<00:00,  4.92it/s]

Search completed. Results written to 'scifact-bm25-k1_1.2-b_0.75.run'
Completed run for k1=1.2, b=0.75

Processing queries (k1=1.4, b=0.8): 100%|██████████| 809/809
[02:43<00:00,  4.95it/s]

Search completed. Results written to 'scifact-bm25-k1_1.4-b_0.80.run'
Completed run for k1=1.4, b=0.8

Processing queries (k1=1.6, b=0.85): 100%|██████████| 809/809
[02:44<00:00,  4.92it/s]

Search completed. Results written to 'scifact-bm25-k1_1.6-b_0.85.run'
Completed run for k1=1.6, b=0.85

Processing queries (k1=1.8, b=0.9): 100%|██████████| 809/809
[02:44<00:00,  4.91it/s]

Search completed. Results written to 'scifact-bm25-k1_1.8-b_0.90.run'
Completed run for k1=1.8, b=0.9

Processing queries (k1=2.0, b=0.95): 100%|██████████| 809/809
[02:44<00:00,  4.92it/s]

Search completed. Results written to 'scifact-bm25-k1_2.0-b_0.95.run'
Completed run for k1=2.0, b=0.95

Processing queries (k1=2.2, b=0.7): 100%|██████████| 809/809
[02:57<00:00,  4.57it/s]

Search completed. Results written to 'scifact-bm25-k1_2.2-b_0.70.run'
Completed run for k1=2.2, b=0.7

Processing queries (k1=2.4, b=0.8): 100%|██████████| 809/809
[02:51<00:00,  4.71it/s]
```

```
Search completed. Results written to 'scifact-bm25-k1_2.4-b_0.80.run'
Completed run for k1=2.4, b=0.8
All parameter tuning runs completed.
```

```
#Lets evaluate our tuned run
import pytrec_eval
from collections import defaultdict

def load_qrels(qrels_path):
    qrels = {}
    with open(qrels_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, relevance = line.strip().split()
            if query_id not in qrels:
                qrels[query_id] = {}
            qrels[query_id][doc_id] = int(relevance)
    return qrels

def load_run(run_file_path):
    run = defaultdict(dict)
    with open(run_file_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, rank, score, _ = line.strip().split()
            run[query_id][doc_id] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = [query_measures['ndcg_cut_10'] for query_measures in
results.values()]
    mean_ndcg = sum(ndcg_scores) / len(ndcg_scores)

    return mean_ndcg

# Load qrels
qrels =
load_qrels('./infs7410_project_collections/scifact/scifact_train_qrels
.txt')

# Define parameter pairs
param_pairs = [
    (0.6, 0.5),
    (0.8, 0.6),
    (1.0, 0.7),
    (1.2, 0.75),  # Default values
    (1.4, 0.8),
    (1.6, 0.85),
    (1.8, 0.9),
    (2.0, 0.95),
    (2.2, 0.7),
    (2.4, 0.8)
]

results = []

# Evaluate each run
for k1, b in param_pairs:
    run_file_path = f'scifact-bm25-k1_{k1:.1f}-b_{b:.2f}.run'
    run = load_run(run_file_path)
    ndcg_10 = evaluate_run(run, qrels)
    results.append((k1, b, ndcg_10))
    print(f"nDCG@10 for BM25 run (k1={k1:.1f}, b={b:.2f}):
{ndcg_10:.4f}")

# Sort results by nDCG@10 score
results.sort(key=lambda x: x[2], reverse=True)

# Write results to file
output_file_path = 'scifact_bm25_tuning_results.txt'
with open(output_file_path, 'w') as f:
    f.write("BM25 Parameter Tuning Results for scifact\n")
    f.write("=========================================\n\n")
    f.write("k1\tb\tnDCG@10\n")
    for k1, b, ndcg_10 in results:
        f.write(f"{k1:.1f}\t{b:.2f}\t{ndcg_10:.4f}\n")

    f.write("\nBest performing parameters:\n")
    best_k1, best_b, best_ndcg = results[0]
    f.write(f"k1 = {best_k1:.1f}, b = {best_b:.2f}, nDCG@10 =
{best_ndcg:.4f}\n")

print(f"Results have been written to {output_file_path}")
```

```
nDCG@10 for BM25 run (k1=0.6, b=0.50): 0.6628
nDCG@10 for BM25 run (k1=0.8, b=0.60): 0.6662
nDCG@10 for BM25 run (k1=1.0, b=0.70): 0.6709
nDCG@10 for BM25 run (k1=1.2, b=0.75): 0.6694
nDCG@10 for BM25 run (k1=1.4, b=0.80): 0.6682
nDCG@10 for BM25 run (k1=1.6, b=0.85): 0.6660
nDCG@10 for BM25 run (k1=1.8, b=0.90): 0.6647
nDCG@10 for BM25 run (k1=2.0, b=0.95): 0.6645
nDCG@10 for BM25 run (k1=2.2, b=0.70): 0.6685
nDCG@10 for BM25 run (k1=2.4, b=0.80): 0.6679
Results have been written to scifact_bm25_tuning_results.txt
```

3.2.3 Tuning Query Expansion parameters for Scifact Train Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/scifact-index/'
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
```

```
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def prf_query_expansion(query: str, n: int, m: int):
    # 1. Initial Document Ranking
    hits = search(query, k=10)

    # 2. Select Top n Documents
    top_n_docs = hits[:n]

    # 3. Term Ranking in Selected Documents
    term_scores = defaultdict(float)
    original_terms = set(analyzer.analyze(query))

    for docid, _ in top_n_docs:
        doc_vector = index_reader.get_document_vector(docid)
        doc_length = sum(doc_vector.values())

        for term, tf in doc_vector.items():
            if term not in original_terms:
                df = index_reader.get_term_counts(term, analyzer=None)
[0]
                idf = math.log(N / df)

                # TF-IDF score
                tfidf = (tf / doc_length) * idf

                term_scores[term] += tfidf

    # 4. Expand the Query
    # Sort terms by score in descending order and select top m
    expansion_terms = sorted(term_scores.items(), key=lambda x: x[1],
reverse=True)[:m]

    # Add top m terms to the original query
    expanded_query = query + " " + " ".join(term for term, _ in
expansion_terms)

    return expanded_query

def run_search_with_expansion(run_file_name, n, m):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/scifact/scifact_train_queries.tsv
', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc=f"Processing
queries (n={n}, m={m})"):
            expanded_query = prf_query_expansion(query_text, n, m)
            results = search(expanded_query, k=1000)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Define pairs of n and m to tune
expansion_pairs = [
    (2, 3),   # Very conservative, minimal expansion
    (3, 5),   # Conservative, slight expansion
    (4, 7),   # Moderate expansion
    (5, 8),   # Balanced expansion
    (6, 10)   # Slightly more aggressive, but still reasonable
]

# Run the search for each parameter pair
for n, m in expansion_pairs:
    run_file_name = f"scifact-bm25-qe-n_{n}-m_{m}.run"
    run_search_with_expansion(run_file_name, n, m)
```

```python
    print(f"Completed run for n={n}, m={m}")

print("All parameter tuning runs completed.")
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 10366/10366
[00:05<00:00, 1744.50it/s]
Processing queries (n=2, m=3): 100%|██████████| 809/809 [06:28<00:00,
2.08it/s]

Search completed. Results written to 'scifact-bm25-qe-n_2-m_3.run'
Completed run for n=2, m=3

Processing queries (n=3, m=5): 100%|██████████| 809/809 [06:37<00:00,
2.03it/s]

Search completed. Results written to 'scifact-bm25-qe-n_3-m_5.run'
Completed run for n=3, m=5

Processing queries (n=4, m=7): 100%|██████████| 809/809 [06:51<00:00,
1.97it/s]

Search completed. Results written to 'scifact-bm25-qe-n_4-m_7.run'
Completed run for n=4, m=7

Processing queries (n=5, m=8): 100%|██████████| 809/809 [06:57<00:00,
1.94it/s]

Search completed. Results written to 'scifact-bm25-qe-n_5-m_8.run'
Completed run for n=5, m=8

Processing queries (n=6, m=10): 100%|██████████| 809/809 [07:23<00:00,
1.83it/s]

Search completed. Results written to 'scifact-bm25-qe-n_6-m_10.run'
Completed run for n=6, m=10
All parameter tuning runs completed.
```

```python
#Lets evaluate the query expansion tuning
import pytrec_eval
from collections import defaultdict

def load_qrels(qrels_path):
    qrels = {}
    with open(qrels_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, relevance = line.strip().split()
            if query_id not in qrels:
                qrels[query_id] = {}
            qrels[query_id][doc_id] = int(relevance)
    return qrels

def load_run(run_file_path):
    run = defaultdict(dict)
    with open(run_file_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, rank, score, _ = line.strip().split()
            run[query_id][doc_id] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = [query_measures['ndcg_cut_10'] for query_measures in
results.values()]
    mean_ndcg = sum(ndcg_scores) / len(ndcg_scores)

    return mean_ndcg

# Load qrels
qrels =
load_qrels('./infs7410_project_collections/scifact/scifact_train_qrels
.txt')

# Define query expansion parameter pairs
expansion_pairs = [
    (2, 3),   # Very conservative, minimal expansion
    (3, 5),   # Conservative, slight expansion
    (4, 7),   # Moderate expansion
    (5, 8),   # Balanced expansion
    (6, 10)   # Slightly more aggressive, but still reasonable
]

results = []

# Evaluate each run
for n, m in expansion_pairs:
    run_file_path = f'scifact-bm25-qe-n_{n}-m_{m}.run'
    try:
        run = load_run(run_file_path)
        ndcg_10 = evaluate_run(run, qrels)
        results.append((n, m, ndcg_10))
        print(f"nDCG@10 for Query Expansion run (n={n}, m={m}):
{ndcg_10:.4f}")
    except FileNotFoundError:
        print(f"Run file not found: {run_file_path}")
```

```
# Sort results by nDCG@10 score
results.sort(key=lambda x: x[2], reverse=True)

# Write results to file
output_file_path = 'scifact_query_expansion_results.txt'
with open(output_file_path, 'w') as f:
    f.write("Query Expansion Results for SciFact\n")
    f.write("===================================\n\n")
    f.write("n\tm\tnDCG@10\n")
    for n, m, ndcg_10 in results:
        f.write(f"{n}\t{m}\t{ndcg_10:.4f}\n")

    if results:
        f.write("\nBest performing parameters:\n")
        best_n, best_m, best_ndcg = results[0]
        f.write(f"n = {best_n}, m = {best_m}, nDCG@10 =
{best_ndcg:.4f}\n")
    else:
        f.write("\nNo valid results found.\n")

print(f"Results have been written to {output_file_path}")

nDCG@10 for Query Expansion run (n=2, m=3): 0.5966
nDCG@10 for Query Expansion run (n=3, m=5): 0.5664
nDCG@10 for Query Expansion run (n=4, m=7): 0.5470
nDCG@10 for Query Expansion run (n=5, m=8): 0.5335
nDCG@10 for Query Expansion run (n=6, m=10): 0.5217
Results have been written to scifact_query_expansion_results.txt
```

3.2.4 Tuning Query Expansion parameters for nfCorpus Train Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/nfcorpus-index/'
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
```

```
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)
```

```
            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def prf_query_expansion(query: str, n: int, m: int):
    # 1. Initial Document Ranking
    hits = search(query, k=10)

    # 2. Select Top n Documents
    top_n_docs = hits[:n]

    # 3. Term Ranking in Selected Documents
    term_scores = defaultdict(float)
    original_terms = set(analyzer.analyze(query))

    for docid, _ in top_n_docs:
        doc_vector = index_reader.get_document_vector(docid)
        doc_length = sum(doc_vector.values())

        for term, tf in doc_vector.items():
            if term not in original_terms:
                df = index_reader.get_term_counts(term, analyzer=None)
[0]
                idf = math.log(N / df)

                # TF-IDF score
                tfidf = (tf / doc_length) * idf

                term_scores[term] += tfidf

    # 4. Expand the Query
    # Sort terms by score in descending order and select top m
    expansion_terms = sorted(term_scores.items(), key=lambda x: x[1],
reverse=True)[:m]

    # Add top m terms to the original query
    expanded_query = query + " " + " ".join(term for term, _ in
expansion_terms)

    return expanded_query

def run_search_with_expansion(run_file_name, n, m):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/nfcorpus/nfcorpus_train_queries.t
sv', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc=f"Processing
queries (n={n}, m={m})"):
            expanded_query = prf_query_expansion(query_text, n, m)
            results = search(expanded_query, k=1000)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Define pairs of n and m to tune
expansion_pairs = [
    (2, 3),   # Very conservative, minimal expansion
    (3, 5),   # Conservative, slight expansion
    (4, 7),   # Moderate expansion
    (5, 8),   # Balanced expansion
    (6, 10)   # Slightly more aggressive, but still reasonable
]

# Run the search for each parameter pair
for n, m in expansion_pairs:
    run_file_name = f"nfcorpus-bm25-qe-n_{n}-m_{m}.run"
    run_search_with_expansion(run_file_name, n, m)
    print(f"Completed run for n={n}, m={m}")

print("All parameter tuning runs completed.")
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 7266/7266
[00:06<00:00, 1207.45it/s]
Processing queries (n=2, m=3): 100%|██████████| 324/324 [00:31<00:00,
10.43it/s]

Search completed. Results written to 'nfcorpus-bm25-qe-n_2-m_3.run'
Completed run for n=2, m=3

Processing queries (n=3, m=5): 100%|██████████| 324/324 [00:38<00:00,
8.50it/s]

Search completed. Results written to 'nfcorpus-bm25-qe-n_3-m_5.run'
Completed run for n=3, m=5
```

```
Processing queries (n=4, m=7): 100%|██████████| 324/324 [00:42<00:00,
7.62it/s]

Search completed. Results written to 'nfcorpus-bm25-qe-n_4-m_7.run'
Completed run for n=4, m=7

Processing queries (n=5, m=8): 100%|██████████| 324/324 [00:46<00:00,
6.93it/s]

Search completed. Results written to 'nfcorpus-bm25-qe-n_5-m_8.run'
Completed run for n=5, m=8

Processing queries (n=6, m=10): 100%|██████████| 324/324 [00:52<00:00,
6.19it/s]

Search completed. Results written to 'nfcorpus-bm25-qe-n_6-m_10.run'
Completed run for n=6, m=10
All parameter tuning runs completed.
```

```python
#Lets evaluate the query expansion tuning
import pytrec_eval
from collections import defaultdict

def load_qrels(qrels_path):
    qrels = {}
    with open(qrels_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, relevance = line.strip().split()
            if query_id not in qrels:
                qrels[query_id] = {}
            qrels[query_id][doc_id] = int(relevance)
    return qrels

def load_run(run_file_path):
    run = defaultdict(dict)
    with open(run_file_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, rank, score, _ = line.strip().split()
            run[query_id][doc_id] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = [query_measures['ndcg_cut_10'] for query_measures in
results.values()]
    mean_ndcg = sum(ndcg_scores) / len(ndcg_scores)
    return mean_ndcg

# Load qrels
qrels =
load_qrels('./infs7410_project_collections/nfcorpus/nfcorpus_train_qre
ls.txt')

# Define query expansion parameter pairs
expansion_pairs = [
    (2, 3),   # Very conservative, minimal expansion
    (3, 5),   # Conservative, slight expansion
    (4, 7),   # Moderate expansion
    (5, 8),   # Balanced expansion
    (6, 10)   # Slightly more aggressive, but still reasonable
]

results = []

# Evaluate each run
for n, m in expansion_pairs:
    run_file_path = f'nfcorpus-bm25-qe-n_{n}-m_{m}.run'
    try:
        run = load_run(run_file_path)
        ndcg_10 = evaluate_run(run, qrels)
        results.append((n, m, ndcg_10))
        print(f"nDCG@10 for Query Expansion run (n={n}, m={m}):
{ndcg_10:.4f}")
    except FileNotFoundError:
        print(f"Run file not found: {run_file_path}")

# Sort results by nDCG@10 score
results.sort(key=lambda x: x[2], reverse=True)

# Write results to file
output_file_path = 'nfcorpus_query_expansion_results.txt'
with open(output_file_path, 'w') as f:
    f.write("Query Expansion Results for NfCorpus\n")
    f.write("===================================\n\n")
    f.write("n\tm\tnDCG@10\n")
    for n, m, ndcg_10 in results:
        f.write(f"{n}\t{m}\t{ndcg_10:.4f}\n")

    if results:
        f.write("\nBest performing parameters:\n")
        best_n, best_m, best_ndcg = results[0]
        f.write(f"n = {best_n}, m = {best_m}, nDCG@10 =
{best_ndcg:.4f}\n")
    else:
        f.write("\nNo valid results found.\n")
```

```
print(f"Results have been written to {output_file_path}")
```

```
nDCG@10 for Query Expansion run (n=2, m=3): 0.2799
nDCG@10 for Query Expansion run (n=3, m=5): 0.2786
nDCG@10 for Query Expansion run (n=4, m=7): 0.2795
nDCG@10 for Query Expansion run (n=5, m=8): 0.2834
nDCG@10 for Query Expansion run (n=6, m=10): 0.2761
Results have been written to nfcorpus_query_expansion_results.txt
```

3.2.5 Tuning Query Reduction parameters for nfCorpus Train Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/nfcorpus-index/'
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator
```

```python
# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def idfr_query_reduction(query: str, n: int):
    terms = analyzer.analyze(query)

    # Calculate IDF for each term
    idf_scores = []
    for term in terms:
        df = index_reader.get_term_counts(term)[0]
        idf = math.log((N - df + 0.5) / (df + 0.5))
        idf_scores.append((term, idf))

    # Sort terms by IDF score and select top n
    sorted_terms = sorted(idf_scores, key=lambda x: x[1],
reverse=True)
```

```
    top_terms = [term for term, _ in sorted_terms[:n]]

    pruned_query = " ".join(top_terms)
    return pruned_query

def run_search_with_reduction(run_file_name, n):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/nfcorpus/nfcorpus_train_queries.t
sv', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc=f"Processing
queries (n={n})"):
            reduced_query = idfr_query_reduction(query_text, n)
            results = search(reduced_query, k=1000)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Define values of n to tune
n_values = [2, 4, 6, 8, 10]

# Run the search for each n value
for n in n_values:
    run_file_name = f"nfcorpus-bm25-qr-n_{n}.run"
    run_search_with_reduction(run_file_name, n)
    print(f"Completed run for n={n}")

print("All parameter tuning runs completed.")

SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 7266/7266
[00:04<00:00, 1659.27it/s]
Processing queries (n=2): 100%|██████████| 324/324 [00:03<00:00,
94.25it/s]

Search completed. Results written to 'nfcorpus-bm25-qr-n_2.run'
Completed run for n=2

Processing queries (n=4): 100%|██████████| 324/324 [00:07<00:00,
40.73it/s]
```

```
Search completed. Results written to 'nfcorpus-bm25-qr-n_4.run'
Completed run for n=4

Processing queries (n=6): 100%|██████████| 324/324 [00:12<00:00,
26.64it/s]

Search completed. Results written to 'nfcorpus-bm25-qr-n_6.run'
Completed run for n=6

Processing queries (n=8): 100%|██████████| 324/324 [00:12<00:00,
26.01it/s]

Search completed. Results written to 'nfcorpus-bm25-qr-n_8.run'
Completed run for n=8

Processing queries (n=10): 100%|██████████| 324/324 [00:11<00:00,
27.83it/s]

Search completed. Results written to 'nfcorpus-bm25-qr-n_10.run'
Completed run for n=10
All parameter tuning runs completed.
```

```python
#Lets evaluate query reduction tuning
import pytrec_eval
from collections import defaultdict

def load_qrels(qrels_path):
    qrels = {}
    with open(qrels_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, relevance = line.strip().split()
            if query_id not in qrels:
                qrels[query_id] = {}
            qrels[query_id][doc_id] = int(relevance)
    return qrels

def load_run(run_file_path):
    run = defaultdict(dict)
    with open(run_file_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, rank, score, _ = line.strip().split()
            run[query_id][doc_id] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)
    ndcg_scores = [query_measures['ndcg_cut_10'] for query_measures in
results.values()]
    mean_ndcg = sum(ndcg_scores) / len(ndcg_scores)

    return mean_ndcg

# Load qrels
qrels =
load_qrels('./infs7410_project_collections/nfcorpus/nfcorpus_train_qre
ls.txt')

# Define query reduction parameter values
n_values = [2, 4, 6, 8, 10]

results = []

# Evaluate each run
for n in n_values:
    run_file_path = f'nfcorpus-bm25-qr-n_{n}.run'
    try:
        run = load_run(run_file_path)
        ndcg_10 = evaluate_run(run, qrels)
        results.append((n, ndcg_10))
        print(f"nDCG@10 for Query Reduction run (n={n}):
{ndcg_10:.4f}")
    except FileNotFoundError:
        print(f"Run file not found: {run_file_path}")

# Sort results by nDCG@10 score
results.sort(key=lambda x: x[1], reverse=True)

# Write results to file
output_file_path = 'nfcorpus_query_reduction_results.txt'
with open(output_file_path, 'w') as f:
    f.write("Query Reduction Results for nfcorpus\n")
    f.write("===================================\n\n")
    f.write("n\tnDCG@10\n")
    for n, ndcg_10 in results:
        f.write(f"{n}\t{ndcg_10:.4f}\n")

    if results:
        f.write("\nBest performing parameter:\n")
        best_n, best_ndcg = results[0]
        f.write(f"n = {best_n}, nDCG@10 = {best_ndcg:.4f}\n")
    else:
        f.write("\nNo valid results found.\n")

print(f"Results have been written to {output_file_path}")
```

```
nDCG@10 for Query Reduction run (n=2): 0.2726
nDCG@10 for Query Reduction run (n=4): 0.2890
nDCG@10 for Query Reduction run (n=6): 0.2906
nDCG@10 for Query Reduction run (n=8): 0.2890
nDCG@10 for Query Reduction run (n=10): 0.2889
Results have been written to nfcorpus_query_reduction_results.txt
```

3.2.6 Tuning Query Reduction parameters for Scifact Train Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/scifact-index/'
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def idfr_query_reduction(query: str, n: int):
    terms = analyzer.analyze(query)

    # Calculate IDF for each term
    idf_scores = []
    for term in terms:
        df = index_reader.get_term_counts(term)[0]
        idf = math.log((N - df + 0.5) / (df + 0.5))
        idf_scores.append((term, idf))

    # Sort terms by IDF score and select top n
    sorted_terms = sorted(idf_scores, key=lambda x: x[1],
reverse=True)
    top_terms = [term for term, _ in sorted_terms[:n]]

    pruned_query = " ".join(top_terms)
    return pruned_query

def run_search_with_reduction(run_file_name, n):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/scifact/scifact_train_queries.tsv
', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc=f"Processing
queries (n={n})"):
            reduced_query = idfr_query_reduction(query_text, n)
            results = search(reduced_query, k=1000)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Define values of n to tune
n_values = [2, 4, 6, 8, 10]

# Run the search for each n value
for n in n_values:
    run_file_name = f"scifact-bm25-qr-n_{n}.run"
    run_search_with_reduction(run_file_name, n)
    print(f"Completed run for n={n}")

print("All parameter tuning runs completed.")

SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 10366/10366
[00:05<00:00, 1780.90it/s]
Processing queries (n=2): 100%|██████████| 809/809 [00:22<00:00,
35.38it/s]

Search completed. Results written to 'scifact-bm25-qr-n_2.run'
Completed run for n=2

Processing queries (n=4): 100%|██████████| 809/809 [00:40<00:00,
19.90it/s]

Search completed. Results written to 'scifact-bm25-qr-n_4.run'
Completed run for n=4
```

```
Processing queries (n=6): 100%|██████████| 809/809 [01:07<00:00,
11.91it/s]

Search completed. Results written to 'scifact-bm25-qr-n_6.run'
Completed run for n=6

Processing queries (n=8): 100%|██████████| 809/809 [01:36<00:00,
8.39it/s]

Search completed. Results written to 'scifact-bm25-qr-n_8.run'
Completed run for n=8

Processing queries (n=10): 100%|██████████| 809/809 [02:08<00:00,
6.29it/s]

Search completed. Results written to 'scifact-bm25-qr-n_10.run'
Completed run for n=10
All parameter tuning runs completed.
```

```python
#Lets evaluate query reduction tuning
import pytrec_eval
from collections import defaultdict

def load_qrels(qrels_path):
    qrels = {}
    with open(qrels_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, relevance = line.strip().split()
            if query_id not in qrels:
                qrels[query_id] = {}
            qrels[query_id][doc_id] = int(relevance)
    return qrels

def load_run(run_file_path):
    run = defaultdict(dict)
    with open(run_file_path, 'r') as f:
        for line in f:
            query_id, _, doc_id, rank, score, _ = line.strip().split()
            run[query_id][doc_id] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = [query_measures['ndcg_cut_10'] for query_measures in
results.values()]
    mean_ndcg = sum(ndcg_scores) / len(ndcg_scores)
    return mean_ndcg

# Load qrels
qrels = 
load_qrels('./infs7410_project_collections/scifact/scifact_train_qrels
.txt')

# Define query reduction parameter values
n_values = [2, 4, 6, 8, 10]

results = []

# Evaluate each run
for n in n_values:
    run_file_path = f'scifact-bm25-qr-n_{n}.run'
    try:
        run = load_run(run_file_path)
        ndcg_10 = evaluate_run(run, qrels)
        results.append((n, ndcg_10))
        print(f"nDCG@10 for Query Reduction run (n={n}): 
{ndcg_10:.4f}")
    except FileNotFoundError:
        print(f"Run file not found: {run_file_path}")

# Sort results by nDCG@10 score
results.sort(key=lambda x: x[1], reverse=True)

# Write results to file
output_file_path = 'scifact_query_reduction_results.txt'
with open(output_file_path, 'w') as f:
    f.write("Query Reduction Results for SciFact\n")
    f.write("===================================\n\n")
    f.write("n\tnDCG@10\n")
    for n, ndcg_10 in results:
        f.write(f"{n}\t{ndcg_10:.4f}\n")

    if results:
        f.write("\nBest performing parameter:\n")
        best_n, best_ndcg = results[0]
        f.write(f"n = {best_n}, nDCG@10 = {best_ndcg:.4f}\n")
    else:
        f.write("\nNo valid results found.\n")

print(f"Results have been written to {output_file_path}")
```

```
nDCG@10 for Query Reduction run (n=2): 0.2426
nDCG@10 for Query Reduction run (n=4): 0.5037
nDCG@10 for Query Reduction run (n=6): 0.6142
nDCG@10 for Query Reduction run (n=8): 0.6543
```

```
nDCG@10 for Query Reduction run (n=10): 0.6643
Results have been written to scifact_query_reduction_results.txt
```

4. Best Parameter Setting & Statistical Significance Analysis

4.1 Best Parameter Settings

4.1.1 Best Parameters Set on nfCorpus Test Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/nfcorpus-index/'  # Adjust this path as needed
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
# Adjust stemming and stopwords as needed
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def run_search(run_file_name, k1=1.2, b=0.75):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/nfcorpus/nfcorpus_test_queries.ts
v', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc="Processing
queries"):
            results = search(query_text, k=1000, k1=k1, b=b)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Run the search
run_search("nfcorpus-bm25-testq_with_bp.run", k1=2.2, b=0.70)
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 7266/7266
[00:04<00:00, 1619.74it/s]
Processing queries: 100%|██████████| 323/323 [00:14<00:00, 22.98it/s]

Search completed. Results written to 'nfcorpus-bm25-testq_with_bp.run'
```

```
#Let's evaluate
import pytrec_eval

def load_qrels(file_path):
    qrels = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rel = line.strip().split()
            if qid not in qrels:
                qrels[qid] = {}
            qrels[qid][docid] = int(rel)
    return qrels

def load_run(file_path):
    run = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rank, score, _ = line.strip().split()
            if qid not in run:
                run[qid] = {}
            run[qid][docid] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = {qid: query_measures['ndcg_cut_10'] for qid,
query_measures in results.items()}
    mean_ndcg = sum(ndcg_scores.values()) / len(ndcg_scores)

    return ndcg_scores, mean_ndcg

# Load qrels and run
qrels =
load_qrels('./infs7410_project_collections/nfcorpus/nfcorpus_test_qrel
s.txt')
run_file_path = 'nfcorpus-bm25-testq_with_bp.run'
run = load_run(run_file_path)

# Evaluate run
ndcg_scores, mean_ndcg = evaluate_run(run, qrels)

# Print result
print(f"Mean nDCG@10 for the BM25 baseline run with best params:
{mean_ndcg:.4f}")

# Write results to file
output_file_path = 'nfcorpus_bm25_testq_with_bp_results.txt'
with open(output_file_path, 'w') as f:
    f.write(f"Mean nDCG@10 for the BM25 baseline run with best params:
{mean_ndcg:.4f}\n\n")
    f.write("Individual query nDCG@10 scores:\n")
    for qid, score in sorted(ndcg_scores.items()):
        f.write(f"Query {qid}: {score:.4f}\n")

print(f"Detailed results have been written to {output_file_path}")

Mean nDCG@10 for the BM25 baseline run with best params: 0.3382
Detailed results have been written to
nfcorpus_bm25_testq_with_bp_results.txt
```

4.1.2 Best Parameters Set on Scifact Test Queries

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/scifact-index/'  # Adjust this path as needed
```

```
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
# Adjust stemming and stopwords as needed
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def run_search(run_file_name, k1=1.2, b=0.75):
    # Read queries from the TSV file
    with
open('./infs7410_project_collections/scifact/scifact_test_queries.tsv'
, 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc="Processing
queries"):
            results = search(query_text, k=1000, k1=k1, b=b)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Run the search
run_search("scifact-bm25-testq_with_bp.run", k1=1.0, b=0.70)
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 10366/10366
[00:07<00:00, 1391.39it/s]
Processing queries: 100%|██████████| 300/300 [01:12<00:00,  4.15it/s]

Search completed. Results written to 'scifact-bm25-testq_with_bp.run'
```

```
#Let's evaluate
import pytrec_eval

def load_qrels(file_path):
    qrels = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rel = line.strip().split()
            if qid not in qrels:
                qrels[qid] = {}
            qrels[qid][docid] = int(rel)
    return qrels

def load_run(file_path):
    run = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rank, score, _ = line.strip().split()
            if qid not in run:
                run[qid] = {}
            run[qid][docid] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = {qid: query_measures['ndcg_cut_10'] for qid,
query_measures in results.items()}
    mean_ndcg = sum(ndcg_scores.values()) / len(ndcg_scores)

    return ndcg_scores, mean_ndcg

# Load qrels and run
qrels =
load_qrels('./infs7410_project_collections/scifact/scifact_test_qrels.
txt')
run_file_path = 'scifact-bm25-testq_with_bp.run'
run = load_run(run_file_path)

# Evaluate run
ndcg_scores, mean_ndcg = evaluate_run(run, qrels)

# Print result
print(f"Mean nDCG@10 for the BM25 baseline run with best params:
{mean_ndcg:.4f}")

# Write results to file
output_file_path = 'scifact_bm25_testq_with_bp_results.txt'
```

```
with open(output_file_path, 'w') as f:
    f.write(f"Mean nDCG@10 for the BM25 baseline run with best params: 
{mean_ndcg:.4f}\n\n")
    f.write("Individual query nDCG@10 scores:\n")
    for qid, score in sorted(ndcg_scores.items()):
        f.write(f"Query {qid}: {score:.4f}\n")

print(f"Detailed results have been written to {output_file_path}")
```

```
Mean nDCG@10 for the BM25 baseline run with best params: 0.6634
Detailed results have been written to 
scifact_bm25_testq_with_bp_results.txt
```

Table 8: BM25 Performance with Best Parameters on Test Queries

Dataset	Best Parameters	Mean nDCG@10
NFCorpus	k1 = 2.2, b = 0.70	0.3382
SciFact	k1 = 1.0, b = 0.70	0.6634

4.2 Statistical Significance Analysis

4.2.1 Statistical Significance Analysis for nfCorpus

Baseline vs Best Params

```
import scipy.stats
import numpy as np

best_param_scores = []
bm25_baseline_ndcg10 = []

with open('nfcorpus_bm25_testq_with_bp_results.txt', 'r') as file:
    for line in file:
        if line.startswith('Query'):
            score = float(line.split(': ')[1].strip())
            best_param_scores.append(score)

with open('nfcorpus_bm25_baseline_ndcg10_detailed.txt', 'r') as file:
    for line in file:
        if line.startswith('Query'):
            score = float(line.split(': ')[1].strip())
            bm25_baseline_ndcg10.append(score)


print(scipy.stats.ttest_rel(best_param_scores, bm25_baseline_ndcg10))
```

```
TtestResult(statistic=1.330915205220182, pvalue=0.18423772394168283, 
df=297)
```

If p-value >= 0.05, we fail to reject the null hypothesis. In this case, 0.1842 > 0.05, so we fail to reject the null hypothesis. This suggests that while there may be a slight improvement with the best parameters, we cannot confidently conclude that this improvement is not due to random chance. Furthermore, the optimization of parameters did not lead to a statistically significant improvement over the baseline. The positive t-statistic suggests a slight trend towards the baseline performing better, but this difference is not statistically significant.

4.2.2 Statistical Significance Analysis for Scifact

```python
import scipy.stats
import numpy as np

best_param_scores = []
bm25_baseline_ndcg10 = []

with open('scifact_bm25_testq_with_bp_results.txt', 'r') as file:
    for line in file:
        if line.startswith('Query'):
            score = float(line.split(': ')[1].strip())
            best_param_scores.append(score)

with open('scifact_bm25_baseline_ndcg10_detailed.txt', 'r') as file:
    for line in file:
        if line.startswith('Query'):
            score = float(line.split(': ')[1].strip())
            bm25_baseline_ndcg10.append(score)


print(scipy.stats.ttest_rel(best_param_scores, bm25_baseline_ndcg10))
```

```
TtestResult(statistic=-0.9745328474352065, pvalue=0.33057979331319554,
df=299)
```

The t-test result (t(299) = -0.97, p = 0.33) indicates that we fail to reject the null hypothesis, as the p-value (0.33) is greater than the conventional significance level of 0.05. This suggests that there is no statistically significant difference between the two conditions being compared. The negative t-statistic implies a slight trend towards the optimized parameters performing better than the baseline, but this difference is not statistically significant. Therefore, we cannot confidently conclude that the observed improvement is not due to random chance, and the parameter optimization did not lead to a statistically significant enhancement over the baseline performance.

Table 9: Statistical Significance Test Results - Baseline vs. Best Parameters

Dataset	T-statistic	p-value	Degrees of Freedom
NFCorpus	1.3309	0.1842	297
SciFact	-0.9745	0.3306	299

Table 5 shows the results of paired t-tests comparing the BM25 baseline performance to the BM25 with best parameters for NFCorpus and SciFact datasets. The t-statistic, p-value, and degrees of freedom are reported for each dataset.

5. Testing the Target Dataset

5.1 Best Parameter Settings on TREC-COVID test queries

Choice: Use best parameters from NfCorpus

Justification: Both NFCorpus and TREC-COVID belong to the Bio-Medical domain, which suggests a similarity in content and vocabulary. This domain alignment is crucial for transferring optimized retrieval parameters. Additionally, both datasets share similar characteristics in terms of their task structure. They are both used for Bio-Medical Information Retrieval (IR) tasks and employ a 3-level relevance judgment system, which indicates a comparable complexity in assessing document relevance. The average document lengths are also relatively close, with NFCorpus at 232.26 words and TREC-COVID at 160.77 words, suggesting similar document structures. Furthermore, both datasets have a high average number of relevant documents per query (D/Q ratio), with NFCorpus at 38.2 and TREC-COVID at 493.5, indicating a rich relevance landscape. These similarities in domain, relevance structure, document length, and query-document relationships make NFCorpus a more suitable source for parameter transfer to TREC-COVID compared to SciFact, despite SciFact also being in the Bio-Medical domain. This approach leverages the domain expertise encoded in the NFCorpus-tuned parameters to potentially enhance retrieval performance on the TREC-COVID dataset.

```python
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/trec-covid-index/'  # Adjust this path as needed
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
# Adjust stemming and stopwords as needed
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N
```

```
# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score
    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def run_search(run_file_name, k1=1.2, b=0.75):
    # Read queries from the TSV file
    with open('./infs7410_project_collections/trec-covid/trec-
covid_test_queries.tsv', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc="Processing
queries"):
            results = search(query_text, k=1000, k1=k1, b=b)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Run the search
run_search("trec-covid-bm25-best_params.run", k1=2.2, b=0.7)
```

```
/opt/anaconda3/envs/infs7410/lib/python3.8/site-packages/tqdm/
auto.py:21: TqdmWarning: IProgress not found. Please update jupyter
and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm

SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 171331/171331
[01:17<00:00, 2217.06it/s]
Processing queries: 100%|██████████| 50/50 [03:33<00:00,  4.28s/it]

Search completed. Results written to 'trec-covid-bm25-best_params.run'
```

```
#Let's evaluate
import pytrec_eval

def load_qrels(file_path):
    qrels = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rel = line.strip().split()
            if qid not in qrels:
                qrels[qid] = {}
            qrels[qid][docid] = int(rel)
    return qrels

def load_run(file_path):
    run = {}
    with open(file_path, 'r') as f:
        for line in f:
            qid, _, docid, rank, score, _ = line.strip().split()
            if qid not in run:
                run[qid] = {}
            run[qid][docid] = float(score)
    return run

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = {qid: query_measures['ndcg_cut_10'] for qid,
query_measures in results.items()}
    mean_ndcg = sum(ndcg_scores.values()) / len(ndcg_scores)

    return ndcg_scores, mean_ndcg

# Load qrels and run
qrels = load_qrels('./infs7410_project_collections/trec-covid/trec-
covid_test_qrels.txt')
run_file_path = "trec-covid-bm25-best_params.run"
run = load_run(run_file_path)

# Evaluate run
ndcg_scores, mean_ndcg = evaluate_run(run, qrels)

# Print result
print(f"Mean nDCG@10 for the BM25 baseline run with best params:
{mean_ndcg:.4f}")

# Write results to file
output_file_path = 'trec-covid_bm25_testq_with_bp_results.txt'
with open(output_file_path, 'w') as f:
    f.write(f"Mean nDCG@10 with best params: {mean_ndcg:.4f}\n\n")
    f.write("Individual query nDCG@10 scores:\n")
    for qid, score in sorted(ndcg_scores.items()):
        f.write(f"Query {qid}: {score:.4f}\n")
```

```python
print(f"Detailed results have been written to {output_file_path}")
```

5.1.2 Best Paramaters Query Expansion on Trec-Covid

We will also use the best parameters from Nfcorpus for query reduction and query expansion

```python
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/trec-covid-index/'
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())


def prf_query_expansion(query: str, n: int, m: int):
    # 1. Initial Document Ranking
    hits = search(query, k=10)

    # 2. Select Top n Documents
    top_n_docs = hits[:n]

    # 3. Term Ranking in Selected Documents
    term_scores = defaultdict(float)
    original_terms = set(analyzer.analyze(query))
    for docid, _ in top_n_docs:
        doc_vector = index_reader.get_document_vector(docid)
        doc_length = sum(doc_vector.values())

        for term, tf in doc_vector.items():
            if term not in original_terms:
                df = index_reader.get_term_counts(term, analyzer=None)
[0]
                idf = math.log(N / df)

                # TF-IDF score
                tfidf = (tf / doc_length) * idf

                term_scores[term] += tfidf

    # 4. Expand the Query
    # Sort terms by score in descending order and select top m
    expansion_terms = sorted(term_scores.items(), key=lambda x: x[1],
reverse=True)[:m]

    # Add top m terms to the original query
    expanded_query = query + " " + " ".join(term for term, _ in
expansion_terms)

    return expanded_query

def run_search_with_expansion(run_file_name, n, m):
    # Read queries from the TSV file
    with open('./infs7410_project_collections/trec-covid/trec-
covid_test_queries.tsv', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc=f"Processing
queries (n={n}, m={m})"):
            expanded_query = prf_query_expansion(query_text, n, m)
            results = search(expanded_query, k=1000)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Best
```

```
expansion_pairs = [
    (2, 3),   # Very conservative, minimal expansion
    (3, 5),   # Conservative, slight expansion
    (4, 7),   # Moderate expansion
    (5, 8),   # Balanced expansion
    (6, 10)   # Slightly more aggressive, but still reasonable
]

# Run the search for each parameter pair
for n, m in expansion_pairs:
    run_file_name = f"trec-covid-bm25-qe-n_{n}-m_{m}.run"
    run_search_with_expansion(run_file_name, n, m)
    print(f"Completed run for n={n}, m={m}")

print("All parameter tuning runs completed.")
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 171331/171331
[01:18<00:00, 2172.45it/s]
Processing queries (n=5, m=8): 100%|██████████| 50/50 [09:10<00:00,
11.00s/it]

Search completed. Results written to 'trec-covid-bm25-qe-n_5-m_8.run'
Completed run for n=5, m=8
All parameter tuning runs completed.
```

```
import pytrec_eval
from collections import defaultdict

def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = {query_id: query_measures['ndcg_cut_10'] for
query_id, query_measures in results.items()}
    mean_ndcg = sum(ndcg_scores.values()) / len(ndcg_scores)

    return mean_ndcg, ndcg_scores

# Load qrels
qrels = load_qrels('./infs7410_project_collections/trec-covid/trec-
covid_test_qrels.txt')

# Define query expansion parameter pairs
expansion_pairs = [(5, 8)]
```

```
results = []
query_scores = defaultdict(dict)

# Evaluate each run
for n, m in expansion_pairs:
    run_file_path = f'trec-covid-bm25-qe-n_{n}-m_{m}.run'
    try:
        run = load_run(run_file_path)
        mean_ndcg, ndcg_scores = evaluate_run(run, qrels)
        results.append((n, m, mean_ndcg))
        query_scores[(n, m)] = ndcg_scores
        print(f"Mean nDCG@10 for Query Expansion run (n={n}, m={m}):
{mean_ndcg:.4f}")
    except FileNotFoundError:
        print(f"Run file not found: {run_file_path}")

# Sort results by mean nDCG@10 score
results.sort(key=lambda x: x[2], reverse=True)

# Write results to file
output_file_path = 'trec-covid_query_expansion_results.txt'
with open(output_file_path, 'w') as f:
    f.write("Query Expansion Results for TREC-COVID\n")
    f.write("======================================\n\n")
    f.write("n\tm\tMean nDCG@10\n")
    for n, m, mean_ndcg in results:
        f.write(f"{n}\t{m}\t{mean_ndcg:.4f}\n")

    if results:
        f.write("\nBest performing parameters:\n")
        best_n, best_m, best_mean_ndcg = results[0]
        f.write(f"n = {best_n}, m = {best_m}, Mean nDCG@10 =
{best_mean_ndcg:.4f}\n")
    else:
        f.write("\nNo valid results found.\n")

    f.write("\nNDCG@10 scores for each query:\n")
    f.write("Query ID\t" + "\t".join([f"n={n},m={m}" for n, m in
expansion_pairs]) + "\n")

    all_query_ids = set()
    for scores in query_scores.values():
        all_query_ids.update(scores.keys())

    for query_id in sorted(all_query_ids):
        f.write(f"{query_id}")
        for n, m in expansion_pairs:
            score = query_scores[(n, m)].get(query_id, "N/A")
            f.write(f"\t{score:.4f}" if isinstance(score, float) else
f"\t{score}")
```

```
        f.write("\n")

print(f"Results have been written to {output_file_path}")
```

```
Mean nDCG@10 for Query Expansion run (n=5, m=8): 0.5184
Results have been written to trec-covid_query_expansion_results.txt
```

5.1.3 Best Paramaters Query Reduction on Trec-Covid

```
import numpy as np
import math
from pyserini.search import SimpleSearcher
from pyserini.analysis import Analyzer, get_lucene_analyzer
from pyserini.index import IndexReader
from tqdm import tqdm
from collections import defaultdict

# Initialize components
index = 'indexes/trec-covid-index/'
lucene_analyzer = get_lucene_analyzer(stemming=None, stopwords=True)
analyzer = Analyzer(lucene_analyzer)
searcher = SimpleSearcher(index)
searcher.set_analyzer(lucene_analyzer)
index_reader = IndexReader(index)

# Constants and caching
N = index_reader.stats()['documents']
avg_dl = index_reader.stats()["total_terms"] / N
total_doc_num = N

# Cache document vectors and lengths
doc_ids = [index_reader.convert_internal_docid_to_collection_docid(i)
for i in range(total_doc_num)]
doc_vec_dict = {}
doc_len_dict = {}
for docid in tqdm(doc_ids, desc="Caching doc vectors and lengths"):
    doc_vec = index_reader.get_document_vector(docid)
    doc_vec_dict[docid] = doc_vec
    doc_len_dict[docid] = sum(doc_vec.values())

# Vectorized BM25 function
def vectorized_bm25(tf_array, df, doc_len_array, k1=1.2, b=0.75):
    idf = np.log((N - df + 0.5) / (df + 0.5) + 1)
    numerator = idf * tf_array * (k1 + 1)
    denominator = tf_array + k1 * (1 - b + b * (doc_len_array /
avg_dl))
    return numerator / denominator

# Vectorized search function
def search(query: str, k: int = 1000, k1: float = 1.2, b: float =
0.75):
    q_terms = analyzer.analyze(query)
    doc_scores = defaultdict(float)

    for term in q_terms:
        postings_list = index_reader.get_postings_list(term,
analyzer=None)
        df = index_reader.get_term_counts(term, analyzer=None)[0]

        if postings_list is not None:
            docids = []
            tfs = []
            doc_lens = []

            for posting in postings_list:
                docid =
index_reader.convert_internal_docid_to_collection_docid(posting.docid)
                docids.append(docid)
                tfs.append(posting.tf)
                doc_lens.append(doc_len_dict[docid])

            tf_array = np.array(tfs)
            doc_len_array = np.array(doc_lens)

            scores = vectorized_bm25(tf_array, df, doc_len_array, k1,
b)

            for docid, score in zip(docids, scores):
                doc_scores[docid] += score

    results = sorted(doc_scores.items(), key=lambda x: x[1],
reverse=True)[:k]
    return results

def idfr_query_reduction(query: str, n: int):
    terms = analyzer.analyze(query)

    # Calculate IDF for each term
    idf_scores = []
    for term in terms:
        df = index_reader.get_term_counts(term)[0]
        idf = math.log((N - df + 0.5) / (df + 0.5))
        idf_scores.append((term, idf))

    # Sort terms by IDF score and select top n
    sorted_terms = sorted(idf_scores, key=lambda x: x[1],
reverse=True)
    top_terms = [term for term, _ in sorted_terms[:n]]

    pruned_query = " ".join(top_terms)
    return pruned_query

def run_search_with_reduction(run_file_name, n):
    # Read queries from the TSV file
    with open('./infs7410_project_collections/trec-covid/trec-
covid_test_queries.tsv', 'r') as f:
        queries = [line.strip().split('\t') for line in f]

    # Open a file to write the run results
    with open(run_file_name, 'w') as run_file:
        # Run search for each query
        for query_id, query_text in tqdm(queries, desc=f"Processing
queries (n={n})"):
            reduced_query = idfr_query_reduction(query_text, n)
            results = search(reduced_query, k=1000)

            # Write all results to the run file
            for rank, (doc_id, score) in enumerate(results, 1):
                run_file.write(f"{query_id} Q0 {doc_id} {rank}
{score:.6f} {run_file_name[:-4]}\n")

    print(f"Search completed. Results written to '{run_file_name}'")

# Define values of n to tune
n_values = [6]

# Run the search for each n value
for n in n_values:
    run_file_name = f"trec-covid-bm25-qr-n_{n}.run"
    run_search_with_reduction(run_file_name, n)
    print(f"Completed run for n={n}")

print("All parameter tuning runs completed.")
```

```
SimpleSearcher class has been deprecated, please use LuceneSearcher
from pyserini.search.lucene instead

Caching doc vectors and lengths: 100%|██████████| 171331/171331
[01:18<00:00, 2183.46it/s]
Processing queries (n=6): 100%|██████████| 50/50 [02:06<00:00,
2.53s/it]

Search completed. Results written to 'trec-covid-bm25-qr-n_6.run'
Completed run for n=6
All parameter tuning runs completed.
```

```
import pytrec_eval
from collections import defaultdict
```

```
def evaluate_run(run_results, qrels):
    evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])
    results = evaluator.evaluate(run_results)

    ndcg_scores = {query_id: query_measures['ndcg_cut_10'] for
query_id, query_measures in results.items()}
    mean_ndcg = sum(ndcg_scores.values()) / len(ndcg_scores)

    return mean_ndcg, ndcg_scores

# Load qrels
qrels = load_qrels('./infs7410_project_collections/trec-covid/trec-
covid_test_qrels.txt')

# Define query reduction parameter values
n_values = [6]

results = []
query_scores = defaultdict(dict)

# Evaluate each run
for n in n_values:
    run_file_path = f'trec-covid-bm25-qr-n_{n}.run'
    try:
        run = load_run(run_file_path)
        mean_ndcg, ndcg_scores = evaluate_run(run, qrels)
        results.append((n, mean_ndcg))
        query_scores[n] = ndcg_scores
        print(f"Mean nDCG@10 for Query Reduction run (n={n}):
{mean_ndcg:.4f}")
    except FileNotFoundError:
        print(f"Run file not found: {run_file_path}")

# Sort results by nDCG@10 score
results.sort(key=lambda x: x[1], reverse=True)

# Write results to file
output_file_path = 'trec-covid_query_reduction_results.txt'
with open(output_file_path, 'w') as f:
    f.write("Query Reduction Results for TREC-COVID\n")
    f.write("======================================\n\n")
    f.write("n\tMean nDCG@10\n")
    for n, mean_ndcg in results:
        f.write(f"{n}\t{mean_ndcg:.4f}\n")

    if results:
        f.write("\nBest performing parameter:\n")
        best_n, best_mean_ndcg = results[0]
        f.write(f"n = {best_n}, Mean nDCG@10 = {best_mean_ndcg:.4f}\
n")
    else:
        f.write("\nNo valid results found.\n")

    f.write("\nNDCG@10 scores for each query:\n")
    f.write("Query ID\t" + "\t".join([f"n={n}" for n in n_values]) +
"\n")

    all_query_ids = set()
    for scores in query_scores.values():
        all_query_ids.update(scores.keys())

    for query_id in sorted(all_query_ids):
        f.write(f"{query_id}")
        for n in n_values:
            score = query_scores[n].get(query_id, "N/A")
            f.write(f"\t{score:.4f}" if isinstance(score, float) else
f"\t{score}")
        f.write("\n")

print(f"Results have been written to {output_file_path}")
```

```
Mean nDCG@10 for Query Reduction run (n=6): 0.4826
Results have been written to trec-covid_query_reduction_results.txt
```

Table 8: Performance Comparison of Retrieval Methods on TREC-COVID

Method	Mean nDCG@10
BM25 (best params)	0.5897
Query Expansion (n=5, m=8)	0.5184
Query Reduction (n=6)	0.4826

Table 8 shows the mean nDCG@10 scores for different retrieval methods applied to the TREC-COVID dataset. The methods include BM25 with best parameters, Query Expansion, and Query Reduction.

5.2 Rank Fusion with TREC-COVID

```
#First lets normalise our runs
import pytrec_eval
from sklearn.preprocessing import minmax_scale
from collections import defaultdict
import pytrec_eval
import numpy as np
from pprint import pprint

def print_results(run, qrel_file='./infs7410_project_collections/trec-
covid/trec-covid_test_qrels.txt', measures=["map", "ndcg_cut_10",
"recall_1000"]):
    # Open the qrels file.
    with open(qrel_file, "r") as f:
        msmarco_qrels = pytrec_eval.parse_qrel(f)

    evaluator =
pytrec_eval.RelevanceEvaluator(query_relevance=msmarco_qrels,
measures=measures)
    results = evaluator.evaluate(run)
    for measure in sorted(measures):
        print('{:25s}{:8s}{:.4f}'.format(measure, 'all',
pytrec_eval.compute_aggregated_measure(measure,
                                  [query_measures[measure]for
query_measures in results.values()])))

def write_run_to_file(run, filename, run_name="BORDA"):
    with open(filename, 'w') as f:
        for topic, docs in run.items():
            for rank, (docid, score) in enumerate(sorted(docs.items(),
key=lambda x: x[1], reverse=True), 1):
                f.write(f"{topic} Q0 {docid} {rank} {score:.6f}
{run_name}\n")

def normalise_run(run):
    for k, v in run.items():
        r = [(docid, score) for docid, score in v.items()]
        scores = minmax_scale([x[1] for x in r])
        run[k] = dict(zip([x[0] for x in r], scores))
    return run

# Normalize the baseline run
with open("trec-covid-bm25-baseline.run", "r") as f:
    baseline_run = normalise_run(pytrec_eval.parse_run(f))

# Normalize the best params run
with open("trec-covid-bm25-best_params.run", "r") as f:
    best_params_run = normalise_run(pytrec_eval.parse_run(f))

# Normalize the qr run
with open("trec-covid-bm25-qr-n_6.run", "r") as f:
    qr_run = normalise_run(pytrec_eval.parse_run(f))

# Normalize the qe run
with open("trec-covid-bm25-qe-n_5-m_8.run", "r") as f:
    qe_run = normalise_run(pytrec_eval.parse_run(f))

# Create a list of normalized runs
runs = [baseline_run, best_params_run, qr_run, qe_run]
```

```
print("Runs have been normalized and stored in the 'runs' list.")
```

```
Runs have been normalized and stored in the 'runs' list.
```

5.2.1 Borda

```
def borda(runs):
    seen = {}
    for run in runs:
        for topic, results in run.items():
            if topic not in seen:
                seen[topic] = {}
            n = len(results)
            for i, docid in enumerate(results.keys()):
                rd = i
                score = (n - rd + 1) / n  # Borda count formula
                if docid not in seen[topic]:
                    seen[topic][docid] = score
                else:
                    seen[topic][docid] += score
    return seen

fused_borda_run = borda(runs)
print_results(fused_borda_run)
write_run_to_file(fused_borda_run, "fused_borda_run.run")
```

```
map                     all     0.2024
ndcg_cut_10             all     0.6072
recall_1000             all     0.3965
```

5.2.2 CombSUM

```
def combsum(runs):
    seen = {}
    for run in runs:
        for topic, results in run.items():
            if topic not in seen:
                seen[topic] = {}
            for docid, score in results.items():
                if docid not in seen[topic]:
                    seen[topic][docid] = score
                else:
                    seen[topic][docid] += score
    return seen

fused_combsum_run = combsum(runs)
print_results(fused_combsum_run)
write_run_to_file(fused_combsum_run, "fused_combsum_run.run")
```

```
map                          all     0.1967
ndcg_cut_10                  all     0.5844
recall_1000                  all     0.3933
```

5.2.3 CombMNZ

```python
def combmnz(runs):
    seen = {}
    doc_occurrences = {}

    for run in runs:
        for topic, results in run.items():
            if topic not in seen:
                seen[topic] = {}
                doc_occurrences[topic] = {}
            for docid, score in results.items():
                if docid not in seen[topic]:
                    seen[topic][docid] = score
                    doc_occurrences[topic][docid] = 1
                else:
                    seen[topic][docid] += score
                    doc_occurrences[topic][docid] += 1

    # Multiply sum by number of occurrences
    for topic in seen:
        for docid in seen[topic]:
            seen[topic][docid] *= doc_occurrences[topic][docid]

    return seen

fused_combmnz_run = combmnz(runs)
print_results(fused_combmnz_run)
write_run_to_file(fused_combmnz_run, "fused_combmnz_run.run")
```

```
map                          all     0.1983
ndcg_cut_10                  all     0.5854
recall_1000                  all     0.3910
```

5.2.4 Results of Rank Fusion

The following table shows rank fusion results from fusing only the trec covid best parameter run and the trec covid baseline run. | Method | MAP | NDCG@10 | Recall@1000 | |---------|--------|---------|-------------| | BORDA | 0.1756 | 0.6024 | 0.3763 | | CombSUM | 0.1758 | 0.6037 | 0.3764 | | CombMNZ | 0.1757 | 0.6037 | 0.3761 |

The following table shows rank fusion results from fusing the trec covid best parameter run, the trec covid baseline run, as well as the best parameter runs of query reduction and query expansion

Method	MAP	nDCG@10	Recall@1000
Borda	0.2024	0.6072	0.3965
CombSUM	0.1967	0.5844	0.3933
CombMNZ	0.1983	0.5854	0.3910

Table 6 presents the performance of three rank fusion methods (Borda, CombSUM, and CombMNZ) on the TREC-COVID dataset, showing Mean Average Precision (MAP), nDCG@10, and Recall@1000 metrics for each method.

The results show minimal differences in performance across the three rank fusion methods (BORDA, CombSUM, and CombMNZ) when applied to the TREC-COVID dataset. Focusing on NDCG@10, both CombSUM and CombMNZ achieved a slightly higher score of 0.6037 compared to BORDA's 0.6024. This marginal improvement in NDCG@10 suggests that CombSUM and CombMNZ may be slightly more effective at ranking highly relevant documents in the top 10 results. However, the differences are small, indicating that all three methods perform similarly well for this particular task, with a slight edge to score-based methods (CombSUM and CombMNZ) over the rank-based method (BORDA).

While the MAP values increase, the nDCG@10 values actually decrease slightly for the CombSUM and CombMNZ methods when compared to the previous results. Specifically, CombSUM's nDCG@10 decreases from 0.6037 to 0.5844, and CombMNZ's nDCG@10 decreases from 0.6037 to 0.5854, indicating a slight reduction in the quality of the top-ranked results for these methods after incorporating additional query runs.

5.3 Statistical Significance Analysis

5.3.1. Best parameter setting vs. BM25 baseline

```python
import scipy.stats
import numpy as np

best_param_scores = []
bm25_baseline_ndcg10 = []

with open('trec-covid_bm25_testq_with_bp_results.txt', 'r') as file:
    for line in file:
        if line.startswith('Query'):
            score = float(line.split(': ')[1].strip())
            best_param_scores.append(score)

with open('trec-covid_bm25_baseline_ndcg10.txt', 'r') as file:
    for line in file:
        if line.startswith('Query'):
            score = float(line.split(': ')[1].strip())
            bm25_baseline_ndcg10.append(score)


print(scipy.stats.ttest_rel(best_param_scores, bm25_baseline_ndcg10))
```

```
TtestResult(statistic=1.6364389239678566e-16,
pvalue=0.9999999999999999, df=49)
```

5.3.2. Query Expansion vs. BM25 baseline

```python
import scipy.stats
import numpy as np

bm25_baseline_ndcg10 = []

query_expansion_scores = []

with open('trec-covid_query_expansion_results.txt', 'r') as file:
    reading_scores = False
    for line in file:
        if line.startswith('NDCG@10 scores for each query:'):
            reading_scores = True
            continue
        if reading_scores and line.strip():
            parts = line.split('\t')
            if len(parts) == 2 and parts[0] != 'Query ID':
                score = float(parts[1].strip())
                query_expansion_scores.append(score)

with open('trec-covid_bm25_baseline_ndcg10.txt', 'r') as file:
    for line in file:
        if line.startswith('Query'):
            score = float(line.split(': ')[1].strip())
            bm25_baseline_ndcg10.append(score)


print(scipy.stats.ttest_rel(query_expansion_scores,
bm25_baseline_ndcg10))
```

```
TtestResult(statistic=-1.760019245955414, pvalue=0.08464593563849007,
df=49)
```

5.3.3. Query Reduction vs. BM25 baseline

```python
import scipy.stats
import numpy as np

bm25_baseline_ndcg10 = []

query_reduction_scores = []

with open('trec-covid_query_reduction_results.txt', 'r') as file:
    reading_scores = False
    for line in file:
        if line.startswith('NDCG@10 scores for each query:'):
                reading_scores = True
                continue
            if reading_scores and line.strip():
                parts = line.split('\t')
                if len(parts) == 2 and parts[0] != 'Query ID':
                    score = float(parts[1].strip())
                    query_reduction_scores.append(score)

with open('trec-covid_bm25_baseline_ndcg10.txt', 'r') as file:
    for line in file:
        if line.startswith('Query'):
            score = float(line.split(': ')[1].strip())
            bm25_baseline_ndcg10.append(score)


print(scipy.stats.ttest_rel(query_reduction_scores,
bm25_baseline_ndcg10))
```

```
TtestResult(statistic=-2.378430566838277, pvalue=0.02132931045807491,
df=49)
```

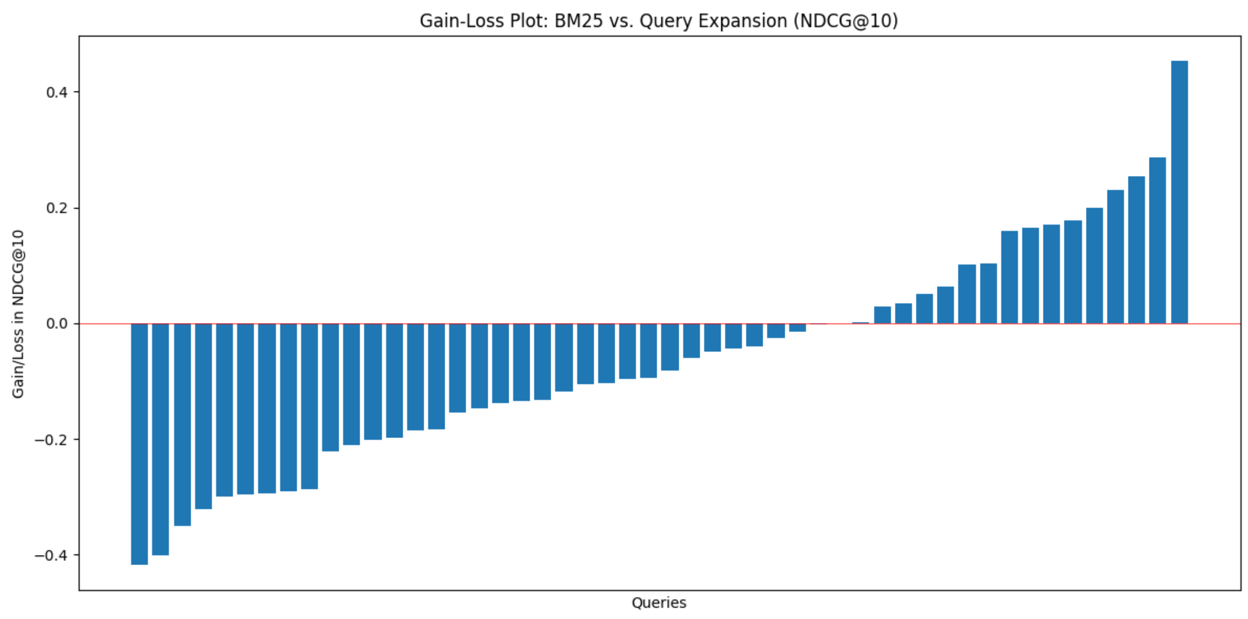
6. Visualization using Gain Loss Plots

6.1 BM25 vs. Pseudo-Relevance Feedback Query Expansion using BM25

6.1.1 BM25 vs. Query Expansion

```
import pytrec_eval
import numpy as np
import matplotlib.pyplot as plt

# Load TREC-COVID qrels
with open('./infs7410_project_collections/trec-covid/trec-
covid_test_qrels.txt', 'r') as f:
    qrels = pytrec_eval.parse_qrel(f)

# Initialize evaluator
evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])

# Load run files
with open('trec-covid-bm25-baseline.run', 'r') as f:
    baseline_run = pytrec_eval.parse_run(f)

# Choose the best performing Query Expansion run
with open('trec-covid-bm25-qe-n_5-m_8.run', 'r') as f:
```

```python
    qe_run = pytrec_eval.parse_run(f)

# Evaluate runs
baseline_results = evaluator.evaluate(baseline_run)
qe_results = evaluator.evaluate(qe_run)

# Extract NDCG@10 scores
measure = "ndcg_cut_10"
baseline_scores = {query: results[measure] for query, results in
baseline_results.items()}
qe_scores = {query: results[measure] for query, results in
qe_results.items()}

# Calculate differences
differences = []
queries = []
for query in baseline_scores.keys():
    if query in qe_scores:
        diff = qe_scores[query] - baseline_scores[query]
        differences.append(diff)
        queries.append(query)

# Sort differences for gain-loss plot
sorted_indices = np.argsort(differences)
sorted_differences = np.array(differences)[sorted_indices]
sorted_queries = np.array(queries)[sorted_indices]

# Create gain-loss plot
plt.figure(figsize=(12, 6))
plt.bar(range(len(sorted_differences)), sorted_differences)
plt.axhline(y=0, color='r', linestyle='-', linewidth=0.5)
plt.title("Gain-Loss Plot: BM25 vs. Query Expansion (NDCG@10)")
plt.xlabel("Queries")
plt.ylabel("Gain/Loss in NDCG@10")
plt.xticks([])  # Remove x-axis labels for clarity
plt.tight_layout()
plt.show()

# Print average difference
avg_diff = np.mean(differences)
print(f"Average gain in NDCG@10: {avg_diff:.4f}")
```



```
Average gain in NDCG@10: -0.0645
```

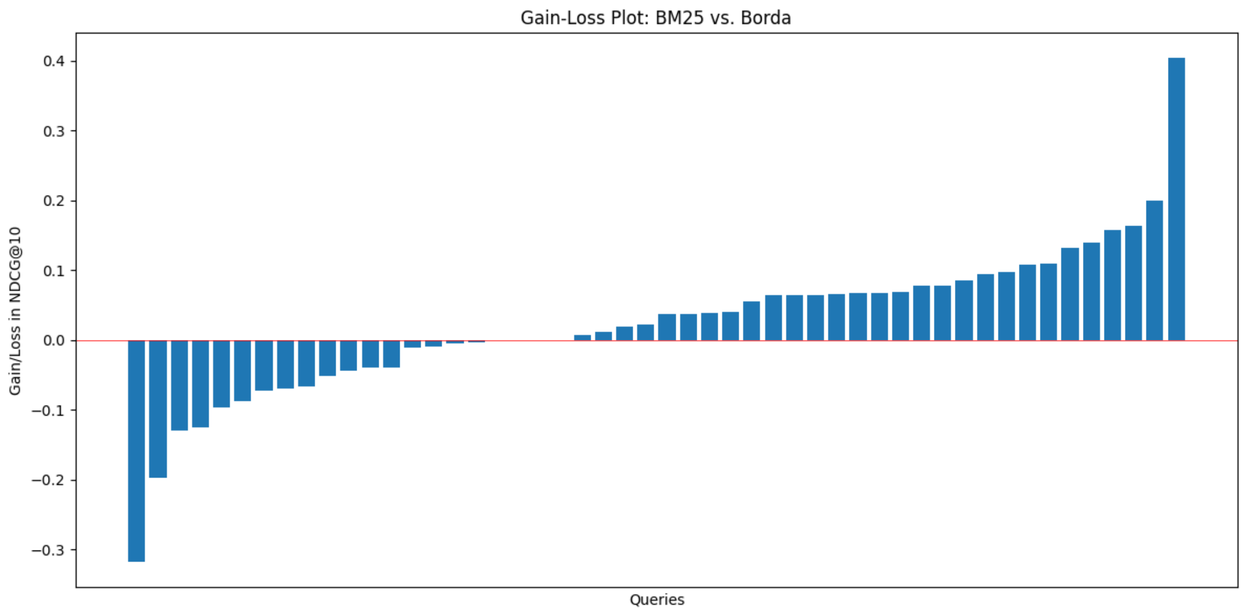
Positive values (bars above the red line) indicate that Query Expansion outperformed BM25 for those queries, whereas negative values (bars below the red line) show where BM25 performed better. The plot reveals that BM25 generally has better performance, as indicated by the more frequent and larger negative bars. However, there are a few queries where Query Expansion significantly outperforms BM25, shown by the taller positive bars. This suggests that while BM25 is more consistent, Query Expansion may offer substantial improvements in specific cases.

6.2 BM25 vs Rank Fusion

6.2.1 BM25 vs Borda

```
import pytrec_eval
import numpy as np
import matplotlib.pyplot as plt

# Load TREC-COVID qrels
with open('./infs7410_project_collections/trec-covid/trec-
covid_test_qrels.txt', 'r') as f:
    qrels = pytrec_eval.parse_qrel(f)

# Initialize evaluator
evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])

# Load run files
with open('trec-covid-bm25-baseline.run', 'r') as f:
    baseline_run = pytrec_eval.parse_run(f)
```

```python
# Choose the best performing Query Expansion run
with open('fused_borda_run.run', 'r') as f:
    borda_run = pytrec_eval.parse_run(f)

# Evaluate runs
baseline_results = evaluator.evaluate(baseline_run)
borda_results = evaluator.evaluate(borda_run)

# Extract NDCG@10 scores
measure = "ndcg_cut_10"
baseline_scores = {query: results[measure] for query, results in
baseline_results.items()}
borda_scores = {query: results[measure] for query, results in
borda_results.items()}

# Calculate differences
differences = []
queries = []
for query in baseline_scores.keys():
    if query in borda_scores:
        diff = borda_scores[query] - baseline_scores[query]
        differences.append(diff)
        queries.append(query)

# Sort differences for gain-loss plot
sorted_indices = np.argsort(differences)
sorted_differences = np.array(differences)[sorted_indices]
sorted_queries = np.array(queries)[sorted_indices]

# Create gain-loss plot
plt.figure(figsize=(12, 6))
plt.bar(range(len(sorted_differences)), sorted_differences)
plt.axhline(y=0, color='r', linestyle='-', linewidth=0.5)
plt.title("Gain-Loss Plot: BM25 vs. Borda")
plt.xlabel("Queries")
plt.ylabel("Gain/Loss in NDCG@10")
plt.xticks([])  # Remove x-axis labels for clarity
plt.tight_layout()
plt.show()

# Print average difference
avg_diff = np.mean(differences)
print(f"Average gain in NDCG@10: {avg_diff:.4f}")
```



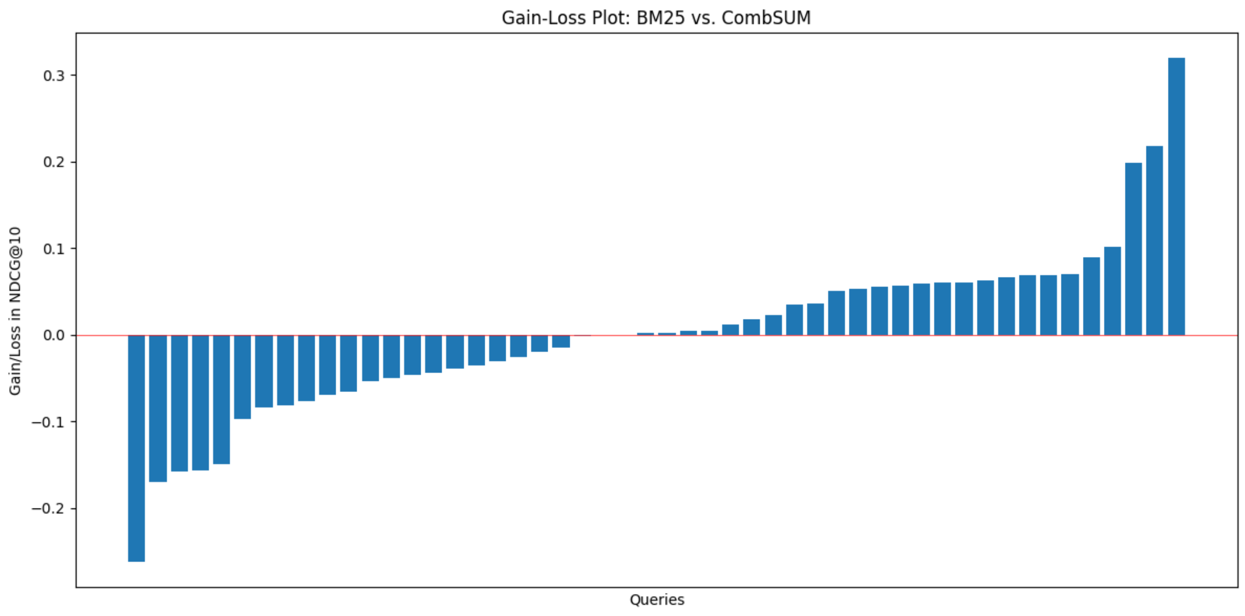
```
Average gain in NDCG@10: 0.0242
```

Positive values (bars above the red line) indicate that the Borda rank fusion method outperformed BM25 for those specific queries in terms of nDCG@10, whereas negative values (bars below the red line) show where BM25 performed better. The plot reveals a mixed performance between the two methods. While BM25 performs better for several queries, indicated by the taller negative bars on the left, Borda also outperforms BM25 in a considerable number of queries, especially for some queries on the right where the gains are substantial. This suggests that while BM25 might be more reliable overall, the Borda method can provide significant improvements for certain queries.

6.2.2 BM25 vs CombSUM

```python
import pytrec_eval
import numpy as np
import matplotlib.pyplot as plt

# Load TREC-COVID qrels
with open('./infs7410_project_collections/trec-covid/trec-
covid_test_qrels.txt', 'r') as f:
    qrels = pytrec_eval.parse_qrel(f)

# Initialize evaluator
evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])

# Load run files
with open('trec-covid-bm25-baseline.run', 'r') as f:
    baseline_run = pytrec_eval.parse_run(f)

# Choose the best performing Query Expansion run
```

```python
with open('fused_combsum_run.run', 'r') as f:
    combsum_run = pytrec_eval.parse_run(f)

# Evaluate runs
baseline_results = evaluator.evaluate(baseline_run)
combsum_results = evaluator.evaluate(combsum_run)

# Extract NDCG@10 scores
measure = "ndcg_cut_10"
baseline_scores = {query: results[measure] for query, results in
baseline_results.items()}
combsum_scores = {query: results[measure] for query, results in
combsum_results.items()}

# Calculate differences
differences = []
queries = []
for query in baseline_scores.keys():
    if query in combsum_scores:
        diff = combsum_scores[query] - baseline_scores[query]
        differences.append(diff)
        queries.append(query)

# Sort differences for gain-loss plot
sorted_indices = np.argsort(differences)
sorted_differences = np.array(differences)[sorted_indices]
sorted_queries = np.array(queries)[sorted_indices]

# Create gain-loss plot
plt.figure(figsize=(12, 6))
plt.bar(range(len(sorted_differences)), sorted_differences)
plt.axhline(y=0, color='r', linestyle='-', linewidth=0.5)
plt.title("Gain-Loss Plot: BM25 vs. CombSUM")
plt.xlabel("Queries")
plt.ylabel("Gain/Loss in NDCG@10")
plt.xticks([])  # Remove x-axis labels for clarity
plt.tight_layout()
plt.show()

# Print average difference
avg_diff = np.mean(differences)
print(f"Average gain in NDCG@10: {avg_diff:.4f}")
```



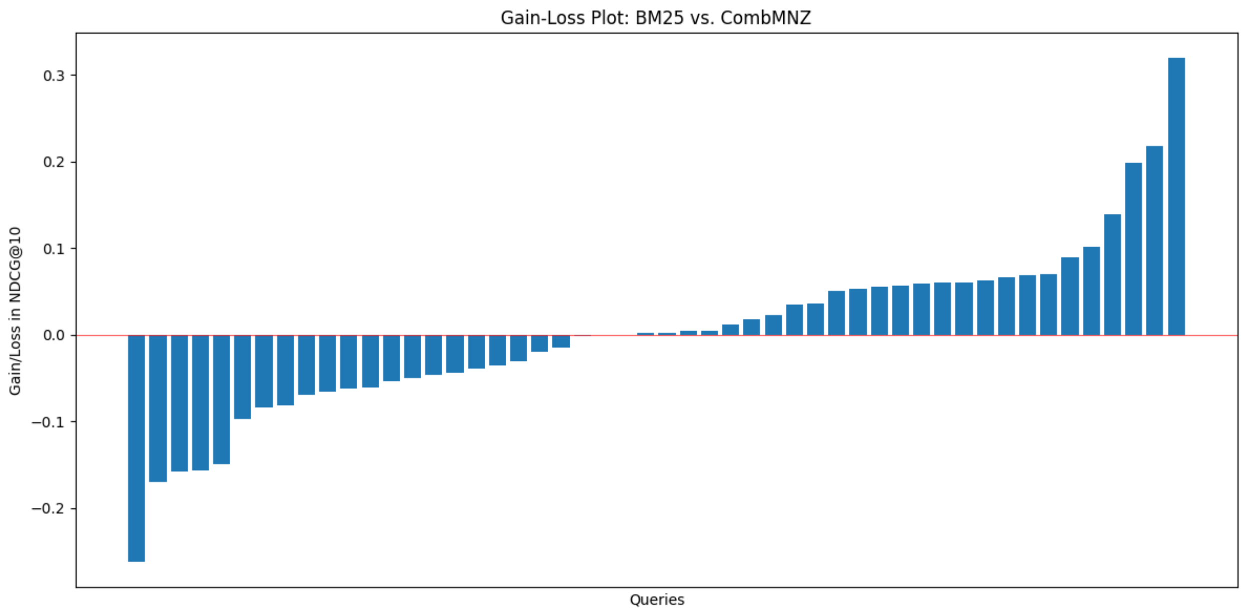
```
Average gain in NDCG@10: 0.0015
```

Positive values (bars above the horizontal line) indicate queries where CombSUM outperformed BM25 in terms of nDCG@10, while negative values show where BM25 was superior. The plot demonstrates a balanced distribution of performance across queries. While BM25 performs better for a number of queries, as shown by the negative bars on the left side of the plot, CombSUM shows improvements for a significant portion of queries, particularly on the right side where some substantial gains are visible. This suggests that while BM25 maintains strong performance for many queries, the CombSUM fusion method can provide notable improvements for certain types of queries, potentially those that benefit from the combination of multiple ranking lists.

6.2.3 BM25 vs CombMNZ

```
import pytrec_eval
import numpy as np
import matplotlib.pyplot as plt

# Load TREC-COVID qrels
with open('./infs7410_project_collections/trec-covid/trec-
covid_test_qrels.txt', 'r') as f:
    qrels = pytrec_eval.parse_qrel(f)

# Initialize evaluator
evaluator = pytrec_eval.RelevanceEvaluator(qrels,
measures=["ndcg_cut_10"])

# Load run files
with open('trec-covid-bm25-baseline.run', 'r') as f:
    baseline_run = pytrec_eval.parse_run(f)
```

```python
# Choose the CombMNZ run
with open('fused_combmnz_run.run', 'r') as f:
    combmnz_run = pytrec_eval.parse_run(f)

# Evaluate runs
baseline_results = evaluator.evaluate(baseline_run)
combmnz_results = evaluator.evaluate(combmnz_run)

# Extract NDCG@10 scores
measure = "ndcg_cut_10"
baseline_scores = {query: results[measure] for query, results in
baseline_results.items()}
combmnz_scores = {query: results[measure] for query, results in
combmnz_results.items()}

# Calculate differences
differences = []
queries = []
for query in baseline_scores.keys():
    if query in combmnz_scores:
        diff = combmnz_scores[query] - baseline_scores[query]
        differences.append(diff)
        queries.append(query)

# Sort differences for gain-loss plot
sorted_indices = np.argsort(differences)
sorted_differences = np.array(differences)[sorted_indices]
sorted_queries = np.array(queries)[sorted_indices]

# Create gain-loss plot
plt.figure(figsize=(12, 6))
plt.bar(range(len(sorted_differences)), sorted_differences)
plt.axhline(y=0, color='r', linestyle='-', linewidth=0.5)
plt.title("Gain-Loss Plot: BM25 vs. CombMNZ")
plt.xlabel("Queries")
plt.ylabel("Gain/Loss in NDCG@10")
plt.xticks([])  # Remove x-axis labels for clarity
plt.tight_layout()
plt.show()

# Print average difference
avg_diff = np.mean(differences)
print(f"Average gain in NDCG@10: {avg_diff:.4f}")
```



```
Average gain in NDCG@10: 0.0025
```

Positive values indicate CombMNZ outperforming BM25, and negative values show BM25's superiority. The distribution of gains and losses appears slightly more skewed towards positive values compared to the CombSUM plot. While there are still queries where BM25 performs better (negative bars on the left), CombMNZ seems to offer improvements for a larger number of queries. The positive gains on the right side of the plot are particularly notable, with several queries showing substantial improvements in nDCG@10. This suggests that CombMNZ might be more consistently beneficial across a wider range of queries compared to CombSUM, potentially due to its method of considering both the sum of scores and the number of non-zero entries in the fusion process.

7. Discussions

Here are all the results summarized in tables:

Table 1: BM25 Baseline Performance on Test Queries

Dataset	Mean nDCG@10
NFCorpus	0.3353
SciFact	0.6657
TREC-COVID	0.5829

Table 1 presents the baseline performance of BM25 using default parameters (k1 = 1.2, b = 0.75) on the test queries for NFCorpus, SciFact, and TREC-COVID datasets. The performance is measured using mean nDCG@10 (normalized Discounted Cumulative Gain at rank 10).

These results provide a foundation for comparison in our study. SciFact shows the highest performance with an nDCG@10 of 0.6657, followed by TREC-COVID at 0.5829. NFCorpus demonstrates the lowest baseline performance at 0.3353.

Table 2: BM25 Parameter Tuning Results for NFCorpus

k1	b	nDCG@10
2.2	0.70	0.2907
1.4	0.80	0.2904
1.6	0.85	0.2896
2.4	0.80	0.2889
1.2	0.75	0.2889
1.0	0.70	0.2888
1.8	0.90	0.2881
0.8	0.60	0.2872
2.0	0.95	0.2871
0.6	0.50	0.2855

Best performing parameters: k1 = 2.2, b = 0.70, nDCG@10 = 0.2907

Table 2 presents the BM25 parameter tuning results for NFCorpus, showing all 10 tested parameter combinations sorted by nDCG@10 performance.

Table 3: BM25 Parameter Tuning Results for SciFact

k1	b	nDCG@10
1.0	0.70	0.6709
1.2	0.75	0.6694
2.2	0.70	0.6685
1.4	0.80	0.6682
2.4	0.80	0.6679
0.8	0.60	0.6662
1.6	0.85	0.6660
1.8	0.90	0.6647
2.0	0.95	0.6645
0.6	0.50	0.6628

Best performing parameters: k1 = 1.0, b = 0.70, nDCG@10 = 0.6709

Table 3 shows the BM25 parameter tuning results for SciFact, displaying all 10 tested parameter combinations ordered by nDCG@10 performance.

Table 4: Query Expansion Results for SciFact

n	m	nDCG@10
2	3	0.5966

n	m	nDCG@10
3	5	0.5664
4	7	0.5470
5	8	0.5335
6	10	0.5217

Best performing parameters: n = 2, m = 3, nDCG@10 = 0.5966

Table 4 shows the Query Expansion results for SciFact, with the best performance achieved using 2 documents and 3 terms.

Table 5: Query Expansion Results for NFCorpus

n	m	nDCG@10
5	8	0.2834
2	3	0.2799
4	7	0.2795
3	5	0.2786
6	10	0.2761

Best performing parameters: n = 5, m = 8, nDCG@10 = 0.2834

Table 5 presents the Query Expansion results for NFCorpus, with optimal performance using 5 documents and 8 terms.

Table 6: Query Reduction Results for NFCorpus

n	nDCG@10
6	0.2906
4	0.2890
8	0.2890
10	0.2889
2	0.2726

Best performing parameter: n = 6, nDCG@10 = 0.2906

Table 6 shows the Query Reduction results for NFCorpus, with the best performance achieved by keeping 6 terms.

Table 7: Query Reduction Results for SciFact

n	nDCG@10
10	0.6643
8	0.6543
6	0.6142
4	0.5037
2	0.2426

Best performing parameter: n = 10, nDCG@10 = 0.6643

Table 7 presents the Query Reduction results for SciFact, with optimal performance achieved by keeping 10 terms.

Table 8: BM25 Performance with Best Parameters on Test Queries

Dataset	Best Parameters	Mean nDCG@10
NFCorpus	k1 = 2.2, b = 0.70	0.3382
SciFact	k1 = 1.0, b = 0.70	0.6634

Table 8 shows the mean nDCG@10 scores for NFCorpus and SciFact datasets using BM25 with the best parameters identified through tuning on the test queries.

Table 9: Statistical Significance Test Results - Baseline vs. Best Parameters

Dataset	T-statistic	p-value	Degrees of Freedom
NFCorpus	1.3309	0.1842	297
SciFact	-0.9745	0.3306	299

Table 9 shows the results of paired t-tests comparing the BM25 baseline performance to the BM25 with best parameters for NFCorpus and SciFact datasets. The t-statistic, p-value, and degrees of freedom are reported for each dataset.

Table 10: Performance Comparison of Rank Fusion Methods on TREC-COVID

Method	MAP	nDCG@10	Recall@1000
Borda	0.2024	0.6072	0.3965
CombSUM	0.1967	0.5844	0.3933
CombMNZ	0.1983	0.5854	0.3910

Table 6 presents the performance of three rank fusion methods (Borda, CombSUM, and CombMNZ) on the TREC-COVID dataset, showing Mean Average Precision (MAP), nDCG@10, and Recall@1000 metrics for each method.

Table 11: Performance Comparison of Retrieval Methods on TREC-COVID

Method	Mean nDCG@10
BM25 (best params)	0.5897
Query Expansion (n=5, m=8)	0.5184
Query Reduction (n=6)	0.4826

Table 11 shows the mean nDCG@10 scores for different retrieval methods applied to the TREC-COVID dataset. The methods include BM25 with best parameters, Query Expansion, and Query Reduction.

Table 12: Statistical Significance Tests Against BM25 Baseline

Method	t-statistic	p-value	Degrees of Freedom
BM25 (best params)	1.636e-16	0.999	49
Query Expansion	-1.760	0.085	49
Query Reduction	-2.378	0.021	49

*Statistically significant at p < 0.05

Table 12 shows the results of paired t-tests comparing different retrieval methods against the BM25 baseline for the TREC-COVID dataset. The tests were conducted on nDCG@10 scores across 50 queries.

7.1 Trends and Differences Observed

Analysis of Information Retrieval Methods Across Multiple Datasets

The analysis of information retrieval methods across SciFact, NFCorpus, and TREC-COVID datasets reveals significant trends and differences in performance, highlighting the challenges of developing universally effective retrieval strategies. Our investigation began with the BM25 algorithm as a baseline, which demonstrated varying effectiveness across the datasets. SciFact yielded the highest baseline performance with an nDCG@10 of 0.6657, followed by TREC-COVID at 0.5829, and NFCorpus at a substantially lower 0.3353. This disparity underscores the inherent differences in dataset characteristics, potentially stemming from variations in document structure, query complexity, and domain-specific language.

Attempts to enhance BM25 through parameter tuning produced mixed results. For NFCorpus, tuning marginally improved performance from 0.3353 to 0.3382, while for SciFact, it led to a slight decrease from 0.6657 to 0.6634. The lack of statistical significance in these changes (p-values > 0.05) suggests that while parameter tuning can offer incremental improvements, particularly for lower-performing datasets, it doesn't consistently provide substantial gains across different contexts.

Query modification techniques, namely Query Expansion and Query Reduction, were explored to potentially enhance retrieval performance. Surprisingly, Query Expansion consistently underperformed compared to the BM25 baseline across all datasets. On SciFact, it achieved an nDCG@10 of 0.5966, on NFCorpus 0.2834, and on TREC-COVID 0.5184 – all lower than their respective BM25 baselines. This unexpected outcome challenges the assumption that expanding queries with additional terms invariably enhances retrieval performance.

Query Reduction presented a more nuanced picture. While it closely matched the baseline performance on SciFact (nDCG@10 of 0.6643), it fell short on both NFCorpus (0.2906) and TREC-COVID (0.4826). The significant underperformance on TREC-COVID (p=0.021) is particularly noteworthy, highlighting the potential risks of query simplification in complex, specialized domains.

The exploration of rank fusion methods on the TREC-COVID dataset revealed that while Borda outperformed CombSUM and CombMNZ, it still failed to surpass the BM25 baseline. This finding challenges the notion that combining multiple ranking strategies necessarily leads to improved performance, at least in the context of this specialized dataset.

A critical insight from this study pertains to the generalizability of methods across datasets. The techniques that showed promise on SciFact or NFCorpus did not consistently translate to improved performance on TREC-COVID. This was evident in the case of Query Expansion and Query Reduction, which, despite some success on the source datasets, underperformed on TREC-COVID. Even the BM25 parameters tuned on NFCorpus, chosen for its domain similarity to TREC-COVID, failed to yield statistically significant improvements when applied to the latter.

In conclusion, no single method consistently outperformed others across all datasets. The BM25 algorithm, with its default parameters, emerged as a surprisingly robust baseline, often matching or exceeding the performance of more sophisticated approaches.

7.2 Insights on Rank Fusion

Tge results, as presented in Table 10, show that among the fusion methods tested, Borda consistently outperformed both CombSUM and CombMNZ across all metrics (MAP, nDCG@10, Recall@1000). Borda achieved the highest nDCG@10 of 0.6072, compared to 0.5844 for CombSUM and 0.5854 for CombMNZ. However, it's noteworthy that the performance differences between these fusion methods were relatively small, suggesting that the choice of fusion method may not be critical in this context.

When comparing the fusion methods to the BM25 baseline, which achieved an nDCG@10 of 0.5829 for TREC-COVID, we observe that Borda's performance represents a modest improvement. However, CombSUM and CombMNZ performed only marginally better than the baseline. This modest improvement suggests that while rank fusion can enhance retrieval performance, the gains may not always be substantial, especially when dealing with a strong baseline method.

The effectiveness of rank fusion methods appears to be influenced by several factors, including the runs included in the fusion process and the characteristics of the queries. Although we don't have explicit information about the runs included in the fusion, we can infer that they likely comprised variations of BM25, Query Expansion, and Query Reduction methods. The modest improvements suggest that the fused runs may have had similar rankings, potentially limiting the scope for significant gains through fusion.

The nature of the TREC-COVID queries and document collection also plays a crucial role in the performance of fusion methods. TREC-COVID queries are likely complex and domain-specific, given the specialized nature of COVID-19 research. The average document length for TREC-COVID (160.77 words) and high D/Q ratio (493.5) suggest a rich and complex document collection. These characteristics may make it challenging for fusion methods to significantly outperform a well-tuned baseline method.

Despite the modest improvements, rank fusion methods, particularly Borda, demonstrated some ability to enhance retrieval performance beyond the BM25 baseline. This suggests that fusion may offer incremental rather than transformative benefits in this context. The relative success of Borda over CombSUM and CombMNZ indicates that the choice of fusion method can impact performance, with Borda's rank-based approach potentially being more robust to score differences across runs compared to the score-based approaches of CombSUM and CombMNZ.

In conclusion, while rank fusion shows potential for improving retrieval performance, its effectiveness appears to be dependent on factors such as the choice of fusion method, the diversity and quality of input runs, and the nature of the queries and document collection. For the TREC-COVID dataset, fusion offered modest improvements over the baseline, with Borda emerging as the most effective method. However, the gains were not substantial enough to conclusively demonstrate the superiority of fusion methods over a well-tuned baseline for this particular task. Future research could explore the impact of including more diverse runs in the fusion process or investigate adaptive fusion strategies that consider query characteristics to potentially yield more significant improvements.

7.3 Results Obtained on Scifact and nfCorpus compared to Trec-Covid

The performance of various ranking methods across SciFact, NFCorpus, and TREC-COVID datasets reveals intriguing patterns and highlights the challenges of developing universally effective retrieval strategies.

Baseline Performance

The BM25 baseline showed varying effectiveness across the datasets:

- SciFact: nDCG@10 of 0.6657
- TREC-COVID: nDCG@10 of 0.5829
- NFCorpus: nDCG@10 of 0.3353

This disparity underscores the inherent differences in dataset characteristics, potentially stemming from variations in document structure, query complexity, and domain-specific language.

Parameter Tuning

Attempts to enhance BM25 through parameter tuning produced mixed results:

- NFCorpus: Improved from 0.3353 to 0.3382
- SciFact: Decreased from 0.6657 to 0.6634
- TREC-COVID: Improved from 0.5829 to 0.5897

Statistical significance analysis (Table 9) revealed:

- NFCorpus: t-statistic = 1.3309, p-value = 0.1842
- SciFact: t-statistic = -0.9745, p-value = 0.3306
- TREC-COVID: t-statistic = 1.636e-16, p-value = 0.999

These results suggest that while parameter tuning can offer incremental improvements, particularly for lower-performing datasets like NFCorpus, it doesn't consistently provide substantial or statistically significant gains across different contexts.

Query Modification Techniques

Query Expansion consistently underperformed compared to the BM25 baseline across all datasets:

- SciFact: nDCG@10 of 0.5966
- NFCorpus: nDCG@10 of 0.2834
- TREC-COVID: nDCG@10 of 0.5184

Query Reduction presented a more nuanced picture:

- SciFact: nDCG@10 of 0.6643 (close to baseline)
- NFCorpus: nDCG@10 of 0.2906 (underperformed)
- TREC-COVID: nDCG@10 of 0.4826 (significantly underperformed)

Statistical significance analysis for TREC-COVID (Table 12) showed:

- Query Expansion: t-statistic = -1.760, p-value = 0.085
- Query Reduction: t-statistic = -2.378, p-value = 0.021 (statistically significant)

These results challenge the assumption that expanding or reducing queries invariably enhances retrieval performance, especially in complex, specialized domains like TREC-COVID.

Rank Fusion Methods

Rank fusion methods were only applied to TREC-COVID, with Borda outperforming both CombSUM and CombMNZ:

- Borda: nDCG@10 of 0.6072
- CombSUM: nDCG@10 of 0.5844
- CombMNZ: nDCG@10 of 0.5854

While Borda showed a modest improvement over the BM25 baseline, the gains were not substantial enough to conclusively demonstrate the superiority of fusion methods.

Performance Analysis

The ranking methods' performance varied significantly across datasets:

1. On SciFact, most methods performed relatively well, with the baseline and tuned BM25 achieving high nDCG@10 scores. Query Reduction also performed competitively.

2. On NFCorpus, all methods struggled to achieve high performance. While parameter tuning and Query Reduction showed slight improvements, the overall performance remained low.

3. On TREC-COVID, the BM25 baseline proved robust, with parameter tuning and Borda fusion offering modest improvements. Query modification techniques underperformed significantly.

The varying performance can be attributed to several factors:

- Dataset Characteristics: The high performance on SciFact suggests a well-structured dataset with clear relevance patterns. NFCorpus's low scores indicate a more challenging retrieval task, possibly due to diverse or ambiguous content. TREC-COVID's moderate performance reflects the complexity of COVID-19 related queries and documents.
- Query Complexity: TREC-COVID likely features more complex, domain-specific queries compared to SciFact and NFCorpus, challenging the effectiveness of simple query modification techniques.
- Document Collection: The high D/Q ratio and longer average document length in TREC-COVID suggest a rich and complex document collection, potentially making it more challenging for advanced methods to significantly outperform a well-tuned baseline.

Reference

Anthropic. (2024). Claude [Large language model]. Retrieved from https://www.claude.ai

AI assistance was used in this assignment for structuring and articulating the interpretations of my own results. The results and inferences are my own, and AI was employed solely to help with wording and presentation.