CSI 4107 Group 1: Assignment 1

https://www.site.uottawa.ca/~diana/csi4107/A1_2023.htm

Group Members

Howard Hao En Tseng, 300108234

Leah Young, 300118869

Shang Lin Hsieh, 300121996

Contribution

Name	Contribution
Howard Hao En Tseng	Initializing assignment project, "Getting Started" section, **Preprocessing**, initialized indexing, Topic querying with descriptions, Report
Leah Young	**Indexing** using the Pyterrier system, Report
Shang Lin Hsieh	**Retrieval and Ranking**, processing the 50 queries and writing the results to Results.txt, fine-tuning results, Report

About the Program

This program uses PyTerrier to index documents and perform retrieval and ranking. The program also uses the NLTK library to preprocess documents and queries.

Preprocessing is done by iterating through the files in the provided `AP_collection`, finding the documents in each file, and then tokenizing the documents and queries. The documents and queries are then converted to lowercase, and stopwords from the provided stopwords file are removed. The documents are also stemmed using the PorterStemmer algorithm. The queries are also lemmatized using the WordNetLemmatizer algorithm.

The documents are then indexed using PyTerrier. The queries are then processed and ranked using PyTerrier. The results are then written to a file called `Results.txt`.

Getting Started

This section is the complete guide to setting up the project and running the program. Follow the steps below to get started.

Python 3.8 is **required** to install the packages used in this project. If you do not have Python 3.8 installed on your machine, you can download it from here.

Java 11 or greater is also required, and the JAVA_HOME environment variable must be set to the Java installation directory. Instructions on how to do so can be found here.

Recommended: Setting up a virtual environment

This is an optional, but recommended, step to set up a virtual environment for this project. This is especially important if your default Python version is not 3.8.

Before running the following commands, make sure you have python 3.8 installed on your machine. If not, you can download it from here.

To set up the virtual environment, run the following commands:

```
pip install virtualenv
python -m virtualenv .venv --python=python3.8
.venv/Scripts/Activate.ps1
```

Installing packages

To install the packages used in this project, run the following command:

```
pip install -r requirements.txt
```

Running the program

To run the program, run the following command:

```
python main.py
```

Explanation of Algorithms

Preprocessing

The preprocessing functions are defined in `preprocessing.py`, which contains the following functions and classes:

- `Document` class
- An object class to store attributes about preprocessed documents, such as the document number, document text, and tokens from preprocessing.
- `get_stop_words()` function
- A function to read the stopwords from the provided `Stopwords.txt` file and store them in a set.
- `preprocess(file: str): Document[]` function
- A function to preprocess a file, returning an array of documents. A string for the path of the file is provided, and the function opens the file. The function then extracts the documents in the file using RegEx, then iterates through each document in the file. For each document, it extracts the document number, as well as the text of the document. It then creates a `Document` object, calling the `preprocess_text` function on the document text to generate tokens for the text.
- `preprocess_text(text: str, stem: bool, stopwords: bool): str[]` function

- A function to generate a list of tokens given a text input. On each text, it brings the entire text to lowercase, then uses the `word_tokenize` function from nltk to tokenize the text. Stopwords are removed from the tokens. The Porter Stemmer is then applied to the tokens list. Punctuation is then removed, as well as empty tokens and tokens that are not alphabetic.
- Optional parameters indicating whether the stemming step or the stopword removal step should be run are available. When not provided, both of these steps will be run.
- `preprocess_directory(directory, num_files: int): Document[]` function
- A function to preprocess a directory of files. This function fetches the filenames in the given directory, and calls `preprocess_file` on the filename. The returned list of Documents is added to a list, which is returned in the end.
- An optional parameter to dictate how many files are preprocessed by this function. When not provided, this function will preprocess all the files in the directory.

Indexing

The indexing functions are defined in `main.py`, which contains the following functions:

- `generate_index()`
- This function generates an index, calling the `preprocess_directory` function on the `AP_collection`. The resulting list of Documents are stored in a Dataframe, which is then used to build an index for PyTerrier.

The `main.py` file also runs the following:

- A check to see whether the index already exists on the disk. If it does exist, PyTerrier will load the index from the disk. If it does not exist, it will call the `generate_index` function to generate an index.
- A call to `query_retrieve` to retrieve the results of the provided test queries.

Retrieval

The retrieval functions are defined in `retrieval.py`, which contains the following functions:

- `query_retrieve(model)` function
- This function is used for querying, and obtaining and outputting the results. It begins by extracting the queries that are listed in the "topics1-50.txt" file provided. It then utilizes the model index that is passed in as an argument to obtain the results. Once the results are obtained, they are outputted to a file named "Results.txt".
- Example of model parameters
- Pyterrier BM25
- Pyterrier TF-IDF
- `query(model, topic, descriptions)`
- This function queries the model given a topic input. It preprocesses the title of the query, not opting to stem or remove stopwords. The preprocessed title is then used to search the model, and the results of the search is returned.
- The description of the topic can optionally be added to the query.

Vocabulary

Sample of 100 tokens

```
munchkinland, muncho, munci, muncip, muncypar, mund, munda, mundai, mundal,
mundan, mundari, mundel, mundelein, munden, mundi, mundia, mundial, mundo,
mundoga, mundri, mundth, mundula, muneco, muneira, munem, munez, munford,
munfordvil, mung, mungai, mungala, mungar, mungari, mungbean, munger, mungo,
munhal, munhango, muni, munib, munich, munichba, munichoi, municip, municipalbond,
municipalrevenu, municipio, munier, muniinsur, munijcip, munim, munip, munir,
munira, munist, munit, muniyandi, muniz, munj, munk, munkacsi, munlei, munn,
munnetra, munninghoff, munno, munoz, munro, muns, munsan, munsch, munsel, munshi,
munshick, munshiganj, munshik, munsingwear, munson, munster, munstergeleen,
munsterman, munt, muntoni, munusami, munyarara, muoi, muon, muphi, muppet, muqass,
muqtadir, mura, murabito, murad, muradyan, murakami, murakawa, murakhovski, mural,
murali
```

Testing and Results

Testing methodology

In order to pick the best results, we tried both the BM25 and TF-IDF weighting schemes through PyTerrier.

Results

BM25:

Run	MAP score
Titles Only	0.3183
Titles and Descriptions	0.3214

TF-IDF:

Run	MAP score
Titles Only	0.3206
Titles and Descriptions	0.3238

Discussion of Answers

Query 1:

```
1 Q0 AP881225-0044 1 12.182148982691446 results
1 Q0 AP880519-0231 2 11.983979805583845 results
1 Q0 AP881002-0014 3 11.862121283807937 results
1 Q0 AP881005-0001 4 11.783656930245298 results
1 Q0 AP881108-0076 5 11.782417467982505 results
1 Q0 AP881021-0218 6 11.772874992612106 results
1 Q0 AP880310-0051 7 11.746609943265465 results
1 Q0 AP880926-0180 8 11.741859679584111 results
```

```
1 Q0 AP880825-0054 9 11.362680553201457 results
1 Q0 AP881223-0053 10 11.287671728609233 results
```

- In Query 1, “Coping with overcrowded prisons”, a total of 6 results were found that specifically mentioned overcrowded prisons. Two additional results were related to prison and inmate topics but did not mention overcrowding. The remaining 2 results were not related to the prison topic but included relevant keywords such as "overcrowded."

Query 25:

```
25 Q0 AP880811-0163 1 23.594005790790312 results
25 Q0 AP880427-0240 2 22.707425197026886 results
25 Q0 AP880605-0026 3 22.612478590984693 results
25 Q0 AP880606-0019 4 22.569974652977574 results
25 Q0 AP880427-0150 5 22.487396819029012 results
25 Q0 AP880812-0017 6 21.992687763615642 results
25 Q0 AP880929-0184 7 21.585161972678083 results
25 Q0 AP880426-0221 8 21.419545620606367 results
25 Q0 AP880310-0257 9 21.311114740002726 results
25 Q0 AP880424-0020 10 21.04631521200367 results
```

- When analyzing the search results for Query 25 on "NRA Prevention of Gun Control Legislation," it was found that all ten results mentioned relevant keywords such as "NRA," "prevention," "gun control," and "legislation" pertaining to firearms. However, it was observed that only a few of the results were directly related to the query. Instead, approximately half of the results were focused on passing legislation that mandated guns to contain a certain amount of metal to prevent potential terrorist attacks in public areas like airports. The remaining results were miscellaneous documents that contained the keywords, but did not directly relate to the query at hand.

Optimizations

The primary optimization we performed were fine-tuning the search parameters for the retrieval function, in order to improve the MAP score. Through emperical testing, we found that using the same preprocessing function, but opting to not stem the words or removing stopwords, yielded the best results.