PDF to Text + Search Engine for MOOCs
By Alex Zurawski

Overview function:

This set of scripts, when run in conjunction, takes a data set of lecture PDF files and creates a search engine that returns the best matching PDFs. The goal is for students taking MOOCs to be able to search their courses for specific topics and ideas rather than requiring them to manually sift through all their videos. This idea could be used in conjunction with a search engine based on video transcripts to provide better results to students.

Implementation:

The steps to initialize the search engine:

1. Collect data as PDFs and categorize by course and then by week
2. Collect a set of queries in excel sorted by week and then by course
3. Find a word collection sorted by descending frequency
4. Decide on a set of appropriate stop words
5. Run `create_textdata.py`
6. Run `psuedo.py'
7. Run `bm25_master_eval.py`
    a. Manually insert chosen parameters into search_engine.py

After initialization, run `search_engine.py,` insert query and receive results.

Functionality is achieved with the following files in the same directory:

- Data – PDFs gathered from all courses
- allwords.txt – Initial word collection
- bm25_eval.py – Used to test one set of parameters for BM25
- bm25_master_eval.py – Used to test many different parameters for BM25
- config.toml – Designates configuration of ranking functions and text data
- create_bow.py – Turns PDF files into text data
- create_corpus.py – Creates corpus from text data
- create_dict.py – Creates PDF lookup dictionary
- create_ii.py – Creates inverted index of corpus
- create_query.py – Transforms query excel sheet into query text file
- create_textdata.py – Runs all other create_xxx scripts except for create_dict
- create_vocab.py – Creates a vocabulary from the initial language model and the test queries
- psuedo.py – Script to create pseudo relevance judgements on the test queries
- query.xls – Query excel sheet divided into courses and weeks
- search_engine.py – Source code for search_engine.py
- stopwords.txt – Stop words to be used in the ranking functions

Algorithms and specifics can be found in the comments of the source code.

The following files are created during initialization:

- courses – A directory with the line information and corpus
    - Created by create_corpus.py
- idx – Inverted index used by ranking functions
    - Created by create_ii.py
- test_data – A directory with all PDFs transformed into text data
    - Created by create_bow.py
- lecture-qrels.txt – Relevance judgements
    - Created by psuedo.py
- qallwords.txt – Mixture of queries and initial word collection
    - created by create_vocab.py
- qallwords.txt.gz – Zipped version of above for use in the language model
    - created by create_bow.py
- query.txt – Text file of queries
    - Created by create_query.py

The file “End Results” shows the results of running on a data set curated from the UIUC MOOCs CS410: Text Information Systems, CS598: Foundations of Data Curation, and STAT542: Practical Statistical Learning. The executable `search_engine.exe` runs on the source code from ‘search_engine.py’.

To reproduce the results, use the folder “Reproduce Results” as the working directory. Then run the following in a terminal:

- create_textdata.py
- psuedo.py
- bm25_master_eval.py
    - Insert the values into the source code of search_engine.py
- search_engine.py

You can use your own data (Data), query set (query.xls), word collection (allwords.txt) and / or stop words (stopwords.txt) to create a new search engine. The current versions of them are the ones used to create the “End Result” results.

Usage:

The scripts are all written in python. Python 3.6 or higher is mandatory. Python 3.6 introduced dictionaries that retain insertion order, and this is paramount for the `create_vocab.py` script. The following packages are used:

- metapy
- math
- sys
- time
- pytoml
- PyPDF2
- os
- shutil
- re
- wordninjaz
- gzip
- pandas
- string

To run the test search engine created, enter the "End Result" folder. Either open `search_engine.exe` or run `search_engine.py` in a terminal with the "End Result" folder as the working directory.

To create a new search engine, use the folder "Reproduce Results" as the working directory. Replace the data (Data), query set (query.xls), word collection (allwords.txt) and / or stop words (stopwords.txt). Then run the following in a terminal:

- create_textdata.py
- psuedo.py
- bm25_master_eval.py
    - Insert the values into the source code of search_engine.py

The new search engine can be accessed by running search_engine.py in the terminal.

Future Possibilities:

Improvements on current methods:

- The text extraction from the PDFs overperformed expectations but could still be improved.
    - The method used sometimes returns each individual letter rather than any words, so a worse method would be used.
- Apply the procedures used to clean text in `create_bow.py` to queries after they are inputted into the search engine.
    - For example, 'adaboost' may be split to 'ad a boost' in the text, so a query of 'adaboost' would not turn up any results.
- Improve relevance judgements.
    - Pseudo relevance judgements are worse than a human annotated dataset.
- In some of the comments, I refer to an ordered collection of words as a language model.
- I misspelled pseudo as 'psuedo' in a lot of places.

Future goals:

- Improve allwords.txt
    - The used allwords.txt is from a subset of Wikipedia articles. This causes an issue as many topics are specific, such as PLSA. A method of creating an ordered collection of words not relying on the PDFs or transcripts of Coursera courses would be very helpful. Scraping textbooks or recommended reading could work.
- Develop a crawler that would extract a student's lecture PDFs.
    - The data used was extracted manually.
- Return a specific slide instead of an entire lecture PDF.
- Automate the process of developing the search engine
    - Currently, a developer must observe the best values of the BM25 ranking function and insert them manually.
- Package this well.
    - A good-looking app and secure data sets were not the focus of this project. Making all of this happen 'behind the scenes' would improve the usability for students.
- Integrate with other methods to improve results.
    - Each lecture in Coursera generally has lecture slides and a transcript for what the instructor said. This project only focuses on the former. Incorporating the latter could improve results and functionality.

Acknowledgements:

The following (and many more) were helpful in the creation and development of this project:

- Extacting data:
    - https://www.geeksforgeeks.org/extract-text-from-pdf-file-using-python/
- Moving through files:
    - https://stackoverflow.com/questions/3207219/how-do-i-list-all-files-of-a-directory
- Data cleaning:
    - CS410 MP2.1
    - https://stackoverflow.com/questions/265960/best-way-to-strip-punctuation-from-a-string
    - https://stackoverflow.com/questions/8870261/how-to-split-text-without-spaces-into-list-of-words
- Search Engine:
    - CS410 MP 2.4