Program Usage

Requirements: Java 8

You can start the program by executing the “java -jar <program name>” command in the console. The program tries to be as self-explanatory as possible (prompts with what you can do/ examples of input). Here is a summary:

- After executing the program you are asked to enter the path to the directory containing the PDFs. A simple period “.” is the current directory (containing the jar) and subdirectories are accessed by“./subdir1”, “./subdir1/subdir2”, etc.
Subdirectories contained in the specified directory are also searched for PDFs.

- The program now indexes the PDFs.

- Upon completion you may enter your search query. (eg.: “(apple OR pear) AND sauce”)

- You are presented with the top results matching your query, and may start another search.

Task 1.1

Text extraction from the PDF collection is performed on all PDFs contained within a directory named via user input. If this directory is found to exist, text is extracted to a string array from each pdf using the Apache PDFBox library in the following manner: each PDF file found in the given directory is parsed to a PDDocument object using a PDFParser. The PDDocument is then stripped of formatting to plain text and stored in an ArrayList of strings.

Task 1,2

Text indexing and search is utilised using the Apache Lucene indexing engine with the intention of utilising the LIRE engine in conjunction during the second exercise. Indexing functions based on two text fields; the content of the pdf, and the title, here assumed to be contained within the first 100 characters of the pdf. Each of these fields is added to an individual document stored in string format during Task 1.1. Additionally, the term vectors of each field are stored; that is, the single-document inverted index containing information regarding positions, frequencies, and offsets of a term within a field. Using this information, an IndexWriter object adds the document and its associated index information to the index for the collection of PDFs. Indexing for each field is performed prior to the performance of any queries and remains valid for all queries afterwards.

Task 1,3

After the indexing has successfully completed, the user has access to the interactive query interface. In order to make multi-keyword search possible, we make use of the lucene-library’s QueryParser. This way you can use logical operations in your search-string

(e.g.: “(apple OR pear) AND sauce”) to further specify your search. After the search is completed you are presented with a list of the top matching PDFs, sorted by relevance.

Task 1.4

Ranking is performed using the Lucene Scoring algorithm, which performs similarity checks in order to produce a “score” for the relevance of each document to the provided query. Each field can be boosted; by default the occurrence of a term in each field is scored with equal weighting. Using boosting, however, you can increase the weighting an occurrence within a certain field contributes to the overall score. It is clear that the occurrence of a term within the title of a PDF should hold more weighting than the occurrence of a term within the body of text. As such, in our algorithm, the introduction is assumed to be the first 100 words of a document and the intro field is provided with a field boost of 100.0.

There are various implementations of the Lucene Similarity class available, as well as the option to override the methods within. The chosen similarity calculation utilises a vector space model; a multidimensional vector exists in which documents and queries are represented using a dimension per term. This is altered using overridden functions. For example, using the original implementation, the length of a document is normalised in such a way that the discovery of a term in a shorter text is more significant than the discovery in a longer text, as a term makes up a larger percentage of the text. This was changed to remove this effect, as this feature may result in the appearance of a term in the title of a long document being less significant than the discovery of the same term in a shorter text.

Additionally, the term frequency within a field is taken into consideration. Under the default scoring, Lucene calculates the square root of the frequency of a term in each document, meaning that large increases in the term frequency are not linearly proportional to an increase in score. In our implementation, this is simply altered to return the term frequency and maintain linearity.

The selected scoring also considers how many of the queried terms appear in a document, query and index field boosts, and computes a normalisation factor to allow queries to be compared.

Result example

The following results were outputted upon searching for the query term “organic”:

- A total of 24 PDFs matched your query.
- The top results are:
- Conrad_et_al.pdf with a score of: 0.8914751
- Beekeretal.pdf with a score of: 0.31153703
- davisandoldfield.pdf with a score of: 0.11502905
- 126-1240-1-PB.pdf with a score of: 0.11502905

- 43-567-1-PB.pdf with a score of: 0.11502905
- 40-321-2-PB.pdf with a score of: 0.09585754
- conrad_etal.pdf with a score of: 0.07668603
- 103-1016-2-PB.pdf with a score of: 0.07668603
- Boomert.pdf with a score of: 0.057514526
- siegel_etal.pdf with a score of: 0.057514526

From this example, it is clear that the very top scoring document, Conrad_etal.pft, is the best match by a significant scoring margin. This is the expected result, as Conrad_etal.pdf contains the term “organic” in the title, making this a significant result over those which simply contain the term in the body of the text.