Design Document

Code Authors: Brendan Connolly, Ronan Loftus, Conor Hickey, Ultan Griffin

Architecture Design

This program was written in 4 distinct parts: the stemmer (Conor Hickey), the Indexer (Ronan Loftus), the Rank Generator (Ultan Griffin) and the evaluator (Brendan Connolly).

Our program takes as inputs a document collection, a corresponding query collection and a set of relevance judgements for the query collection. The program outputs a series of precision-recall values that can be easily turned into graphs. For more information on how the individual parts work and what kinds of operations they are performing please read the breakdown below.

The Stemmer

The stemmer takes a document collection (in this case MED.ALL) as an input file. It proceeds to take all punctuation and stop words out of the collection. It also stems the words. Once it has done this it outputs the processed document collection into a comma separated value file with can be used by the Indexer.

The stemmer works similarly with respect to the Query collection with one notable difference, the stemmer also takes the query identifiers out. This was decided upon because once in csv format the query identifier terms (i.e. “I <queryNumber> W”) are obsolete; the query number is the same as the line number in the csv file. Once processed the queries are outputted to a csv file that will be used when calculating the BM25 ranks.

The Indexer

Our program reads words out of the processed document collection and puts them into the index. Each term is a key in an unordered map. Each key points to the set of document ids in which the term is present in and the term frequency in those documents. Each key-value pair is called a posting. Whenever a term is added to the index from a certain document the length of that document in the Indexer is incremented. Documents are stored with ids (a hashed form of their name), name and length. Access is provided to this indexer class by use of accessor methods to do such things as getting a posting for an individual term, getting the length of a document or even getting the average length of all documents in the collection. These methods are used in the rank generator when working with the BM25 formula. In our program the Indexer is created in main.cpp and passed into the Rank generator function for use.

The Rank Generator

The rank generator reads in the set of queries from a csv file that was created previously in the stemmer. For each query, it proceeds to calculate a rank for all documents that contain terms present in the query. To do this it takes each term (T) from the query (Q) and gets a posting from the indexer for T. This gives a list of all documents where T is present. Each one of these returned documents gets its rank with respect to Q updated. All documents that don't contain T are left alone. This leaves a data structure (called ranks) where each query has an entry. Inside the entry of each query, the list of document ids and their respective ranks are stored. If a document has rank 0 with respect to a certain query its rank will not get stored. This saves on space and makes the data structure more efficient. This data structure is passed onto the evaluator to see how the ranking system has performed.

The Evaluator

The evaluator gets passed the same data structure (ranks) as was created in the rank generator. This means that for each query the all of the relevant ranks and document ids are available. This data is sorted so that for each query the document with the highest rank is first in the list and the document with the lowest non-zero rank is last. The document ids are also changed back into the document names for comparison with the relevance judgements. The relevance judgements are read in from "MED.REL".

Then, for each query (Q), the evaluator starts at the beginning of the ranks of Q and checks to see if the document with the highest rank is relevant according to the user (This information can be found in the relevance judgements). It then checks the second highest ranked document, then third and so on... On each check precision and recall values are updated depending on whether a document is relevant or not. Every time a new recall milestone (10% recall, 20%, 30%, etc...) is met the corresponding precision is recorded. When all of the queries have been analysed, the recorded precision values are outputted to a csv file along with the recall milestone values (to make it easier to plot in excel). This is the final output of the program.

Interface Design

Most of our code was written in C++ but the stemmer was written in python. The interfacing between these two parts of our code was done by the use of external comma separated value files. These files are created in the stemmer and read back in again at the appropriate point in the various other parts of the program.

All parts of the code have some interfacing with external files, mainly in the way described above but with two exceptions. The stemmer interfaces with the document and query collection. The evaluator interfaces with the relevance judgements file.

Our program does not have a user interface beyond just running the code from a command line or an IDE. Refer to the user manual on how to do this.

Data Design

A description of our data structures used is discussed in a separate document: "Data structures used.pdf".