User Manual for Query Expansion Software

Zheyun Feng
fengzhey@msu.edu
Department of Computer Science and Engineering
Michigan State University

August 29, 2012

1 Introduction

The *DocRetri* package includes a library(*libDocRetri.jar*) and a runnable application (*RunnableDocRetri.jar*) for the implementation of Okapi formulation using the Lucene search engine and four methods for automatic query expansion. The source codes for the implementation of Okapi formulation and query expansion can also be founded in package. Additionally, an example to run the application is also concluded in the package, including the a small data set together with its standard seed file, and a default configuration file (*configuration.prop*).

2 Running the Program

To run the application, the user need to input the following command in the terminal:
cd (the directory of RunnableDocRetri.jar)
java -jar RunnableDocRetri.jar

All the user defined parameters are passed by the configuration file (*configuration.prop*). Below, we first assign the task of the run, then discuss the construction of document index and at last describe the program for document retrieval and query expansion.

2.1 Task assignment

Before the application running, the user need to specify the task.

- **option** that indicates the task of the run. The user need to select one of the three: indexing(i), document retrieval(r) or query expansion(e). You need to inform the program to perform indexing, document retrieval or query expansion. If you choose to perform index, an index for the given collection will be constructed. If you choose to perform document retrieval, the resulting file will contain a list of ranked documents and their relevance scores. If you choose to perform query expansion, the resulting file will contain a list of expanded words and their weights.

2.2 Indexing

There are three parameters needed to set for document indexing.

- **Index directory** that stores the indexing files.
- **Document directory or seed file** that stores files to be indexed and the corresponding relevance judgments. When it is a directory, all the documents under this directory and its child directories will be indexed, and the relevance will be automatically set to -1. When it is a *.csv* file that stores both the files to be indexed and relevance judgments, the documents in the files, their relevance judgments and the index order will be indexed. In this case, each row represents the relevance judgment of a document to a given query, and the columns in the file, from the left to the right, are:

  topic/query number, relevance($\pm$ 1 or 0), document name, document path

  An example of a row in the standard seed file could be:

  1,0,AP880216-0195,example/data/Col/AP880216/0195.txt

  If you have a non-standard seed file, you need to convert it using the function *CreateSeedDic(String, String)* first.

- **Stop word file** (optional) that stores the list of stopwords to be eliminated from indexing and searching. An example of stop word file is shown in $codes/example/src/stopfile.txt$

If an index already exists in the assigned directory, the application will ask:

Index exists. Do you want to index again?

So if you want to replace the old one, type "**y**"; otherwise, type "**n**".

2.3 Document Retrieval

Once the indexing is finished, you can begin to perform the retrieval or query expansion. There are two parts of setting needs to be determined for running the program.

Setting for all query expansion methods

- **Index directory** where the indexing files are stored
- **Query file path** for the given query(ies). A query file can include more than one queries, with each row for one query followed by the query id. Below is an example of the query file:

  1,Accusations of Cheating by Contractors on U.S. Defense Projects
  2,education

  This query file is consisted of two queries, where the first query is "Accusations of Cheating by Contractors on U.S. Defense Projects", and the second query is "education". The query is usually a sentence when the requested results are the retrieval documents, *i.e.* 1; and it is usually a word when the requested results are expanded queries, *i.e.* 2.
- **Output file** that specifies the file for storing the intermediate and the file results.
- **Number of documents/words returned** that specified the number of documents to be returned or the number of keywords to be expanded from the original query.

Setting for specific methods

- **Pseudo relevance feedback and Google**
  - Number of returned documents used for query expansion (i.e. m).
  - Parameter of expansion (i.e. α) used by the Rocchio formulation.
- **WordNet**
  - Path of the WordNet library file (i.e. wn_s.pl, which can be found in *codes/example/src/wn_s.pl* in the package).
  - Directory for indexing WordNet Synonym.

  - Parameter for WordNet expansion (i.e. α)

- **Random walk model**
  - Parameter m

3 Other Problems

When perform the Google document retrieval, the program may terminate in the middle of extracting Google returned documents using Google Web document retrieval API. This is due to the restriction of Google API service term. If you didn't see the following sign in the terminal,

$$**********Finish!**********$$

just run the program again until see it.

Because of Google's term, the user can't send request to Google frequently or automatically, the documents returned by Google would be stored in the local disk, whose directory is "*OutputDirectory/GoogleResults/*". Each query corresponds to a file named with the query id in that directory.