COEN 169 Web Information Management

Project 1 (100 points)

Due: TBD, W5~

Introduction

In this assignment, you will build indexes for a corpus and implement several information retrieval algorithms with the Lemur Toolkit (http://www.lemurproject.org/) in C/C++.

Both the text document collection and the incomplete implementation of a retrieval system can be found in a light version of the Lemur toolkit (see **Instructions** below). The light version only runs on Unix/Linux. You should remotely Login to the Engineering Computing Center's Linux machine (ftp.engr.scu.edu) by using ssh or putty to complete this assignment.

Instructions

Follow the procedures below to install the light Lemur toolkit:

1. Login to the Linux machine (ftp.engr.scu.edu). Use the following command

wget http://www.cse.scu.edu/~yfang/lemur.tar.gz

to get the toolkit.

1. Use "gzip -d lemur.tar.gz" and "tar -xvf lemur.tar" to uncompress and unpack the package.

2. Enter the lemur root directory and use "./configure" to configure the Lemur toolkit

3. Use "gmake" in the lemur root directory to compile the package

4. Every time you change the code go to the lemur root directory to rebuild the code by "gmake"

There are two directories under the lemur root directory that are particular useful. The "eval_data/" directory contains the text document collection (i.e., database.sgml) to analyze and several parameter files to run the retrieval system. The "app/src/" directory contains the code of the applications. "BuildIndex.cpp" is the application for building indexes (complete). "RetrievalEval.cpp" is incomplete, which you will work on.

Implementation of several retrieval algorithms

In this task, you are going to use lemur toolkit for retrieval and implement several retrieval algorithms by yourself. First, please build the inverted index files. Enter the directory of "eval_data" under the light Lemur root directory. Use the command "../app/obj/BuildIndex build_param database.sgml" to build the index files with raw text data. Use the command "../app/obj/BuildIndex build_stemmed_nostopw_param database.sgml" to build the index files with text data processed by stemming and removing stop words.

The source file "RetrievalEval.cpp" under "app/src/" contains several retrieval algorithms. Please read through the annotated code to understand the framework. Different retrieval algorithms have different implementation of functions as computeMETHODWeight and/or computeMETHODAdjustedScore (METHOD should be replaced by the names of different retrieval method like RawTF). The retrieval score of a document (D) with respect to a query (Q) is calculated as follows:

$$S(Q, D) = \text{scoreAdjust}(\text{Weight}(q_1,d_1,Q,D) + ... + \text{Weight}(q_N,d_N,Q,D), Q, D)$$

Where computeMETHODWeight calculates the value of the Weight function for a matched term of a document and a query; computeMETHODAdjustedScore is used to calculate the value of scoreAdjust function (e.g., for score normalization).

One simple retrieval algorithm "RawTF" has been implemented. You can refer to the code for implementing other retrieval algorithms.

To play with Lemur toolkit and implement your retrieval algorithm, please complete the following experiments:

1. Test the performance of retrieval algorithm "RawTF" with two types of text data (i.e., raw text data and text data by stemming and removing stopwords). (20 points)

Enter the directory of "eval_data" under the light Lemur root directory. Use the command "../app/obj/RetrievalEval eval_param query" to generate the result file (i.e., result_rawtf) of the RawTF retrieval algorithm for the raw text data. Use the command "../app/obj/RetrievalEval eval_stemmed_nostopw_param query_stemmed_nostopw" (before this please set the weight scheme parameter in param file to RawTF and set corresponding name of result file by the result parameter) to generate the result file (i.e., result_rawtf_stemmed_nostopw) of the RawTF retrieval algorithm for the text data by stemming and removing stopwords.

2. Evaluate the results by using "../trec_eval qrel result_rawtf" and "../trec_eval qrel result_rawtf_stemmed_nostopw". Please include the results in your report. Can you tell which result is better? If one is better than the other, please provide a short analysis. Please answer what stemmer is used in the index. Can you also use another stemmer and compare the results? The instruction of how to defining the BuildIndex parameter file can be found at http://www.cs.cmu.edu/~lemur/3.1/indexing.html#BuildIndex

3. Evaluate the results by NOT removing the stopwords. A stopword list is contained in eval_data/stopwordlist. You need to modify the parameter file (e.g., remove <stopwords>stopwordlist</stopwords> in build_stemmed_nostopw_param) when apply BuildIndex.

   Please provide a short analysis on whether removing stopwords helps or not.

2. Implement three different retrieval algorithms and evaluate their performance. (80 points)

You only need to change the computeMETHODWeight functions of the three algorithms. The weight functions of the three algorithms and the RawTF method (implemented) are:

RawTF	$weight(q_t, d_t, Q, D) = term\_freq(d_t) * weight(q_t)$
RawTFIDF	$weight(q_t, d_t, Q, D) = term\_freq(d_t) * \log log\left(\frac{totalNumDocs}{numDocsContain(term\_t)}\right) * weight(q_t)$
LogTFIDF	$weight(q_t, d_t, Q, D) = (log(term\_freq(d_t)) + 1) * \log log\left(\frac{totalNumDocs}{numDocsContain(term\_t)}\right) * weight(q_t)$
Okapi	$weight(q_t, d_t, Q, D) = \frac{term\_freq(d_t)}{term\_freq(d_t)+0.5+1.5\frac{len(D)}{avgDocLen}} * \log log\left(\frac{totalNumDocs-numDocsContain(term\_t)+0.5}{numDocsContain(term\_t)+0.5}\right) * \frac{8+weight(q_t)}{7+weight(q_t)}$

In order to obtain statistics like the average document length, please check the reference of Lemur::api::index class. More general information about Lemur classes can be found here.

Please check the parameters of weightingScheme and resultFile within "eval_stemmed_nostopw_param" for different retrieval algorithms. Follow the step in task 1 to generate the evaluation results.

In summary, you need to generate the results (Average Precision) with different combinations of retrieval models and preprocessing steps and fill in the table as below

	Remove Stopwords & Stemming	Remove Stopwords & No Stemming	No Removing Stopwords & Stemming	No Removing Stopwords & No Stemming
RawTF				
RawTFIDF				
LogTFIDF				
Okapi				

Please compare the results and provide a short discussion about the advantage/disadvantage of the algorithms.

What to turn in

1. Please write a report of your experiments.

2. Please attach the hardcopy of your code (RetrievalEval.cpp) with the report.