INFORMATION RETRIEVAL PROJECT

COURSE: CS 6200 (INFORMATION RETRIEVAL)

INSTRUCTOR: NADA NAJI

PROJECT MEMBERS: Rajkumar Murukeshan,
Revanth Raju Alluri,
Sangeetha Sivaramakrishnan

INTRODUCTION

Description: The idea of the project is to build information retrieval systems, evaluate and compare their performance levels in terms of retrieval effectiveness. The dataset provided to us contains 3204 raw documents which is the CACM test collection as our textual corpus, 64 unprocessed queries, Relevance judgements and a stop-list.

PHASE 1:
Task 1 - Build a search engine with BM25 as a retrieval model, with tf-idf as a retrieval model, and Lucene's default retrieval model and return the top 100 retrieved ranked lists for all three models.
Task 2 - Pick any one of the three runs and perform query expansion using two approaches. We picked Thesaurus and Pseudo relevance feedback
Task 3 - Perform stopping and indexing the stemmed version of the corpus provided to us and perform a query-by-query analysis.

PHASE 2 : (Choose 1 or 2)
1- Combine a query expansion technique with stopping.
2- To use a different base search engine than the ones chosen earlier, and adopt either query expansion technique and/or stopping.
(Implemented Pseudo Relevance with stopping and also tf.idf with Thesaurus query expansion technique and also performed stopping.)

A total of 7 distinct runs will be generated. To assess the performance of these search engines, implement and perform
1- Mean Average Precision
2- Mean Reciprocal Rank
3- P@K, K = 5 and 20
4- Precision & Recall

CONTRIBUTIONS

PHASE 1
Task 1 Contributions:
Tf.idf: Rajkumar and Sangeetha
BM25: RajKumar and Revanth
Lucene: Revanth and Sangeetha
Task 2 Contributions:
Pseudo Relevance: Rajkumar
Thesaurus: Sangeetha and Revanth
Task 3 Contributions:
Stopping: Rajkumar
Index the stemmed version: Rajkumar

PHASE 2
Task 1 Contributions:
Query Expansion with Stopping: Sangeetha and Revanth
Different search engine with both query expansion and Stopping: Sangeetha and Revanth
Performance assessment: Rajkumar
Documentation: Revanth, Rajkumar and Sangeetha

LITERATURE AND RESOURCES

Phase 1 Implementation: Indexing and Retrieval
TASK 1
1. TF.IDF:
The only external file that is required for the implementation of tf.idf is the Jsoup.jar file and the libraries need to be imported. Other necessary libraries required for the implementation include the Collection framework, IOException, FileWriter, File and BufferedReader.
We also referred to the following links:

http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting.html
http://docs.oracle.com/javase/1.5.0/docs/guide/collections/overview.html

Books referred:
Search Engines: Information Retrieval in Practice, Book by Donald Metzier, Trevor Strohman and W Bruce Croft.

2. LUCENE:
The implementation of Lucene required the use of external jars namely lucene-analyzers-common-4.7.2.jar, lucene-core-4.7.2.jar and lucene-queryparser-4.7.2.jar files.
We also referred to the following links

http://en.wikipedia.org/wiki/Information_retrieval
http://lucene.apache.org
http://www.lemurproject.org
http://www.search-engines-book.com
We used the below link to download the files required for setting up Lucene
https://archive.apache.org/dist/lucene/java/4.7.2/

3. BM25
The implementation of BM25 required the use of jsoup-1.8.3.jar external jar file. We re-used our code from assignment 3 to complete this task

TASK 2
1. BM25 with Pseudo Relevance
The implementation of BM25 required the use of jsoup-1.8.3.jar external jar file. We re-used our code from assignment 3 to complete this task

2. BM25 with Query Expansion (Thesaurus and Ontologies)
We used Java API for WordNet Searching (JAWS).
The external jar files we use to implement query expansion in BM25 are:
Jaws-bin.jar, jsoup-1.8.3.jar.
We used the following references

http://lyle.smu.edu/~tspell/jaws/index.html#downloads
http://lyle.smu.edu/~tspell/jaws/doc/edu/smu/tspell/wordnet/WordNetDatabase.html
http://wordnet.princeton.edu/wordnet/download/
http://lyle.smu.edu/~tspell/jaws/doc/overview-summary.html
http://lyle.smu.edu/~tspell/jaws/TestJAWS.java

TASK 3
STOPPING, STEMMING
Used the commons_words.txt provided to perform stopping and also referred the textbook to implement them.

Phase 2 Implementation: Evaluation
TASK 1.
TFIDF with Query Expansion and Stopping
Java API for WordNet Searching (JAWS)Jaws-bin.jar, jsoup-1.8.3.jar
We used the following references

http://lyle.smu.edu/~tspell/jaws/index.html#downloads
http://lyle.smu.edu/~tspell/jaws/doc/edu/smu/tspell/wordnet/WordNetDatabase.html
http://wordnet.princeton.edu/wordnet/download/
http://lyle.smu.edu/~tspell/jaws/doc/overview-summary.html
http://lyle.smu.edu/~tspell/jaws/TestJAWS.java

TASK 2.
MAP, MRR, P@K, K = 5 and 20, Precision & Recall
Search Engines: Information Retrieval in Practice, Book by Donald Metzier, Trevor Strohman, and W. Bruce Croft.

IMPLEMENTATION AND DISCUSSION

Implementation of TF.IDF

Global Data Structures needed:
1.'Queries' is a LinkedHashMap which is used to preserve the insertion order. It contains the queryid and the query terms.
2.'Doclist' is a LinkedHashSet which contains all the document names that are collected from the cacm folder. A LinkedHashSet is used to preserve insertion order and to maintain unique document set.
3.'Documents' is a LinkedHashSet which contains the document objects which in turn have the document name, length of the document and the term count in the document
4.'DocFrequency' is a LinkedHashMap to maintain the number of times the term occurs in the document.
5.'TermCount is the number of times the term occurs in a given document

Main Method:
We start from the main method. This is where we get the queries by calling the getQueries() method. In this method we parse the cacm.html file in order to get the document id and the content of the document. We use JSOUP to parse by tags and for every element we get the document number. Then we get the text and convert it to lowercase and apply regular expressions to remove digits, and punctuation marks at appropriate locations. We add these terms to a string array where each term is split by 'space' and add the document id and the query terms to the queries data structure mentioned above.
Now we need to get all the file names in the cacm corpus.This function adds all the file names from the corpus folder to the doclist linkedhashmap.
Now we add all the document names to the documents data structure.
Then get the document frequency using the getDocumentFrequency() function. We then get the term count for every document and loop through the terms data structure to see if the term exists or not. If it doesn't then we add to the term and the count to docFrequency, if not we add the term and the integer 1 to represent that it adds the term for the first time.
Now for all the queries we retrieve the documents. Here we get the tfidf score using the document object and the query terms. This is calculated by getting each term from query terms and if the document contains the term by get the term count, we perform executing the formula provided in the book. N is the doclist size and if the docFrequency contains the term we set that

to a temporary variable and then get the inverse document frequency by using the Math.log function which gives the total number of documents/ document size.
Now multiple the term frequency and the inverse term frequency to return a final value for the document. Now in retrieveDocuments() we just compare the scores and get the top 100 scores which are stored in a text file.

TFIDF with Stopping and Thesaurus

For TFIDF Thesaurus, There will be some changes to the main method. THe retrieve documents will now be called in a function called the PerformThesaurus() which in-turn retrieves synonyms on every term for a query and then adds the new query to a new arraylist. The synonyms come from the Java API called wordnet. Then the retrieveDocuments() function is called to get the top 100 documents.
For Stopping: Use the commons_words.txt to perform this process.We have a new data structure where if any of the query terms are present in the stop list, these terms will be added to the stoplist data structure and a new query will be generated and added to a new array without the stop-words in them. This implementation is carried out in the getQueries() function.

Implementation of BM25
BM25 with Relevance Information
Short Report describing the implementation of BM25 scoring and Retrieval:
Implementation of the main method:
Algorithm:
Step 1: Read the input corpus file which contains the document collection along with the data that the document contains.
Step 2: Create a LinkedHashMap<String, ArrayList<DocInfo>> ind data structure to store the inverted index for each word.
The Doc.java is a new class which consist of docId and term_req as it's data member.
Create a LinkedHashMap<Integer,Integer> dl to store the number of tokens for a particular document id. docId is key.
Call the indexer method on the corpus file.
Step 3: Read directly from the inverted index file and load it into a LinkedHashMap<K,V> inverted_index.
Step 4: Call the calculate BM25 function on inverted_index , dl, input queries and the numOfDocs. (i.e. no of documents to display in the final result).
Step 5: End Of Main.
Implementation of the calculateBM25 method
Algorithm:
Step 1: Read the input queries.txt file.
Step 2: Initialize the constants used in the calculation of BM25 score
i.e. k1 = 1.2, k2 = 100 and b = 0.75
Step 3: For each line read
create a LinkedHashMap<String, Double> qfreq to store the query term frequency.
call build_frequency_for_query to load the above hash map.
create a LinkedHashMap<Integer, Double> score_list to store the BM25 scores of all the documents containing the query term.
split the line based on space
for each string
get the ArrayList of all the documents dlist for the corresponding word.
for each document d in list
calculate the BM25 score using the given formula

store each result obtained in the HashMap i.e. if the docId is already present then add the new result to the previous value else store the new value along with the document id.

write the final result to a file called “result.eval” i.e. write the contents of score_list data structure to the file in a sorted order using a comparator class.

Step 4: Close the input and the output files after completion of read and write operations.

BM25 with Pseudo Relevance with Stopping

In this implementation

Step 1: Retrieve top 100 ranked documents according to BM25 scoring and retrieval

Step 2: Assume top 5 documents are highly relevant.

Step 3: Based on tf value as weights of the top 5 most frequent stopped tokens/query terms we perform query expansion

BM25 with Thesaurus

In this implementation:

Step 1: We perform query expansion using a thesaurus. This refers to a dictionary of synonyms and ontologies which we achieve by implementing the Java API for WordNet Searching.

Step 2: With the retrieved synonyms now appended to the queries we perform document scoring and retrieval using the BM25 scoring method and get a list of top 100 ranked documents for each expanded query.

Implementation of Lucene

The only implementation done for Lucene was to fetch the queries from the cacm.html file. Apply case folding to each of the query terms. Now use a loop to add the terms as a single string and retrieve the top 100 documents for the query this continues for all 64 queries to get a total of 6400 documents where each query gets the top 100 relevant documents.

Query-by-Query Analysis for Stemmed Corpus and Query:

PFB the analysis for Stemmed Queries :

Query 1 : "parallel algorithm"

Top 5 Retrieved Documents :

DocId	Rank	Score	# of Occurrences of Parallel	# of Occurrences of Algorithm
CACM-2714	1	7.049768	7	4
CACM-2973	2	6.652688	4	3
CACM-0950	3	6.456513	4	2
CACM-2664	4	6.444489	4	1
CACM-2785	5	6.330522	4	1

From the above values, we can see that the there is a direct proportionality between the No Of Occurrences of term frequency and the scores. Also, we see that Document at rank 4 and 5 have same number of occurrences of both terms. However, there is a difference in rank. It is because the length of the document at rank 4 is less than the document length at rank 5. Here comes the picture of length normalization which uses the formula to calculate K in BM25 score calculation.

K = k1 ((1 -b) + b (dl/avdl));

where, b = normalizing constant , dl = length of the document and avdl = average length of the document in the corpus.

After comparing with the original document, we also see that there is high topical relevance with the results obtained using the Stemmed Approach.

Query 2: “portabl oper system”

Top 5 Retrieved Documents :

DocId	Rank	Score
CACM-3127	1	14.84204
CACM-2246	2	11.51344
CACM-1930	3	9.373616
CACM-3196	4	8.962202
CACM-2593	5	6.359929

By Comparing the documents retrieved with the query, we see that there is high match of topical relevance between the document retrieved and the stemmed query. For example document at rank 1 CACM-3127 provides information of Portable Real-Time Operating System.

Query 3 : “appli stochast process”

Top 5 Retrieved Documents :

DocId	Rank	Score
CACM-1696	1	9.340863
CACM-0268	2	8.088408
CACM-1410	3	6.813854
CACM-2535	4	6.601747
CACM-1233	5	6.226788

Similar to Query 2, the documents retrieved for this query also has high topical relevance to the query.

PERFORMANCE ASSESSMENT

MEAN AVERAGE PRECISION:

The below given table gives the comparison of mean average precision values obtained using different systems.

System Name	MAP Value
BM25-BaseSystem	0.546656916
Tf-Idf-System	0.374895583
Lucene- System	0.478948772
PseudoRelevance-BM25	0.550548459
Thesaurus-BM25	0.455683033
Stopped-BM25	0.531330787
Thesaurus-TF-IDF-System	0.300787524

MEAN RECIPROCAL RANK:

The below given table gives the comparison of mean reciprocal rank values obtained using different systems

System Name	MRR Value
BM25-BaseSystem	0.812188
Tf-Idf-System	0.603602
Lucene- System	0.728641
PseudoRelevance-BM25	0.825008
Thesaurus-BM25	0.726582
Stopped-BM25	0.773237
Thesaurus-TF-IDF-System	0.51186

Precision@RankK, K = 5 and 20:

The precision value at rank 5 and 20 is provided in the spreadsheet SystemResults.xlsx

PRECISION AND RECALL

The full table containing the precision and recall values for all queries and all runs is provided in Precision&Recall-Results.xlsx

RESULTS

Documents Scores, Effectiveness metrics are provided in the spreadsheet SystemResults.xlsx and Precision&Recall-Results.xlsx. Also, the text files generated during each run and for each system are provided in the folder SystemResults.

CONCLUSIONS AND OUTLOOK

In short, the idea of this project was to develop information retrieval system. Once this base is ready, we evaluate and compare their performances using different techniques. The dataset provided to us contained 3204 raw html documents, and also we are given 64 unprocessed queries.

We have implemented BM25, TFIDF and Lucene information retrieval models. To improve the performance we have implemented BM25 such that it can also perform stopping, stemming and also implemented different query expansion techniques for terms in the queries. We have implemented stopping and synonym generation(Thesaurus) for BM25 System and TF.IDF system. We have provided a performance assessment using MAP, MRR, P@K, K=5 & 20 and Precision and Recall.

With the results generated, we can see that the performance of BM25 Scoring System with the PseudoRelevance Feedback is more effective compared to other systems

All the third party tools, libraries and any other references have been mentioned above. The implementation has been mentioned in detail above and the results for these implementation have been provided separately in an Excel and .txt format.

BIBLIOGRAPHY

1. General Review: Search Engines: Information Retrieval in Practice, Book by Donald Metzier, Trevor Strohman, and W. Bruce Croft.

2. Jones, Sparck/Robertson “The BM25 Weighting Scheme”, 1976.
URL: https://xapian.org/docs/bm25.html

3. Java API for WordNet Searching(JAWS), Spell Brett.
URL:http://lyle.smu.edu/~tspell/jaws/

4. Collections Framework Overview, Oracle
URL: http://docs.oracle.com/javase/1.5.0/docs/guide/collections/overview.html

5. Stopword Lists, Stopwords
URL: http://www.ranks.nl/stopwords

6. Tf.idf weighting, tf.idf reference, NLP Stanford
URL: http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html

7. BM25 vs Lucene Default Similarity, Konrad Beiske, 14 January 2014,
URL: https://www.elastic.co/blog/found-bm-vs-lucene-default-similarity

8. Okapi BM25 , Wikipedia
URL: https://en.wikipedia.org/wiki/Okapi_BM25

9. Mean Average Precision, NLP Stanford
URL: http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html

10. Mean Reciprocal Rank, Search Engines: Information Retrieval in Practice, Book by Donald Metzier, Trevor Strohman, and W. Bruce Croft.

11. Precision and Recall, Search Engines: Information Retrieval in Practice, Book by Donald Metzier,
Trevor Strohman, and W. Bruce Croft.

12. Mean Average Precision, Search Engines: Information Retrieval in Practice, Book by Donald Metzier,
Trevor Strohman, and W. Bruce Croft.

13. Precision and Recall, kdnuggests
URL: http://www.kdnuggets.com/faq/precision-recall.html