Name: Suprith Chandrashekharachar
Username: suprchan
University ID: 0003392300

Assignment 2: Retrieval Algorithm and Evaluation

Task 1: Implement your first search algorithm

For each given query, the code 1. Parse the query using Standard Analyzer, 2. Calculate the relevance score for each query term, and 3. Calculate the relevance score (F, doc).

Please find the code attached with my oncourse submission.

The class is called easySearch, which contains a constructor easySearch() and member methods computeRelScore(), addValues() and main().

Short description:
easySearch () – Sets up variables pertaining and index location and query string.
computeRelScore () – Handles the calculation of relevance score for the given query string.
addValues () – Handles the maintenance of the map object containing the computed relevance score.
main() – sets the ball rolling.

The output will contain a list of sum of relevance scores for each query term for all the relevant documents in the corpus.

Usage: easySearch.java
Output Sample:

```
Enter the path to index dir
D:\IR_Materials\assignment2\index\default
Enter the query string
new governor
Computing the relevance scores based on the given input
Total Number of Documents in the corpus: 84474
Query term: new
Number of docs matching the query term: new is:38592
IDF: 0.47712125471966244
Query term: governor
Number of docs matching the query term: governor is:1792
IDF: 1.6812412373755872
************************************************************
Relevance Scores for all the relevant docs in the corpus for the given query is listed below
{AP890101-0001=30.535760302058396 ……
```

Task 2: Test your search function with TREC topics

Software outputs up to top 1000 search results to a result file in a format that enables the trec_eval program to produce evaluation reports.

Please find the source code attached with my on course submission.

The class is called searchTRECtopics, which contains a constructor searchTRECtopics () and member methods setHomeDir (),computeRelScore (),entriesSortedByValues (),addValues(),readForAP89Corpus(),inititateSearch() and main().

Short description:
searchTRECtopics () – Sets up variables pertaining and index location, query file path and home dir location.
setHomeDir () – setter for homeDir
readForAP89Corpus() – Parse function for retrieving field values from trec text files.
computeRelScore () – Calculates relevance score for every query and outputs the first 1000 relevant documents to a text file.
entriesSortedByValues() – Sorts a tree map on values.
inititateSearch() – gets query list and passes onto relevance score computation function.
addValues() - Handles the maintenance of the map object containing the computed relevance score.
main() – sets the ball rolling.

Usage: searchTRECtopics.java
Sample Output:

```
Enter the path to index
D:\IR_Materials\assignment2\index\default
Enter the path to query topics file
D:\IR_Materials\assignment2\topics.51-100
Calling searcher
Setting the directory location for storing the output files as: "D:\IR_Materials\assignment2"
Number of Queries:50
Number of Titles:50
Number of Descriptions:50
Output File Name:SearchTRECshortQuery.txt
Output File Name:SearchTREClongQuery.txt
```

The output files will be placed in the file directory where topics.51-100 is located.

Task 3: Test Other Search Algorithms

The software outputs two files each (for both short and long queries) of below listed retrieval and ranking algorithms.

1. Vector Space Model (org.apache.lucene.search.similarities.DefaultSimilarity)
2. BM25 (org.apache.lucene.search.similarities.BM25Similarity)
3. Language Model with Dirichlet Smoothing(org.apache.lucene.search.similarities.LMDirichletSimilarity)
4. Language Model with Jelinek Mercer Smoothing
(org.apache.lucene.search.similarities.LMJelinekMercerSimilarity, set λ to 0.7)

Software outputs up to top 1000 search results to a result file in a format that enables the trec_eval program to produce evaluation reports.

Please find the source code attached with my on course submission.

The class is called compareAlgorithms, which contains a constructor compareAlgorithms () and member methods setHomeDir (),computeRelScore (),entriesSortedByValues (),addValues(),readForAP89Corpus(),inititateSearch() and main().

Short description:
compareAlgorithms () – Sets up variables pertaining and index location, query file path and home dir location.
setHomeDir () – setter for homeDir
readForAP89Corpus() – Parse function for retrieving field values from trec text files.

computeRelScore () – Calculates relevance score for every query and outputs the first 1000 relevant documents to a text file.
entriesSortedByValues() – Sorts a tree map on values.
inititateSearch() – gets query list and passes onto relevance score computation function.
addValues() - Handles the maintenance of the map object containing the computed relevance score.
main() – sets the ball rolling.

Usage: compareAlgorithms.java
Sample Output:

```
Please enter the path to index
D:\IR_Materials\assignment2\index\default
Please enter the path to query topics file
D:\IR_Materials\assignment2\topics.51-100
Calling searcher
Setting the directory location for storing the output files as: "D:\IR_Materials\assignment2"
Number of Queries:50
Number of Titles:50
Number of Descriptions:50
Intiating Vector Space Ranking Algorithm
Output File Name:DefashortQuery.txt
Output File Name:DefalongQuery.txt
Vector Space Ranking Completed
Intiating BM25 Ranking Algorithm…..
```

The output files will be placed in the file directory where topics.51-100 is located.

Task 4: Algorithm Evaluation

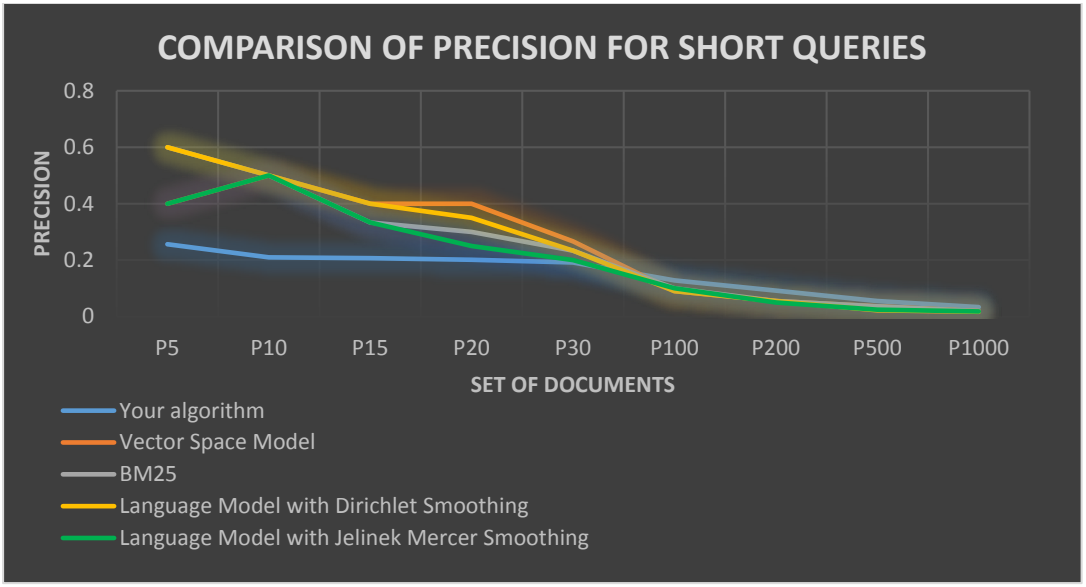
The tables listed below depict how each individual algorithm performs and gives us a great insightful comparison.
For Short Query:

Evaluation Metric	Your algorithm	Vector Space Model	BM25	Language Model with Dirichlet Smoothing	Language Model with Jelinek Mercer Smoothing
map	0.1234	0.1833	0.1894	0.1404	0.1462
R-prec	0.1514	0.2581	0.2258	0.2258	0.1935
bpref	0.2544	0.4748	0.4684	0.4348	0.4426
recip_rank	0.3818	1	1	0.5	1
P5	0.256	0.4	0.6	0.6	0.4
P10	0.21	0.5	0.5	0.5	0.5
P15	0.2067	0.4	0.3333	0.4	0.3333
P20	0.201	0.4	0.3	0.35	0.25
P30	0.1913	0.2667	0.2333	0.2333	0.2
P100	0.128	0.09	0.1	0.09	0.1
P200	0.0919	0.055	0.055	0.055	0.05
P500	0.055	0.036	0.036	0.022	0.024
P1000	0.0333	0.019	0.019	0.017	0.018

For Long Query:

Evaluation Metric	Your algorithm	Vector Space Model	BM25	Language Model with Dirichlet Smoothing	Language Model with Jelinek Mercer Smoothing
map	0.0696	0.0966	0.1333	0.0693	0.0853
R-prec	0.1081	0.1935	0.1935	0.2258	0.1935
bpref	0.2264	0.5135	0.5574	0.44	0.5032
recip_rank	0.3055	0.5	1	0.125	0.2
P5	0.156	0.2	0.6	0	0.2
P10	0.148	0.3	0.3	0.2	0.3
P15	0.1267	0.3333	0.2667	0.2	0.2667
P20	0.119	0.25	0.3	0.25	0.25
P30	0.1213	0.2	0.2	0.2333	0.2
P100	0.0922	0.1	0.1	0.1	0.09
P200	0.0662	0.06	0.06	0.055	0.055
P500	0.041	0.03	0.03	0.026	0.028
P1000	0.0269	0.017	0.019	0.015	0.017

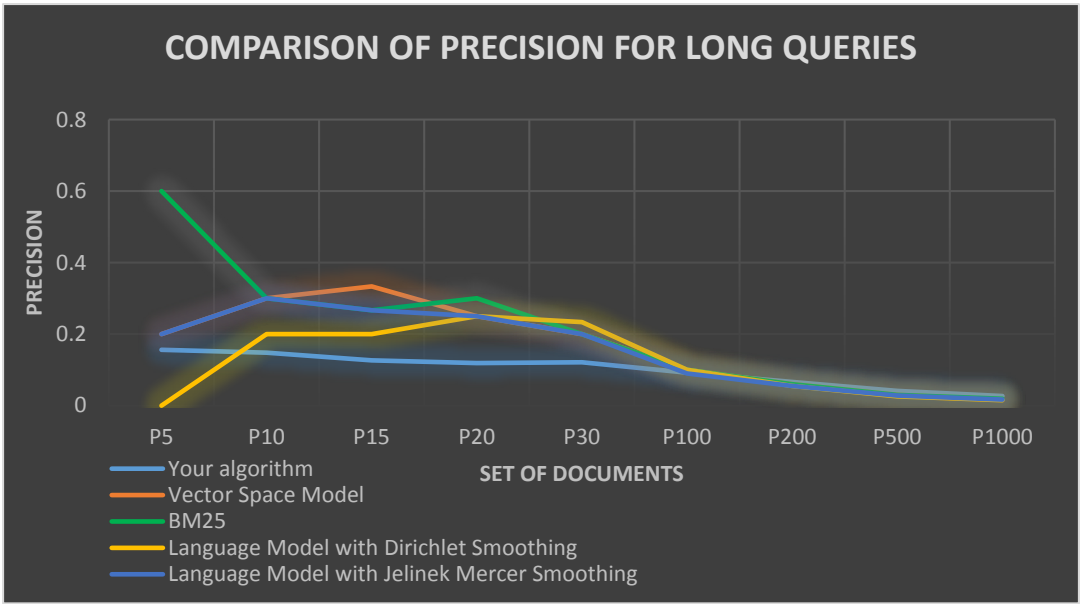
Summary of findings:



By looking at the above chart comparing the performance of different algorithms for *short queries*, we can conclude that "*Language Model with Dirichlet Smoothing*" is high on precision for the top 15 documents and continues to maintain a considerably high precision thereafter.

Assuming that satisfying the user information need with highest possible accuracy is of significant importance in information retrieval systems, the following ranked order of algorithms can be used for short queries:

- Dirichlet Smoothing
- BM25
- Vector Space model
- Jelinek Mercer smoothing
- Your algorithm (TF-IDF)



By looking at the above chart comparing the performance of different algorithms for *long queries*, we can conclude that "*BM25*" is high on precision for the top 10 documents and continues to maintain a considerably high precision thereafter.

Assuming that satisfying user information need with highest possible accuracy is of significant importance in information retrieval systems, the following ranked order of algorithms can be used for short queries:

- BM25
- Dirichlet Smoothing
- Vector Space model
- Jelinek Mercer smoothing
- Your algorithm (TF-IDF)

NOTE: Please include the jar files lucene-analyzers-common-4.10.1.jar, lucene-core-4.10.1.jar, lucene-queries-4.10.1.jar, commons-lang3-3.3.2.jar and lucene-queryparser-4.10.1 while running the programs.

Resources:

http://stackoverflow.com/questions/2864840/treemap-sort-by-value