Z534 Search
Assignment 2: Retrieval Algorithm and Evaluation
Suhas Jagadish

Task 1: Implement your first search algorithm
Solution: The code for this task is saved as **easySearch.java**. The filepaths in the program is absolute. For example,

```
IndexReader reader = DirectoryReader.open(FSDirectory.open(Paths
    .get("E:/Suhas(D drive)/Suhas/IUB/3rd Sem/Search/Assignments/Assignment 2/index/index")));
```

Please replace this path with your filepath before running the program.

Task 2: Test your search function with TREC topics
Solution: The code for this task is saved as **searchTRECtopics.java**. Once again, apologies for providing absolute file path everywhere. The result files are stored as "LongQuery" and "ShortQuery"

Task 3: Test Other Search Algorithms
Solution: The code for this task is saved as **CompareAlgorithms.java**. I have implemented a switch case where you can select different algorithms each time. The respective result files are uploaded.

Task 4: Algorithm Evaluation
Solution:

Short query

Evaluation Metric	Your algorithm	Vector Space Model	BM25	Language model with Dirichlet Smoothing	Language model with Jelinek Mercer Smoothing
P@5	0.0776	0.2960	0.3080	0.3480	0.2880
P@10	0.0857	0.3020	0.3	0.3260	0.2800
P@20	0.0827	0.2600	0.2710	0.2890	0.2430
P@100	0.0612	0.1648	0.1678	0.1690	0.1602
Recall@5	0.014	0.0539	0.0488	0.0620	0.0534
Recall@10	0.0343	0.0960	0.0879	0.0995	0.0911
Recall@20	0.0593	0.1416	0.1378	0.1467	0.1302
Recall@100	0.1404	0.3578	0.3572	0.3468	0.3320
MAP	0.0682	0.1975	0.2005	0.2059	0.1932
MRR	0.1557	0.4696	0.4772	0.4785	0.4637
NDCG@5	0.086	0.3116	0.3246	0.3517	0.3063
NDCG@10	0.0931	0.3183	0.3199	0.3420	0.3011
NDCG@20	0.0976	0.3043	0.3124	0.3326	0.2888
NDCG@100	0.1158	0.3213	0.3259	0.3311	0.3121

Long query

Evaluation Metric	Your algorithm	Vecor Space Model	BM25	Language model with Dirichlet Smoothing	Language model with Jelinek Mercer Smoothing
P@5	0.0042	0.2560	0.2840	0.2560	0.2320
P@10	0.0063	0.2440	0.2440	0.2420	0.2140
P@20	0.0042	0.2210	0.2340	0.2340	0.2120
P@100	0.0065	0.1406	0.1488	0.1460	0.1378
Recall@5	0.0001	0.0349	0.0402	0.0403	0.0406
Recall@10	0.0004	0.0621	0.0703	0.0708	0.0658
Recall@20	0.0006	0.1064	0.1159	0.1270	0.1136
Recall@100	0.0073	0.2929	0.3167	0.3340	0.2901
MAP	0.0034	0.1529	0.1676	0.1586	0.1514
MRR	0.015	0.4528	0.4597	0.3475	0.3640
NDCG@5	0.0035	0.2819	0.3029	0.2499	0.2348
NDCG@10	0.0055	0.2682	0.2729	0.2473	0.2294
NDCG@20	0.0042	0.2586	0.2718	0.2590	0.2410
NDCG@100	0.0072	0.2694	0.2872	0.2753	0.2609

The above table shows in general that our algorithm performs poorly as compared to other algorithms. The reason I believe is that TF-IDF is based on Bag of Words model and hence does not capture the position in text, semantics or co-occurrences in different documents. So the highest TF-IDF words of a document may not make sense with the topic of the document.
This can be a huge setback for text retrieval as outlined by the precision and recall values of TF-IDF against other algorithms.