CS 6200: INFORMATION RETRIEVAL

SPRING 2017

INSTRUCTOR: PROF. NADA NAJI

Abhishek Jain
Prasad Tajane
Sourabh Punja

INDEX

1. INTRODUCTION:

The aim of this project was to implement various retrieval systems and evaluating their performance. We have implemented concepts such as ‘stopping’, ‘stemming’, ‘query expansion using pseudo-relevance feedback’ in our retrieval models.
Retrieval models were then analyzed using metrics such as ‘Precision’, ‘Recall’, ‘P@K | K = {5,20}’, ‘MAP’ and ‘MMR’.
We also performed the statistical test using ‘T-Test’ to enable us to reject or accept the ‘Null-Hypothesis’. We also implemented ‘snippets’ generation and ‘query highlighting’ to show the association with the retrieved document.

2. CONTRIBUTIONS:

Abhishek Jain

Indexer
BM25, TF-IDF
Evaluation Metrics
Documentation:-
Introduction and Overview
Query by Query Analysis, Results
Conclusion and Outlook

Prasad Tajane

Pseudo Relevance Feedback
Lucene Implementation Part
T-Test
Documentation:-
Implementation Details
Retrieval Model Comparisons
Bibliography

Sourabh Punja

Stopping, Stemming
Snippet Generation
Tokenizer
Documentation:-
Literature and Resources
Spreadsheet generation

3. <u>LITERATURE AND RESOURCES</u>:

BM25:
BM25 is based upon Binary Independence Model, it extends the scoring function to include document and query term weights. The extension is based on probabilistic arguments and experimental validation. BM25 has performed very well in TREC retrieval experiments and many commercial search engines do use some variation of the scoring function for BM25.

Lucene:
Apache Lucene is a free and open-source information retrieval software library. It is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

TF-IDF:
TF-IDF short for Term Frequency – Inverse Document Frequency. It is intended to reflect how important a word is to a document in a corpus. The tf-idf value is directly proportional to the number of occurrences of a word in a document but is normalized by its frequency in the corpus. This helps to adjust for the fact that some words that generally appear more frequently are normalized and do not affect the results as much.

Query Expansion:

<u>Pseudo Relevance Feedback –</u>
Pseudo Relevance Feedback is a technique to automatically expand the query. Pseudo-relevance feedback assumes that the top 'k' retrieved results for the initial query are relevant and then expands the original query to include the significant terms in those documents.
The values of 'k' and 'n' was decided empirically after doing several experimental runs. Please refer 'Implementation and Discussion' section for further details about the analysis of the experimental runs.

Python Libraries:

<u>Collections –</u>
The Collections is a python library that has implemented specialized container datatypes as an alternative for the build in containers like dictionary, list, tuple and set.
In the project 'Counter' and 'OrderedDict' has been used in the process of index creation and pseudo relevance feedback implementation.

Pickle –
Pickle is a python library that allows easy access to the data on the disk. The project makes use of ‘dump’ and ‘load’ methods to save data from python objects in the file and to load data into python object from file respectively.

Glob –
A python library that allows accessing file system with an ease. The ‘glob’ method from this library has been used in the project to access all the files from a dictionary structure in sequential manner.

Math –
Math is a python library that provides direct implementations of various mathematics functions. Project make use of methods such as ‘log’, ‘ceil’ and ‘floor’ methods.

Beautiful soup –
bs4 is a python library that provides functionalities to extract text from HTML, XML documents.

SciPy –
SciPy is a collection of open source software for scientific computing in python. This was used for ‘T-Test’ and calculating p value in it.

NLTK –
NLTK is the library for natural language processing. It provides text processing methods for sentence breaking by considering abbreviation, decimal point etc. The package nltk.tokenize provides tokenize method which splits text into sentences and returns the list of sentences in the given text. This method is used in the snippet generation and term highlighting.

Query Processing:

Stopping –
Words that appear most frequently and our very less significant in a document are called as ‘Stop Words’. Stop words are removed to improve the performance of the search engines. We can have stoplist which can be Corpus specific or we can also have Stoplist which is general for all the corpus.

Stemming –
Stemming is a process of substituting the inflected or derived words by their stems. Stems are the base or root from which the words are derived. Stemming has been implemented in the runs of BM25 and Lucene.

Snippet Generation –
As an important part of searching result presentation, snippet generation has become a prevalent method of search engines that makes the result list more informative and attractive to users. Snippets give an epitome of the result set for a query and highlight the significant terms in it. Luhn`s approach and Document-Based snippet generation were implemented to rank and display the sentences in the documents.

T-Test –

T-test is a significance test implemented to analyze the effectiveness of every query of two search engines. 'Average Precision' has been used to calculate the test statistic results for each query.

4. IMPLEMENTATION AND DISCUSSION:

Tokenizing –
In this process, each document is parsed into unigram tokens and then filtered to remove the unwanted tags or less significant part of the document(like ads). All the punctuations except for the hyphens that appear in the corpus are removed. The 'string' library of python is used for this task, which provides us with 'string.punctuation', which in turn was used to replace it with empty string. For numeric tokens, simple punctuations that appear within a digit are retained as it can have significant repercussions if removed. For example if decimal point is removed from a number, it will lose it's significance.

Implementation :

Script Name	- tokenizers.py
Input	- raw 'html' files in a directory '**cacm**'
Output	- parsed document in 'txt' format in a directory '**lib/tokenized**'

Indexing –
In this step, Unigram Inverted Index is created with the help of the parsed documents generated in step 1. Each posting is created with term as key and values as another dictionary.
In values dictionary, we maintain documents (doc-id) and frequency of the key term in that document. We also create one more dictionary and save it in a pickle file for storing doc-id along with total length of that document.

Implementation :

Script Name	- indexer.py
Input	- parsed document in 'txt' format in a dictionary '**tokenized**'
Output	- Inverted Index in the file 'Index.pickle'

Document length dictionary in the file 'DocLength.pickle'

Structure of Inverted Index:

{ DOC_ID : LEN }

DOC_ID	:	Represents the document number in the corpus.
LEN	:	Represents the length of the document.

{ TERM : { DOC_ID : TERM_FREQ } }

TERM	:	Represents each token in the corpus.
DOC_ID	:	Represents the document number in the corpus.
TERM_FREQ	:	Represents frequency of a term/token a document.

BM25 –

BM25 is used as one of the baseline retrieval models in the project. Two different cases of the BM25 model were used in the project.
The first model the value of 'R' and 'ri' is taken as '0' and 'query expansion' is then implemented on the top results of this model.
The second model takes the relevance judgment into consideration and uses appropriate values of 'R' and 'ri' from set of relevant documents provided.

Formula is as follows :

$$\sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1) f_i}{K + f_i} \cdot \frac{(k_2 + 1) q f_i}{k_2 + q f_i}$$

$$K = k_1((1 - b) + b \cdot \frac{dl}{avdl})$$

Implementation, Term Connotation and Value Calculation -

ri: Relevant documents containing word i.
    The value is calculated with the help of 'inverted index' and relevance document 'cacm.rel'
R: Total Relevant documents
    The value is calculated with the help of relevance document 'cacm.rel'
qi: Frequency of word i in the query
    This is calculated by creating a list tokens for every query and then counting term occurrences
N: Total documents in the corpus
    (3204)
K, k1, k2: parameters with values generated empirically (Please refer 'Discussion' section)
n: Number of docs containing word i
    This is calculated with the help of the 'inverted index' dictionary
dl: Document length
    This is calculated with the help of 'document length' dictionary
fi: Frequency of word i in the given document
    This is calculated with the help of the 'inverted index' dictionary
avdl: Average document length

This is calculated with the help of ‘document length’ dictionary

Discussion –

k1 controls the fluctuation of term frequency (tf) with a change in the frequency of the word (fi). A typical value of k1 is 1.2. At value 0 the term frequency weighting in the formula becomes zero and it will act as a binary model. At the larger values of k1, the term weight will increase linearly with f1.
The impact of different values of k1 is analyzed on basis of the MAP values and then the appropriate value of 1.2 is selected.

k1	k2	MAP
0.6	100	0.391613
0.8	100	0.408671
1.00	100	0.410134
1.2	100	0.419149
1.4	100	0.411212

k2 controls the weight of a ‘query term’ in the result and its value ranges from 0 to 1000. As the value of k2 increases the importance of the query terms in the ranking of result document increases. A typical value of 100 was selected as many different combination of retrieval model forms were to be used.

Script Name	-	ranking.py
Input	-	Inverted Index in the file ‘Index.pickle’
		Document length dictionary in the file ‘DocLength.pickle’
		A file contacting queries
Output	-	Ranking of the document for each query

TF-IDF –

TF-IDF is implemented as one of the baseline model in the project. In this all the documents are ranked and are represented according to their ‘tf-idf’ values.

Formula is as follows:

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$$

Term frequency is calculated by dividing frequency of a term in a document by its length.

$$idf_k = \log \frac{N}{n_k}$$

Inverse Document Frequency is calculated by taking log of division of total documents in a corpus and document containing a term k. log is taken in order to minimize the effect of large value of the division.

Implementation:

First term frequency is calculated using the inverted index and document length dictionaries for each term in the query. Inverse document frequency is then easily calculated with the help of the inverted index dictionary. Then for each term in a query the value of tf-idf is calculated. Suppose the corpus has 4 document and the query has 3 terms in it, then tf-idf calculation can be shown as follows –

	w1	**w2**	**w3**
d1	tf*idf of (d1, w1)	tf*idf of (d1, w2)	tf*idf of (d1, w3)
d2	tf*idf of (d2, w1)	tf*idf of (d2, w2)	tf*idf of (d1, w3)
d3	tf*idf of (d3, w1)	tf*idf of (d3, w2)	tf*idf of (d3, w3)
d4	tf*idf of (d4, w1)	tf*idf of (d4, w2)	tf*idf of (d4, w3)

After generation of individual values, total value of a document is measured by adding the tf*idf weight. In the above example, total value for 'document 1' is calculated as –

Tf*idf(d1) = tf*idf of (d1, w1) + tf*idf of (d1, w2) + tf*idf of (d1, w3)

The retrieved documents are then ranked based on this total term frequency value of each document.

Lucene –

The project make use of 'Lucene Simple Analyzer'. The 'search' method of the class 'IndexSearcher' is used to create an index of the corpus documents. It then uses the 'topDocs' method of class 'TopScoreDocCollector' to rank the documents according for a query.
Lucene is automated to process documents and queries directly from given location and to write the results into another directory.

Query Expansion with Pseudo Relevance Feedback –

As mentioned earlier, top k results of the initial query search are considered as relevant and words from them are used for query expansion. Original query is then modified by adding more significant words from the top retrieved documents of initial run.

Implementation Technique–

1> Run BM25 model (without relevance document information) and select top ‘k’ documents
2> Calculate and sort the terms according to their Term Frequency value in each document.
3> Add the values of individual documents to calculate total value of frequency for each term.
4> Select first ‘n’ terms from the final sorted list.

Discussion –

Value of ‘k’:

k controls the presence of significant (relevant) words in the expanded query. If the value is chosen in the range where Recall rate is always 0, then in that case the expanded query will not have any relevant words. Similarly, if the value of k is chosen in the range where Recall is always 1 then in that case probability of containing relevant words in the expanded query is very high.
For the project, value of was finalized rationally.
For all the queries the average value of rank of document where the Recall crosses ‘0.70%’ are found, that was coming as 20 and hence value of ‘k’ for the project is ‘20’.

Value of threshold ‘n’:

n controls the cardinality of the extra words that are added to expand the query. The threshold has significant impact on the effectiveness of the expanded query. Including large amount of words will decrease the importance of the original query terms. On the other hand, including only 1 word might not have any impact on the effectiveness.
For the project, value of was finalized empirically.
Impact of n is analyzed on basis of the MAP values and then the appropriate value of 4 is selected. Results of the runs at different values of n are as follows –

n	k	MAP
0	20	0.411149
1	20	0.411140
4	20	0.439781
8	20	0.420508
12	20	0.412134

Use of Third Party Tools:
Availability of wide range of tools for synonyms makes the task of searching similar meaning words very easy. However, most of the libraries and API`s are not free for public usage. Dictionaries such as WordNet from NLTK provides synonyms of a given terms. In the project for given corpus, effectiveness analysis showed that using synonyms for query expansion does not help in improving the precision and recall. Thus project makes use of ‘pseudo relevance feedback’ for the purpose of query expansion.

Snippet Generation –

Snippet is generated to provide detailed summary and highlight the significant words in a document for a query. There are several ways to highlighting words in a text, project uses bold tag <b> for this purpose.

Implementation Technique –

1> Select top 'k' documents from the ranked list of documents.
2> Sort the sentences in a document according to value of Luhn`s approach.
3> Find out the significant/relevant query terms in those sentences and add those words in the html bold tags.
4> Print the first sentence of each of the document.
5> Follow steps 1 through 4 for all the queries.

Discussion –

Value of 'k':

For the simplicity of time and space complexity of the task, small value of k is chosen. In the project k is taken as 10.

Document-Base vs Index-Base snippet generation:

Document-based model is selected because of its simple implementation and low space, time complexity. However, it has many cons in terms of Non-literal matches, code duplication and large document size.

T Test –

Significance tests are used for comparing if one retrieval model is better for given set of topics and documents in the evaluation by rejecting Null hypothesis.

Implementation Technique –

In project T-Tests are used for reporting performance of a given system with a baseline system. In general, t-test assumes that data values are sampled from normal distribution. T test is appropriate for comparing difference between means. 'Average Precision' values of the queries are used as the evaluation parameter. T-Test compares the p value obtained from two systems with the threshold value 'alpha' which is taken as 0.1 in the project.

Discussion –

Typical value of alpha is between 0.1 and 0.05.
The value of 0.1 is taken to avoid Type I error. In type I error, null hypothesis is rejected when it is in fact true. Null hypothesis is that there is no difference in the effectiveness of retrieval algorithms.
Thus, if the p value obtained from t test is less than 0.1 (alpha) then the new system is better than the baseline system.

Query by Query Analysis –
As part of the project ranking for 9 different types of system run was done.
The systems are as follows –

1> TF-IDF normal
2> BM25 normal (Baseline for all BM25 variation models)
3> BM25 stemmed
4> BM25 stopped
5> BM25 relevance feedback
6> Lucene normal (Baseline for all Lucene variation models)
7> Lucene stemmed
8> Lucene stopped
9> Lucene relevance feedback

To do the analysis randomly 15 queries were selected and then they are compared with the baseline model of same type of model.

Sample of Analysis of the different systems with its baseline:

1] Baseline Vs Stopped

1	Q0	1410	1	15.9935830609	BM25_Base
1	Q0	2379	2	15.9925140525	BM25_Base
1	Q0	1572	3	15.5049607479	BM25_Base
1	Q0	1605	4	15.1285406393	BM25_Base
1	Q0	1033	5	15.1145668138	BM25_Base

1	Q0	1410	1	29.3441785402	BM25_Stop
1	Q0	1605	2	29.3422171803	BM25_Stop
1	Q0	1572	3	28.4476802173	BM25_Stop
1	Q0	2379	4	27.7570445523	BM25_Stop
1	Q0	1033	5	27.7314061179	BM25_Stop

1	Q0	1657	1	0.33750585	Lucene_Base
1	Q0	1410	2	0.32017016	Lucene_Base
1	Q0	2319	3	0.3107554	Lucene_Base
1	Q0	1938	4	0.2908424	Lucene_Base
1	Q0	1827	5	0.2730184	Lucene_Base

1	Q0	2629	1	0.37257034	Lucene_Stop
1	Q0	2319	2	0.34464914	Lucene_Stop
1	Q0	1410	3	0.3326643	Lucene_Stop
1	Q0	2151	4	0.29704997	Lucene_Stop
1	Q0	1657	5	0.27952933	Lucene_Stop

Comments:
Query 12 - Original Query– *What articles exist which deal with TSS Time Sharing System an operating system for IBM computers*
Stopped Query – *articles exist deal TSS Time Sharing System operating system IBM computers*

<u>MODEL – BM25</u>
After applying stopping words like which and what were removed from the original text. This has effected the ranking of documents, documents such as '1605' surged up in the ranking list as these documents had more number of stop list words that were removed by the stopping process, thereby increasing the significance of the relevant words.

On the other hand, the scores of documents like '2379' decreased, since these documents had words like 'real' and 'time' in it, that were matched in the baseline model however, was removed in the stopping process, resulting in the decrease in the value of BM25 score.

MODEL – LUCENE
Similar behavior is observed in documents retrieved by the Lucene model as well. Documents like ' 2629', '2319' that had many stop words improved in the ranking as the relative weightage of the exact matching terms is increased after the stopping. Whereas the ranking of the documents like '1657', '1410' and '1938' is decreased as they had less stop words and hence even though their score remained relatively same their ranking became worse as compared to other documents whose score was increased after stopping.

Conclusion:
From the analysis of different queries, it can be inferred that Search Engine works better with the stopping for given corpus. As the stopping removes the most frequent and insignificant terms from the document that affects the ranking of the document without giving more information about the query. Thus, effectiveness score of an engine is improved after including stopping process.

2] Baseline Vs Stemmed

12	Q0	1747	1	94.2216232383	BM25_Base
12	Q0	2089	2	93.8567777801	BM25_Base
12	Q0	1959	3	92.6882632315	BM25_Base
12	Q0	2358	4	91.605617972	BM25_Base
12	Q0	2379	5	89.819999942	BM25_Base

12	Q0	3127	1	8.70586043412	BM25_Stem
12	Q0	2358	2	7.06457462517	BM25_Stem
12	Q0	1742	3	5.78279487155	BM25_Stem
12	Q0	1472	4	4.5488220769	BM25_Stem
12	Q0	3174	5	4.33711202732	BM25_Stem

12	Q0	3127	1	1.1755296	Lucene_Base
12	Q0	1461	2	0.50642985	Lucene_Base
12	Q0	3068	3	0.40902123	Lucene_Base
12	Q0	2111	4	0.3230007	Lucene_Base
12	Q0	2069	5	0.31651866	Lucene_Base

12	Q0	3127	1	1.2878323	Lucene_Stem
12	Q0	2246	2	0.76457447	Lucene_Stem
12	Q0	3196	3	0.5516708	Lucene_Stem
12	Q0	2593	4	0.37100837	Lucene_Stem
12	Q0	1461	5	0.3229165	Lucene_Stem

Comments:
Query 12 - Original Query – *portable operating systems*
After Stemming – *portabl oper system*

MODEL – BM25
After applying stemming inflected words like operate, operating are replaced by their stem, 'oper'.

After stemming ranking of documents such as ‘3127’ and ‘2358’ improved as these documents had more number of inflected words of the query term that were replaced by same stem term, thereby increasing number of exact matching words in a document.
Conversely, the scores of documents like ‘1747’ and ‘2089’ decreased, since these documents had words like ‘real’ and ‘time’ in it, that were matched in the baseline model however, was removed in the stemming process. Also, these documents had comparatively less variants of stem words in it, thereby even though their ranking value did not fall, the improvement in score of other documents decreased their ranking.

<u>MODEL – LUCENE</u>

Just like the analysis of BM25 model, in Lucene ranking of the documents with different variants of stem class is improved drastically. Documents like ‘2950’, ‘1410’ that had more occurrences of word ‘system’ in it and hence after stemming its ranking is increased. Whereas the ranking of the documents like ‘2812’, ‘1892’ and ‘1805’ is decreased as they had variants of query term stem words and hence even though their ranking score remained relatively same the score as compared to other documents whose score was increased after stopping.

Conclusion:
Depending on the analysis, it is safe to infer that appropriate amount of stemming in the searching process is beneficial for a given corpus. In general, for the retrieval models based on Boolean Retrieval Model, performance is mostly increased after stemming. Since all such models give higher importance to the documents that has direct matching of query term and stemming increases the percentage of the matching terms in a document, the effectiveness is improved after stemming.

3] Baseline vs Feedback

12	Q0	1747	1	94.2216232383	BM25_Base
12	Q0	2089	2	93.8567777801	BM25_Base
12	Q0	1959	3	92.6882632315	BM25_Base
12	Q0	2358	4	91.605617972	BM25_Base
12	Q0	2379	5	89.819999942	BM25_Base

12	Q0	1747	1	94.2216232383	BM25_Feed
12	Q0	2080	2	93.8567777801	BM25_ Feed
12	Q0	2317	3	92.6882632315	BM25_ Feed
12	Q0	1959	4	91.605617972	BM25_ Feed
12	Q0	2358	5	89.819999942	BM25_ Feed

12	Q0	3127	1	1.1755296	Lucene_Base
12	Q0	1461	2	0.50642985	Lucene_Base
12	Q0	3068	3	0.40902123	Lucene_Base
12	Q0	2111	4	0.3230007	Lucene_Base
12	Q0	2069	5	0.31651866	Lucene_Base

12	Q0	3127	1	0.39532316	Lucene_Feed
12	Q0	1523	2	0.22908121	Lucene_Feed
12	Q0	1951	3	0.2156378	Lucene_Feed
12	Q0	2372	4	0.21252857	Lucene_Feed
12	Q0	1461	5	0.19706321	Lucene_Feed

Comments:

Query 12 - Original Query – *portable operating systems*

After Stemming – *portable operating systems four prefix woods ca631109* (LUCENE)

– *portable operating systems four prefix woods ca631106* (BM25)

MODEL – BM25

In query expansion, original query is modified to include four new possible relevant terms. As compared to other two models there is very slight change in the ranking after the query expansion. Most of the change is in the orderly manner, top 4 documents out of 5 have remained the same. After query expansion, scores of the documents that have newly added query terms is improved as compared to other documents. Hence document '2317' that was originally on rank 8 is now jumped to the 3rd rank. And the ranks of documents '2358', '1959' and '2379' slightly became worse.

MODEL – LUCENE

As there is slight change in the top 20 documents retrieved by both the search engines, the pseudo relevance feedback created two slightly different queries after the expansion. However, behavior of the result is same, there is very less change in the top 5 documents before and after query expansion. Ranking of the document '1877' is improved a lot as it has many occurrences of term 'woods' and 'four' in it. On the other hand, since '2570' had less number of new query terms its rank is degraded slightly as compared to others.

Conclusion:

Based on the analysis of results after query expansion, we can conclude that including optimum relevant words in the original query, helps the good search engine in finding and ranking relevant documents. However, in the case of bad search engines, as the initial query would return non-relevant documents in the top k results, the performance in worsened after query expansion. Effectiveness of query expansion is highly dependent upon the number of new terms addition in the original query. Including many words decreases the overall effectivity, whereas including only one term might not affect the effectivity at all.

3. RESULTS:

Below are the various effectiveness metrics for all the 7 runs that we did.

Retrieval Model	MAP	MRR	P@5	P@20
BM25_feedback	0.439684822	0.731650641	0.53461538	0.31923076
BM25	0.416134589	0.669123932	0.52307692	0.28653846
BM25_stoplist	0.432304919	0.677700692	0.50769231	0.29519231
Lucene	0.413683042	0.683313123	0.36538462	0.19903846
Lucene_feedback	0.477081935	0.824839744	0.57373733	0.27337373
Lucene_stoplist	0.438220246	0.708040428	0.2563633	0.16727692
TF_IDF	0.379331422	0.634020147	0.27291308	0.15828277

All the other tables are provided in Project under results directory.

Retrieval Model Run results:
Lucene results: results/Lucene/
Lucene stemmed: results/Lucene_stemmed/
Lucene stoplist: results/Lucene_stoplist/
Lucene feedback: results/Lucene_feedback/
Tf-idf: results/tf-idf/
BM25: results/BM25/
BM25 stemmed: results/BM25_stemmed/
BM25 stoplist: results/BM25_stoplist/
BM25 feedback: results/BM25_feedback/

T-Test
t-test: results/ t-test-scores.txt

Snippet Generation and query highlighting (BM25 results):
Snippets for all the queries is inside folder - results/SnippitResults/
Example: results_1.txt - Snippet for first 10 Ranked docs for query ID 1.

Evaluation:
Lucene: results/Lucene_evaluation.txt
Lucene stoplist: results/Lucene_stoplist_evaluation.txt
Lucene feedback: results/Lucene_feedback_evaluation.txt
Tf-idf: results/tf-idf_evaluation.txt
BM25: results/BM25_evaluation.txt
BM25 stoplist: results/BM25_stoplist_evaluation.txt
BM25 feedback: results/BM25_feedback_evaluation.txt

All the above files contain MAP and MRR values for each run along with Precision and Recall values at each rank and for each queries.
P@K, K =5, 10 are also added below each query for each run.

6. CONCLUSION AND OUTLOOKS:

CONCLUSION:

We have compared the Lucene, tf-idf and BM25 on the basis of mono-lingual runs for English. We were able to infer that Lucene outperforms BM25 marginally. Query expansion increases performance of both Lucene and BM25. Lucene with all variants(feedback, stoplist) performed better than BM25. At the same time we concluded tf-idf performs satisfactorily but it still lacks behind both Lucene and BM25.
We also noticed that performance of BM25 decreases when long queries are provided.
We were able to conclude the above by looking at both MRR and MAP values for all the runs.

OUTLOOK:

- We can integrate User Interface with our Retrieval models.
- Use of word dictionary to find synonyms for query expansion in a better way.
- Use of better indexer such as Solr and Elastic search.
- Add stemming techniques like porter stemmer while indexing.
- Use of query logs to improve the performance of our retrieval mode.
- Use of fields or extends while creating index in order to add weightage to the terms in particular field of the document.

7. BIBILIOGRAPHY:

1. Bast, H., Chitea, A., Suchanek, F., and Weber, I. 2007a. ESTER: Efficient search on text, entities, and relations. In Proceedings of the 30th Annual International Conference on Research and Development in Information Retrieval (SIGIR). ACM, New York, NY, 671–678.
2. https://en.wikipedia.org/wiki/Apache_Lucene, Wikipedia, 18th April 2017.
3. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html#scipy.stats.ttest_rel, Scipy.org, 17th April 2017.
4. https://nlp.stanford.edu/IR-book/html/htmledition/relevance-feedback-and-query-expansion-1.html. 18th April 2017.
5. https://researchrundowns.files.wordpress.com/2009/07/rrsignificancettest_71709.pdf, 15th April 2017.
6. J. J. Rocchio. Relevance feedback in information retrieval, pages 313–323. Prentice-Hall Inc., 1971.
7. Lecture Notes by Professor. Nada Naji
8. Search Engines: Information Retrieval in Practice – Croft
9. White, R. W., Ruthven, I., and Jose, J. M. 2002. Finding relevant documents using top ranking sentences: An evaluation of two alternative schemes. In Proceedings of the 25th Annual International Conference on Research and Development in Information Retrieval (SIGIR). ACM, New York, NY, 57–64.