**Project Members:** Fibin Francis Assissi
Jatin Taneja
Sugandha Kher

**Course Name :** Information Retrieval

**Semester :** Fall 2016

**Instructor :** Prof. Nada Naji

## Project Description

In this project, we have worked on creating an information retrieval system using various retrieval models (cosine similarity, BM25, tf-idf and Lucene) with the document corpus provided by professor. We have also implemented query expansion using pseudo relevance feedback and also compared the results with stopped and stemmed queries. At the end, we have evaluated our different retrieval outputs using MAP and MRR techniques.

Work division:

Fibin’s Contribution

- Cosine similarity
- Tf-idf
- Pseudo implementation with stopping
- Phase 2 – evaluation

Sugandha’s Contribution

- Read me.txt
- Project Report
- Lucene implementation
- Result combiner

Jatin’s Contribution

- BM25 implementation
- Pseudo implementation
- BM25 with stopping
- BM25 using stemmed corpus

## <u>Literature and Resources:</u>

## Phase1: Indexing and Retrieval:

### Task1:

For our retrieval models, we reused our Cosine Similarity retrieval model and Lucene.

**BM25** is based on binary independence model.
For BM25 the formula we used: (Taken from Search Engines: Information Retrieval in Practice by Croft, Metzler and Strohman: http://ciir.cs.umass.edu/irbook/)
)

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

Where $k_1$, $k_2$ and K are parameters whose values are set empirically.
Typically value for $k_1$ is 1.2 and $k_2$ ranges from 0 to 1000, in our retrieval model we took $k_1$ as 1.2 and $k_2$ as 500.

Second model we implemented is **tf-idf**. (Concept understood from https://www.tfidf.com)

1) Find the term frequency(tf) of each term in the document.
2) Normalise it using the total number of terms in the document
3) Inverse document frequency:
   Idf(term) = log(float (len (allDocuments)) / numDocuments with this Term)
4) Weight of a term = tf*idf
5) Score of a document is the sum of weights of query terms present in that document.
6) The documents are scored based on the score obtained

For **Cosine Similarity**

Reference: https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/

1)Find the term frequency(tf) of each term in the document

2)Find the normalised term frequency of the terms which is equal to tf divided by no. of terms in the document

3) Idf(term) = 1.0 + log(float(len(allDocuments)) / numDocumentsWithThisTerm)

4) Weight of a term=tf*idf

5) Both document vector and query vectors are created and the similarity between the vectors are found out using cosine similarity:

Cosine Similarity (Query, Document1) = Dot product(Query, Document1) / ||Query|| * ||Document1||

Here we are calculating the score for each document based on the query.

Dot product (Query, Document1) is the sum of product of weight of a term in the document and its weight in the query for all the terms.

||Document1|| is the magnitude of document vector

||Query|| is the magnitude of query vector

**SCORING FUNCTION USED BY LUCENE**

In case of Lucene, we are inputting the corpus . The parsing and scoring is done using the algorithms already defined in Lucene. For scoring, Lucene combines Boolean model (BM) of Information Retrieval with Vector Space Model (VSM) of Information Retrieval.

Cosine Similarity (q, d) = V(q). V(d) / |V(q)| |V(d)|

Where V(q) · V(d) is the dot product of the weighted vectors, and |V(q)| and |V(d)| are their Euclidean norms.

The weights of term in the vector is calculated as follows:

tf (t in d) correlates to the term's frequency, defined as the number of times term t appears in the currently scored document d.. The default computation for tf (t in d) in DefaultSimilarity is:

tf (t in d) = frequency $^{1/2}$

idf(t) stands for Inverse Document Frequency. This value correlates to the inverse of docFreq (the number of documents in which the term t appears). This means rarer terms give higher contribution to the total score. idf(t) appears for t in both the query and the document, hence it is squared in the equation. The default computation for idf(t) in DefaultSimilarity is:

idf(t) = 1 + log (numDocs / docFreq+1)

The default computation in DefaultSimilarity produces a Euclidean norm:
queryNorm(g) = queryNorm (sumOfSquaredWeights) = 1 / sumOfSquaredWeights

## Task2:

For query expansion, we used pseudo relevance feedback technique.
For this, we ran the BM25 using the original query and got the top 100 results. Out of this, we filtered out the top 10 documents and assumed that these are the relevant documents. The top frequent terms, excluding the stop words, are taken and it is used to the expand the query.

For eg: For the query, “what articles exist which deal with tss, time sharing system an operating system for ibm computers?”,
The **expansion terms** we obtained were :
performance developed design tutorial simulation display search retrieval operations interarrival data distribution communication network time-sharing.

## Task3:

In task3, we took the BM25 as base search engine. Using the stop words given, common_words.txt, we removed the common words while indexing the corpus. BM25 was used to rank the documents.

In task3 B, we used the stemmed version of the corpus and query and ran the BM25 retrieval model to rank the documents.

## Phase2: Evaluation

For all the seven runs, we performed the evaluation using the relevant documents provided to us. For each query, we calculated the precision and recall values using:

$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$$

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|}$$

Once we got this information, we calculated the mean average precision(MAP) and mean reciprocal rank(MRR).

## IMPLEMENTATION & OBSERVATION:

In Cosine Similarity measure we tried 2 different methods for implementing the formula, one using site,
https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/

Using cosine similarity formula as:

Cosine Similarity (d1, d2) = Dot product (d1, d2) / ||d1|| * ||d2||

Dot product (d1, d2) = d1[0] * d2[0] + d1[1] * … * d1[n] * d2[n]
||d1|| = square root (d1[0]$^2$ + d1[1]$^2$ + … + d1[n]$^2$)
||d2|| = square root (d1[0]$^2$ + d1[1]$^2$ + … + d1[n]$^2$)

We used two approaches in **Cosine similarity**. In the first approach, we normalised the denominator of the cosine similarity formula by taking all the terms in the documents. In our second approach we took only the terms in the query to normalise the denominator. Since the second approach was giving better results, we carried forward the approach where we took the query terms to normalize.

**Top 5 Results of Cosine Similarity for Q1:**
Results after taking total words in the document to normalize the denominator in Cosine Similarity formula.

```
CACM-1591 1 0.200544279326
CACM-1033 2 0.178576068885
CACM-2542 3 0.171556342261
CACM-1519 4 0.166745872741
CACM-1680 5 0.16317130098
```

Whereas on taking query words to normalize the denominator we get results as:
Final result

```
CACM-1410 1 0.636555216239
CACM-1046 2 0.572388333005
CACM-1698 3 0.568103542632
CACM-2947 4 0.549628104412
CACM-1506 5 0.547936673554
```

On the basis of reciprocal rank, CACM-1410 is a relevant document which is coming as the top result in our second approach. Hence, we take this approach to measure cosine similarity.

For **BM25**, we tried different values of b ranging from 0 to 1 and k2 which ranges from 0 to 1000. Upon taking k2 as 500 and b as 1, we got the results in which our relevant document 1410 was coming on 7th rank but on taking b as 0.75 and k2 as 100, the relevant document 1410 shifted to 4th rank, which according to Reciprocal Rank which would be incorrect, since the lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined.

**Top 5 Results of BM25 for Q1:**
On taking b as 1 and k as 500
1680 1 9.3760522355
2319 2 8.94587362091
2371 3 8.94579413578
1236 4 8.78018991948
1168 5 8.72134862702

Taking b as 0.75 and k2 as 500
2319 1 9.86481989693
1680 2 9.72607004789
1236 3 9.08566397854
1410 4 8.96081225322
2371 5 8.81367981145

Taking b as 0.75 and k2 as 100
2319 1 9.83499435029
1680 2 9.69123838624
1236 3 9.05707365678
1410 4 8.93717960334
2371 5 8.77743879692

Since by taking b as 0.75 and k2 as 100, the document 1410 which is a relevant one is coming at the 4th position, so we take the approach with b=0.75 and k2=100.

In **tf.idf** a simple approach was followed wherein, we calculated the term frequency for all the terms per document and then then took out the normalised term frequency by dividing term frequency by total number of terms in particular document.
Then calculate the idf
Idf = log N/ $n_k$ where N is the total number of documents in corpus and $n_k$ is the number of documents where term occurs.
Calculate the weight using tf*idf for the documents as well as for the queries.
Here also initially, we tried with normal term frequency approach but we found out that reality each document will be of different size. On a large document the frequency of the terms will be much higher than the smaller ones. Hence we need to **normalize** the document based on its size. Otherwise the documents of larger length will be having more score which is not always right.

**Top 5 Results of tf-idf for Q1:**

CACM-2371 1 0.305168830315
CACM-1591 2 0.299880050314
CACM-1033 3 0.291805418555

CACM-1519 4 0.285964768691
CACM-1304 5 0.266288027745

Results of the tf-idf retrieval system takes simple the term frequency and the document frequency and calculated tf*idf score for documents.

In Lucene, we used the concepts cited at:
https://lucene.apache.org/core/2_9_4/api/all/org/apache/lucene/search/Similarity.html

Lucene combines Boolean model (BM) and Vector Space Model (VSM). In VSM, documents and queries are represented as weighted vectors in a multi-dimensional space, where each distinct index term is a dimension, and weights are tf.idf values.

VSM does not require weights to be tf.idf values but the tf.idf values are believed to produce search results of high quality, and so Lucene uses tf.idf

**For Task2**, using the run from BM25 we performed query expansion using pseudo-relevance,
We are reformulating the query by adding terms. In our approach, we ran an original query using BM25 retrieval method. Then, related terms are extracted from top 5 documents. We are taking 3 words from each document.

For this approach, we tried **different combinations of number of words and documents and got the optimum result for top 3 words from top 10 documents.**

Observation results:
Taking 1 word from each document:
    Doc #1572, at 97th position
    Doc #1410, at position 4th

Taking 5 words from each document:
    Doc #1605, at position 37th

Taking 3 words from each document:
    Doc #1572, at 19th position
    Doc #1410, at position 1,
        therefore, Reciprocal rank increased, recall increased
    Doc #1605, at position 20th
    Similarity score improved efficiently to 26.84 which was 8.17 before taking 5 words from each document.

**Results for Pseudo-relevance:**

Case 1:
Taking 10 top documents and 3 most frequent words from each document:
1410 1 28.1957921664
1680 2 24.1344962859
3025 3 23.003652679
1750 4 21.4841937177
2371 5 20.2388846723

**Observation**: Better scores and better results

Case 2:

Taking 5 top documents and 1 most frequent word from each document:

1410 1 21.5352739427
1680 2 13.7295950175
2371 3 11.6226071628
2319 4 10.9805107712
1236 5 10.5747088717

**Observation**: Scores are lesser and lesser number of relevant documents in top 100 results.

Case 3:

Taking 5 top documents and 5 most frequent word from each document:

1410 1 32.8569885643
2371 2 27.9724258879
1680 3 20.3444028742
2951 4 16.4704709489
1236 5 16.2623335671

**Observation**: Relevant documents ranked lower

Case 4:

Taking 5 top documents and 3 most frequent words from each document:

1410 1 26.8420581312
1680 2 19.1894345712
2371 3 18.1518815441
1236 4 12.6071621183
2951 5 11.9733665547
Observation: Better results. Both 10 docs and 5 docs have almost same results but 10 docs is preferred because it gave slightly better MAP.

As described above the taking 10 top documents and among that taking 3 most frequent words from each document, gives us better results such as relevant document is obtained at position 1 which increases the reciprocal rank. Also recall is increased, so we take 3 words per document for query expansion.

**Query by Query Analysis for Stemmed and Non-stemmed**:

| Rel_Doc | NonStem(Rank) | Stem(Rank) |
|---|---|---|
| 1523 | 48 | 0 |
| 2080 | 80 | 90 |
| 2246 | 17 | 2 |
| 2629 | 26 | 24 |
| 3127 | 70 | 1 |

We observe that the relevant documents are present at better ranks in case of stemmed version. For eg: 3127 is a relevant document which is present at 70 th position for non-stemmed version while its at rank 1 for stemmed version.

For **evaluation**, we performed 7 runs, namely, Cosine Similarity, BM25, tf-idf, Lucene, pseudo-relevance, BM25 after stopping, indexing stemmed corpus and the outputs were compared.

## CONCLUSION:

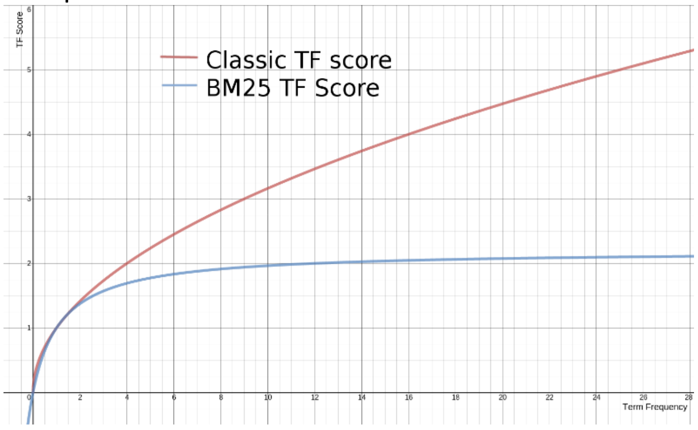
Based on the results we obtained, BM25 is giving the better results.

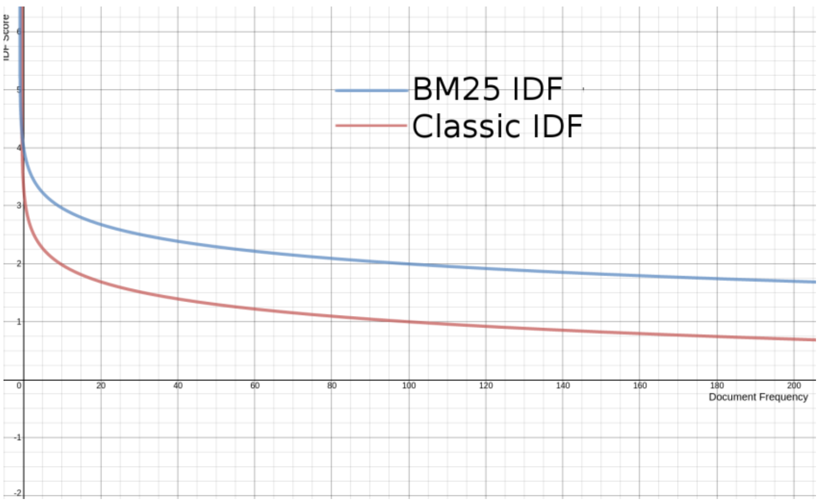
```
BM25

Mean Average Precision: 0.49
Mean Reciprocal Rank: 0.73

Cosine Simlarity

Mean Average Precision: 0.33
Mean Reciprocal Rank: 0.52

Lucene

Mean Average Precision: 0.42
Mean Reciprocal Rank: 0.69

Pseudo Relevance output

Mean Average Precision: 0.49
Mean Reciprocal Rank: 0.79
```

From the above data, BM25 is having better MRR (Mean Reciprocal Rank) and MAP (Mean Average Precision). This result is obtained on an average of 64 queries.
Actually, output of IR system that uses query expansion using pseudo relevance feedback is giving us the best results. In our retrieval system, we are using BM25 for pseudo relevance feedback.

BM25 is considered better. It uses probabilistic model. A relevance score, according to probabilistic information retrieval, ought to reflect the probability a user will consider the result relevant. Also we are provided with relevant set which we used in BM25 computation. That also helped improve the results.

Comparison between BM25 and tf-idf:





We get more flattened graphs for BM25.

OUTLOOK:

According to our observations, query expansion technique that uses BM25 was giving us the best results. BM25 is undoubtedly better retrieval model and it is being used in the latest versions of Lucene. So one scope of improvement would be to include the newer version of Lucene that uses BM25.
Another scope of improvement would be to use the better query expansion technique that is **query log.** We get better expanded query and hence better results. This takes more of a user relevance into account. It is very difficult to implement but it will definitely improve the results.

## BIBLIOGRAPHY:

https://www.tfidf.com

https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/

https://lucene.apache.org/core/2_9_4/api/all/org/apache/lucene/search/Similarity.html

Search Engines: Information Retrieval in Practice
by Croft, Metzler and Strohman: http://ciir.cs.umass.edu/irbook/

Introduction to Information Retrieval by Manning, Raghavan,
and Schütze (2008) http://www-nlp.stanford.edu/IR-book/

1- Lucene in Action (2nd Edition). McCandless, Hatcher, and Gospodnetić. (Manning)
Fast Generation of Result Snippets in Web Search by Felix Geller