***Information Retrieval***

CS 6200: Information Retrieval

Northeastern University

Spring 2018, Prof. Nada Naji

***TEAM MEMBERS:***

*Kevin Shah*

*Manoj Venkatesan*

*Twisha Vyas*

**ii. Introduction:**

The aim of the project is to design a retrieval system using BM25, Lucene, QLM and TF-IDF models to evaluate their effectiveness.

The project is divided into 4 parts including extra credit as explained below:

(The contribution of each team member is written beside the steps)

1. **Phase 1: Indexing and Retrieval**

It comprises of 3 Tasks:

1. Task 1:
2. Step 1-Corpus Generation: It takes raw HTML files (corpus) as input and returns same number of files with each file tokenized by case-folding and removing punctuations.
3. Step 2-Index Generation: It takes the document generated in step 1and generates an inverted index of the form Term 🡪[(Document\_id, Term Frequency)].
4. Step 3-Query Cleaning: The queries provided are processed in the same way as corpus and written into a file.
5. Step 4-Retrieval Models: It implements 4 retrieval models by using the cleaned queries and index generated in prev. steps.
6. BM25:

BM25 considering no relevance

BM25 Considering relevance

1. Lucene
2. Smoothed Query Likelihood
3. TF-IDF
4. Task 2: Pseudo relevance feedback is used for query enrichment which re-computes scores calculated for BM25 Model by considering no relevance. The output contains top 100 documents after expanding the query with most frequent terms in the top documents from BM25 Model (No relevance).
5. Task 3:
6. Part A: A new corpus is created by removing the stop words provided. The stop words are also removed from the queries and three baseline runs – BM25, Lucene, Smoothed Query Likelihood are used to rank the documents.
7. Step 1: Corpus Generation
8. Step 2: Index Generation
9. Step 3: Query Cleaning
10. Step 4: Retrieval Models.
11. Part B: Stemming is performed on the terms present in corpus and the queries. Three baseline runs – BM25, Lucene, Smoothed Query Likelihood are used to rank the documents.

1)Step 1: Corpus Generation

2)Step 2: Index Generation

3)Step 3: Query Cleaning

4)Step 4: Retrieval Models.

1. **Phase 2: Displaying Results:**

Snippet generation and query term highlighting performed on results of Lucene model.

1. **Phase 3: Evaluation:**

It implements various evaluation metrics like MAP, MRR, P@K, Precision and Recall on the 8 runs

1. **Extra-credit:**

This phase consists of implementing an error-generator model and soft matching query handler which introduces noise to the query terms and then will reduce the impact of noise on the effectiveness of the retrieved results.

* Part A: Query error generator
* Part B: Soft matching query handler

**iii) Literature and Resources:**

Following approaches were observed for each of the below mentioned tasks:

* tf-idf measure: For calculating tf-idf for each document, we are using the value of normalized term frequency for the document multiplied by its inverse document frequency and summing this score for each query term.
* BM25 Model: For BM25 model we are using the formula from the book *‘Search Engines: Information Retrieval in Practice’* [1] where the values of ‘K’, ‘k1’ and ‘k2’ are chosen as per TREC standards.
* Lucene: We have used standard Lucene library (4.7.2 version) with its default retrieval model and standard analyzer for indexing and retrieval operations.
* Query Enrichment: The query enrichment run for our project has been done using Pseudo Relevance Feedback technique on the BM25 model without relevance.
* Stopping: The standard stop list – ‘common\_words.txt’ has been used to perform stopping. Any word appearing in the above-mentioned stop list has not been indexed and then 3 baseline models - Lucene, BM25 and Smoothed Query Likelihood are executed.
  + - Stemming: We have performed stemming with the help of the stemmed version of the corpus ‘cacm\_stem.query’ and query ‘cacm\_stem.query’ and run 3 baseline models -Lucene, BM25 and Smoothed Query Likelihood.
    - Snippet Generation: Generates snippets for top 5 ranked documents from Lucene model by computing the significance factor of each sentence in the documents and ranking those sentences in decreasing order by using the formula in book [1].
    - Precision & Recall:
      * Precision = |Relevant ∩ Retrieved| / |Retrieved|
      * Recall = |Relevant ∩ Retrieved| / |Relevant|
    - MAP:
      * MAP = Σ Average Precision / Number of Queries
    - MRR: Reciprocal rank(RR) is reciprocal of the rank at which the first relevant document is retrieved.
      * MRR = Σ RR / Number of Queries
    - P@K: P@K is calculated as the precision obtained at rank K. P@K for K = 5 and 20 is evaluated.

**iv) Implementation and Discussion:**

1) Phase 1/Task 1/Step 1 – Corpus Generation:

While de-punctuating documents in corpus, occurrences of ‘-’, ‘:’, ‘,’ or ‘.’ within digits are retained and occurrence of ‘$’ before a number is retained. Also, same processing is performed on queries for consistency in index.

2) Phase 1/Task 1/Step 3- Query cleaning:

* Remove the tags <doc> and <doc\_no> from cacm.query.text.
* We then break the query into a dictionary with the mapping Query\_id(obtained form <doc\_no>) 🡪 Query(from <doc>).
* Perform casefolding on the terms in the query
* Remove punctuations and extra spaces from the query except ‘-‘. Keep ‘.’ between digits.
* Write the clean queries to a file

3) Phase 1/Task 1/Step 4- Retrieval Models:

BM25:

BM25 with relevance (using cacm.rel.txt, we got the relevant documents for each query and plugged in values for R and r using the same) and without using relevance is implemented.

The formula from the book[1] is used,

Where:

r = no. of relevant documents containing the query term

R = no. of relevant documents for that query

N = total no. of documents in the collection

q = frequency of the query term in the query

K = k1(b\*L+(1-b))

f = term frequency in that document  
n = total number of documents in which the term appears  
L = doc length / average doc length

Lucene:

The Lucene model uses Standard Analyzer for tokenizing/analyzing text and the default retrieval model for Lucene 4.7.2.

Smoothed Query Likelihood Model:

Smoothing parameter (λ) is set to 0.35 for computing scores for QLM. The formula from the book[1] is used,

where:

λ’s value set to 0.35 as per TREC standards

D: is the document length

C: corpus length

tf: term frequency in the document

cf: frequency of the term in the corpus

TF-IDF

For TF-IDF calculations, the following formulae is used:-) Normalized\_tf = term freq. in the document/ length of document

-) idf = 1+math.log (total number of documents in the corpus /doc freq. +1)

-) document tf-idf score=Normalized\_tf\*idf

Value 1 is added to the denominator to prevent it from becoming 0 when the term does not appear in any of the document in corpus. We are adding 1 to the log values to prevent entire “idf” value from becoming 0.

4) Phase 1 Task 2 (Pseudo Relevance Feedback):

In pseudo-relevance feedback model, words occurring frequently in top documents ranked by a model are added in query to perform query expansion and re-rank the documents using this expanded query. (CMS Page 208).

Considering top “k” frequent words from each of the top “n” documents. After 4 iterations of trial and error we found that k=5 and n=5 worked best for query expansion. Thus, 5\*5=25 words are added to each query, thereby performing query expansion. While selecting these 25 terms we do not consider the stop words that are given to us in common\_words file and do not consider the terms already present in the query before expansion. Run the bm25 (without relevance) model for these expanded queries.

5) Phase 2 (Snippet Generation):

- In snippet generation, top 5 documents obtained from Lucene model is considered.

- Documents are fetched and a mapping document\_id  [Sentences] is generated. Document is broken into sentences by splitting with ‘.’ or ‘\n’ depending on its location.

- Significance factor is calculated for each sentence by calculating the first and last significant words in the sentence, counting the number of significant words within this window.

            Significance factor = square of the number of significant words in the window [1]

                                                            Total number of words

We have considered the terms in the query to be significant words.

We are displaying an entire sentence as a snippet. The snippet for each document contains two sentences with the highest scores. After the top two scores, the fall in the score was high and hence, we did not consider those sentences.

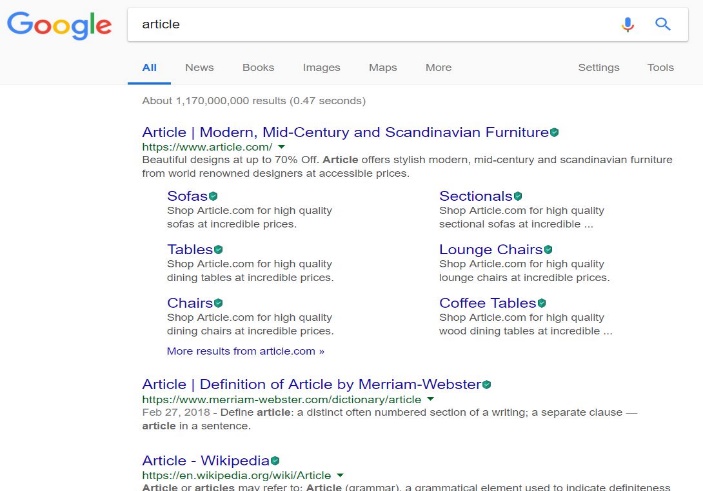
We also experimented by starting and ending the sentence with significant words, but the sentences ended up making no sense and hence, we went ahead with the former approach.

We also ran tests for multiple values of ‘k’ value such that, in a snippet, no the significant words should have more than k non-significant words between them. A very few documents had such sentences and hence, we decided to discard this approach.

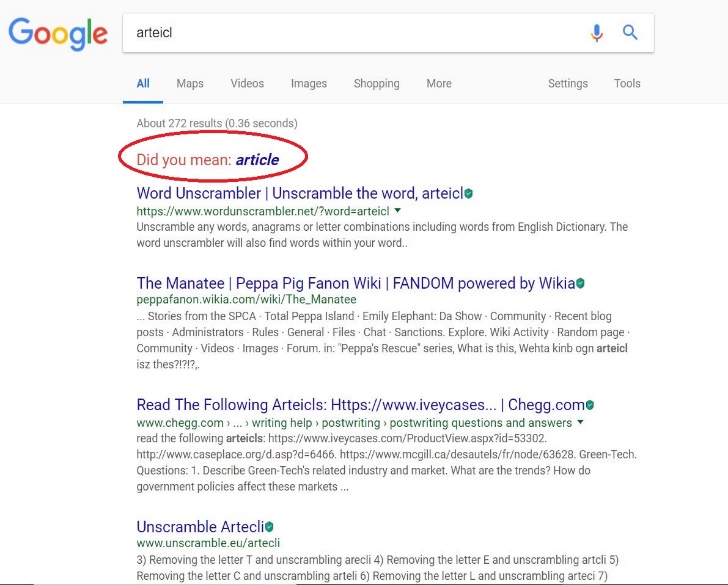
6) Extra Credit:

According to the reference in the book [1], studies have shown that 80% or more of spelling errors are caused within an edit distance of one or two. Based on this assumption, most of the search engine’s word suggestion (“did you mean?”) were implemented. It can be observed as follows,

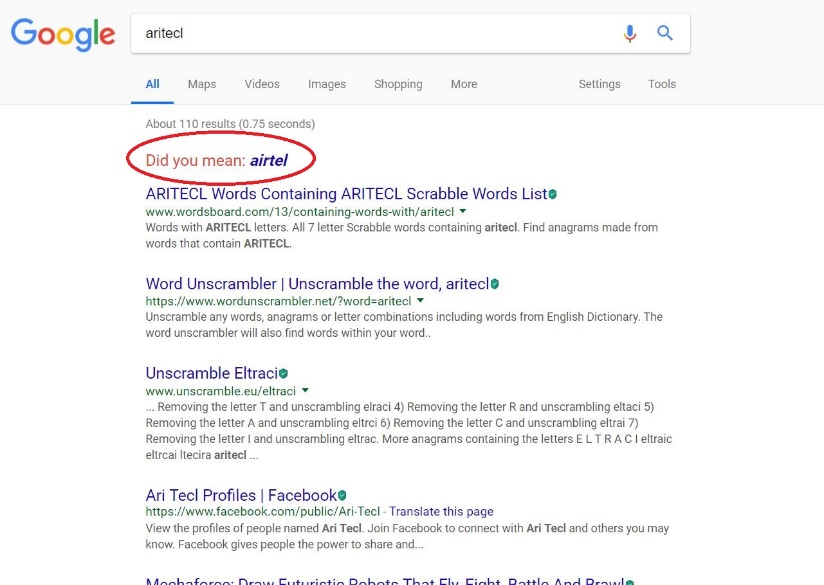
For the search term “article”, the following is the result



“arteicl” is edit distance 1 for “article”, Google correctly predicts the word as follows



“aritecl” is edit distance 2 for “article” , Google’s suggestion is not as expected



This proves that the word suggestion is designed to return more robust result for terms with edit distance less than or equal to two when compared to more edit distance, as 80% of errors in the queries would be of edit distance 2.

Part A:

Based on the above observations, the error generator model is implemented as follows,

1. Read the cleaned queries from ‘cleanQueries.txt’
2. Parse and store each query in a list.
3. For each query, identify the top 40% of the query terms that needs to be affected.
4. Introduce noise to the selected terms by shuffling the non-boundary characters.
5. Reconstruct the query with affected terms and write the query into the file ‘ErrorQueries.txt’

Part B:

The soft query matching is inspired from Peter Norvig’s article on designing the spelling corrector.

The probability theory is used to find the most suitable word for the misspelled word. It is because there is no definite way to identify the original word (eg.whether "lates" must be exchanged with "lattes" or "latest" or "late"). We are finding the suitable word , out of all possible suitable words, that maximizes the probability that must be the desired word, given the original misspelled word :

Equivalent to Baye’s theorem,

On factoring out , we get

The main 5 parts of the model that takes care of the above formula are as follows:

1. Selection Mechanism:
   1. It is used to select the most suitable word based on highest probabilities.
   2. If both the probabilities are same, then the one with higher frequency is chosen.
   3. A dictionary of = {mis-spelled word: suitable word} is updated and finally written to a file “replacements.txt”
2. Candidate Model:
   1. Gives all possible corrections for the word
   2. It uses hamming distance to consider a word that can be generated by deletion, insertion, replacement and transposition, vowel swapping, word frequency model based suggestions.
   3. known words from dictionary is used to reduce the generated list.
3. Language Model:
   1. To generate a database for finding the probability of the word in the corpus.
4. Error Model:
   1. It takes care of calculating the probability that word would be typed in a query when the user thinks of word .
   2. The input to the model will be a non-empty list of possible words sorted in the order of precedence.
5. Query generator Model (modify\_queries.py):
   1. “ErrorQueries.txt” and “replacement.txt” files are parsed and stored in a container.
   2. Misspelled words are identified in each query and replaced with the suitable words.
   3. The query is regenerated and stored in “correctQueries.txt”

**Query by Query Analysis:**

Three queries we chose are:

1. portabl oper system (Stemmed Version)

portable operating systems (Non-Stemmed Version-Guess)

1. parallel algorithm (Stemmed Version)’

Parallel algorithms (Non-Stemmed Version-Guess)

1. appli stochast process (Stemmed Version)

Applied stochastic processes (Non-Stemmed Version-Guess)

In the first query, the query terms ‘portable’, ‘operating’ and ‘systems’ are probably stemmed to ‘portabl’, ‘oper’ and ‘system’ respectively. In the second query, only the term ‘algorithms’ is probably stemmed to ‘algorithm’. The term ‘Parallel’ is not stemmed. In the third query, the query terms ‘Applied’, ‘stochastic’ and ‘processes’ are probably stemmed to ‘appli’, ‘stochast’ and ‘process’ respectively.

In case of stemming, all the words that belong to the same stem class get stemmed to that one stem root. Eg., in the first query, the words ‘operation’, ‘operable’, ‘operating’,’operand’,etc. all get stemmed to single stem ‘oper’. Similarly, in the third query, the words ‘application’,’applied’,’applying’,’applicable’,etc get stemmed to single stem ‘appli’. Hence, documents having words stemming down to a single word would score lesser, since previously, individual unique terms would be considered and now only a single stemmed word for all those stem class words would be considered.

If we consider the first query, only six documents match when we compare the top 20 documents obtained by the bm25 model with stemming and of that without stemming, namely, ‘CACM-3127’, ‘CACM-2541’,’CACM-2246’,’CACM-3068’,’CACM-2740’ and ‘CACM-1750’. Reason for lesser matches is that stemming would have had a high impact on the query terms.

If we do the same analysis for the second query between the same two versions of the aforementioned model, 14 documents match, namely ‘CACM-2714’, ‘CACM-2973’, ‘CACM-0950’, ‘CACM-2433’, ‘CACM-2785’, ‘CACM-2266’, ‘CACM-1262’, ‘CACM-2700’, ‘CACM-2685’, ‘CACM-3156’, ‘CACM-1158’, ‘CACM-3075’, ‘CACM-1828’ and ‘CACM-2289’. Reason for higher number of matches is that stemming of just one query term ‘algorithms’ to ‘algorithm’ has a very small impact on the query terms.

If we do the same analysis for the second query between the same two versions of the aforementioned model, 7 documents match, namely,’CACM-1696’, ‘CACM-0268’, ‘CACM-1410’, ‘CACM-2882’, ‘CACM-1540’, ‘CACM-1194’ and ‘CACM-3120’. Reason for lesser matches is that stemming would have had a high impact on the query terms.

**v) Results:**

Phase 1 (8 runs):

Final results can be found as the following text files:

1. Phase 1/Task 1/Step 4 – Retrieval Models/BM25/BM25Scores\_NoRelevance.txt
2. Phase 1/Task 1/Step 4 – Retrieval Models/Lucene/Lucene\_Scores.txt
3. Phase 1/Task 1/Step 4 – Retrieval Models/Query Likelihood/QueryLikelihoodScores.txt
4. Phase 1/Task 1/Step 4 – Retrieval Models/TF\_IDF/TF\_IDF\_SCORE.txt
5. Phase 1/Task 2/Step 2 – Retrieval PRF/ BM25Scores\_NoRelevance\_PRF.txt
6. Phase 1/Task 3/Task 3-A/Step 4 – Retrieval Models/BM25/ Stop\_BM25Scores\_NoRelevance.txt
7. Phase 1/Task 3/Task 3-A/Step 4 – Retrieval Models/Lucene/ Stop\_Lucene\_Scores.txt
8. Phase 1/Task 3/Task 3-A/Step 4 – Retrieval Models/Query Likelihood/ StopQueryLikelihoodScores.txt

Phase 2:

Final html file containing snippets with query terms highlighted can be found as following:

Phase2/Snippets\_Lucene.html

Phase 3:

Results for effectiveness metrics for each model can be found as shown:

1. Phase 3\Precision Recall Tables\Baseline Lucene
2. Phase 3\Precision Recall Tables\Baseline Smoothed Query Likelihood
3. Phase 3\Precision Recall Tables\Baseline TF-IDF
4. Phase 3\Precision Recall Tables\BM25 (No-Relevance)
5. Phase 3\Precision Recall Tables\BM25 Pseudo-relevance Feedback
6. Phase 3\Precision Recall Tables\Stopped BM25
7. Phase 3\Precision Recall Tables\Stopped Lucene
8. Phase 3\Precision Recall Tables\Stopped Smoothed Query Likelihood

**vi. Conclusion and outlook:**

|  |  |  |
| --- | --- | --- |
| **Retrieval Model** | **Mean Average Precision** | **Mean Reciprocal Rank** |
| BM25 (Without Relevance) | 0.435550 | 0.695191 |
| BM25 (Without Relevance With PRF) | 0.396661 | 0.565076 |
| Lucene | 0.412352 | 0.701790 |
| TF-IDF | 0.258652 | 0.449084 |
| Smoothed Query Likelihood | 0.382881 | 0.649601 |
| BM25 (With Stopping) | 0.474866 | 0.745006 |
| Lucene (With stopping) | 0.433673 | 0.734005 |
| Smoothed Query Likelihood (With Stopping) | 0.426498 | 0.737010 |

NOTE: PRF stands for Pseudo Relevance Feedback

* After analyzing the evaluation of top 100 documents of each query by all the models, a conclusion can be made that for the given combination of corpus and queries, the three runs with stopping gave almost the same results, amongst which BM25 with stopping gave the best results (Mean Average Precision of 0.475 and Mean Reciprocal Rank of 0.745)
* Outlook:

- If we were provided with any user query logs information (such as session history, etc.), the query refinement technique which involved query expansion could use query logs as its source for better results for expanded queries than using pseudo relevance feedback because query log information is the best source for knowing effective context of the query.

- Also, storing the term positions in the index and thus considering the proximity of terms in documents would yield better retrieval results.

**vii. Bibliography and References:**

[1] Search Engines: Information Retrieval in Practice by Croft, Metzler, Strohman

(For concepts and logic behind implementations)

[2] Manning, Christopher D; Raghavan, Prabhakar; Schutze Hinrich An Introduction to Information Retrieval. Cambridge England: Cambridge University Press 2009

[3] <http://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf>

[4] <http://nlp.stanford.edu/IR-book/pdf/09expand.pdf>

[5] <http://nlp.stanford.edu/IR-book/essir2011/pdf/11prob.pdf>

[6] <http://norvig.com/spell-correct.html>

[7] https://tinyurl.com/yclxj834