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)

1) Phase 1: Indexing and Retrieval

1) Task 1:
    1) Step 1-Corpus Generation: Take raw HTML files as input and return same number of files with each file tokenized by case-folding and removing punctuations. ($Kevin$)
    2) Step 2-Index Generation: Generates an inverted index of the form Term →[(Document_id, Term Frequency)] using documents generated in step 1. ($Kevin$)
    3) Step 3-Query Cleaning: The queries provided are processed in the same way as corpus and written into a file. ($Twisha$)
    4) Step 4-Retrieval Models: It implements 4 retrieval models by using the cleaned queries and index generated in prev. steps.
        1. BM25: ($Kevin$)
            BM25 considering no relevance
            BM25 Considering relevance
        2. Lucene ($Kevin$)
        3. Smoothed Query Likelihood ($Twisha$)
        4. TF-IDF ($Manoj$)
2) Task 2: Pseudo relevance feedback is used for query enrichment which re-computes scores calculated for BM25-no relevance. Returns top 100 documents after expanding the query with most frequent terms in the top documents from BM25-No relevance. ($Kevin$)
3) Task 3:
    1) Part A: Generates new corpus by removing the stop words. Stop words are also removed from queries and three baseline runs – BM25, Lucene, Smoothed Query Likelihood are executed. ($Manoj$)
    2) Part B: Perform stemming on corpus and queries and execute BM25, Lucene, Smoothed Query Likelihood. ($Manoj,\ Twisha$)

2) Phase 2: Displaying Results: ($Twisha$)

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

3) Phase 3: Evaluation: . ($Kevin$)

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

4) Extra-credit: $(Manoj, Twisha, Kevin)$

Implemented an error-generator model and soft matching query handler which introduces noise to the query terms and then reduces the impact of noise on the effectiveness of the retrieved results.

III. Literature and Resources:

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

- TF-IDF measure: 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. [1]
- BM25 Model: For BM25 model we are using the formula from the book [1]; the values of 'b', '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: used Pseudo Relevance Feedback [1] for query enrichment on the BM25 model without relevance.
- Stopping: Used 'common_words.txt' to perform stopping. The words in the stop list are not indexed and 3 models - Lucene, BM25 and Smoothed Query Likelihood are executed.
- Stemming: Performed stemming using '$cacm\_stem.query$' for corpus and '$cacm\_stem.query$' for query; ran on 3 models -Lucene, BM25 and Smoothed Query Likelihood.
- Snippet Generation: Generated snippets for top 5 documents from Lucene, computing the significance factor of each sentence in the documents and ranking them in decreasing order by using the formula in book [1].
- Precision & Recall:
    - Precision $= |Relevant \cap Retrieved| \,/\, |Retrieved|$
    - Recall = $|Relevant \cap Retrieved| \,/\, |Relevant|$
- MAP:
    - MAP = $\Sigma\ Average\ Precision\ /\ Number\ of\ Queries$
- MRR: Reciprocal rank(RR) is reciprocal of the rank at which the first relevant document is retrieved.
    - MRR = $\Sigma\ RR\ /\ Number\ of\ Queries$
- 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 from <doc_no>) → Query (from <doc>).
- Perform the same process for cleaning as done for documents above

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

1. 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] ($Page\ 250$) is used for calculating BM25 scores.

2. Lucene:

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

3. Smoothed Query Likelihood Model:

Smoothing parameter (λ) is set to 0.35 for computing scores for QLM. The formula from the book [1] ($Page\ 257$) is used for calculating log(P(Q|D)).

4. 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. [1]($Page\ 208$).

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.

$$\text{Significance factor} = \frac{\text{square of the number of significant words in the window}}{\text{Total number of words}} \quad [1]$$

We have considered the terms in the query (excluding stop words) 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.

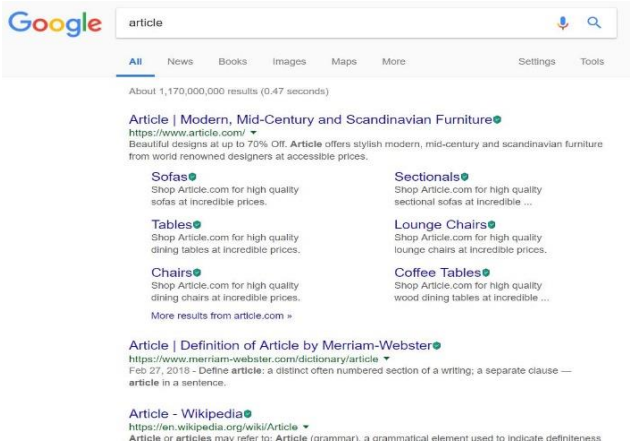
We also ran tests for multiple values of 'k' value such that, in a snippet, no the significant words should have more than k non-significant words between them. A very few documents had such sentences and hence, we decided to discard this approach.

6) Extra Credit:

Studies have shown that 80% or more of spelling errors are caused within an edit distance of one or two. It can be observed as follows,

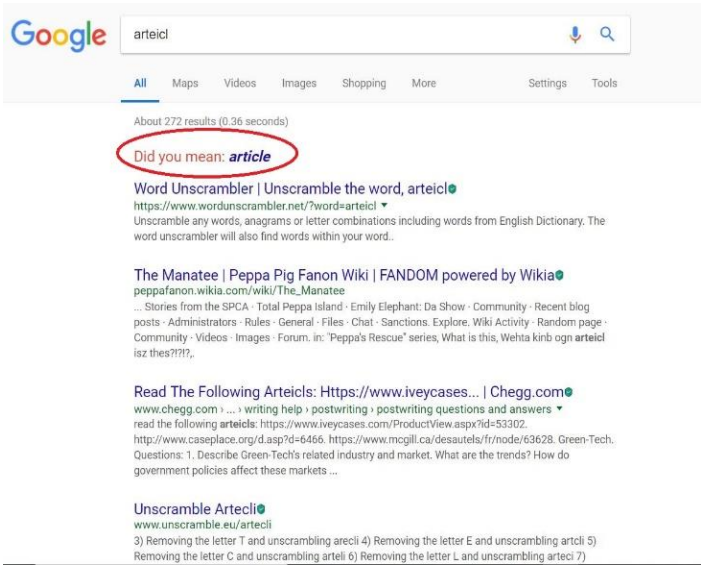
For the search term "article", following is the result observed in different scenarios

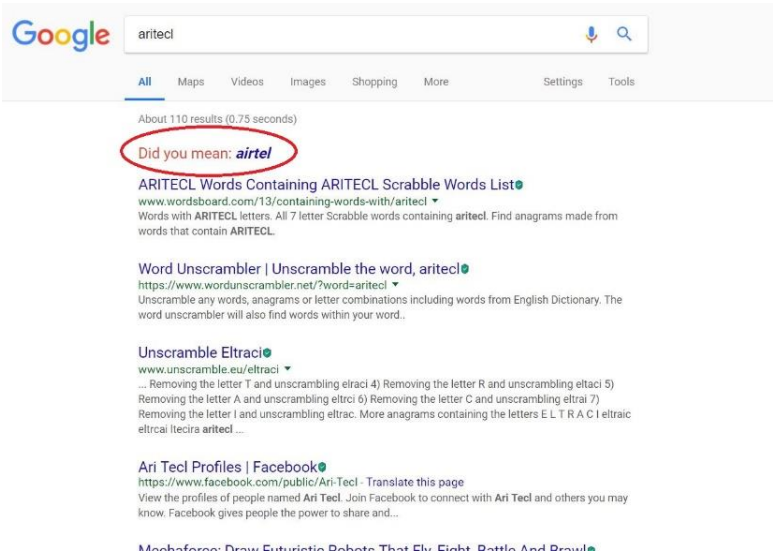
1. No word suggestion when the given word is correct.
2. "$arteicl$" is edit distance 1 for "$article$", Google correctly predicts the word.
3. "$aritecl$" is edit distance 2 for "$article$", Google's suggestion is not as expected.



Case 1

Case 2





Case 3

We can observe that the word suggestor is designed to return more robust result for terms with edit distance less than or equal to two when compared to more edit distance.

Part A:

Based on these observations, the error generator model is implemented,

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 [6][7]. 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 c, out of all possible suitable words, that maximizes the probability that c must be the desired word, given the original misspelled word w:

$$argmax_{c \in candidates} P(c|w)$$

Equivalent to Baye's theorem,

$$argmax_{c \in candidates} P(c)\ P(w|c)\ /\ P(w)$$

On factoring out $P(W)$, we get

$$argmax_{c \in candidates} P(c)\ P(w|c)$$

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

1. Selection Mechanism: $argmax$
    a. It is used to select the most suitable word based on highest probabilities.
    b. If both the probabilities are same, then the one with higher frequency is chosen.
    c. A dictionary of d = {mis-spelled word: suitable word} is updated and finally written to a file "replacements.txt"
2. Candidate Model: $c \in candidates$
    a. Gives all possible corrections for the word w
    b. 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.
    c. known words from dictionary (generated from corpus) is used to reduce the generated list.
3. Language Model: $P(c)$
    a. To generate a database for finding the probability of the word c in the corpus.
4. Error Model: $P(w|c)$
    a. It takes care of calculating the probability that word w would be typed in a query when the user thinks of word c.

   b. 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):
   a. "ErrorQueries.txt" and "replacement.txt" files are parsed and stored in a container.
   b. Misspelled words are identified in each query and replaced with the suitable words.
   c. The query is regenerated and stored in "correctQueries.txt"

Query by Query Analysis (Phase 1 / Task 3B):

Three queries that we chose are:

1) Distribut comput structur and algorithm (Stemmed Version)
   Distributed computing structures and algorithms (Non-Stemmed Version)
2) parallel algorithm (Stemmed Version)'
   Parallel algorithms (Non-Stemmed Version)
3) appli stochast process (Stemmed Version)
   Applied stochastic processes (Non-Stemmed Version)

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 'computer, 'compute, 'computation', 'computing' ,etc. all get stemmed to single stem 'comput'. 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 8 documents match when we compare the top 20 documents obtained by the BM25 model with stemming and of that without stemming, namely, CACM-2276, CACM-2849, CACM-2406, CACM-2454, CACM-3137, CACM-2949, CACM-3082, CACM-3148. Hence, stemming has a high impact on the retrieval of common documents for the same query terms.

If we do the same analysis for the second query between the same two versions of the aforementioned model, 13 documents match, namely 'CACM-2973, 'CACM-3075', 'CACM-2266', 'CACM-2557', 'CACM-0950', 'CACM-2714', 'CACM-2685', 'CACM-1262', 'CACM-2700', 'CACM-2433', 'CACM-0141', 'CACM-1828', 'CACM-2896'. There is very high overlap between the two, because 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 third query between the same two versions of the aforementioned model, 10 documents match, namely,'CACM-1696', 'CACM-0268', 'CACM-1410', 'CACM-2882', 'CACM-1540', 'CACM-1194', 'CACM-3120', 'CACM-0293', 'CACM-0942' and 'CACM-0020'. Although all the three query terms are stemmed, their stem classes are not too large because not many terms would be stemmed to stems like '$stochast$' and '$process$'. Hence, we have half of the documents overlapping which is not too small or not too large.

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:
  1. 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.
  2. Also, storing the term positions in the index and thus considering the proximity of terms in documents would yield better retrieval results.

Extra Credit evaluation

Retrieval Model	Mean Average Precision	Mean Reciprocal Rank
BM25 (Baseline Without Relevance)	0.435550	0.695191
BM25 (With Query Errors – Task 3-A)	0.318129	0.467176
BM25(With Modified Queries -Task 3-B)	0.412030	0.676416

After comparing the MAP and MRR values for the BM25 model for different types as described above, we observe that after introducing errors in the query, these values go down because, the words with errors are not present in the corpus. After replacing the words that do not exist in the corpus with replacements found using the algorithm described, the values of MAP and MRR increase and are almost equal to the baseline run.

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