***Information Retrieval***

CS 6200: Information Retrieval

Northeastern University

Spring 2018, Prof. Nada Naji

***TEAM MEMBERS:***

*Kevin Shah*

*Manoj Venkatesan*

*Twisha Vyas*

**ii. Introduction:**

The main aim of this project is to design and implement an information retrieval system using models such as BM25, Lucene, Query Likelihood and TF-IDF and 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 as below:

1. Task 1: It includes 4 steps:
2. Step 1-Corpus Generation: This step takes Raw HTML files (given corpus) as input and returns same number of files but with each input file tokenized by case-folding and removing punctuations. (Kevin)
3. Step 2-Index Generation: This step takes the clean documents created in step 1 as input and generates an inverted index of the form Term 🡪[(Document\_id, Term Frequency)]. (Kevin)
4. Step 3-Query Cleaning: The queries provided are processed in the same way as documents and the cleaned queries are written into a file. (Twisha)
5. Step4-Retrieval Models: There are 4 parts in this step. 4 retrieval models are implemented using the clean queries and index generated in steps 3 and 2. They are as follows:
6. BM25:

BM25 considering no relevance (Kevin)

BM25 Considering relevance (Kevin)

1. Lucene (Kevin)
2. Smoothed Query Likelihood (Twisha)
3. TF-IDF (Manoj)
4. Task 2: We have used pseudo relevance feedback for query enrichment which re-computes scores calculated for BM25 Model which considered no relevance. The output of this task contains top 100 documents after expanding the query with most frequent terms in top documents from BM25 Model (No relevance). (Kevin)
5. Task 3: This task comprises of 2 parts:
6. Part A: In this part, 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 (Manoj)
8. Step 2: Index Generation (Manoj)
9. Step 3: Query Cleaning (Manoj)
10. Step 4: Retrieval Models. (Manoj)
11. Part B: In this part, stemming is performed on the terms in the corpus as well as the queries and three baseline runs – BM25, Lucene, Smoothed Query Likelihood are used to rank the documents.

1)Step 1: Corpus Generation (Manoj)

2)Step 2: Index Generation (Manoj)

3)Step 3: Query Cleaning (Manoj)

4)Step 4: Retrieval Models. (Manoj)

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

This part of the project consists of snippet generation and query term highlighting performed on results of Lucene model. (Twisha)

1. **Phase 3: Evaluation:**

(Kevin)

In this phase, we have calculated the following effectiveness metrics:

1. MAP

2. MRR

3. P@K, K = 5 and 20

4. Precision & Recall

For the following models:

Decide the model on which you want to perform evaluation:

1. Baseline BM25 (No relevance)

2. Baseline Lucene

3. Baseline Smoothed Query Likelihood

4. Baseline TF-IDF

5. BM25 Pseudo-relevance Feedback

6. Stopped BM25

7. Stopped Smoothed Query Likelihood

8. Stopped Lucene

1. **Extra-credit:**

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

1. Part A: Query error generator (Manoj)
2. Part B: Soft matching query handler (Twisha)

**iii) Literature and Resources:**

Following approaches were used while performing each of the below mentioned tasks:

1. Phase 1:

Task 1:

Step 1:

While de-punctuating, occurrences of ‘-’, ‘:’, ‘,’ or ‘.’ within digits are retained and occurrence of ‘$’ before a number is retained

………………………………………

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

Step 4- Retrieval Models:

BM25:

We have code for both, BM25 using 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.

We are using the formula from the book *‘Search Engines: Information Retrieval in Practice’*[1].

((k2+1)q)/((k2+q))\*((k1+1)f)/((K+f))\*log((r+0.5)(N-n-R+r+0.5))/((n-r+0.5)\*(R-r+0.5))

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 and analyzing text.

Smoothed Query Likelihood Model:

The smoothing parameter (λ) is set to 0.35 for computing scores using Query Likelihood Model. We have used the following formula from *‘Search Engines: Information Retrieval in Practice’*[1].

score=(1- λ)\*(tf/D)+(λ\*(cf/C))

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

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

Task 2:

In pseudo-relevance feedback model, words occurring frequently in top documents are added in query to perform query expansion. (CMS Page 208).

……………………………………………………

1. Phase 2:

While dividing each of the top 5 documents’ contents into sentences, an assumption is made that each sentence consists of a maximum of 10 tokens. The algorithms for snippet generation include a Luhn’s Approach of calculating significant factor to select top sentences for summary (CMS Page 216). To find whether a word is significant or not, a modified version of frequency-based criterion has been used (CMS Page 216, 217) as below:

A word is significant iff

f(d,w)>=4-0.1\*(25-sd) if sd<25 or

f(d,w)>=4 if 25<=sd<=40 or

f(d,w)>=4+0.1\*(sd-40) otherwise

where,

f(d,w) is the frequency of word w in document d and w is not a stop-word

sd is the number of sentences in document d.

This algorithm is used to calculate significance factor of all the sentences and top 3 sentences are chosen to be included in snippet.

1. Phase 3: For a uniform test for all runs, only the queries with relevance information given have been used, rest have been excluded i.e. 52 queries have been evaluated for each model in this case.

**OTHER TOOLS USED:**

1. Beautiful SOUP: Used for parsing the documents in the given corpus.
2. Lucene libraries: lucene-core-4.7.2.jar

lucene-queryparser-4.7.2.jar

lucene-analyzers-common-4.7.2.jar

**iv) Implementation and Discussion:**

Thorough descriptions for tasks are as below:

1. Phase 1 Task 2:

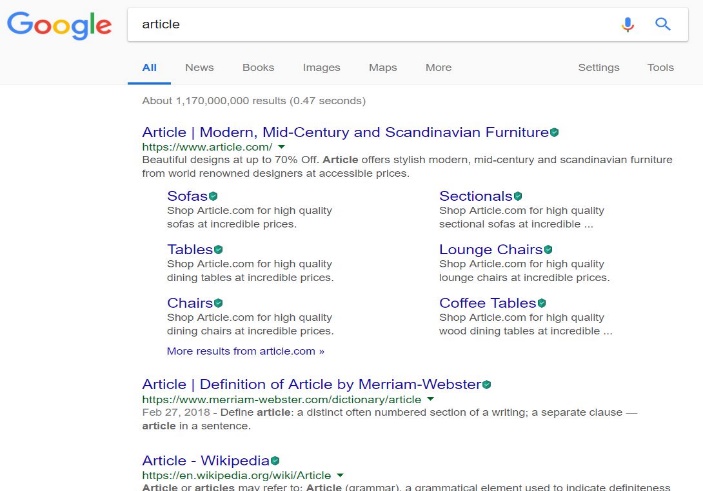
First, consider top “k” words from each of the top “n” documents. Consider these factors (k, n) each to be 5. Thus, 5\*5=25 words are to be added to each given query, thereby performing query expansion. Since the top 5 words from each document can be a “stop word”, do not include the stop words as per ‘common\_words.txt’. Also, remove the words that are present in this expansion term list, which is already present in the original query. Run the bm25 (with relevance) model for these expanded queries.

Start from here for Extra Credit theory

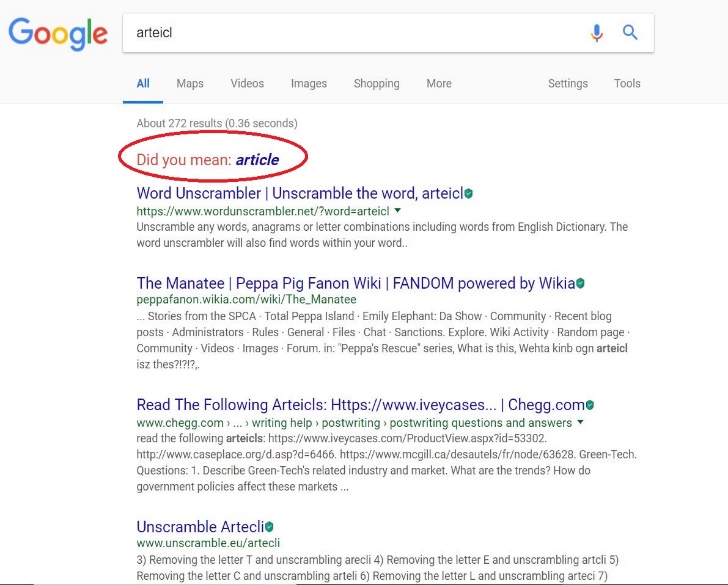
1. Extra Credit:

According to the reference in the Croft book, 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?”) features are implemented. It can be observed as follows,

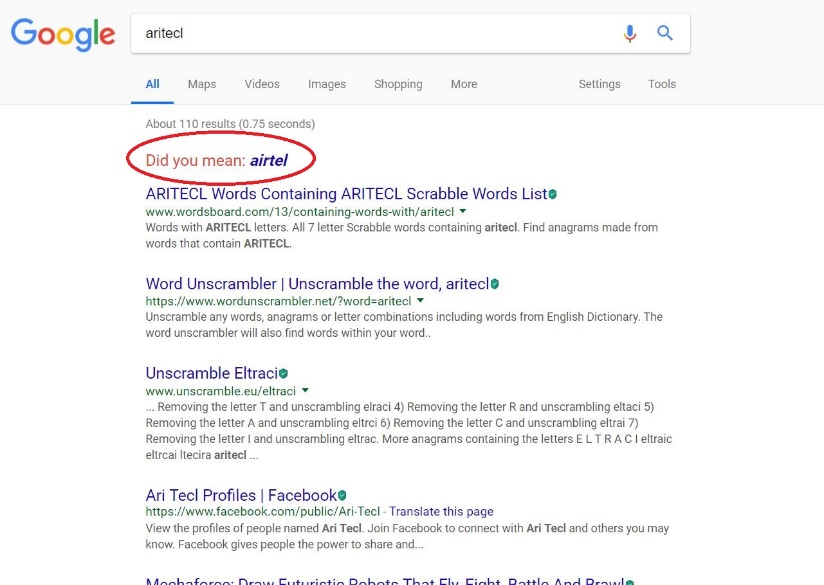
For the search term “article”, the following is the result



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



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



Thus, this proves 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, 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 clean queries from the file ‘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 by noise model.
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 for sure (eg.whether "lates" must be exchanged with "lattes" or "latest" or "late") and thus probabilities will come handy. 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 :

This is equivalent to Baye’s theorem,

On factoring out the , 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
   4. Finally, the dictionary is written to a file “replacements.txt”
2. Candidate Model:
   1. Gives all the 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. The generated list is reduced by considering only the known words in the dictionary
3. Language Model:
   1. It is used to generate the 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 on mind.
   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. This will parse the “ErrorQueries.txt” and “replacement.txt” and store in a container.
   2. The misspelled words are identified in each query and replaced with the suitable words.
   3. The query is regenerated and stored in “correctQueries.txt”

End of Extra Credit copy

**Query by Query Analysis:**

Three queries we chose are:

1. portabl oper system (Stemmed Version)

portable operating systems (Non-Stemmed Version)

1. parallel algorithm (Stemmed Version)

Parallel algorithms (Non-Stemmed Version)

1. appli stochast process (Stemmed Version)

Applied stochastic processes (Non-Stemmed Version)

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

NOTE: A relatively different query-by-query analysis is placed in “Query-By-Query-Analysis.txt” which contains the analysis for top 5 documents obtained by the three baseline runs- Lucene;s retrieval model, BM25(with relevance) retrieval model and tf-idf retrieval model. We talk about the drops in scores between the ranks, common documents obtained by the three aforementioned retrieval models, etc. and provide a generalized speculation, per query, for three queries we found interesting.

**v) Results:**

The following documents contain the results of the implementation and comparison of the various baseline runs that are performed for the retrieval model.

* Phase 1 – Task 1

BM 25(With Relevance) - BM25Scores\_Relevance.txt

BM 25(No Relevance) - BM25Scores\_NoRelevance.txt

Lucene - Lucene\_Scores.txt

QLM - QueryLikelihoodScores.txt

TF-IDF - TF\_IDF\_SCORE.txt

* Phase 1 – Task 2

Query Expansion - BM25Scores\_NoRelevance\_PRF.txt

* Phase 1 – Task 3

Stopping BM25 – Stop\_BM25Scores\_NoRelevance.txt

Stopping Lucene – Stop\_Lucene\_Scores.txt

Stopping QLM – StopQueryLikelihoodScores.txt

Stemming BM25 - Stem\_BM25Scores\_NoRelevance.txt

Stemming Lucene - Stemmed\_Lucene\_Scores.txt

Stemming QLM - StemmedQueryLikelihoodScores.txt

* Phase 2:

Snippet Generation – Snippets\_Lucene.html

* Phase 3:

Baseline Lucene –

Baseline Smoothed Query Likelihood –

Baseline TF-IDF –

BM25 (No-Relevance) –

BM25 Pseudo-relevance Feedback –

Stopped BM25 –

Stopped Lucene –

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 (Without Stopping) | 0.474866 | 0.695006 |
| 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, BM25 with relevance considered and with stopping, gave the best results (Mean Average Precision of 0.554 and Mean Reciprocal Rank of 0.824)
* As BM25 with Relevance and Stopping harnessed the relevance information provided in cacm.rel.txt unlike other models, it tends to show better results for this test collection. Also, it also used stopping to improve its score.
* 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. PRF is only as we have no query log information which is the best source for a knowing effective context of the query.

**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