PROJECT REPORT

Course Name: CS6200 Information Retrieval

Sri Annapurna Bhupatiraju

Rachana Tondare

Semester: Fall 2016

Under guidance of: Prof. Nada Naji

PROJECT DESCRIPTION

Goal: Design and build your information retrieval systems, evaluate and compare their performance levels in terms of retrieval effectiveness

With these goals in mind, we designed different versions of information retrieval systems with various models (BM25, Cosine Similarity, Lucene, TF-IDF), processing (stemming, stopping) and evaluated their performances based on precision and recall values. This project encompasses the core information retrieval concepts and processes learnt throughout the semester resulting in building our very own search engines. We used the Cosine Similarity model as our baseline run and Pseudo Relevance feedback as our query expansion technique.

MEMBER CONTRIBUTION

Sri Annapurna

1. Implemented Cosine Similarity and BM25 model
2. Implemented Pseudo Relevance for query expansion
3. Implemented Task 3B i.e. Stopping
4. Helped refine precision and recall
5. Worked on query-by-query analysis based on the evaluation metrics
6. Reviewed and helped in editing the report

Rachana

1. Implemented the Tf-Idf and default Lucene model.
2. Implemented Task 3B i.e. stemming
3. Implemented the initial version of precision and recall algorithm
4. Worked on query-by-query analysis based on the evaluation metrics
5. Drafted the report

Overall, each member touched on all the aspects of the project and was aware of each task as we practiced pair programming.

LITERATURE AND RESOURCES

OVERVIEW

Baseline

We have used the Cosine Similarity model as the base search engine because we found the results to be most efficient and consistent when compared with results from "*cacm.rel*".

Query Expansion

The technique used for query expansion was *Pseudo relevance feedback*. We preferred this technique over the others because based on a research performed by Hull, D. A. on Stemming algorithms - a case study for detailed evaluation. *Journal of the American Society for Information Science it stated that* this approach is an efficient and less time-consuming and reduces load on the user. Many expansion terms identified in traditional approaches are indeed unrelated to the query and harmful to the retrieval. Hence we decided to proceed with this approach.

Tools

The tools we used for this project are Eclipse-Neon, Eclipse-Mars, notepad++ to compare results and Microsoft excel to display results. We have also used Lucene for its default and BM25 implementation. The citations for more resources used for research as given at the end of this document.

IMPLEMENTATION

Task 1: Baseline

1. Parsed the raw html files and extracted the pre-tags

2. Processing done to the query as well corpus:
   - a. Removed all punctuations except commas, full stops and $ within digits.
   - b. Retained hyphens in the string
   - c. Converted to lowercase

We took this decision as we realized without parsing the corpus a lot of irrelevant documents were retrieved which was noise and some terms were left out. For example: a query term (Time did not have any term in the inverted index due to the opening bracket and lack of case folding. We have not removed the stop words from the corpus as the we wanted to stick to basic pre-processing for the base line runs.

3. Copied the processed text to text files based on processed filenames (just the document number)

4. Implemented the following 4 models:
   - a. Lucene default: Used our HW4 assignment code
   - b. Lucene BM25: On research we found that Lucene has the BM25 model with the default b, k1, k2 values and hence, made code changes to implement the same.
   - c. Cosine:
     - Parsed each file in the corpus and tokenized them based on space as asked.
     - Once tokenized, added the length of this token array as the size of the respective document.
     - Iterated through the token array based on the value of n, and added an entry in the inverted index. Also, maintained a term frequency and document frequency table at the same time.
     - For calculating the cosine similarity score between a query and a document, I needed:

Query Weights for each term in the query

1. Initially, if the query is present in the index, then
   tf = 1
   Else
   tf = 0
2. Normalizing the tf if present in index else it is 0:
   tf = 1 + log10 (tf)
3. Retrieve the document frequency i.e. df for that term
4. Calculate the normalized idf (Here: N = 3204)
   idf = Math.log10(N/df)
5. Calculate the query weight
   queryWeight = tf*idf

Document Weights corresponding to each query term

1. Iterate through each term in the query

2. For every term in the query, get its inverted list
3. Calculate the weight of the document as we know the frequency of that term in the document
4. Thus, every document would get n weights corresponding to n terms in a query.

Calculate the cosine similarity score now that we have the term weights and the document weights

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}}$$

d. Tf-Idf: Basic formula, where for every term in the query, we summed the term frequency for a document multiplied by its normalized inverse document frequency. If there are, no documents containing the term then frequency of documents with the word is 0 and which would lead to an invalid computation of idf hence to normalize it we have defined idf as log of the ratio of total size of the document set to frequency of documents containing the term to base 10.

Task 2: Query Expansion

We chose to apply Pseudo relevance feedback technique on Cosine model since it retrieved the most relevant set of documents with least noise. Based on the research paper (http://www.cicling.org/2008/RCS-vol-33/14-Perez.pdf) the Cosine model works best with pseudo relevance feedback. In case of Thesauri and Ontologies, building thesaurus manually requires the study of words meanings and based on these word meanings the relation between these words, their synonyms and antonyms. These studies are expensive, need a lot of time and effort and suffer from the bias problems. Hence we ruled out this approach. In case of pseudo relevance feedback process the system retrieves a set of the ranked documents and then the user determines which of these documents is relevant to his query. The system reforms the query based on the user judgment (the determination of the relevant and irrelevant documents). The system repeats the retrieval process by using the modified query. This process is still repeated until the user gets the suitable documents for him. Due to lack users and other resources and the good efficiency of cosine model with this system we decided to implement pseudo relevance feedback approach.

Steps Taken:

1. Retrieve the initial run of 100 documents from Cosine model.
2. Consider the top documents as most relevant and reindex.
3. Get the top 10 highest frequency words and add them to the current query
4. Perform a search again to get new list of documents.

Task 3:

Stopping: Used the Cosine Model for the implementation of stopping. We did not index any of the stop words and removed them from the query as well.

Stemming: Used the cosine model for the implementation of stemming.

Task 4: Evaluation

$$\text{Precision} = \frac{\text{Number of relevant documents retrieved so far}}{\text{Total Number of documents retrieved}}$$

$$\text{Recall} = \frac{\text{Number of relevant documents retrieved so far}}{\text{Total Number of relevant documents retrieved}}$$

$$\text{MAP} = \frac{\text{Average Precision}}{\text{Number of Queries}}$$

$$\text{MRR} = \frac{\text{Reciprocal Rank}}{\text{Number of Queries}}$$

We did not consider the queries for which relevance was not given.

The 7th run was implemented using stopping for the tf-idf model.

Query by query analysis:

For analysis, we considered a few select queries:

1. Portable operating systems
   Consider precision at 5 for each of these systems:
   Lucene Default: 0.4, 0.2
   Lucene BM25: 0.4, 0.2
   Cosine: 0.4, 0.15
   Tf-Idf: 0.2, 0.1
   Stemming (query is also stemmed): 0.0, 0.0
   Stopping: 0.4, 0.15
   PseudoRelevanceFeedback: 0.0, 0.0

   As we can see, the Lucene systems worked better in this query as recall increased whereas
   Even another query, "parallel algorithms" had almost similar results. Hence, for shorter queries the precision at k is almost like each of these search engines.

2. Query:

List all articles on EL1 and ECL (EL1 may be given as EL/1; I don't remember how they did it.

Consider precision at 5 for each of these systems:
Lucene_default: 0.04 0.6
Lucene_BM25: 0.04 0.6
Cosine: 0.34 0.62
TfIdf: 0.01 1
Stemming: 0.07 0.17
Stopping: 0.01 1
PseudoRelevanceFeedback: 0.01 1

As we can see, the Cosine worked better in this query has highest recall and precision whereas other systems don't have such efficient statistics.

3. Dictionary construction and accessing methods for fast retrieval of words or lexical items or morphologically related information. Hashing or indexing methods are usually applied to English spelling or natural language problems.

   Consider precision at 20 for each of these systems:
   Lucene_default: 0.45 0.32
   Lucene_BM25: 0.6 0.33
   Cosine: 0.34 0.4
   TfIdf: 0. 35 0.1
   Stemming: 0.5 0.1
   Stopping: 0.01 0.1
   PseudoRelevanceFeedback: 0.2 0.1

For big queries Lucene BM25 does give a high precision but recall is less than that of cosine though the precision for cosine is half that o BM25 hence BM25 is a good model in cases where there are huge queries.

4. Fast algorithm for context-free language recognition or parsing

   BM 25: Precision at 20: 0.45 Recall 0.85
   Lucene: Precision at 20: 0.45 Recall 0.85
   Cosine: Precision at 20: 0.35
   TFIDF: Precision at 20: 0.2 0.45
   Stemming: Precision at 20: 0.5 0.1
   PseudoRelevanceFeedback: Precision at 20: 0.1 0.1
   Stopping: Precision at 20: 0.25 0.85

For this query as well we see that BM25 has a good precision and recall, hence proves gain to be a good system.

Results:

PRTable_Cosine.txt PRTable_Lucene_BM25.txt PRTable_Lucene_default.txt PRTable_PSR.txt

PRTable_Stemming.txt PRTable_Stopping.txt PRTable_TfIdf.txt PRTable_TfIdf_Stopping.txt

Conclusion: Hence for big queries and high precision and recall based on the evaluation BM25 is a good retrieval system.
Outlook: The efficiency of the PseudoRelevanceFeedback systems can be improved by choosing better words for query expansion, we can perform supervised learning rather than unsupervised learning.

REFERENCES

http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/stats/PrecisionRecallStats.html

https://www.creighton.edu/fileadmin/user/HSL/docs/ref/Searching_-_Recall_Precision.pdf

https://www.youtube.com/watch?v=WH4YHYABvQo

http://www.informationr.net/ir/19-1/paper605.html

https://www.inf.ed.ac.uk/publications/thesis/online/IM050335.pdf

http://dl.acm.org/citation.cfm?id=1390377