CS6200

Information Retrieval

Fall 2016

Group Members:
Purnesh Dixit
Rushabh Shah
Mahesh Kumar Bhaskaran

Nada Naji

12-9-2016

Contents

Introduction

This project was intended to implement several retrieval models namely *Tf-Idf, Vector Space, Lucene, BM25* and evaluate them over the CACM dataset based on standard parameters *MAP, MRR, P@K and Precision & Recall*. Additionally, it also throws a light upon how incorporation of techniques like *stopping, stemming* and *query expansion* affect the results of base model.

Members Contribution

Purnesh Dixit

Project Implementation

- Built the indexer and text parser for the given dataset (including stem).
- Implemented baseline retrieval models for Tf-Idf and Vector Space Cosine Similarity.
- Infused multi-threading to rank set of documents in parallel which reduced the response time.

Documentation

- Contributed in project description.
- Contributed in implementation and design decisions.

Rushabh Shah

Project Implementation

- Implemented Lucene retrieval model for the CACM corpus and modified the java file so as to process all the 64 queries all at once.
- Performed query by query analysis for the TF-IDF stemmed as well as non-stemmed system.
- Implemented evaluation metrics.

Documentation

- Contributed in Literature and resources.
- Contributed in Implementation and discussion, Results.

Mahesh Kumar Bhaskaran

Project Implementation

- Implemented query expansion with pseudo relevance feedback.
- Implemented baseline retrieval model for BM25.

Documentation

- Contributed in conclusion and outlook.

Literature and Resources

Retrieval Models Implemented

Tf-Idf

Tf-Idf stands for *term frequency-inverse document frequency*. Though, it is often used as weighing scheme for the term in a corpus, it is also one of the simplest ranking function which compute the document score by summing up the tf-idf score of each query term. Many sophisticated ranking algorithms are variations of this simple model [3].

Vector Space Cosine Similarity

The vector space model is the first model-providing framework for implementing term weighting, ranking and relevance feedback. In this model, documents and queries are represented as *t-dimensional vector* where, *t* is terms in the document and query respectively. Given this representation, documents could be ranked by computing the distance between the points representing query and document.

Lucene

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

BM25

BM25 is one of the most sophisticated probabilistic retrieval model which extends the binary independence model to include query and document term weights.

Query Processing Techniques

Several query processing techniques are used to enhance the base model ranking results. We have implemented following techniques:

Stopping

The best terms in a document are those with higher tf-idf because those terms are common in the document, while being rare in the collection. Stop words are those words that appear often across the documents, hence become less important. Hence, in one of our run we skipped computing scores for terms from stop word list.

It fetched more relevant documents at higher ranks and produced better throughput even with tf-idf retrieval model which is a very naïve retrieval model.

Stemming

Stemming is a technique that converts all the variants of a term to a single term and does the same while processing the query. This in turn prevent the sharing of scores by variants of words and converges the scores towards most prevalent term in the query.

We implemented stemming with tf-idf retrieval model in one of our run to evaluate its effectiveness as tf-idf is most sensitive to weight of query terms in document.

Query Expansion (Pseudo-relevance feedback):

The general idea used in the technique of pseudo-relevance feedback is instead of asking the user to identify relevant documents, the system simply assumes that the top-ranked documents (top 10 in general) are relevant. Words that occur frequently in these documents may then be used to expand the initial query.

In our implementation, we have used top 10 documents from 1st run of baseline Tf-Idf and expanded the query with terms having frequency greater than 3(excluding the stop words) in a given document.

Index creation and parsing techniques

Regular expressions

In our implementation we have use several regex expressions to filter out plain text from the raw files. Furthermore, regex is used to read the indexes and inverted lists. Some of the regex expressions are:

```
pTagRegex              = '<pre>.*?</pre>'

textExtractingRegex    = r'<[^>]+>|[^<]+'

numericRegex           = r'(\d{1,3},\d{3}(,\d{3})*)(\.\d*)?|\d+\.?\d*'

alphanumericRegex      = '.*\\d+.*'
```

Dictionary

Dictionary is used to store the indexes during all computations. Both read and write from and to indexes are done through dictionary.

The terms in corpus are used as the keys of the dictionary. Each key contains its documents and document frequency.

Snippet Generation

We have implemented snippet generation and highlighting technique to show the summary of significant sentences of the top 10 documents. The query terms are highlighted in tag <HL>..</HL>. We have used Luhn's methodology to fetch significant sentences. [5]

Implementation and discussion

We have implemented seven retrieval model. The table gives a summary of the models.

Table of Retrieval Models

Retrieval Model	Stemming	Query Expansion	Stopping
BM25	No	No	No
TF-IDF	No	No	No
Cosine Similarity	No	No	No
Lucene	No	No	No
TF-IDF	No	Pseudo-Relevance	No
TF-IDF	Yes	No	No
TF-IDF	No	No	Yes
BM25	No	No	Yes

Description

Indexing & Parsing

The CACM dataset consists of 3204 raw html files and each file have their own unique ID. The indexer reads the dataset and extract the plain text using regular expression. We have used regular expressions to separate the tags of the html files, removing punctuations, preserving punctuations in digits and alphanumeric terms.

Post extraction, some regular expressions are used to calculate term frequency and document frequency. We have used simple unigram model. An inverted index is created in Dictionary form where terms are keys and 2 inverted lists are produced: *"Term Frequency in corpus"* and *"Document Frequency of each term".*

TF-IDF

A simple tf-idf ranking function, which computes the document score by summing up the tf-idf score of each query term, is implemented.

Computation Used

TF: Term Frequency, which measures how frequently a term occurs in a document. The term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization.[3]

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = loge(Total number of documents / Number of documents with term t in it).

The document score would be sum of tf-idf of all query terms in a given document

Vector Space Cosine Similarity

The vector space model procedure contains three steps:

- Fetching all the documents containing at least one of the query term.
- Summing the product of weights of each query term in document and query.
- Normalizing the weights by dividing with magnitude of query and document vector.

Computation Used

Empirically, *Cosine Similarity* measure found out to be most effective to assign highest scores to most similar documents to query.

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

Where,
Di is the document
Q is the query
Dij is the term weight in document
Qj is the term weight in query

Term weights can be calculated by various weighting schemes. In our implementation we have used a typical form of tf-idf to weight terms in documents.

$$d_{ik} = \frac{(\log(f_{ik}) + 1) \cdot \log(N/n_k)}{\sqrt{\sum_{k=1}^{t} [(\log(f_{ik}) + 1.0) \cdot \log(N/n_k)]^2}}$$

Lucene

Lucene is implemented in Java using Lucene core jar. We have used simple analyzer and simple indexer provided by the Lucene 4.7.2.

While implementing Lucene, we passed the Corpus for indexing and then read the query from "cacm.query" file and wrote the output in "Lucene_QueryResults.txt" which contains top 100 documents for all 64 queries in descending order of their scores.

BM25

BM25 is one of the most sophisticated probabilistic retrieval model which extends the binary independence model to include query and document term weights.

Computation Used

Most common form of BM25 formula is:

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

Where,

N is the number of documents in corpus
ri is the number of relevant documents containing the term i
R is the number of relevant documents
ni is the number of documents containing term i

K is the parameter to normalize tf component, $K = k_1((1 - b) + b \cdot \frac{dl}{avdl})$
k1 determines how tf changes based on document length. Typical value is 1.2
k2 has similar role for the query. It varies from 0 to 1000.

K1 as 1.2 gives produces non-linear effect for tf. This means after 2-3 occurrences of term, additional occurrences have little impact

Stopping

We have implemented tf-idf with stopping by skipping the words from common_words.txt to prevent their contribution in the document score. The reason for choosing tf-idf is that it is most sensitive to term weights.

Query Expansion (Pseudo-Relevance Feedback)

Pseudo-Relevance Feedback, also known as blind relevance feedback, provides a method for automatic local analysis. It automates the manual part of relevance feedback, so that the user gets improved retrieval performance without an extended interaction. The method is to do normal retrieval to find an initial set of most relevant documents, to then assume the top k ranked documents are relevant, and expand the initial query with high frequent terms of top k documents. The results are then retrieved for modified query. [4]

Design logic

In one of our run, we chose tf-idf retrieval model to combine with pseudo-relevance as it is not as sophisticated as other algorithms who produces good results even in baseline. Tf-Idf results allows us to evaluate pseudo-relevance better. Below are some of the examples of expanded query from our run for Query 1 and Query 3 respectively:

Initial query: What articles exist which deal with TSS (Time Sharing System), an operating system for IBM computers?

Expanded Query: what articles exist which deal with tss time sharing system   an operating system for ibm computers system operating teaching system operating performance system operating system operating sharing time system tenex users computers system terminal computer systems processes semaphores system project operating monitoring system performance software system simulation performance model evaluation system

Initial query: Intermediate languages used in construction of multi-targeted compilers; TCOLL

Expanded Query: intermediate languages used in construction of multi-targeted compilers tcoll simulation languages computer packages language programming languages future history comparison list-processing languages grammar algorithm parser context-free bounded-context compiler neumann von combining functional forms style languages state systems algebra programs programming system languages systems abstractions programming clu data languages system neliac compiler language programming languages mathematics

Query-by-Query Analysis

Stemming

We have implemented tf-idf with stemming as it is the simplest model and produce contrasting results for good and bed stemmed text.

Analysis for Query No. 1

Stemmed Query: portabl oper system

1 Q0 1591 1 4.214420 TfIdf_Stem
1 Q0 1680 2 4.013733 TfIdf_Stem
1 Q0 1930 3 3.742477 TfIdf_Stem
1 Q0 2319 4 3.612360 TfIdf_Stem
1 Q0 1747 5 2.608927 TfIdf_Stem
1 Q0 3087 6 2.408240 TfIdf_Stem
1 Q0 1033 7 2.307897 TfIdf_Stem
1 Q0 1462 8 2.207553 TfIdf_Stem
1 Q0 2188 9 2.207553 TfIdf_Stem
1 Q0 1728 10 1.906523 TfIdf_Stem

Without Stemming Query: portable operating systems

12 Q0 3127 1 4.619636 TfIdf
12 Q0 3068 2 3.562409 TfIdf
12 Q0 2740 3 3.130977 TfIdf
12 Q0 1747 4 3.050526 TfIdf
12 Q0 1680 5 2.960349 TfIdf
12 Q0 2246 6 2.903633 TfIdf
12 Q0 1930 7 2.903633 TfIdf
12 Q0 2319 8 2.659319 TfIdf
12 Q0 2317 9 2.619093 TfIdf
12 Q0 1728 10 2.619093 TfIdf

Analysis:
Document CACM-1591 is at the top of the list. This document contains information about - A Model for a Multifunctional Teaching System, which is not relevant to the query. The only reason for this document

to appear so high up on the list is that the document contains the words “operating” and “system” in its text, which are query words, but the document is not topically relevant to the query. This highlights the flaw in the tf-idf retrieval model, which focused on the query terms and not the content.

The next document CACM –1680 talk about - “A General-Purpose Display Processing and Tutorial System” which is not relevant to the topic of the query. One reason why it could be fetched is because of “over stemming”. This document contain words like operation; operate which are derived from the same root word- oper. Hence, these documents appear in the top most ranks containing non-relevant information.

Document CACM-3127 talks about – Portable Real-Time Operating System. This document has appeared on the top of the list as it contains all the query terms and is topically relevant. This document does not appear in the top 10 list for stemmed system, as stemming does not improve the results.

Therefore, we can conclude that stemming does not improve the effectiveness of the retrieval system.

Analysis for Query No. 2

Stemmed Query: perform evalu and model of comput system

6 Q0 2318 1 3.270387 TfIdf_Stem
6 Q0 3048 2 3.197022 TfIdf_Stem
6 Q0 2542 3 2.834396 TfIdf_Stem
6 Q0 3070 4 2.742421 TfIdf_Stem
6 Q0 2344 5 2.727764 TfIdf_Stem
6 Q0 1827 6 2.612838 TfIdf_Stem
6 Q0 1680 7 2.566455 TfIdf_Stem
6 Q0 2319 8 2.502902 TfIdf_Stem
6 Q0 1719 9 2.441064 TfIdf_Stem
6 Q0 2188 10 2.420267 TfIdf_Stem

Un-stemmed Query: performance evaluation and modelling of computer systems

25 Q0 1653 1 2.424466 TfIdf
25 Q0 2344 2 2.169946 TfIdf
25 Q0 2542 3 2.004461 TfIdf
25 Q0 1844 4 1.975254 TfIdf
25 Q0 1908 5 1.935303 TfIdf
25 Q0 1719 6 1.885485 TfIdf
25 Q0 2812 7 1.854196 TfIdf
25 Q0 1827 8 1.836374 TfIdf
25 Q0 1792 9 1.818881 TfIdf
25 Q0 1771 10 1.812667 TfIdf

Analysis:

The document CACM-1653 appears at the top of the un-stemmed system. This document talks about – “System Performance Evaluation: Survey and Appraisal”. Although this document explains the performance evaluation in systems, it does not mention about modelling which is a query term. This document only explains only a part of the query.

The document CACM-2318 appears at the top of the stemmed system. This document talks about- "Role of Computer System Models in Performance Evaluation". The reason it appears at the top is that it contains all the query words and it is topically relevant to the query.

Therefore, we can conclude that stemming does improve the effectiveness of the retrieval system depending on the user query.

Snippet Generation

As per Luhn's algorithm, The significant sentence is calculated based on the occurrence of significant words which are words of medium frequency the document, where "medium" means that the frequency is between predefined high-frequency(8 in our case) and low-frequency(3 in our case) cutoff values. Given the significant words, portions of the sentence that are "bracketed" by these words are considered, with a limit set for the number of non-significant words that can be between two significant words (typically four).

After extracting the snippets for each query we highlighted the query terms in each snippet between <HL>..</HL> tags. The output file for BM25+stopping can be found at "Snippet_BM25Stopping_7thRun/snippets.txt".

For some documents, no snippet could be generated as no contiguous significant words were found.

Results

The values of each retrieval model is presented in the table below.

	BM25	TF-IDF	CosineSim	Lucene	TF-IDF + Pseudo Relevance	BM25 + stopping	TF-IDF + stopping
MAP	0.313	0.289	0.387	0.412	0.168	0.395	0.331
MRR	0.561	0.537	0.643	0.680	0.272	0.654	0.572
P@5	0.304	0.227	0.323	0.365	0.123	0.373	0.265
P@20	0.161	0.139	0.203	0.200	0.106	0.220	0.174

Calculation for P@5 and P@20 can be found in code "evaluation.py" inside method "calculateMeasure()". The numbers shown above are the mean of all query results.

Conclusions and Outlook

Analysis of results

Among the baseline models effectiveness order is as follows: Lucene(Simple Analyzer) > CosineSim > BM25>TF-IDF. Lucene and CosineSim are clear winners here due to high MAP and MRR. One reason that CosineSim outperforms BM25 is the usage of good term weighting scheme and value of k2(100) being small. The given queries are quite large and a higher value of k2 could have produced better results.

BM25 and TF-IDF performs better than CosineSim(baseline) when combined with stopping as the weightage of common terms is negated, thus shortening the query which is an ideal situation for BM25 and Tf-Idf models.

Tf-Idf with pseudo relevance produces a very interesting result. Its effectiveness decreases with expanded query. This highlights two things about pseudo relevance:

1. It is effective only if the base retrieval model is extremely effective.
2. It, in fact produces more bad result if the base model is not capable of retrieving large chunk of relevant documents at top ranks.

Conclusion

Clearly, the best systems are BM25 with stopping and Lucene (SimpleAnalyzer). BM25 could have outperformed Lucene if a higher k2 value is used. The results are too close that it gives out the impression that Lucene is using a similar, if not equal, variation of BM25 model.

Outlook

Possible improvements to the current implementation:

1. Use a higher value for k2 parameter in BM25 implementation.
2. Removing the non-word (digits etc.) types from indexes.
3. Pre-computing the term weights and other possible values to increase the query throughput.
4. Use a better query expansion technique than pseudo-relevance.

Bibliography

[1] Vector Space Model :http://cogsys.imm.dtu.dk/thor/projects/multimedia/textmining/node5.html

[2] Apache Lucene : http://lucene.apache.org/core/

[3] TF-IDF : http://www.tfidf.com/

[4] Pseudo Relevance Feedback: http://nlp.stanford.edu/IR-book/html/htmledition/pseudo-relevance-feedback-1.html

[5] “Information Retrieval in Practice” by W. Bruce Croft, Donald Metzler, Trevor Strohman, Chapter 6, page 216