Optimizing IR models

Srinivas Reddy M
IIT Kharagpur

Abhishek Kalyan A
IIT Kharagpur

Implementing machine local search engine to search 1.3 millionTREC (Text Retrieval Conference) documents using Apache Lucene.

Indexing

org.apache.lucene.search/index/queryparser.classic

Analyse each document and make content index and store it in easily retrievable format.

If document is readable and exists then create and append it to the IndexWriterConfig. Use iwc.setRAMBufferSizeMB for extra ram or distributed indexing so that computation doesn't fall short of RAM. For directory get all component files, recursively call TrecDocIterator and iterate with it's next method.

Using the following lucene defined entities:
IndexReader IndexSearcher Analyzer QueryParser

TrecDocIterator

Compile a pattern with tags needed, consider various parts of the document and give them various weightages if needed.

If Matcher matches the line to the compiled pattern store stringfield "docno" with document number and store text field "contents" with contents.

Search engine

org.apache.lucene.search/index/queryparser.classic

Initialise 50 Stop words taken from Google. Use narratormap to map QUERY to integer index, searchindex for word to it's equivalents. Tokenise them into words or strings into stringbuilder and get string from stringBuilder and add it to narratormap followed by cleaning it.

For each query parse to QUERY and search through indexed files to find potential matches into TopDocs. Save each query's matches retrieved, correspoding overlap extent into a arraylist of tuples.

rankedRetrival

org.apache.lucene.search/index/queryparser.classic

Uses a Hashmap and a NarrMap. Read from query file from betwee <top> and </top> and tokenise it into string builder, put it in hashmap followed by cleaning the stringbuilder. For each query get hash map and parse it to query and search IndexSearcher for query, get Topdocs from there for all manipulation. From searchengine get list of more similar matches ,get matches hits from searcher.doc and match with most relevant answers for queries from searchIndex of search engine. if searchIndex contains the document give 1 else give 0.

Use PrecisionCalc class.for statistics like

$$P_5, P_{10}, P_{20}, P_{100}, P_{500}, P_{1000}..$$

Just count number of ones in retrieved document's scores ranked 0 to 'k' and normalise it. for map repeat it for each 'k' within our interest.

IR Models

Models Used are :
1.Vector Space model
2.Language Model
3.Okapi BM 25
Models can beselected by using IndexSearcher.setSimilarity method.

Vector Space model. Each string is split into a vector with words or tokens as base vectors. Similarity between ant two strings is the cosine of angle between them.

This is used to rank and compare performance of various methods.

Language Model. A document is a good match to a query if the document model is likely to generate the query, which will in turn happen if the document contains the query words often.

The original and basic method for using language models in IR is the query likelihood model . In it, we construct from each document d in the collection a language model M_d. Our goal is to rank documents by $P(d|q)$, where the probability of a document is interpreted as the Bayesian likelihood that it is relevant to the query.

OKapi BM25. The simplest score for document d is just idf weighting of the query terms present, as in Equation :

$$RSV_d = \sum_{t \in q} \log \frac{N}{docf_t} \quad (or) \quad RSV_d = \sum_{t \in q} \log \frac{N - docf_t + \frac{1}{2}}{docf_t + \frac{1}{2}}$$

If we start with the absence of relevance feedback information we estimate that $S = s = 0$, then we get the alternative idf formulation.