Luna Szymanski
11/6/2019
Information Retrieval

HW3 - Build your own indexing and query a TREC data set

Compiling Instructions

1. Add the ap89 documents to the folder src/ap89_collection
2. Run Index.Java
3. Output is RESULTS.TXT

In RESULTS.TXT you will see a query from the 51-100 querylist, followed by 50 documents and their relevance according to the qrels51-100. Then the Precision at 50 is calculated before moving on to the next query.

Running the index will take approximately 7 minutes, and searching takes about 30 seconds, depending on your machine.

Indexing Design

For this assignment I used several HashMaps to index the ap89 document collection. I knew that I wanted each document represented as a set of weighted term frequency values so I used a HashMap<DOCID,<HashMap<term,tf>>. For each DOCID I made a seperate map that stored each term as a key and the term frequency value as the value. I thought it would be wise to use a HashMap because of the constant cost of searching. I knew that searching and inserting would be common operations so I thought it made sense to use a data structure that has cheap searching and insertion costs.

I also decided to keep a separate HashMap for my document normalization scores. I decided to calculate these while indexing as to have these scores available at constant time during future tf * idf calculations.

I also kept a termMap that had several pair structures in it but I barely used this, only to calculate df. In the future I would simplify this data structure a lot to avoid using custom types because I think it makes it hard to read.

Scoring Design

In order to determine if a document is relevant I calculated a score using vector space analysis and cosine similarity. To accomplish this I used vector representations for my queries and my documents, and then compared based off of those vectors. Each query and document was transformed into a HashMap<term,tf> where the term is the word, and tf is the term frequency of that word in the query or document. In addition I created an idf HashMap to keep track of the idf for each word.

For each term frequency I did 1+log(tf) to weigh the term frequency, and then multiplied it by the idf which i calculated using log10(N/df). This gave me a tf*idf score for each term in each query.

For each document I also made a HashMap<term,tf> and applied the same weighing method to the tf, 1+log(tf). I then normalized this by the normalization factor i stored in my seperate map.

To create a final score for each document I multiplied my query tf*idf score by my document normalized tf score for every term in the query. The sum of these scores per document gave me the relevance score, and all that was left to do was to sort by my relevance score.

Precision Calculation

In order to calculate precision I compared my top 50 results for each query to the results listed in the qrels51-100 text file. If the qrels marked the document that i returned in my top50 as relevant I would mark it as relevant, and my precision score represents the total # of relevant documents divided by the total # of documents returned(constant of 50).

I was very surprised to see some scores as high .62, meaning that 31 out of the 50 documents i returned were marked as relevant by the qrels. Others were not precise at all with scores of 0.0.