Probabilistic-R Summary

Zachary McFarland - https://github.com/zwmcfarland/ProbabalisticR

In this project I explored three different methods of generating the initial relevant document set for the probabilistic method of information retrieval. The results of each experiment were compared and considered for their strengths, weaknesses, differences, and similarities. This paper will discuss the method for setting up each experimental method, challenges that arose during implementation and how they were overcome, and the results that were realized after running these experiments and what they mean.

The Probabilistic model for information retrieval calculates the probability that a document is in the set of relevant documents pertaining to a query. The documents in the relevant document set are then ordered based on the probability that they are relevant to the users query. The focus of this experiment was to explore different means of generating *R* or the initial set of relevant documents. This initial set of relevant documents is used by the probabilistic model as a starting place to iterate from to generate its set of documents that are probably relevant to the users query. In order to calculate the probability that a document is relevant to a query the probabilistic model assigns weights to each term based on the frequency of that term in the document set and the number of documents in the set plus some normalizer, in our experiments we used 0.5 which we will discuss later. Using the term weights the probabilistic method then inspects each document to see if a query term is present. If a document contains a query term the weight of the query term is multiplied by the current relevancy of the document. This process is repeated until all documents containing query terms are scored. After the process of scoring documents is complete the document with the largest weight is considered the most relevant.

To calculate *R*, the initial relevant document set, I explored 3 different methods; the Vector model, the Boolean model, and the inclusive model. Each model's name is intended to be obvious as to the means to which they calculate the relevant document set. The Vector model uses the Vector model for information retrieval to identify relevant documents. Using the Boolean model, we generate *R* by using the Boolean model of information retrieval, which when broken down simply sets R to the documents that contain any of the query terms. Lastly the inclusive model simple sends all documents in the document set as R.

The inclusive model is the simplest model that was implemented, and can be considered trivial. It can be viewed as the control for these experiments as the inclusive model only filters documents using the probabilistic model. This model, as with all other models passes R on to a single implementation of the probabilistic model, only in this case the initial relevant document set is equivalent to the entire document set.

The Boolean model was implemented in a fairly straight forward manner. The typical implementation of the Boolean model constructs a term document matrix that identifies which terms are in which documents, and then ands the documents together with the query term vector. Since this project has a set number of queries, and are not running at web scale, this method was not necessary. Instead we were able identify documents at runtime instead of having a pre-computed term document matrix. Using the Boolean model of information retrieval we went through each document looking for any of the terms in the provided query, if a term was found in a document the document was added to the initial relevant document set R. After all documents were considered the list of initial relevant documents were sent to the probabilistic model implementation along with the collection of all documents. The set of all documents must be sent along with the set of relevant documents as it is needed in the probabilistic model for term weighting purposes, to be elaborated upon later.

The Vector model was the most complicated model that was implemented. The Vector model of information retrieval was used to find the initial set of relevant documents, *R*, for the probabilistic model. In order to identify relevant documents the Vector model constructs document vectors in *T* dimensional space, *T* is equal to the total number of terms in the set of all documents. The values in the document vector are set to the 0 if the term doesn't appear in the document, or the term frequency over the inverse document frequency for each term in the document. Then the vector model also creates a vector representing the query. For each of the document vectors the model calculates the cosine distance between the query vector and the document vector. In this projects implementation I moved reduced some of the processing and memory overhead by representing the document vectors as maps from term to term frequency over inverse document frequency for each term in the document instead of each term in the entire document set. After each is considered we consider the distance of the document vector to the query vector, and have the opportunity to filter out distances that we deem too far away and therefore irrelevant. I tried using distances of 0, 10, and 100 results of these different values will be discussed in the results. After this consideration the initial relevant document set, *R*, is formed and forwarded on the probabilistic model implementation along with the entire set of documents.

Lastly the probabilistic model was implemented in this project in such a way that it could accept input from all three of the other methods of generating the initial set of relevant documents, *R*, and provide detailed information on how the different values of *R* effected the probabilistic method. The original proposal for this project stated that I would create a ranking score for the results of the probabilistic method, and how well it performed utilizing the differentiated initial *R* values. However after working with these three methods and seeing the output of the probabilistic method implementation it was obvious that each model would have its own strengths and weaknesses and any single value score I could provide would not be capable of capturing the key differences of each method. Therefore I opted to log a large amount of information during the execution of the probabilistic method that allowed me to make clear observations about what effects each model was having on the probabilistic methods final outcome. A few pieces of important information that were captured were as follows:

- The number of relevant documents in the initial set, *R*, for each method
- The performance, in execution time, at which the iteration of the probabilistic method executed
- The number of relevant documents identified by the probabilistic method
- Whether or not the highest rank document matched the intended document for the query
- The probability that the document was relevant to the given query

With this information we are able to see some significant differences in the way the initial set of relevant documents, *R*, is effecting the execution of the probabilistic method.

During implementation of this project we did run into several issues, primarily regarding scale and performance. The 20 newsgroups dataset provided for this project was quite large containing 358964 keywords and 2000 documents. Two of the models, the inclusive model and the Boolean model, were not an issue as they didn't have any additional data structures outside of the ones required to hold the dataset. The issues I ran into were all revolving around the implementation of the vector space model. The vector space model depends on a value for each term in all documents called IDF or inverse document frequency. The inverse document frequency takes into considerations how often a word occurs in all documents, this helps identify common terms that shouldn't be weighted as highly as very specific words in few documents. In my initial implementation I was trying to calculate the IDF at the same time as calculating the document vectors which was resulting in an additional load of t^2, where *t* is the number of terms in all of the documents. I was able to remove that t^2 load by pre-calculating an "IDF Map" that contained the inverse document frequency for every term. This allowed me to access the inverse document frequency in constant time and significantly reduce the computational requirements. However by creating the IDF map I was also increasing the memory footprint of the execution by adding a map of over 350,000 keywords and weights. While creating the 2000 document vectors, each 350,000 plus terms long, were alongside the inverse document frequency map the program could not complete with the limited amount of space I had available on my machine. To shrink the memory requirements of the program, and serendipitously decrease the computational load, a reduction of $t – D(t)$ where t is the number of terms in all of the documents and D(t) is the number of terms in a single document, we switched the document vector from a list of 350,000 plus weights, of mostly 0 to a map of term to weights for only the terms that were found in the document. By doing this I was able to vastly reduce the memory requirements and successfully generate a vector model index.

In conclusion I found results somewhat to the contrary of what I expected. In my project proposal I had stated that I expected to find the vector model the superior method of calculating the initial relevant document set, *R*. After experimentation I have found that the method to choose depends on your use case. The short coming of the Vector model method is that the document vector index is extremely expensive to compute, even after making numerous optimizations. However if your use case allows you to "pre-compute" the document vector index it does narrow down the initial set of relevant documents, *R*, to the smallest number, depending on the settings for document cosine distance that you choose. I found that choosing a cosine difference constraint of 0 for the vector model made this model equivalent to the Boolean model method. In contrast if you set the value to 100 you run the risk of filtering out all documents to be sent to the probabilistic model. These results are of course unique to the 20 newsgroups dataset, I think with a different dataset, different distance constraints would be needed. For this dataset a cosine distance constraint of 10 was sufficient. Overall each model that was explored was able to identify the same "most" relevant document, assuming the vector model didn't have a distance constraint too high and filter out all documents. The main difference between the methods assuming you could pre-calculate your document indexes, was performance. The vector space model provided the best platform to get the core set of relevant documents the quickest, providing the smallest set for the initial set of relevant documents, *R*, which in turn made the probabilistic model execution faster. The Boolean was the second fastest, and assuming the document index could not be pre computed, the Boolean model would be the most efficient method to generate *R*. Lastly in my initial project proposal I estimated that it would be the inclusive model that would have the highest eventual accuracy as it wouldn't miss any documents. I think I mis-understood the way the probabilistic method weighted terms. I thought that the term weights would be calculated based on the terms in the relevant document set, however as it is weighted based on the terms in all documents relevant and not, all methods produce the same final result, however some methods may have fewer final relevant documents depending on the documents in *R.* Overall I think I learned quite a lot from this project from how to implement information retrieval models, to how to optimize data processing techniques in Java.