Information Retrieval and Analysis
OCTOBER 2020

Lab Session 4: User relevance feedback

PROJECT REPORT

Elías Abad Rocamora
Victor Novelle Moriano
Barcelona, UPC - FIB & FME

Rocchio's rule implementation.

In this laboratory we were asked to implement a Pseudo-revelance feedback system that uses Rocchio's rule to improve the query provided by the user. In order to do so, we have generated the *lab4.py* code as well as used the code *TFIDFMod.py*, previously implemented in lab 3.

The first step consists in resolving the original query introduced by the user and retrieving the *k* most relevant documents. After computing their respective TF-IDF's weights, an average of them is calculated (using dictionaries to achieve a better performance, as we will see in the following section), and the obtained result is combined with the original query. The weight of the terms of the user-introduced query and the new ones found on the results are determined by the α and β parameters, which are introduced by the user. When this process is finished, the top *R* terms with higher weight are selected (using a priority queue to boost performance) and are used to define a new query. This refinement process is repeated iteratively *nrounds* (which is also defined by the user). After the iterations are completed, the evolution of the query is shown to the user, as well as the k most relevant documents obtained and the final number of files that have matched the modified query.

Computational cost of Averaging vectors

The cost of averaging vectors is influenced by two parameters:

- k := Number of documents to average
- n := Number of different words within all documents

If we implement the average of the documents using a Dictionary, the total cost will be asymptotically:

O((k-1)*n) = O(k*n)

Where O(n) is the length of a document and, as average, the cost of looking for a key inside the final document is O(1) (due to the hash implementation of dictionaries in python). So the cost of merging two dictionaries is O(n). Because we have to do this operation k-1 times, the total cost will be the one given above.

If this operation had to be done with lists, the cost would be:

O(k*n*log(n)) (Sort every list) + O(k*n) (merge k-1 lists)

As the first term dominates when n gets bigger, the asymptotic cost would be:

O(k*n*log(n))

Which will be worse than when using dictionaries.

Experiments and results.

To test the performance of the Pseudo-relevance feedback, we searched some generic terms in order to see if the new terms suggested had a strong relationship with the original ones.

Influence of K:

With R = 3, nrounds = 5, α = 1 and β = 0.5:

Original query	Final query (K = 5)	Final query (K = 50)
space	[nasa,data,space]	[nasa,station,space]
god	[god,creates,he]	[god,jesus,he]
science	[percent, origin, science]	[venus, surface, science]
city	[city,mwra,water]	[City,inner,hockey]
music	[music,n,deaf]	[hrivnak,gtd597a,deaf]

As it can be seen in the table above, when k = 5, the obtained results are pretty satisfactory. The first four terms, that are quite generic, derive in more specific words related to the original topic. However, in the *City* and *Music* case, the obtained results are too specific. The final words come only from one document in both cases, which relates to the *Massachusetts Water Resources Authority* in the first case and a survey discussing the implementation of musical education of deaf kids in the second.

As for k = 50, the words in the final query are more closely related to the original one. But some words as music, lead to quite strange results as “gtd597a”, which is an email address. We think this problem is due to the document repository being too small or specific.

Influence of α and β:

The values of α and β determine how conservative is the algorithm when modifying the query. When we assign higher values for α, the terms present in the new query are dominated by the ones in the previous query, achieving a more static query. However, if β is the predominant term, the new queries will be formed by taking more importance to the query results, meaning that the user-introduced terms could be lost.

Influence of R:

As we increase the number of words in the query, the number of documents that our algorithm returns is much lower because every word in the query has to appear in the resulting documents. This leads to an increase in precision but a decrease in recall.

Influence of nrounds:

After performing several experiments modifying the *nrounds* parameter, we could not see a large variation in the results obtained, both resulting documents and query weights are quickly stabilized. If the code is used with a large value of k, the number of documents that fulfill the constraints is very small. That produces that,even applying more rounds to the algorithm, the obtained documents are the same. However, if the k parameter is small, we can see some variation in the initial rounds, but this is mainly produced by the values of α and β and not the *nrounds* per se. We could avoid using this parameter by setting as a stop condition that the difference in the querywords’ weight doesn’t exceed a certain value ε.