**INFORMATION RETRIEVAL**

Project Part II

**Text-based searching application with Lucene**

Oğuz Birdal

Alp Tunçay

Homa Hassannia

17.12.2018

**Introduction**

In this project, we were given the task for implementing an additional functionality on top of what we achieved in the first part. We chose to implement Pseudo-Relevance Feedback using the Rocchio Algorithm. Our idea here was to expand the query that was issued by the user and then run the search algorithm once more so that users query would return more relevant results than before. Since Lucene does not support Query Expansion we had search for alternate implementations and based on those we implemented our Query Expansion algorithm. We also found an additional package which was built on top of Lucene that is for Query Expansion, LucQE, but it does not have many resources so we chose to stick with our implementation. After we implemented the Rocchio Algorithm, we proceeded with the evaluations of our method. For that purpose we chose to use Precision and Recall values.

In the following parts of the report, we will give you detailed information about the Pseudo-Relevance Feedback, Rocchio Algorithm and the evaluation method.

**Pseudo-relevance feedback**

In general, relevance feedback means that user is involved in retrieval process. User can send a query, and the system gives back a set of results. After that, user sends a feedback about the relevant and non-relevant documents, then system process a better result based on the user’s feedback and show the final result to him/her. You can see this kind of technique that is shown as “more like this” in some web pages.

There are three types of feedback such as: explicit feedback, implicit feedback and blind or “pseudo” feedback. We are going to talk about pseudo-relevance feedback that we use in our project.

This technique is an automatic method to perform query expansion and there is no interaction between user and system for getting feedback. So, user sends a simple query and then system assumes that top K files are relevant. After that, we do query expansion, and add these weighted terms from results to query. And finally system returns the most relevant documents.

There are several algorithms for doing this technique. We use Rocchio Algorithm to do that. It implements relevance feedback in VSM (Vector Space Model). You can see the formula in below:

: Expanded or Optimal query

: Original query

: Set of relevant result documents (which is top k results in here.)

: Set of non-relevant result documents

: Weights

Starting from, new query moves towards relevant documents and away from non-relevant documents. In this algorithm, negative term weight is ignored. It means that it is 0. So, the formula would be like this:

**Advantages and Disadvantages**

There are some problems that relevance feedback couldn’t solve it alone. For example: misspelling, cross-language information retrieval and mismatch of searcher's vocabulary versus collection vocabulary.

Misspellings: when user uses the wrong spell of a term that it is in the documents, relevance feedback is not useful.

Cross-language information retrieval: for documents in another language is not effective.

Mismatch of searcher’s vocabulary versus collection vocabulary: if the term that we search is different with the collection vocabulary, it will be not effective.

The other problem is that most users are unwilling to send the feedback and make searching process longer. So, they prefer to just receive their result immediately. Also, in this technique, system should spend much more time to analyze documents, so queries that are longer are usually slower. Then, the cost of the retrieval systems will increase. But we can reduce this problem by reweighting certain prominent terms.

However, using pseudo relevance feedback could improve some problems like user interaction. So, it automates the practical part of the relevance feedback and works on average.

But, the main problem of pseudo relevance feedback is “Topic Drift”. Topic drift is an issue that happens when we have irrelevant terms, which do not represent the intent of the user, that are fed into the algorithm and our original query is completely lost after the calculations. If there was a query Drugs in Soccer and the feedback documents talked about “Maradona and his use of drugs” then there would be a drift of the query from “Drugs in Soccer” to “Maradona”.

So, documents in the collections can influence the intent of the feedback model. The more irrelevant documents that it has, the more problem that will have. If we take terms from these irrelevant documents and feed them into the Rocchio Algorithm, in the end we may obtain a query which is far off from the original query that was issued by the user. So it is important to be careful when choosing relevant documents.

**What We Have Done**

We have chosen Pseudo-relevance Feedback method as a query expansion method due to its ability to work without any feedback from user. Our strategy was taking the top thirty relevant documents based on our scoring algorithm in the first part of the project and expanding the original query according to the vector which created from the top thirty relevant documents. To expand the original query Rocchio Algorithm is being used. After creating the new query with new words our search algorithm works again and retrieves documents that are hopefully more relevant to the user.

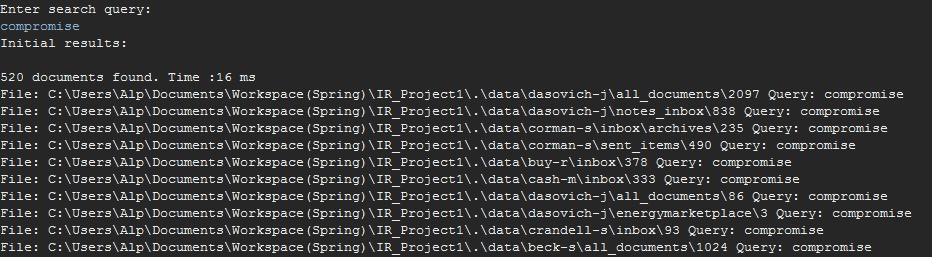
A new class named “Rocchio.java” added to our application to implement pseudo-relevance feedback with Rocchio Algorithm. In the previous version of the application, it was just capable of finding the exact query in the dataset. This class handles the all query expansion methods and implements the Rocchio Algorithm afterwards. In this class, we have all the necessary methods for expanding our initial query. We implemented this class in a way that the number of terms after the expansion of the query is limited to five. This is mainly because after applying the Rocchio algorithm, we receive a very long query and it is hard to process this query, even impossible sometimes. In our application, we return the top *k* of the found documents for better results. For applying the Rocchio algorithm, we assume that at least half of those top *k* are relevant to the initial query.

After obtaining the assumed relevant documents we turn them into vectors with TFIDF weights. This part was a bit problematic since we do not store the content of the document in the index. So we had to read and parse each file in the relevant documents and calculate the TFIDF weights for each term in the document. This part increases the complexity and decreases the run time efficiency. We also have to turn the query into a vector of TFIDF weights so that we can apply the Rocchio formula.

We applied the Rocchio algorithm in hopes to increase the number of relevant documents that are returned. What we observed was, the modified query returns more results than the initial query and the query sometimes may drift away from the original query. We wanted to prevent this drift away by limiting the number of query terms that are added. Still there might be circumstances where the expanded query is not representative of the initial query. The queries below have ten additional terms to the original query. These screenshots were taken before the change in implementation.

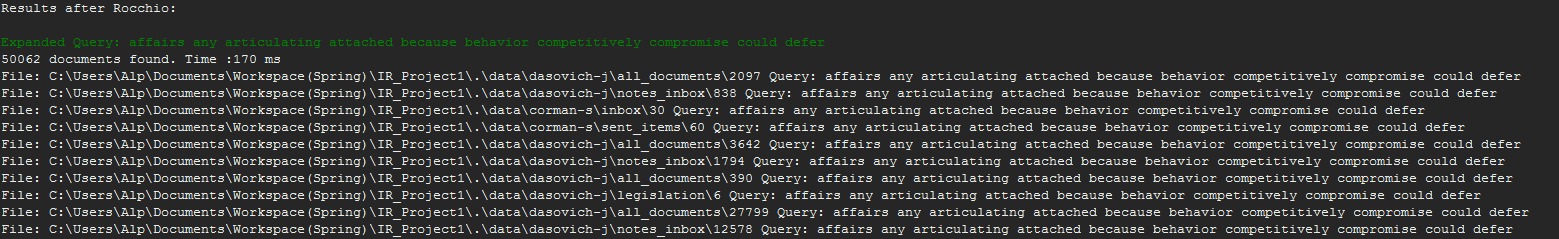
**Example Queries**

* **Query:** compromise

Results before query expansion

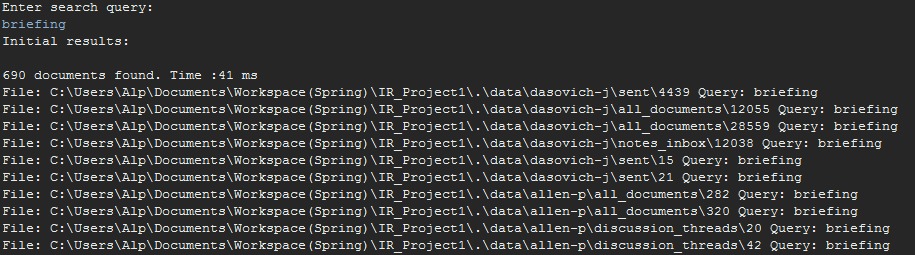
**Expanded query:** affairs any articulating attached because behavior competitively

New query and results after query expansion



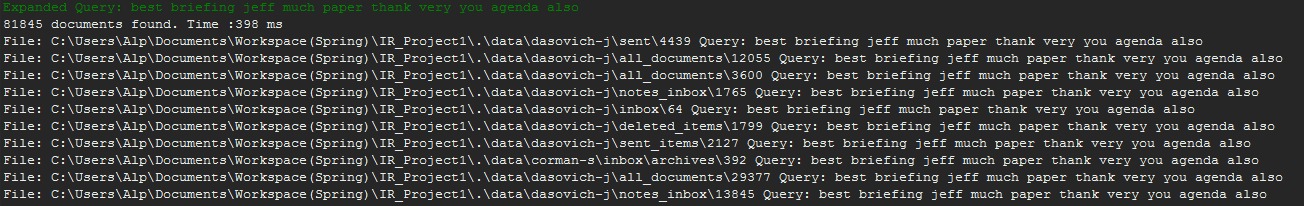
* **Query:** briefing

Results before query expansion



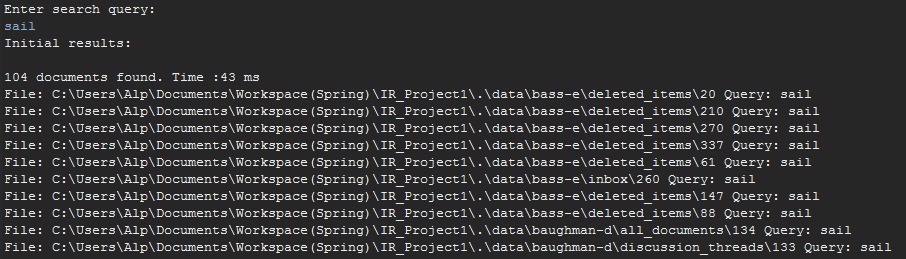
**Expanded query:** best briefing jeff much paper thank very you agenda also

New query and results after query expansion



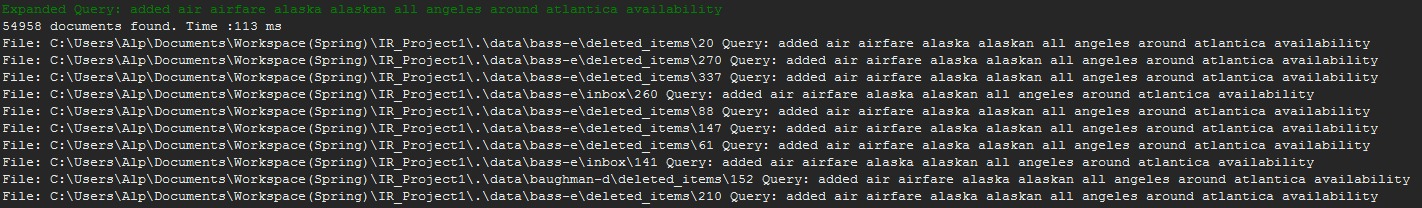
* **Query:** sail

Results before query expansion



**Expanded query:** added air airfare alaska alaskan all angeles around atlantica availability

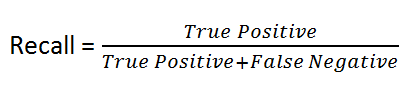
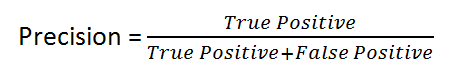
New query and results after query expansion



**Evaluation Strategy**

To evaluate the application we created a class named “PrecisionRecall.java”. For calculating the *precision* and *recall* values, we had to obtain the number of true positives, false positives and false negatives. Obtaining these values required some assumptions to be made. Since we have 103,256 files we could not label each and every one of those files with respect to the issued query. So for the initial querying process we assumed that the top *k* results that we obtain are relevant to the query that was issued. For the expanded query, we assume that the 30 of those top *k* results are relevant. The calculations for finding the number of false positives is based upon the aforementioned assumptions.

The false positives are the documents which retrieved but not relevant and the false negatives are the documents which are not retrieved but they should have been. Since we have 103,256 documents in the dataset, we assumed that 5 percent of the documents which are not retrieved are relevant and calculated false negatives based on this assumption. We used the formulas below to calculate precision and recall:



**Code**

Our application can be found on:

<https://github.com/birdaloguz/IR_project1>