VRIJE
UNIVERSITEIT
BRUSSEL

INFORMATION RETRIEVAL

Project - Group 3 (total of 3 group members)

Kenzo Clauw (0525128), Steven Vanden Broucke (0526527) and Badreddine Hachoumi (0507890)

December 22, 2017

Sciences and Bioengineering Sciences

Contents

1 Introduction

As an assignment for the course Information Retrieval, we have build a system that allows the user to query on a large set of documents. Based upon the query we will use different techniques shown during this course to optimize the result in terms of quality and in terms of performance.

This was done by the implementation of the following aspects: Full Boolean querying supporting , , ! ,(,). Our system would automatically translate and process the combination of terms and symbols into an operational query which would provide the user with correct results. Take note that there is no inclusion of nested queries as this was not described within the assignment. Our system however is capable of handling more complex queries where the combination of all symbols within in query is possible.

Soundex deals with misspelling and improve recall; Permuterm index over the dictionary for * wild-cards; Ranking based on Euclidean (using heap to get top k docs); Improved ranking by using Cosine instead of Euclidean; Clustering using K-means.

2 Datasets

For development purposes we used a sample of 30 HTML pages of consisting out of articles from the Communication of the A CM journal. Afterwards we used it in on a larger set of 1000+ pages to confirm our working functionality. However some aspects, like clustering evidently produced a better result using larger sets of data during the development process.

We have noticed that the clustering effect on the CACM dataset was not as pronounced when compared to a dataset on news items for the past couple of years. This was solely dependent on the type of content. The clustering functions perfectly but its effect is less visible to the human eye when its about small-page journal data. A sample of the CACM dataset is shown below.

```
<html>
<pre>


Two Square-Root Approximations

CACM November, 1958

Wadey, W. G.

CA581102 JB March 22, 1978  8:33 PM

5         5         5
5         5         5
5         5         5

</pre>
</html>
```

```
<html>
<pre>


Request for Methods or Programs

CACM April, 1958

Corley, H. P. T.

CA580402 JB March 22, 1978  9:18 PM

28        5         28
28        5         28
28        5         28

</pre>
</html>
```

3 Technology

Our system is primarily build on **Java** using **Apache Lucene** as a major backbone. The system takes advantage of the standard build-in properties of Lucene that allowed us to focus on the implementation of the aspects rather then investing time in the actual retrieval of documents and counting terms. Take note that there was no feature that was solely on Lucene. For every individual aspect we have build a abstract layer that allowed us to test all of our programmed aspects.

4 Architecture

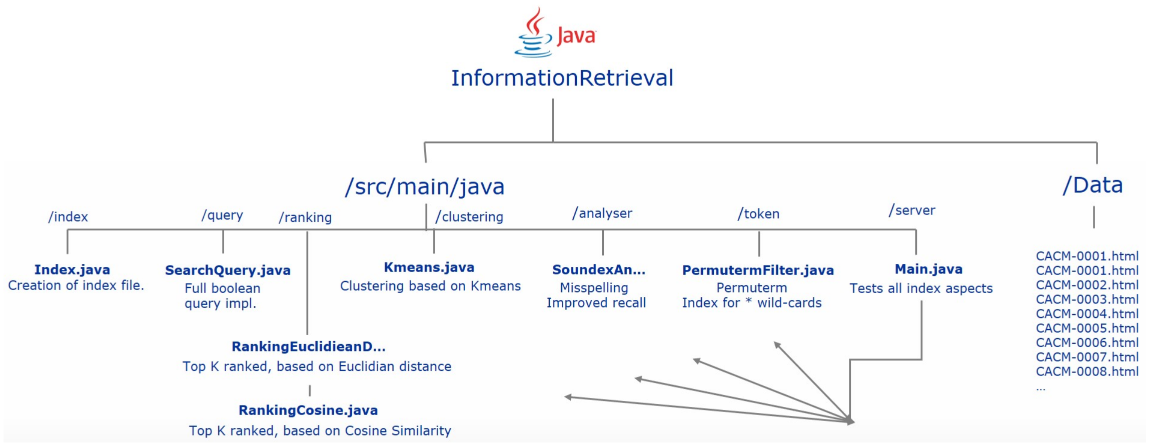


Figure 1: Overview of the Information Retrieval Project Structure

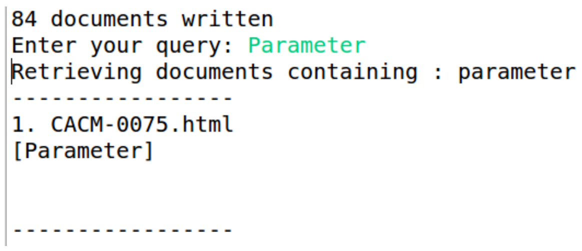
The architecture of our system is simple and straightforward. We have a singular file in /server, namely **Main.java** which is capable of testing and executing all of our programmed aspects such as: Full Boolean querying supporting , , ! ,(,). Soundex deals with misspelling and improve recall, Permuterm index over the dictionary for * wild-cards, Ranking based on Euclidean (using heap to get top k docs), Improved ranking by using Cosine instead of Euclidean, Clustering using K-means.

Take note that we can easily interchange the input dataset to test for the efficiency of our project if it is dependent on the source or not.

5 Boolean Query

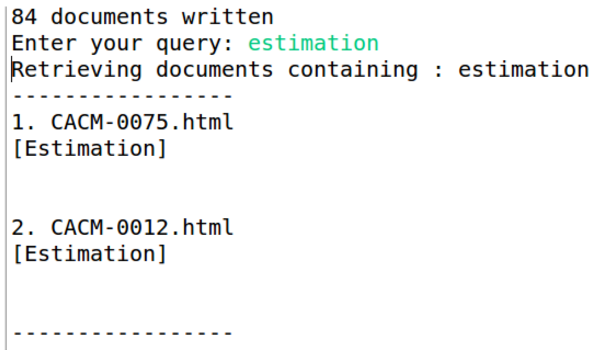
The assignment described to included the functionality of AND, OR and NOT within boolean queries. Additionally we also have added bracket functionality which makes the addition of brackets within queries possible. This then again causes a broader range of complex queries to be formed and executed. The implementation can be found in /query/SearchQuery.java. The following figures will provide some input queries accompanied with their results proving the working functionality of our system.

5.1 Single term boolean query



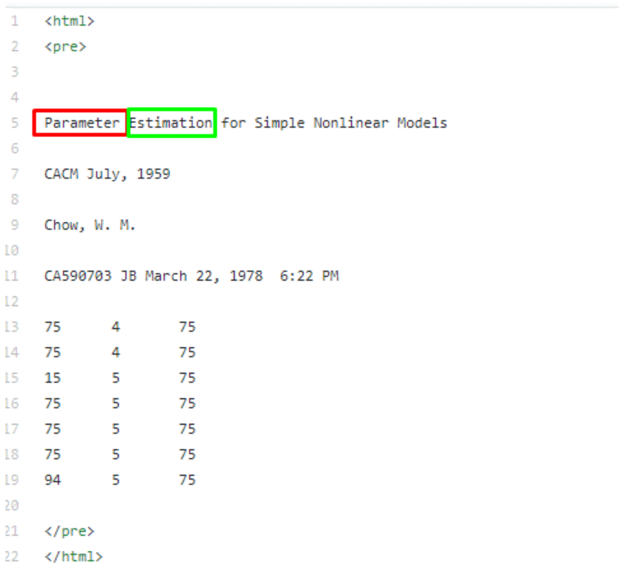
```
84 documents written
Enter your query: Parameter
Retrieving documents containing : parameter
-----------------
1. CACM-0075.html
[Parameter]


-----------------
```

Figure 2: Single boolean query term **Parameter**



```
84 documents written
Enter your query: estimation
Retrieving documents containing : estimation
-----------------
1. CACM-0075.html
[Estimation]


2. CACM-0012.html
[Estimation]


-----------------
```

Figure 3: Single boolean query term **estimation**

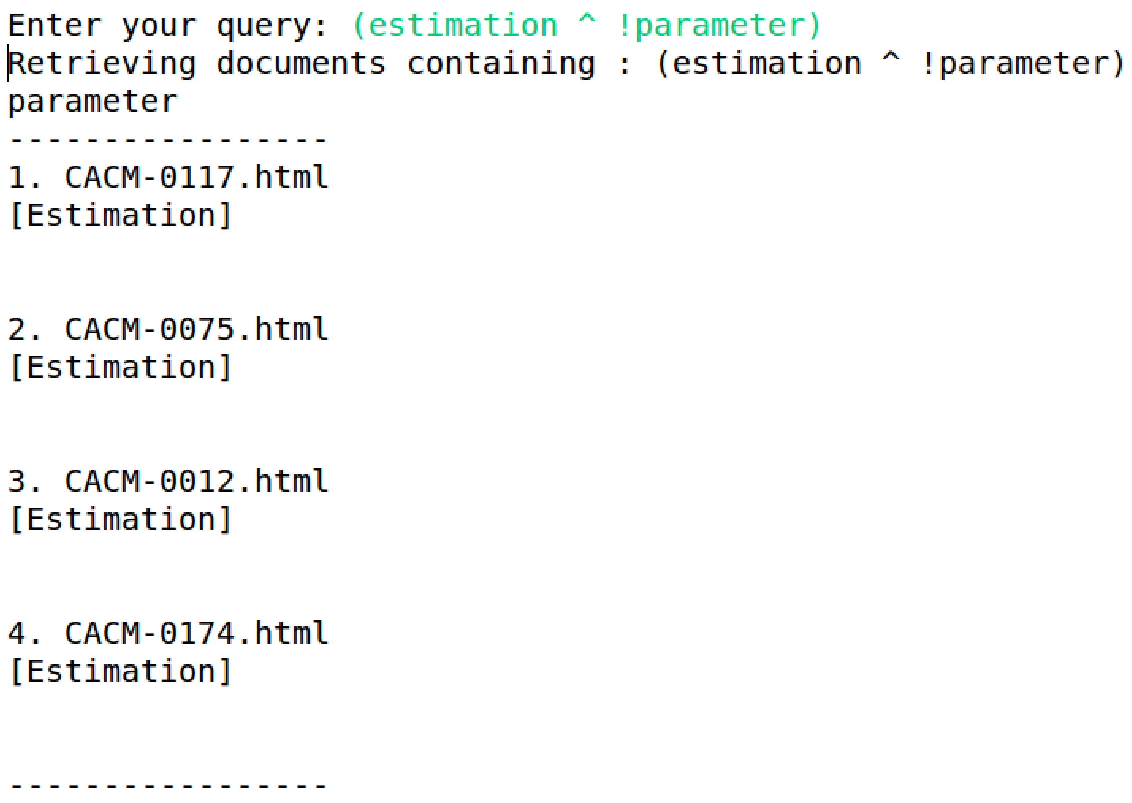


```
<html>
<pre>


Parameter Estimation for Simple Nonlinear Models

CACM July, 1959

Chow, W. M.

CA590703 JB March 22, 1978  6:22 PM

75      4       75
75      4       75
15      5       75
75      5       75
75      5       75
75      5       75
94      5       75

</pre>
</html>
```

Figure 4: Result Single boolean query term **estimation** and parameter

Given a single term search the user gets presented with a list of files containing the word at least once. This can be seen in Figures 6 and 3. Figure 4 indicates one of the resulting files: CACM-0075.HTML. Containing both the terms **parameter** and **estimation**.

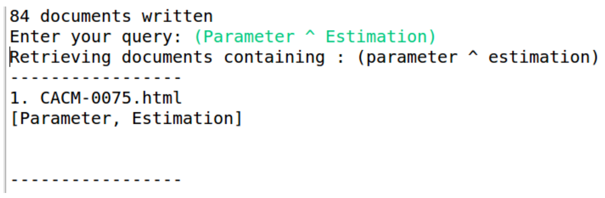
5.2 Boolean query using the NOT operator.



```
Enter your query: (estimation ^ !parameter)
Retrieving documents containing : (estimation ^ !parameter)
parameter
-----------------
1. CACM-0117.html
[Estimation]


2. CACM-0075.html
[Estimation]


3. CACM-0012.html
[Estimation]


4. CACM-0174.html
[Estimation]


-----------------
```

Figure 5: Result of the not query

5.3 Two term boolean query using the AND operator

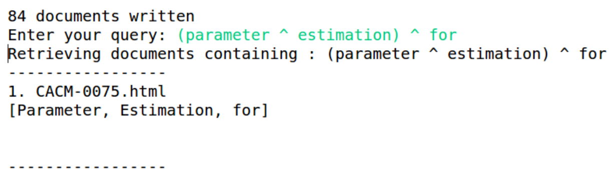


```
84 documents written
Enter your query: (Parameter ^ Estimation)
Retrieving documents containing : (parameter ^ estimation)
-----------------
1. CACM-0075.html
[Parameter, Estimation]


-----------------
```

Figure 6: Two term boolean query

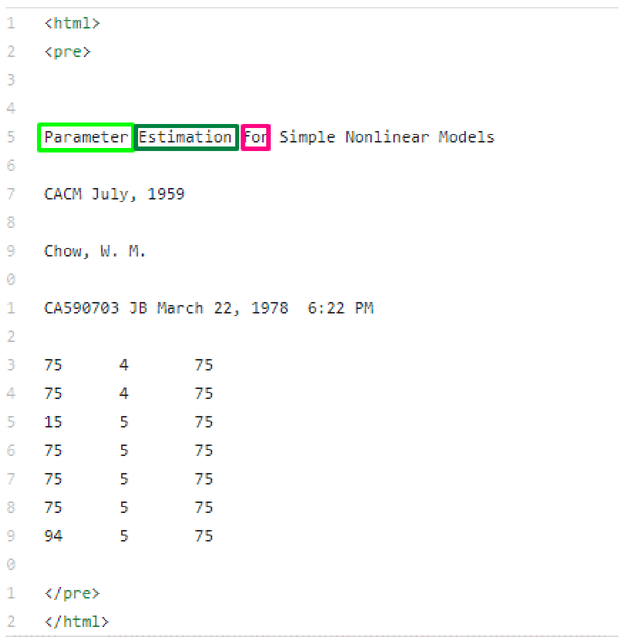
This again providing the same file **CACM-0075.HTML** as a result of the query.

5.4 Advanced boolean query using a combination of brackets and AND operators.



```
84 documents written
Enter your query: (parameter ^ estimation) ^ for
Retrieving documents containing : (parameter ^ estimation) ^ for
----------------
1. CACM-0075.html
[Parameter, Estimation, for]


----------------
```

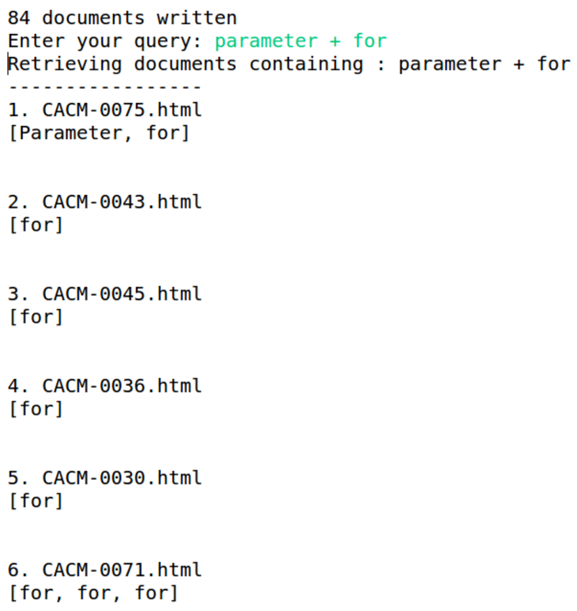
Figure 7: Two term boolean query with brackets and AND operators



```
<html>
<pre>


Parameter Estimation For Simple Nonlinear Models

CACM July, 1959

Chow, W. M.

CA590703 JB March 22, 1978  6:22 PM

75      4       75
75      4       75
15      5       75
75      5       75
75      5       75
75      5       75
94      5       75

</pre>
</html>
```

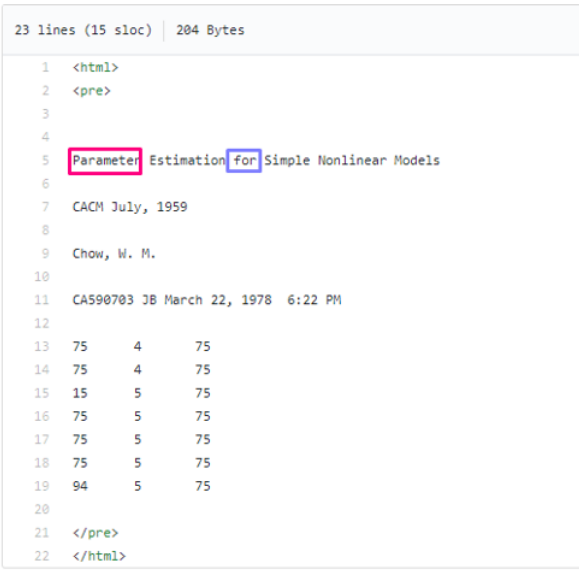
Figure 8: Result two term boolean query with brackets and AND operators

5.5 Boolean query using the OR operator.



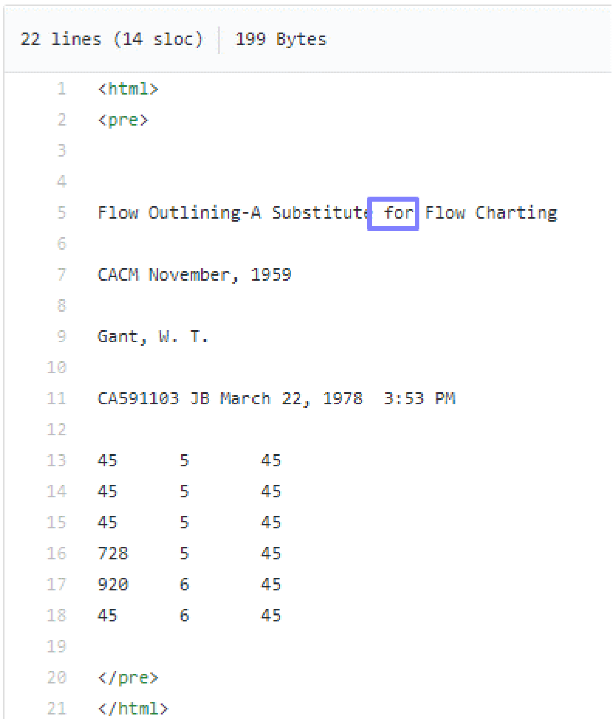
```
84 documents written
Enter your query: parameter + for
Retrieving documents containing : parameter + for
-----------------
1. CACM-0075.html
[Parameter, for]


2. CACM-0043.html
[for]


3. CACM-0045.html
[for]


4. CACM-0036.html
[for]


5. CACM-0030.html
[for]


6. CACM-0071.html
[for, for, for]
```

Figure 9: Two term boolean using an or (+) operator.



```
23 lines (15 sloc) | 204 Bytes

<html>
<pre>


Parameter Estimation for Simple Nonlinear Models

CACM July, 1959

Chow, W. M.

CA590703 JB March 22, 1978  6:22 PM

75      4       75
75      4       75
15      5       75
75      5       75
75      5       75
75      5       75
94      5       75

</pre>
</html>
```

Figure 10: Result one for boolean OR query.



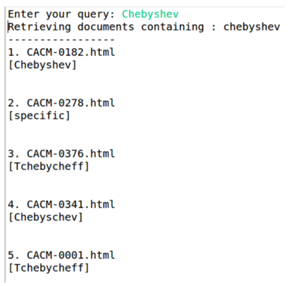
```
22 lines (14 sloc) | 199 Bytes

<html>
<pre>


Flow Outlining-A Substitute for Flow Charting

CACM November, 1959

Gant, W. T.

CA591103 JB March 22, 1978  3:53 PM

45      5       45
45      5       45
45      5       45
728     5       45
920     6       45
45      6       45

</pre>
</html>
```

Figure 11: Result two for boolean OR query.

As we can see in Figure 9 multiple results get produced. However, only one file containing both search terms, the rest of the resulting files containing only the keywords for. This is indicated with a sample subset of the result shown in Figures 10 and 11.

6 Normalization

For normalization purposes we make use of the following classes provided to us by Lucene: StandardTokenizer, StandardFilter, PermutermFilter, PhoneticFilter, LowerCaseFilter. This in turn caused and improvement of query results having a positive effect on misspelled queries. An example is given in Figure 12.



```
Enter your query: Chebyshev
Retrieving documents containing : chebyshev
----------------
1. CACM-0182.html
[Chebyshev]

2. CACM-0278.html
[specific]

3. CACM-0376.html
[Tchebycheff]

4. CACM-0341.html
[Chebyschev]

5. CACM-0001.html
[Tchebycheff]
```

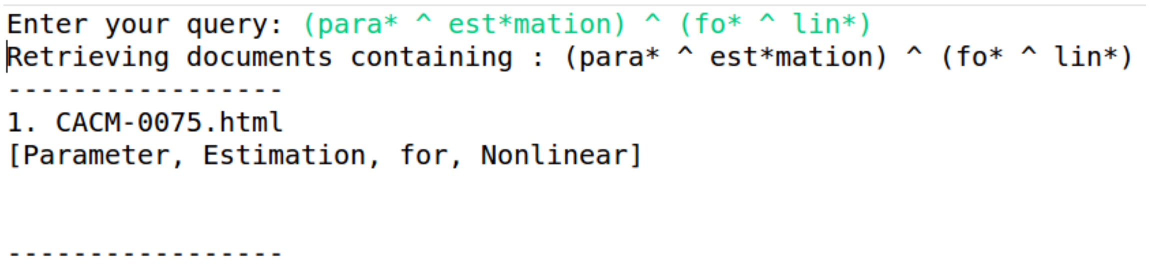
Figure 12: Normalization effect on a misspelled query search

7 Wildcards

For the wildcards we used Lucene's wildcard querying functionalities. We had some struggles since Lucene is not supporting different wildcards in one word, it only supports wildcards at the end of the word. We countered this problem by using permuterm. Permuterm will map every term with one wildcard to a word with multiple. With this it is possible to have multiple wildcards in one word. The down side of this is scalability. Since we search for each subset of a word.

7.1 Example

In this query we can clearly see the power of the wildcards. Note that the for us found even though it is a stop word, this is because we did not normalization to run our examples.



```
Enter your query: (para* ^ est*mation) ^ (fo* ^ lin*)
Retrieving documents containing : (para* ^ est*mation) ^ (fo* ^ lin*)
-----------------
1. CACM-0075.html
[Parameter, Estimation, for, Nonlinear]


-----------------
```

Figure 13: Result of a complex query using some wildcards

8 Ranking

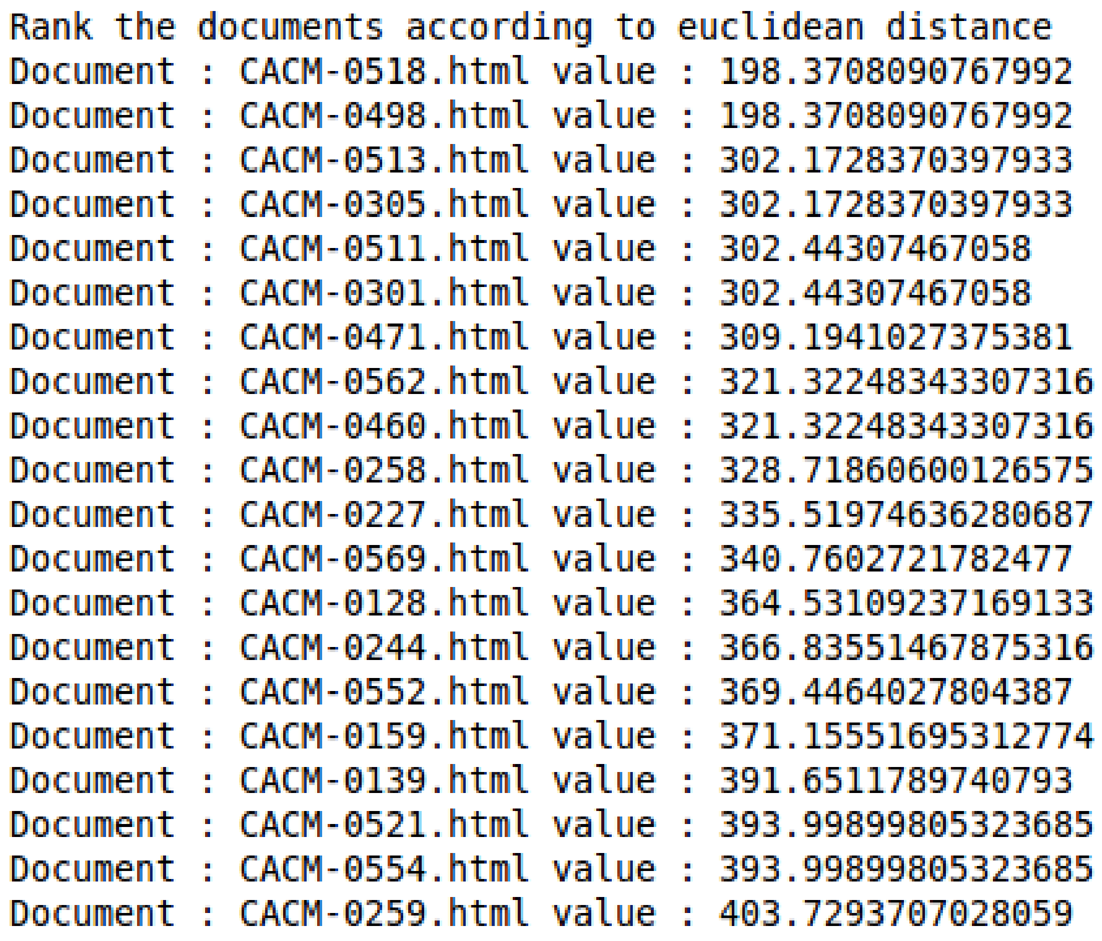
The ranking was implemented using Lucene's BM25 tf-idf scoring functionalities. This is because Lucene gives negative result when tested with a little document-set. This has led to some problems with our rankings and clustering, but was solved using the BM25.

8.1 Euclidean distance

In the listing below, we can find the implementation of the Euclidean distance calculation. For each file we take the square difference of all the terms and sum them. To finish it off we take the square root of that sum.

```
private static double GetEuclideanDistance(List<Double> file1,
                                            List<Double> file2) {
        double diff_square_sum = 0.0;

        for (int i = 0; i < file1.size(); i++) {
                double x = file1.get(i).doubleValue();
                double y = file2.get(i).doubleValue();
                diff_square_sum = diff_square_sum + (x - y) * (x - y);
        }
        return Math.sqrt((double) diff_square_sum);
}
```

8.1.1 Example



```
Rank the documents according to euclidean distance
Document : CACM-0518.html value : 198.3708090767992
Document : CACM-0498.html value : 198.3708090767992
Document : CACM-0513.html value : 302.1728370397933
Document : CACM-0305.html value : 302.1728370397933
Document : CACM-0511.html value : 302.44307467058
Document : CACM-0301.html value : 302.44307467058
Document : CACM-0471.html value : 309.1941027375381
Document : CACM-0562.html value : 321.32248343307316
Document : CACM-0460.html value : 321.32248343307316
Document : CACM-0258.html value : 328.71860600126575
Document : CACM-0227.html value : 335.51974636280687
Document : CACM-0569.html value : 340.7602721782477
Document : CACM-0128.html value : 364.53109237169133
Document : CACM-0244.html value : 366.83551467875316
Document : CACM-0552.html value : 369.4464027804387
Document : CACM-0159.html value : 371.15551695312774
Document : CACM-0139.html value : 391.6511789740793
Document : CACM-0521.html value : 393.99899805323685
Document : CACM-0554.html value : 393.99899805323685
Document : CACM-0259.html value : 403.7293707028059
```

Figure 14: Result of our Euclidean based ranking

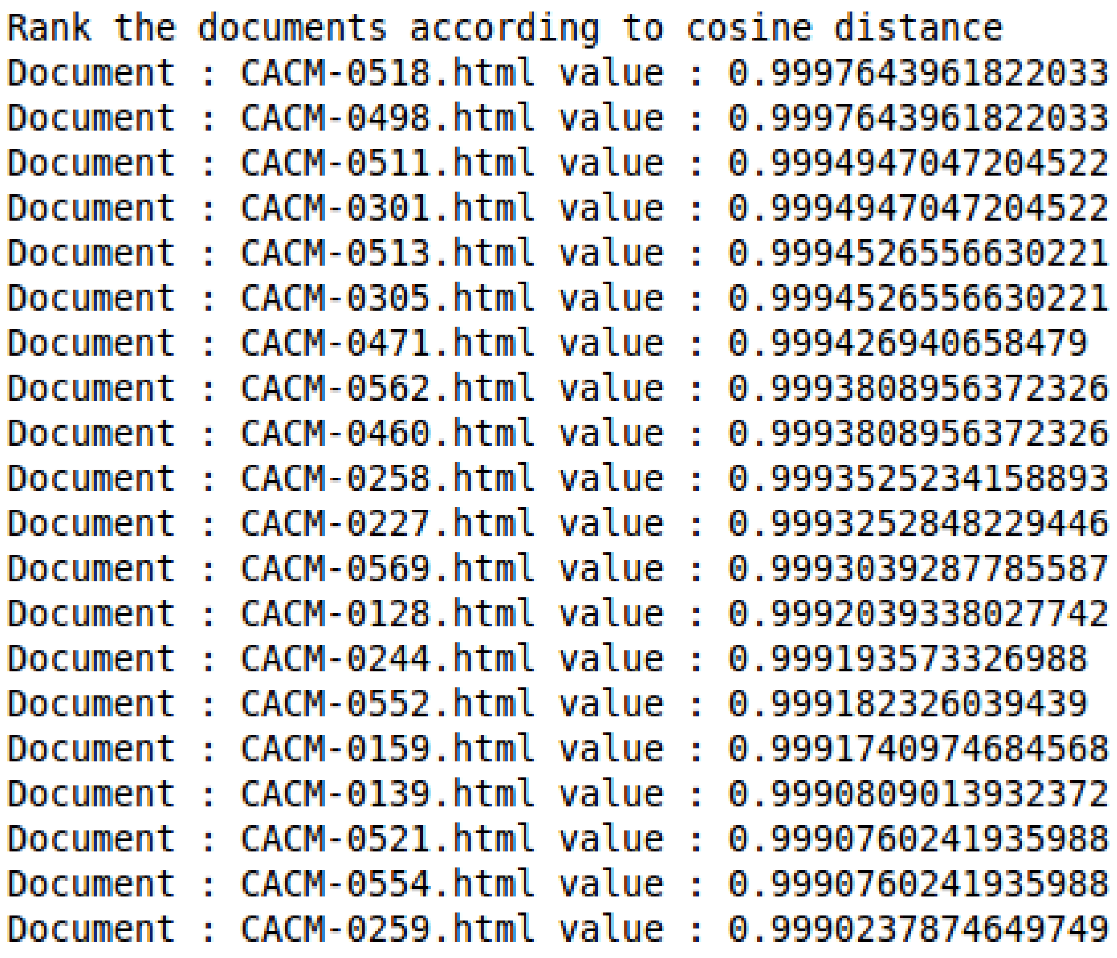
In Figure 14 we can see the ranking for the query ”CA” in our dataset.

8.2 Ranking Optimization - Cosine similarity

In the listing below, we can find the implementation of the Cosine similarity calculation. For each file we take all the terms tf-idfs’ to the power of 2 and sum them to get the magnitude of each file. In the same iteration we calculate the dot product of the term in the two files. To get the Cosine similarity we need to divide the dot product by the product of the square roots of the magnitudes.

```
private static double GetCosineSimilarity(List<Double> collection,
                                          List<Double> collection2) {
        double dot_product_total = 0;
        double length_product = collection.size() * collection2.size();
        double sum_x = 0;
        double sum_y = 0;

        for (int i = 0; i < collection.size(); i++) {
                double x = collection.get(i).doubleValue();
                double y = collection2.get(i).doubleValue();
                dot_product_total += x * y;
                sum_x += (double)Math.pow(x,2);
                sum_y += (double)Math.pow(y,2);
        }
        sum_x = Math.sqrt(sum_x);
        sum_y = Math.sqrt(sum_y);
        double cosineSimilarity = dot_product_total / (double)(sum_x * sum_y);
        return (double) cosineSimilarity;

}
```

8.2.1 Example



```
Rank the documents according to cosine distance
Document : CACM-0518.html value : 0.9997643961822033
Document : CACM-0498.html value : 0.9997643961822033
Document : CACM-0511.html value : 0.9994947047204522
Document : CACM-0301.html value : 0.9994947047204522
Document : CACM-0513.html value : 0.9994526556630221
Document : CACM-0305.html value : 0.9994526556630221
Document : CACM-0471.html value : 0.999426940658479
Document : CACM-0562.html value : 0.9993808956372326
Document : CACM-0460.html value : 0.9993808956372326
Document : CACM-0258.html value : 0.9993525234158893
Document : CACM-0227.html value : 0.9993252848229446
Document : CACM-0569.html value : 0.9993039287785587
Document : CACM-0128.html value : 0.9992039338027742
Document : CACM-0244.html value : 0.999193573326988
Document : CACM-0552.html value : 0.999182326039439
Document : CACM-0159.html value : 0.9991740974684568
Document : CACM-0139.html value : 0.9990809013932372
Document : CACM-0521.html value : 0.9990760241935988
Document : CACM-0554.html value : 0.9990760241935988
Document : CACM-0259.html value : 0.9990237874649749
```

Figure 15: Result of our Cosine based ranking

In Figure 15 we can see the ranking for the query "CA" in our dataset. We can also note a difference in the ranking of the files. When manually analyzing the difference in ranking, by looking at the content of each file, we concluded that the Cosine ranking was more precise than the Euclidean one. This is because of how the Cosine similarity works. It is only based on the mutual term frequency, whereas the Euclidean is not.

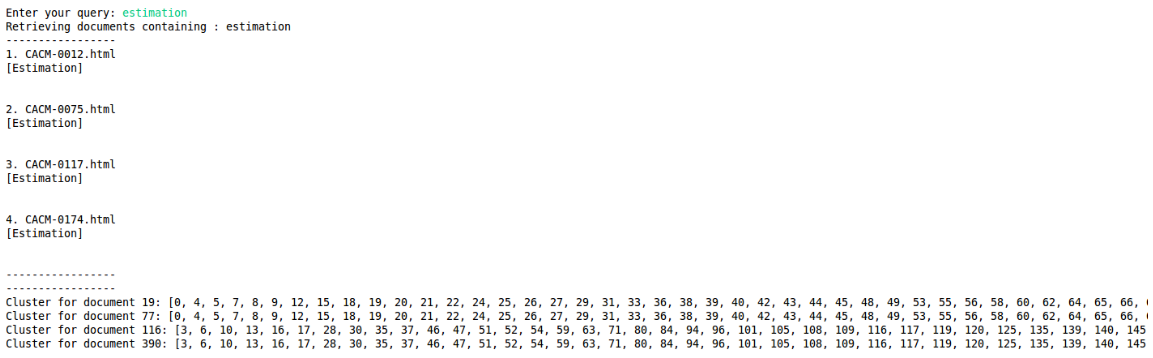
9 Clustering

For the clustering we needed to implement and use one of the clustering algorithms thought during the lectures. We decided to use the K-Means. Since we were using custom data-structures for our files, we couldn't use libraries or other implementations of K-Means that could be found online.

The K-Means is implemented in two main parts. The first part is assigning the documents to a centroid, the other one is to recalculate the new position of the centroid. A centroid is defined as the mean of the tf-idfs per term over all the files in a certain cluster. Since our data points for the clustering are files, a centroid is a new file as well.

9.1 Example

In the example code we give per file the corresponding cluster. This can be seen as a sort of suggested documents feature. Since the documents in the cluster are closely related to each other, a user wanting a certain document, may also like to see other ones.



```
Enter your query: estimation
Retrieving documents containing : estimation
-----------------
1. CACM-0012.html
[Estimation]


2. CACM-0075.html
[Estimation]


3. CACM-0117.html
[Estimation]


4. CACM-0174.html
[Estimation]


-----------------
-----------------
Cluster for document 19: [0, 4, 5, 7, 8, 9, 12, 15, 18, 19, 20, 21, 22, 24, 25, 26, 27, 29, 31, 33, 36, 38, 39, 40, 42, 43, 44, 45, 48, 49, 53, 55, 56, 58, 60, 62, 64, 65, 66,
Cluster for document 77: [0, 4, 5, 7, 8, 9, 12, 15, 18, 19, 20, 21, 22, 24, 25, 26, 27, 29, 31, 33, 36, 38, 39, 40, 42, 43, 44, 45, 48, 49, 53, 55, 56, 58, 60, 62, 64, 65, 66,
Cluster for document 116: [3, 6, 10, 13, 16, 17, 28, 30, 35, 37, 46, 47, 51, 52, 54, 59, 63, 71, 80, 84, 94, 96, 101, 105, 108, 109, 116, 117, 119, 120, 125, 135, 139, 140, 145
Cluster for document 390: [3, 6, 10, 13, 16, 17, 28, 30, 35, 37, 46, 47, 51, 52, 54, 59, 63, 71, 80, 84, 94, 96, 101, 105, 108, 109, 116, 117, 119, 120, 125, 135, 139, 140, 145
```

Figure 16: Clustering result of files.

10 Difficulties

First of we expected to have scaling issues with wild-cards, but this was not the case. However, we did have some scaling issues using a high-document frequency within Lucene. We constantly retrieved results that were not making any sense. Most of our implementation relied on the implementation of tf-idf scores from Lucene. Finally we discovered that because of this high document frequency Lucene was producing negative scores, which would be incorrect. Luckily some quick research provided us with some similar projects having the exact same issue and we instead relied on BM25 to provide us with correct results.

11 Workload

Since the start of this project Kenzo started building the foundation of the code using Lucene where we are all able to build further upon. We divided the aspects as follows: Kenzo was responsible for boolean querying with normalization and wild cards. Steven would take care of ranking & optimization using Cosine similarity and Badreddine was assigned to the clustering aspect where he decided to do K-means clustering.

Most of the implemented features had to be thoroughly researched before being implemented. This took part in the first 3 weeks after the first version of the project assignment went online. We then came together on a 2-weekly basis at the campus to discuss difficulties and help each other with challenges within each others given aspect. It was only when Kenzo had produced the first boolean queries that we are able to further make use of some overlapping code in order to implement our features.

Luckily, every group member succeeded in doing his part and all helped contributing to not only the source code, but also the presentation and report part. We did not have any problems in miscommunication and can say that the workload was equally divided.

12 Conclusion

Our group is quiet satisfied with the end result as our solution produces correct results for all aspects that should have been implemented. However, due to lack of time we did not provide any graphical user interface to make it more easy for a user to test it's queries. We felt that the essence of the project was focusing on learning the individual aspects and being capable of reproducing them in code, rather than making an easy to use interface.