MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Data Management

Lab 03 - Indexation et recherche avec Lucene

Frédéric Montet
Brian Nydegger

Rendu le 27 décembre 2016
à Lausanne

Professeurs :
Prof. Nastaran Fatemi
Prof. Maria Sokhn
Assistant Shaban Shabani

Table des matières

Introduction

Pour ce troisième laboratoire de cours de Data Management, nous allons découvrir la plateform Lucene et apprendre ces fonctionnalités en utilisant l'API Java. Lucene est une bibliothèque pour l'indexation et la recherche dans un fichier texte. Il est maintenu par Apache.

Dans la première partie de ce travail, nous allons créer un indexe avec la collection CACM et y exécuter des requêtes. Une fois ceci fait, nous testerons plusieurs `Analyzers` qui prétraitent la collection de différentes manières.

Finalement, nous modifierons le calcul du score fait par Lucene et observerons les changements que cela implique sur les résultats.

1 | Installation de Lucene

Pour installer Lucene, nous avons téléchargé son archive en version 6.3.0. Puis, nous avons suivi la page de demo [1]. Cette dernière nous a mené à un environnement fonctionnel de Lucene ou il est possible d'indexer une collection de document et d'effectuer des recherches dessus.

Le listing Figure 1.2 montre les variables d'environnement ajoutées au `CLASSPATH`.

```
# Lucene
export LUCENE_HOME="/Users/fredmontet/Dropbox/project/master/dmg/project/labs/lab03-lucene/app/lucene-6.3.0"
export LUCENE_CORE=${LUCENE_HOME}/core/lucene-core-6.3.0.jar
export LUCENE_ANALYZERS=${LUCENE_HOME}/analysis/common/lucene-analyzers-common-6.3.0.jar
export LUCENE_QUERYPARSER=${LUCENE_HOME}/queryparser/lucene-queryparser-6.3.0.jar
export LUCENE_DEMO=${LUCENE_HOME}/demo/lucene-demo-6.3.0.jar
export LUCENE=$LUCENE_CORE:$LUCENE_ANALYZERS:$LUCENE_QUERYPARSER:$LUCENE_DEMO

# Classpath
export CLASSPATH=$LUCENE:$CLASSPATH
```

FIGURE 1.1 – Ajout au CLASSPATH

Une fois la variable `CLASSPATH` mise à jour, il est possible d'utiliser Lucene et d'indexer

```

# Command to index
java org.apache.lucene.demo.IndexFiles -docs $LUCENE_HOME/docs

# Command to search
java org.apache.lucene.demo.SearchFiles
```

FIGURE 1.2 – Ajout au CLASSPATH

1. http://lucene.apache.org/core/6_3_0/demo/overview-summary.html

2 | Comprendre l'API Lucene

2.1 Question 1

Does the command line demo use stopword removal ? Explain how you find out the answer.

Oui, les stop words sont supprimés avec la ligne de commande de la démo.

Dans Lucene, les `Analyzer` ont la tâche d'enlever les stop words. Étant donné qu'il y en a plusieurs types, il faut dans un premier temps savoir lequel est utilisé.

En regardant la page de demo il est écrit : `The main()` method parses the command-line parameters, then in preparation for instantiating IndexWriter, opens a Directory, and instantiates StandardAnalyzer and IndexWriterConfig. Le `StandardAnalyzer` est donc utilisé. Dans la documentation Lucene 6.3.0, on trouve que la classe textitStandardAnalyzer [1] utilise le `StopFilter` avec une liste de stop words en anglais.

2.2 Question 2

Does the command line demo use stemming ? Explain how you find out the answer.

Non, le stemming n'est pas effectué avec la ligne de commande la demo.

Les `Analyzer` ont la tâche d'effectuer le stemming. Cependant, suivant lequel est utilisé, cela n'est pas le cas de tous. Comme cité dans la section 2.1 le `StandardAnalyzer` est utilisé. Selon la documentation le stemming n'est mentionné à aucun moment, on en déduit donc qu'il n'est pas effectué.

Notre deduction est comfortée en regardant le code source du `StandardAnalyzer` [2] et en le comparant avec celui de l'`EnglishAnalyzer` [3] à la ligne 107 qui lui, fait le stemming avec l'algorithme de Porter.

1. http://lucene.apache.org/core/6_3_0/core/org/apache/lucene/analysis/standard/StandardAnalyzer.html?is-external=true
2. https://github.com/apache/lucene-solr/blob/master/lucene/core/src/java/org/apache/lucene/analysis/standard/StandardAnalyzer.java
3. https://github.com/apache/lucene-solr/blob/master/lucene/analysis/common/src/java/org/apache/lucene/analysis/en/EnglishAnalyzer.java

2.3 Question 3

Is the search of the command line demo case insensitive ? How did you find out the answer ?

Non, la demo n'est pas sensible à la casse.

On le constate très simplement en comparant le nombre de résultats de deux requêtes similaires. Nous avons efféctué plusieurs recherches, entre autre "ANALYZER" et "analyzer". Dans chacun des cas, le nombre de résultats obtenu à été le même pour les requêtes ont donnés le même nombre de résultat, respéctivement 522 pour la requuquête citée ci-dessus.

2.4 Question 4

Does it matter whether stemming occurs before or after stopword removal ? Consider this as a general question.

Oui, cet ordre joue un rôle dans le résultat.

Si le stemming est fait avant la suppression des stopwords, il est possible que certain mot de la liste des stopwords soient égaux à certains mots "stemmés" et donc, seront supprimés alors qu'à la base, ils ne sont pas des stopwords. Cet ordre n'est donc pas approprié.

Dans le cas ou la suppression des stopwords est faite avant le stemming, on enlève d'abord les mots considérés comme inutiles et ensuite on "stem" ce sous-ensemble pour en trouver les racines en fonction de la langue désirée. C'est avec cet ordre que le résultat sera le plus juste.

3 | Utiliser Luke

Luke est un logiciel permettant de visualiser un index créé avec Lucene. Après avoir téléchargé la version 6.3.0, nous avons exécuté le fichier `luke.sh` pour visualiser l'index créé avec la demo de Lucene visible sur la Figure 3.1

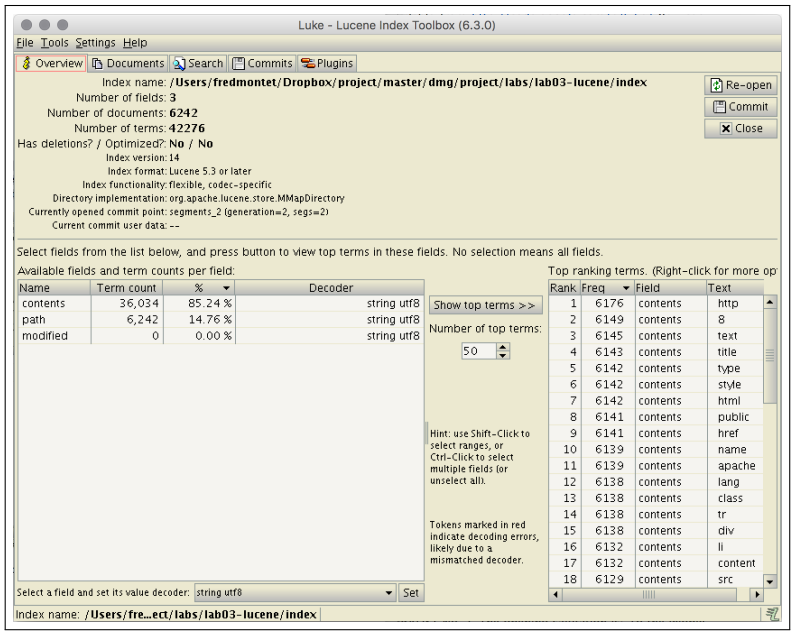

FIGURE 3.1 – Interface de Luke avec l'index de la documentation de Lucene

4 | Indexation et recherche dans la collection CACM avec Lucene

4.1 Indexation

Voir code sources attachés en annexe.

4.2 Utilisation de différents `Analyzers`

Lucene implémente plusieurs `Analyzers` pour traiter un document. Nous allons en essayer quelques-uns pour l'indexation et la recherche.

Voici les cinq questions auxquelles il faut répondre pour chaque `Analyzer` :

1. The number of indexed documents and indexed terms.
2. The number of indexed terms in the summary field.
3. The top 10 frequent terms of the summary field in the index.
4. The size of the index on disk.
5. The required time for indexing.

4.2.1 WhitespaceAnalyzer

1. 3'203 / 38'029
2. 26'821
3. of / the / is / a / and / to / in / for / The / are
4. 2'937 kB
5. 2'347 ms

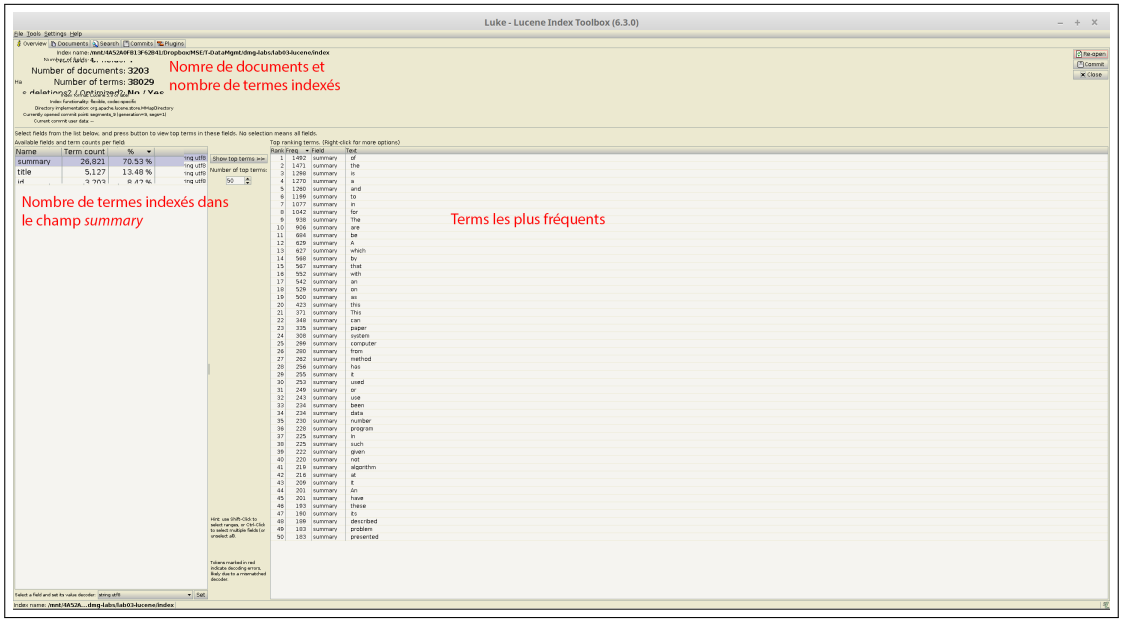

FIGURE 4.1 – Luke : WhitespaceAnalyzer exemple

4.2.2 EnglishAnalyzer

1. 3'203 / 26'212
2. 16'724
3. us / which / comput / program / system / present / describ / paper / method / can
4. 2'388 kB
5. 2'463 ms

4.2.3 ShingleAnalyzerWrapper (using shingle size 2)

1. 3'203 / 106'272
2. 85'610
3. which / system / paper / computer / can / " paper" / described / given / presented / time
4. 5'449 kB
5. 3'293 ms

4.2.4 ShingleAnalyzerWrapper (using shingle size 3)

1. 3'203 / 148'948

2. 125'776
3. which / system / computer / paper / can / described / time / given / presented / from
4. 7'053 kB
5. 3'580 ms

4.2.5 StopAnalyzer

1. 3'203 / 24'662
2. 18.342
3. system / computer / paper / presented / time / method / program / data / algorithm / discussed
4. 2'214 kB
5. 3'183 ms

4.3 Lecture de l'index

Plutôt que d'utiliser Luke pour lire l'index, nous allons maintenant utiliser l'API Lucene.

En utilisant la classe *HighFreqTerms* nous devons répondre aux deux questions suivantes :

1. What is the author with the highest number of publications ? How many publications does he/she have ?

Résultats :

Terms	Nb
Thacher Jr., H. C.	38
Naur, P.	19
Hill, I. D.	16
Wirth, N.	15
Pike, M. C.	14
Herndon, J. R.	14
Gautschi, W.	14
Boothroyd, J.	14
George, R.	12
Floyd, R. W.	12
Bemer, R. W.	12
McKeeman, W. M.	11

Code :

```
1  public void authorWithMaxPublication() throws Exception {
2
3  Directory d = FSDirectory.open(FileSystems.getDefault().getPath("index"));
4  IndexReader ir = DirectoryReader.open(d);
5
6  HighFreqTerms.DocFreqComparator cmp = new HighFreqTerms.DocFreqComparator();
7  TermStats[] fields = HighFreqTerms.getHighFreqTerms(ir, 12, "author", cmp);
8  for (TermStats ts: fields)
9      System.out.println(ts.termtext.utf8ToString() + " => " + ts.docFreq);
10
11 }
```

2. List the top 10 terms in the title field with their frequency.

Résultats :

Terms	Nb
algorithm	963
computer	260
system	172
programming	154
method	125
data	110
systems	108
language	99
program	93
matrix	82

Code :

```
public void titleTermeTop() throws Exception {

Directory d = FSDirectory.open(FileSystems.getDefault().getPath("index"));
IndexReader ir = DirectoryReader.open(d);

HighFreqTerms.DocFreqComparator cmp = new HighFreqTerms.DocFreqComparator();
TermStats[] fields = HighFreqTerms.getHighFreqTerms(ir, 10, "title", cmp);
for (TermStats ts: fields)
    System.out.println(ts.termtext.utf8ToString() + " => " + ts.docFreq);

}
```

4.4 Recherche

Pour cette partie nous avons reconstruit notre index avec *EnglishAnalyzer*.

Voici le prototype de notre fonction de requête :

```
public void query(String fieldStr, String queryStr, String analyzer_type,
    ↪ int max_outputs) throws ParseException, IOException {...};
```

Voici les requêtes effectuées avec *QueryParser* sur le champ *summary* :

1. *Publications containing the term "Information Retrieval".*

   ```
   cacm.query("summary", "Information Retrieval", analyzer_type, 10);
   ```

   Voici les 10 premiers résultats de cette requête sur les **188** :
   - Query is : summary :inform summary :retriev
   - 1457 : Data Manipulation and Programming Problemsin Automatic Information Retrieval (8.651913)
   - 891 : Everyman's Information Retrieval System (8.181953)
   - 1699 : Experimental Evaluation of InformationRetrieval Through a Teletypewriter (7.5747085)
   - 2307 : Dynamic Document Processing (7.3587627)
   - 3134 : The Use of Normal Multiplication Tablesfor Information Storage and Retrieval (7.3557515)
   - 1032 : Theoretical Considerations in Information Retrieval Systems (7.312654)
   - 1935 : Randomized Binary Search Technique (7.1063213)
   - 1681 : Easy English,a Language for InformationRetrieval Through a Remote Typewriter Console (6.70207)
   - 2990 : Effective Information Retrieval Using Term Accuracy (6.70207)

— 2519 : On the Problem of Communicating Complex Information (6.2497764)

2. *Publications containing both "Information" and "Retrieval".*

```
cacm.query("summary", "Information AND Retrieval", analyzer_type, 10);
```

Voici les 10 premiers résultats de cette requête sur les **23** :

— Query is : +summary :inform +summary :retriev
— 1457 : Data Manipulation and Programming Problemsin Automatic Information Retrieval (8.651913)
— 891 : Everyman's Information Retrieval System (8.181953)
— 1699 : Experimental Evaluation of InformationRetrieval Through a Teletypewriter (7.5747085)
— 2307 : Dynamic Document Processing (7.3587627)
— 3134 : The Use of Normal Multiplication Tablesfor Information Storage and Retrieval (7.3557515)
— 1032 : Theoretical Considerations in Information Retrieval Systems (7.312654)
— 1935 : Randomized Binary Search Technique (7.1063213)
— 1681 : Easy English,a Language for InformationRetrieval Through a Remote Typewriter Console (6.70207)
— 2990 : Effective Information Retrieval Using Term Accuracy (6.70207)
— 2519 : On the Problem of Communicating Complex Information (6.2497764)

3. *Publications containing at least the term "Retrieval" and, possibly "Information" but not "Database".*

```
cacm.query("summary", "+Retrieval Information -Database",
    ↪ analyzer_type, 10);
```

Voici les 10 premiers résultats de cette requête sur les **54** :

— Query is : +summary :retriev summary :inform -summary :databas
— 1457 : Data Manipulation and Programming Problemsin Automatic Information Retrieval (8.651913)
— 891 : Everyman's Information Retrieval System (8.181953)
— 1699 : Experimental Evaluation of InformationRetrieval Through a Teletypewriter (7.5747085)
— 2307 : Dynamic Document Processing (7.3587627)
— 3134 : The Use of Normal Multiplication Tablesfor Information Storage and Retrieval (7.3557515)
— 1032 : Theoretical Considerations in Information Retrieval Systems (7.312654)
— 1935 : Randomized Binary Search Technique (7.1063213)
— 1681 : Easy English,a Language for InformationRetrieval Through a Remote Typewriter Console (6.70207)

— 2990 : Effective Information Retrieval Using Term Accuracy (6.70207)
— 2519 : On the Problem of Communicating Complex Information (6.2497764)

4. *Publications containing a term starting with "Info".*

```
cacm.query("summary", "Info*", analyzer_type, 10);
```

Voici les 10 premiers résultats de cette requête sur les **193** :

— Query is : summary :info*
— 222 : Coding Isomorphisms (1.0)
— 272 : A Storage Allocation Scheme for ALGOL 60 (1.0)
— 396 : Automation of Program Debugging (1.0)
— 397 : A Card Format for Reference Files in Information Processing (1.0)
— 409 : CL-1, An Environment for a Compiler (1.0)
— 440 : Record Linkage (1.0)
— 483 : On the Nonexistence of a Phrase Structure Grammar for ALGOL 60 (1.0)
— 616 : An Information Algebra - Phase I Report-LanguageStructure Group of the CODASYL Development Committee (1.0)
— 644 : A String Language for Symbol Manipulation Based on ALGOL 60 (1.0)
— 655 : COMIT as an IR Language (1.0)

5. *Publications containing the term "Information" close to "Retrieval" (max distance 5).*

```
cacm.query("summary", "'Information Retrieval'~5", analyzer_type, 10);
```

Voici les 10 premiers résultats de cette requête sur les **159** :

— Query is : summary :inform summary :retrieval' 2
— 2160 : Canonical Structure in Attribute Based File Organization (11.125051)
— 2832 : Faster Retrieval from Context Trees (Corrigendum) (5.5161905)
— 1516 : Automatic Data Compression (4.020056)
— 1746 : Protection in an Information Processing Utility (4.0018125)
— 2870 : A Lattice Model of Secure Information Flow (3.8555226)
— 1267 : Performance of Systems Used for Data TransmissionTransfer Rate of Information Bits -An ASA TutorialStandard (3.8451033)
— 3134 : The Use of Normal Multiplication Tablesfor Information Storage and Retrieval (3.7823412)
— 2905 : Perfect Hashing Functions : A SingleProbe Retrieving Method for Static Sets (3.6556962)
— 1457 : Data Manipulation and Programming Problemsin Automatic Information Retrieval (3.5519412)

— 1745 : A Position Paper on Computing and Communications (3.5519412)

4.5 Améliorer le Score de Lucene

Nous exécutons la première requête du point précédent (*Information Retrieval* avec *EnglishAnalysez*) avec la fonction de similarité par défaut et ensuite avec une fonction customisée.

Le code de la fonction similarité customisée :

```
package cacm;

import org.apache.lucene.index.FieldInvertState;
import org.apache.lucene.search.similarities.ClassicSimilarity;

public class CustomSimilarity extends ClassicSimilarity {

    @Override
    public float tf(float freq) {
        return (float) (1 + Math.log((double) freq));
    }

    @Override
    public float idf(long docFreq, long numDocs) {
        return (float) Math.log(numDocs / docFreq);
    }

    @Override
    public float coord(int overlap, int maxOverlap) {
        return (float) Math.sqrt(overlap / maxOverlap);
    }

    @Override
    public float lengthNorm(FieldInvertState state) {
        return 1f;
    }
}
```

Le code suivant ajoute cette classe au moment de l'indexation :

```
IndexWriterConfig iwc = new IndexWriterConfig(index_analyzer);
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
iwc.setUseCompoundFile(false);
iwc.setSimilarity(new CustomSimilarity());
```

Et ce code au moment de la recherche :

```
IndexSearcher indexSearcher = new IndexSearcher(indexReader);
indexSearcher.setSimilarity(new CustomSimilarity());
```

Voici les 10 premiers résultats avec la fonction de similarité par défaut avec le score entre paranthèses :

- 1457 : Data Manipulation and Programming Problemsin Automatic Information Retrieval (8.651913)
- 891 : Everyman's Information Retrieval System (8.181953)
- 1699 : Experimental Evaluation of InformationRetrieval Through a Teletypewriter (7.5747085)
- 2307 : Dynamic Document Processing (7.3587627)
- 3134 : The Use of Normal Multiplication Tablesfor Information Storage and Retrieval (7.3557515)
- 1032 : Theoretical Considerations in Information Retrieval Systems (7.312654)
- 1935 : Randomized Binary Search Technique (7.1063213)
- 1681 : Easy English,a Language for InformationRetrieval Through a Remote Typewriter Console (6.70207)
- 2990 : Effective Information Retrieval Using Term Accuracy (6.70207)
- 2519 : On the Problem of Communicating Complex Information (6.2497764)

Et voici maintenant les 10 premiers résultats avec la fonction de similarité customisée :

- 1032 : Theoretical Considerations in Information Retrieval Systems (8.500153)
- 1457 : Data Manipulation and Programming Problemsin Automatic Information Retrieval (8.500153)
- 3134 : The Use of Normal Multiplication Tablesfor Information Storage and Retrieval (8.295944)
- 891 : Everyman's Information Retrieval System (7.9694023)
- 1699 : Experimental Evaluation of InformationRetrieval Through a Teletypewriter (7.9694023)
- 2307 : Dynamic Document Processing (7.3886204)
- 1681 : Easy English,a Language for InformationRetrieval Through a Remote Typewriter Console (7.0620785)
- 2990 : Effective Information Retrieval Using Term Accuracy (7.0620785)
- 1527 : A Grammar Base Question Answering Procedure (6.8578696)
- 1652 : A Code for Non-numeric Information ProcessingApplications in Online Systems (5.865016)

Et l'on observe très bien des différents scores sur les documents. Ceux avec la fonction customisée sont plus élevé.

Nous avons vu dans la donnée de ce laboratoire que le score était calculé par défaut selon cette formule :

$$score(q,d) = \sum [tf(t_d) \times idf(t) \times boost(t.field_d) \times norm(t,d)] \times coord(q,d) \times qNorm(q) \tag{4.1}$$

Dans la classe `CustomSimilarity`, quatre méthodes ont été réécrites et ont prit la place des méthodes par défaut selon le tableau ci-dessous.

	`ClassicSimilarity`	`CustomSimilarity`
`tf()`	$sqrt(freq)$	$1 + log(freq)$
`idf()`	$log((numDocs + 1)/(docFreq + 1)) + 1$	$log(numDocs/docFreq)$
`coord()`	$overlap/maxOverlap$	$sqrt(overlap/maxOverlap)$
`lengthNorm()`	$state.getBoost() * 1/sqrt(numTerms)))$	1

Dans la fonction de calcul du score ci-dessus, ces modifications ont les influences suivantes :

— $tf(t_d)$ sera très amortie dans `CustomSimilarity`

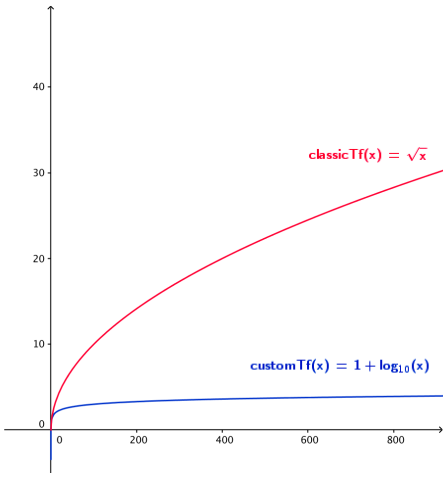

FIGURE 4.2 – Plot des fréquences de termes

— $idf(t)$ sera très amortie dans `CustomSimilarity`, ceci est majoritairement dû à son ordonnée qui n'a plus d'offset. [1]

1. 3203 est le nombre de documents dans `cacm.txt`

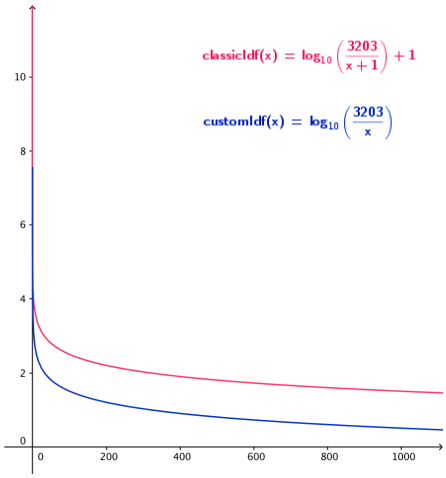

FIGURE 4.3 – Plot des fréquences documentaires inversée

— $coord(q, d)$ est plus faible dans `CustomSimilarity` car $overlap/maxOverlap$ est mis à la racine.
— le terme $norm(t, d)$ n'a plus d'influence sur le score car il est remplacé par 1.

L'obtention d'un score final plus élevé avec le scorrer `CustomSimilarity` est expliquée par le dernier point. Par défaut, les valeurs que prendra $norm(t, d)$ vont dépendre de $boost * 1/sqrt(numTerms)))$. On constate qu'à moins que $boost$ soit plus grand que $sqrt(numTerms)$, $norm(t, d)$ sera compris entre 0 et 1. Dans notre cas, nous n'avons pas appliqué de boost lors de l'indexation et le nombre de termes est égal à 3203 dans notre corpus. Le résultat prendra donc une valeur de $1 * 1/sqrt(3203) \approx 0.018$.

5 | Conclusion

En faisant ce laboratoire, nous avons découvert la puissance de l'API Lucene. Il nous a permis de concrétiser les multiples concepts vu dans le cours de Data Management.

Le logiciel Luke nous a permis d'assimiler le fonctionnement de l'indexage et de la recherche pour ensuite comprendre les différents filtres et les différentes requêtes possibles.

A | Code source

Listing A.1 – ../app/src/cacm/CacmItem.java

```
1  package cacm;
2
3  public class CacmItem {
4
5      public String id;
6      public String[] authors;
7      public String title;
8      public String summary;
9
10     public CacmItem(String line) {
11
12         String[] splitted = line.split("\t");
13         this.id = splitted[0];
14
15         String authorList = splitted[1];
16         this.authors = authorList.split(";");
17         if(this.authors[0].equals(""))
18             this.authors = null;
19
20         this.title = splitted[2];
21         if (splitted.length == 4) {
22             this.summary = splitted[3];
23         } else {
24             this.summary = null;
25         }
26     }
27 }
```

Listing A.2 – ../app/src/cacm/ReadIndex.java

```
1  package cacm;
2
3  import org.apache.lucene.index.DirectoryReader;
4  import org.apache.lucene.index.IndexReader;
5  import org.apache.lucene.misc.HighFreqTerms;
6  import org.apache.lucene.misc.TermStats;
7  import org.apache.lucene.store.Directory;
8  import org.apache.lucene.store.FSDirectory;
9
10 import java.nio.file.FileSystems;
11
12 /**
13  * Created by brian on 12/20/16.
14  */
15 public class ReadIndex {
16     public void authorWithMaxPublication() throws Exception {
```

```
        Directory d = FSDirectory.open(FileSystems.getDefault().getPath("
            ↪ index"));
        IndexReader ir = DirectoryReader.open(d);

        HighFreqTerms.DocFreqComparator cmp = new HighFreqTerms.
            ↪ DocFreqComparator();
        TermStats[] fields = HighFreqTerms.getHighFreqTerms(ir, 12, "author"
            ↪ , cmp);
        for (TermStats ts: fields)
            System.out.println(ts.termtext.utf8ToString() + " => " + ts.
                ↪ docFreq);
    }

    public void titleTermeTop() throws Exception {
        Directory d = FSDirectory.open(FileSystems.getDefault().getPath("
            ↪ index"));
        IndexReader ir = DirectoryReader.open(d);

        HighFreqTerms.DocFreqComparator cmp = new HighFreqTerms.
            ↪ DocFreqComparator();
        TermStats[] fields = HighFreqTerms.getHighFreqTerms(ir, 10, "title",
            ↪  cmp);
        for (TermStats ts: fields)
            System.out.println(ts.termtext.utf8ToString() + " => " + ts.
                ↪ docFreq);

    }
}
```

Listing A.3 – ../app/src/cacm/SearchEngine.java

```
package cacm;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.core.StopAnalyzer;
import org.apache.lucene.analysis.core.WhitespaceAnalyzer;
import org.apache.lucene.analysis.en.EnglishAnalyzer;
import org.apache.lucene.analysis.shingle.ShingleAnalyzerWrapper;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.FieldType;
import org.apache.lucene.document.StringField;
import org.apache.lucene.index.*;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.similarities.ClassicSimilarity;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.queryparser.classic.QueryParser;
```

```java
import org.apache.lucene.queryparser.classic.ParseException;


import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.FileSystems;
import java.nio.file.Path;

public class SearchEngine {

    private String index_path;
    private String file_path;
    private Analyzer index_analyzer;
    private Analyzer query_analyzer;

    public SearchEngine(String file_path, String index_path) throws
        ↪ IOException {
        setFilePath(file_path);
        setIndexPath(index_path);
    }

    public void index(String analyzer_type) throws IOException {
        index(analyzer_type, null);
    }

    public void index(String analyzer_type, ClassicSimilarity similarity)
        ↪ throws IOException {
        System.out.println("Start indexing CACM collection");
        this.index_analyzer = getAnalyzerByName(analyzer_type);

        // Configure indexer
        IndexWriterConfig iwc = new IndexWriterConfig(index_analyzer);
        iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
        iwc.setUseCompoundFile(false);
        if (similarity != null)
            iwc.setSimilarity(similarity);

        // Create index writer
        Path path = FileSystems.getDefault().getPath("index");
        Directory dir = FSDirectory.open(path);
        IndexWriter indexWriter = new IndexWriter(dir, iwc);

        // Index files
        try (BufferedReader br = new BufferedReader(new FileReader(this.
            ↪ file_path))) {
            String line;
            while ((line = br.readLine()) != null) {

                // Create document
                Document doc = new Document();
```

```
69                  CacmItem item = new CacmItem(line);
70
71                  // Create field
72                  FieldType fieldType = new FieldType();
73                  fieldType.setIndexOptions(IndexOptions.
                        ↪ DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS);
74                  fieldType.setTokenized(true);
75                  fieldType.setStored(true);
76                  fieldType.setStoreTermVectors(true);
77                  fieldType.setStoreTermVectorOffsets(true);
78                  fieldType.setStoreTermVectorPayloads(true);
79                  fieldType.setStoreTermVectorPositions(true);
80                  fieldType.freeze();
81
82                  // Add fields to document
83                  if (item.id != null)
84                      doc.add(new Field("id", item.id, fieldType));
85
86                  if (item.authors != null)
87                      for (String authorName : item.authors)
88                          doc.add(new StringField("author", authorName, Field.
                                ↪ Store.YES));
89
90                  if (item.title != null)
91                      doc.add(new Field("title", item.title, fieldType));
92
93                  if (item.summary != null)
94                      doc.add(new Field("summary", item.summary, fieldType));
95
96                  // Add document to index
97                  indexWriter.addDocument(doc);
98              }
99          }
100
101         // Terminate indexing
102         indexWriter.close();
103         dir.close();
104         System.out.println("Indexing done");
105     }
106
107     public void query(String fieldStr, String queryStr, String analyzer_type
            ↪ , int max_outputs) throws ParseException, IOException {
108         query(fieldStr, queryStr, analyzer_type, max_outputs, null);
109     }
110
111     public void query(String fieldStr, String queryStr, String analyzer_type
            ↪ , int max_outputs, ClassicSimilarity similarity) throws
            ↪ ParseException, IOException {
112         System.out.println("Start search");
113         this.query_analyzer = getAnalyzerByName(analyzer_type);
114
```

```
115         // Create query parser
116         QueryParser parser = new QueryParser(fieldStr, this.query_analyzer);
117 
118         // Parse query
119         Query query = parser.parse(queryStr);
120 
121         // Create index reader
122         Path path = FileSystems.getDefault().getPath(this.index_path);
123         Directory dir = FSDirectory.open(path);
124         IndexReader indexReader = DirectoryReader.open(dir);
125 
126         // Create index searcher
127         IndexSearcher indexSearcher = new IndexSearcher(indexReader);
128         if (similarity != null)
129             indexSearcher.setSimilarity(similarity);
130 
131         // Search query
132         ScoreDoc[] hits = indexSearcher.search(query, 1000).scoreDocs;
133 
134         // Retrieve results
135         System.out.println("Results found: " + hits.length);
136         System.out.println("Query is: " + query.toString());
137         int i = 0;
138         for (ScoreDoc hit : hits) {
139             if (i == max_outputs)
140                 break;
141             Document doc = indexSearcher.doc(hit.doc);
142             System.out.println(doc.get("id") + ": " + doc.get("title") + " (
                ↪ " + hit.score + ")");
143             i++;
144         }
145 
146         // Close index reader
147         indexReader.close();
148         dir.close();
149 
150         System.out.println("Search is done !");
151     }
152 
153     private Analyzer getAnalyzerByName(String analyzer_type) {
154 
155         System.out.println("Analyzer type : " + analyzer_type);
156         Analyzer analyzer;
157 
158         switch (analyzer_type) {
159             case "standard":
160                 return analyzer = new StandardAnalyzer();
161             case "whitespace":
162                 return analyzer = new WhitespaceAnalyzer();
163             case "english":
164                 return analyzer = new EnglishAnalyzer();
```

```
            case "shingle_2":
                return analyzer = new ShingleAnalyzerWrapper(2, 2);
            case "shingle_3":
                return analyzer = new ShingleAnalyzerWrapper(3, 3);
            case "stop":
                try {
                    return analyzer = new StopAnalyzer(new FileReader("asset
                        ↪ /common_words.txt"));
                } catch (IOException e) {
                    e.printStackTrace();
                }
            default:
                System.out.println(analyzer_type + " analyzer is not
                    ↪ existing, StandardAnalyzer is used instead");
                return analyzer = new StandardAnalyzer();
        }
    }

    public String getFilePath() {
        return this.file_path;
    }

    public void setFilePath(String file_path) {
        this.file_path = file_path;
    }

    public String getIndexPath() {
        return index_path;
    }

    public void setIndexPath(String index_path) {
        this.index_path = index_path;
    }
}
```

Listing A.4 – ../app/src/cacm/CustomSimilarity.java

```
package cacm;

import org.apache.lucene.index.FieldInvertState;
import org.apache.lucene.search.similarities.ClassicSimilarity;

public class CustomSimilarity extends ClassicSimilarity {

    @Override
    public float tf(float freq) {
        return (float) (1 + Math.log((double) freq));
    }

    @Override
    public float idf(long docFreq, long numDocs) {
```

```
        return (float) Math.log(numDocs / docFreq);
    }

    @Override
    public float coord(int overlap, int maxOverlap) {
        return (float) Math.sqrt(overlap / maxOverlap);
    }

    @Override
    public float lengthNorm(FieldInvertState state) {
        return 1f;
    }
}
```

Listing A.5 – ../app/src/Main.java

```
import cacm.CustomSimilarity;
import cacm.SearchEngine;

public class Main {

    public static void main(String[] args) throws Exception {

        String index_path = "index";
        String file_path = "asset/cacm.txt";
        String analyzer_type = "english";

        SearchEngine cacm = new SearchEngine(file_path, index_path);

        long start_time = System.nanoTime();

        CustomSimilarity customSimilarity = new CustomSimilarity();

        /*/
        cacm.index(analyzer_type);
        /*/
        cacm.index(analyzer_type, customSimilarity);
        /**/

        long end_time = System.nanoTime();
        System.out.println("Time of indexing (ms) : " + (end_time -
            ↪ start_time) / 1e6);

        cacm.query("summary", "Information Retrieval", analyzer_type, 10);
        cacm.query("summary", "Information Retrieval", analyzer_type, 10,
            ↪ customSimilarity);

        /*
        cacm.query("summary", "Information Retrieval", analyzer_type, 10);
        cacm.query("summary", "Information AND Retrieval", analyzer_type,
            ↪ 10);
```

```
33          cacm.query("summary", "+Retrieval Information -Database",
                ↪ analyzer_type, 10);
34          cacm.query("summary", "Info*", analyzer_type, 10);
35          cacm.query("summary", "'Information Retrieval'~5", analyzer_type,
                ↪ 10);
36
37          ReadIndex readIndex = new ReadIndex();
38          readIndex.authorWithMaxPublication();
39          readIndex.titleTermeTop();
40          */
41      }
42  }
```