Forestier Quentin & Herzig Melvyn

MAC – 30.11.2021

Indexing and Search with Apache LuceneProtocoles applicatifs

Labo 3



Table des matières

[Familiarisation avec Lucene 2](#_Toc88848856)

[Indexing and Searching the CACM collection 4](#_Toc88848857)

[Indexing 4](#_Toc88848858)

[Utiliser différents analyseurs 6](#_Toc88848859)

[Lire l’index 8](#_Toc88848860)

[Recherches 10](#_Toc88848861)

[Diverses exécutions 11](#_Toc88848862)

[Personnaliser le score de Lucene 14](#_Toc88848863)

# Familiarisation avec Lucene

**Quels sont les types des *fields* dans l’index :**

* StringField
* LongPoint
* TextField

**Quels sont les caractéristiques des types précédents :**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Indexing | Storage | tokenization |
| StringField | Oui | Oui | Non |
| LongPoint | Sous forme de long | Non | Non |
| TextField | Oui | Non | Oui |

**Est-ce que la démo de la ligne de commande retire les stopwords :**

Non, le queryParser est fait avec un StandardAnalyzer et son constructeur vide qui n’utilise aucun stop words.



Pour les appliquer il aurait fallu les lui passer à la construction.

**Est-ce que la démo de la ligne de commande utilise un filtre de stemming :**

Non, le queryParser est fait avec un StandardAnalyzer qui n’utilise pas le filtre PorterStemFilter.

Comme le prouve notre test, apples n’est pas transformé en apple. De plus, ils ne retournent pas les même documents.

Une image contenant texte

Description générée automatiquement

**Est-ce que la démo de la ligne de commande est non sensible à la casse ?**

Oui car le StandardAnalyzer utilise un LowerCaseFilter.

Une image contenant texte

Description générée automatiquement

Comme nous pouvons le voir, les deux requêtes retournent les mêmes documents.

**Est-ce important de faire le stemming avant ou après le retrait des stopwords ?**

D’une manière générale, nous estimons qu’il est plus sage de retirer les stopwords avant de stemmer. Dans le cas inverse, nous n’excluons pas qu’un mot stemmé deviennent un stopword et soit retiré de la requête.

# Indexing and Searching the CACM collection

## Indexing

**Qu’est qu’un vecteur de termes ?**

Selon [northcoder](https://northcoder.com/post/lucene-fields-and-term-vectors/#:~:text=Term%20vectors%20are%20an%20alternative,for%20it%20to%20be%20created.&text=Note%20that%20you%20need%20to,term%20vector%20positions%20and%20offsets.), un vecteur de terme est une manière alternative de structurer des données indexables. Par défaut, un vecteur de terme n’est pas stocké dans l’index.

Les données sont normalement indexées de la manière suivante :  
field > term > doc > freq/pos

À l’inverse, le vecteur :  
doc > field > term > freq/pos

Nous constatons que l’indexation n’est plus faite selon les champs mais selon les documents

**Que faut-il ajouter dans le code pour accéder au vecteur de termes ?**

La méthode la plus basique est :

summaryField.setStoreTermVectors(true);

Mais il existe également :

summaryField.setStoreTermVectorOffsets(true);

summaryField.setStoreTermVectorPayloads(true);

summaryField.setStoreTermVectorPositions(true);

Après avoir ajouté la première méthode, si un document présente le champ *summary* nous pouvons accéder à son vecteur de terme avec un clic droit.



Ce qui ouvre le vecteur de terme du document en question.

Une image contenant table

Description générée automatiquement

**Comparer la taille de l’index avant et après avoir activé les vecteurs de termes. Discuter du résultat.**

|  |  |
| --- | --- |
| Sans vecteur de termes | Avec vecteur de terme |
|  |  |

L’index avec vecteur de terme basique est environ 1,5 fois plus lourd que l’index sans vecteur de terme. Pourquoi ? L’index sans vecteur de termes stocke uniquement des informations générales pour chercher un document. Cependant, l’index avec vecteur de terme stocke, en plus, des informations de recherche spécifique à chaque document. Nous comprenons alors que ce second index stocke une structure de données supplémentaire par document, ce qui alourdit l’index.

Les indexes sans vecteur de termes sont utilisés pour chercher un document parmi une collection. Une fois le document trouvé, si nous bénéficions d’un vecteur de terme, nous pouvons affiner la recherche en récupérant où est située l’information pertinente.

**Code ajouté dans CACMIndexer.onNewDocument :**

doc.add( new StoredField("id", id));

for(String author: authors.split(";")) {

  doc.add( new StringField("authors", author, Field.Store.YES));

}

doc.add( new StringField("title", title, Field.Store.YES));

if(summary != null) {

  FieldType summaryField = new FieldType();

  summaryField.setIndexOptions(IndexOptions.DOCS\_AND\_FREQS\_AND  
 \_POSITIONS\_AND\_OFFSETS);

  doc.add(new Field("summary", summary, summaryField));

}

## Utiliser différents analyseurs

**Différences entre les différents index**

* **WhitespaceAnalyzer**: Crée les tokens en séparant le texte entre chaque [espace blanc](https://lucene.apache.org/core/8_0_0/analyzers-common/org/apache/lucene/analysis/core/WhitespaceTokenizer.html). Il n’effectue aucun stemming et il conserve la casse.
* **EnglishAnalyzer** : Est un analyseur spécialisé pour l’anglais. Il est possible de trouver des analyser pour différentes langues comme le FrenchAnalyzer et le SpanishAnalyzer. Il applique un stemming adapté ainsi qu’une liste de stopwords spécifique à l’anglais. De plus, tout est convertit en minuscule. Les tokens sont formées à partir d’un [StandardTokenizer](https://lucene.apache.org/core/8_0_0/core/org/apache/lucene/analysis/standard/StandardTokenizer.html) qui est un tokenizer basé sur la grammaire.
* **ShingleAnalyzerWrapper** est un wrapper. Cela signifie qu’il va se baser sur un autre analyseur avant d’effectuer des tokens formés de N mots. Dans le cadre du laboratoire, nous avons encapsulé un StandardAnalyzer. De ce fait, notre ShingleAnalyzerWrapper créé des tokens à partir d’un StandardTokenizer. Les tokens sont convertis en minuscules et ne sont pas stemmé. En plus des effets du StandardAnalyzer, le ShingleAnalyzerWrapper crée des tokens à partir de deux ou trois mots adjacents, toujours non stemmés et en minuscule.
* **StopAnalyzer** crée des token à partir d’un LetterTokenizer, il divise le texte aux endroits qui ne sont pas des lettres. Il convertit en minuscule et applique une liste de stopwords passée à la construction.

**Inspection des index**

|  |  |
| --- | --- |
| WhitespaceAnalyzer | |
| Documents indexés | 3202 |
| Termes indexés global | 21539 |
| Termes indexés summary | 15655 |
| Top 10 termes | of, the, is, and, a, to, in, for, The, are |
| Taille de l’index | 1.053 MB |
| Temps d’indexation | 943 ms |

|  |  |
| --- | --- |
| EnglishAnalyzer | |
| Documents indexés | 3202 |
| Termes indexés global | 11149 |
| Termes indexés summary | 5265 |
| Top 10 termes | which, us, comput, program, system, present, describ, paper, can, method |
| Taille de l’index | 0.79 MB |
| Temps d’indexation | 1054 ms |

|  |  |
| --- | --- |
| ShingleAnalyzerWrapper 1 2 | |
| Documents indexés | 3202 |
| Termes indexés global | 84492 |
| Termes indexés summary | 78608 |
| Top 10 termes | the, of, a, is, and, to, in, for, are, of the |
| Taille de l’index | 2.401 MB |
| Temps d’indexation | 1984 ms |

|  |  |
| --- | --- |
| ShingleAnalyzerWrapper 1 3 | |
| Documents indexés | 3202 |
| Termes indexés global | 136069 |
| Termes indexés summary | 130185 |
| Top 10 termes | the, of, a, is, and, to, in, for, are, this |
| Taille de l’index | 3.321 MB |
| Temps d’indexation | 2418 ms |

|  |  |
| --- | --- |
| StopAnalyzer | |
| Documents indexés | 3202 |
| Termes indexés global | 13816 |
| Termes indexés summary | 7932 |
| Top 10 termes | computer, system, paper, presented, time, program, data, method, algorithm, discussed |
| Taille de l’index | 0.745 MB |
| Temps d’indexation | 772 ms |

**Trois phrases de conclusion basées sur les observations précédentes**

Lors de l’utilisation d’un analyseur sans stopword qui forme des tokens basés sur les mots et non pas sur une combinaison de mots, nous perdons du temps et de l’espace à indexer des tokens vides de sens.

Dans tous les cas, la différence entre les termes globaux et les termes du summary vaut 5884. Nous en déduisons que l’analyseur n’a un impact que sur le TextField représentant le summary.

L’analyseur spécialisé en anglais n’est pas complétement pertinent dans son indexation. Nous retrouvons dans le top 10 des mots vides de sens tels que which et us. Ces derniers sont même dans le top 2.

Le temps d’indexation, le poids de l’index et le nombre de termes sont corrélés.

## Lire l’index

Pour cette partie, nous avons utilisé l’index créé à partir du StopAnalyzer.

**Quel est l’auteur avec le plus de publications**

**Une image contenant texte

Description générée automatiquement**

Officiellement l’auteur avec le plus de publications est l’auteur qui n’existe pas. N’ayant aucune précision quant à la façon de considérer ce cas, nous avons créé une deuxième version qui ne considère pas les auteurs vides.

Une image contenant texte

Description générée automatiquement

Cette fois-ci les termes vides ne sont pas pris en compte dans la comparaison. Nous retrouvons donc *Thacher Jr., H. C.* en première position.

**Listez le top 10 des termes des titres avec leurs fréquences**

**Une image contenant texte

Description générée automatiquement**

**Code**

Cette version ne différencie pas les termes vide ( « » ).

public void printTopRankingTerms(String field, int numTerms) {

    TermStats[] stats = null;

    try {

      /\* ------------ Empty terms like others --------------\*/

      stats = HighFreqTerms.getHighFreqTerms(indexReader, numTerms, field,  
 Comparator.comparingLong(a -> a.totalTermFreq));

    } catch (Exception e) {

      e.printStackTrace();

    }

    System.out.println("Top 10 ranking terms for field [" + field +"] are: ");

    if( stats != null) {

      for(TermStats stat : stats) {

        System.out.println( " \* " + stat.totalTermFreq + " time(s) in " +  
 ! stat.docFreq + " docs: |" + stat.termtext.utf8ToString() + "|");

      }

    }

  }

Nous avons écrit une version qui ne recherche pas les termes vides

public void printTopRankingTerms(String field, int numTerms) {

    TermStats[] stats = null;

    try {

      /\* ------------ Empty terms at the end --------------\*/

      stats = HighFreqTerms.getHighFreqTerms(indexReader, numTerms, field,

          (a,b) -> {

            if (a.termtext.utf8ToString().equals("") ^  
 b.termtext.utf8ToString().equals("")) {

              if (a.termtext.utf8ToString().equals("")) return -1;

              if (b.termtext.utf8ToString().equals("")) return 1;

            }

            return Long.compare(a.totalTermFreq, b.totalTermFreq);

      });

    } catch (Exception e) {

      e.printStackTrace();

    }

    // [...] same ending

  }

## Recherches

Code produit :

public void query(String q) {

    // Text used on query

    System.out.print("\nSearching for [" + q +"]");

    QueryParser parser = new QueryParser("summary", analyzer);

    try {

      // Query prepare

      Query query = parser.parse(q);

System.out.println("as: " + query.toString("summary"));

      // Query exec for top 10 best results

      TopDocs results = indexSearcher.search(query, 10);

      System.out.println("Total result: " + results.totalHits);

      // Printing top 10

      System.out.println("\nTop 10: ");

      ScoreDoc[] hits = results.scoreDocs;

      for(ScoreDoc hit : hits) {

        Document doc = indexSearcher.doc(hit.doc);

        System.out.println(  
 doc.get("id")+ ": " + doc.get("title") + " (" + hit.score + ")");

      }

    } catch (ParseException | IOException e) {

      e.printStackTrace();

    }

  }

### Diverses exécutions

|  |  |  |
| --- | --- | --- |
| Publications contenant le terme « Information Retrieval » | | |
| Avant parsing | **Après parsing** | **Nombre de hits** |
| "Information Retrieval" | "inform retriev" | 20 |

|  |  |  |
| --- | --- | --- |
| Top 10 | | |
| Id | **Score** | **Titre** |
| 891 | 1.5950764 | Everyman's Information Retrieval System |
| 1935 | 1.4266797 | Randomized Binary Search Technique |
| 2288 | 1.3989745 | File Organization: The Consecutive Retrieval Property |
| 1457 | 1.3023745 | Data Manipulation and Programming Problems in Automatic Information Retrieval |
| 655 | 1.2811962 | COMIT as an IR Language |
| 1699 | 1.1888998 | Experimental Evaluation of Information Retrieval Through a Teletypewriter |
| 1514 | 1.1007073 | On the Expected Gain From Adjust ing Matched Term Retrieval Systems |
| 1627 | 1.0754004 | Application of Level Changing to a Multilevel Storage Organization |
| 2519 | 1.0088149 | On the Problem of Communicating Complex Information |
| 2650 | 1.0088149 | Order-n Correction for Regular Languages |

|  |  |  |
| --- | --- | --- |
| Publications contenant le terme « Information » et le terme « Retrieval » | | |
| Avant parsing | **Après parsing** | **Nombre de hits** |
| Information && Retrieval | +inform +retriev | 36 |

|  |  |  |
| --- | --- | --- |
| Top 10 | | |
| Id | **Score** | **Titre** |
| 2288 | 1.9864943 | File Organization: The Consecutive Retrieval Property |
| 891 | 1.7366376 | Everyman's Information Retrieval System |
| 1457 | 1.7002742 | Data Manipulation and Programming Problems in Automatic Information Retrieval |
| 3134 | 1.5496716 | The Use of Normal Multiplication Tables for Information Storage and Retrieval |
| 1032 | 1.5208457 | Theoretical Considerations in Information Retrieval Systems |
| 1699 | 1.4560988 | Experimental Evaluation of Information Retrieval Through a Teletypewriter |
| 1935 | 1.4266797 | Randomized Binary Search Technique |
| 1514 | 1.3622651 | On the Expected Gain From Adjust ing Matched Term Retrieval Systems |
| 2519 | 1.3236521 | On the Problem of Communicating Complex Information |
| 655 | 1.2811962 | COMIT as an IR Language |

|  |  |  |
| --- | --- | --- |
| Publications contenant le terme « Retrieval », possiblement « Information » et pas « Database » | | |
| Avant parsing | **Après parsing** | **Nombre de hits** |
| +Retrieval Information NOT Database | +retriev inform -databas | 69 |

|  |  |  |
| --- | --- | --- |
| Top 10 | | |
| Id | **Score** | **Titre** |
| 2288 | 1.9864943 | File Organization: The Consecutive Retrieval Property |
| 891 | 1.7366376 | Everyman's Information Retrieval System |
| 1457 | 1.7002742 | Data Manipulation and Programming Problems in Automatic Information Retrieval |
| 3134 | 1.5496716 | The Use of Normal Multiplication Tables for Information Storage and Retrieval |
| 1032 | 1.5208457 | Theoretical Considerations in Information Retrieval Systems |
| 1699 | 1.4560988 | Experimental Evaluation of Information Retrieval Through a Teletypewriter |
| 1935 | 1.4266797 | Randomized Binary Search Technique |
| 1514 | 1.3622651 | On the Expected Gain From Adjust ing Matched Term Retrieval Systems |
| 2519 | 1.3236521 | On the Problem of Communicating Complex Information |
| 655 | 1.2811962 | COMIT as an IR Language |

|  |  |  |
| --- | --- | --- |
| Publications contenant un terme commençant par « Info » | | |
| Avant parsing | **Après parsing** | **Nombre de hits** |
| Info\* | info\* | 205 |

|  |  |  |
| --- | --- | --- |
| Top 10 | | |
| Id | **Score** | **Titre** |
| 222 | 1.0 | Coding Isomorphisms |
| 272 | 1.0 | A Storage Allocation Scheme for ALGOL 60 |
| 396 | 1.0 | Automation of Program Debugging |
| 397 | 1.0 | A Card Format for Reference Files in Information Processing |
| 409 | 1.0 | CL-1, An Environment for a Compiler |
| 440 | 1.0 | Record Linkage |
| 483 | 1.0 | On the Nonexistence of a Phrase Structure Grammar for ALGOL 60 |
| 616 | 1.0 | An Information Algebra - Phase I Report-Language Structure Group of the CODASYL Development Committee |
| 644 | 1.0 | A String Language for Symbol Manipulation Based on ALGOL 60 |
| 655 | 1.0 | COMIT as an IR Language |

|  |  |  |
| --- | --- | --- |
| Publications contenant un terme commençant par « Info » | | |
| Avant parsing | **Après parsing** | **Nombre de hits** |
| "Information Retrieval"~5 | "inform retriev"~5 | 30 |

|  |  |  |
| --- | --- | --- |
| Top 10 | | |
| Id | **Score** | **Titre** |
| 891 | 1.5950764 | Everyman's Information Retrieval System |
| 1935 | 1.4266797 | Randomized Binary Search Technique |
| 2288 | 1.3989745 | File Organization: The Consecutive Retrieval Property |
| 1457 | 1.3023745 | Data Manipulation and Programming Problems in Automatic Information Retrieval |
| 655 | 1.2811962 | COMIT as an IR Language |
| 1699 | 1.1888998 | Experimental Evaluation of Information Retrieval Through a Teletypewriter |
| 1514 | 1.1007073 | On the Expected Gain From Adjust ing Matched Term Retrieval Systems |
| 1627 | 1.0754004 | Application of Level Changing to a Multilevel Storage Organization |
| 2519 | 1.0088149 | On the Problem of Communicating Complex Information |
| 2650 | 1.0088149 | Order-n Correction for Regular Languages |

## Personnaliser le score de Lucene

### MySimilarity

|  |  |  |
| --- | --- | --- |
| Top 10 | | |
| Id | **Score** | **Titre** |
| 1647 | 13.897842 | WATFOR-The University of Waterloo FORTRAN IV Compiler |
| 637 | 13.25242 | A NELIAC-Generated 7090-1401 Compiler |
| 2534 | 12.476707 | Design and Implementation of a Diagnostic Compiler for PL/I |
| 2923 | 12.421366 | High-Level Data Flow Analysis |
| 2423 | 11.444084 | A Parser-Generating System for Constructing Compressed Compilers |
| 1807 | 10.668371 | Optimization of Expressions in Fortran |
| 1135 | 10.559103 | A General Business-Oriented Language Based on Decision Expressions\* |
| 1788 | 10.559103 | Toward a General Processor for Programming Languages |
| 1988 | 10.360685 | A Formalism for Translator Interactions |
| 2652 | 10.360685 | Reduction of Compilation Costs Through Language Contraction |

### ClassicSimilarity

|  |  |  |
| --- | --- | --- |
| Top 10 | | |
| Id | **Score** | **Titre** |
| 3189 | 1.400693 | An Algebraic Compiler for the FORTRAN Assembly Program |
| 1215 | 1.3382424 | Some Techniques Used in the ALCOR ILLINOIS 7090 |
| 1988 | 1.2983286 | A Formalism for Translator Interactions |
| 799 | 1.2843311 | Design of a Separable Transition-Diagram Compiler\* |
| 1647 | 1.2778773 | WATFOR-The University of Waterloo FORTRAN IV Compiler |
| 1122 | 1.2518111 | A Note on Some Compiling Algorithms |
| 1183 | 1.2494278 | A Note on the Use of a Digital Computer for Doing Tedious Algebra and Programming |
| 1459 | 1.2196473 | Requirements for Real-Time Languages |
| 637 | 1.1973245 | A NELIAC-Generated 7090-1401 Compiler |
| 46 | 1.193163 | Multiprogramming STRETCH: Feasibility Considerations |

Code produit :

@Override  
public float tf(float freq)  
{  
 return 1f + (float) Math.*log*(freq);  
}  
  
@Override  
public float idf(long docFreq, long docCount)  
{  
 return (float)Math.*log*(docCount / (docFreq + 1.0)) + 1f;  
}  
  
@Override  
public float lengthNorm(int numTerms)  
{  
 return 1f;  
}

On peut remarquer qu’avec les nouveaux paramètres, le score a fondamentalement augmenté. De plus, l’ordre n’est plus le même. On peut cependant apercevoir certains titres qui reviennent, notamment :

* WATFOR-The University of Waterloo FORTRAN IV Compiler, id : 1647
* A Formalism for Translator Interactions, id : 1988

Nous avons également pu constater que c’était l’override de la méthode lengthNorm qui changeait majoritairement le score. L’override que nous avons fait permet de ne pas s’intéresser à la taille des Field lors du calcul du score. La normalisation de la longueur étant pareil pour chaque Field, il est maintenant possible d’obtenir des valeurs non normalisées, qui sont en général bien plus extrêmes.