Cours de Data Mining – Indexation des documents

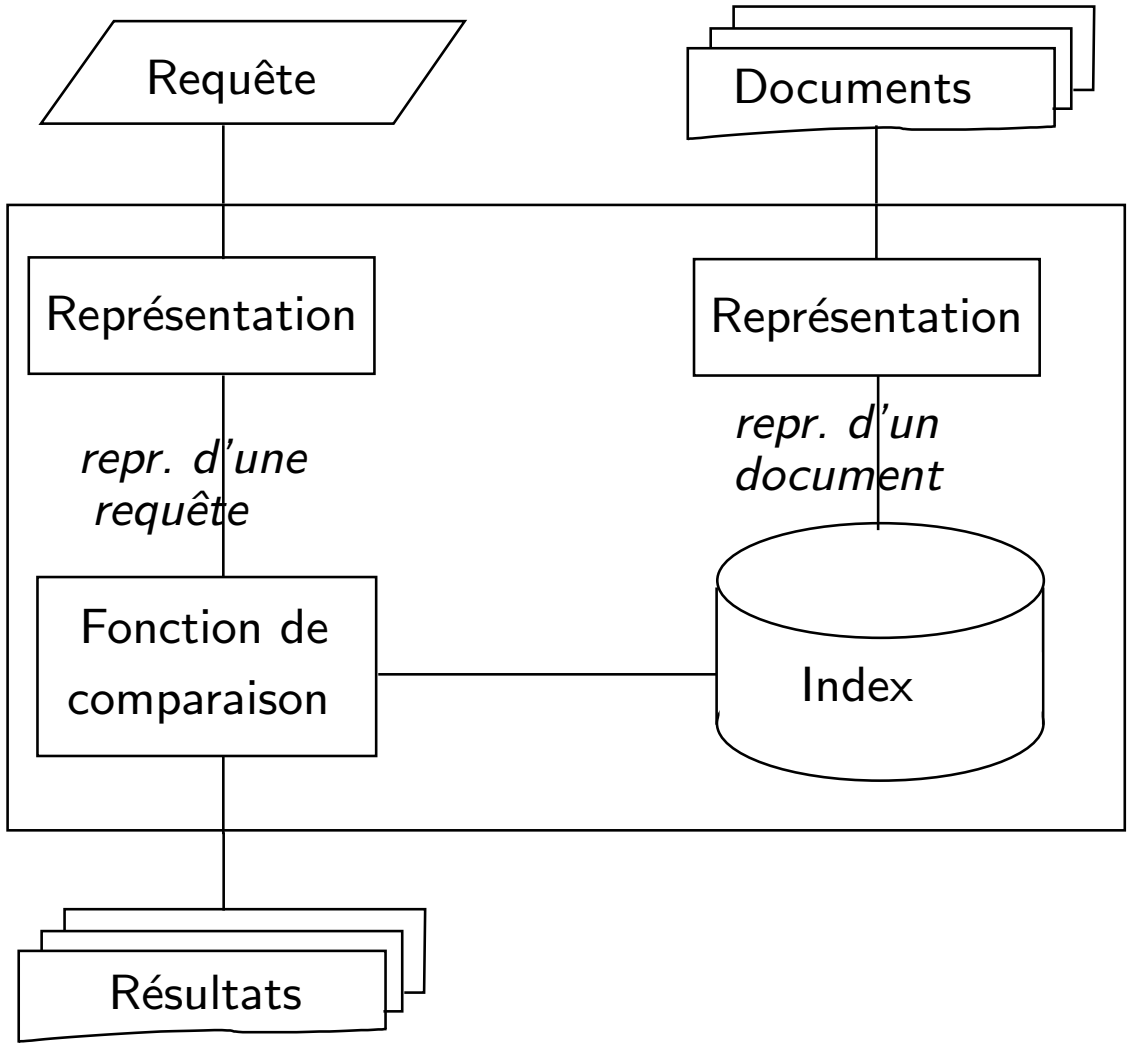
Andreea Dragut

Univ. Aix-Marseille, IUT d'Aix-en-Provence

Plan du cours – Recherche d'information

- Indexation
- Mesures de Similarité
- Requêtes

Indexation et Recherche : La boîte noire "RI"



Problème de la RI

Recherche d'information

Auteur

concepts

concepts

Termes de la recherche $\Longleftrightarrow$ Termes du document

sont-ils pareils ?

Modèles de RI

- *L'approche ensembliste* classique BD :
  - l'ensemble des documents s'obtient par une série d'opérations ($\cup$, $\cap$, passage au complémentaire, le langage de requêtes SQL)
- *L'approche algébrique* (ou vectorielle) :
  - les documents et les requêtes font partie d'un même espace vectoriel
- *L'approche probabiliste* :
  - essaie de modéliser la notion de pertinence

Le correcteur orthographique

Le lemmatiseur

L'anti dictionnaire supprime dans l'index et dans les requêtes tous les mots vides de contenu

Indexation : filtrage des mots « vides »

- Les mots très fréquents n'aident pas à différencier les documents
- Liste des mots fonctionnels=modes « vides » de contenu
  - Anglais : a, about, above, according, across, after, afterwards, again, against, albeit, all, almost, alone, already, also, although, always, among, as, at
  - Français : de, un, les, le, la

Indexation : stemming/lemmatisation

- Stemming= effacer les terminaisons :
  - flexions de nombre, genre
  - conjugaison
  - déclinaison
  - enlevement de suffixes/suffix stripping
    - develop $\Rightarrow$ develop
    - developing $\Rightarrow$ develop
    - developments $\Rightarrow$ develop afin de retrouver les racines des mots
- Les fréquences des mots cumulent les occurences des variations des mêmes mots
- Perte de précision

Exemple d'index inversé

Doc1

Ceci est un exemple de document avec un exemple de phrase

Doc2

Ceci est un autre exemple de document

	Dictionnaire		Assignations	
Terme	Nbr docs	Fréq totale	Doc Id	Fréq
Ceci	2	2	1	1
			2	1
est	2	2	1	1
			2	1
exemple	2	3	1	2
			2	1
autre	1	1	2	1
...	...	...	...	...

API (interface de programmation -Application Programming Interface) :
une interface (un ensemble de fonctions, procédures ou classes) fournie par un programme informatique permettant l'interaction des programmes
`http://lucene.apache.org/java/3_0_1/api/core/org/apache/lucene/`

org.apache.lucene.search

Class Similarity

java.lang.Object

org.apache.lucene.search.Similarity

All Implemented Interfaces : Serializable

Direct Known Subclasses :

DefaultSimilarity, SimilarityDelegator

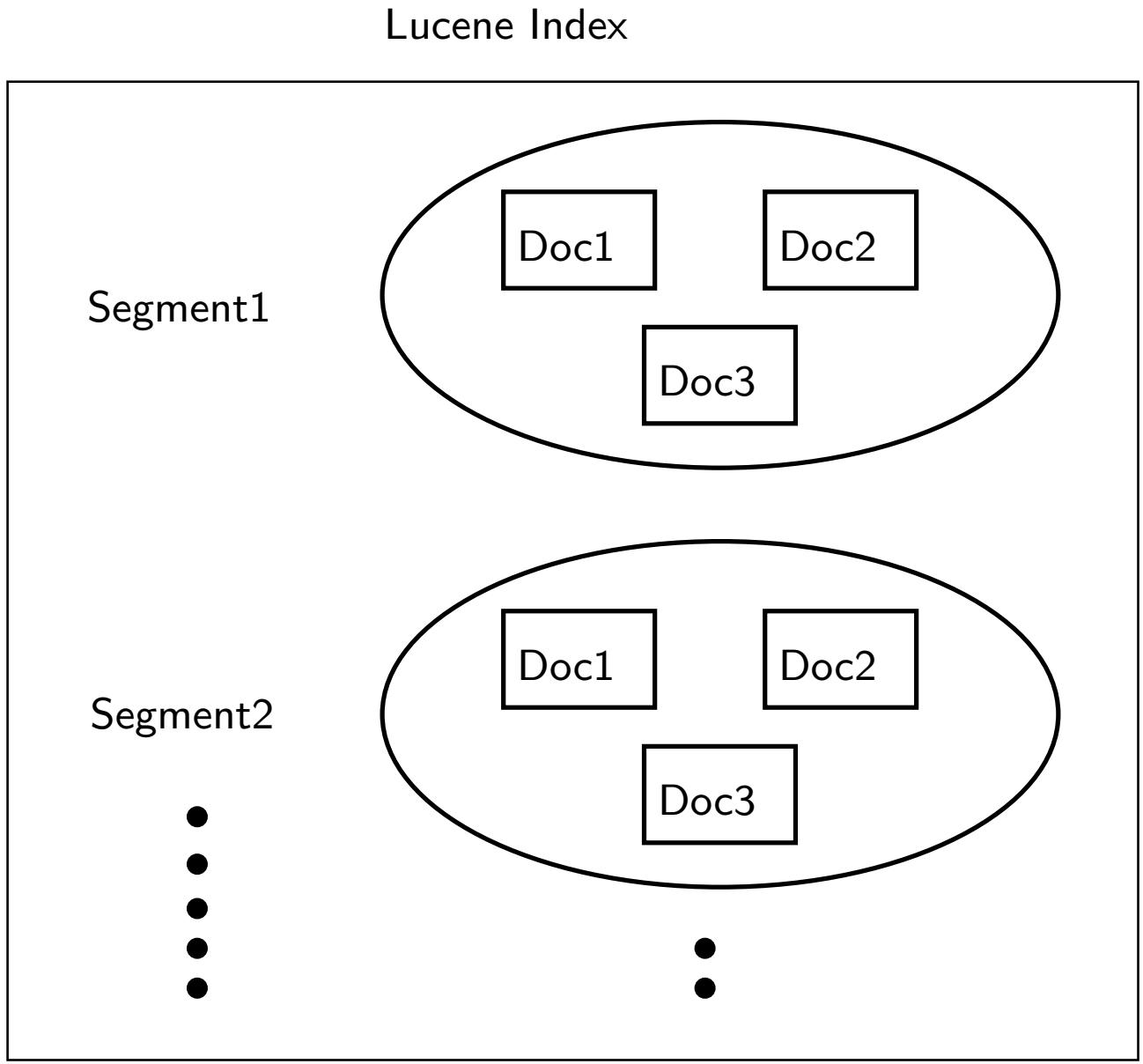
Modèles utilisés par Lucene

Boolean model (BM) l'approche ensembliste des BD

Vector Space Model (VSM) l'approche vectorielle

Les documents pertinants sont obtenus avec BM et présentés dans l'ordre donnée par VSM.

Structure de l'index



Création d'un index

- les données à indexer : les fichiers HTML,PDF, Word, Text d'un répertoire **Test**
- **prétraitement pour extraire du fichier le texte (ex. : JSOUP)**
  - **un analyseur de HTML filtre le texte à l'aide des étiquettes HTML**
- **construction d'un objet de type FSDirectory pour gerer l'ecriture sur disque d'un index**
- **configuration des objets de la classe Document**
- **construction d'un objet de type IndexWriter**

.add() **rajoute un objet Document**
.close() **écrit l'index sur disque, afin de pouvoir rechercher dedans**

`SimpleLuceneIndex` — Indexation des documents

- Indexation – construction de l'index à la Lucene – classe `IndexWriter`
- Répertoriage des documents – classe `Document`
  - identificateur,
  - titre
  - url de provenance
  - contenu (le texte)
  - liens sortants
  - résultats de l'analyse (score, etc.)
- $\Longrightarrow$ ensemble de "champs" d'indexation – classe `Field`
- $\Longrightarrow$ un objet de la classe `Field` pour chaque champ
- $\Longrightarrow$ liste `C` des (types de) champs à pourvoir pour chaque `Document`
- Données fournies en entrée :
  - fichiers texte (téléchargés : crawler) – avec leur docId dans leur nom
  - un fichier `P` avec des paires (url de provenance,docId du fichier)

SimpleLuceneIndex — Indexation des documents

Étapes du traitement :

1. création d'une liste `L` énumérant tous les docId (par simple lecture de `P`)
2. construction objet `X` de la classe `IndexWriter`
3. "rassemblage des docs" : pour chaque docId `d` de la liste `L`
   1. création objet `D` de la classe `Document`
   2. pour chaque type champ d'indexation de la liste `C`
      1. création objet `F` de la classe `Field` avec configuration appropriée
      2. rajout de `F` dans `D` (méthode `Document::add()`)
   3. rajout de `D` dans `X` (méthode `IndexWriter::addDocument()`)

Concrètement, dans le code à réaliser

- étape 1 création `L` — méthode `SimpleLuceneIndex::setupDocIds()`
- étape 3 "rassemblage des docs" — méthode `SimpleLuceneIndex::gatherAndIndexDocs()`
- le tout à partir du constructeur de la classe `SimpleLuceneIndex`

SimpleLuceneIndex – répertoire

```
docIdList              = null
setupDocIds(docPath+"/"+inputListFile);

FSDirectory  indexDir = FSDirectory.open (new File(indexPath));
indexWriter           = new IndexWriter(indexDir,
                                        new StandardAnalyzer(),
                                        true);
System.out.println("Optimized Indexing to directory '" +
                   indexDir + "'...");
gatherAndIndexDocs(docPath);
indexWriter . optimize();
indexWriter . close();
```

SimpleLuceneIndex – stemming/lemmatisation

```
Analyzer luceneAnalyzer = new StandardAnalyzer();
```

Variantes :

WhitespaceAnalyzer	Un analyseur très simple qui sépare juste la marque en utilisant l'espace blanc.
StopAnalyzer	Enlève les mots anglais communs et de liaison, inutiles pour l'indexation.
SnowballAnalyzer	Un analyseur expérimental intéressant qui travaille sur le racines (recherche sur la raining renvoie aussi rained, rain)
FrenchAnalyzer	Un analyseur pour le francais

SimpleLuceneIndex – création de l'index

```
IndexWriter indexWriter = new IndexWriter(indexDir,luceneAnalyzer,true);
...
indexWriter.close();
```

- cette classe peut créer un nouvel index ou ouvrir un index existant et lui ajouter des documents.
- le premier paramètre : ou on stocke les fichiers d'index ;
- le deuxième paramètre : l'analyseur qui sera employé ;
- le dernier paramètre : si vraie, la classe crée un nouvel index ; si faux, il ouvre un index existant
- en fermant l'index on l'écrit

SimpleLuceneIndex – format de Document

```
String       textFile  = path + "/download_" + docId + ".txt";
System.out.println("gatherAndIndexDocs(): Processing doc " + textFile);
String       docText   = readTextFile(textFile);
Document     luceneDoc = new Document();
luceneDoc . add(new Field("content",   docText,
      Field.Store.NO,
      Field.Index.ANALYZED,
      Field.TermVector.YES));
```

- cette classe crée un nouvel document virtuel : collection de champs
- chaque champ a un nom de champ, son contenu peut etre stocké dans le document (ou non)
- le champ peut (ou pas servis) à indexer

SimpleLuceneIndex – base de donées

```
final File INDEX_DIR = new File(indexPath);

try{
   Class.forName("com.mysql.jdbc.Driver").newInstance();
   Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/test",
                                                 "root", "password");
   IndexWriter writer = new IndexWriter(INDEX_DIR, new StandardAnalyzer(), true);
   System.out.println("Indexing to directory '" + INDEX_DIR + "'...");
   indexDocs(writer, conn);
   writer.optimize();
   writer.close();
} catch (Exception e) {
   e.printStackTrace();
}
```

SimpleLuceneIndex – base de donées, employe :"id" "name"

```
void indexDocs(IndexWriter writer, Connection conn) throws Exception {
  String sql = "select id, name from employe";
  Statement stmt = conn.createStatement();
  ResultSet rs = stmt.executeQuery(sql);
  while (rs.next()) {
     Document d = new Document();
     d.add(new Field("id", rs.getString("id"), Field.Store.YES,
                     Field.Index.NO));
     d.add(new Field("name", rs.getString("name"), Field.Store.NO,
                     Field.Index.TOKENIZED));
     writer.addDocument(d);
 }
}
```

Similarity(). Méthodes :

float computeNorm(String field, FieldInvertState state) Compute the normalization value for a field, given the accumulated state of term processing for this field (see FieldInvertState).

abstract float idf(int docFreq, int numDocs) Computes a score factor based on a term's document frequency (the number of documents which contain the term).

abstract float tf(float freq) Computes a score factor based on a term or phrase's frequency in a document.

....

float tf(int freq) Computes a score factor based on a term or phrase's frequency in a document.

abstract float sloppyFreq(int distance) Computes the amount of a sloppy phrase match, based on an edit distance.

public abstract float idf(int docFreq,int numDocs)

Returns : a score factor based on a term's within-document frequency

idf :la fréquence inverse de document (inverse document frequency)

- donne l'importance d'un terme dans l'ensemble du corpus des documents
- considère plus discriminants les termes les moins fréquents
- est le logarithme de l'inverse de la proportion de documents du corpus qui contiennent le terme

$$\mathrm{idf_i} = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|} \quad (1)$$

où

- $|D|$ est le nombre total de documents dans le corpus
- $|\{d_j : t_i \in d_j\}|$ le nombre de documents contenant le terme t_i

public float tf(int freq)

Returns : a score factor based on a term's within-document frequency

tf : la fréquence d'un terme (term frequency)

- donne l'importance d'un terme dans un document
- est le nombre d'occurrences du terme dans le document considéré, normalisée

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}} \quad (2)$$

où

- d_j un document,
- t_i un terme,
- $n_{i,j}$ le nombre d'occurrences du terme t_i dans d_j.

Calcul de tf-idf : multiplication de deux mesures :

$$\mathrm{tfidf_{i,j}} = \mathrm{tf_{i,j}} \cdot \mathrm{idf_i} \quad (3)$$

public float computeNorm(String field,FieldInvertState state)

Document :

Lucene termFreq=<term, countD> <fieldName,<...,termFreqi,...> >

Acces Java public String getField() ; public String [] getTerms() et getTermFrequencies() ;

Au moment de l'indexation pour chaque champ/field :

$$n(t,d) = doc.getBoost() \times lengthNorm(field) \times \prod_{t \in field f \in d} f.getBoost() \quad (4)$$

Document/Champ boost setBoost() initialise l'importance d'un document, champ/field et getBoost() la retrouve.

lengthNorm(champ/field) public abstract float lengthNorm(String fieldName, int numTokens)normalise l'importance d'un champ du document. Les champs plus courts comptent plus.

n(t,d) la norme résultat est codée que sur un octet et $decod(cod(x)) \neq x$

Facteur de boosting : par défaut 1, changer avec le mot clé "^"de la syntaxe d'une requête

Boosting d'un terme : jakarta^4 apache ou jakarta^0.2 apache

Boosting d'un terme composé : "jakarta apache"^4 "Apache Lucene"

Retrouver l'importance d'une requête r : r.getBoost()

- **t.getBoost()** donne le facteur d'importance du terme t de la requête r. Il n'y a pas d'API directe : pour chaque terme d'une recherche multiple $\Rightarrow$ un objet TermQuery $\Rightarrow$ getBoost() pour la sous-requête.
- **queryNorm(q)** normalise les requêtes en utilisant ce que renvoie **public float queryNorm(float sumOfSquaredWeights)**

$$\frac{1}{\sqrt{(sumOfSquaredWeights)}} \quad (5)$$

$$sumOfSquaredWeights = r.getBoost()^2 \times \sum t \in r\,(idf(t)t.getBoost())^2 \quad (6)$$

Similarité entre un document et une requête : cosinus

Requête : $r = (t_{1,r}, t_{2,r}, ..., t_{k,r})$

Document : $d_j = (t_{1,j}, t_{2,j}, ..., t_{k,j})$

$$\cos(\overrightarrow{r}, \overrightarrow{d_j}) = \frac{\langle \mathbf{d_j}, \mathbf{r} \rangle}{\|\mathbf{d_j}\| \, \|\mathbf{r}\|} \quad (7)$$

où

- $\langle \mathbf{d_j}, \mathbf{r} \rangle = \sum_{i=1}^{n} t_{i,r} \times t_{i,j} = t_{1,r} t_{1,j} + \cdots + t_{k,r} t_{k,j}$
- la norme d'un vecteur v est : $\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$

Cette mésure ne tient pas compte de la longueur des documents.

Requête : $r = (t_{1,r}, t_{2,r}, ..., t_{k,r})$

Document : $d_j = (t_{1,j}, t_{2,j}, ..., t_{k,j})$

$$\cos(\overrightarrow{r}, \overrightarrow{d_j}) = \frac{\langle \mathbf{d_j}, \mathbf{r} \rangle}{\|\mathbf{d_j}\| \, \|\mathbf{r}\|} \quad (8)$$

Ajustements pour la taille et pour l'importance dans le document et dans la requête : score(r,d)

$$reqLength(r,d) \times reqBoost(r) \times \frac{\langle \mathbf{d_j}, \mathbf{r} \rangle}{\|\mathbf{r}\| \, \|\mathbf{r}\|} \times docLength(d) \times docBoost(d)$$

API – Lucene score

Requête : $r = (t_{1,r}, t_{2,r}, ..., t_{t,r})$

Document : $d_j = (t_{1,j}, t_{2,j}, ..., t_{t,j})$

$$score(r, d_j) = coord(r, d_j) \times queryNorm(r) \times \\ \times \sum_{t \in r} \left(tf(tind_j) \times idf^2(w) \times t.getBoost() \times n(t, d_j) \right)$$

où

$tf(tind_j) = freq^{1/2}$, le nombre de fois que le terme t apparaît dans d_j

$idf(t)$ utilise idf(int docFreq,int numDocs) pour donner l'importance du terme t dans l'ensemble du corpus des documents

$queryNorm(q)$ utilise public float queryNorm(float sumOfSquaredWeights) pour normaliser les requêtes

$t.getBoost()$ le facteur d'importance du terme t de la requête r.

$n(t, d_j)$ la norme du terme t calculé à l'indexation du d_j

public abstract float sloppyFreq(int distance)

Returns : the frequency increment for this match

sloppyFreq :

- compte les occurences de phrases semblables à des phrases non-desirées
- ”semblable” mesuré avec la **distance d’édition** : le nombre minimal de caractères qu’il faut supprimer, insérer ou remplacer pour passer d’une chaîne à l’autre.

Documentation Java – index `Document`, `IndexWriter`

```
public final class Document
       implements java.io.Serializable {
 // Les Documents sont l'unite pour l'indexation et la recherche
 // Un Document est un ensemble de Fields
 // Chaque Field a un nom et une valeur textuelle
```

Documentation Java – index `Document`, `IndexWriter`

```
 public Document()                              // constructeur d'un nouveau
                                                // document, sans Fields
 public final void add(Fieldable field)         // rajout d'un Field
 public final String get(String name)           // obtenir la valeur
                                                // textuelle d'un Field
}
 // Un Field peut etre stocke dans le document (ou non). On
 // l'indique lors de la construction de l'objet Field, avec
 // les autres attributs (a indexer ou pas, etc.).
```

Documentation Java – index Field

```
public final class Field
   extends AbstractField implements Fieldable, Serializable {
  public static final class Store
     extends Parameter implements Serializable {
    public static final Store YES = new Store("YES");
    public static final Store NO  = new Store("NO");
  }
  public static final class Index
     extends Parameter implements Serializable {
    public static final Index ANALYZED     = new Index("ANALYZED");
    public static final Index NOT_ANALYZED = new Index("NOT_ANALYZED");
    public static final Index NO           = new Index("NO");
  }
  public static final class TermVector
     extends Parameter implements Serializable {
    public static final TermVector YES = new TermVector("YES");
  }
}
```

Documentation Java – index IndexWriter

```
public class IndexWriter {
  // L'IndexWriter cree et maintient un index
  public void addDocument(Document doc) // rajoute un document dans
                                        // l'indexe, une fois celui-ci
                                        // proprement configure

  public void optimize()                // fait en sorte que l'index
                                        // permette des recherches
                                        // plus rapides -- a appeler
                                        // une fois avoir fini de
                                        // rajouter de docs (pour un
                                        // bout de temps)

  public void close()                   // finalise l'ecriture de
                                        // l'index sur disque, afin de
                                        // pouvoir par la suite
                                        // rechercher dedans
}
public class FSDirectory extends Directory {
  public static FSDirectory open(File path) // pour gerer l'ecriture
                                        // sur disque d'un index
}
```

Documentation Java – index Query

```
public abstract class TopDocsCollector extends Collector {
   public final TopDocs topDocs() // utilisee pour le resultat
}
public abstract class TopScoreDocCollector extends TopDocsCollector {
 // pour collecter les resultats de la recherche dans l'index
 // dans l'ordre decroissant de leur score

  public static TopScoreDocCollector create(int numHits,
                                            boolean docsScoredInOrder)
  // cree un nouveau collecteur, limitant le nombre de documents
  // on l'appellera avec 'true'
}

public class QueryParser implements QueryParserConstants {
   public QueryParser(String f, Analyzer a) // constructeur, que nous
                                    // allons utiliser avec un nouvel
                                    // objet StandardAnalyzer

  public Query parse(String query) // analyse lexicalement une requete
                                    // construit un objet Query pour la
                                    // representer, et le rend
}
```

Documentation Java – index IndexSearcher

```
public abstract class Searcher implements Searchable {
  public void search(Query query, Collector results) // recherche
  // deposant les resultats dans le collecteur donne en parametre
}

public class IndexSearcher extends Searcher {
   public IndexSearcher(Directory directory) // constructeur
   public Document doc(int i) // pour retrouver le Document a partir
                              // de son docId
}
public class TopDocs implements java.io.Serializable {
  public ScoreDoc[] scoreDocs;  // les resultats de la recherche
}
public class ScoreDoc implements java.io.Serializable {
 // represente un document parmi ceux retrouves suite a une recherche
 // obtenue depuis la donnee-membre scoreDocs
 public int doc; // le docId
}
```