PRÁCTICA III:

...

Implementación de un Sistema de Recuperación de Información utilizando Lucene

...

Parte B: Indexación

28 de octubre de 2019

Índice

1. Objetivo

El objetivo final de esta práctica es que el alumno comprenda todos los procesos que intervienen en el diseño de un Sistema de Recuperación de Información y cómo puede ser implementado utilizando la biblioteca Lucene. Para ello, en este curso utilizaremos la base de datos WikiMovie, vista en la práctica anterior. Nuestro objetivo será construir un programa que se pueda ejecutar en línea de comandos y que sea el encargado de generar/actualizar nuestro índice.

Esta práctica, es la segunda parte de la práctica final que representará el 80 % de la parte práctica de asignatura. Para su realización se ha dividido el trabajo en cuatro grandes bloques, haciendo referencia esta documentación a la segunda de ellas. En las siguientes semanas se entregará la documentación relativa a las otras partes, relacionadas con búsqueda y uso de facetas.

2. Procesando los documentos

Una vez que conocemos los datos de nuestra práctica, podremos imaginar qué tipos de consultas se podrían realizar por un usuario así como la/las respuesta que daría el sistema, en la cual se devuelven los elementos ordenados por relevancia, facilitando las labores del usuario. Lucene permite hacer consultas por campos, por lo que será necesario identificar los mismos. Los campos concretos dependerá de la aplicación concreta sobre la que estemos trabajando, pero como hemos visto si es necesario un mismo campo podrá utilizarse para dos funciones distintas, por ejemplo como texto sin tokenizar y texto tokenizado.

En cualquier caso, en nuestra aplicación deberemos identificar al menos los siguientes tipos de campos sobre los documentos de entrada:

- Texto simple que se considera literalmente (no se tokeniza), útil para la búsqueda por facetas, filtrado de consultas y también para la presentación de resultados en la interfaz de búsqueda, por ejemplo ID, tags, direcciones webs, nombres de fichero, etc.

- Secuencia de términos que es procesada para la indexación, esto es, convertida a minúscula, tokenizada, estemizada, etc. Como podría ser el título,

resument, etc de un artículo científico o de la consulta de Stack Overflow.

- Numérico. En nuestro caso puede ser el score de una respuesta o pregunta.
- Facetas (Categorías) que permiten una agrupación lógica de los documentos con la misma faceta, como por ejemplo keywords (tags), fechas, si ha sido aceptada la respuesta, ID de usuario, si es pregunta o respuesta, etc.

Una vez que tenemos nuestra colección de documentos, el siguiente paso es convertir cada documento en texto plano que pueda ser "digerido"por Lucene, librería de referencia para implementar herramientas de recuperación de información, llegando a ser casi un estándar. Para ello podemos utilizar la librería TIKA, o herramientas más específicas como PDFBox para trabajar sobre ficheros pdf, JDOM para trabajar con ficheros XML o JSoup, Jtidy o HTMLParser para HTML.

En nuestro caso, debemos de identificar cada una de líneas de nuestro CSV asociadas a un elemento (película) y procesarlas de forma adecuada para extraer cada una de las partes con el formato adecuado.

Como entrada asumimos que tenemos cada ítem sobre un `String` y debemos transformarlo para extraer los campos de interés. Para ello, Java nos da alternativas para poder transformar un String a tipos de dato fecha (Date) o entero,

3. Indexación de documentos

Indexar es una de las principales tareas que podemos encontrar en Lucene. La clase que se encarga de la indexación de documentos es IndexWriter. Esta clase se encuentra en `lucene-core` y permite añadir, borrar y actualizar documentos Lucene. Un documento Lucene esta compuesto por un conjunto de campos: par nombre del campo (string)- contenido del campo (string), como por ejemplo (“autor”, “Miguel de Cervantes”) o (“Título”, “Don Quijote de la Mancha”) o ‘(“Cuerpo”,“En un lugar de la Mancha de cuyo....”). Cada uno de estos campos será indexado como parte de un documento, después de pasar por un Analyzer. En el Cuadro 1 podemos ver resumido el conjunto de clases que son usadas frecuentemente en la indexación.

Cuadro 1: Clases involucradas en la indexación

Clase	Descripción
IndexWriter	Clase esencial que crea/modifica un índice.
Directory	Representa la ubicación del índice.
Analyzer	Es responsable de analizar un texto y obtener los tokens de indexación.
Document	Representa un documento Lucene, esto es, un conjunto de campos (fields) asociados al documento.
Field	La unidad más básica, representa un par clave-valor, donde la clave es el nombre que identifica al campo y el valor es el contenido del documento a indexar.

El IndexWriter `http://lucene.apache.org/core/7_1_0/core/org/apache/lucene/index/IndexWriter.html` toma como entrada dos argumentos:

- Directory: Representa el lugar donde se almacenará el índice `http://lucene.apache.org/core/7_1_0/core/org/apache/lucene/store/Directory.html`. Podemos encontrar distintas implementaciones, pero para un desarrollo rápido de un prototipo podemos considerar RAMDirectory (el índice se almacena en memoria) o FSDirectory (el índice se almacena en el sistema de ficheros)

- IndexWriterConfig: Almacena la configuración utilizada `http://lucene.apache.org/core/7_1_0/core/org/apache/lucene/index/IndexWriterConfig.html`

Aunque se recomienda mirar la documentación de las distintas clases, ilustraremos su uso mediante el siguiente ejemplo,

```
1 FSDirectory dir = FSDirectory.open( Paths.get(INDEX_DIR) );
2 IndexWriterConfig config = new IndexWriterConfig( analyzer );
3 config.setOpenMode(IndexWriterConfig.OpenMode.CREATE).set; //
    Crea un nuevo indice
4 IndexWriter writer = new IndexWriter(dir, config);
5
6 List<string> listaFicheros = .....
7
```

```
8  for (String nombre: listaFicheros)
9          Document doc = ObtenerDocDesdeFichero(nombre);
10         writer.addDocument(doc);
11
12 writer.commit(); // Ejecuta todos los cambios pendientes en el
       indice
13 writer.close();
```

Las líneas 1 a 4 son las encargadas de crear las estructuras necesarias para el índice. En concreto el índice se almacenará en disco en la dirección dada por el path. La línea 2 declara un IndexWriterConfig config con el analizador que se utilizará para todos los campos (si no se indica, utilizará por defecto el standardAnalyzer). Para ello, debemos de seleccionar, de entre los tipos que tiene Lucene implementados, el que se considere adecuado para nuestra aplicación. Este es un criterio importante para poder alcanzar los resultados óptimos en la búsqueda. Puede que sea necesario el utilizar un analizador distinto para cada uno de los posibles campos a indexar, en cuyo caso utilizaremos un PerFieldAnalyzerWrapper.

La línea 3 se indica que el índice se abre en modo CREATE (crea un nuevo índice o sobreescribe uno ya existente), otras posibilidades son APPEND (añade documentos a un índice existente) o CREATE_OR_APPEND (si el índice existe añade documentos, si no lo crea para permitir la adición de nuevos documentos. Es posible modificar la configuración del índice considerando múltiples setter (por ejemplo, indicar la función de similitud que se utiliza, criterios de mezcla de índices, etc.)

Una vez definida la configuración tenemos el IndexWriter listo para añadir nuevos documentos. Los cambios se realizarán en memoria y periódicamente se volcarán al Directory, que cuando sea necesario realizará la mezcla de distintos segmentos.

Una vez añadidos los documentos, debemos asegurarnos de llamar a commit() para realizar todos los cambios pendientes, línea 12. Finalmente podremos cerrar el índice mediante el comando close (línea 13).

4. Creación de un Document Lucene

Hemos visto que para indexar la información es necesario la creación de un documento, `Document`, Lucene `http://lucene.apache.org/core/7_1_0/core/org/apache/lucene/document/Document.html`. Un Document es la unidad de indexación y búsqueda. Está compuesto por un conjunto de campos Fields, cada uno con su nombre y su valor textual. Un field puede ser el nombre de un producto, su descripción, su ID, etc. Veremos brevemente cómo se gestionan los Fields ya que es la estructura básica de la que está compuesta un Docuement. Información sobre los Fields la podemos encontrar en `http://lucene.apache.org/core/7_1_0/core/org/apache/lucene/document/Field.html`.

Un Field tiene tres componentes, el nombre, el tipo y el valor. En nombre hace referencia la nombre del campo (equivaldría al nombre de una columna en una tabla). El valor hacer referencia al contenido del campo (la celda de la tabla) y puede ser texto (String, Reader o un TokenStream ya analizado), binario o numérico. El tipo determina como el campo es tratado, por ejemplo indicar si se almacena (store) en el índice, lo que permitirá devolver la información asociada en tiempo de consulta, o si se tokeniza.

Para simplificar un poco la tarea, Lucene dispone de tipos predefinidos

- TextField: Reader o String indexado y tokenizado, sin term-vector. Puede ser utilizado para todo el cuerpo del documento
- StringField: Un campo String que se indexa como un único token. Por ejemplo, se puede utilizar para ID de un producto, el path donde se encuentra el archivo, etc. Si queremos que la salida de una búsqueda pueda ser ordenada según este campo, se tiene que añadir otro campo del tipo SortedDocValuesField.
- IntPoint: Entero indexado para búsquedas exactas o por rango. Si queremos devolverlo en consultas, lo debemos de añadir como StoredField
- LongPoint: Long indexado para búsquedas exactas o por rango. Si queremos devolverlo en consultas, lo debemos de añadir como StoredField

- FloatPoint: float indexado para búsquedas exactas o por rango. Si queremos devolverlo en consultas, lo debemos de añadir como StoredField
- DoublePoint: double indexado para búsquedas exactas o por rango. Si queremos devolverlo en consultas, lo debemos de añadir como StoredField.
- SortedDocValuesField: byte[] indexados con el objetivo de permitir la ordenación o el uso de facetas por el campo
- SortedSetDocValuesField: Permite añadir un conjunto de valores, SortedSet, al campo para su uso en facetas, agrupaciones o joinings.
- NumericDocValuesField: Field que almacena a valor long por documento con el fin de utilizarlo para ordenación, facetas o el propio cálculo de scoeres.
- SortedNumericDocValuedFiled: Añade un conjunto de valores numéricos, SortedSet, al campo
- StoredField: Valores almacenados que solo se utilizan para ser devueltos en las búsquedas.

Los distintos campos de un documento se añaden con el método add.

```
1 Document doc = new Document();
2
3 doc.add(new StringField("isbn", "978-0071809252", Field.Store.
    YES));
4 doc.add(new TextField("titulo", "Java: A Beginner's Guide,
    Sixth Edition",Field.Store.YES)); // por defecto no se
    almacena
5 doc.add(new TextField("contenido", "Fully updated for Java
    Platform, Standard Edition 8 (Java SE 8), Java ....."));
6 doc.add(new IntPoint("size", 148));
7 doc.add(new StoredField("size",148));
8 doc.add(new SortedSetDocValuesField("format", new BytesRef("
    paperback")));
9 doc.add(new SortedSetDocValuesField("format", new BytesRef("
    kindle")));
```

```
10
11 Date date = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'").
       parse(...);
12 doc.add(new LongPoint("Date", date.getTime()));
```

Como podemos imaginar, para cada tipo tenemos un comportamiento especifico, aunque nosotros podremos crear nuestro propio tipo de campo.

```
1
2  FieldType authorType = new FieldType();
3  authorType.setIndexOptions(IndexOptions.DOCS_AND_FREQS);
4  authorType.setStored(true);
5  authorType.setOmitNorms(true);
6  authorType.setTokenized(false);
7  authorType.setStoreTermVectors(true);
8
9  doc.add(new Field("author", "Arnaud Cogoluegnes", authorType));
10 doc.add(new Field("author", "Thierry Templier", authorType));
11 doc.add(new Field("author", "Gary Gregory", authorType));
```

5. Ejercicios

1. Basándonos en el código anterior, implementar un pequeño programa que nos permite añadir varios documentos a un índice Lucene, podemos seguir el siguiente esquema.

```
1
2  import org.apache.lucene.analysis.Analyzer;
3  import org.apache.lucene.analysis.standard.StandardAnalyzer
       ;
4  .....
5
6  public class IndiceSimple {
7
8   String indexPath = "./index";
9   String docPath = "./DataSet";
10
11  boolean create = true;
12
```

```
13   private IndexWriter writer;
14
15 public static void main(String[] args) {
16     // Analizador a utilizar
17     Analyzer analyzer = new StandardAnalyzer();
18     // Medida de Similitud (modelo de recuperacion) por
           defecto BM25
19     Similarity similarity = new ClassicSimilarity();
20     // Llamados al constructor con los parametros
21     IndiceSimple baseline = new IndiceSimple( ... );
22
23     // Creamos el indice
24     baseline.configurarIndice(analyzer, similarity);
25     // Insertar los documentos
26     baseline.indexarDocumentos();
27     // Cerramos el indice
28     baseline.close();
29
30
31  }
32
33 public void configurarIndice(Analyzer analyzer, Similarity
       similarity) throws IOException {
34
35
36       IndexWriterConfig iwc = new IndexWriterConfig(analyzer
             );
37       iwc.setSimilarity(similarity);
38       // Crear un nuevo indice cada vez que se ejecute
39       iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
40       // Para insertar documentos a un indice existente
41       // iwc.setOpenMode(IndexWriterConfig.OpenMode.
             CREATE_OR_APPEND);
42
43
44
45       // Localizacion del indice
46       Directory dir= FSDirectory.open(Paths.get(indexPath));
47
```

```
48        // Creamos el indice
49        writer = new IndexWriter(dir, iwc);
50
51
52    }
53
54    public void indexarDocumentos() {
55
56     // Para cada uno de los documentos  a insertar
57     for (elementos d : docPath) {
58
59        //leemos el documento sobre un string
60        String cadena = leerDocumento( d );
61        // creamos el documento Lucene
62        Document doc = new Document();
63
64        //parseamos la cadena (si es necesario)
65        Integer start,end; // Posiciones inicio,fin
66
67        // Obtener campo Entero de cadena
68        Integer start = ... // Posicion de inicio del campo;
69        Integer end =  ... // Posicion  fin del campo;
70        String aux = cadena.substring(start, end);
71        Integer valor = Integer.decode(aux);
72
73        // Almacenamos en el campo en el documento Lucene
74        doc.add(new IntPoint("ID", valor));
75        doc.add(new StoredField("ID", valor));
76
77         // Obtener campo texto de cadena
78        start = ... // Posicion de inicio del campo;
79        end =  ... // Posicion  fin del campo;
80        String cuerpo = cadena.substring(start,end);
81        // Almacenamos en el campo en el documento Lucene
82        doc.add(new TextField("Body", body, Field.Store.YES));
83
84        //Obtenemos los siguientes campos
85
86        // ......
```

```
87
88          // Insertamos el documento Lucene en el indice
89          writer.addDocument(doc);
90          // Si lo que queremos es actualizar el documento
91          // writer.updateDocument(new Term("ID", valor.toString
                ()), doc);
92        }
93
94      }
95
96
97      public void close() {
98        try {
99          writer.commit();
100         writer.close();
101       } catch (IOException e) {
102         System.out.println("Error closing the index.");
103       }
104     }
105
106   }
```

2. Utilizar Luke para ver el índice y realizar distintas consultas sobre el mismo.

6. Práctica Lucene: Indexación

Este práctica tiene dos partes, la primera que es la que hemos visto, corresponde a la indexación y una segunda que corresponde a las necesidades de búsqueda del usuario. En cualquier caso, el proceso de diseño de la misma debe ser inverso. En primer lugar plantearnos las capacidades de búsqueda que queremos dotar a la página que son las que nos dirán que es lo que debemos de indexar de un documento.

6.1. Selección del analizador

El proceso de análisis nos permite identificar qué elementos (términos) serán utilizados en la búsqueda y cuales no (por ejemplo mediante el uso de palabras vacías)

Las operaciones que ejecuta un analizador incluyen: extracción de tokens, supresión de signos de puntuación, acentos o palabras comunes, conversión a minúsculas (normalización), stemización. Para ello, debemos de seleccionar de entre los tipos que tiene Lucene implementados, el que se considere adecuado para nuestra aplicación, justificando nuestra decisión. Este es un criterio importante para poder alcanzar los resultados óptimos en la búsqueda. En esta práctica será necesario utilizar un analizador distinto al por defecto para algunos de los posibles campos a indexar, para ello podemos utilizar un `PerFieldAnalyzerWrapper` como indica el siguiente ejemplo.

```
1 // map field-name to analyzer
2 Map<String, Analyzer> analyzerPerField = new HashMap<String,
    Analyzer>();
3 analyzerPerField.put("uncampo", new StandardAnalyzer());
4 analyzerPerField.put("otrocampo", new EnglishAnalyzer());
5
6 // create a per-field analyzer wrapper using the Whitespace as
    .. default analyzer ;)
7 PerFieldAnalyzerWrapper analyzer = new PerFieldAnalyzerWrapper(
    new WhitespaceAnalyzer(), analyzerPerField);
```

6.2. Uso de Facetas

Se deberá realizar la búsqueda por facetas. Para ello deberá identificar los campos por los que podrá clasificar los documentos. Así, como resultado de la búsqueda, podremos tener los resultados agrupados por categorías, permitiendo al usuario bucear por ellas en busca de la información de su interés. La documentación más extensa sobre esta parte la veremos en el capítulo C de la práctica.