PRÁCTICA III:
Implementación de un Sistema de Recuperación de Información utilizando Lucene
Parte A: Indexación

2 de noviembre de 2017

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, el alumno debe escoger el dominio del problema entre las dos opciones que se le dan, buscador para páginas web de la UGR o buscador para artículos científicos, y desarrollar dos aplicaciones separadas:

- La primera, será un programa que podrá ejecutarse en línea de comandos y será el encargado de realizar el índice.
- La segunda, una aplicación de escritorio sobre la que se podrán ejecutar consultar al sistema y que mostrará los documentos relevantes.

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

2. Diseñando el Sistema de Recuperación de Información

A la hora de diseñar cualquier aplicación de búsqueda, el primer paso es analizar qué tipo de información se va a buscar y cómo se realizan las búsquedas por parte de un usuario. Así, si el objetivo es buscar bibliografía sobre un determinado, por ejemplo “Sentiment Analysis”, podemos necesitar conocer el nombre del trabajo y el revista donde se publicó. En esta práctica nos propondremos crear un buscador sobre páginas webs o documentos científicos. Cada grupo debe seleccionar que dominio del problema desea tratar, aunque para explicar esta práctica asumiremos por simplicidad que los datos provienen de documentos científicos.

En cualquier aplicación de búsqueda podemos distinguir tres pasos, que detallaremos a continuación:

1. Adquisición y tratamiento de los datos, que ya hemos considerado en parte en prácticas anteriores.

2. Indexación y almacenamiento de los mismos.

3. Búsqueda sobre el índice y muestra de los resultados.

3. Adquisición de los datos

Aunque no sería del todo necesario, pues podríamos crear el índice obteniendo la información que necesitemos online, sin embargo, para realizar esta práctica se recomienda descargarnos en nuestro ordenador los datos. Esto nos facilitará las tareas de programación ya que no dependemos de conexiones a internet ni saturaremos los sitios con múltiples peticiones durante el proceso de depuración de nuestro software. En esta sección veremos como podremos crear nuestras bases de datos.

3.1. Descargas de un sitio web (UGR)

Para descargarnos en nuestro ordenador un sitio web completo, o mejor dicho la parte del mismo sobre la que queremos trabajar, podemos utilizar herramientas como **WebCopy** (`https://www.cyotek.com/cyotek-webcopy`) en Windows o **HTTrack** (`https://www.httrack.com/`) en Windows o Linux. En ambas herramientas es posible utilizar un conjunto de reglas para controlar la descarga.

En la página de la asignatura disponemos de una copia de parte del sitio accesible desde `http://www.ugr.es/estudiantes/`, que se puede utilizar para realizar esta práctica.

3.2. Generar base de datos desde Scopus

Para obtener los datos se puede realizar una consulta a la base de datos Scopus sobre una determinada temática y descargarnos los resultados en un fichero CSV con los campos separados por comas, y donde cada línea del fichero representa a

una publicación distinta. En caso de necesidad, si campo contiene el separador coma, sus elementos aparecen entrecomillados. Al realizar la consulta a Scopus sólo podremos descargarnos (en la opción de exportar a csv) algunos campos, teniendo un límite de 2000 publicaciones por descarga (si queremos obtener también el resumen del trabajo).

En concreto, y por razones obvias de homogeneidad en los procesos de lectura, en la práctica sólo consideraremos los siguientes campos que para todas las consultas que queramos hacer.

- Autor, Título, Año de publicación, fuente de la publicación, número de citas, enlace, resumen (abstract), palabras clave del autor, palabras claves indexadas, EID.

En este caso, una línea del fichero ante la consulta "Sentiment Analysis" es la siguiente.

- "Taboada M., Brooke J., Tofiloski M., Voll K., Stede M.",Lexicon-basedmethods for sentiment analysis,2011,Computational Linguistics,623, https://www.scopus.com/inward/..., "We present a lexicon-based approach to extracting sentiment from text. The Semantic Orientation CALculator (SO-CAL) uses dictionaries of words annotated with their semantic orientation (polarity and strength), and incorporates intensification and negation. SO-CAL is applied to the polarity classification task, the process of assigning a positive or negative label to a text that captures the text's opinion towards its main subjectmatter.We show that SO-CAL's performance is consistent across domains and on completely unseen data. Additionally, we describe the process of dictionary creation, and our use of Mechanical Turk to check dictionaries for consistency and reliability. © 2011 Association for Computational Linguistics.",,2-s2.0-79958257877

Como vemos, esta línea del fichero CSV tiene toda la información que necesitamos sobre el documento (publicación a indexar). En la página web de la asignatura también se encuentra un fichero de prueba, aunque en nuestra aplicación debemos suponer que tenemos múltiples ficheros CSV con el mismo formato.

3.3. Procesado de documentos

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 esta fase de análisis de requisitos se considerarán las necesidades de los hipotéticos usuarios finales de la aplicación para determinar qué objetivos se deben cubrir. En este sentido se debe proporcionar la documentación de la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos. La consulta termina cuando Lucene devuelve una lista de documentos, ordenados por relevancia. Esta salida de Lucene debe ser transformada para que el usuario del buscador pueda comprenderla y utilizarla de forma sencilla. Esta lista se genera a partir de la lista interna de identificadores de documentos ordenados que concuerdan con una consulta.

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 EID o fuente para Scopus, o la dirección física de la página para UGR.

- Secuencia de términos que es procesada para la indexación, esto es, convertida a minúscula, tokenizada, stemizada, etc. En el caso de Scopus, el título, abstract, fuente, etc. En el caso de UGR el body de la página, el title de la página.

- Numérico. En Scopus el número de citas o el año, en la UGR podremos asumir como tal el número de caracteres de la página

- Facetas (Categorías). En el caso de Scopus el año o las keywords y en el caso de UGR podrían ser las keywords de la página o bien extraerlas de la propia URL.

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.

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.

4. 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.

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  ....
7
8  List<Document> docs = ObtenerDocs(fichero);
9  for (Document d:docs)
10         writer.addDocument(d);
11
12 writer.commit(); // Ejecuta todos los cambios pendientes en el
       ndice
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, indicar criterios de mezcla de índices, etc.)

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

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

5. 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")));
```

Como podemos imaginar, para cada tipo tenemos un comportamiento específico, 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));
```

6. Ejercicios

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

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

7. 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.

7.1. Diseño de la interfaz de búsqueda

El primer paso, debe ser la identificación de los campos que serán recuperables. Obviamente, para este proceso es conveniente el diseñar previamente la propia interfaz de búsqueda donde se especifique qué y cómo se desea buscar.

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

- StringField: Texto simple que se considera literalmente (no se tokenika), útil para la búsqueda por facetas, filtrado de consultas y también para la presentación de resultados en la interfaz de búsqueda.

- TextField: Secuencia de términos que es procesada para la indexación, esto es, convertida a minúscula, tokenizada, stemizada, etc.

- Numérico

- Facetas (Categorías)

Además de una consulta por texto libre, en la aplicación se deberá poder realizar como mínimo una consulta booleana que involucre a los operadores lógicos OR, AND o NOT a la misma. Además deberemos proporcionar algún tipo de consulta avanzada como por ejemplo las consultas por proximidad, así como permitir presentar la información utilizando distintos criterios de ordenación (esto es, además de presentar los elementos ordenados por relevancia, debemos de poder presentarlo utilizando un orden distinto).

7.2. 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. Puede que sea necesario el utilizar un analizador distinto para cada uno de los posibles campos a indexar.

7.3. 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.

8. Fecha de entrega y defensa de la práctica

La fecha de entrega de la parte de la práctica relativa a indexación, junto con su documentación asociada será el Miércoles 23 de Noviembre de 2017. La defensa se realizará los siguientes días de clase.

8.1. Documentación a entregar

- Análisis previo de los requisitos
- Diseño de la solución.
- Manual de Usuario
- Presentación. En este documento el alumno presentará en clase, en formato PowerPoint y con una duración de 15 minutos como máximo, la aplicación realizada.