

APÉNDICE A: INDEXACIÓN DE INFORMACIÓN, LUCENE

1. INTRODUCCIÓN: INDEXACIÓN DE INFORMACIÓN

El desarrollo y crecimiento masivo de las redes de computadoras y medios de almacenamiento a lo largo de los últimos años, ha motivado la aparición de un creciente interés por los sistemas de clasificación automática de documentos. Estos sistemas realizan diferentes operaciones de clasificación basándose en el análisis del contenido del texto de los documentos que procesan. La mayoría de las técnicas de análisis y representación de documentos utilizadas en la actualidad en los sistemas de clasificación, se basan en criterios fundamentalmente estadísticos, centrados en frecuencias de aparición de términos en documentos.

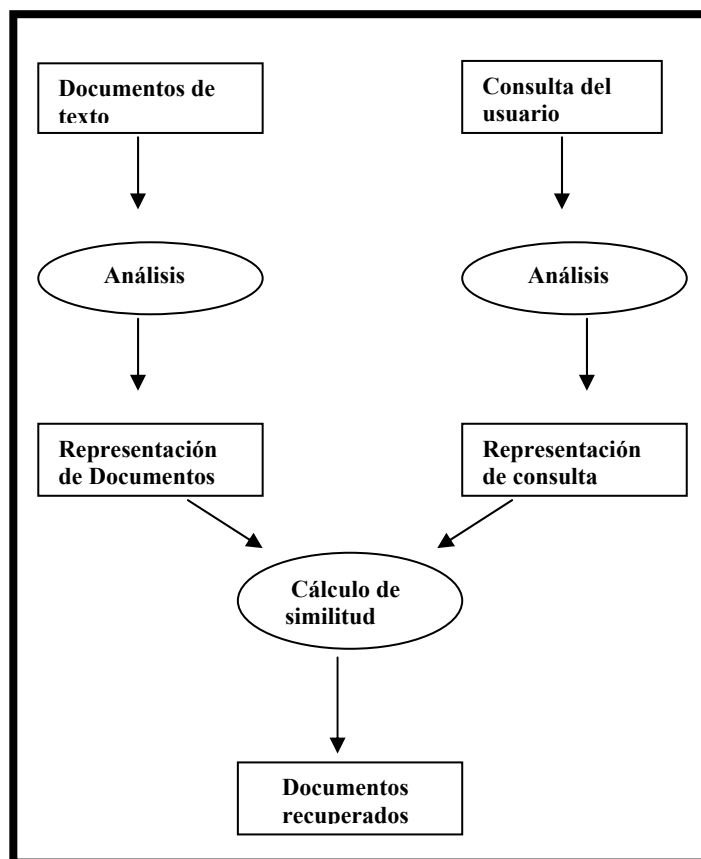
Dentro de los sistemas de clasificación de documentos podemos distinguir los **sistemas de recuperación de texto**, que seleccionan aquellos textos o documentos que son adecuados a una necesidad del usuario entre un conjunto más amplio, y **sistemas de agrupamiento de textos** que, a partir de un conjunto de textos, construyen subconjuntos de éstos con contenidos semejantes.

Si trabajamos con este tipo de sistemas, debemos centrarnos en tres cuestiones fundamentales:

- ✓ **Método de representación:** forma en que serán representados internamente los documentos.
- ✓ **Método de análisis:** proceso que permitirá obtener las representaciones concretas de los documentos a partir del análisis de su contenido.

- ✓ **Método de cálculo de similitud:** entre las representaciones de los documentos y la representación del otro elemento de la operación de clasificación.

En la figura que se muestra a continuación se representa la operación de recuperación de textos centrada en las tres cuestiones descritas. En ella se puede observar que, en este caso, el elemento específico de la operación de clasificación es una consulta de usuario; es decir, que los documentos que se recuperarían serían aquellos cuya representación interna presentase una mayor similitud con la de la consulta del usuario.



Una vez introducida la recuperación de textos, diremos que el término de **indexación** hace referencia a un método que engloba la definición de uno de representación y uno de análisis. El término **proceso de indexación** hace referencia al proceso de análisis de documentos para la obtención de una representación concreta de los mismos.

Existe una serie de elementos que se pueden utilizar para la definición de métodos de indexación y cálculo de similitud. En concreto, **el modelo del espacio vectorial** proporciona las bases para definir un método de representación y cálculo de similitud.

En el modelo del espacio vectorial se propone la representación de cada documento mediante un vector cuyos componentes son los pesos asociados a los términos utilizados en la representación.

Para realizar el proceso de indexación utilizando una representación basada en el espacio vectorial se pueden utilizar los siguientes elementos, que permiten obtener la representación interna de los documentos mediante un análisis automático de su contenido:

1. Peso de los términos

El concepto de poder de resolución de un término proporciona una base para los métodos de indexación basados en frecuencia de aparición de términos. El poder de resolución de un término proporciona información acerca de su adecuación como término de indexación.

2. Listas de parada

Las listas de parada(stoplists) se utilizan en el análisis de los documentos para la eliminación de una serie de palabras que no resultan útiles para la obtención de términos de indexación, por ejemplo, de, en, el, etc.

3. Extracción de raíces

Los algoritmos de extracción de raíces(stemming), o de eliminación de sufijos, se encuentran orientados a obtener un único término a partir de

diferentes palabras que constituyen, esencialmente, variaciones morfológicas con un mismo significado. El resultado del algoritmo debe ser una misma forma regular para las diferentes variantes morfológicas de una palabra, que no tiene por qué ser, necesariamente, la raíz lingüística.

4. Frases de términos

Las frases de términos se orientan a la obtención de términos de indexación con un significado más preciso que el de los términos obtenidos directamente a partir de las palabras individuales. Una frase de términos es una tupla de términos y constituye en sí misma un nuevo término de indexación.

5. Thesaurus

Un thesaurus proporciona una agrupación o clasificación de términos en un determinado dominio o área en categorías denominadas clases. Permite recuperar documentos que son relevantes a la consulta de un usuario, aunque no aparezcan en ellos los términos de la consulta, pero sí sinónimos de estos.

2. LUCENE

2.1. INTRODUCCIÓN

Lucene es una novedosa herramienta que permite tanto la indexación como la búsqueda de documentos. Creada bajo una metodología orientada a objetos e implementada completamente en Java, no se trata de una aplicación que pueda ser descargada, instalada y ejecutada sino de una API flexible, muy potente y realmente fácil de utilizar, a través de la cual se pueden añadir, con pocos esfuerzos de programación, capacidades de indexación y búsqueda a cualquier sistema que se esté desarrollando.

Originalmente escrita por Doug Cutting, en Septiembre de 2001 pasó a formar parte de la familia de código abierto de la fundación Jakarta. Desde

entonces, debido a su mayor disponibilidad, ha atraído a un gran número de desarrolladores, incluso empresas como Hewlett Packard, FedEx, etc. usan, o al menos lo han evaluado.

Existen otras herramientas, a parte de Lucene, que permiten realizar la indexación y búsqueda de documentos pero dichas herramientas han sido optimizadas para usos concretos, lo que implica que el intentar adaptar dichas herramientas a un proyecto específico sea una tarea realmente difícil. La idea que engloba Lucene es completamente diferente, ya que su principal ventaja es su flexibilidad, que permite su utilización en cualquier sistema que lleve a cabo procesos de indexación.

2.2. CARACTERÍSTICAS

A continuación se detallan algunas características que hacen de Lucene una herramienta flexible y adaptable:

✓ *Indexación incremental vs indexación por lotes.*

El término de indexación por lotes se utiliza para referirse a aquellos procesos de indexación, en los cuales, una vez que ha sido creado el índice para un conjunto de documentos, el intentar añadir algunos documentos nuevos es una tarea difícil por lo que se opta por reindexar todos los documentos de nuevo. Sin embargo en la indexación incremental se pueden añadir documentos a un índice ya creado con anterioridad de forma fácil. Lucene soporta ambos tipos de indexación.

✓ *Origen de datos.*

Muchas herramientas de indexación sólo permiten indexar ficheros o páginas web, lo que supone un serio inconveniente cuando se tiene que indexar contenido almacenado en una base de datos. Lucene permite indexar tanto

documentos y páginas web como el contenido procedente de una base de datos.

✓ *Contenido Etiquetado.*

Algunas herramientas, tratan los documentos como simples flujos de palabras. Pero otras como Lucene permiten dividir el contenido de los documentos en campos y así poder realizar consultas con un mayor contenido semántico. Esto es, se pueden buscar términos en los distintos campos del documento concediéndole más importancia según el campo en el que aparezca. Por ejemplo, si se dividen los documentos en dos campos, título y contenido, puede concederse mayor importancia a aquellos documentos que contengan los términos de la búsqueda en el campo título.

✓ *Técnica de indexación.*

Existen palabras tales como a, unos, el, la ...etc. que añaden poco significado al índice, son palabras poco representativas del documento. Al eliminar estas palabras del índice se reduce considerablemente el tamaño del mismo así como el tiempo de indexación. Estas palabras están contenidas en lo que se denomina lista de parada, que es la técnica de indexación contemplada por Lucene.

✓ *Concurrencia.*

Lucene gestiona que varios usuarios puedan buscar en el índice de forma simultánea así como también que un usuario modifique el índice al mismo tiempo que otro lo consulta.

✓ *Elección del idioma*

Tal y como ya se indicó con anterioridad Lucene trabaja con listas de parada, las cuales son proporcionadas por el desarrollador que está utilizando Lucene, esto permite escoger el idioma a utilizar.

2.3. COMO OBTENER LUCENE

Como la mayoría de los proyectos de Jakarta, Lucene se distribuye en forma de ficheros binarios precompilados o de código fuente. Ambas distribuciones pueden ser descargadas de la página oficial de Jakarta, <http://jakarta.apache.org/lucene>, así como también una demo que permite ver el funcionamiento de Lucene.

2.4. FUNCIONALIDAD BÁSICA

Como ya se ha mencionado anteriormente, en la introducción de este anexo, Lucene es una herramienta que permite tanto la indexación como la búsqueda de documentos. A continuación, y puesto que indexación y búsqueda son dos operaciones muy generales, que abarcan multitud de aspectos, se trata en detalle cada una de ellas.

2.4.1. Indexación De Documentos

Creación de un índice

La creación de un índice constituye el punto de partida para el trabajo con Lucene, puesto que una vez que ha sido creado, se irán añadiendo todos aquellos documentos susceptibles de ser indexados.

El sencillo programa que se muestra a continuación, `CrearIndice.java`, construye un índice vacío, para lo cual simplemente crea un objeto `IndexWriter`.

```
public class CreateIndice {  
  
    public static void main(String[] args) throws Exception {  
        String indexPath = args[0];  
        IndexWriter writer;  
  
        writer = new IndexWriter(indexPath, null, true);  
        writer.close();  
    }  
}
```

La clase `IndexWriter` se utiliza tanto para la creación de índices como para su mantenimiento. Cuando se crea un objeto de esta clase, como se puede observar en el ejemplo, al constructor se le pasan tres parámetros. Puesto que en este caso, lo que nos interesa es la creación del índice, solamente son relevantes el primero y el tercero: el primero representa el path dónde será almacenado el índice(en el ejemplo, se introduce en línea de comandos)y con el valor del tercero establecido a `true`, indicamos que lo que estamos es creando el índice y no abriéndolo para su mantenimiento. El segundo de los parámetros se establece a un valor `null`.

El método `close()` se llama para liberar todos los recursos asociados a la creación del índice.

Añadir documentos a un índice

Una vez que tenemos un índice creado, está listo para empezar a añadirle documentos. La siguiente porción de código, `IndexFile.java`, muestra cómo podemos llevar a cabo esta operación. Para cada uno de los documentos (nombrados en la línea de comandos), se crea un objeto `Document` y posteriormente se llama al método `addDocument` del `IndexWriter` para añadirlo al índice.

```
public class IndexFiles {

    public static void main(String[] args) throws Exception {
        String indexPath = args[0];
        IndexWriter writer;

        writer = new IndexWriter(indexPath, new SimpleAnalyzer(), false);
        for (int i=1; i<args.length; i++) {
            System.out.println("Indexing file " + args[i]);
            InputStream is = new FileInputStream(args[i]);

            Document doc = new Document();
            doc.add(Field.UnIndexed("path", args[i]));
            doc.add(Field.Text("body", (Reader) new InputStreamReader(is)));
            writer.addDocument(doc);
            is.close();
        };

        writer.close();
    }
}
```

Los documentos constituyen la unidad de indexación y búsqueda en Lucene. Un objeto *Document* representa un único documento, modelado como un conjunto de campos (*Fields*) de la forma <nombre,valor>.

Cuando creamos un objeto *Document*, la llamada al constructor, como se puede observar en el ejemplo, no requiere ningún parámetro y construye un nuevo documento sin ningún campo. Una vez creado el documento, para añadirle campos, se utiliza el método **add**, al que pasamos el campo como parámetro.

A la hora de crear un objeto *Field*, tenemos que tener en cuenta una serie de importantes consideraciones, ya que algunas de las decisiones que se tomen en este punto afectarán en los posteriores procesos de búsqueda sobre el índice. Cuando creamos un objeto *Field*, aparte del nombre de dicho campo y de su valor, debemos proporcionar tres valores booleanos adicionales que indican lo siguiente:

- El primero de ellos indica si el campo será indexado para las búsquedas. En determinados casos, puede interesarnos almacenar campos de un documento que no son relevantes para los procesos de búsqueda.
- El segundo de los valores indica si el valor de un campo será “tokenizado” previamente a que sea indexado; es decir, en caso de que este parámetro sea **true** el valor de dicho campo es convertido en una secuencia de “tokens” o trozos de texto.
- Finalmente se puede indicar si se desea que el valor de un campo sea almacenado en el índice. En el caso de que el contenido del campo sea razonablemente pequeño, Lucene permite que éste sea almacenado en el índice. Con los campos almacenados en el índice, en lugar de utilizar el Documento para localizar el fichero o los datos originales, se pueden recuperar directamente del índice.

Una vez construido el documento con los campos deseados, éste se añade al índice a través del método **addDocument** de *IndexWriter*. Esta clase ya ha sido comentada para la operación de creación del índice, pero en este caso se utiliza para la manipulación de uno ya creado. Cuando queremos abrir un índice para añadir documentos, al constructor de *IndexWriter* le pasamos el nombre de dicho índice, un objeto *Analyzer*, y un valor booleano establecido a *false*, para indicar que no estamos creando un índice sino abriendo uno ya existente.

Un objeto *Analyzer* se crea para analizar un texto y en base a un determinado criterio, obtener la representación interna de dicho texto en el índice. Los *Analizadores* preprocesan el texto de entrada convirtiendo éste en una secuencia de tokens y se utilizan tanto al añadir un documento a un índice como en los procesos de búsqueda, puesto que el texto o criterio de búsqueda debe de ser procesado de la misma manera que el contenido de los campos del documento cuando éste es añadido al índice.

Uno de los criterios de análisis más utilizados es el de la lista de parada, que consiste en eliminar del texto una serie de palabras que no resultan útiles para la obtención de términos tanto de indexación como de búsqueda, como por ejemplo, de, en, el, etc. En este caso, el objeto que se pasa al constructor de *IndexWriter*, es un objeto *StopAnalyzer*, subclase de *Analyzer*.

Cualquier aplicación que haga uso de Lucene debe proporcionar los datos que van a ser indexados bien como un *String* o bien como *InputStream*. Es por esta razón que Lucene permite la indexación de datos, no sólo de ficheros, sino de cualquier fuente (bd, etc.) En el caso de que los documentos se encuentren almacenados en ficheros, se utiliza *FileInputStream* para recuperarlos; En caso de que se encuentren almacenados en una base de datos, se hace uso de *InputStream* y de manera similar, se puede recuperar contenido HTML, por ejemplo, utilizando *FilterInputStream*, que permite la eliminación de etiquetas.

2.4.2. Búsqueda De Documentos

La búsqueda de documentos constituye la funcionalidad principal proporcionada por Lucene. Para ello aporta múltiples clases y métodos para la representación de consultas y para buscar en el índice aquellos documentos que son relevantes y cumplen con los criterios de la búsqueda. El código que se muestra a continuación, `Search.java`, es un sencillo ejemplo de cómo buscar en un índice.

```
public class Search {
    public static void main(String[] args) throws Exception {
        String indexPath = args[0], queryString = args[1];

        Searcher searcher = new IndexSearcher(indexPath);
        Query query = QueryParser.parse(queryString, "body",
            new SimpleAnalyzer());
        Hits hits = searcher.search(query);

        for (int i=0; i<hits.length(); i++) {
            System.out.println(hits.doc(i).get("path") + "; Score: " +
                hits.score(i));
        }
    }
}
```

Las tres clases más importantes que proporcionan métodos relacionados con la búsqueda de documentos indexados son *Search*, *Hits* y *Query*(con sus respectivas subclasses)

La clase *Searcher* es una clase abstracta y constituye la base para cualquier búsqueda en un índice de documentos. Declara métodos que son implementados por sus subclases, como por ejemplo, *IndexSearcher*, proporcionando así la forma de acceder y recuperar información indexada.

En el caso de *Search.java*, se crea un objeto de este tipo, pasando al constructor un único parámetro, que no es otro que el path dónde se encuentra el índice de documentos.

El método más importante de esta clase es el método **search()**, que devuelve todos aquellos documentos que cumplen con las condiciones de la búsqueda. Su sintaxis es la siguiente:

Hits search (Query query)

en la cual se corrobora que junto con *Searcher*, a la hora de realizar búsquedas de documentos en un índice destacan *Hits* y *Query*.

La clase *Query*, al igual que *Searcher* es una clase abstracta siendo *QueryParser* su subclase más importante, de la cual el método que más funcionalidad proporciona a la búsqueda de documentos es **parse()**. Este método partiendo de una cadena de entrada, del campo del documento dónde se desea realizar la búsqueda y de un analizador⁷, obtiene como parámetro de salida una *Query*. Una *Query* está formada por una serie de cláusulas de la forma:

1. Toda cláusula debe ir precedida de:
 - Un símbolo más(+) o un símbolo menos(-) indicando si la cláusula es requerida o es rechazada, o
 - Un termino seguido de una coma, indicando el campo dónde se va a realizar la búsqueda, lo cual permite la construcción de *Querys* que implementen búsquedas en varios campos de un documento indexado.
2. Además de:

⁷ Los analizadores ya han sido explicados en el apartado anterior.

- Un término, indicando todos los documentos que contienen a dicho término, o
- Una Query anidada, encerrada entre paréntesis.

En formato BNF, la gramática que resume esto, sería:

Query ::= (Clause)*

Clause ::= ["+", "-"] [<TERM> ":"] (<TERM> | "(" Query ")")

Una vez que hemos obtenido una Query, pasada como argumento al método **search()**, éste devuelve un objeto *Hits*, que representa una colección ordenada de documentos (podemos pensar en un Hits como en un Vector, ordenado, claro). El orden viene determinado por la relevancia de cada documento en la búsqueda implementada por la Query.

De esta manera habremos obtenido aquellos documentos que cumplen con las expectativas de la búsqueda introducida.