PRÁCTICA II: Indexación de Textos

Parte I. Tratamiento de textos

October 7, 2015

1 Objetivo

El objetivo de la práctica es conocer los procesos claves en la creación de un índice para Recuperación de Información. Se debe trabajar en grupos de dos personas. En particular, nos centraremos en las primerasd fases de un proceso de indexación que nos permitirán analizar los documentos a indexar para extraer finalmente los tokens de indexación.

La práctica se realizará en grupo, si tienes algún problema en la formación del grupo puedes enviarme un correo para poder indicarte algúon compañero que también este buscando grupo.

Al final de la práctica se debe entregar un informe que necesarimente debe incluir una sección denominada *Trabajo en Grupo* en el que se indicará de forma clara la contribución de cada alumno. Se recomienda seguir la metodología SCRUM para el desarrollo del proyecto.

1.1 Colecciones de documentos

Para esta práctica, el alumno dispondrá de tres colecciones distintas:

- **Cervantes**: Esta colección, en texto html que contiene las obras completas de Cervantes.
- **ParlamentoXML:** Será una colección XML con información de los Diarios de Sesiones de Parlamento de Andalucía.

- **ParlamentoPDF:** La colección de los Diarios de Sesiones de Parlamento de Andalucía en PDF.

2 Tareas a realizar

Como entrada, tomaremos todos los documentos de un directorio (será nuestra colección de documentos) y los procesaremos de forma que seamos capaces de identificar los distintos tokens (una vez eliminados signos de puntuación) que pasarán al proceso de indexación. En esta práctica utilizaremos dos herramientas, Snowball para hacer el stemming y Tika que nos ayudará a extraer la información relevante de los documentos (ver secciones 4 y 5, respectivamente, de este documento).

Las tareas a realizar son tres, la primera en verificar que se cumple la Ley de Zipf en nuestras colecciones, y la segunda es construir un índice (en memoria) y la tercera utilizar dicho índice para resolver pequeñas consultas.

2.1 Ley de Zipf

El alumno deberá realizar un programa que lea todos los documentos de la colección indicada y nos de como salida para cada término (incluyendo palabras vacías) el número de ocurrencias de la misma, esto es su frecuencia. Una vez obtenida dicha salida, debemos hacer un gráfico donde se presenten en el eje de las X los términos ordenados en orden decreciente de frecuencia y el el eje de las Y la frecuencia de los mismos. De igual forma se presentará el gráfico log-log. .

Una versión más actualizada de la Ley de Zipf, viene dada por la ecuación de Booth y Federowicz, esto es,

$$F = \frac{k}{R^m}$$

donde F representa la frecuencia, R la posición en el ranking (ordenación) y k y m son constantes. Para obtener dichas constantes podemos hacerlo a partir del gráfico log-log teniendo en cuenta que

$$\ln(F) = \ln(\frac{k}{R^m}) = \ln(k) - m\ln(R)$$

Por tanto, si realizamos sobre el gráfico log-log un ajuste lineal, podremos obtener dichas constantes k y m de forma sencilla.

Se pide determinar dichas constantes para las distintas colecciones que se entregan, así como proporcinar distintos estadísticos sobre la ocurrencia de los términos en el documento (número de términos, frecuencia máxima, mínima. media, mediana, desviación típica).

Para crear una lista con todos los ficheros que cuelgan un directorio, podremos utilizar el siguiente código

```
1   ArrayList<File> listaFicheros = new ArrayList<File>();
2
3   private void addFiles(File file) {
4
5     if (!file.exists()) {
6       System.out.println(file + " no existe.");
7     } else if (file.isDirectory()) {
8         for (File f : file.listFiles()) {
9            addFiles(f);
10        }
11    } else {
12       queue.add(file);
13    }
14 }
```

2.2 Construcción del índice

En la segunda parte de la práctica se pide la creación de un índice invertido. Importante, en el proceso eliminaremos las palabras vacías (en `http://decsai.ugr.es` podeis encontrar dos ficheros con las palabras vacías en castellano e inglés). Para ello serán necesario obtener para cada término los pesos $tf \times idf$. Como las colecciones son de tamaño relativamente pequeño podremos realizar el proceso de indexación en memoria, pero siempre utilizando las estructuras de datos que se considere más conveniente (diccionarios o tablas hash).

En el indice debemos ser capaces de obtener para cada término una lista con los documentos en los que aparece, así como la frecuencia de este dentro del mismo. Podemos utilizar estructuras auxiliares para almacenar los datos asociados

a la colección (valores idf de cada término). En concreto deberemos tener las siguientes estructuras.

1. Vocabulario: Tabla Hash que asocie un término con su ID, tID,
2. Documentos: Tabla Hash que asocie un documento con su ID, dID
3. IDF: Tabla Hash que asocie un tID con su idf
4. NormaDoc: Tabla Hash que asocie un dID con su factor de normalización (el $\max_{t \in D} tf(D)$.
5. Indice: Tabla Hash que asocie un tID con el conjunto ORDENADO de documentos en los que aparece, así como el tf del término en el mismo, esto es pares $< dID, tf >$

2.2.1 Eficiencia

Se pìde evaluar el tiempo necesario para las Construcción del índice.

2.3 Consultas Simple

Una vez realizado el índice debemos de permitir devolver los documentos más similares (nos restringiremos a los 20 más relevantes) a consultas que:

1. Tienen un único término
2. Consultas que involucran al menos dos términos.

2.3.1 Eficiencia

Una vez realizada la consulta, evaluar el tiempo necesario para poder dar como salida la lista de documentos relevantes. Para ello, podremos lanzar una batería de consultas aleatorias (obteniendo los términos del vocabulario) calculando el tiempo promedio en realizar las consultas.

3 A entregar

Debeis entregar un informe, en pdf que incluya

1. Trabajo en Grupo: Detalles de la colaboración en el grupo
2. Documentación del software, incluyendo entre otras
   (a) Arquitectura del sistema
   (b) Estructuras de datos empleadas.
   (c) Limitaciones (si las tiene).
   (d) Instrucciones de uso (en líneas de comandos).
   (e) Código desarrollado (en un fichero zip).

3.1 Fecha de entrega

1. Ley de Zipf: 28 de Octubre
2. Construcción del índice: 18 de Noviembre
3. Consultas: 30 de Noviembre

4 Snowball

Snowball es un pequeño lenguaje de procesamiento de cadenas diseñado para poder reducir una palabra a su raíz (stemming). Así, por ejemplo, palabras como `toreado, toreados, toreándolo,torear, toteara, torearlo` tienen todos la misma raíz `tor`, que será el término que finalmente se indexará. Por tanto, esto nos permite incrementar el número de documentos que se pueden encontrar para la consulta `torear` pues cualquier documento que contenga una palabra con la raíz `tor` será considerado como relevante, incrementando el recall.

Normalmente el proceso de stemming es heurístico, que aplica un conjunto de reglas a la cadena a stemizar. En esta práctica consideraremos Snowball (`http://snowball.tartarus.org/`) como herramienta para realizar el stemming.

Una demo de ejemplo de su uso (en inglés) lo podemos ver en `http://snowball.tartarus.org/demo.php`.

Snowball se puede considerar como un lenguaje que nos permite declarar nuestras propias reglas para realizar el stemming, sin embargo, en esta práctica nos centraremos en como utilizar los algoritmos que ya tiene implementados, que permiten realizar el proceso en lenguajes como el Inglés (implementa el algoritmo de Porter), Ruso, Francés, Español, Italiano, Alemán, etc.

Más detalles de las reglas que emplean los distintos algoritmos, tanto en inglés como en español, los podemos encontrar en

`http://snowball.tartarus.org/algorithms/english/stemmer`

y

`http://snowball.tartarus.org/algorithms/spanish/stemmer.html`,

respectivamente. Si estamos interesados en cómo funcionan dichos algoritmos se recomienda leer la documentación asociada.

4.1 Download

Snowball está disponible en varios lenguajes de programación como C, Java o Phyton y lo podremos descargar de `http://snowball.tartarus.org/download.php`. En nuestro caso, nos centraremos en la versión en Java, pues lo debemos integrar con el resto del software que desarrollemos. Auqnue en la página de la asignatura podemos encontrar el fichero **libstemmer.jar** para incluirlo en el class path de Java, estos son los pasos que deberías seguir para poder generarlo.

- Descarga el fichero tgz.
- Descomprimirlo
- Ve al directorio libstemmer_java y mira el directorio README.
- Sigue las instrucciones para compilarlo (utilizando javac)
- Crear el fichero jar: Ve al directorio libstemmer_java/java y ejecuta

  jar cvf libstemmer.jar *

4.2 Ejemplo de uso

Mostraremos el primer programa en Java para realizar el stemmer. Por ejemplo, desde Netbeans podemos crear una nuevo proyecto que llamaremos testsnowball, añadiéndole el fichero libstemmer.jar dentro de la carpeta de librerias.

```
1 import org.tartarus.snowball.ext.spanishStemmer;
2 ...
3 spanishStemmer stemmer = new spanishStemmer();
4 stemmer.setCurrent("cadenas");
5 if (stemmer.stem()){
6     System.out.println(stemmer.getCurrent());
7 }
```

La primera linea nos permite importar la clase spanishStemmer. setCurrent es un método que nos permite indicar que palabra queremos stemizar, en el ejemplo “cadenas”. Al ejectuar el método stem, se hace la transfomación de la palabra, para dar como salida “caden”.

Hay que notar que si intentamos hacer el stemming de un string, como por ejemplo “estamos trabajando con una secuencia de cadenas”, sólo hará el stemming de la última palabra. Por lo que hará falta tokenizar dicho string. Para ello, se puede utilizar la clase StringTokenizer como indica el siguiente ejemplo

```
1  String texto="estamos trabajando con una secuencia de cadenas";
2  StringTokenizer tokens = new StringTokenizer(texto);
3  SnowballStemmer stemmer;
4
5  stemmer = (SnowballStemmer) new spanishStemmer();
6  while (tokens.hasMoreTokens()){
7        stemmer.setCurrent(tokens.nextToken());
8        if (stemmer.stem()){
9                System.out.println(stemmer.getCurrent());
10       }
11   }
```

La salida de este código será: “estam trabaj con una secuenci de caden”

Notar la diferencia en la declaración del objeto stemmer. Para más información se recomienda consultar la documentación y ejemplos que nos dan en Snowball.

5 Tika

El mundo de los documentos digitales y sus distintos formatos es un mundo donde cada uno tiene un lenguaje diferente (pdf, jpg, gif, png, xls, doc, odt, html, xml, rss, mp3, ...). La mayoría de los programas sólo entienden su propio formato o un conjunto pequeño de formatos relacionados. Sin embargo, en el proceso de desarrollo de un buscador capaz de encontrar cualquier documento debemos de ser capaces de extraer la información sobre el contenido del documento. El primer paso será entender las propiedades de los formatos de ficheros.

El estádar MIME (Multipursose Internet Mail Extension) especifica que el formato de un fichero consiste en un identificador de `tipo/subtipo` y un conjunto opcional de elementos `atributo=valor`. Por ejemplo

- `text/plain; charset=ISO-8859-1`
- `application/msword;`
- `application/pdf; version=1.5`
- `image=jpeg`

Tika nos permite detectar los distintos tipos de ficheros de forma automática así como bucear dentro de los mismos para extraer el contenido así como sus metadata. Los metadatos nos proporcionan información sobre un derterminado fichero, que puede ir desde el tamaño del fichero, localización, autor, versión, etc.)

Ejemplos de metadata son,

```
1  Author: Witten, I. H.; Frank, Eibe.
2  Content-Length: 8138814
3  Content-Type: application/pdf
4  Creation-Date: 2005-09-28T07:50:58Z
5  ISBN: 0120884070
6  Last-Modified: 2005-09-28T07:51:42Z
7  Last-Save-Date: 2005-09-28T07:51:42Z
8  cp:subject: Team DDU
9  created: Wed Sep 28 09:50:58 CEST 2005
10 creator: Witten, I. H.; Frank, Eibe.
```

```
11 ....
```

o bien

```
1 Content-Encoding: ISO-8859-1
2 Content-Length: 3821
3 Content-Type: text/plain; charset=ISO-8859-1
4 X-Parsed-By: org.apache.tika.parser.DefaultParser
5 resourceName: kapins.log
```

De forma genérica, en Tika podemos detectar tres tipos de componentes:

- Parser: Que permite extraer el contenido textual del fichero
- Detección de MIME: Detecta los metadatos
- Detección de idioma: Puede determinar de forma automática si en texto está en castellano, inglés, francés, ...

5.1 Descargar Tika

El código de completo de Tika lo podemos descargar de `http://tika.apache.org` y seguir los pasos allí indicados. En la página web de la asignatura nos podremos descargar el fichero .jar con todo lo necesario para poder trabajar.

Una forma fácil de entender las potencialidades de Tika es utilizar la GUI, ejecutando `java -jar tika-app-1.6.jar -g`, o bien a través de la líne de comandos, ejecutando `java -jar tika-app-1.6.jar -help` nos muestra la lista de las opciones disponibles.

```
1  usage: java -jar tika-app.jar [option...] [file|port...]
2
3  Options:
4      -? or --help          Print this usage message
5      -v or --verbose       Print debug level messages
6      -V or --version       Print the Apache Tika version number
7
8      -g or --gui           Start the Apache Tika GUI
9      -s or --server        Start the Apache Tika server
10     -f or --fork          Use Fork Mode for out-of-process
           extraction
```

```
11
12      -x or --xml             Output XHTML content (default)
13      -h or --html            Output HTML content
14      -t or --text            Output plain text content
15      -T or --text-main       Output plain text content (main
            content only)
16      -m or --metadata        Output only metadata
17      -j or --json            Output metadata in JSON
18      -y or --xmp             Output metadata in XMP
19      -l or --language        Output only language
20      -d or --detect          Detect document type
21      -eX or --encoding=X     Use output encoding X
22      -pX or --password=X     Use document password X
23      -z or --extract         Extract all attachements into current
            directory
24      --extract-dir=<dir>     Specify target directory for -z
25      -r or --pretty-print    For XML and XHTML outputs, adds
            newlines and
26                              whitespace, for better readability
27
28      --create-profile=X
29            Create NGram profile, where X is a profile name
30      --list-parsers
31            List the available document parsers
32      --list-parser-details
33            List the available document parsers and their supported
                mime types
34      --list-parser-details-apt
35            List the available document parsers and their supported
                mime types in apt format.
36      --list-detectors
37            List the available document detectors
38      --list-met-models
39            List the available metadata models, and their supported
                keys
40      --list-supported-types
41            List all known media types and related information
42
43  Description:
```

```
44      Apache Tika will parse the file(s) specified on the
45      command line and output the extracted text content
46      or metadata to standard output.
47
48      Instead of a file name you can also specify the URL
49      of a document to be parsed.
50
51      If no file name or URL is specified (or the special
52      name "-" is used), then the standard input stream
53      is parsed. If no arguments were given and no input
54      data is available, the GUI is started instead.
55
56  - GUI mode
57
58      Use the "--gui" (or "-g") option to start the
59      Apache Tika GUI. You can drag and drop files from
60      a normal file explorer to the GUI window to extract
61      text content and metadata from the files.
62
63  - Server mode
64
65      Use the "--server" (or "-s") option to start the
66      Apache Tika server. The server will listen to the
67      ports you specify as one or more arguments.
```

5.2 Explorando Tika

En este caso, consideramos distintos tipos de ficheros que tengamos en nuestro ordenador, y veremos que es lo que pasa cuando los exploramos con la interfaz de usuario de Tika, donde podremos ver

- Texto formateado: Extrae el texto formateado como XHTML. Con esto podemos ver cómo Tika entiende la estructura del documento. Idealmente, podremos ver el contenido en orden correcto, con elementos como enlaces y cabeceras bien identificados.
- Texto Plano: Texto extraido, sin tener en cuenta la estructura del documento

- Texto estructurado: fuente del texto en XHTML
- Metadatos: Los metadatos del fichero.

Como hemos visto, también lo podemos ejecutar desde línea de comandos, por ejemplo podemos ejecutar

```
java -jar tika-app-1.6.jar < prueba.txt > fichero.xhtml
```

o incluso podemos utilizar una url

```
java -jar tika-app-1.6.jar < http://decsai.ugr.es
```

Si queremos la salida en texto plano podemos hacer

```
java -jar tika-app-1.6.jar -text < http://decsai.ugr.es
```

en este caso, utilizará la codificación por defecto del sistema, pero podemos cambiar la codificación de salida utilizando –encoding.

Si solo estamos interesados en los metadatos, los obtenemos mediante

```
java -jar tika-app-1.6.jar -metadata < http://decsai.ugr.es
```

5.3 Incluir Tika en nuestros programas

Tika proporciona un API que implementa la mayoría de los métodos básicos, sin tener que centrarnos en su codificación. Las clases las podemos encontrar en `org.apache.tika.Tika`. Veamos como podemos implementar un programa para extraer el contenido textual de forma simple.

```
1 import java.io.File;
2 import org.apache.tika.Tika;
3
4 public class EjemploSimple {
5   public static void main(String[] args) throws Exception {
6
7   // Creaamos una instancia de Tika con la configuracion por
        defecto
8   Tika tika = new Tika();
9   // Se parsean los ficheros pasados como argumento y se extrae
        el contenido
```

```
10    for (String file : args) {
11        File f = new File(file);
12
13    // Detectamos el MIME tipo del fichero
14        String type = tika.detect(f);
15        System.out.println(file +":"+type);
16
17     // Extreamos el texto plano en un string
18        String text = tika.parseToString(f);
19        System.out.print(text);
20        }
21    }
22 }
```

para ejecutarlo podemos hacer

```
javac -cp tika-app-1.6.jar EjemploSimple.java
 java -cp tika-app-1.6.jar:.  EjemploSimple documento
```

5.4 Extracción del contenido

Como hemos visto, Tika puede extraer el contenido de un docuemento de forma simple, cuando llamamos al método `parseToString` primero se detecta el MIME del fichero, para después se utiliza el parser correcto para avanzar a través del mismo. Así, si el fichero es un pdf, se utilizará la clase que da soporte al MIME `application/pdf`, esto es, `org.apache.tika.parser.pdf.PDFParser`, conviertiendo el contenido y los metadatos a un formato que entiende Tika.

En este caso, el texto se pasa a un String, con un tamaño limitado, por lo que si el fichero pdf es grande (por ejemplo un libro) no se analizará todo el texto, por lo que se recomienda hacer un parser incremental que hace el parse sobre un Reader, clase abstracta de Java que permite leer streams de caracteres.

```
Reader texto = tika.parser(file)
```

Otra alternativa, que nos da más control es utilizar la clase Parser, `org.apache.tika.parser.Parser`[1], de la que el principal método es `parse`.

[1]Más información la podemos encontrar en `https://tika.apache.org/1.6/parser.html`

```
void parse( InputStream stream, ContentHandler handler,
Metadata metadata, ParseContext context)
throws IOException, SAXException, TikaException;
```

donde

- InputStream representa el stream de bytes del que lee el parser. Es leido pero no cerrado por parser Por tanto, el proceso típico es

```
1  InputStream stream = ...;       // open the stream
2 try {
3     parser.parse(stream, ...); // parse the stream
4 } finally {
5     stream.close();             // close the stream
6 }
```

- ContentHandler el stream de datos es escrito en el handler en formato estructurado, XHTML, permitiendo que Tika, y las aplicaciones donde se utilice, consideren conceptos como cabeceras, enlaces, etc. La estructura de la salida es:

```
1 <html xmlns="http://www.w3.org/1999/xhtml">
2   <head>
3     <title>...</title>
4   </head>
5   <body>
6     ...
7   </body>
8 </html>
```

- Metadata, es un parámetro tanto de entrada como de salida, y es utilizado para representar los metadatos del documento. Así podemos considerar metadatos de entrada de la aplicación como RESOURCE_NAME_KEY (nombre del fichero); CONTENT_TYPE (tipo); TITLE (título),... y también podemos incluir metadatos propios que podemos esperar para este tipo de fichero.

  Por ejemplo, podríamos considerar

```
1    Metadata met = new Metadata();
2
3    parser.parse(is,ch, met,...);
4    String docType = met.get("Content-Type");
```

- ParseContext, es utilizado para modificar el comportamiento de ContentHandler indicando información del contexto concreto sobre el vamos a trabajar, dando un control más preciso para el parser.

5.5 Ejemplos

5.5.1 Análisis genérico

Podemos extraer el texto y los metadatos de forma genérica considerando el seguiente código

```
1    File file = new File("......1/index.xml");
2
3  InputStream is = new FileInputStream(file);
4  Metadata metadata = new Metadata();
5  ContentHandler ch = new BodyContentHandler();
6  ParseContext parseContext = new ParseContext();
7
8
9  AutoDetectParser parser = new AutoDetectParser();
10
11 parser.parse(is, ch, metadata, parseContext);
12
13 //Tenemos el texto
14  System.out.println("ch  " + ch.toString());
15
16 //Tenemos los metadatos
17  for (String name : metadata.names()){
18         String valor = metadata.get(name);
19         if (valor!=null){
20              System.out.println("metadata: "+name + " " + valor);
21         }
22
```

```
23      }
24    System.out.println("la "+metadata.get(Metadata.CONTENT_TYPE));
25  }
```

5.5.2 Trabajando con XML

Los distintos formatos que puede entender Tika los podemos encontrar en `https://tika.apache.org/1.6/formats.html`, así por ejemplo si queremos extraer el texto de un fichero XML, podriamos considerar el `XMLParser`.

```
1  Parser pxml     = new XMLParser();
2  ToXMLContentHandler chxml    = new  ToXMLContentHandler();
3
4  pxml.parse(is, chxml, metadata, parseContext);
5
6  System.out.println("texto   " + chxml.toString());
```

5.5.3 Un ejemplo más completo

En este caso, podemos ver cómo se puede parsear una página web, sacándole distintos tipos de contenido. Para ello se puede utilizar un TeeContentHandler que nos permite agrupar distintos ContentHandler en uno sólo

```
1  import java.io.InputStream;
2  import java.net.URL;
3  import org.apache.tika.metadata.Metadata;
4  import org.apache.tika.parser.ParseContext;
5  import org.apache.tika.parser.html.HtmlParser;
6  import org.apache.tika.sax.BodyContentHandler;
7  import org.apache.tika.sax.LinkContentHandler;
8  import org.apache.tika.sax.TeeContentHandler;
9  import org.apache.tika.sax.ToHTMLContentHandler;
10 import org.xml.sax.ContentHandler;
11
12 public class ParsearHTML {
13
14     public static void main (String args[]) throws Exception {
15         URL url = new URL("http://www.ideal.es");
```

```
16          InputStream input = url.openStream();
17
18          LinkContentHandler linkHandler = new LinkContentHandler
                ();
19          ContentHandler textHandler = new BodyContentHandler();
20          ToHTMLContentHandler toHTMLHandler = new
                ToHTMLContentHandler();
21
22          TeeContentHandler teeHandler = new TeeContentHandler(
                linkHandler, textHandler, toHTMLHandler);
23
24          Metadata metadata = new Metadata();
25          ParseContext parseContext = new ParseContext();
26          HtmlParser parser = new HtmlParser();
27          parser.parse(input, teeHandler, metadata, parseContext);
28          System.out.println("title:\n" + metadata.get("title"));
29          System.out.println("links:\n" + linkHandler.getLinks());
30          System.out.println("text:\n" + textHandler.toString());
31          System.out.println("html:\n" + toHTMLHandler.toString())
                ;
32      }
33  }
```

5.5.4 Identificación del lenguaje

Con Tika podemos identificar de forma simple el lenguaje en que esta escrito un texto, con esto podremos identificar el stemmer a utilizar

```
1   import org.apache.tika.language.LanguageIdentifier;
2   ...
3   public static  String identificaLanguaje(String text) {
4           LanguageIdentifier identifier = new LanguageIdentifier(
                text);
5           return identifier.getLanguage();
6       }
```