Recuperación de Información (2019-2020)
Doble grado en Informática y Matemáticas
Universidad de Granada

Práctica 1: Parser de Documentos con TIKA

Simón López Vico
Miguel Cantarero López
Alberto Jesús Durán López

7 de octubre de 2019

Índice

0. Introducción

En esta práctica veremos cómo extraer información con Tika a partir de un documento dado, independientemente del formato del archivo, paso previo para cualquier proceso de recuperación de información o análisis de textos.
Hemos realizado un programa capaz de extraer información de los archivos que cuelgan de un directorio, pasado como entrada al programa. Las diferentes opciones son:

1. Opción -d

- -d : Se muestra una tabla que contiene el nombre del fichero, tipo, codificación e idioma.

  Primero de todo, tomamos el archivo y usando la función *parse* de tika, parseamos dicho archivo junto a *metada*, variable de tipo *Metadata*, declarada al principio del programa y común para todas las opciones.

  A continuación, detectamos el MIME del archivo, extraemos el texto plano en un string y por último, el idioma, la codificación y el nombre.

```
//Se parsean los ficheros pasados como argumento y se extrae el contenido
for(String f : ficheros){
        File file = new File("./"+args[0]+"/"+f);

        //Parseamos
        tika.parse(file, metadata);
        String contenido = tika.parseToString(file);

        // Detectamos el MIME tipo del fichero
        String type = tika.detect(file);

        //Extraemos el texto plano en un string
        String text = tika.parseToString(file);

        //Extraemos el idioma
        String idioma = identifyLanguage(contenido);

        //Extraemos la codificacion y el nombre
        String cod = metadata.get(Metadata.CONTENT_ENCODING);
        String name = metadata.get(Metadata.RESOURCE_NAME_KEY);
}
```

2. Opción -l

- -l : Se obtienen todos los enlaces extraíbles de cada documento.

  Primero, una vez abierto el archivo a extraer los links, declaramos todas las variables (de tipo *LinkContentHandler, ContentHandler, TeeContentHandler y ParseContext*) necesarias para dicha opción.

  A continuación, detectamos el fichero , proceso previo a parsear. Como para la función *parser* necesitamos una variable de tipo *InputStream*, la declaramos a partir de nuestro fichero *file* y parseamos.
  Por último, extraemos los links de la variable *linkHandler* y los mostramos por pantalla.

```
for(String f : ficheros){
        File file = new File("./"+args[0]+"/"+f);

        //Declaracion de variables necesarias para extraer los links
        LinkContentHandler linkHandler = new LinkContentHandler();
        ContentHandler textHandler = new BodyContentHandler();
        TeeContentHandler teeHandler = new TeeContentHandler(linkHandler);
        ParseContext parseContext = new ParseContext();

        //Creamos variable de tipo InputStream a partir del fichero 'file'
        InputStream target = new FileInputStream(file);

        //Detectamos el fichero. Proceso previo a parsear
        AutoDetectParser parser = new AutoDetectParser();

        //Parseamos
        parser.parse(target,teeHandler,metadata,parseContext);

        List<Link> links=linkHandler.getLinks();
        System.out.println("LINKS EN EL ARCHIVO " +file.getName()+ ":");

        if(links.isEmpty())
                System.out.println("No hay links :(");
        else
                for(Link link:links)
        System.out.println(link);
}
```

3. Opción -t

- -t : Para cada documento se genera un fichero con formato CSV en el que se muestra cada palabra junto a su ocurrencia.

  Comenzaremos extrayendo el texto plano del documento por completo, al cuál aplicaremos la función `split([...])` usando una expresión regular, que creará un vector con todas las palabras del texto. La expresión regular se encargará de reconocer como palabras los elementos del texto plano que estén separados por:

  - Uno o más espacios.
  - Uno o dos puntos, una coma o un punto y coma seguidos de uno más espacios.
  - Paréntesis, interrogaciones, exclamaciones, etc.

  Una vez obtenidas todas las palabras, recorremos el vector creado para contar el número de ocurrencias de cada una de ellas; tras ello, eliminaremos los caracteres erróneos contados[1].

  Finalmente, ordenaremos las palabras por su número de ocurrencias de mayor a menor y las añadiremos a un fichero CSV.

```
for(String f : ficheros){
        File file = new File("./"+args[0]+"/"+f);

        // Parseamos el fichero y obtenemos su texto plano.
        tika.parse(file, metadata);
        String contenido = tika.parseToString(file);

        // Separamos todas las palabras que aparecen en el fichero y las
            anadimos a vector
        String[] todas=contenido.split("\\s+|[\\.\\;\\:\\,]\\s+|[()?!]");

        // Creamos un ArrayList donde guardaremos el numero de veces que se
             repite cada palabra
        ArrayList<PalabraOcurrencias> palaocu=new ArrayList<
            PalabraOcurrencias>();

        //Recorremos todas las palabras. Si la palabra ya se encuentra
            aumentamos el contador
        //de dicha palabra. En casa contraria la introducimos en 'palaocu'
        for(String palabra:todas){
                Boolean encontrado=false;
                int i=0;
                while(i<palaocu.size() && !encontrado){
                        if(palaocu.get(i).palabra.compareTo(palabra)==0)
                                encontrado=true;
                        else
                                i++;
                }
                if(encontrado)
                        palaocu.get(i).ocurrencias++;
                else
                        palaocu.add(new PalabraOcurrencias(palabra, 1));
        }

        // Eliminamos las palabras que no queremos tratar
```

[1] A veces se guardan palabras vacías.

```
        String filename="./CSV";
        if(rminutiles){
                [...]
                filename+="/rminutiles/"+file.getName()+"_inutiles.csv";
        }else{
                [...]
                filename+="/normal/"+file.getName()+".csv";
        }

        // Ordenamos las palabras con su numero de ocurrencias en orden
            descendente
        Collections.sort(palaocu, new SortbyOcurrencias());

        // Anyadimos la informacion al fichero CSV
        String palcsv="";

        for(int i=0; i<palaocu.size(); i++){
                palcsv+=palaocu.get(i).palabra+";"+palaocu.get(i).
                    ocurrencias+"\n";
        }

        PrintWriter writer = new PrintWriter(filename);
        writer.print(palcsv);
        writer.close();
}
```

El resultado obtenido tras ejecutar este código sobre el poema de Homero *"La Odisea"* es el siguiente:

```
de;8120
y;7112
a;6959
que;5599
la;5043
[...]
Ulises;1674
por;1452
[...]
Telemaco;734
[...]
```

Podemos ver que las palabras más repetidas no aportan información importante sobre el texto, por lo que desarrollamos la siguiente opción.

4. Opción -t -rminutiles

Esta opción permite mejorar la nube de palabras que podemos crear a partir del CSV que se obtiene de cada archivo. Actúa de la misma manera que la opción `-t`, pero elimina palabras que no aportan información del tema sobre el que trata el texto (determinantes,artículos, etc). Dichas palabras se extraerán del vector de ocurrencias de cada palabra en el mismo momento que eliminamos caracteres erróneos (visto en la anterior sección). Al ejecutar esta nueva opción sobre *"La Odisea"* obtenemos:

```
Ulises;1674
Telemaco;734
pretendientes;497
Jupiter;478
pues;464
<<;436
todos;403
palacio;396
[...]
```

5. Nube de palabras

Éstas son las dos nubes de palabras obtenidas tras generar los ficheros CSV del texto *"La Odisea"* e introducir el documento CSV en la web `https://wordart.com/create`:

(a) Nube de palabras con la opción `-t` (izquierda) y `-t -rminutiles` (derecha).

6. Compilación

Situados en la carpeta *code*, compilamos con el comando:
javac -cp ../tika-app-1.22.jar p1.java PalabraOcurrencias.java
y ejecutamos con **java -cp ../tika-app-1.22.jar:. p1 documentos -*opción***

Referencias

[1] https://tika.apache.org/