Práctica 1

Recuperación de Información

Amin Kasrou Aouam

UNIVERSIDAD
DE GRANADA

2020-10-29

Índice

Práctica 1

En esta práctica, vamos a obtener información de una serie de documentos usando *Apache Tika*:

1. Obtener los metadatos (nombre , tipo, codificación e idioma)
2. Extraer todos los enlaces que aparecen
3. Generar un fichero con las ocurrencias de cada término, ordenados de forma descendente
4. Generar una gráfica con los términos y su ocurrencia

Instalación

Implementamos la práctica usando *Java* como lenguaje de programación, y *Maven* como herramienta de gestión del proyecto. No es necesario utilizar esta herramienta, pero nos ofrece una mayor reproducibilidad del proyecto.

En el caso de que deseemos instalar fácilmente todas las dependencias, podemos instalar el gestor de paquetes Nix (compatible con Linux, MacOS y WSL)

Usando Nix

1. Instalamos Nix:

```
sh <(curl -L https://nixos.org/nix/install) --daemon
```

1. Cambiamos la ruta al directorio del proyecto:

```
cd P1
```

1. Entramos en el entorno de desarrollo reproducible y aislado:

```
nix-shell
```

Nix se encargará de resolver las dependencias, podemos dirigirnos directamente a la sección de ejecución.

Sin Nix

En el caso de que no deseemos usar Nix, deberemos instalar las siguientes dependencias en nuestro sistema:

- OpenJDK (> 8.0)
- Maven
- Gnuplot

Ejecución

En el caso que deseemos utilizar *Maven*, debemos ejecutar los siguientes comandos:

1. Compilar el proyecto

```
mvn compile
```

1. Ejecutar el proyecto

```
mvn exec:java -Dexec.mainClass="org.RI.P1.AnalyzeDirectory" -Dexec.args
    ="data metadata"
```

Debemos modificar el argumento **metadata** según la salida que deseemos:

- **metadata**: obtenemos la información de los archivos (nombre, codificación, tipo)
- **links**: obtenemos la lista de enlaces de cada archivo
- **frequency**: se guarda la frecuencia de las palabras de cada documento en un archivo (se encuentran en la carpeta output).

Implementación

Tal y como podemos observar en la sección previa de ejecución, hemos optado por añadir un argumento para elegir la acción que deseamos realizar. En caso de que se omita éste, las instrucciones de ejecución aparecen por pantalla.

```
private static void usage() {
    System.out.println("Usage: AnalyzeDirectory <directory> <option>");
    System.out.println("option metadata: shows the filename/file type/
        encoding and language of the files");
    System.out.println("option links: shows all the links contained in
        each file");
    System.out.println("option frequency: saves word frequency to a
        file");
    System.exit(1);
}
```

```java
public static void main(String[] args) throws IOException,
    TikaException, SAXException {
     if (args.length != 2) {
         usage();
     }
     String directory = args[0];
     String action = args[1];
     readFiles(directory);
     chooseAction(action);
}
```

Esta funcionalidad permite al usuario seleccionar la salida que desea, además de disminuir el contenido que aparece por pantalla y el tiempo de ejecución (dado que no computamos acciones innecesarias).

En el caso de la opción *frequency*, guardamos las ocurrencias de los términos en un fichero en el directorio **output** con el formato *filename.dat*. Utilizaremos estos archivos para generar las gráficas.

Mostramos el extacto de uno de los documentos, con carácter ilustrativo:

1075	the
562	and
543	of
543	a
371	in
370	to
295	i
200	that
188	it
180	his

Gráficas

El objetivo final de esta práctica es comprobar si se cumple la ley de Zipf en el conjunto de documentos que analizamos. Para ello vamos a generar gráficas para observar si se cumple, con documentos en distintos idiomas.

En cada gráfica vamos a generar la función generada a partir de los archivos de datos, y la función normalizada mediante log.

Incluimos un script de gnuplot genérico, al cual le pasamos el fichero de datos como argumento.

```
# Variables
FILEDATA = ARG1

# Parameters
set terminal png size 500,500
set title FILEDATA
set xlabel 'word'
set ylabel 'ocurrence'
set multiplot

# Plotting
plot FILEDATA using 1:xtic(2) lt rgb '#6E016B'
set logscale
plot FILEDATA using 1:xtic(2) lt rgb '#41AB5D'
```

Procedemos a automatizar la generación de 3 gráficas, a partir de 3 archivos con idiomas distintos, mediante un *shell script*.

```
#!/bin/sh

gnuplot -c script/plot_datafile.gp "output/Moby Dick; Or, The Whale.dat" >output/plot_MobyDick.png
gnuplot -c script/plot_datafile.gp "output/Szerelem bolondjai.dat" >output/plot_Szerelembolondjai.png
gnuplot -c script/plot_datafile.gp "output/Älykkään ritarin Don Quijote de la Manchan elämänvaiheet.dat" >output/plot_Älykkää nritarinDonQuijotedelaManchanelämänvaiheet.png
```

Ambos scripts se encuentran en el directorio *script*.

Nota: es importante ejecutar el *shell script* desde el directorio raíz del proyecto (donde se encuentra el archivo shell.nix).

Con fines ilustrativos, mostramos la gráfica correspondiente al libro del Quijote en finlandés:

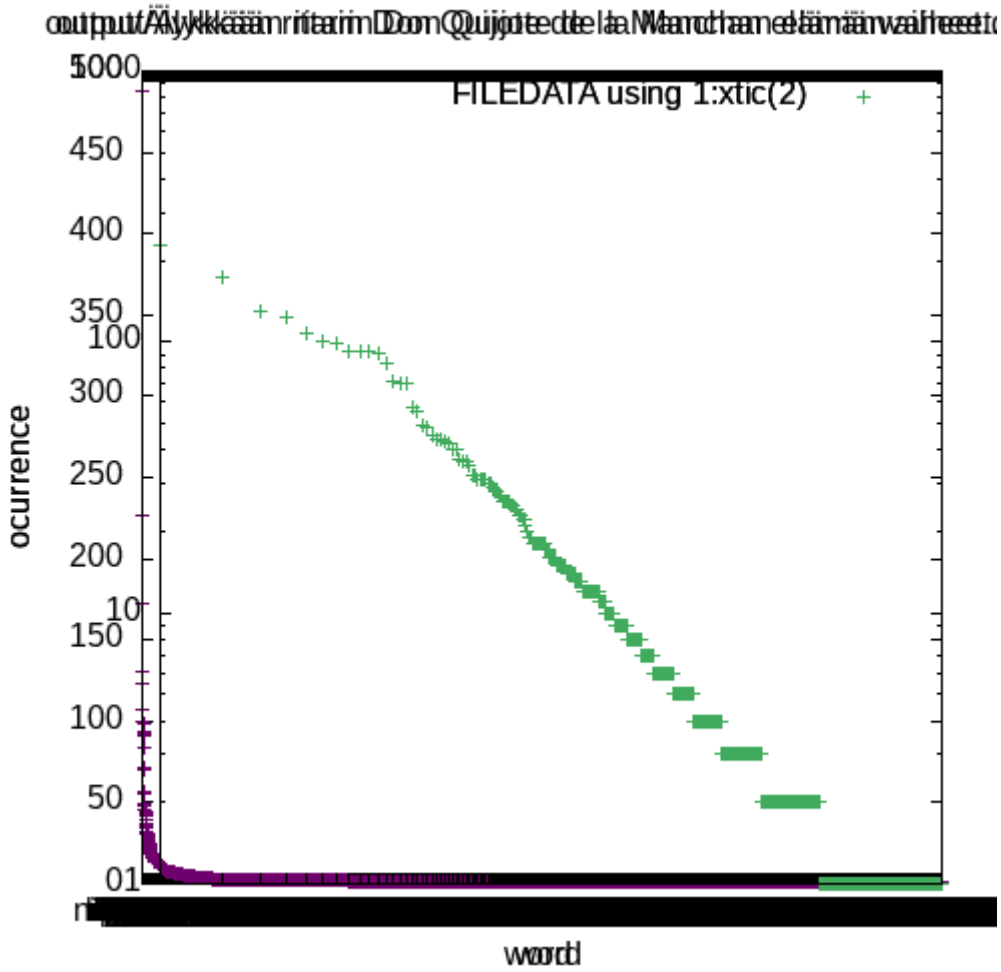

Figura 1: Gráfica de Älykkään ritarin Don Quijote de la Manchan elämänvaiheet

La función en color violeta corresponde a la generación de la gráfica sin alteraciones, mientras que la función en color verde se encuentra en escala logarítmica.

Conclusiones

Hemos analizado documentos, de distinto formato y en idiomas distintos, y según los datos empíricos obtenidos observamos que la ley de Zipf se cumple.