Trabajo TP6 de Fundamentos de Informática

Primer Curso Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2021-2022

Escuela de Ingeniería y Arquitectura

Departamento de Informática e Ingeniería de Sistemas

Área de Lenguajes y Sistemas Informáticos

v 1.0 Octubre 2021

Análisis de redes sociales

Objetivo del trabajo

El objetivo de este trabajo es realizar un programa en el que puedas aplicar todos los conocimientos y habilidades adquiridas durante el curso. En esta ocasión, el dominio del problema es el análisis de los contenidos publicados en la Redes Sociales, concretamente, de los mensajes publicados en Twitter en torno a la erupción del volcán de la Palma.

Descripción del problema

La erupción del volcán de la Palma se ha convertido en un fenómeno ampliamente comentado en las Redes Sociales, tanto por organismos oficiales como usuarios particulares. Twitter no ha sido ajeno a este fenómeno y son numerosos los mensajes que se pueden encontrar en esta red en torno a este tema de actualidad. Dado que Twitter permite descargarse en formato fichero de texto parte de los mensajes que publican sus usuarios, vamos a aprovecharnos de esa funcionalidad para acceder a algunos de esos mensajes y procesar y analizar su contenido.

La estructura del fichero de texto descargado es la siguiente. La primera línea es lo que se denomina la *cabecera del fichero*, es decir, una descripción de cómo se estructuran los datos contenidos a partir de la segunda línea. Después la información de cada *tweet* es almacenada en una línea de texto, conforme a lo descrito en la cabecera, y debería contener los siguientes campos: la fecha/hora en la que se publicó el mensaje, el tipo de aplicación que utilizó el usuario para publicar el mensaje, el identificador del publicador y, finalmente, el contenido del mensaje. Estos campos están separados entre sí por un carácter punto y coma.

Si analizamos visualmente el contenido de los mensajes (último campo de cada tweet), podemos observar que es frecuente que contenga "*tags*" (los cuales comienzan con el carácter '#'), *URL*s (en formato Web http://...), emoticonos, etc. También se puede observar que contiene diferentes signos de puntuación (comas, puntos, dos puntos, punto y coma, etc.). Todos estos elementos deberán ser tratados adecuadamente como parte del trabajo de procesado del fichero, conforme a lo que se explica en el enunciado de las distintas actividades.

El objetivo del trabajo es procesar los tweets del fichero proporcionado para extraer cierto conocimiento de interés.

Notas de interés sobre la realización del trabajo

Es importante que antes de comenzar a trabajar leas con detenimiento las siguientes notas:

- **El trabajo es individual**. Las copias serán calificadas con un cero (a todos los estudiantes implicados, sin excepción).
- **El trabajo constará de tres partes bien diferenciadas**, y cada parte de una o varias tareas sencillas.
- **Para superar el trabajo TP6 será necesario realizar y entregar en plazos las dos primeras partes, pudiendo obtener una nota máxima de 7.0. La tercera parte será "opcional" y permitirá aspirar a la calificación máxima, un 10.0.**
- **La naturaleza del trabajo es no presencial**, aunque los estudiantes dispondrán de sesiones de tutoría concretas para esta actividad.

Notas de interés sobre la entrega final del trabajo

Es importante que desde el primer día tengas presente en todo momento las siguientes notas:

- **Fechas de entrega:** El trabajo TP6 tiene como fecha límite de entrega el miércoles 9/1/2022, a las 23:59h, horario peninsular. Esta fecha es no negociable.
- **Debes entregar** en un único fichero comprimido, llamado `tweetsLP_NIP.zip` (donde NIP debe sustituirse por el NIP del estudiante), todos los ficheros fuente que has programado (sólo los ficheros fuente .java). No puedes utilizar ninguna librería que no haya sido utilizada, explicada o comentada durante el curso o en este guion.
- **La evaluación del trabajo será presencial y tendrá lugar los días 13 y 14 de enero del 2022, aunque estas fechas son aún tentativas y están pendientes de confirmar**. La fecha/hora y lugar en la que cada estudiante tendrá que exponer y defender su trabajo será anunciada con suficiente antelación a través de la plataforma Moodle. Sólo podrán defender su trabajo aquellos estudiantes que lo hayan entregado antes de la fecha límite establecida. La no asistencia a la defensa implica una calificación de No Presentado.

Notas relativas a la gestión del trabajo

Cada estudiante deberá llevar un control individual de la dedicación en horas que invierte en el trabajo de la asignatura. Como parte del material entregado, se ha publicado un documento Excel "*controlDedicacion.xls*" donde deberéis apuntar estas horas. La unidad de control son las horas, por ejemplo, si un día dedicáis 1 hora y 12 minutos, tendréis que anotar 1,25 horas: o si dedicáis 38 minutos, apuntaréis 0,75 horas.

El control de dedicación es obligatorio. **El profesor lo podrá solicitar en cualquier instante para comprobar que se está realizando correctamente**, penalizando si éste no se está completando.

Notas de interés sobre las tutorías del TP6

Los estudiantes dispondrán de 2,5h a la semana de **tutorías especializadas** (**tutorías TP6**) en esta actividad, conforme a los siguientes horarios:

- **Grupo de mañana:** los lunes lectivos de 15:30h a 18h.
- **Grupo de tardes:** los viernes lectivos de 9:30h a 12h.

Es necesario **reservar un turno de "Tutoría TP6" con SUFICIENTE ANTELACIÓN** a través del Calendario Google de tutorías de Pedro Álvarez (la información está en Moodle). Podrás comprobar qué **turnos de "tutorías TP6"** están libres y reservar el que más te interese (uno por día). Al hacer la reserva se te enviará una notificación que contiene un enlace a ***Google Meet***. Dependiendo de la situación sanitaria, estas tutorías se realizarán de forma presencial o telemática a través del enlace generado.

PRIMERA PARTE DEL TRABAJO

Cuando se quieren aplicar técnicas de análisis sobre ficheros de datos es preciso siempre realizar ciertos procesamientos y filtrados previos. El objetivo es "limpiar" de errores e información innecesaria el fichero original, quedándote únicamente con la parte del fichero que te interesa. Estas tareas de pre-procesamiento suelen ser costosas, pero son imprescindibles.

A pesar del cuidado puesto en el proceso de generación del fichero de tweets "*tweets-LaPalma.txt*", éste aún contiene algunos errores y deficiencias que deben ser resueltos. Por este motivo, el objetivo de esta primera parte del trabajo TP6 es que completéis este pre-procesamiento realizando las tareas que se proponen a continuación.

Tarea 1. Pre-procesamiento del fichero para eliminar los caracteres extraños.

Programar un método que pre-procese un fichero de entrada de tweets (el nombre del fichero se especifica a través del parámetro "*input*") con objeto de sustituir por espacios en blanco todos aquellos caracteres no deseables. El resultado de este proceso de filtrado se almacenará en un fichero de salida especificado por medio del parámetro "*output*" (nombre del fichero). Se consideran como caracteres válidos: las letras minúsculas y mayúsculas, los dígitos, el punto, la coma, el punto y coma, los dos puntos, los espacios en blanco y los caracteres '_', '-', '#', '@', y '/'. Todos los demás caracteres deberán ser sustituidos, cada carácter por un espacio en blanco. Además, el fichero de salida deberá contener la cabecera del fichero original y respetar su estructura de líneas.

Adicionalmente, el método programado escribirá por pantalla el número de caracteres sustituidos, conforme al siguiente formato de ejemplo:

```
--------------------------------------
(Tarea 1) Estadísticas del filtro por contenido:
Número de caracteres sustituidos = 43
--------------------------------------
```

Especificación del método a implementar

```
/**
* filtroContenido: elimina del fichero los caracteres no deseables
* @param input   Nombre del fichero de entrada a procesar
* @param output  Nombre del fichero resultado
*/
public static void filtroContenido(String input, String output){…}
```

Tarea 2. Pre-procesamiento del fichero para eliminar los tweets con formato incorrecto.

Programar un método que pre-procese un fichero de entrada de tweets (el nombre del fichero se especifica a través del parámetro "*input*") con objeto de eliminar aquellos

mensajes (líneas de fichero) que no tienen la estructura de información adecuada. Un tweet correcto consta de cuatro campos de información separados entre sí por el carácter ';'. El resultado de este proceso de filtrado se almacenará en un fichero de salida especificado por medio del parámetro "*output*" (nombre del fichero). Además, el fichero de salida deberá contener la cabecera del fichero original y respetar su estructura de líneas.

El método también deberá escribir por pantalla el número de tweets que han sido eliminados, conforme al siguiente formato de ejemplo:

```
-------------------------------------
(Tarea 2) Estadísticas del filtro por estructura:
Número de tweets incorrectos eliminados = 12
-------------------------------------
```

Especificación del método a implementar

```
/**
* filtroEstructura: elimina del fichero los tweets que no constan de 4 campos
* de información
* @param name Nombre del fichero de entrada a procesar
* @param res  Nombre de fichero de resultado
*/
public static void filtroEstructura(String input, String output){…}
```

Tarea 3. Pre-procesamiento del fichero para eliminar los espacios en blanco innecesarios.

Como parte de la Tarea 1 se han sustituido un número significativo de caracteres no deseables por espacios en blanco. Esa sustitución provoca que en el contenido de los mensajes (cuarto campo de un tweet) aparezcan secuencias de dos o más caracteres en blanco consecutivos. El objetivo de esta tarea es compactar esas secuencias, sustituyéndolas por un único espacio en blanco.

Programar un método que pre-procese un fichero de entrada de tweets (el nombre del fichero se especifica a través del parámetro "*input*") con objeto de sustituir las secuencias de dos o más espacios en blanco por un único espacio. Este procesamiento sólo se aplica al contenido (o mensaje) de los tweets, es decir, al cuarto campo de información de cada línea. El resultado de este proceso de filtrado se almacenará en un fichero de salida especificado por medio del parámetro "*output*" (nombre del fichero). Además, el fichero de salida deberá contener la cabecera del fichero original y respetar su estructura de líneas.

El método también deberá escribir por pantalla el número de secuencias compactadas, conforme al siguiente formato de ejemplo:

```
-------------------------------------
(Tarea 3) Estadísticas del filtro de secuencias:
Número de secuencias compactadas = 96
-------------------------------------
```

Especificación del método a implementar

```
/**
* compactaBlancos: sustituye dos o más caracteres ' ' consecutivos por una
* sola ocurrencia
* @param input   Nombre del fichero de entrada a procesar
* @param output  Nombre del fichero de resultado
*/
public static void compactaBlancos(String input, String output) {…}
```

Tarea 4. Extracción de los "tags" utilizados por los usuarios.

El objetivo de esta tarea es extraer la lista de "tags" utilizados por los usuarios en los tweets que publican (no debe contener "tags" repetidos). Un "tag" está formado por el carácter '#' y una secuencia de uno o más caracteres, por ejemplo, "#LaPalma" o "#Volcan2021_LaPalma". El proceso de reconocimiento de "tags" no es sensitivo a letras minúsculas y mayúsculas, es decir, consideraremos que "#LaPalma" y "#lapalma" son el mismo "tag".

Antes de programar la extracción de "tags" es necesario implementar los siguientes métodos:

Cálculo del número de "tags" contenidos en una línea de texto

Dada una línea de texto, especificada como parámetro de entrada ("*línea*"), el método devuelve como resultado el número total de "tags" encontrados.

Especificación del método a implementar

```
/**
* numTagLinea: devuelve el número de tags encontrados en la línea de texto
* de entrada
* @param linea  Línea de texto a procesar
* @return       El número de tags encontrados
*/
public static int numTagLinea(String linea) {…}
```

Extracción de los "tags" contenidos en una línea de texto

Dada una línea de texto, especificada como parámetro de entrada ("*línea*"), el método devuelve los "tag" encontrados en dicha línea. El resultado es un vector de *String*, donde cada componente almacenará un "tag" concreto. Este vector puede contener "tags" repetidos. Si la línea no contiene ningún "tag", el resultado debe ser *null*.

Especificación del método a implementar

```
/**
* extraccionTagLinea: devuelve los tag (como String) encontrados en una
* línea de texto de entrada
* @param linea  Línea de texto a procesar
* @return       Los tags encontrados.
*               Si no encuentra ninguno devuelve el valor null
```

```
*/
public static String[] extraccionTagLinea(String linea) {…}
```

Chequea si existe un "tag" en un conjunto

En Java una estructura de tipo *ArrayList<String>* es similar a un vector de datos *String*. En el **Anexo 1** de este enunciado se explica en detalle este tipo de estructura y la forma de utilizarla. El objetivo de este método es, dado un conjunto de "tags" almacenados en una estructura *ArrayList<String>* (primer parámetro), comprueba si un "tag" concreto (especificado como segundo parámetro) está contenido en la estructura. El resultado del método es un booleano que representa si está (true) o no contenido (false).

Especificación del método a implementar

```
/**
* existeTag: chequea si un tag está contenido en un conjunto de tags
* @param tags   Estructura que contiene el conjunto de tags
* @param tag    Tag a buscar
* @return       Si el tag especificado está contenido en el conjunto
*/
public static boolean existeTag(ArrayList<String> tags, String tag) {…}
```

Una vez programados estos métodos, el objetivo final de la tarea es programar el método de extracción a nivel de ficheros.

Especificación del método a implementar

```
/**
* extraccionTagsFichero: devuelve todos los tags diferentes encontrados
* en un fichero de texto de entrada
* @param input   Nombre del fichero a procesar
* @return        Estructura de datos que contiene los tags
*/
public static ArrayList<String> extraccionTagsFichero(String input) {…}
```

El método debe procesar un fichero de entrada de tweets (el nombre de este fichero se especifica a través del parámetro "input"), extraer todos los "tags" diferentes que en él están contenidos y devolverlos como resultado en una estructura de tipo *ArrayList<String>*.

El método también deberá escribir por pantalla el número de "tags" encontrados, conforme al siguiente formato de ejemplo:

```
-------------------------------------
(Tarea 4) Estadísticas de la extracción de tags:
Número de tags diferentes encontrados = 43
-------------------------------------
```

Tarea 5. Primera versión del programa principal

Como parte del material de este trabajo se proporciona una primera versión del programa principal del sistema de análisis de tweets. Esta versión invoca a los métodos anteriores en el orden correcto. El objetivo de esta tarea es integrar los métodos anteriores en este programa principal y comprobar que funcionan correctamente.

¿Cómo realizar esta comprobación? Se proporciona un fichero de texto, llamado "*tweets-de-prueba.txt*", con sólo 10 tweets. Por tanto, este fichero es muy simple con respecto a los ficheros que el programa debería procesar. Tenéis que usar este fichero para chequear que vuestro programa funciona. Se recomienda que abráis los distintos ficheros temporales que se generan como resultado de los distintos métodos y visualmente comprobéis que estáis resolviendo cada tarea concreta. También podéis buscar visualmente qué "tags" deberíais extraer y comprobar si corresponden con vuestra solución.

Una vez os aseguréis que el programa funciona, lo debéis probar un fichero de datos real. Como parte del material se proporciona un fichero de datos más complejo que el anterior de prueba, llamado "*tweets-LaPalma.txt*".

ANEXO 1: Estructuras de tipo ArrayList<>

Este anexo tiene como objetivo explicar brevemente cómo trabajar con una estructura *ArrayList* de Java. Estas estructuras son similares a los *Array* presentados en clase, donde la diferencia principal es que no es necesario fijar su tamaño o número de componentes en el momento de la declaración (se trata de una estructura de naturaleza dinámica). En el trabajo nos limitaremos a utilizar *ArrayList* de cadenas.

- Declaración de un *ArrayList* de cadenas:

```
ArrayList<String> students = new ArrayList<String>();
```

Nótese que no se especifica un tamaño concreto para la estructura.

- Operaciones necesarias para manipular un *ArrayList* de cadenas:

```
String ejemplo = new String();

students.add(ejemplo);    // Almacena la cadena ejemplo en la estructura

boolean existe = students.contains(ejemplo);  // Comprueba si existe un String
ejemplo en la estructura y guarda el resultado en el booleano existe

int numDatos = std.size();  // Almacena en numDatos el número de datos String
almacenados en ese instante en la estructura

String cadena = std.get(2);  // Accede al String almacenado en la componente 2
de la estructura y lo guarda en la variable String cadena
```

En la siguiente página Web podéis acceder a más información sobre el uso y las operaciones disponibles en torno a las estructuras *ArrayList*:

https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html