ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Grado en Ingeniería del Software

Herramienta para el análisis de opiniones y sentimientos sobre Twitter

Tool for Opinion Mining and Sentiment Analysis on Twitter

Realizado por
David Rubio Cortés
Tutorizado por
Eduardo Guzmán de los Riscos
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, julio de 2017

Fecha defensa:
El Secretario del Tribunal

Resumen

Las redes sociales se han convertido en medios de comunicación en los que participamos toda clase de personas desde todo el mundo, publicando continuamente contenidos acerca de diversos temas. Estos contenidos son frecuentemente comentarios en los que se expresan de alguna forma opiniones. Por otro lado, existen técnicas de procesamiento de lenguaje natural con funcionalidades de análisis de sentimientos, también conocido como minería de opinión, que permiten decidir dado un texto si expresa una opinión positiva, negativa o neutral, e incluso detectar emociones como enfado o sorpresa.

El objetivo de este TFG es desarrollar una aplicación web que se conecte los servicios de la red social Twitter para descargar comentarios en base a criterios de búsqueda, y los utilice como datos de entrada para una o varias herramientas de análisis de sentimientos que estudiaremos y adaptaremos. De esta forma se podrá extraer información referente a la actitud general de los usuarios ante ciertos temas: la proporción de opiniones positivas y negativas respecto del total, los comentarios más extremos, etc. Esto puede ser de utilidad para un amplio número de personas, desde consumidores a la hora de elegir un producto, a empresas que quieren conocer la satisfacción de sus clientes o partidos políticos que orientan sus discursos.

Palabras clave

Twitter, análisis de sentimientos, minería de opinión.

Abstract

Social networks have become a communication media where everybody take part from all over the world, by continuously publishing contents about different topics. These contents are often comments that express opinions. There are techniques of natural language processing, with functions of sentiment analysis, also known as opinion mining, that can decide given a text whether its opinion is positive, negative or neutral, and even detect emotions like anger or surprise.

The goal of this project is developing a web application that connects to Twitter services to download comments by some search criteria, and uses those tweets as input data to one or more sentiment analysis tools which will be studied and adapted. As a result, we could extract information about the general feeling about certain topics: the amount of positive and negative opinions related to the total, the most extreme comments… This can be useful to many people, from a consumer who is choosing a product to companies which want to know the satisfaction of its customers or politicians to set their speeches.

Keywords

Twitter, sentiment analysis, opinion mining.

Índice

Índice de figuras

1. Introducción

La motivación de este Trabajo de Fin de Grado es, a grandes rasgos, integrar comentarios de redes sociales con herramientas de análisis de sentimientos para experimentar con las funcionalidades de estas herramientas utilizando textos reales de usuarios, de manera que se pueda obtener información que refleje opiniones generales sobre algunos temas.

Así, el objetivo es desarrollar una aplicación web en la que un usuario puede realizar una búsqueda e iniciar la descarga y análisis de comentarios, que se guardarán en una base de datos para su posterior consulta y extracción de estadísticas.

1.1. Resumen de tecnologías utilizadas

Para hablar de las tecnologías empleadas en la realización del trabajo podemos comenzar por la base de datos. Los datos con los que trabajamos son fundamentalmente los comentarios, que consisten en un texto e información relativa al mismo como el usuario autor, la fecha, etc. De cada comentario se guardará la información de su respectivo análisis de sentimientos, cuyos campos pueden formar parte del mismo comentario. Así, no hay aparentemente ninguna relación entre entidades típica de las bases de datos SQL. Por esta razón, se ha decidido usar una base de datos no relacional o “no SQL”. Concretamente, la elección ha sido MongoDB.

Figura 1.1. Logo de MongoDB

MongoDB es probablemente la base de datos no relacional más conocida actualmente. Se trata de una base de datos documental, dado que se almacenan los datos como documentos en formato BSON, que es una representación en binario de JSON. Estos documentos no necesariamente deben seguir un esquema fijo.

Eliminar las restricciones de las bases de datos relacionales supone sacrificar la validación y la consistencia en favor de la rapidez y eficiencia. De esta manera, este sistema es adecuado para cantidades grandes de datos en las que no hay muchas operaciones que impliquen trabajar con relaciones entre objetos y no es crítica la consistencia de los datos. Esto se adapta correctamente al que puede ser nuestro modelo.

Pasando a hablar del backend de la aplicación web, la elección ha sido Java Enterprise Edition, dada la práctica que tenemos en esta tecnología en el Grado y la presencia de numerosas bibliotecas en casi todos los campos de aplicación. Particularmente, como se verá más adelante, emplearemos la especificación de JAX-RS para proveer servicios RESTful de manera que el frontend queda independiente solo accediendo a través de métodos HTTP a la capa de servicios. Como servidor utilizamos Glassfish, implementación de referencia de las especificaciones de Java EE.

Figura 1.2. Logo de Java EE

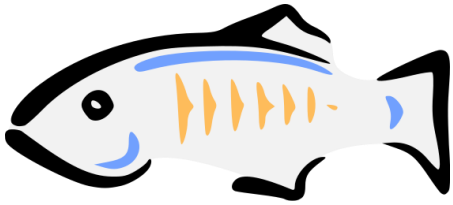


Figura 1.3. Logo de Glassfish

Asimismo, en el frontend hemos optado por utilizar la biblioteca de Javascript jQuery para la realización de peticiones AJAX al servidor y el tratamiento de las respuestas. Para el estilo hemos usado el conocido framework Bootstrap.

Figura 1.4. Logo de jQuery

Figura 1.5. Logo de Bootstrap

Para el desarrollo hemos utilizado principalmente el entorno NetBeans, por su integración preinstalada con JavaEE en general y Glassfish en particular. Además, hemos utilizado Apache Maven para la gestión de dependencias, lo que ahorra mucho esfuerzo. No enumeramos ahora las bibliotecas que hemos añadido, sino que lo haremos en su correspondiente sección más adelante.

Figura 1.6. Logo de Netbeans

Figura 1.7. Logo de Maven

Otros programas utilizados son, entre otros, editores ligeros como Sublime Text, las herramientas de desarrollo del navegador Google Chrome, la utilidad Postman para probar peticiones al servidor, o Robomongo para gestionar la base de datos, etc.

1.2. Metodología

Hemos planteado la metodología para un desarrollo iterativo e incremental, es decir, siguiendo unas fases de una duración más o menos similar en las que se van realizando conjuntos de tareas teniendo siempre una base en funcionamiento. No nos extenderemos en detallar los tiempos y tareas concretas de cada iteración, pero sí enumeraremos en términos generales cuáles han sido y qué se iba teniendo en cada momento.

En primer lugar, se estudiaron sin entrar en detalles de implementación algunas herramientas de análisis de sentimientos por un lado, y por otro las posibilidades de extracción de comentarios de redes sociales, Twitter en concreto. Una vez que se vio qué se podía hacer a grandes rasgos, se pasó a implementar las funcionalidades necesarias para descargar tweets en la base de datos.

En este punto tuvimos funcionando una aplicación, todavía sin interacción del usuario, que permitía conectarse a Twitter y comenzar una descarga de tweets con términos de búsqueda. Entonces, lo siguiente fue integrar ya herramientas de análisis de sentimientos que analizaran el texto de los tweets y guardaran su valoración en la misma base de datos, pudiendo iniciar y detener el proceso así como cargar los datos externamente a través de una capa de servicios.

Para entonces, puede decirse que teníamos, al menos en cuanto a la parte de backend, un producto mínimo viable. A continuación, trabajamos en el frontend hasta desarrollar una página funcional donde se puede acceder a los servicios desde una interfaz web, completando así el mínimo producto viable.

Luego en la última iteración se trató de añadir el diccionario personalizado de análisis de sentimientos que luego detallaremos y algunos añadidos más, así como paralelamente la redacción de este documento y continuamente probando el funcionamiento.

Como es habitual en los enfoques ágiles, hay constantes evoluciones y cambios. No es la intención de la memoria describir el desarrollo en el orden real, ya que sería muy complicado establecer partes diferenciadas de la aplicación para estructurarla. Como consecuencia, en los siguientes capítulos no se tendrán en cuenta directamente las iteraciones del desarrollo sino el producto conseguido finalmente, lo que nos parece una descripción más adecuada de cara a su lectura.

1.3. Estructura de la memoria

Una vez conocidos en general los objetivos y motivaciones del TFG, así como las tecnologías utilizadas y la metodología seguida, describimos el contenido del documento que se desarrolla de aquí en adelante.

En el segundo capítulo hablaremos del estudio realizado en cuanto a las herramientas de análisis de sentimientos, con una visión general y detalles particulares de las bibliotecas y APIs seleccionadas, terminando con la implementación de una propia.

El tercer capítulo tratará lo relacionado con la extracción de comentarios, viendo las APIs disponibles en Twitter con sus posibilidades y limitaciones.

El cuarto capítulo quizás puede considerarse el núcleo, comentando el desarrollo del backend de la aplicación web. Se verá la arquitectura general, el acceso a la base de datos, la integración con herramientas de análisis de sentimientos de las elegidas en el segundo capítulo, la descarga de los tweets con lo descrito en el tercero, y por último la implementación de la capa de servicios necesarios para la interacción del usuario.

El quinto capítulo, de forma complementaria al cuarto, comprende el desarrollo del frontend de la aplicación. Mostramos el esquema general y cómo son las funcionalidades en la interfaz: iniciar o parar los análisis, ver los tweets recientes, las estadísticas, el top, introducir palabras personalizadas para el análisis, etc.

Para finalizar, el sexto capítulo contiene resultados y conclusiones: validaciones que se han podido realizar, errores y problemas de las tecnologías y la aplicación, funcionalidades que se pueden ampliar o añadir en el futuro y algunas conclusiones definitivas.

2. Herramientas de análisis de sentimientos

2.1. Visión general

El análisis de sentimientos forma parte del campo del procesamiento del lenguaje natural, conocido por sus siglas en inglés como NLP. Este campo de las Ciencias de la Computación comprende múltiples funcionalidades: traducción, reconocimiento del habla, análisis sintáctico y morfológico, identificación de temas, etc. La evolución en estas técnicas está avanzando rápidamente en los últimos años, debido seguramente a los avances en inteligencia artificial, en particular con aprendizaje automático (*machine learning*), y a la necesidad de crear software capaz de entender el lenguaje natural de los usuarios.

Entrando al análisis de sentimientos, existen varias tareas a resolver y enfoques a aplicar. La tarea fundamental es averiguar en un texto la connotación o polaridad del mismo, lo que se traduce normalmente en una puntuación que implica clasificar el texto como positivo, negativo o neutral. Por ejemplo, la frase “me lo estoy pasando estupendamente” tendría una connotación positiva, mientras que la frase “vaya día más aburrido”, tendrá una connotación negativa. En principio puede parecer sencillo, pero la tarea se complica si así lo hacemos con el lenguaje: el texto “no me gusta mucho, pero realmente no está tan mal” tiene una connotación negativa desde un punto de vista subjetivo a la vez que una connotación positiva objetivamente. Detectar si el texto es objetivo o subjetivo es precisamente otra de las tareas. También podemos hablar de identificar características o entidades a las que se dirige la opinión, o detectar emociones (análisis de emociones) entre ira, miedo, felicidad, tristeza, sorpresa, etc. Otra tarea difícil para la que existen tratamientos es detectar la ironía. En la aplicación nos vamos a centrar en la primera tarea de puntuar el texto por su polaridad.

Los enfoques para las técnicas de análisis de sentimientos son bastante variados, desde complejas redes neuronales, máquinas de vectores soporte o redes bayesianas hasta diccionarios de palabras más sencillos. No entraremos de momento en más detalles de cómo funcionan los métodos de análisis de sentimientos, sino que nos interesamos en la aplicación práctica. Para ello, estudiamos varias herramientas existentes, que en principio incluyan modelos, y comprobamos sus ventajas e inconvenientes. Un aspecto interesante que no hemos comentado hasta ahora es el idioma, por simplicidad nos centraremos en herramientas compatibles con el inglés, que es lo más extendido, aunque tendremos en cuenta las posibilidades respecto a otros idiomas, especialmente el español.

2.2. Estudio de herramientas

Hemos seleccionado varias herramientas para analizarlas, algunas son bibliotecas que hay que añadir al proyecto donde se vayan a usar mientras que otras se basan en APIs que están funcionando en servidores externos, teniendo por lo general límites de uso.

2.2.1. Stanford CoreNLP

Se trata de una biblioteca Java de código abierto desarrollada por la Universidad de Stanford, que proporciona múltiples aplicaciones de procesamiento de lenguaje natural. Una de ellas es el análisis de sentimientos, cuya implementación está basada en redes neuronales. Incorpora modelos para varios lenguajes, pero para el análisis de sentimientos parece ser que solo está disponible en inglés. La herramienta proporciona dado un texto una clasificación (positivo, negativo o neutral), así como la puntuación. Podemos ver un ejemplo a través de su página web en la Figura 2.1.

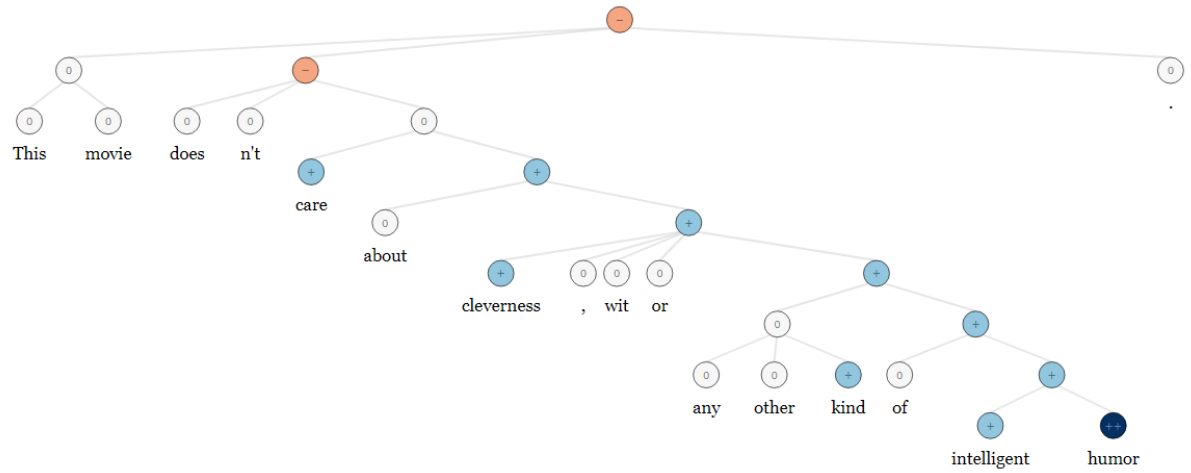


Figura 2.1. Demo de Stanford CoreNLP

2.2.2. Apache OpenNLP

OpenNLP es otra biblioteca de código abierto de la fundación Apache escrita en Java que está pensada para procesamiento de lenguaje en general. Al contrario que CoreNLP, no parece haber documentación específica para análisis de sentimientos, aunque existen en la red algunos ejemplos en los que usuarios utilizan datos de entrenamiento para un clasificador que etiqueta textos de la misma manera.

2.2.3. Sentiment140

Sentiment140 es una aplicación web bastante similar a la nuestra, dado que está orientada a tweets y si se inicia sesión con una cuenta de Twitter se puede hacer una búsqueda y obtener resultados de análisis de sentimientos. Fue desarrollada por estudiantes de la Universidad de Stanford, utilizando técnicas de aprendizaje automático. En la Figura 2.2, podemos ver la apariencia de la página tras hacer una búsqueda.

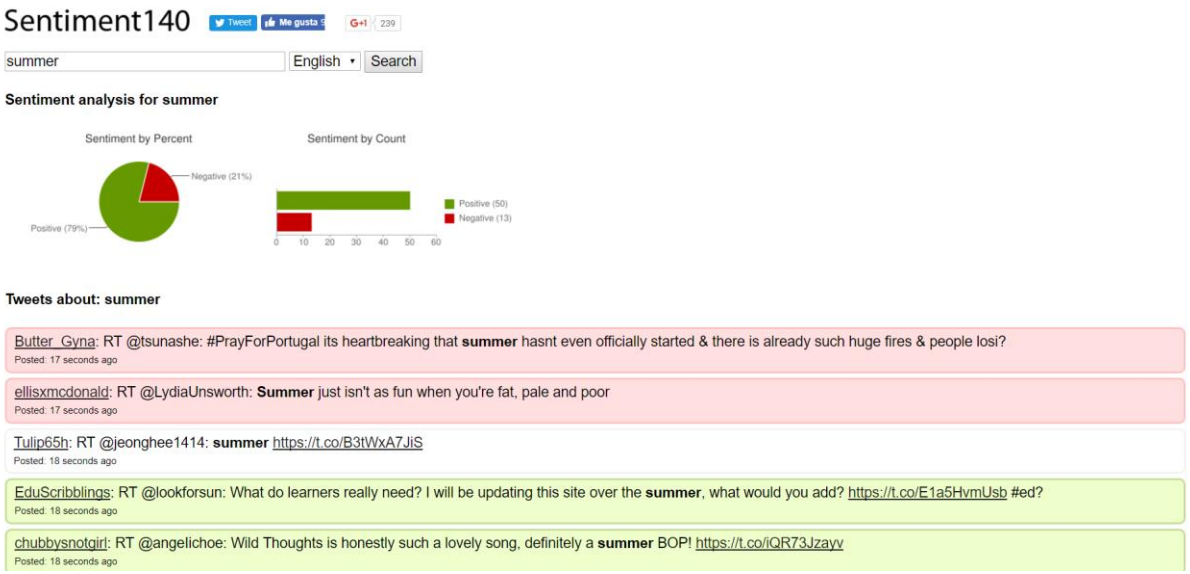


Figura 2.2. Captura de ejemplo de Sentiment140

Sentiment140 es compatible con el español además del inglés, y lo más relevante para nuestros intereses es que proporciona una API REST para clasificar tweets en positivos, negativos o neutros. No aparecen muchos detalles sobre los límites de la misma, solamente se dice que no se debe hacer más de una petición individual por segundo sin registro o una petición de múltiples tweets que dure más de un minuto en procesarse.

2.2.4. Google Cloud Natural Language API

Entre los muchos servicios en la nube de Google, está la API de lenguaje natural, que presenta un servicio de análisis de sentimientos, en versión beta. Anteriormente existía un servicio similar con la llamada Prediction API, pero ha quedado obsoleto y dejará de funcionar en abril de 2018. Se utilizan algoritmos de aprendizaje automático, teniendo modelo predefinido y la posibilidad de entrenar con información propia.

Las llamadas a la API devuelven dado un texto una puntuación similar a la que dan otras herramientas, determinando si es positivo, negativo o neutral, y en principio cuenta con soporte multilenguaje. Sin embargo, el servicio solo es gratuito hasta las 5000 peticiones al mes, lo que nos parece bastante escaso teniendo en cuenta el volumen de comentarios que pueden analizarse.

2.2.5. Microsoft Azure Text Analytics API

Habiendo visto las herramientas en la nube de Google, podemos echar un vistazo a otra referencia en este tipo de servicios como es Microsoft con su plataforma Azure. El servicio de análisis de sentimientos da unos resultados similares, pero da la sensación de que está más avanzado, al ser una versión 2.0 y ofrecer una prueba online sin necesidad de ningún registro, que podemos ver en la Figura 2.3.

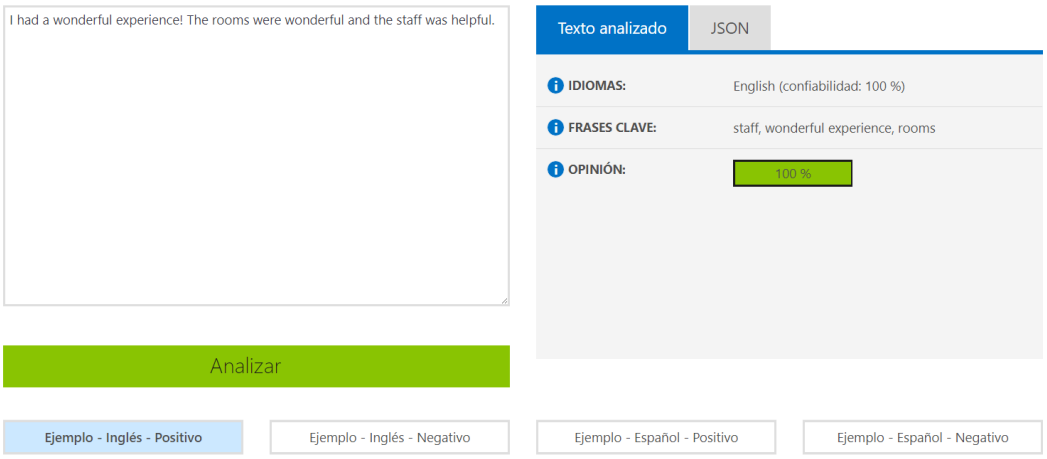


Figura 2.3. Demo de Azure Text Analytics

Es compatible con inglés y español, pudiendo detectar el idioma automáticamente, y además de la puntuación en porcentaje se dan frases clave que pueden servir para detectar temas. El problema vuelve a ser el precio: el límite gratuito es el mismo que el caso de Google, 5000 transacciones por mes.

2.2.6. Text-proccessing

Se trata de una API REST basada en el paquete de procesamiento de lenguaje natural de Python NLTK. El resultado del servicio es análogo al de otros indicando la polaridad, pero en este caso se indica la probabilidad de cada etiqueta (positivo, negativo o neutro) en lugar de una sola puntuación. La limitación es de 1000 peticiones diarias. Soporta inglés y también francés y holandés.

2.2.7. TheySay PreCeive REST API

Este servicio REST ofrece muchas más características aparte de la típica evaluación de polaridad, como la detección de emociones que se había comentado antes. Un ejemplo lo vemos en la Figura 2.4.

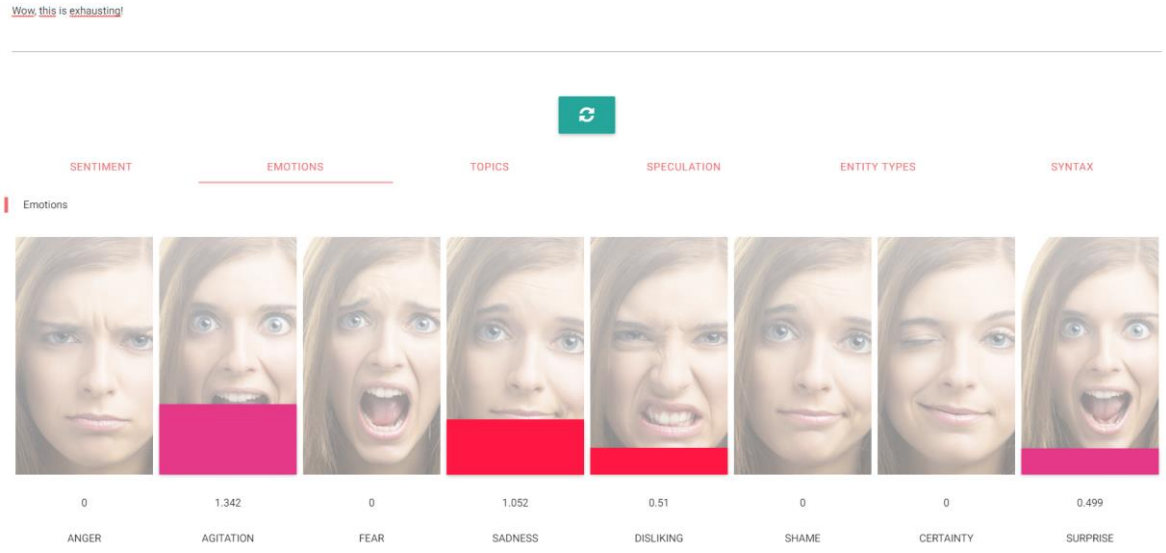


Figura 2.4. Demo de TheySay API

Tiene soporte para inglés, alemán y español. Las limitaciones son 500 peticiones al día y 30 por minuto.

2.2.8. IBM Watson Natural Language Understanding

Otro servicio cloud puntero es el de IBM, que tiene entre sus campos de aplicación el procesamiento de lenguaje natural. Igualmente ofrece puntuar un texto según su opinión con gran soporte de idiomas: inglés, español, alemán, italiano, etc. También como es habitual tiene estrictos límites, de 1000 elementos analizados diarios.

2.2.9. MonkeyLearn

MonkeyLearn es una API con bastante buena apariencia para analizar textos con machine learning. En su web hay una pequeña demo para puntuar un texto y etiquetarlo con análisis de sentimientos, pero esto es solo una mínima parte de la herramienta ya que hay posibilidades para crear clasificadores personalizados, detectar temas, etc. Es multilenguaje, pero limitada en su plan gratuito a 1000 peticiones al mes.

2.2.10. AFINN: Sentiment / Sentimental / SentimentalJ

En este punto hablamos de varias herramientas en lugar de una sola, que comparten internamente el método con el que realizan el análisis de sentimientos. En lugar de utilizar enfoques complejos con aprendizaje automático más o menos visibles, se utiliza lo que podemos llamar un diccionario de palabras. Se trata de una lista de pares clave-valor, en la que las claves son palabras y los valores son puntuaciones arbitrarias que señalan la polaridad de la opinión de la palabra individualmente. De esta manera, los modelos son tan simples como ficheros en los que se listan estos pares en algún formato, y los análisis en principio no tienen en cuenta factores de sintaxis como las negaciones, solo palabras independientes.

Una conocida lista de palabras (en inglés) es AFINN, con varias versiones realizadas etiquetando palabras manualmente por el autor, Finn Årup Nielsen, de la Universidad Técnica de Dinamarca. Como es obvio, se trata de un sistema menos sofisticado que otros anteriormente comentados, pero se gana en eficiencia y sencillez de implementación con unos resultados que pueden llegar a tener más precisión (de entre los textos clasificados con una etiqueta, los aciertos) aunque menos sensibilidad (de entre todos los textos que tendrían la etiqueta, los clasificados con ella).

Lo que es especialmente interesante de este enfoque es que el modelo podría ser modificado por el usuario, introduciendo este palabras con su valoración. Esta característica resultará un punto importante en la aplicación, como veremos más adelante.

Una primera biblioteca que implementa esta técnica es Sentimental, para Node.js. Como la gran mayoría de herramientas de análisis de sentimientos, devuelve de un texto una puntuación que determina si es positivo, negativo o neutral. Esta biblioteca ha sido codificada en Java, llamada SentimentalJ. Análogamente, otra biblioteca de Node.js llamada sentiment funciona igual pero mejorando el rendimiento y añadiendo emojis a la lista, lo que también adquiere importancia hoy en día cuando los emojis se están usando cada vez más. Las tres bibliotecas son de código abierto.

2.3. Resumen y tabla comparativa

Entre las herramientas que hemos visto, hay desde bibliotecas sencillas como Sentiment hasta avanzadas APIs como MonkeyLearn. El problema común de las APIs es la dependencia de un servicio externo con sus limitaciones. Por ello, nos centraremos en las bibliotecas, eligiendo para usarlas Stanford CoreNLP y SentimentalJ como implementación de AFINN. La tabla 1 muestra una comparativa para terminar con el estudio de las herramientas de análisis de sentimientos:

Herramienta	Tipo	Idiomas	Funciones	Límites	Licencia
Stanford NLP	Biblioteca	Inglés	Polaridad	-	Open source (GPL)
OpenNLP	Biblioteca	-	Clasificar	-	Open source (Apache)
Sentiment140	Aplicación web con API REST	Inglés Español	Polaridad	1 /segundo 1-4M /día (comercial)	Propietario
Google Cloud Platform	API REST	Inglés + Otros	Polaridad	5000 /mes	Propietario
Microsoft Azure	API REST	Inglés Español	Polaridad Palabras clave	5000 /mes	Propietario
Text Processing	API REST	Inglés Francés Holandés	Polaridad	1000 /día	NLTK: Open source (Apache)
They Say	API REST	Inglés Español + Otros	Polaridad Emociones	500 /día + 30 /minuto	Propietario
IBM Watson	API REST	Inglés Español + Otros	Polaridad	1000 /día	Propietario
MonkeyLearn	API REST	Inglés + Otros	Polaridad	1000 /mes	Propietario
AFINN	Listas de palabras (Archivo)	Inglés	Polaridad	-	Open Database License

Tabla 1. Comparativa de herramientas de análisis de sentimientos

2.4. Implementación propia: Sentictionary

La última de las herramientas que hemos estudiado consistía en una lista de palabras anotadas con un valor que determina su puntuación de análisis de sentimientos. Teniendo dicha lista y varias implementaciones de código abierto, planteamos una implementación Java propia que podamos personalizar, haciendo énfasis en la posibilidad de modificar la lista o diccionario de análisis. Creamos así nuestra propia biblioteca de análisis de sentimientos, que llamamos “Sentictionary”.

Para ello nos basamos en la biblioteca sentiment de Node.js, realizada por el desarrollador thisandagain. Lo hemos pensado así dado que esta herramienta transforma la lista AFINN original en un fichero JSON, añadiendo además emojis. Así, usamos este fichero para cargar la lista de una forma más sencilla, ya que no se hace necesario codificar la lógica para leer la lista original sino usar alguna biblioteca de JSON.

Entrando más en detalles, tenemos un fichero sentiment.json, en el que queda definido un diccionario de pares clave-valor siendo cada clave una palabra o emoji y cada valor un número entre -5 y 5, siendo -5 una valoración muy negativa y 5 muy positiva. Así, por ejemplo, la palabra “excellent” tiene una valoración de 3, mientras que la palabra “hazardous” tiene -3. Creamos una clase Sentictionary, cuyos objetos se utilizarán para analizar textos según el diccionario que tengan cargado. Para inicializarlos utilizamos la biblioteca de Google GSON para transformar el JSON de un String o de un fichero (por defecto, el sentiment.json) en un Map<String, Double> que queda guardado en una variable de instancia. La clase entonces tiene un método que realiza el análisis de sentimientos propiamente dicho, recibiendo un String como parámetro que es separado en palabras mediante un StringTokenizer; cada palabra es consultada en el diccionario y si se encuentra, se suma la valoración a un acumulador, que es devuelto al final del proceso. También están los métodos para añadir una entrada (par clave-valor) al diccionario o para añadir varias a la vez de un diccionario adicional.

```
public double sentimentScore(String text) {
    double score = 0;
    StringTokenizer tokenizer = new StringTokenizer(text, " \t\n\r\f.:,;!?¡¿\"@#");
    while (tokenizer.hasMoreTokens()) {
        Double tokenScore = this.dictionary.get(tokenizer.nextToken().toLowerCase());
        if (tokenScore != null) {
            score += tokenScore;
        }
    }
    return score;
}

public void putInDictionary(String word, double sentiment) {
    this.dictionary.put(word.toLowerCase(), sentiment);
}

public void putInDictionary(Map<String, Double> additionalDictionary) {
    this.dictionary.putAll(additionalDictionary);
}
```

En definitiva, hemos implementado en Java una biblioteca sencilla para obtener la polaridad dado un texto, con la peculiaridad de poder añadir entradas al diccionario de análisis de sentimientos. Esto se podía hacer en parte en la biblioteca Sentiment en la que nos hemos basado, pero de una forma más rudimentaria teniendo que pasar un JSON al método para analizar el texto, sin poder guardar internamente los cambios. Utilizaremos en la aplicación final las bibliotecas antes seleccionadas (Stanford CoreNLP y SentimentalJ, así como Sentictionary). Entraremos en detalles de implementación en el apartado 4.3.

3. Extracción de comentarios

En capítulos anteriores, hemos estudiado diversas herramientas existentes de análisis de sentimientos e implementado una propia. Ahora veamos cómo recolectamos los comentarios que serán datos de entrada para las herramientas.

Hemos elegido la red social Twitter, por varias razones. Primero, aunque existe una componente multimedia, desde su origen la red social se ha basado en mensajes cortos. Este tipo de textos son bastante adecuados para ser analizados dado que seguramente traten de un único tema y constituyan una única opinión, lo que no ocurre en textos largos que tendríamos que fragmentar para razonar sobre ellos. Otra razón de peso es que existe en la red bastante material y documentación acerca de las APIs que proporciona, lo que indica gran apoyo por parte de la comunidad. Prueba de ello es la cantidad de aplicaciones externas que usan Twitter.

Se valoró utilizar la que es la red social más utilizada: Facebook. Sin embargo, pronto fue descartada por la razón de que la mayoría de publicaciones en la misma son privadas, es decir, buena parte de los usuarios escriben comentarios que solamente pueden ver sus amigos. Esto no suele ser así en Twitter (aunque existen cuentas privadas), con lo que se adapta mejor a nuestros objetivos, en los que pretendemos conocer la opinión general de usuarios que hacen pública la suya.

Así pues, hemos de estudiar qué posibilidades y limitaciones tienen las APIs de Twitter para buscar tweets. Hay que destacar que la API de Twitter ha pasado por distintas versiones, siendo el cambio quizás más grande en el caso de las búsquedas el que desde hace algunos años es necesaria autenticación, es decir, credenciales de un usuario registrado, para poder realizar búsquedas, aunque no tengan relación con la cuenta en sí. Paradójicamente, desde la web se pueden realizar búsquedas sin iniciar sesión, y según se reconoce en la documentación los resultados de las APIs y de la web pueden diferir.

Como veremos en los próximos capítulos, por el carácter prototipo de la aplicación objetivo usaremos unas únicas credenciales propias, lo que obliga a que solo se pueda ejecutar un único análisis a la vez.

3.1. API REST de Twitter

Profundizando ya en las APIs de Twitter, la más usada y conocida es la API REST. Esta permite varias funciones para interactuar en la red social, pero nosotros nos quedaremos con el servicio de búsquedas. Este servicio nos permite enviar términos de búsqueda combinados con operadores de manera que se puedan añadir expresiones lógicas, filtrar por usuario, hashtag, fecha, etc. Twitter no permite mediante este servicio encontrar tweets más antiguos que una semana. Además, la limitación es de hasta 450 peticiones cada 15 minutos (con la llamada autenticación por aplicación, más permisiva pero sin acceso a la cuenta de usuario, irrelevante para nuestro caso). Cada respuesta puede configurarse para contener un número de tweets, hasta 100.

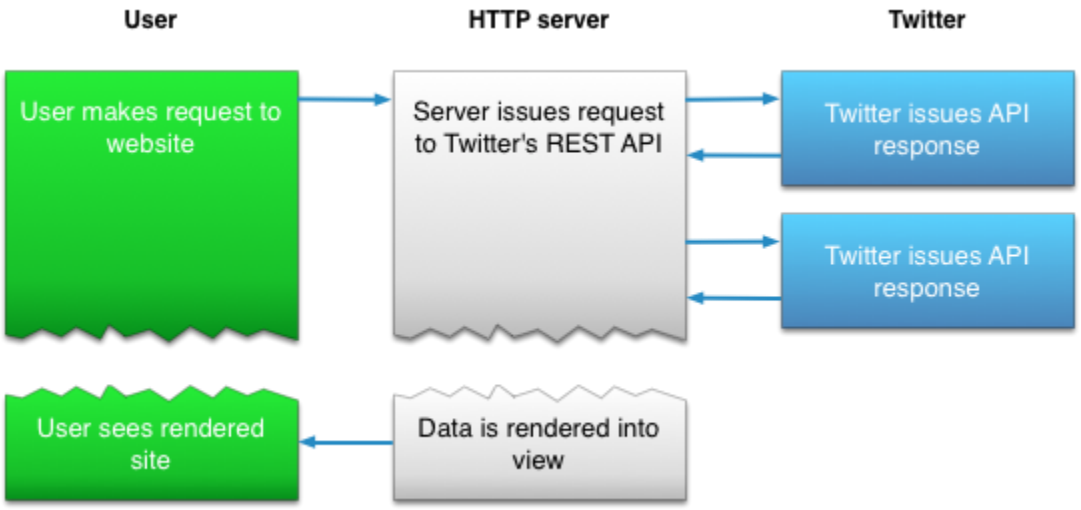


Figura 3.1. Esquema de funcionamiento de la API REST de Twitter

Teniendo en cuenta los factores comentados, se pensaron en algunas posibilidades de cara a la implementación. El enfoque al que se llegó, sin entrar todavía en detalles de implementación, para obtener un número considerable de tweets fue realizar un método con la anotación Scheduled, que se ejecutará continuamente cada cierto tiempo ajustado para no sobrepasar el límite, descargando en cada petición 100 tweets y quedándonos con el identificador del tweet más antiguo, que será utilizado en la siguiente petición como máximo identificador tras restarle uno. De esta manera, los 100 tweets de cada petición tienen que ser más antiguos que los 100 de la anterior, y así obtenemos el mayor número posible de tweets yendo hacia atrás en el tiempo. El único problema sería tratar la llegada al final de los resultados, que podría tardar poco tiempo en caso de temas con poco volumen de tweets o justo al contrario. Finalmente, aunque se implementó una primera versión, este enfoque se quedó fuera en favor de la API de Streaming, que vemos a continuación.

3.2. API de Streaming de Twitter

Otra API de Twitter, quizás menos conocida, es la API de streaming. Se caracteriza por destinarse a aplicaciones de tiempo real, es decir, a obtener tweets conforme estos se van publicando. Esta funcionalidad facilita la implementación, ya que como veremos se trata de esperar a recibir tweets sin tener en cuenta identificadores, fechas o limitación alguna. La desventaja es que no hay manera de recuperar tweets ya publicados, para lo que hay que usar el servicio REST de búsqueda. Por lo tanto, lo ideal sería combinar ambas APIs.

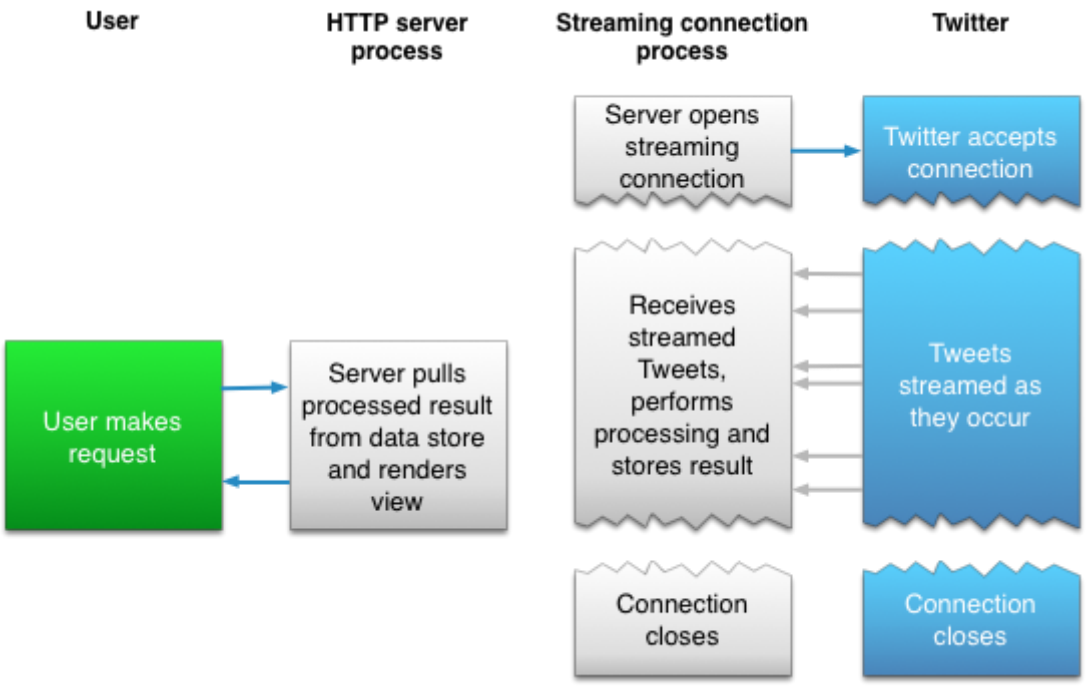


Figura 3.2. Esquema de funcionamiento de la API de Streaming de Twitter

Hay varios flujos de tweets a los que conectarse, los que nos interesan son los públicos, a los que se puede conectar filtrando con términos de búsqueda de forma similar, que no igual, a la búsqueda de la API REST. Las limitaciones del streaming no están tan claras como en la API REST, pero en cualquier caso son más ligeras. Solamente puede haber una conexión por usuario a la vez, y al parecer la cantidad de tweets que lleguen no debe exceder el 1% del total de tweets existentes.

Como decíamos, usar el streaming nos permite conectarnos y esperar tweets, sin tener que controlar nada más que los términos de búsqueda, y con un funcionamiento en teoría ilimitado, sin terminar nunca, pudiendo realizar análisis durante el tiempo que queramos. Por este motivo, utilizaremos este enfoque para la aplicación final, que pasamos a describir en el siguiente capítulo.

4. Backend de la aplicación web

4.1. Arquitectura general

En los anteriores capítulos hemos estudiado herramientas de análisis de sentimientos y la extracción de comentarios de Twitter. Con estas bases, podemos pasar a explicar la aplicación web que se ha planteado, que llamamos Twitter Feeling Gatherer, que viene a significar “recolector de sentimientos de Twitter”, y cuyas iniciales coinciden con las de “Trabajo Fin de Grado”. Comenzamos por el backend, que comprendería la aplicación desde la conexión de la base de datos hasta la exposición de servicios RESTful.

Hemos desarrollado un proyecto Maven con el arquetipo de aplicación web, es decir, se trata de un proyecto WAR. Al principio se planteó como un proyecto empresarial completo, es decir, un EAR con sus módulos WAR y JAR. Sin embargo, no se han incluido EJB y el manejo de dependencias con Maven es más sencillo si tenemos un solo módulo.

Las clases han quedado distribuidas en cuatro paquetes: *database*, para el acceso a la base de datos MongoDB, *sentiment* para el análisis de sentimientos, *service* para los servicios REST y por último *tweets*, el que quizás puede considerarse como el eje central de la aplicación, con las clases para la descarga y el manejo de los tweets.

En la Figura 4.1 se muestra un diagrama con los paquetes comentados y las clases que contienen. Los siguientes apartados describen respectivamente cada paquete con detalles de sus clases.

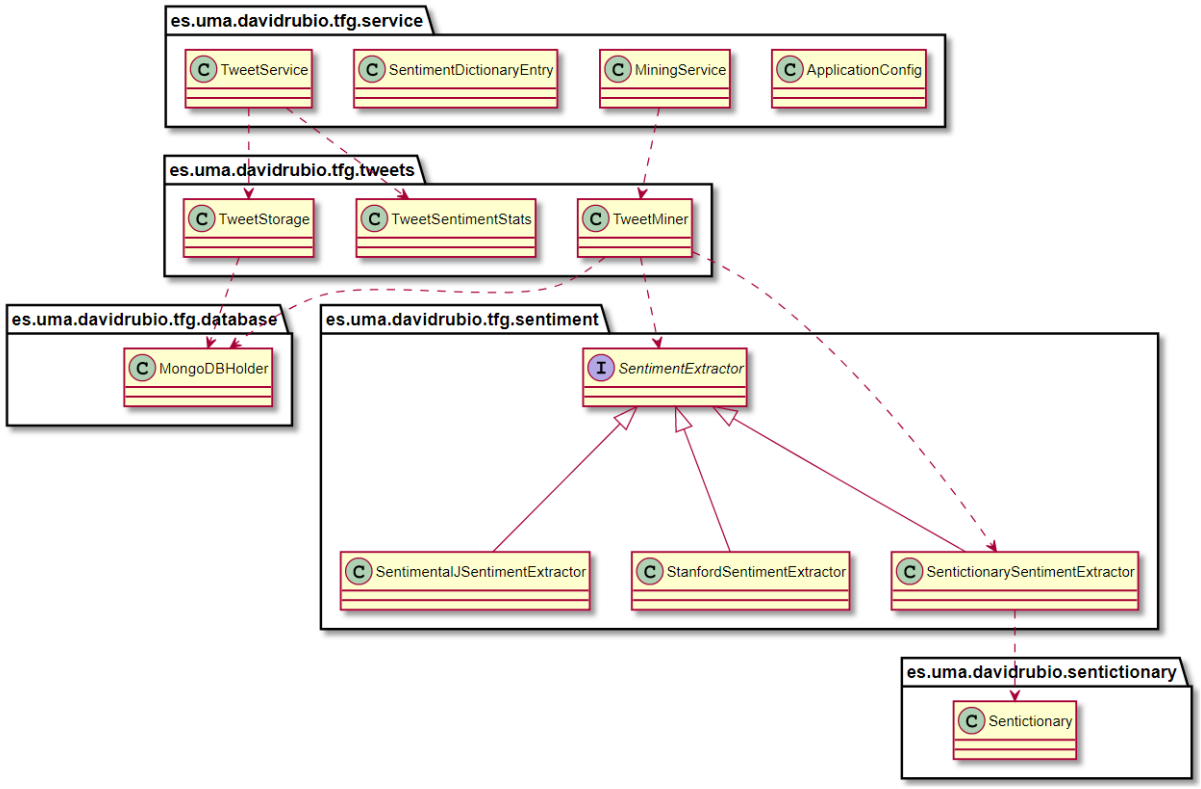


Figura 4.1. Diagrama de paquetes y clases

4.2. Acceso a la base de datos

Como se comentó, hemos elegido MongoDB como base de datos no relacional para almacenar los tweets con sus resultados de análisis de sentimientos. Al contrario que en las bases de datos SQL, no se usan entornos típicos de mapeo objeto-relacional como JPA o Hibernate, sino que hay que buscar otras alternativas. No obstante, optamos por acceder a los documentos BSON de la base de datos sin construir objetos específicos de nuestro dominio, ya que dado que solo trabajamos con objetos que son tweets el manejo de los objetos no se complica demasiado con los medios que proporciona el driver para Java de MongoDB.

El driver de MongoDB, una vez añadido a las dependencias en nuestro pom.xml, proporciona una clase para realizar la conexión, llamada MongoClient. Asimismo, las instancias de esta clase tienen un método para recoger el objeto que abstrae la base de datos concreta con la que vamos a trabajar, de clase MongoDatabase. Desde este objeto ya podemos acceder a las colecciones de documentos y hacer consultas y operaciones sobre los mismos. Para compartir una instancia de clase MongoDatabase en la aplicación, hemos aplicado un patrón Singleton, en la única clase presente en el paquete database, llamada MongoDBHolder. Este es el código de dicha clase.

```
public class MongoDBHolder {

    public static final String MONGODB_URI = "mongodb://localhost:27017";
    public static final String DB_NAME = "tfg";
    public static final String TWEETS_COLLECTION_NAME = "tweets";

    private static MongoDatabase db;

    private MongoDBHolder() {
    }

    public static MongoDatabase getDatabase() {
        if (db == null) {
            db = new MongoClient(new MongoClientURI(MONGODB_URI)).getDatabase(DB_NAME);
        }
        return db;
    }

}
```

En las clases correspondientes veremos cómo llamamos a esta clase y hacemos las operaciones oportunas.

4.3. Análisis de sentimientos

Para realizar el análisis de sentimientos, hemos querido definir una interfaz común para abstraernos de las peculiaridades de las bibliotecas que seleccionamos en el capítulo 2, así como de otras que puedan añadirse en el futuro. De esta manera definimos la interfaz con cuatro métodos: dos de ellos permiten hallar la polaridad (*sentiment score*) bien tal cual se calcula o normalizada siendo el cero el valor neutral. El tercer método es para añadir entradas al diccionario de análisis de sentimientos, y el cuarto es para obtener el diccionario formado por las entradas añadidas. Estos dos últimos métodos no serán implementados en caso de que la biblioteca subyacente no soporte el añadido de palabras, con lo que lanzarían UnsupportedOperationException.

```
public interface SentimentExtractor {

    public double sentimentScore(String text);

    // 0 is neutral, <0 is negative and >0 is positive
    public double sentimentScoreNormalized(String text);

    public void putInDictionary(String word, double sentiment);

    public Map<String, Double> getCustomDictionary();

}
```

De esta manera, las clases del paquete *sentiment* implementan esta interfaz, habiendo una StanfordSentimentExtractor, otra SentimentalJSentimentExtractor y la última SentictionarySentimentExtractor, esta con la biblioteca que hemos implementado siendo la que soporta el diccionario personalizado. Todas las bibliotecas son importadas mediante su dependencia en el pom.xml. Mostramos el código del caso más complejo de StanfordCoreNLP:

```
private StanfordCoreNLP corenlpPipeline;

public StanfordSentimentExtractor() {
    Properties corenlpPipelineProperties = new Properties();
    corenlpPipelineProperties.setProperty("annotators", "tokenize, ssplit, parse,
sentiment");
    this.corenlpPipeline = new StanfordCoreNLP(corenlpPipelineProperties);
}

@Override
public double sentimentScore(String text) {
    Annotation annotation = this.corenlpPipeline.process(text);
    double sentimentScore = 2;
    int sentenceMaxLength = 0, currentSentenceLength;
    for (CoreMap sentence : annotation.get(CoreAnnotations.SentencesAnnotation.class)) {
        currentSentenceLength = sentence.toString().length();
        if (currentSentenceLength > sentenceMaxLength) {
            Tree tree =
sentence.get(SentimentCoreAnnotations.SentimentAnnotatedTree.class);
            sentimentScore = RNNCoreAnnotations.getPredictedClass(tree);
            sentenceMaxLength = currentSentenceLength;
        }
    }
    return sentimentScore;
}

@Override
public double sentimentScoreNormalized(String text) {
    return this.sentimentScore(text) - 2;
}
```

Nótese como hay que restar 2 al valor de polaridad que devuelve StanfordCoreNLP, puesto que está entre 0 y 4.

4.4. Descarga de tweets

Llegamos al punto culminante de la implementación, donde se integra lo visto hasta ahora, en el paquete tweets. La clase fundamental es TweetMiner. En ella, están las credenciales de OAuth para la autenticación con Twitter (obtenidas desde la web de Twitter con nuestra cuenta de usuario). Para trabajar con Twitter utilizamos la biblioteca Twitter4J, de código abierto y recomendada desde la propia red social.

Figura 4.2. Logo de Twitter4J

Las variables de instancia de la clase son, primero un objeto de tipo MongoDatabase para la base de datos, que queda inicializado llamando a la clase MongoDBHolder con el método estático que devuelve el Singleton. Luego está un objeto de tipo SentimentExtractor, la interfaz definida para el análisis de sentimientos. Esta variable se inicializa por defecto a un objeto de la clase SentictionarySentimentExtractor, pero bien podría cambiarse por otro de cualquier clase que implemente la interfaz, siendo esto una aplicación del patrón Estrategia. A continuación, está un objeto de tipo TwitterStream, que es el encargado de gestionar el flujo de tweets, inicializado con las credenciales de Twitter. Por último, queda un String donde se guardará la cadena de búsqueda.

La lógica importante está en la configuración del TwitterStream. A este se le añade un objeto de tipo StatusListener, creado anónimamente, que determina qué hacer ante las respuestas de la API de streaming de Twitter. El método más relevante se llama onStatus, que recibe como parámetro un objeto Status de Twitter4J, que corresponde a un tweet.

Así, cuando llega un tweet, se trata de analizar el texto con el SentimentExtractor, guardar la puntuación como un campo más del documento, e insertar el documento en la base de datos:

```
this.twitterStream.addListener(new StatusListener() {
    @Override
    public void onStatus(Status status) {
        Document tweet = Document.parse(TwitterObjectFactory.getRawJSON(status));
        tweet.append("sentimentScore",
sentimentExtractor.sentimentScoreNormalized(status.getText()));
        db.getCollection(MongoDBHolder.TWEETS_COLLECTION_NAME).insertOne(tweet);
    }
    ...
}
```

Entonces, quedan los métodos “getter” y “setter” para las variables y dos más, uno para iniciar el streaming y otro para detenerlo.

```
public void startDownloadTweetsFromStreaming() {
    db.getCollection(MongoDBHolder.TWEETS_COLLECTION_NAME).drop();
    FilterQuery filterQuery = new FilterQuery(this.searchText);
    this.twitterStream.filter(filterQuery);
}

public void stopDownloadTweetsFromStreaming() {
    this.twitterStream.shutdown();
}
```

Pasando a las otras clases del paquete, TweetStorage es la que realiza las consultas a la base de datos para obtener resultados. Esta clase tiene como variable de instancia un objeto MongoDatabase para la base de datos. Destaca el método findTweets, que permite obtener tweets ordenados por un criterio y con un límite que es la cantidad máxima de resultados. Este método se utiliza para obtener los tweets recientes (ordenados de mayor a menor tiempo), los más positivos (ordenados de mayor a menor por puntuación de polaridad) y los más negativos (ordenados de menor a mayor por polaridad).

```
private List<Document> findTweets(Document sortDoc, int limit) {
    List<Document> results = new ArrayList<>();
    for (Document doc : db.getCollection(MongoDBHolder.TWEETS_COLLECTION_NAME)
            .find()
            .sort(sortDoc)
            .limit(limit)) {
        results.add(doc);
    }
    return results;
}

public List<Document> recentTweets(int limit) {
    return this.findTweets(new Document("timestamp_ms", -1), limit);
}

public List<Document> topPositiveTweets(int limit) {
    return this.findTweets(new Document("sentimentScore", -1), limit);
}

public List<Document> topNegativeTweets(int limit) {
    return this.findTweets(new Document("sentimentScore", 1), limit);
}
```

El método que falta consiste en devolver las cantidades de tweets totales, positivos, negativos o neutrales, cantidades que se agrupan en un objeto de la clase restante TwitterSentimentStats.

```
public TweetSentimentStats sentimentStats() {
    long total = db.getCollection(MongoDBHolder.TWEETS_COLLECTION_NAME).count();
    Document docPositive = new Document("$gt", 0);
    long positive = db.getCollection(MongoDBHolder.TWEETS_COLLECTION_NAME)
            .count(new Document("sentimentScore", docPositive));
    Document docNeutral = new Document("$eq", 0);
    long neutral = db.getCollection(MongoDBHolder.TWEETS_COLLECTION_NAME)
            .count(new Document("sentimentScore", docNeutral));
    Document docNegative = new Document("$lt", 0);
    long negative = db.getCollection(MongoDBHolder.TWEETS_COLLECTION_NAME)
            .count(new Document("sentimentScore", docNegative));
    return new TweetSentimentStats(total, positive, neutral, negative);
}
```

4.5. Servicios REST

Para acabar de ver el backend, nos queda la implementación de los servicios que serán utilizados en el frontend. Hemos utilizado la especificación de Java EE JAX-RS, ayudándonos de los asistentes que incorpora el IDE NetBeans. Para el envío de las respuestas serializando los objetos Java en JSON, hemos añadido la dependencia de la biblioteca Jackson, cuya clase JacksonJsonProvider debe ser llamada en la clase ApplicationConfig. Esta clase se encarga de definir la ruta de los servicios en la URL y añadir las clases que contienen los servicios en sí, que son en nuestro caso MiningService y TweetService. Definimos la ruta donde se encuentran los servicios como “api”, de manera que la raíz de los servicios esté, en el caso de que se despliegue la aplicación en la máquina local en la ruta “tfg”, tal que así: http://localhost:8080/tfg/api.

Veamos ahora la clase MiningService. Esta clase posee una variable estática de clase TweetMiner, que será a la que accederán los servicios. Por lo tanto, en principio solo se crea un único objeto TweetMiner para la aplicación, pudiendo en cada momento tener activo solamente un streaming a la vez. Esto lo hemos planteado así al tratarse de una aplicación prototipo queriendo evitar complicarla administrando credenciales de usuarios externos. Aclarado esto, los métodos de la clase están todos en la ruta “mining”, y tenemos cuatro. El primero se accede añadiendo a la ruta “start” y a través de un método HTTP GET, que sirve para comenzar un análisis pasando como parámetro “search” en la URL el texto de búsqueda. Se llama al TweetMiner para parar el análisis si estuviera activo, modificar la cadena de búsqueda, y llamar al método de inicio del streaming. Así, por ejemplo, si queremos hacer un análisis de los tweets que contengan en su texto “Trump”, habría que hacer una petición GET a la dirección http://localhost:8080/tfg/api/mining/start?search=Trump. Se devuelve un texto plano que indica que se ha comenzado el análisis. Análogamente tenemos un método para detenerlo, en la ruta “stop”, sin parámetros.

```
@GET
@Path("/start")
@Produces(MediaType.TEXT_PLAIN)
public String start(@QueryParam("search") String searchText) {
    tweetMiner.stopDownloadTweetsFromStreaming();
    tweetMiner.setSearchText(searchText);
    tweetMiner.startDownloadTweetsFromStreaming();
    return "Tweets download started, searching \"" + searchText + "\"...";
}

@GET
@Path("/stop")
@Produces(MediaType.TEXT_PLAIN)
public String stop() {
    tweetMiner.stopDownloadTweetsFromStreaming();
    return "Tweets download stopped";
}
```

Los otros dos métodos son para manipular el diccionario de análisis de sentimientos, ambos en la ruta “dictionary”, con el método GET obtenemos los pares palabra-polaridad añadidos por el usuario, utilizando para ello una clase SentimentDictionaryEntry de la que cada objeto contiene una variable para la palabra y otra para la puntuación, de manera que el diccionario devuelto es una lista de objetos de esta clase. Con el método PUT, se envía en el cuerpo de la petición un JSON con las mismas propiedades, que quedará añadido al diccionario o modificado si ya existía. Nótese que no hemos implementado un método DELETE, ya que por simplicidad consideramos que una vez añadida la valoración de una palabra, no tiene sentido que deje de considerarse, a lo sumo se modificaría para darle una puntuación de cero de manera que la palabra sea neutral.

Pasamos a la última clase TweetService, que es la que contiene las operaciones que dan los resultados y estadísticas. En esta clase hay una variable estática de tipo TweetStorage, que es llamada en cuatro métodos GET para las listas de tweets y las estadísticas SentimentStats que se traen de la base de datos. La ruta raíz es “tweets” y la de los métodos son “recent”, “stats”, “top/positive” y “top/negative”.

```
@GET
@Path("/recent")
@Produces(MediaType.APPLICATION_JSON)
public List<Document> recent() {
    return tweetStorage.recentTweets(50);
}

@GET
@Path("/stats")
@Produces(MediaType.APPLICATION_JSON)
public TweetSentimentStats sentimentStats() {
    return tweetStorage.sentimentStats();
}

@GET
@Path("/top/positive")
@Produces(MediaType.APPLICATION_JSON)
public List<Document> topPositiveTweets() {
    return tweetStorage.topPositiveTweets(10);
}

@GET
@Path("/top/negative")
@Produces(MediaType.APPLICATION_JSON)
public List<Document> topNegativeTweets() {
    return tweetStorage.topNegativeTweets(10);
}
```

Para terminar el capítulo, en la Figura 4.3 se puede observar el listado de peticiones disponibles de la API REST implementada, en la herramienta Postman.

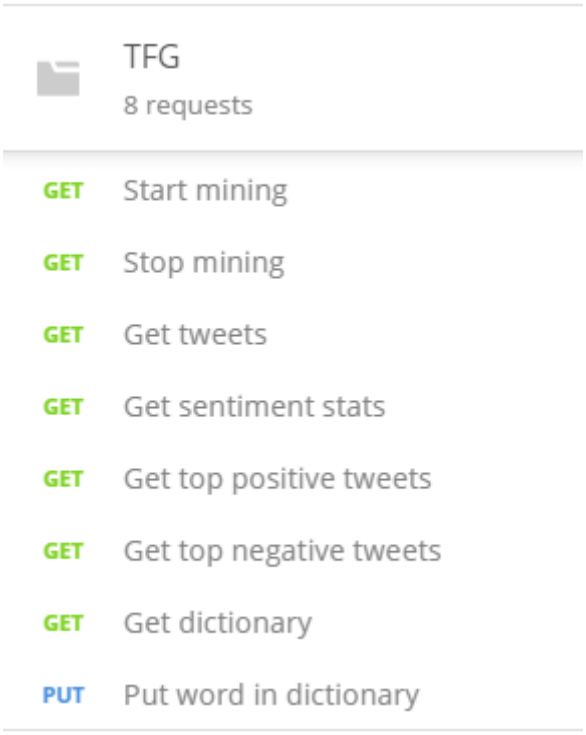


Figura 4.3. Resumen de peticiones a servicios REST en Postman

5. Frontend de la aplicación web

5.1. Esquema general

Al haber desarrollado una arquitectura RESTful, queda separado el backend, es decir, los servicios, del frontend, en este caso las páginas, scripts y estilos que se ejecutan en el lado del cliente. Por lo tanto, cabría la posibilidad de desarrollar las vistas que queramos en proyectos independientes utilizando cualquier tecnología capaz de manejar peticiones HTTP, lo que abre nuestra aplicación a ampliaciones futuras como puede ser una aplicación móvil nativa. Como venimos indicando, el objetivo se estableció en una interfaz web, habiendo elegido utilizar HTML5, Javascript con jQuery y CSS con Bootstrap. Se valoró en su momento crear la interfaz con algún framework más sofisticado como Angular, pero finalmente se mantuvo lo inicialmente planteado, siendo, por decirlo así, un desarrollo de frontend a bajo nivel. Utilizando Bootstrap hemos mantenido un diseño responsive, es decir, adaptable a todo tipo de pantalla, de manera que la apariencia sea válida para móviles (véase Figura 5.2).

Más concretamente, hemos planteado el frontend como una Single Page Application, es decir, tenemos una única página o vista que nunca se recarga por completo, solamente se actualizan las partes que el usuario requiera asíncronamente. Así, está página la tenemos en un único fichero index.html. La lógica Javascript para realizar las peticiones AJAX a los servicios según las acciones sobre la página y actualizar la misma con los resultados está en el fichero script.js. Por último, los estilos CSS particulares están en el fichero style.css. Los ficheros CSS y Javascript de Bootstrap y jQuery se cargan en el HTML desde CDN, esto es, alojados en servidores externos conocidos de manera que se aproveche la velocidad de los mismos y la caché. Otros ficheros que utilizamos son imágenes, para el logo de la aplicación y el icono. Todos estos ficheros están organizados en la carpeta webapp del mismo proyecto Maven donde está el backend, aunque como decíamos bien podría haber sido un proyecto independiente. Veamos la estructura de la página en sí.

Figura 5.1. Apariencia de la página en la pestaña Home

La página se estructura de la siguiente manera. Tenemos una barra superior en la que está el formulario para iniciar y detener los análisis. Luego, tenemos una serie de botones a modo de pestañas, que cambian el contenido principal de la página, pero siempre manteniéndose la barra superior. Cada pestaña es una funcionalidad diferenciada en la aplicación. Esto está planteado así para tener un diseño modular, en el que si añadimos servicios al backend se puedan añadir las vistas correspondientes en nuevas pestañas sin interferir en las otras, lo que ha sido desde nuestro punto de vista una buena decisión que ha favorecido el desarrollo.

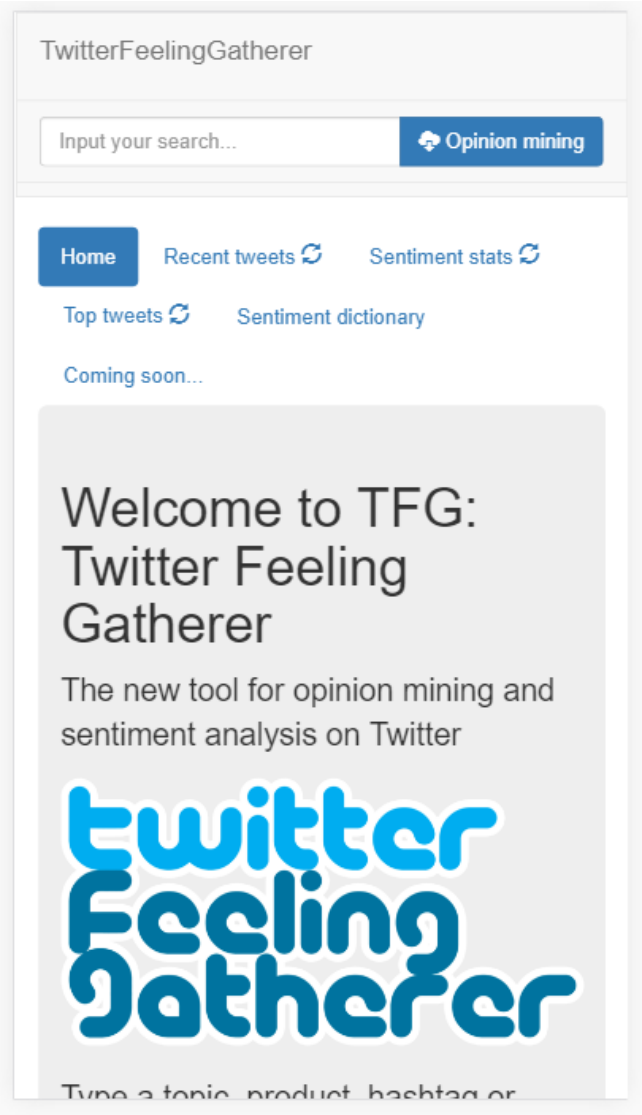


Figura 5.2. Apariencia de la página en una pantalla de móvil

Enumeramos las pestañas, que luego describimos: “Home”, “Recent tweets”, “Sentiment stats”, “Top tweets”, “Sentiment dictionary” y “Coming soon”. La de “Home” sirve de página inicial en la que se muestra el logo y un pequeño texto de bienvenida. La última de “Coming soon” está para indicar posibles funcionalidades que se puedan añadir más adelante.

5.2. Iniciar y detener un análisis

Para iniciar un análisis el usuario debe escribir su búsqueda en el campo de la barra superior y hacer clic en el botón de “Opinion Mining”. Si todo va bien, el botón cambia de apariencia con un icono de “stop”, que se ha de pulsar para detener el análisis en funcionamiento. Las funciones Javascript encargadas de hacer el cambio y la petición se asignan como manejadores del evento “click” en el HTML del botón.

```
function startOpinionMining() {
    var searchText = $("#search-text").val();
    if (!searchText || searchText.length < 3) {
        showError("Search text should be at least 3 characters long");
    } else {
        $.ajax({
            type: "GET",
            url: baseUrl + "/mining/start",
            data: {
                search: searchText
            },
            success: function () {
                $("#opinion-mining-icon").removeClass("glyphicon glyphicon-cloud-
download");
                $("#opinion-mining-icon").addClass("glyphicon glyphicon-stop");

                $("#opinion-mining-button").removeClass("btn btn-primary");
                $("#opinion-mining-button").addClass("btn");
            }
        });
    }
}
```

El código para la función de detener el análisis es análogo.

5.3. Tweets recientes

La primera pestaña después de la de “Home”, sirve para cargar los últimos tweets que se hayan recibido del streaming, con una etiqueta de su puntuación en rojo si es negativa, en gris si neutral y en verde si positiva. Los tweets cargados son añadidos al contenido de la pestaña mediante una función que escribe el HTML necesario para cada tweet, con las clases CSS adecuadas. La función que hace la petición y luego carga los datos utilizando esa, se establece como manejador del clic al enlace de la pestaña. De esta manera, cada vez que se cambia a esta pestaña o se pincha en su título, los tweets se actualizan.

```
function loadRecentTweets() {
    $.ajax({
        type: "GET",
        url: baseUrl + "/tweets/recent",
        dataType: "json",
        success: function (data) {
            showTweets(data, $("#recent-tweets-content"));
        }
    });
}
```

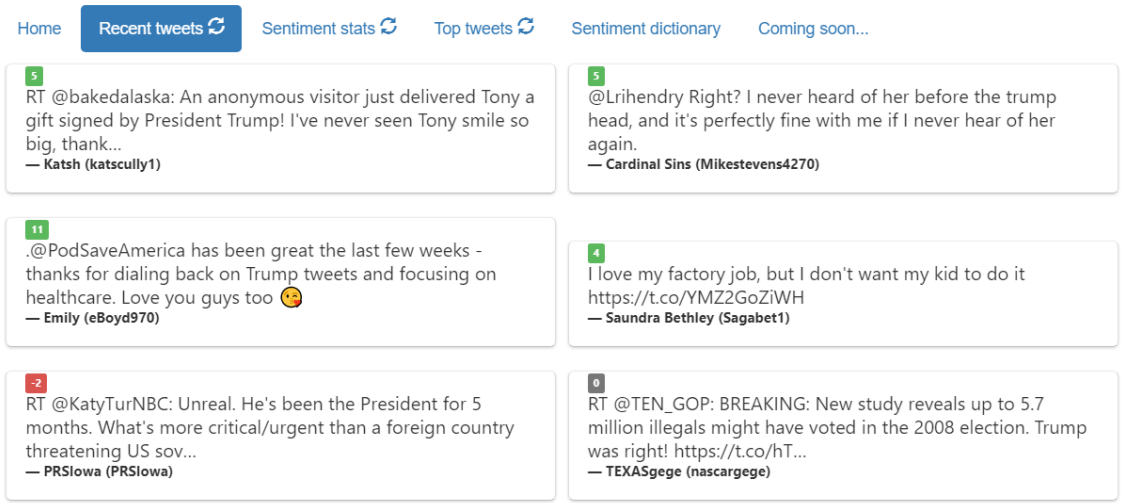


Figura 5.3. Ejemplos de tweets cargados en la pestaña Recent tweets

5.4. Estadísticas

La pestaña de “Sentiment stats” trata de mostrar los números de tweets positivos, negativos, neutrales y totales junto con sus porcentajes y representados gráficamente en una barra. Igual que en la pestaña anterior, tenemos una función que se encarga de escribir en el DOM los datos formateados adecuadamente que son recibidos en una petición, realizada en una función que se asigna al evento de clic en la pestaña.

```
function loadSentimentStats() {
    $.ajax({
        type: "GET",
        url: baseUrl + "/tweets/stats",
        dataType: "json",
        success: function (data) {
            showSentimentStats(data);
        }
    });
}
```

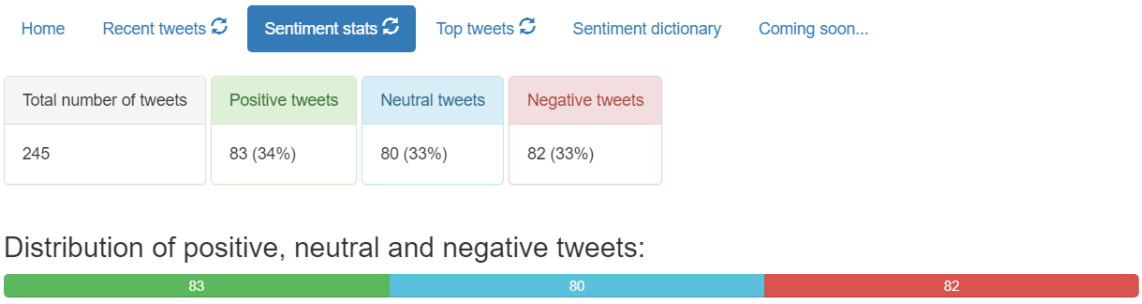


Figura 5.4. Ejemplo de estadísticas en la pestaña Sentiment stats

5.5. Top de tweets

La pestaña “Top tweets” es bastante similar a la de “Recent tweets”, cambiando que se realizan dos peticiones, una para cargar los tweets más negativos y otra para los más positivos, y que dichos tweets se disponen en dos columnas separadas. La función para mostrar los tweets en el contenido es la misma.

```
function loadTopTweets() {
    $.ajax({
        type: "GET",
        url: baseUrl + "/tweets/top/positive",
        dataType: "json",
        success: function (data) {
            showTweets(data, $("#top-positive-tweets-content"));
        }
    });
    $.ajax({
        type: "GET",
        url: baseUrl + "/tweets/top/negative",
        dataType: "json",
        success: function (data) {
            showTweets(data, $("#top-negative-tweets-content"));
        }
    });
}
```

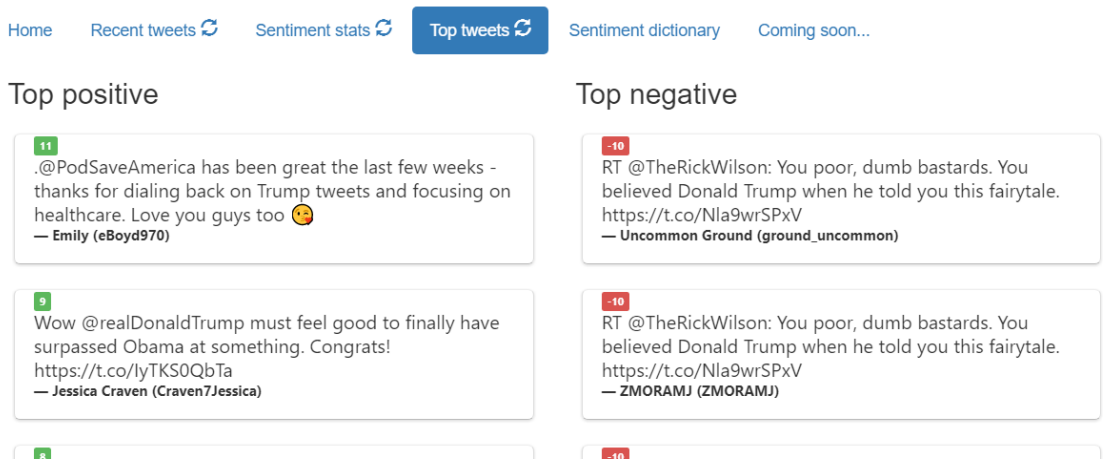


Figura 5.5. Ejemplo de la pestaña Top tweets

5.6. Diccionario de análisis de sentimientos

Llegamos a la última funcionalidad, que es quizás la más novedosa y compleja de la aplicación: el diccionario personalizado de análisis de sentimientos. En esta pestaña “Sentiment dictionary” tenemos primero un formulario para introducir palabras con puntuaciones, y luego una tabla que lista las entradas que hemos introducido. Por tanto, hay dos peticiones a los servicios, una del formulario para las entradas, y otra para cargar la tabla. La función que realiza la primera se asocia al clic del botón del formulario, mientras que la de la segunda queda asociada al clic en la pestaña y también se llama una vez introducida una nueva entrada.

```
function putInSentimentDictionary() {
    var word = $("#dictionary-put-word").val();
    var sentiment = parseFloat($("#dictionary-put-sentiment").val());
    if (!word || isNaN(sentiment) || sentiment < -5 || sentiment > 5) {
        showError("Empty word or invalid sentiment score, "
                + "sentiment should be between -5 and 5");
    } else {
        $.ajax({
            type: "PUT",
            url: baseUrl + "/mining/dictionary",
            contentType: "application/json",
            data: JSON.stringify({
                word: word.trim(),
                sentiment: sentiment
            }),
            success: function () {
                loadSentimentDictionary();
            }
        });
    }
}
```

```
function loadSentimentDictionary() {
    $.ajax({
        type: "GET",
        url: baseUrl + "/mining/dictionary",
        dataType: "json",
        success: function (data) {
            showSentimentDictionary(data, $("#tbody-sentiment-dictionary"));
        }
    });
}
```

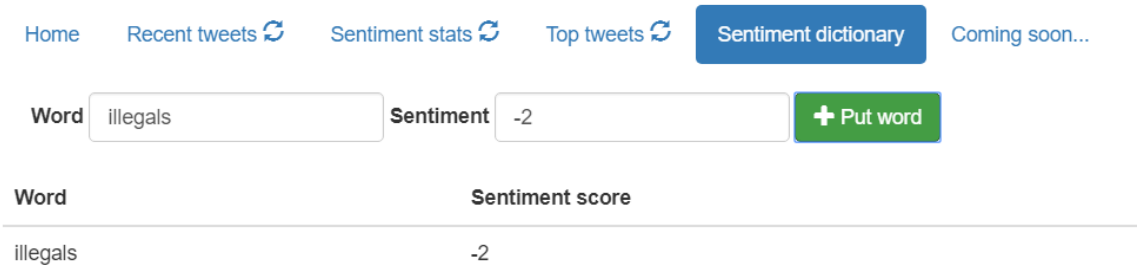


Figura 5.6. Ejemplo de la pestaña Sentiment dictionary

Las palabras añadidas aquí serán utilizadas en los tweets que se analicen desde ese momento. Un detalle interesante que hemos implementado para facilitar la introducción de palabras es una función Javascript que captura el evento del doble clic en el texto de los tweets, de manera que, si queremos añadir una palabra de un tweet al diccionario, haciendo doble clic en ella se vuelca en el formulario de la pestaña del diccionario y se cambia la pestaña activa a esta, tal que solo tenemos que introducir la valoración y pulsar en el botón.

6. Resultados y conclusiones

En este trabajo hemos desarrollado con éxito un prototipo de aplicación capaz de mantener en funcionamiento un análisis de opiniones y sentimientos sobre tweets que se van descargando en segundo plano acerca de un texto introducido. Durante el análisis, que puede durar el tiempo que queramos, se pueden consultar los últimos tweets y su puntuación de polaridad, las estadísticas de cada polaridad respecto del total, y los tweets de máxima y mínima puntuación. Además, si se quiere poner la valoración de palabras a gusto del usuario, es posible a través del diccionario personalizado.

Para terminar esta memoria vamos a comentar algunos aspectos de validación de resultados, errores y problemas, trabajo futuro y conclusiones finales.

6.1. Validación

Normalmente, las herramientas que realizan una clasificación de datos se validan mediante pruebas con unos datos previamente etiquetados, comprobando si los resultados de clasificación coinciden con las etiquetas reales. En este sentido se suele hablar de dos parámetros, que ya hemos comentado en la memoria: la precisión, que es la proporción de datos bien clasificados con una etiqueta respecto de todos los datos clasificados con la misma, y la sensibilidad, que es la proporción de datos bien clasificados con una etiqueta respecto de todos los datos que realmente tendrían esa etiqueta.

Esto se suele aplicar en campos distintos de inteligencia artificial como la visión por computador. Sin embargo, el análisis de sentimientos desde nuestro punto de vista la misma validación no es tan relevante, dado que realizar la clasificación no es tan trivial para un ser humano.

Por ejemplo, si tenemos un programa que tiene que identificar cuántas caras hay en una fotografía, una persona rara vez tendrá dudas en indicar un resultado en el que cualquiera estaría de acuerdo. Decidir si una opinión es positiva o negativa no puede ser igual de sencillo, habiendo factores como la ironía o frases hechas que no todos interpretarán de la misma manera. Por ello, no hemos querido hacer mucho énfasis en realizar una validación de resultados propia, pero no está de más comentar la validación realizada por el autor de la herramienta Sentiment, en la que nos hemos basado para nuestra implementación del análisis de sentimientos.

Para esta validación se utilizan conjuntos de frases previamente etiquetadas, obtenidas desde webs donde los usuarios valoran productos (Amazon), películas (IMDB) y restaurantes (Yelp). Respecto a esos conjuntos se proporcionan los valores de precisión, respectivamente, de un 72%, 79% y 69%. Por comparar con el otro extremo de enfoque de machine learning, Stanford CoreNLP indica en su página web una precisión en torno al 80%.

6.2. Errores y problemas

Durante el desarrollo, como es normal, hemos hecho frente a varios problemas derivados de las tecnologías, que hemos podido resolver gracias a información disponible en la red, destacando la conocida web de StackOverflow. Sin embargo, nos gustaría mencionar un error concreto con la API de Twitter, que en principio escapa a nuestro control. En la prueba de más duración, en la que tuvimos un análisis funcionando durante unas dos horas, se produjo una excepción en la biblioteca de Twitter4J sin más mensaje que “Stream closed”. Según la documentación de Twitter es posible que el streaming se pare por “problemas internos”. Desconocemos la razón real del problema, habría que profundizar en la biblioteca para encontrar el motivo y en cualquier caso implementar una lógica que detecte si es posible volver a conectarse (no ha habido bloqueo por límite, por ejemplo) y en tal caso hacerlo.

Un problema que también hemos visto y habría que afrontar en el futuro, es el manejo de los retweets. Un número considerable de tweets son repetidos de otros usuarios. Hay que contarlos como una opinión más para las estadísticas, pero no deberían aparecer más de una vez en secciones como la de Top tweets.

6.3. Trabajo futuro

Hablando respecto a futuras mejoras o ampliaciones que se puedan realizar, citamos algunos puntos que podría tener una aplicación final desarrollada a partir del prototipo propuesto:

- Autenticación de usuarios: se trata de implementar un sistema que permita a los usuarios autenticarse con su cuenta de Twitter para realizar búsquedas, evitando la situación actual en que solo un streaming puede ser realizado a la vez.
- Diccionario de análisis de sentimientos en español: la biblioteca que hemos desarrollado contiene un modelo de palabras etiquetadas con polaridad, en inglés. Traducirlo y adaptarlo al español con ayuda de alguna herramienta sería interesante.
- Uso de metadatos: los tweets de Twitter poseen muchos metadatos que no hemos considerado, como el idioma o la localización.
- Guardado de estados y gráficas: los análisis se ejecutan durante un tiempo indefinido, pero no se guarda cómo van oscilando los resultados a lo largo del mismo. Guardando esa información se pueden añadir gráficas respecto del tiempo.
- Búsquedas avanzadas: ofrecer los Trending Topics para buscar directamente, alguna interfaz para combinar términos de búsqueda, etc. Aprovechando las cuentas de usuario se podrían filtrar entre seguidores o siguiendo, por ejemplo.

- Mayor control de errores: por ejemplo, implementando una capa de manejo de errores en los servicios REST.
- Pruebas: solamente se han realizado pruebas “a mano”, habría que realizar casos de pruebas unitarias y de integración.
- Despliegue en la nube: una vez terminada la aplicación habría que ponerla disponible a través de Internet, seguramente con algún proveedor cloud.

6.4. Conclusiones finales

El TFG ha resultado ser un reto bastante enriquecedor, ya que hemos tratado de investigar acerca de un campo innovador pero desconocido por nuestra parte como es el análisis de sentimientos dentro del procesamiento del lenguaje natural. De esta manera, quizás el proyecto no es el usual de Ingeniería del Software, ya que no ha habido un planteamiento de una aplicación con una fase de análisis de modelado y especificación de requisitos, sino que se han ido implementando funcionalidades conforme se probaban y estudiaban las posibilidades. Sin embargo, hemos cumplido el objetivo inicial de desarrollar una aplicación que ponga en práctica los conceptos teóricos de análisis de sentimientos. Simple pero funcional, el resultado de la aplicación es bastante satisfactorio.

Referencias bibliográficas

Artículo de Sentiment Analysis de Wikipedia. (s.f.). Obtenido de https://en.wikipedia.org/wiki/Sentiment_analysis

Go, A. B. (2009). Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*.

Nielsen, F. Å. (2011). A new ANEW: Evaluation of a word list for sentiment analysis in microblogs.

Página web de Apache OpenNLP. (s.f.). Obtenido de https://opennlp.apache.org/

Página web de Bootstrap. (s.f.). Obtenido de getbootstrap.com

Página web de Google Cloud API de lenguaje natural. (s.f.). Obtenido de https://cloud.google.com/natural-language/

Página web de jQuery. (s.f.). Obtenido de https://jquery.com/

Página web de la API de Streaming de Twitter. (s.f.). Obtenido de https://dev.twitter.com/streaming/overview

Página web de la API REST de Twitter. (s.f.). Obtenido de https://dev.twitter.com/rest/public

Página web de Maven Repository. (s.f.). Obtenido de https://mvnrepository.com/

Página web de MongoDB. (s.f.). Obtenido de https://www.mongodb.com/

Página web de MonkeyLearn. (s.f.). Obtenido de https://monkeylearn.com/

Página web de Natural Language Understanding (Sentiment) de IBM Watson. (s.f.). Obtenido de https://www.ibm.com/watson/developercloud/doc/natural-language-understanding/#sentiment

Página web de NetBeans. (s.f.). Obtenido de https://netbeans.org/

Página web de Oracle Java EE. (s.f.). Obtenido de http://www.oracle.com/technetwork/java/javaee/overview/index.html

Página web de PlantUML. (s.f.). Obtenido de http://plantuml.com/

Página web de Postman. (s.f.). Obtenido de https://www.getpostman.com/

Página web de Robomongo. (s.f.). Obtenido de https://robomongo.org/

Página web de Sentiment Analysis de la Universidad de Stanford. (s.f.). Obtenido de https://nlp.stanford.edu/sentiment/code.html

Página web de Sentiment140. (s.f.). Obtenido de http://www.sentiment140.com/

Página web de Text Analytics API de Microsoft Azure. (s.f.). Obtenido de https://azure.microsoft.com/es-es/services/cognitive-services/text-analytics/

Página web de Text Processing. (s.f.). Obtenido de http://text-processing.com/docs/sentiment.html

Página web de TheySay PreCeive REST API. (s.f.). Obtenido de http://apidemo.theysay.io/

Página web de Twitter4J. (s.f.). Obtenido de http://twitter4j.org/en/index.html

Pregunta en StackOverflow: Twitter Streaming API limits? (s.f.). Obtenido de https://stackoverflow.com/questions/34962677/twitter-streaming-api-limits

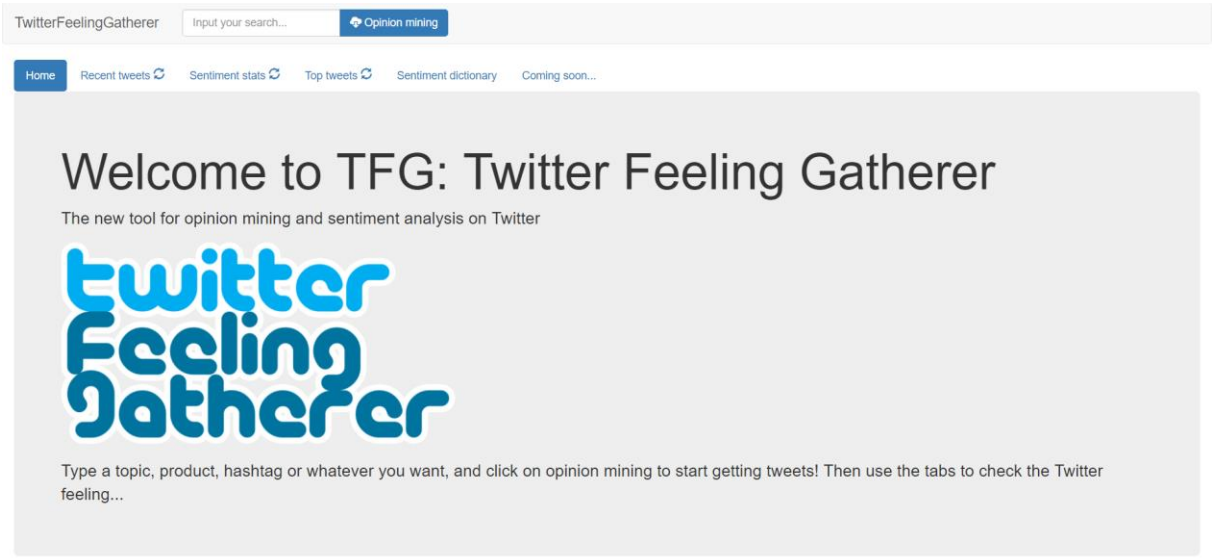
Repositorio en GitHub de sentiment (thisandagain). (s.f.). Obtenido de https://github.com/thisandagain/sentiment

Repositorio en GitHub de SentimentalJ (ukwa). (s.f.). Obtenido de https://github.com/ukwa/SentimentalJ

Socher, R. P. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*.

Anexo I. Manual de usuario (Manual de inicio rápido)

La aplicación se abre inicialmente en la pestaña de Home, siendo este el aspecto:

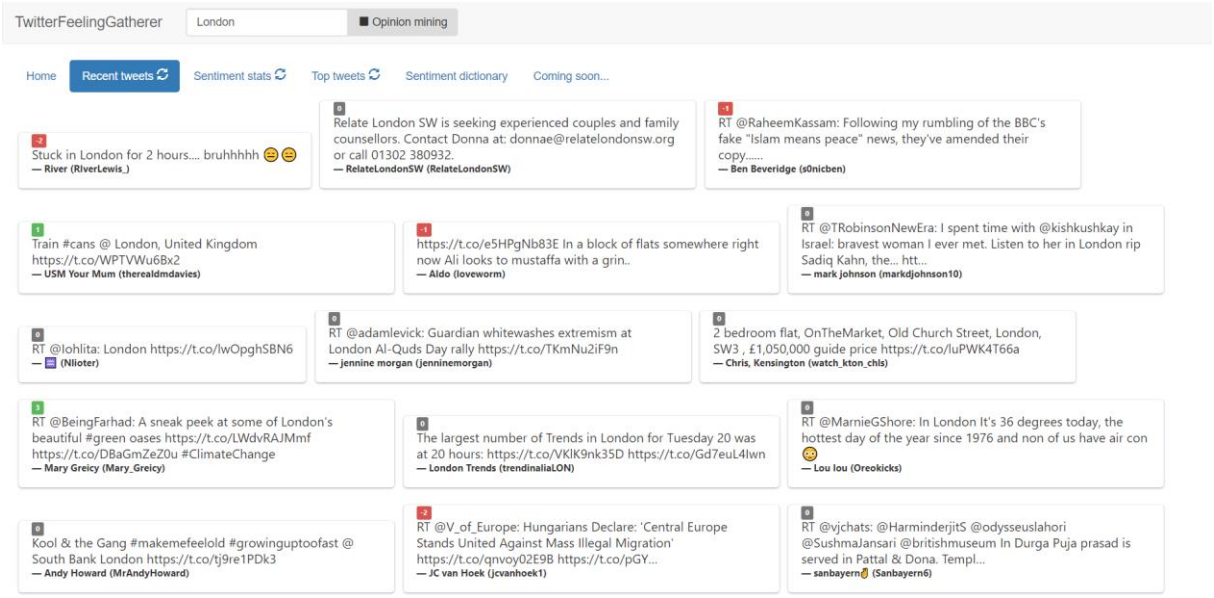


Para comenzar a descargar tweets y hacer análisis de sentimientos, inserte su búsqueda en el cuadro superior y haga clic en Opinion Mining:

Si el proceso se ha iniciado correctamente, el botón cambiará:

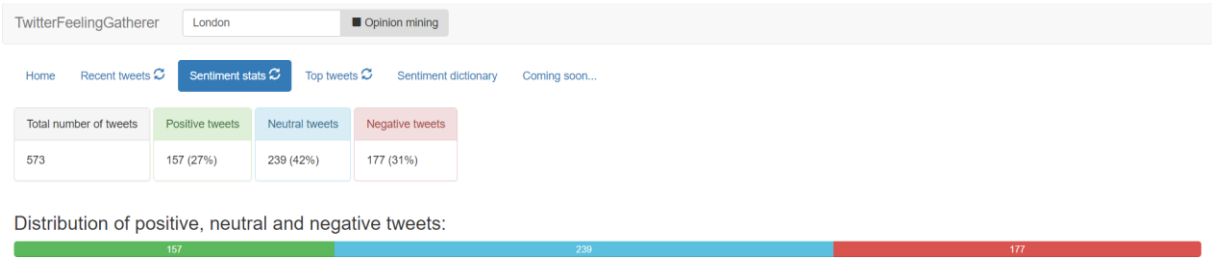
Haciendo clic en el mismo detendrá el análisis. Los resultados se mantendrán mientras no se inicie uno nuevo. Para ver los resultados, haga clic en las pestañas “Recent tweets”, “Sentiment stats” o “Top tweets”.

Si hace clic en Recent tweets, se cargarán los tweets más recientes respecto a la búsqueda introducida, con su puntuación:



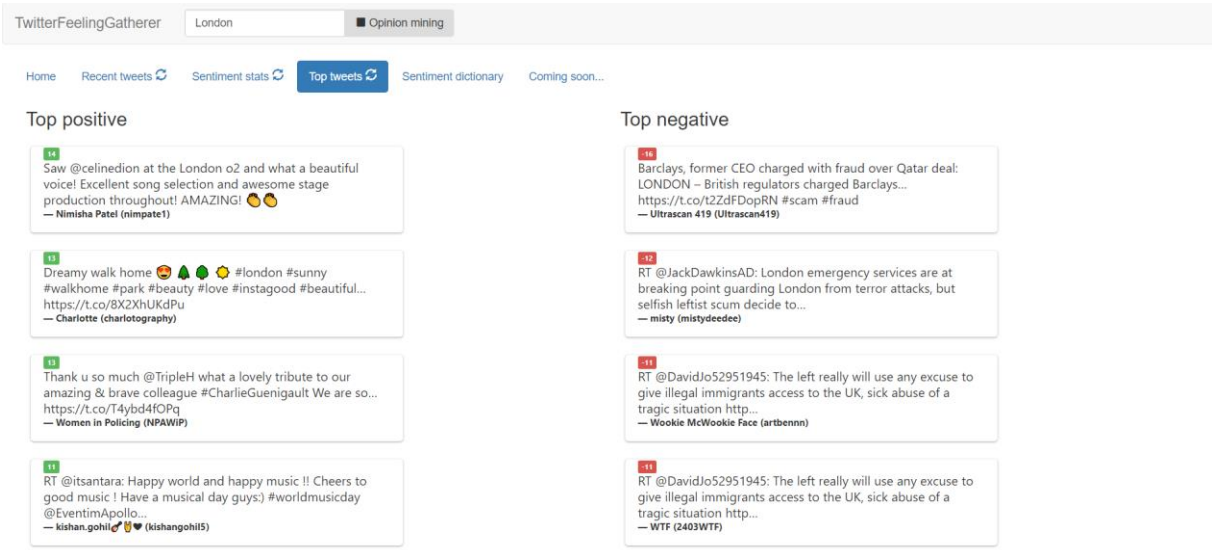
Se cargarán los últimos 50 tweets, con su etiqueta de puntuación en verde para positivo, gris para neutral y rojo para negativo. Si desea actualizar los resultados, haga clic de nuevo en la pestaña.

Para ver estadísticas, haga clic en Sentiment Stats:

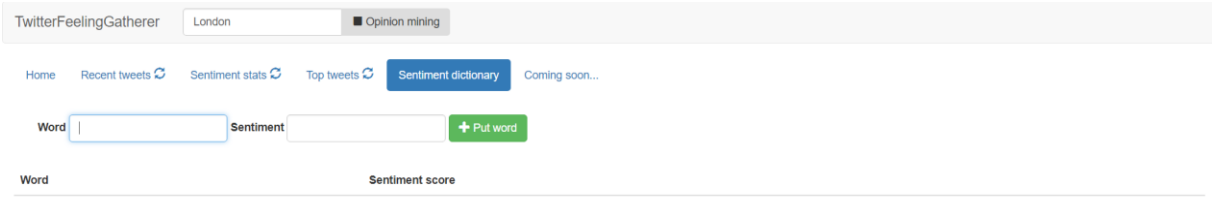


De la misma manera, haga clic de nuevo para actualizar. Aparece el número total de tweets, el número y porcentaje de positivos, neutros y negativos, así como su distribución en una barra.

La pestaña de “Top tweets” funciona como las dos anteriores, actualizándose cada vez que haga clic. Se muestran dos columnas con los tweets más positivos y más negativos:

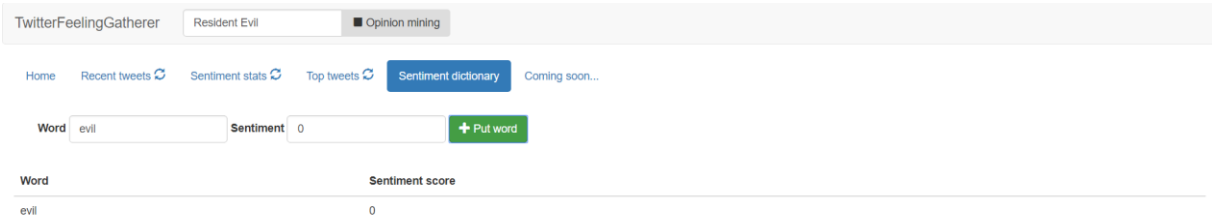


Para añadir sus propias palabras al diccionario de análisis de sentimientos, use la pestaña Sentiment dictionary. Puede añadir una palabra que en el contexto de su análisis tenga una valoración distinta.



Por ejemplo, si quiere buscar opiniones acerca del videojuego “Resident Evil”, la palabra “evil” será detectada como negativa por defecto. Para solucionar esto, introduzca la palabra, la valoración que considere y pulse en Put word:

Las palabras añadidas se mostrarán ahora en la pestaña.



Otra forma muy sencilla de añadir palabras es hacer doble clic en la palabra que quiera cambiar de un tweet de los que aparecen en las otras pestañas. Automáticamente la palabra aparecerá en la pestaña de Sentiment dictionary para que solamente tenga que introducir la valoración y pulsar el botón.

¡Disfrute de la aplicación!

Anexo II. Manual de instalación

Para desplegar la aplicación Twitter Feeling Gatherer en su propio equipo, tiene que usar los proyectos Maven “sentictionary” y “tfg”, que se incluyen adjuntos con esta memoria. El proyecto “sentictionary” incluye la biblioteca de análisis de sentimientos implementada, que se utiliza en el proyecto WAR “tfg” de la aplicación web. Por lo tanto, este primer proyecto debe estar en su repositorio local de Maven antes de cargar el segundo. En principio si se ejecutan en orden no debería haber problema. Es necesaria una conexión a Internet para descargar las dependencias.

El proyecto “sentictionary” incluye una clase SentictionaryDemo con un método main para comprobar su funcionamiento. El proyecto “tfg” genera un fichero con extensión WAR preparado para su despliegue en un servidor de aplicaciones. Aunque en principio se puede realizar el proceso usando la línea de comandos de Maven y un servidor independiente, recomendamos utilizar el IDE NetBeans (versión utilizada 8.2) con el servidor GlassFish instalado desde el mismo (versión utilizada 4.1). En cualquier caso, debe estar instalado MongoDB con una instancia en funcionamiento (comando mongod). Los pasos en estas circunstancias serían los siguientes:

1. Abrir los proyectos desde el diálogo de abrir proyecto del IDE, que deben ser detectados como proyectos Maven.
2. Una vez abiertos, probablemente aparecerá una advertencia de que faltan dependencias. Mediante el diálogo para resolver problemas deben instalarse.
3. Ejecutar, primero en el proyecto sentictionary y luego en el tfg la operación de limpiar y construir o “clean and build”.
4. Pulsar en el botón de ejecutar o “Run” del proyecto tfg. Esperamos unos segundos y si todo va bien se debe abrir el navegador en la dirección http://localhost:8080/tfg/.

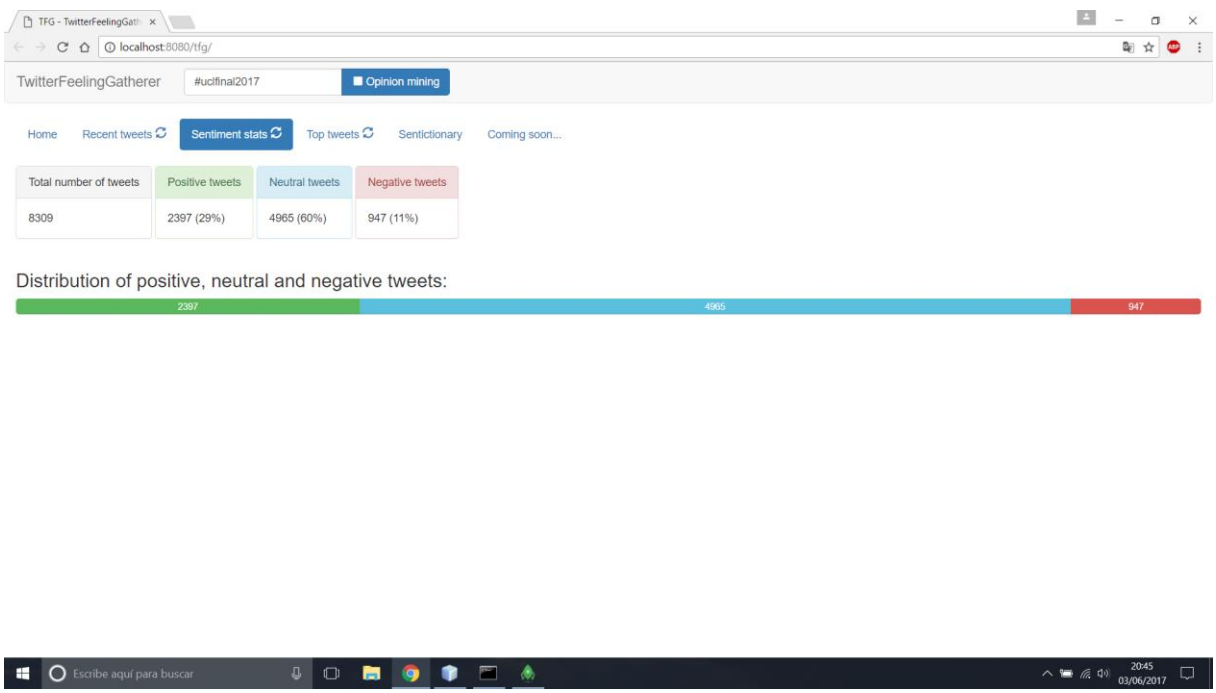
Otras consideraciones:

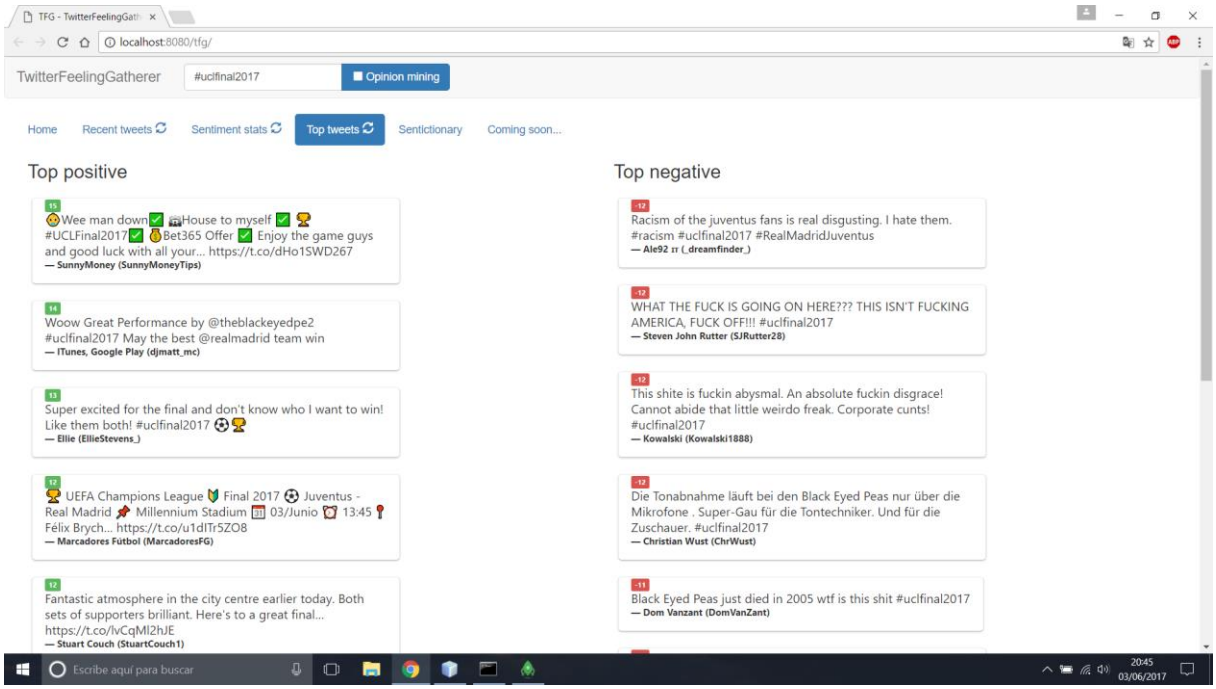
- En caso de usar un servidor proxy HTTP, debe configurarse Maven con él, al igual que debe configurarse para la biblioteca Twitter4J en la clase TweetMiner del paquete tweets en el proyecto tfg. Comentado en el código de dicha clase está la configuración para el proxy de la UMA.
- Si se desean hacer peticiones directamente a la API, se puede usar la herramienta Postman, de la que incluimos un fichero de colección.
- En la clase TweetMiner están las credenciales de OAuth de una aplicación creada con nuestra propia cuenta de Twitter. No se garantiza que estas credenciales estén operativas a largo plazo, y en cualquier caso recomendamos sustituirlas por otras suyas en caso de que se vayan a utilizar más allá de unas pruebas iniciales.

Anexo III. Ejemplo de funcionamiento: Final de la Champions League

Mostramos aquí unas capturas de los resultados de unas pruebas que fueron realizadas el día de la final de la Champions League entre el Real Madrid y la Juventus. Se utilizó el hashtag #uclfinal2017, y se inició el análisis minutos antes del partido, realizando tres capturas de las pestañas de Sentiment stats y Top tweets, al comienzo del partido, en el descanso, y sobre el final del partido, momento en el que por razones que desconocemos se para el streaming de Twitter.

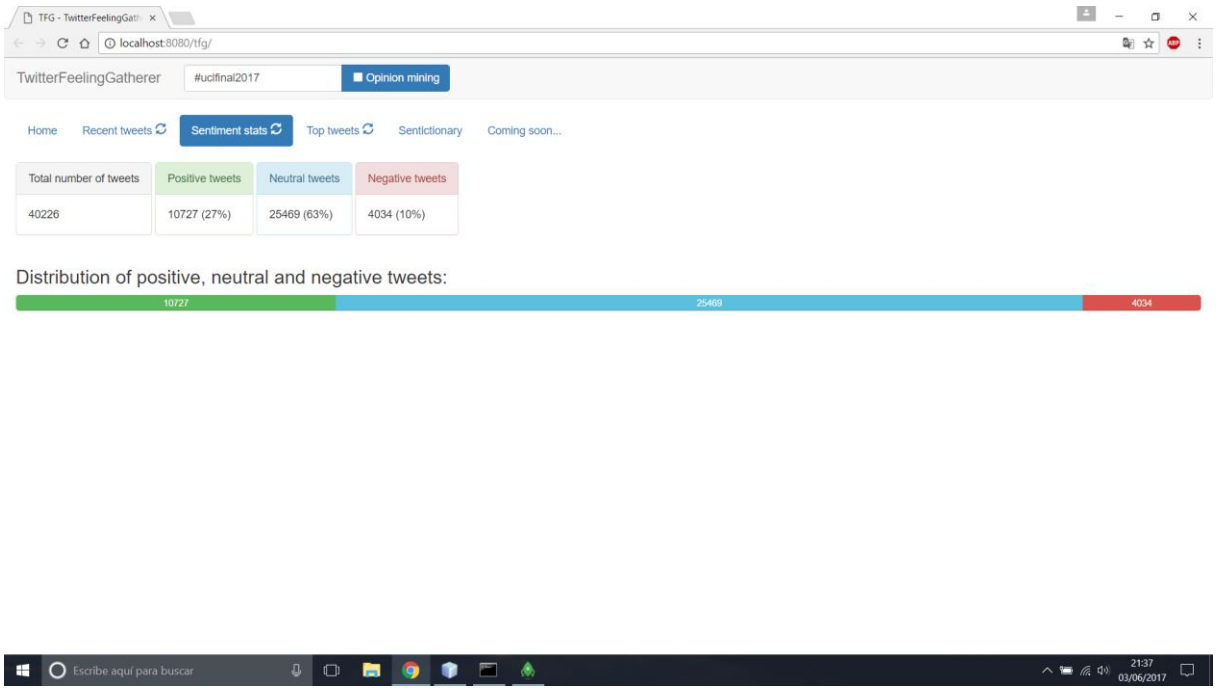
Capturas al comienzo del partido:

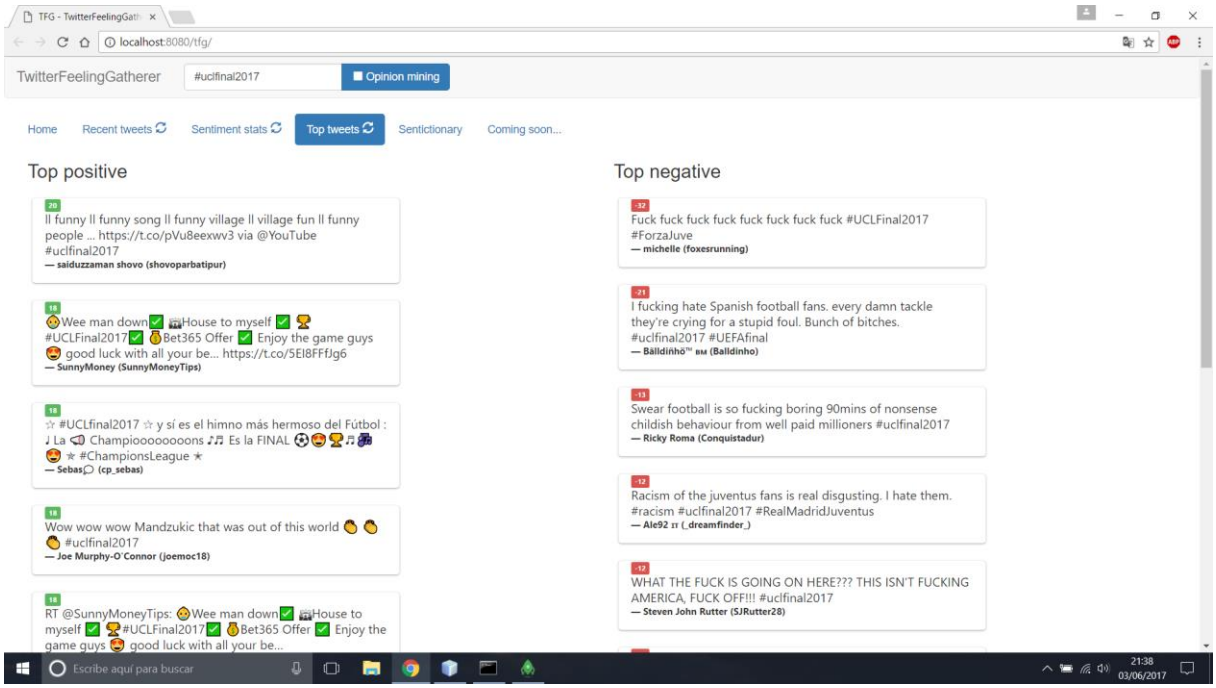




En pocos minutos se descargaron y analizaron más de 8000 tweets. El porcentaje de neutrales es muy superior al resto, en parte debido a la diversidad de idiomas y que solo se detecten palabras en su mayoría en inglés y emojis. Luego, el porcentaje de positivos supera bastante al de negativos, con un 29% y 11% respectivamente, lo que indica que en ese momento los usuarios estaban en una actitud positiva, lo que es razonable en la expectación del partido. En los tops destaca que hay opiniones contrapuestas respecto a la actuación en la previa del grupo musical *Black Eyed Peas*, con presencia en ambas columnas. El comentario más positivo lo es por los emojis y porque desea suerte y que se disfrute el partido. El más negativo hace referencia al racismo de los aficionados.

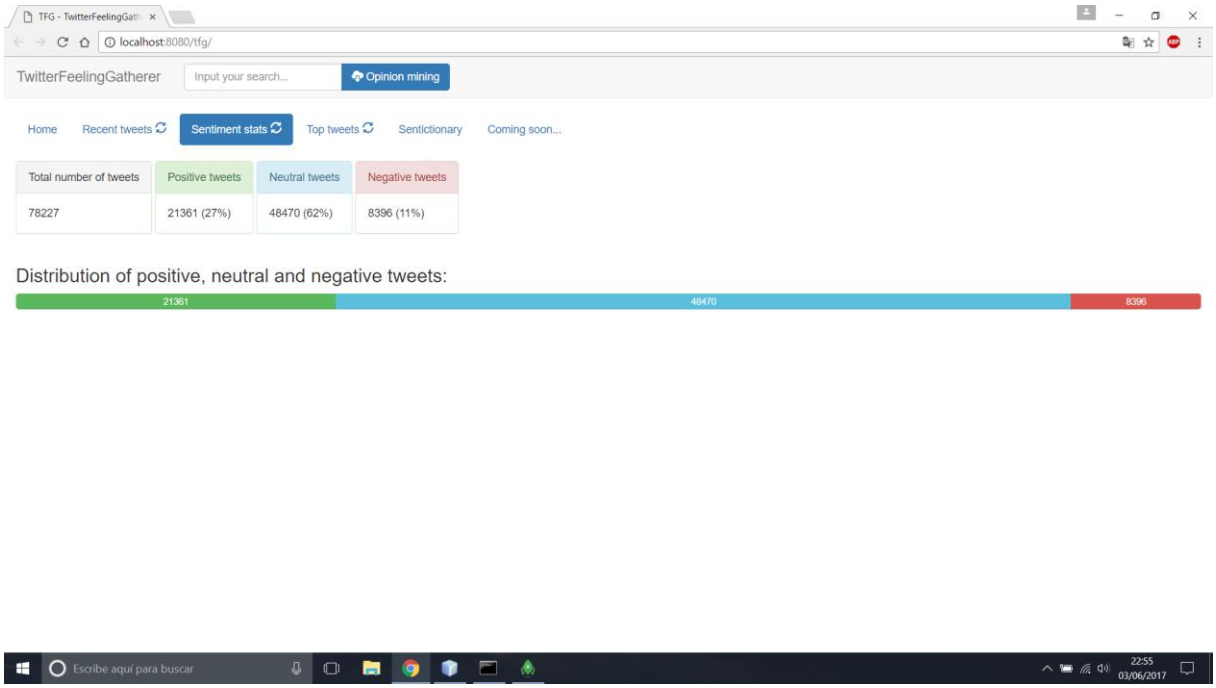
Capturas en el descanso:

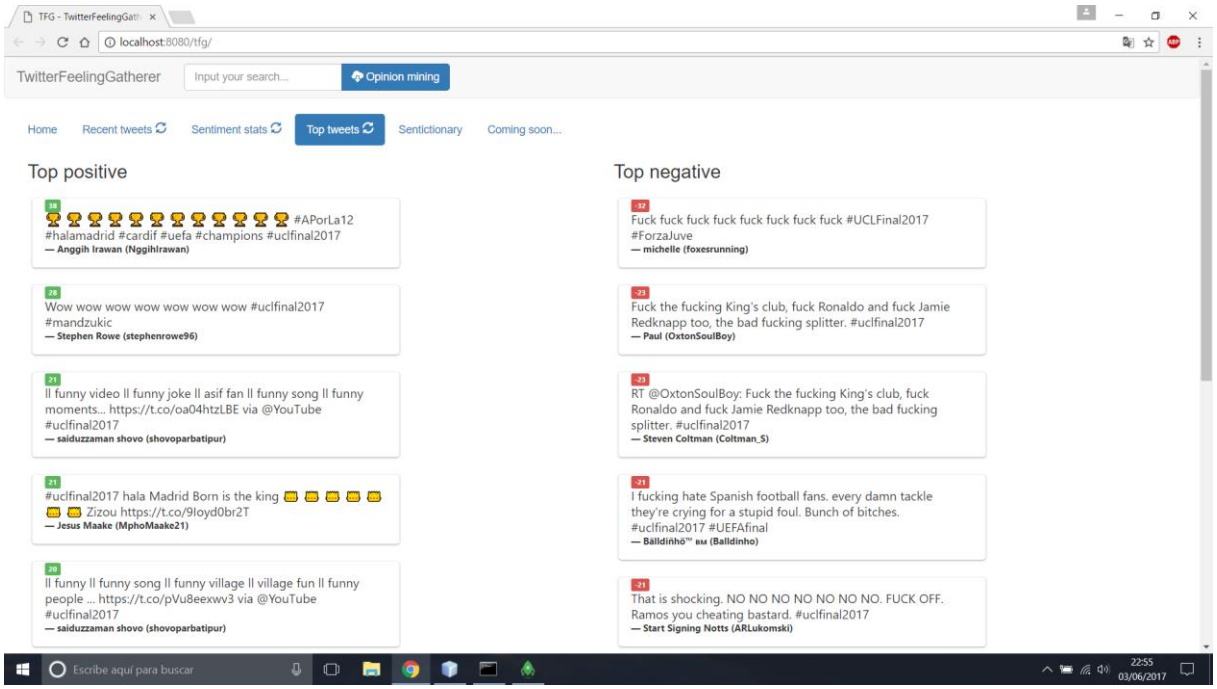




En el descanso la tendencia de las estadísticas era bastante similar, y se han descargado un total de más de 40000 tweets, en unos 50 minutos. En los tops, destaca en el positivo un aficionado feliz por el gol de Mandzukic, y en el negativo destacan un usuario defensor de la Juve que repite la palabra “fuck” seguramente por el gol del Madrid, al igual que otro que insulta a los aficionados españoles.

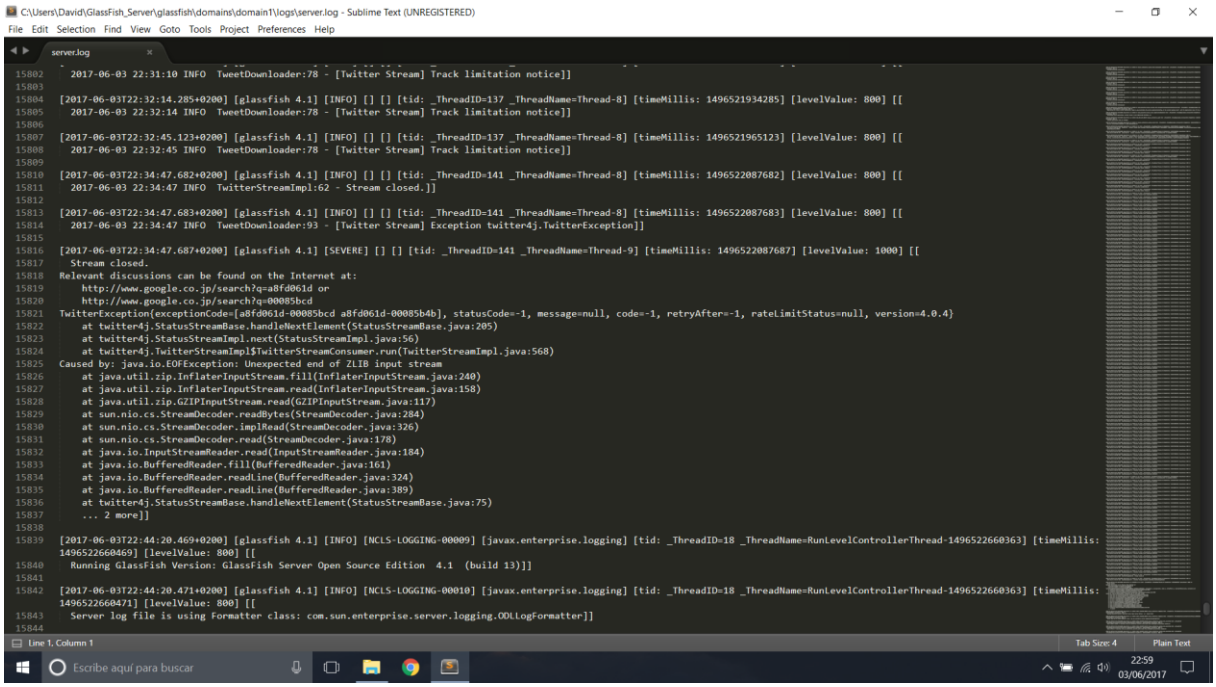
Capturas hacia el final del partido:





Se han descargado un total superior a 70000 tweets, con la tendencia parecida a las anteriores. Destacan en el top positivo comentarios sobre la victoria madridista con 12 emojis de copas y otro que dice que “ha nacido el rey”. En lo negativo aparecen de nuevo algunos insultos, y uno que comenta una acción de Ramos poco antes del final en la que opina que ha hecho trampa, para lograr la expulsión de un rival.

Para terminar, mostramos una captura del registro de GlassFish con la traza del error que detuvo el streaming. A partir de la traza habría que averiguar el motivo del error y detectarlo para reiniciar el streaming en el futuro.


C:\Users\David\GlassFish_Server\glassfish\domains\domain1\logs\server.log - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
server.log
2017-06-03 22:31:10 INFO TweetDownloader:78 - [Twitter Stream] Track limitation notice]]
[2017-06-03T22:32:14.285+0200] [glassfish 4.1] [INFO] [] [] [tid: _ThreadID=137 _ThreadName=Thread-8] [timeMillis: 1496521934285] [levelValue: 800] [[
2017-06-03 22:32:14 INFO TweetDownloader:78 - [Twitter Stream] Track limitation notice]]
[2017-06-03T22:32:45.123+0200] [glassfish 4.1] [INFO] [] [] [tid: _ThreadID=137 _ThreadName=Thread-8] [timeMillis: 1496521965123] [levelValue: 800] [[
2017-06-03 22:32:45 INFO TweetDownloader:78 - [Twitter Stream] Track limitation notice]]
[2017-06-03T22:34:47.682+0200] [glassfish 4.1] [INFO] [] [] [tid: _ThreadID=141 _ThreadName=Thread-8] [timeMillis: 1496522087682] [levelValue: 800] [[
2017-06-03 22:34:47 INFO TwitterStreamImpl:62 - Stream closed.]]
[2017-06-03T22:34:47.683+0200] [glassfish 4.1] [INFO] [] [] [tid: _ThreadID=141 _ThreadName=Thread-8] [timeMillis: 1496522087683] [levelValue: 800] [[
2017-06-03 22:34:47 INFO TweetDownloader:93 - [Twitter Stream] Exception twitter4j.TwitterException]]
[2017-06-03T22:34:47.687+0200] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=141 _ThreadName=Thread-9] [timeMillis: 1496522087687] [levelValue: 1000] [[
Stream closed.
Relevant discussions can be found on the Internet at:
http://www.google.co.jp/search?q=a8fd061d or
http://www.google.co.jp/search?q=00085bcd
TwitterException{exceptionCode=[a8fd061d-00085bcd a8fd061d-00085b4b], statusCode=-1, message=null, code=-1, retryAfter=-1, rateLimitStatus=null, version=4.0.4}
at twitter4j.StatusStreamBase.handleNextElement(StatusStreamBase.java:205)
at twitter4j.StatusStreamImpl.next(StatusStreamImpl.java:56)
at twitter4j.TwitterStreamImpl$TwitterStreamConsumer.run(TwitterStreamImpl.java:568)
Caused by: java.io.EOFException: Unexpected end of ZLIB input stream
at java.util.zip.InflaterInputStream.fill(InflaterInputStream.java:240)
at java.util.zip.InflaterInputStream.read(InflaterInputStream.java:158)
at java.util.zip.GZIPInputStream.read(GZIPInputStream.java:117)
at sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:284)
at sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:326)
at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:178)
at java.io.InputStreamReader.read(InputStreamReader.java:184)
at java.io.BufferedReader.fill(BufferedReader.java:161)
at java.io.BufferedReader.readLine(BufferedReader.java:324)
at java.io.BufferedReader.readLine(BufferedReader.java:389)
at twitter4j.StatusStreamBase.handleNextElement(StatusStreamBase.java:75)
... 2 more]]
[2017-06-03T22:44:20.469+0200] [glassfish 4.1] [INFO] [NCLS-LOGGING-00009] [javax.enterprise.logging] [tid: _ThreadID=18 _ThreadName=RunLevelControllerThread-1496522660363] [timeMillis: 1496522660469] [levelValue: 800] [[
Running GlassFish Version: GlassFish Server Open Source Edition 4.1 (build 13)]]
[2017-06-03T22:44:20.471+0200] [glassfish 4.1] [INFO] [NCLS-LOGGING-00010] [javax.enterprise.logging] [tid: _ThreadID=18 _ThreadName=RunLevelControllerThread-1496522660363] [timeMillis: 1496522660471] [levelValue: 800] [[
Server log file is using Formatter class: com.sun.enterprise.server.logging.ODLLogFormatter]]
Line 1, Column 1
Tab Size: 4
Plain Text
Escribe aquí para buscar
22:59
03/06/2017