Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

MONITOR DE OPINIONES

Alumno: José Carlos Martínez Cazalilla

Tutor: Luis Alfonso Ureña López

Eugenio Martínez Cámara

Dpto: Ingeniería Informática

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Departamento de Informática

Don L. Alfonso Ureña y Don Eugenio Martínez Cámara , tutor y cotutor del Proyecto Fin de Carrera titulado: Monitor de opiniones, que presenta José Carlos Martínez Cazalilla, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

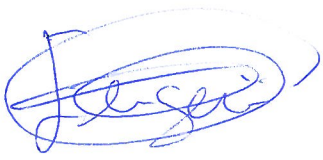
Jaén, Octubre de 2015

El alumno:

José Carlos Martínez Cazalilla

Los tutores:

L.Alfonso Ureña López



Eugenio Martínez Cámara

Índice

Capítulo 1

Introducción

Este capítulo presenta una breve introducción a la memoria del trabajo final de Grado en Ingeniería Informática de la Universidad de Jaén. En él, se pone de manifiesto las razones de la elección del Análisis de Opiniones como objeto de estudio.

1. Introducción

La web ha evolucionado paulatinamente desde sus inicios en 1990, siendo más notable su gran desarrollo en la última década con la universalización de su uso. Hoy ya se puede decir que Internet ha supuesto una verdadera revolución social, y que incluso ha cambiado nuestra forma de vida, de trabajo, y nuestras relaciones sociales. Pero el número de usuarios no debe ser el único factor para determinar el espectacular progreso de Internet, sino también la ingente cantidad de información que circula por la red, que generan y comparten millones de personas en todo el mundo.

Esto propició la aparición de la llamada La Web 2.0, que no es más que la evolución de la Web o Internet en el que los usuarios dejan de ser usuarios pasivos para convertirse en usuarios activos, que participan y contribuyen en el contenido de la red siendo capaces de dar soporte y formar parte de una sociedad que se informa, comunica y genera conocimiento.

La Web 2.0 está permitiendo la aparición de todo tipo de comentarios, experiencias, y opiniones de usuarios sobre cualquier tema. Además, no sólo se escriben, sino que también, cada vez es mayor el número de personas que consultan las opiniones de otros usuarios sobre un determinado servicio antes de solicitarlo. Según un estudio del año 2008 (Horrigan, 2008) el 81% de los usuarios de Internet han realizado, al menos una vez, alguna indagación o exploración online sobre algún producto, y el 20% lo hace a diario. En el mismo estudio de Horrigan (Horrigan, 2008) se pone de manifiesto que las opiniones no solamente son leídas, sino que influyen en la decisión de los futuros consumidores, ya que el 80% de los usuarios que han revisado opiniones sobre restaurantes, hoteles y otros servicios de este tipo, aseguran que las opiniones leídas influyen significativamente en sus decisiones.

Encontrar fuentes de información y monitorizar su evolución en la Web es una tarea muy compleja debido a la gran cantidad de fuentes diferentes, y al gran volumen de textos con opiniones que tiene cada una, complicándose aún más cuando las opiniones no se encuentran expresadas de forma explícita. Esta gran cantidad de información hace que sea muy difícil para un lector humano encontrar y

seleccionar las opiniones presentes en Internet. Debido a esto se ve necesario el desarrollo de sistemas automáticos de búsqueda, extracción, clasificación y presentación de opiniones. La disciplina Minería de Opiniones (*Opinion Mining*) también conocida como Análisis de Sentimientos (*Sentiment Analysis*) surge para dar solución a este problema tan complejo. El Análisis de Sentimientos se encuadra dentro del área del Procesamiento del Lenguaje Natural, la cual se detallará más adelante.

Las opiniones y comentarios vertidos en Internet no tienen restricción en cuanto al idioma utilizado. La gran mayoría de la investigación llevada a cabo relacionada con la Minería de Opiniones se centra casi exclusivamente en textos escritos en inglés. Sin embargo, cada vez son más los textos subjetivos que utilizan otras lenguas como ruso, alemán, árabe… De hecho, el aumento de páginas webs en otros idiomas diferentes al inglés en los últimos años ha sido exponencial. Si bien el inglés es la lengua predominante en Internet, hay otros idiomas como el chino o el español que cada vez tienen más presencia en la Red (Figura 1.1). Así pues, la investigación en Análisis de Sentimientos no se debería centrar exclusivamente en un idioma sino que tendría también que estudiar otras lenguas, e incluso, el Análisis de Sentimientos debería llegar a aplicarse en un ámbito multilingüe.

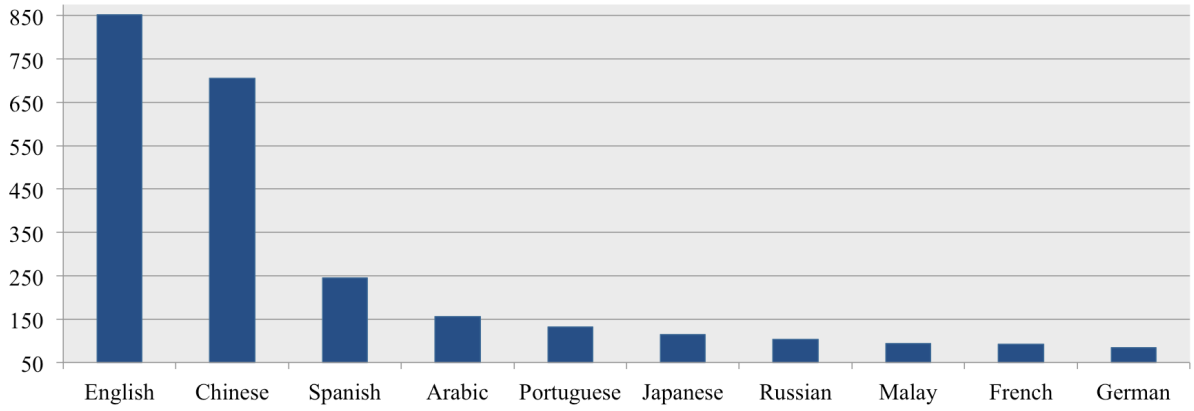


Figura 1.1: Presencia de los 10 primeros idiomas en Internet.

1.2- Motivación

A raíz de lo anteriormente expuesto, presento a continuación las razones que me han llevado a elegir esta rama del conocimiento para realizar mi trabajo final de grado:

1. El Análisis de Sentimientos es una tarea relativamente reciente, que necesita aún un gran esfuerzo investigador para completar la tarea y dar así una respuesta ante las nuevas necesidades de los usuarios. Un ejemplo de la importancia de esta tarea, es el dato de que al menos entre 20 y 30 empresas ofrecen servicios de Análisis de Sentimientos en España (por ejemplo, WebSays) y aun más en los Estados Unidos (por ejemplo, IBM Social Media Analytics).

2. La información no solo se encuentra expresada en una lengua, y menos si nos referimos a productos comerciales. Por lo tanto considero que es muy importante el estudio de técnicas de Minería de Opiniones en otros idiomas cuya presencia en Internet es cada día mayor. El español es una de esas lenguas, y su estudio no solo tiene una importancia investigadora, sino también comercial debido al continuo aumento de la comunidad hispanohablante, la cual solicita cada vez más servicios en español.

3. La web no ha parado de evolucionar en estos últimos años, y ya no es estática, sino vertiginosamente dinámica. Las redes sociales han hecho que la información fluya en tiempo real. Un usuario se compra un móvil, y expresa su opinión sobre la compra en cualquiera de las redes sociales en las que tiene abierta una cuenta. Esta información no puede ser desperdiciada por el mundo empresarial, por lo que son necesarios sistemas que en tiempo real puedan proporcionar la tendencia de opinión de los usuarios.

1.3- Objetivos

Como punto de partida, los objetivos que se plantean para este trabajo son:

- Desarrollar un sistema orientado a la monitorización de la opinión que se publica en la Web en tiempo real.
- Realizar una revisión bibliográfica de la documentación necesaria para el desarrollo del trabajo.
- Analizar y seleccionar el método más adecuado de determinación de la orientación de la opinión.
- Redactar la memoria asociada al sistema.

Capítulo 2

Estado del Arte

En este capítulo se presenta el estado en el que se encuentra el campo del Análisis de Sentimientos.

2.1- Procesamiento del lenguaje natural.

Antes de describir qué es el Análisis de Sentimientos o la Minería de Opiniones, es recomendable conocer la rama de conocimiento a la que pertenece dentro de las ciencias de la computación. La Minería de Opiniones es una de las infinitas aplicaciones del Procesamiento del Lenguaje Natural (PLN), que será brevemente descrito en los siguientes párrafos.

El Procesamiento de Lenguaje Natural (PLN) es la disciplina de la Inteligencia Artificial encargada del análisis de texto. A pesar de que no existe un consenso sobre su definición, se podría definir de la forma siguiente: Conjunto de técnicas informáticas para el análisis y representación de lenguaje natural en uno o más niveles de análisis lingüístico, con el propósito de procesar lenguaje humano aplicable a un extenso conjunto de tareas.

Como ocurre con otras áreas de la ciencia, el PLN tiene un origen multidisciplinar, siendo las siguientes áreas de conocimiento las que más han aportado al PLN:

1. **Lingüística o lingüística computacional**: Centrado en el desarrollo de modelos lingüísticos formales y estructurados.
2. **Ciencias de la computación**: Su misión es el desarrollo de modelos y estructuras para representar los datos, así como mecanismos eficientes de procesamiento de dichas estructuras de representación de la información.
3. **Psicología cognitiva**: Su objetivo es la creación de modelos lingüísticos desde un punto de vista psicológico.

El principal objetivo del PLN es llegar al entendimiento de lenguaje natural, pero esto todavía no se ha conseguido. Un sistema completo de entendimiento de lenguaje natural debería ser capaz de:

1. Parafrasear un texto.
2. Traducir un texto a otro idioma.
3. Responder a preguntas sobre el contenido de un texto.
4. Inferir conocimiento de un texto.

Mientras que la investigación en PLN ha dado pasos para conseguir los objetivos 1-3, todavía no se puede decir que se hayan conseguido resultados relevantes de inferencia de conocimiento a partir de texto, por lo que se puede concluir que el desafío del PLN sigue siendo el entendimiento de lenguaje natural.

2.1- Análisis de Opiniones.

Como punto de partida en el Análisis de Opiniones podemos tomar la definición de Pang y Lee enunciada en (Pang & Lee, 2008), ésta es la más seguida por la comunidad investigadora en Análisis de Opiniones

"Tratamiento computacional de opiniones, sentimientos y subjetividad en textos."

Pero Cambria & Hussain (2012) tras considerar la definición de Pang y Lee algo general, propusieron una nueva definición que intenta detallar algo más la tarea del Análisis de Opiniones.

"Conjunto de técnicas computacionales para la extracción, clasificación, comprensión y evaluación de opiniones expresadas en fuentes publicadas en Internet, comentarios en portales web y en otros contenidos generados por usuarios."

Según (Dave, Lawrence, & Pennock, 2003) el concepto de minería de opiniones se refiere a "procesar un conjunto de resultados de búsqueda de un producto determinado, generando una lista de atributos, a los que se les asocias las opiniones sobre cada uno de ellos", esta fue la primera vez que apareció el termino Minería de Opiniones publicado en las actas de la Conferencia WWW de 2003. Muchas de las investigaciones posteriores han utilizado el término minería de opiniones para referirse a la extracción y análisis de opiniones sobre distintas características de productos.

El uso de la denominación Análisis de Sentimientos ha seguido una evolución paralela a la de minería de opiniones. La primera vez que se utilizó en el contexto del análisis automático de juicios de valor fue en 2001 en (Das & y Chen, 2001; Das & y Chen, 2001) y (Tong, 2001), debido al interés (Das & y Chen, 2001)de los

autores en el estudio de sentimientos en los mercados. En 2002 se publican (Turney, 2002) (Pang, Lee, & Vaithyanathan, 2002) y (Pang, Lee, & Vaithyanathan, Thumbs up? Sentiment Classification using Machine Learning Techniques, 2002) dos trabajos de máxima importancia en los que se utiliza Análisis de Sentimientos como forma para nombrar a esta tarea. Con el paso de los años el uso del término Análisis de Sentimientos se ha centrado en la clasificación de la polaridad de opiniones y críticas, aunque actualmente se utiliza para referirse al tratamiento computacional de opiniones, sentimientos y subjetividad en texto.

Como se puede apreciar, al inicio, Minería de Opiniones y Análisis de Sentimientos tenían matices distintos, pero conforme maduraba la investigación en esta disciplina se han ido utilizando ambos nombres para referirse a lo mismo.

Dentro del Análisis de Sentimientos se profundizará más en la clasificación de textos que expresan opinión. Ésta se divide en dos tipos :

- Binaria, establece solo dos clases para la clasificación (positivo o negativo).
- Grado de positividad, establece varios niveles de positividad (neutra, buena, muy buena,…)

Aunque en principio podamos pensar que el grado de positividad nos proporcionaría más precisión a la hora de clasificar, el objetivo final es ajustarse a la clasificación natural de una opinión (buena o mala). En consecuencia, el uso de la clasificación binaria simplifica la polarización y por este motivo ha sido la opción escogida.

Adelantando en cierta manera la exposición del diseño de la solución, el sistema que se desarrollará se circunscribirá en el contexto de la clasificación binaria de la opinión.

Capítulo 3

Planificación

3.1- Estimación de tiempos

A continuación se muestra una estimación del tiempo de duración de cada actividad respecto al trabajo:

Actividad	**Duración**
Búsqueda bibliográfica	7 días
Estudio de las tecnologías	12 días
Análisis	7 días
Diseño	7 días
Implementación	14 días
Realización de pruebas	7 días
Finalizar memoria del trabajo	14 días
Tiempo total	**68 días**

3.2- Diagrama de Gantt

Para obtener una representación visual de las tareas a realizar en el proyecto se ha creado un diagrama de Gantt usando Microsoft Project.

El resultado ha sido el que se muestra en la figura 2.

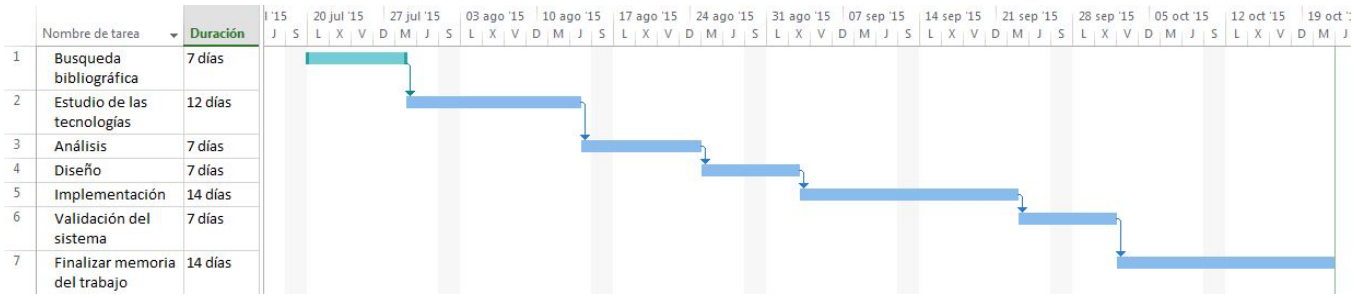


Figura 3.2: Diagrama de Gantt

Como se puede observar en el diagrama el trabajo comenzará el 20 de Julio de 2015, estimando una jornada laboral de 8 horas sin contar fines de semana, éste finalizará el día 20 de Octubre de 2015, haciendo un total de 68 días trabajados.

Capítulo 4

Introducción a la Ingeniería del Software

4.1- Introducción

Para la solución del problema planteado se proponen varias vías, las cuales se desarrollan a continuación.

La primera solución sería realizar una aplicación de escritorio. Éstas tienen la ventaja de ser muy rápidas debido a que el acceso a datos locales y el mayor aprovechamiento de la CPU hacen que la velocidad de la aplicación solo dependa del ordenador. Por el contrario, el acceso a una aplicación escritorio se ve muy limitado, ya que cada usuario que necesite nuestra aplicación tendría que instalarla localmente en su dispositivo, siendo esta característica inapropiada por la naturaleza de nuestro trabajo.

Otra opción es la de crear una aplicación web, donde la principal ventaja es la limitación de la opción anterior, siendo multiplataforma y accesible desde cualquier dispositivo, siempre y cuando se disponga de conexión a internet, por lo que, se considera esta opción como clara candidata para la solución.

Por último, se considera la vía de utilizar una plataforma en la nube ya existente. Ésta mantiene las ventajas de una aplicación web y además le añade otras como puede ser la reutilización de código (facilitando el trabajo de implementación), brindándonos un servidor más potente y económico.

Finalmente, se toma la tercera propuesta como la más óptima, utilizando una metodología de desarrollo basada en componentes orientada a la reutilización.

Se ha elegido este modelo de proceso por la gran ventaja de reducir la cantidad de software a desarrollar y, por lo tanto, la de disminuir costes y riesgos. Asumiendo la pequeña pérdida de control sobre la evolución del sistema que puede conllevar que los componentes reutilizables no estén bajo nuestro control.

Considero que antes de profundizar en el análisis o diseño es necesario conocer tanto la plataforma escogida, como el modelo de desarrollo utilizado ya que la estructura de la plataforma es la base de la elección de este modelo de desarrollo.

4.2- Plataforma de desarrollo

La plataforma escogida para llevar a cabo esta solución ha sido **Sinfonier**[1]. Sinfonier es una comunidad de cooperación en inteligencia de seguridad que se centra en el procesamiento y recolección de nueva información facilitando la reutilización del trabajo ya realizado por la comunidad. Para compartir trabajos se tiene que tener en cuenta que es necesaria una validación por parte de los administradores de la plataforma.

La principal ventaja por la que se ha escogido esta plataforma es que ofrece un soporte en "tiempo-real" para el procesamiento de la información que se recolecta desde diferentes fuentes.

Sinfonier está construida sobre un framework en tiempo real llamado Apache Storm. Éste es un sistema libre y de código abierto que sirve para recuperar streams de datos en tiempo real desde múltiples fuentes de manera distribuida, tolerante a fallos y en alta disponibilidad. Storm está principalmente pensado para trabajar con datos que deben ser analizados en tiempo real, por ejemplo, datos de sensores que se emiten con una alta frecuencia o datos que provengan de las redes sociales donde a veces es importante saber qué se está compartiendo en este momento.

Se compone de dos partes principalmente. La primera es la que se denomina "Spout" y es la encargada de recoger el flujo de datos de entrada. La segunda se denomina "Bolt" y es la encargada del procesado o transformación de los datos. A esto Sinfonier le añade una capa de abstracción agregando otro componente denominado "Drain" que se encarga de volcar los datos procesados a cualquier sistema de almacenamiento o de visualización.

Para facilitar la experiencia del usuario, Sinfonier dota a este sistema de una interfaz gráfica "drag & drop" la cual ofrece la posibilidad de crear nuevas topologías que incluirían los componentes mencionados anteriormente.

4.2 Modelo de desarrollo basado en componentes

Así una vez entendido cómo funciona la plataforma escogida se ha considerado utilizar un modelo de desarrollo basado en componentes ya que es el modelo que más se ajusta al sistema debido a su principal característica basada en la cooperación entre usuarios.

Este modelo permite reutilizar piezas de código pre-elaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión (Casal Terreros).

En la figura 2 se puede ver un modelo del proceso general para el desarrollo basado en componentes, dividido por etapas:

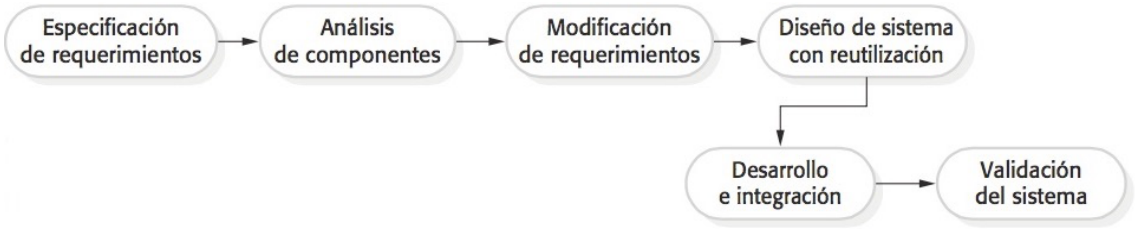


Figura 4.3: Ciclo de vida del modelo de desarrollo basado en componentes

Estas etapas se describen más detalladamente a continuación.

1. **Especificación de requerimientos.** En esta etapa se realizará un estudio de las tareas específicas que el software ha de realizar y el modo de funcionamiento de cada una de ellas durante la interacción con el usuario.
2. **Análisis de componentes.** Una vez establecidos los requisitos, se realiza una búsqueda de componentes para implementar dicha especificación.
3. **Modificación de requerimientos.** Durante esta etapa se vuelve a revisar los requerimientos pero, a diferencia de la primera etapa, se tiene en cuenta la información de los componentes descubiertos, luego se modifican para reflejar los componentes disponibles.

4. **Diseño de sistema con reutilización.** En esta fase se diseña el marco conceptual del sistema teniendo en cuenta los componentes que se reutilizarán.
5. **Desarrollo e integración.** A continuación, se implementa el software que no puede procurarse de forma externa, e integraremos los componentes para crear el nuevo sistema.
6. **Validación del sistema.** Por último se realizarán diversas pruebas para verificar el correcto desempeño del sistema implementado.

Capítulo 5

Especificación de requerimientos

El objetivo de esta fase es reflejar todos los requerimientos necesarios para el desarrollo del trabajo.

5.1- Requisitos funcionales

A continuación se definen las funcionalidades que se deben ofrecer a los usuarios; es decir, los requisitos funcionales o cómo debe comportarse el sistema ante las diferentes entradas dadas por los usuarios.

Los requerimientos funcionales que tiene que cumplir el sistema a desarrollar son los siguientes:

- Obtener mensajes de la red social Twitter relacionados con una consulta definida por un usuario.
- Determinar la orientación de la opinión de los mensajes obtenidos en una escala de intensidad de opinión de dos niveles: Positivo y Negativo.
- Visualizar los resultados de clasificación obtenidos.

5.2- Requisitos no funcionales

A diferencia de los anteriores, con los requisitos no funcionales se busca especificar los criterios que puedan juzgar la operación del sistema y no comportamientos específicos.

Así pues, los requerimientos que están relacionados con la calidad del sistema en desarrollo y no con la funcionalidad que ofrece para nuestro caso son:

- Clasificar los mensajes extraídos en tiempo real.
- El sistema deberá trabajar ininterrumpidamente veinticuatro horas al día.
- Optimizar la tarea de clasificación, aprovechando los recursos al máximo.
- Garantizar la privacidad de los usuarios que utilicen nuestro sistema.
- Notificar errores que puedan surgir instantáneamente al usuario.

Capítulo 6

Análisis de componentes

Antes de elegir que componentes se necesitan para nuestro sistema, se estudiará cómo se va a enfocar el problema.

6.1 Análisis

A continuación, se muestran tanto el diagrama frontera como los distintos casos de uso que se les puede presentar al usuario al enfrentarse a nuestro sistema.

Diagrama frontera

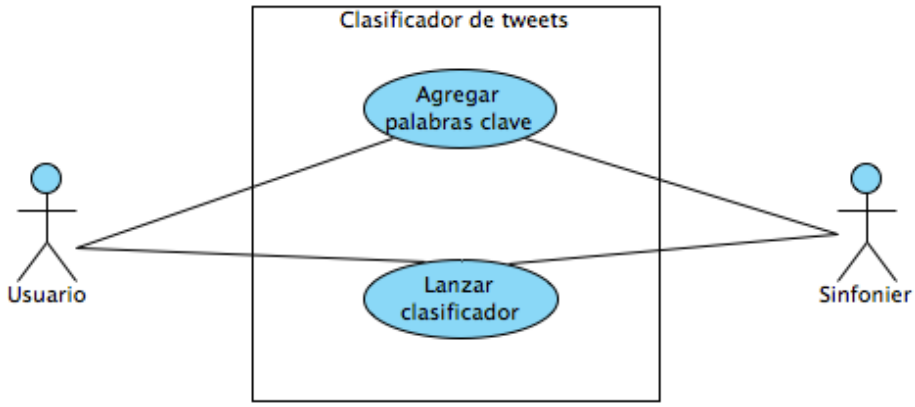


Figura 6.4: Diagrama de frontera

Agregar palabra clave

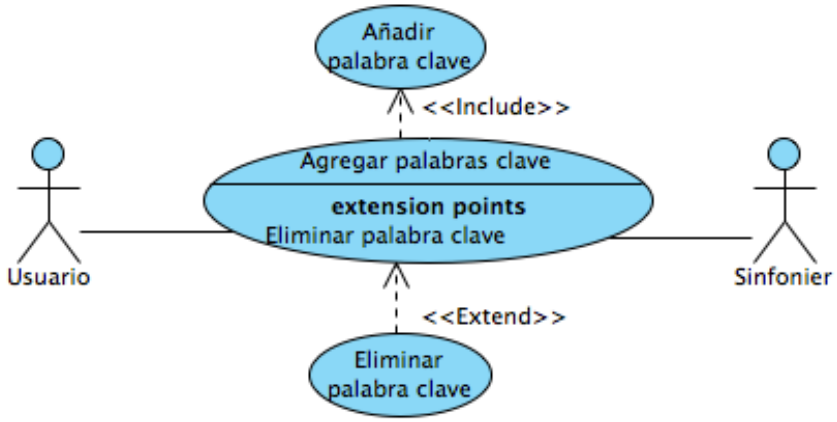


Figura 6.5: Caso de uso 1

Caso de uso 1	
Nombre	Agregar palabra clave
Descripción	Añade las palabras necesarias para la extracción de tweets
Actores	Usuario, Sinfonier
Precondiciones	Ninguna
Flujo de eventos	4. Escribe la palabra 5. Registrar la palabra insertada en el Paso 1
Post-condiciones	Ninguna

Tabla 6.1: Caso de uso 1

Lanzar clasificador

Figura 6.6: Caso de uso 2

Caso de uso 2	
Nombre	Lanzar clasificador
Descripción	Pone en marcha todo el proceso de extracción y clasificación de tweets
Actores	Usuario, Sinfonier
Precondiciones	Agregar al menos una palabra clave
Flujo de eventos	1. Accionar el proceso de clasificación
Post-condiciones	Ninguna

Tabla 6.2: Caso de uso 2

Una vez analizados los casos de uso necesarios para la puesta en marcha del sistema, llegamos a la conclusión de que necesitamos dos componentes: uno que sea capaz de recoger las palabras claves dadas por el usuario y extraiga los tweets y otro para la clasificación de estos tweets. A su vez, también se necesitaría otro componente que muestre los resultados obtenidos ya sea en una interfaz gráfica o simplemente en un log que nos ofrece la plataforma Sinfonier.

Sinfonier dispone tanto del primer componente (un componente de tipo Spout denominado 'Twitter') como del último (Sinfonier cuenta con varios componentes de tipo Drain que permiten realizar visualizaciones en diferentes plataformas), por lo que solo se necesita implementar un componente encargado de clasificar dichos tweets el cual se desarrolla posteriormente.

Capítulo 7

Modificación de requerimientos

Una vez obtenidos los componentes necesarios y comprobados cuáles podemos reutilizar, pasamos a analizar los requisitos obtenidos en la primera etapa usando la información de los nuevos componentes descubiertos.

Requisitos funcionales:

1. Obtener mensajes de la red social Twitter relacionados con una consulta definida por un usuario.
2. Determinar la orientación de la opinión de los mensajes obtenidos en una escala de intensidad de opinión de dos niveles: Positivo y Negativo.
3. Visualizar los resultados de clasificación obtenidos.

Requisitos no funcionales:

1. Clasificar los mensajes extraídos en tiempo real.
2. El sistema deberá trabajar ininterrumpidamente veinticuatro horas al día.
3. Optimizar la tarea de clasificación, aprovechando los recursos al máximo.
4. Garantizar la privacidad de los usuarios que utilicen nuestro sistema.
5. Notificar errores que puedan surgir instantáneamente al usuario.

Para los requisitos funcionales 1 y 2 se reutilizarán componentes ya existentes en la plataforma ya que como se comentó en el capítulo anterior, Sinfonier dispone de varios módulos que realizan tanto la tarea de recolección de tweets como la de visualización de resultados. El cumplimiento del resto de los requisitos no funcionales nos lo garantiza el propio uso de Sinfonier, excepto el tercer requisito que dependerá de nuestro módulo a desarrollar.

El primer requisito no funcional, como se ha comentado anteriormente, lo cubre Sinfonier ya que ofrece un soporte en “tiempo-real” para el procesamiento de la información que se recolecta desde diferentes fuentes. El segundo requisito, en principio es asumido por la plataforma. En el caso en el que la plataforma no estuviese disponible no estaría dentro de nuestro alcance solucionar ese problema. Respecto al cuarto requisito puesto que el registro de los usuarios será llevado a cabo a través de Sinfonier, ésta garantiza la privacidad de sus datos. Por último el

quinto requisito también correrá a cargo de la plataforma ya que dispone de su propio sistema de gestión de errores.

En cambio se ha detectado un nuevo requisito derivado del uso de esta plataforma. Éste es:

1. Los datos necesarios para la ejecución del sistema serán almacenados en la nube, sin posibilidad de guardar nada localmente.

Una vez ajustados los requerimientos pertinentes para nuestro sistema, podremos pasar al diseño del sistema.

Capítulo 8

Diseño de sistema con reutilización

En este apartado se analizan los requisitos para producir una descripción de la estructura interna del software que sirva de base para su construcción.

8.2- Diagrama de despliegue

La intención de la realización de este diagrama de despliegue es la de poner de manifiesto la unión entre las distintas plataformas que trabajarán conjuntamente para alcanzar el objetivo del trabajo: Twitter, Sinfonier y una interfaz de visualización. En él se muestra la disposición física de los artefactos software que componen nuestro sistema, así como, los vínculos entre las diferentes plataformas necesarias para la resolución del problema.

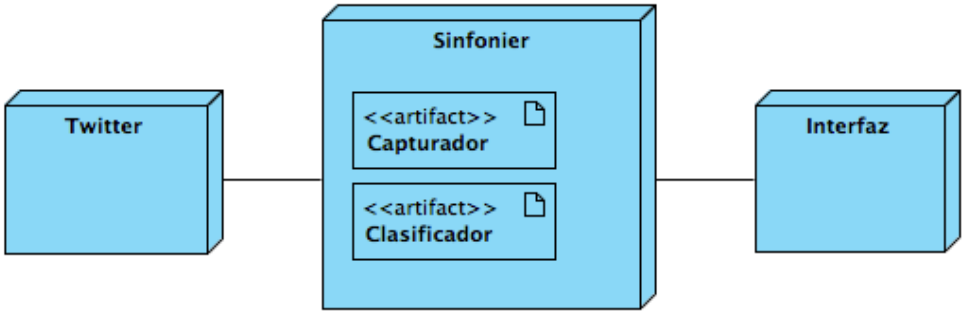


Figura 8.7: Diagrama de despliegue

Como se puede observar en la figura 8.7 en primer lugar, los servidores de Twiter serían los encargados de abastecer a Sinfonier. Éste a su vez tendrá la tarea de extraer esos tweets y clasificarlos para que finalmente sean la entrada de una interfaz que mostraría los resultados obtenidos.

8.2 Diagrama de clases

Cabe destacar, que al utilizar una plataforma ya existente, ésta ya tiene una estructura predefinida, por lo que tenemos que ajustarnos y construir nuestro sistema bajo las pautas que marque. Por este motivo no disponemos de una total libertad para diseñar nuestra arquitectura del sistema.

El desarrollo en Sinfonier está bastante limitado ya que al desarrollador se le impone una plantilla. Ésta dependerá del tipo de modulo que se esté desarrollando (spout, bolt o drain) y estará compuesta por una única clase que recibirá el mismo nombre que el módulo. El hecho de que se imponga esta plantilla nos conduce a

estructurar todo nuestro esquema simplificándolo hasta llegar a tener una sola clase que englobe todos nuestros métodos.

El módulo implementará una clase abstracta proporcionada por la plataforma obligando al desarrollador a implementar tres métodos de vital importancia para el funcionamiento de éste. Estos módulos serán explicados posteriormente con más detalle.

Como se comentó en el apartado 6.1, en principio solo se necesita construir un módulo encargado de realizar la clasificación de los tweets recogidos. El diseño de este módulo bajo la plataforma Sinfonier se muestra en la figura 8.8.

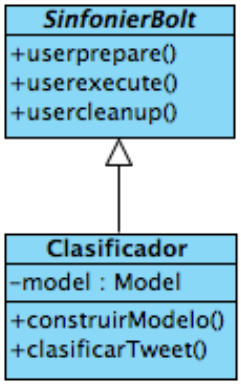


Figura 8.8: Diagrama de clases

Nuestro módulo sería la clase que representamos con el nombre de “Clasificador”. Esta clase implementaría a la clase abstracta “SinfonierBolt” proporcionada por la plataforma y, como explicaremos más adelante, contendrá los métodos necesarios como “construirModelo” y “clasificarTweet” para realizar su función de clasificación.

8.2- Elección del algoritmo de clasificación

En el artículo (Martínez Cámara, Martín Valdivia, Ureña Lopez, & Mitkov, 2015) se analiza un corpus (COST) obtenido de Twitter. La principal característica de este corpus es que todos los tweets que lo componen están escritos en español, y son clasificados en dos clases (positivo y negativo), aplicando distintos algoritmos de clasificación. Entre ellos, SVM, LR (regresión logística) y NB (Naïve Bayes)

concluyendo este articulo tras realizar varias pruebas de configuración de estos algoritmos que la mejor opción es utilizar un algoritmo SVM.

Puesto que el corpus utilizado para este análisis se asemejará al que utilizaremos en nuestra propuesta, podremos servirnos de él para predecir qué algoritmo obtendrá mejores resultados en nuestro caso.

Así pues, tomando como referencia los resultados mostrados en el artículo anterior usaremos una máquina de soporte vectorial (SVM) como algoritmo de clasificación.

8.2.1- Maquinas de soporte vectoriales (SVM)

Este algoritmo está propiamente relacionado con problemas de clasificación y regresión. El algoritmo recibe como entrada un conjunto de ejemplos de entrenamiento previamente etiquetados (es decir, ya clasificados según las clases requeridas) . El objetivo es “entrenar” con estos datos un modelo que prediga la clase de una nueva muestra distinta a las muestras de entrenamiento.

El modelo SVM representa las muestras como puntos en el espacio, donde lo ideal será separar los puntos de distintas clases por un espacio lo más amplio posible. Así, si una nueva muestra se pone en correspondencia con dicho modelo, en función de su proximidad con las muestras ya entrenadas podrá ser clasificada como una u otra clase.

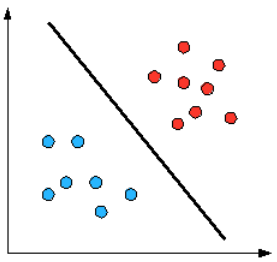


Figura 8.10: Representación gráfica SVM

Para conseguir este modelo se construye un hiperplano que divide los puntos dados en una clase o en otra, por esto, los puntos del vector que son etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado como se muestra en la figura 8.10.

Capítulo 9

Desarrollo e integración

9.1 Introducción

Una vez entendido cómo funcionan las máquinas de soporte vectoriales, se toma la decisión de realizar dos módulos en nuestro proyecto. El objetivo es ofrecer al usuario diferentes posibilidades de utilizar dicha máquina. Esto dará un valor añadido a este trabajo, ya que tras el estudio realizado previamente de la realización de servicios parecidos al objetivo de este proyecto, se ha constatado que no abundan las empresas que permiten entrenar al usuario su propio modelo, sino que parten de un corpus previamente tratado.

Por un lado se desarrollaría un módulo capaz de clasificar tweets a través de un algoritmo SVM previamente entrenado con el corpus mencionado anteriormente. A este corpus se le realizará un pre-procesado, eliminando palabras que no aporten valor, y varios métodos más que se explicarán en el siguiente capítulo. El modelo generado se subirá a un servidor junto con toda la información necesaria para que posteriormente, con la llegada de un tweet, éste sea clasificado en el menor tiempo posible.

En cambio, el segundo módulo ofrecerá al usuario la opción de configurar su propio modelo suministrando un corpus previamente elaborado por él y unos valores correctos para el SVM. Esto implica tener conocimientos sobre el algoritmo para garantizar su correcto funcionamiento. Una vez configurado y entrenado el modelo ofrecerá la ventaja de obtener mejores resultados con respecto al primer modelo. Esto se deberá a que al utilizar un corpus relacionado con la consulta que desee el usuario se incrementa notablemente la precisión del algoritmo.

Pero antes de comenzar explicando la implementación de estos módulos se presentan algunas peculiaridades del desarrollo en Sinfonier.

9.1 – Programación en Sinfonier

Es de vital importancia conocer cómo se desarrolla un módulo en Sinfonier para poder proseguir con la realización del proyecto.

Para desarrollar código en Sinfonier es necesario seguir un patrón. Éste está compuesto principalmente de tres métodos que realizan básicamente la misma función pero se distinguen según el tipo de módulo a realizar; es decir, dependiendo si desarrollamos un módulo de tipo spout, bolt o un drain (tipos explicados en el apartado 4.2):

1. En spout, se denominan useropen(), usernextTuple(), userClose().
2. En bolt, se denominan userprepare(), userexecute(), usercleanup().
3. En drain, se denominan userprepare(), userexecute(), usercleanup().

El método useropen()/userprepare() se ejecuta solo una vez, en el lanzamiento de la topología. Es el encargado tanto de abrir conexiones como recabar toda información necesaria para la construcción del modulo.

El método usernextTuple()/userexecute(), se encarga de procesar cada tupla que le llega al módulo una vez esté lanzado. Básicamente lee, modifica y realiza todas las operaciones necesarias y lo vuelve a mandar mediante el método emit().

Por último el método userClose()/usercleanup, se ejecuta una vez el proceso finalice. Es el encargado de cerrar todas las conexiones o de liberar memoria que use nuestro sistema.

9.2 – Implementación

El primer módulo, el más sencillo para el usuario, contendrá un modelo que será previamente entrenado, por lo que este módulo no requerirá de conocimiento ni configuración alguna por parte del usuario respecto al SVM.

Por otra parte, se le facilitará al usuario otro módulo en el que será él el encargado de entrenar y configurar la máquina de soporte de forma personalizada. Así pues, el usuario deberá proporcionarle un corpus en el que cada tweet deberá tener el siguiente formato, para su posterior procesamiento:

[polaridad] [texto del tweet]

Para la implementación de la máquina de soporte vectorial vamos a utilizar una biblioteca llamada LIBSVM (Chih-Chung & Chih-Jen).

Esta biblioteca nos ofrece varios tipos de SVM:

1. SVM de clasificación tipo 1 (también conocido como C-SVC)
2. SVM de clasificación tipo 2 (también conocido como nu-SVC)
3. SVM de regresión tipo 1 (también conocido como epsilon-SVR)
4. SVM de regresión tipo 2 (también conocido como nu-SVR)

Los dos primeros son utilizados para clasificación, en cambio los dos siguientes son utilizados para problemas de regresión y puesto que en nuestro problema se busca la clasificación de texto en dos clases (positivo y negativo) centraremos nuestro estudio en los dos primeros.

Entre el primer tipo y el segundo apenas hay diferencias, pero utilizan diferentes parámetros. El primer tipo utiliza un parámetro penalización denominado 'C' de dominio entre cero e infinito; sin embargo, en el segundo tipo este parámetro es intercambiado por 'nu' que toma valores en el intervalo [0,1].

Basándonos en la guía publicada por los autores de esta biblioteca (Chih-Wei, Chih-Chung, & Chih-Jen) seguiremos el siguiente procedimiento propuesto:

1. Transformar los datos al formato especifico para el uso del SVM.
2. Realizar un escalado de los datos
3. Inicialmente considerar como núcleo el RBF
4. Utilizar la validación cruzada para encontrar el mejor parámetro C y γ
5. Utilizar el mejor parámetro C y gamma para entrenar el modelo
6. Testear dicho modelo, realizando las pruebas pertinentes

9.3- Pre procesado de datos

En este apartado explicaremos brevemente cómo se han transformado los tweets en función del formato especificado por el algoritmo.

El primer paso para el pre procesado de los datos es eliminar todas las palabras vacías, es decir, palabras que no aportan valor a una frase, de cada tweet. Denominado stopper[2] en el ámbito del PLN.

Seguidamente, se ha obtenido la raíz de cada palabra aplicando un 'stemmer', utilizando una biblioteca llamada libstemmer[3]

SVM requiere que cada dato se represente como un vector de números reales, por lo que hemos tenido que convertir nuestras palabras en datos numéricos. Para realizar esta conversión en primer lugar los hemos leído todos, almacenando cada palabra secuencialmente. Una vez guardadas todas las palabra calculamos su TF-IDF.

El cálculo del TF-IDF no es más que el producto de dos medidas: frecuencia de término y frecuencia inversa de documento.

La frecuencia del término es el número de veces que la palabra 'p' ocurre en el tweet 't'. La frecuencia inversa de documento es un número que representa el grado de aparición de la palabra en el corpus. La obtenemos dividiendo el número total de tweets por el número de tweets que contiene la palabra, y se toma el logaritmo de ese cociente.

$$idf(p,T) = log\frac{|T|}{|\{t \in T: p \in t\}|}$$

dónde:

$|T|$, número total de tweets

$|\{t \in T: p \in t\}|$, número de tweets donde aparece la palabra 'p'

Una vez obtenido el tf-idf de cada palabra, se reconstruye el tweet, formado por un número entero indicando la palabra a la que hace referencia seguido del tf-idf calculado.

Ejemplo:

pensando en lo mas lindo del mundo

1 9908:4.51 16423:5.271 21891:5.023

9.4- Entrenamiento SVM

Como ya hemos visto el proyecto contará con dos módulos (uno previamente entrenado y otro en el que el entrenamiento del modelo correrá a cargo del usuario). Así pues, en este apartado se explicará tanto cómo se realiza el entrenamiento previo como cuáles son los parámetros que tendrá que conocer el usuario para entrenar su propio modelo .

Si el usuario desea entrenar su propio modelo necesitará conocer previamente algunos parámetros necesarios para realizar una correcta elección de los valores de los mismos:

1. **svm_type :** tipo de algoritmo SVM a utilizar. Para nuestro caso sólo será posible elegir un algoritmo de clasificación binaria C-SVC o bien nu-SVC.
2. **kernel_type** : tipo de núcleo. Existen lineal, polinomial, función de base radial y sigmoid.

1. lineal: $u^{'} * v$
2. polinomial: $(\gamma * u^{'} * v)^{grado}$
3. función de base radial: $\exp\left(-\gamma * |u - v|^2\right)$
4. sigmoid: $\tanh\left(\gamma * u' * v\right)$

Para el módulo previamente entrenado necesitaremos determinar los valores de estos parámetros acordes a nuestro corpus.

Éstos se determinarán experimentalmente partiendo (como se en especifica la documentación de la biblioteca 'LIBSVM') de la realización de una validación cruzada a partir de los valores por defecto.

En primer lugar, trataremos de determinar un valor ajustado para los parámetros C y nu. Estos parámetros buscan permitir cierta flexibilidad controlando la compensación entre errores de entrenamiento y los márgenes rígidos. Esta compensación consiste en crear un margen blando donde se incluyan datos mal clasificados (soft margin) penalizándolos consiguiendo así evitar un sobreajuste.

Para realizar el ajuste tanto de los valores como del núcleo utilizaremos el corpus COST[4] proporcionado por el Grupo de investigación de Sistemas Inteligentes de Acceso a la Información de la Universidad de Jaén. De este corpus se seleccionará un 80% de las instancias para el entrenamiento del modelo y se dejará el 20% restante para determinar la precisión de la configuración del algoritmo.

Para las distintas configuraciones de C en el tipo C_SVC se obtuvieron los siguientes resultados:

C_SVC			
Núcleo	**C**	**Precisión(%)**	**Tiempo(s)**
RBF	50	72,948	333
RBF	70	72,802	539
RBF	**60**	**73,027**	**367**
RBF	55	72,99	367

Tabla 9.3: Resultados C_SVC

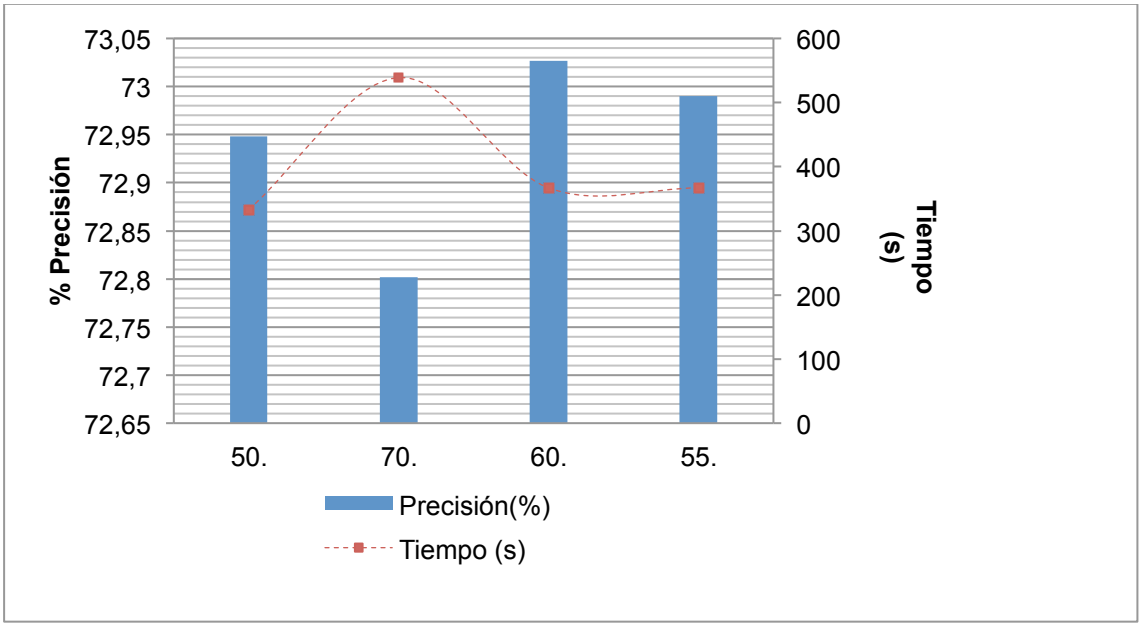


Figura 9.11: Gráfico resultados C_SVC

Como podemos observar en la tabla 9.3 y en la figura 9.11 la configuración que mas precisión nos proporciona es en la que 'C' toma el valor de 60.

Para las distintas configuraciones de nu en el tipo nu_SVC se obtuvieron los siguientes resultados

nu_SVC			
Núcleo	**nu**	**Precisión(%)**	**Tiempo (s)**
RBF	0.5	72,079	441
RBF	0.9	69,935	167
RBF	**0.6**	**72,96**	**275**
RBF	0.55	72,395	210

Tabla 9.4: Resultados nu_SVC

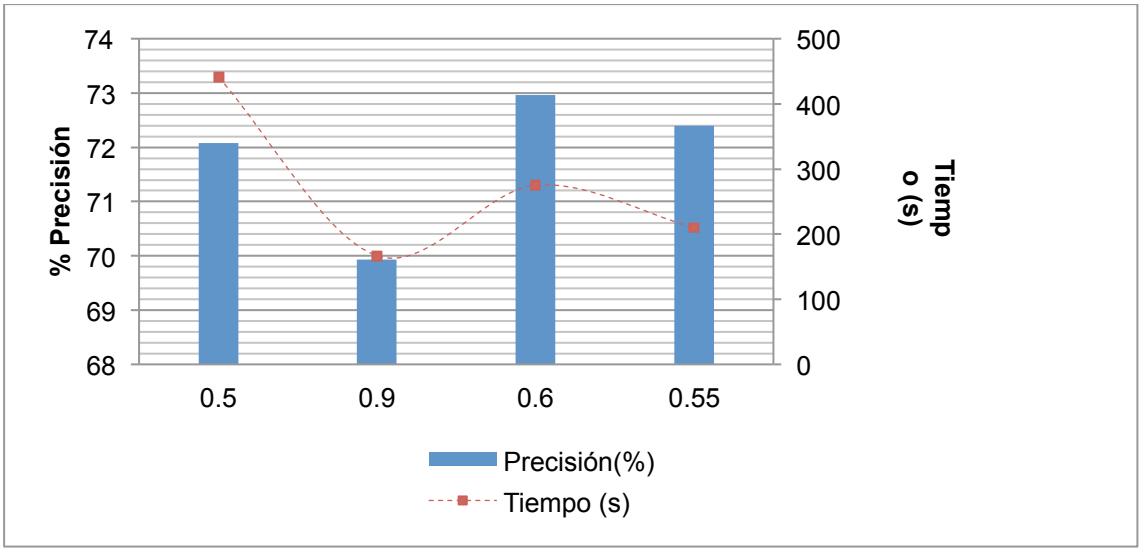


Figura 9.12: Gráfico resultados nu_SVC

Como podemos observar en la tabla 9.4 y en la figura 9.12 la configuración que mas precisión nos proporciona es en la que 'nu' toma el valor de 0.6.

Una vez obtenidos estos valores determinaremos qué núcleo consigue mejores resultados.

C_SVC			
Núcleo	**C**	**Precisión(%)**	**Tiempo**
Lineal	60	67,498	36380
Polinomial	60	50,332	139
RBF	60	73,027	367
Sigmoid	**60**	**73,186**	**193**

Tabla 9.5: Resultados núcleos C_SVC

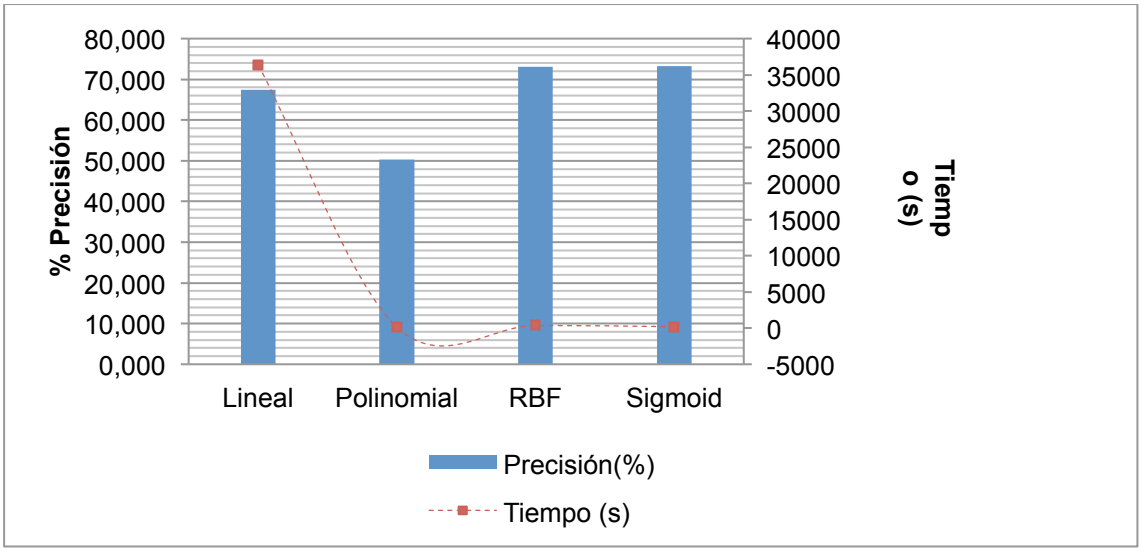


Figura 9.13: Gráfico resultados núcleos C_SVC

La precisión obtenida para el núcleo 'lineal' no es correcta, ya que tras 10 horas de ejecución el sistema llego al número máximo de iteraciones, dejando incompleto el modelo, aún así se muestra el resultado obtenido para este núcleo.

Observando la tabla 9.5 y figura 9.13 la mayor precisión para el tipo C_SVC es del 73,18% proporcionada por el núcleo 'Sigmoid'.

Para los distintos núcleos con el tipo nu_SVC se obtuvieron los siguientes resultados:

nu_SVC			
Núcleo	**un**	**Precisión(%)**	**Tiempo (s)**
Lineal	0.6	72,847	434
Polinomial	0.6	54,057	74

RBF	**0.6**	**72,96**	**275**
Sigmoid	0.6	72,915	214

Tabla 9.6: Resultados núcleos nu_SVC

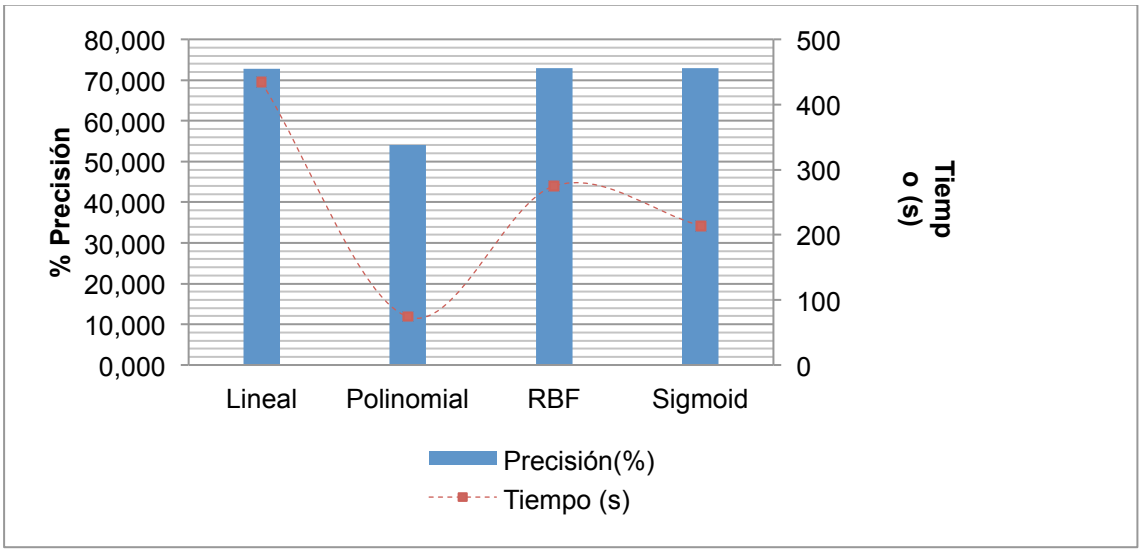


Figura 9.14: Gráfico resultados núcleos nu_SVC

Como podemos observar en la tabla 9.6 y en la figura 9.14 el mejor resultado nos lo proporciona el núcleo RBF con una precisión de 72,96%.

Podemos concluir que la mejor configuración es utilizar el tipo 'C_SVC' con el núcleo 'Sigmoid' ya que este nos garantiza la mayor precisión con un tiempo de ejecución dentro de un rango admisible.

9.5- Inclusión del modelo

Para el modelo previamente entrenado es necesario almacenar en la nube tanto el modelo, como las palabras, como los idfs calculados en el pre procesamiento de éstas.

Se escogió gist.github.com como plataforma para leer y escribir estos datos, para conseguir que sean accesibles tanto el modelo[5], las palabras[6] y los idfs[7] públicamente.

Las palabras y los idfs correspondientes están subidos como texto plano y el modelo subido está con formato JSON.

Una vez subidos todos los archivos necesarios, nuestro modulo en Sinfonier es el encargado de su lectura y de ejecutar las conversiones necesarias para volver a nuestro modelo tras utilizar el formato JSON como pasarela.

Estas conversiones las realizamos a través de la biblioteca Gson de Google, quedando así nuestro modelo previamente entrenado disponible para clasificar nuevos tweets.

Por otro lado, para el modelo que no esta previamente entrenado, en el método de apertura (userprepare) se pre-procesa el corpus, como se explicó anteriormente y se construye y entrena el modelo acorde con los datos introducidos por el usuario.

Así al pasar al siguiente método tanto en un módulo como en el otro, ya disponemos del modelo entrenado, dispuesto a clasificar los nuevos tweets que reciba el sistema.

9.6- Clasificación de nuevos tweets

Con nuestro modelo entrenado nos queda recibir el texto de cada uno de los tweets a clasificar, procesarlos y mandar el resultado a nuestros módulos drains que se encargarán de su visualización.

El procedimiento para el pre procesado de los tweets es el mismo que se sigue para el pre procesado de todo el corpus (eliminación de palabras vacías, extracción de raíces y cálculo de frecuencia de cada término y pesos).

9.7- Visualización

Al principio del proyecto se tomó la decisión de reutilizar los módulos de Sinfonier que ya existían para visualizar nuestros resultados; sin embargo, la visualizaciones ofrecidas por estos módulos eran poco personalizables y no se ajustaban a nuestras necesidades. Así pues, elegimos otra opción de visualización: la plataforma geckoBoard

GeckoBoard es un servicio web que ofrece la capacidad de monitorizar en tiempo real nuestro sistema, pudiendo así visualizar según el intervalo de tiempo dado por el usuario todos los tweets clasificados. Esta plataforma dispone de diversos widgets para visualizar datos como mejor se crea conveniente. Se optó por elegir los siguientes:

1. Una lista que mostrará tweets, colocando a cada lado del tweet una etiqueta con valor 'pos' o 'neg' dependiendo de su clasificación.
2. Un diagrama de sectores, que muestra cuántos tweets se han catalogado como positivos y cuántos como negativos.
3. Un diagrama de líneas, que partiendo del intervalo dado por el usuario (por ejemplo cada hora), nos muestra cuántos tweets negativos y positivos han sido clasificados en ese intervalo.
4. Un mapa, que nos localizaría de dónde proviene cada tweet escogido.

GeckoBoard nos provee de una api para interactuar con nuestro dashboard. Para mostrar nuestros resultados debemos de hacer una petición POST pasándole un JSON en función de cada widget utilizado.

El resultado final quedaría así:

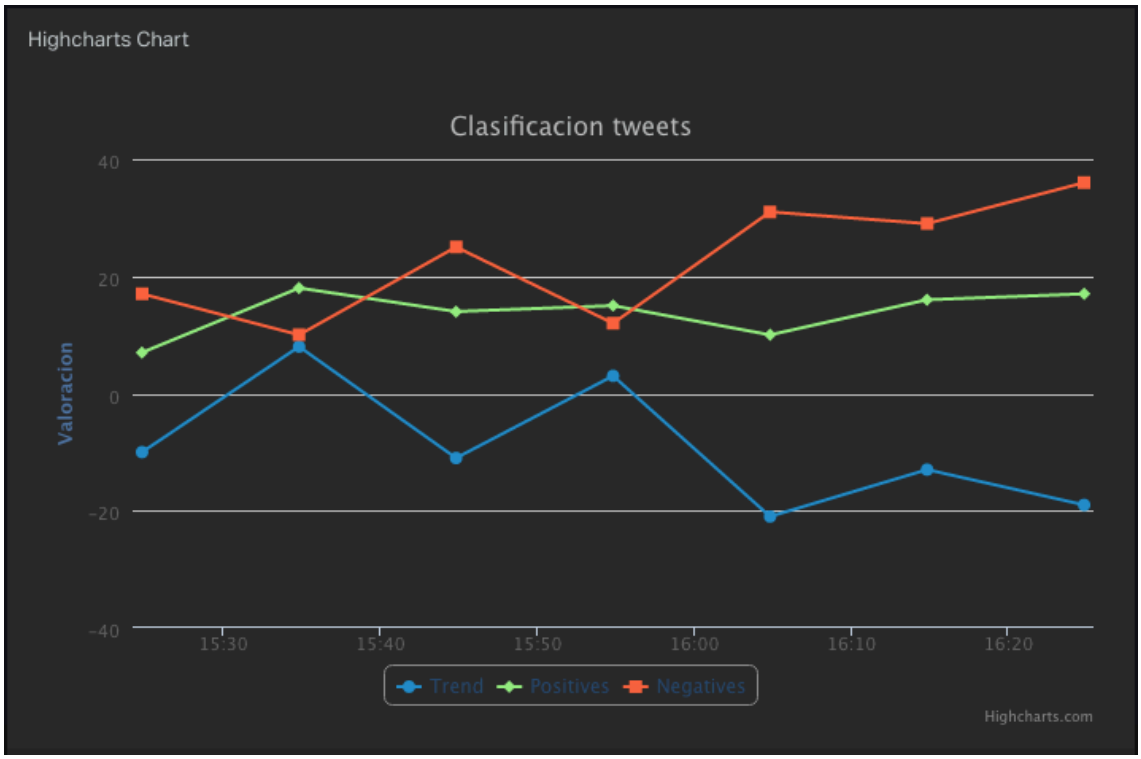


Figura 9.15: Interfaz gráfica

9.8- Módulos desarrollados

A continuación se resumen todos los módulos desarrollados en Sinfonier.

Este módulo implementa un algoritmo SVM previamente entrenado con un corpus de 34630 textos extraídos de la red social Twitter. Éste es capaz de clasificar nuevos comentarios, indicados por el campo "textField". El resultado de la clasificación es almacenada en un campo de un JSON llamado "polarity".

EasySVM	
Tipo	Bolt
Estado	Privado
Lenguaje	Java
URL código	https://gist.github.com/jcmc0011/84207cbfc3dd7ca8a8ab
Descripción	Este módulo implementa el SVM previamente entrenado.
Parámetros de entrada	textField – campo del Json en el que se encuentra el texto que queremos clasificar
Parámetros de salida	polarity – campo agregado con la clasificación realizada

Este módulo implementa un máquina de soporte vectorial destinada a la polarización de tweets, que debe ser previamente configurado con los parámetros especificados en la tabla. Éste módulo permite al usuario entrena el clasificador con su propia colección de tweets etiquetados a nivel de opinión.

AdvancedSVM	
Tipo	Bolt
Estado	Privado
Lenguaje	Java
URL código	https://gist.github.com/jcmc0011/e0b3e96e7a0f914f5cfd
Descripción	Este módulo implementa un SVM para la clasificación de textos
Parámetros de entrada	textField – campo del Json en el que se encuentra el texto que queremos clasificar inputField – ruta local dónde se encuentra el corpus para entrenar el modelo stopWordsField – ruta local dónde se encuentra la bolsa de palabras vacías a eliminar svm_type – tipo de algoritmo SVM para aplicar (C_SVC, un_SVC) svm_kernel – nucleo del algoritmo SVM a utilizar param_nu – parámetro para el tipo nu_SVC que mide el margen de error en la clasificación param_c – parámetro para el tipo C_SVC que mide el margen de error en la clasificación
Parámetros de salida	polarity – campo agregado con la clasificación realizada

Los siguientes módulos hacen referencia a las distintas formas de visualización de los resultados a través de la plataforma GeckoBoard.

GeckoBoardList	
Tipo	Drain
Estado	Privado
Lenguaje	Java
URL código	https://gist.github.com/jcmc0011/a4bf1072bfebf95db8c1
Descripción	Manda la salida a una lista de GeckoBoard
Parámetros de entrada	api_key – clave privada que la api de GeckoBoard nos da widget_key – clave que hace referencia al widget en cuestión. polarity_field – campo de texto dónde se encuentra la solución dentro del JSON
Parámetros de salida	Ninguno

GeckoBoartLineChart	
Tipo	Drain
Estado	Privado
Lenguaje	Java
URL código	https://gist.github.com/jcmc0011/f572c8bbc3f2ad1650d5
Descripción	Manda la salida a un gráfico de lineas de GeckoBoard
Parámetros de entrada	api_key – clave privada que la api de GeckoBoard nos da widget_key – clave que hace referencia al widget en cuestión. polarity_field – campo de texto dónde se encuentra la solución dentro del JSON interval – Intervalo de tiempo dado en milisegundos para la actualización del gráfico
Parámetros de salida	Ninguno

GeckoBoartPieChart	
Tipo	Drain
Estado	Privado
Lenguaje	Java
URL código	https://gist.github.com/jcmc0011/3141c1b4bf9ec695c5c9
Descripción	Manda la salida a un grafico de sectores de GeckoBoard
Parámetros de entrada	api_key – clave privada que la api de GeckoBoard nos da widget_key – clave que hace referencia al widget en cuestión. polarity_field – campo de texto dónde se encuentra la solución dentro del JSON interval – Intervalo de tiempo dado en milisegundos para la actualización del gráfico
Parámetros de salida	Ninguno

GeckoBoartMap	
Tipo	Drain
Estado	Privado
Lenguaje	Java
URL código	https://gist.github.com/jcmc0011/90c37b76adc432698e3d
Descripción	Manda la salida a un mapas de GeckoBoard
Parámetros de entrada	api_key – clave privada que la api de GeckoBoard nos da widget_key – clave que hace referencia al widget en cuestión.
Parámetros de salida	Ninguno

Capítulo 10

Validación del sistema

Una vez integrado el sistema realizamos las siguientes pruebas:

Prueba 1 : Control de la polarización de los tweets negativos.

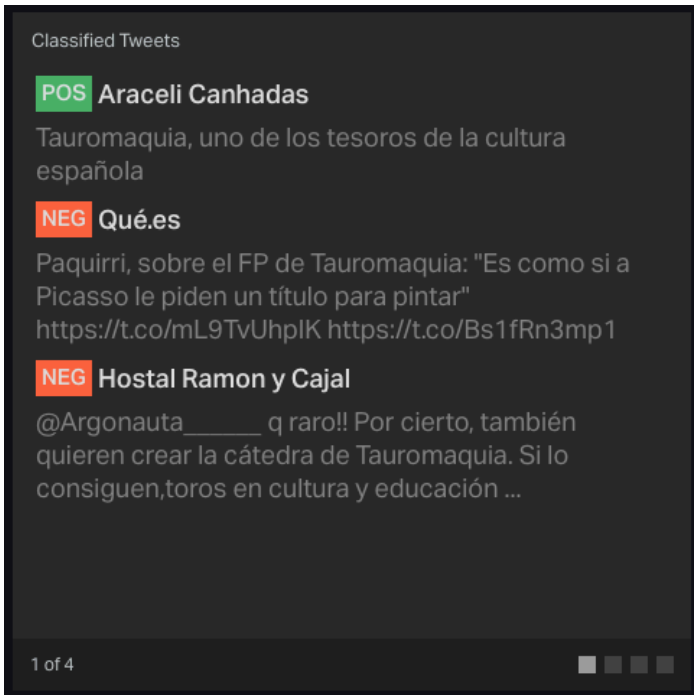
1. Se introdujo como palabra clave ‘tauromaquia’
2. Se escribió un tweet desde una cuenta personal con el siguiente contenido:
3. “No estoy a favor de la tauromaquia”
4. Se comprobó que dicho tweet aparecía en el timeline.
5. Se comprobó la correcta clasificación del texto.

En la siguiente imagen se muestra el resultado:

Prueba 2 : Control de la polarización de los tweets positivos.

1. Se introdujo como palabra clave ‘tauromaquia’
2. Se escribió un tweet desde una cuenta personal con el siguiente contenido:
3. “Tauromaquia, uno de los tesoros de la cultura española”
4. Se comprobó que dicho tweet aparecía en el timeline.
5. Se comprobó la correcta clasificación del texto.

En la siguiente imagen se muestra el resultado:



Tras evaluar el sistema durante una semana vemos que cumple con los requisitos marcados, sin embargo el mapa ha sido eliminado ya que desde que empezamos a realizar las pruebas no conseguimos localizar ningún tweet, debido a la baja utilización de dicha opción en Twitter.

Capítulo 11

Conclusiones

El Análisis de Sentimientos es una tarea relativamente reciente y sobretodo demandada, las empresas cada vez más quieren conocer la opinión sobre sus productos debido a los grandes beneficios que reporta, entre ellos, la gran capacidad de respuesta que proporciona el conocer cómo recibe el público tus productos y poder así anticiparte a otras competidores.

Ante la ingente cantidad de información subjetiva en internet, y el gran interés que suscita ésta por el mundo empresarial surge la idea de este trabajo, ocupando un posible hueco en el mercado, puesto que en España hoy en día pocas empresas se dedican a explotar y obtener toda esta información al alcance de todos.

Este trabajo se podría ampliar de varias formas, una de ellas podría ser el enriquecimiento de la información extraída a través de otros métodos (por ejemplo analizar metadatos en las fotografías) o podría ser completado con un estudio mas profundo a cerca de diferentes algoritmos al usado en este trabajo. Concluyendo el prototipo conseguido da muy buenos resultados, aunque la curva de aprendizaje para el usuario podría ser demasiado alta al inicio, característica que debería de mejorarse en un futuro.

Gracias a este trabajo he descubierto una nueva rama de la Informática para mi desconocida, con mucho trabajo por realizar aún pero a su vez muy productivo.

Anexo I

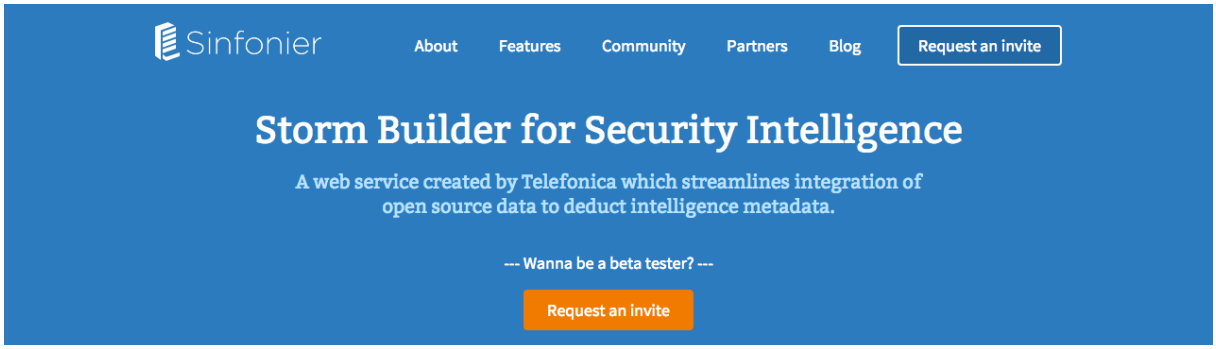
Manual de usuario

A lo largo de este manual, se explicará todo lo necesario para poner en funcionamiento un sistema capaz de extraer y clasificar textos extraídos de la red social Twitter.

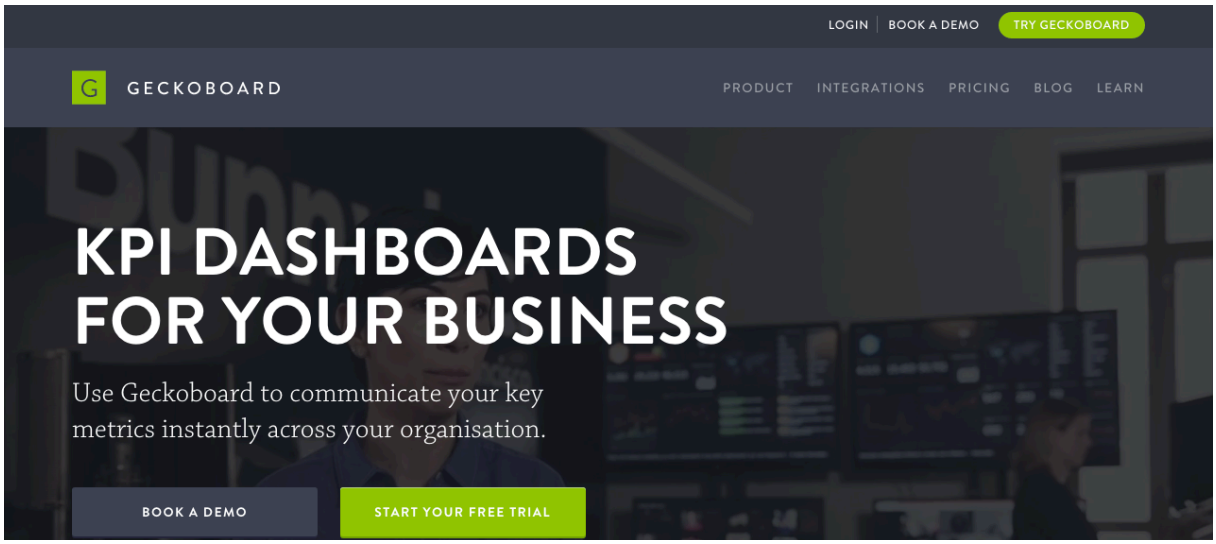
1.- Registro

Antes de nada el usuario deberá completarse un registro en las dos plataformas en la que se han basado el trabajo:

http://sinfonier-project.net

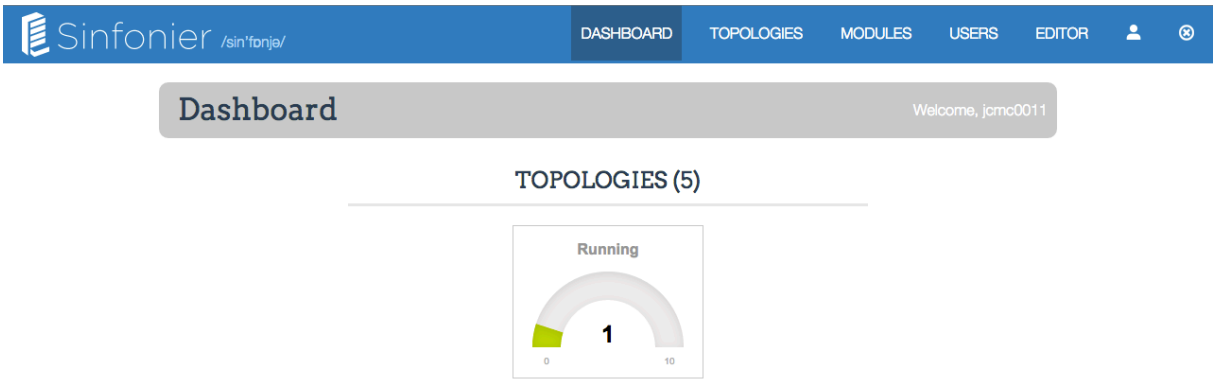


https://www.geckoboard.com/



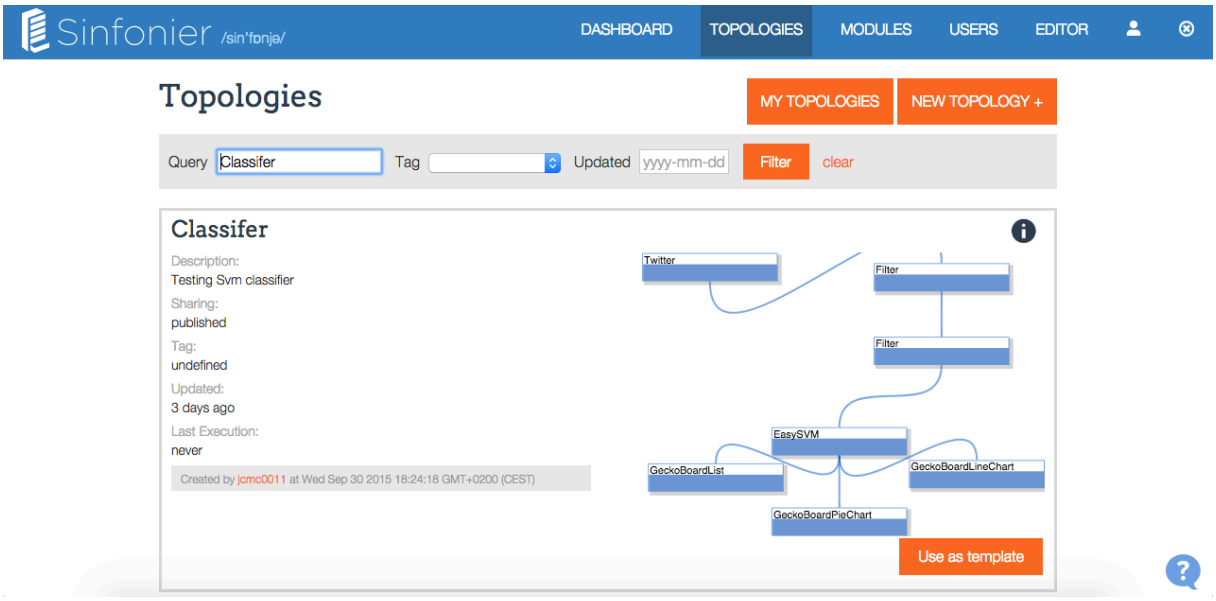
2.- Clasificador

Una vez completado el registro entramos en Sinfonier y nos aparecerá la siguiente pantalla:

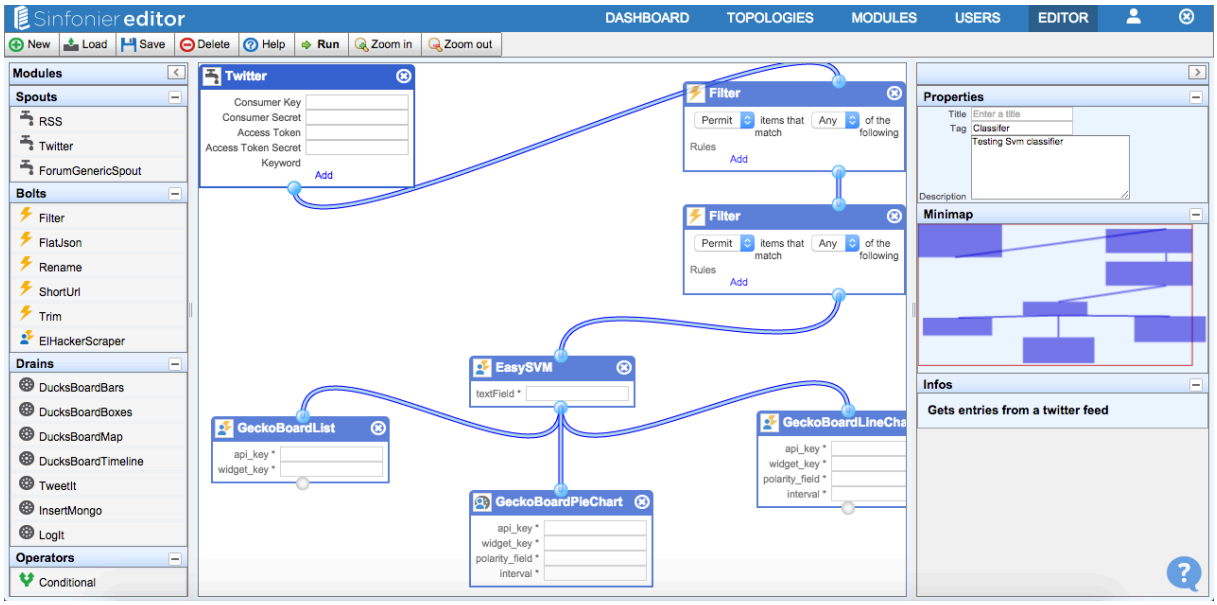


En la barra superior, clicamos sobre TOPOLOGIES, en este apartado nos aparecerán todas las topologías publicadas por los usuarios de Sinfonier, así como un cuadro de texto a modo de buscador.

Para el utilizar el clasificador previamente entrenado buscamos “Classifer”, nos aparecerá la siguiente pantalla con la topología buscada.

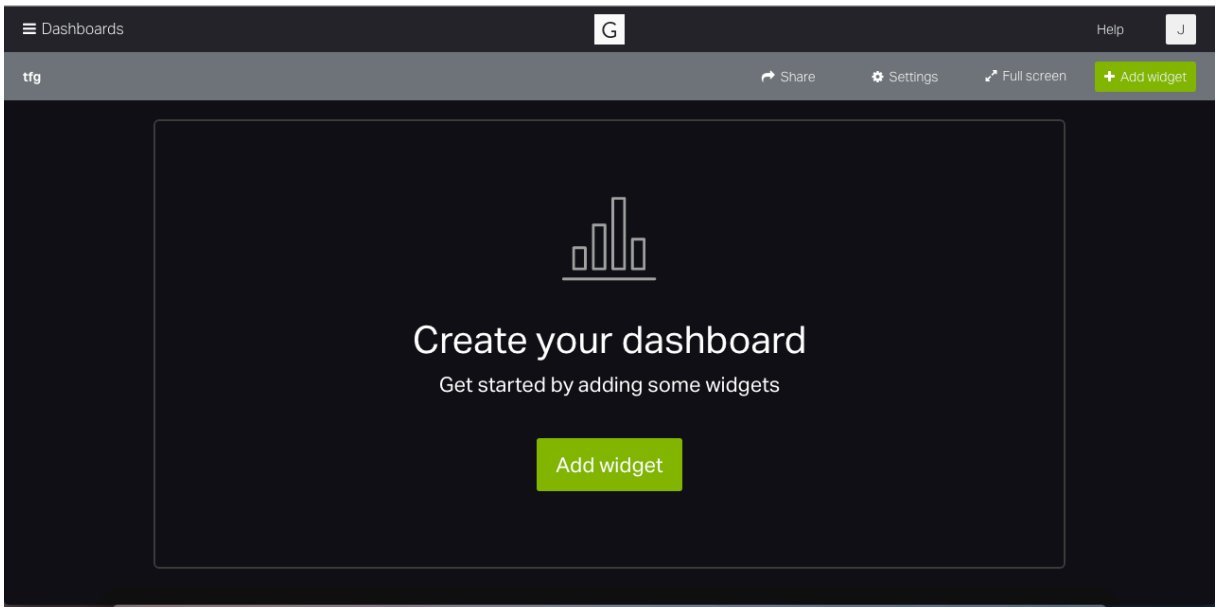


Pulsamos el botón “Use as template” para abrir dicha topología y proceder a su utilización.



Una vez estemos en esta pantalla, lo primero que debemos realizar es ponerle un nombre a esta topología y guardarla. Una vez guardada insertaremos los campos necesarios para su funcionamiento. Cada módulo esta detallado en el apartado 9.8 con los parámetros de entrada necesarios para cada uno.

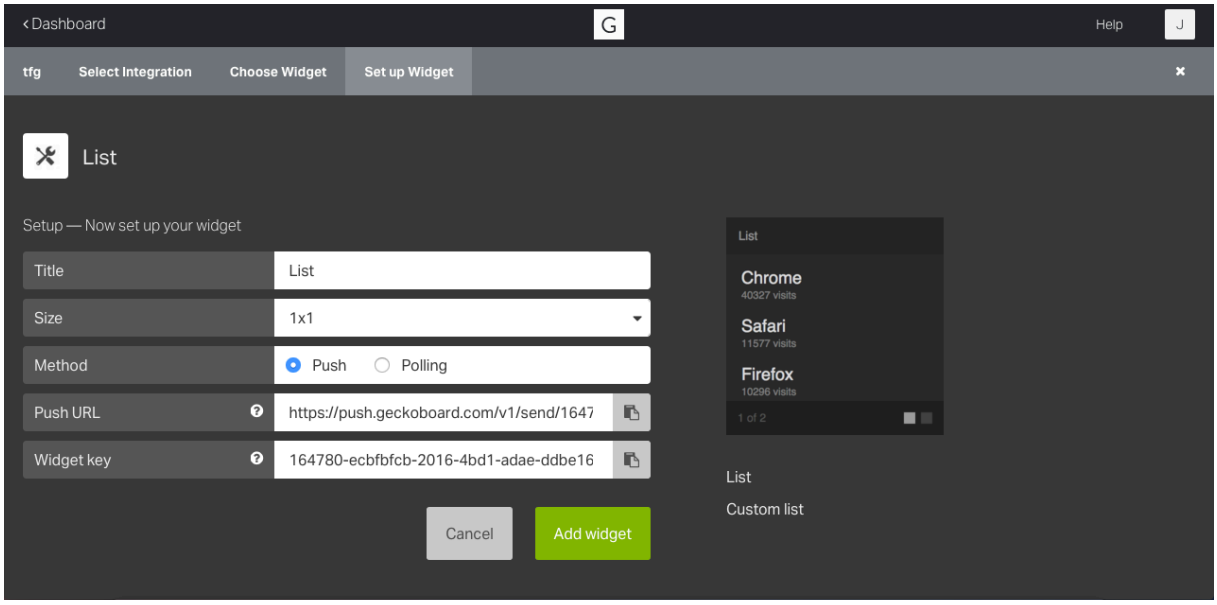
Para visualizar los resultados en GeckoBoard, una vez realizado el registro debemos agregar tres componentes (widgets) para ello lo primero es pulsar el botón de “Add widget”



Dentro de los todos los widgets que nos ofrece entramos en Custom widgets y seleccionamos “List”, “Hightcharts Chart” y “Pie Chart”, que son los tres que

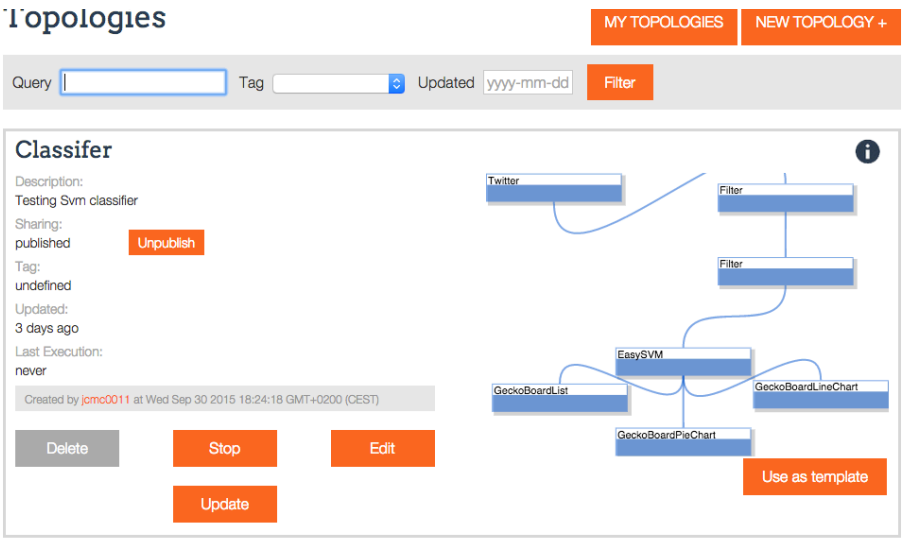
utilizaremos. Al seleccionar cada uno pasaremos a una pantalla de configuración del widget

Para visualizar los resultados en GeckoBoard, una vez realizado el registro debemos agregar tres componentes (widgets) para ello debemos es pulsar el botón de “Add widget”.



Lo único a tener en cuenta es seleccionar en todos los widgets la opción de push y copiar la “Widget Key” para insertarla posteriormente en Sinfonier.

Una vez insertadas todas la información necesaria en Sinfonier, lanzaremos la topología clicando sobre “Launch” hasta obtener la siguiente imagen



Con esto concluiríamos la configuración del sistema y solo nos quedaría visualizar los resultado en GeckoBoard a través de los widgets creados.

Anexo II

Contenido del CD

Todo lo necesario para el correcto desempeño del sistema planteado se encuentra en el CD- ROM adjunto a esta documentación.

Su contenido es el siguiente:

Carpeta Memoria: contiene la memoria en formato PDF.

Carpeta Código fuente: código fuente del sistema

Bibliografía

Casal Terreros, J. (n.d.). From Microsoft Developer Works: https://msdn.microsoft.com/es-es/library/bb972268.aspx#EFAA

Chih-Chung, C., & Chih-Jen, L. (n.d.). *LIBSVM*. From A library for support vector machines. ACM Transactions on Intelligent Systems and Technology: http://www.csie.ntu.edu.tw/~cjlin/libsvm

Chih-Wei, H., Chih-Chung, C., & Chih-Jen, L. (n.d.). From A Practical Guide to Support Vector Classification: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

Das, S., & y Chen, M. (2001). Yahoo! for Amazon: Sentiment extraction from small talk on the web. *Management Science.*

Dave, K., Lawrence, S., & Pennock, D. (2003). Mining the peanut gallery. *opinion extraction and semantic classification of product reviews* .

Horrigan, J. B. (2008). Online Shopping. *Pew Internet & American Life Project Report.* .

Martínez Cámara, E., Martín Valdivia, M., Ureña Lopez, L., & Mitkov, R. (2015). Polarity classification for Spanish tweets using the COST corpus. *Journal of Information Science* .

Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval* .

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment Classification using Machine Learning Techniques.

Tong, R. (2001). An Operational System for Detecting and Tracking Opinions in On-line Discussion. *Asia Pacific Finance Association Anual Conference.*

Turney, P. (2002). Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. *Association for Computational Linguistics* .

Enlaces

1. http://sinfonier-project.net/
2. https://gist.github.com/678ce6f62d17d6e626b4.git
3. http://snowball.tartarus.org/
4. http://sinai.ujaen.es/cost-2/
5. https://gist.github.com/jcmc0011/909f47afd510f09a7f6e
6. https://gist.github.com/jcmc0011/d7b66f2fb0de12b31bc5
7. https://gist.github.com/jcmc0011/4f61a0a8c0b362f914a5