UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería Informática

Grado en Ingeniería Informática

Trabajo Fin de Grado

Desarrollo de un sistema de análisis de sentimiento sobre Twitter

---

Autor: Javier Selva Castelló

Directores: Lluís Felip Hurtado Oliver
Ferran Pla Santamaría

Septiembre de 2015

Resumen

Twitter se ha convertido en una de las plataformas *on-line* más utilizadas para expresar opiniones e ideas. Es debido a esta razón que resulta una fuente ideal de información de la que extraer estadísticas sociales. Este proyecto pretende analizar esta información centrándose en el estudio de la polaridad. El análisis de sentimientos sobre Twitter busca establecer la subjetividad de las opiniones expresadas sobre esta plataforma.

El objetivo del proyecto es el desarrollo de una aplicación web basada en *Django*. Esta debe agrupar diversas herramientas de clasificación que generen estadísticas de polaridad a partir de un conjunto de tuits. Además, la aplicación almacenará un histórico con el que mostrar la evolución de los resultados.

Por otra parte, se ha desarrollado un sistema de resumen automático basado en la extracción de los tuits más representativos para una búsqueda concreta. Para ello se ha implementado un sistema basado en el Análisis Semántico Latente el cual también tiene en cuenta la popularidad de un tuit a la hora de escogerlo para el resumen. Con el objetivo de probar este sistema se ha realizado un proceso de experimentación que abarca desde la elaboración de un corpus de tuits puntuados manualmente por relevancia, hasta el estudio de las diferentes características de Twitter que hacen que un tuit se considere popular.

Palabras clave Twitter, Redes Sociales, Análisis de Sentimientos, Análisis de tendencias, Generación de resúmenes

Abstract

Twitter has become one of the most popular online platforms used to express opinions and ideas. For this reason, Twitter is a great source of information which can be used to produce social statistics. This project is focused on one specific aspect of the analysis of this information: polarity. The goal of Sentiment analysis on Twitter is finding the subjectivity within the opinions expressed in this platform.

The aim of the project is developing a Django based web application. It should gather different classification tools in order to obtain polarity statistics out of a tweet set. Also, the application will store the outcome in order to show evolution of the results.

Moreover, an automatic summarization tool has been also developed. It's based on the extraction of the most relevant tweets for a specific query. In order to do so it has been implemented a system based on Latent Semantic Analysis. This system also considers the tweets' popularity when producing the summary. The system was tested by manually scoring a tweet corpus by relevance and studying the different Twitter features that makes a tweet popular.

Keywords Twitter, Social Network, Sentiment Analysis, Summarization, Tendency Analysis

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

Twitter es una herramienta de *microblogging* social en la que los usuarios pueden publicar contenido textual de hasta 140 caracteres. Desde su nacimiento en 2006, Twitter se ha convertido en una de las plataformas más empleadas para compartir contenido de actualidad, como noticias o eventos en tiempo real, que además permite opinar de manera breve y concisa sobre prácticamente cualquier tema. En 2015 Twitter cuenta con más de 300 millones de usuarios activos [25] los cuales producen grandes cantidades información de manera incesante. Es por esto que se hacen necesarias herramientas de procesamiento que sinteticen todo ese contenido.

El Análisis de Sentimientos es una tarea incluida en el ámbito del Procesamiento de Lenguaje Natural o NLP (*Natural Language Processing*), del Análisis de Textos y de la Lingüística Computacional. Su objetivo es encontrar contenido subjetivo en los textos de entrada. El análisis de sentimientos busca extraer opiniones y la polaridad de estas de uno o más documentos mediante la extracción de una serie de características que determinen cuán positivo o negativo es el texto.

Algunas tareas para las que puede emplearse este campo son, por ejemplo, encontrar la valoración (puntuación) de un restaurante o película en base a la polaridad extraída de diferentes críticas y reseñas que existan al respecto. En el caso de este proyecto se busca encontrar la opinión de un conjunto de usuarios de Twitter acerca de un tema determinado.

El análisis de polaridad sobre Twitter pretende establecer una opinión generalizada de una entidad concreta. La versatilidad de una herramienta como esta radica en la gran variedad de usuarios que cada día publican contenido en Twitter relacionado con una gran variedad de asuntos. Esto convierte a la red social en el lugar perfecto del que extraer la información que permita establecer la opinión mayoritaria acerca de prácticamente cualquier tema. Por otra parte, el análisis de polaridad permite estudiar como afectan diversos factores sociales a la opinión que la sociedad tiene acerca de un asunto. Por ejemplo cómo modifica una determinada campaña política a la valoración de las personas respecto a un partido político concreto. Al estudio de esta evolución se le denomina Análisis de Tendencias.

Por último, los sistemas de resumen automático pueden resultar una herramienta muy útil a la hora de complementar al análisis de sentimientos. Los resumidores.ªutomáticos son herramientas que buscan extraer el contenido más relevante de un conjunto de documentos. El resumen generado automáticamente debe representar la información contenida en el texto que

se esté resumiendo sin incluir redundancias o partes del texto poco importantes. Estos sistemas se pueden emplear para extraer el contenido más representativo de los documentos de los que se esté realizando el análisis de polaridad. De este modo, cuando se trata de Twitter, no solo se obtiene finalmente la polaridad, si no también el conjunto de tuits que representan las opiniones.

Como se ha comentado, la gran cantidad de publicaciones que produce Twitter hacen necesarias herramientas que procesen, extraigan y en cierta medida interpreten esa información para generar un análisis de la polaridad y una representación de las opiniones contenidas en los textos originales.

Así pues, en este proyecto se plantea un sistema de análisis de sentimientos sobre Twitter en español debido a la ausencia de herramientas de este tipo específicas para dicho idioma. Con este objetivo, este proyecto parte del clasificador de polaridad desarrollado por Pla y Hurtado [16] de la *Universitat Politècnica de València* para el TASS [2], para elaborar una aplicación web que extraiga estadísticas de polaridad de un tema concreto a petición del usuario. Se ha escogido partir de este sistema de clasificación basado en Máquinas de Vectores Soporte o SVM (*Support Vector Machines*) por haber obtenido los mejores resultados de asignación de polaridad sobre Twitter en español en el TASS2014 [9] con una precisión del 62'88 %.

La aplicación debe permitir al usuario realizar búsquedas cuyos resultados se etiquetarán por polaridad mediante este clasificador. Estos resultados se almacenarán en una base de datos que permitirá hacer un análisis de tendencia que interprete de alguna manera la evolución de los resultados de polaridad a lo largo del tiempo. Así mismo, se mostrará una sere de resúmenes de los tuits clasificados que muestren las opiniones más relevantes expresadas en el contenido de los mismos.

1.1. Twitter

En esta sección se detallan algunas características y vocabulario relativo a la red social Twitter, de la que se va a extraer la información que esta aplicación se encargará de procesar.

Twitter se creó en 2006 como una red social en base a contenido con forma de SMS. Desde entonces ha crecido rápidamente hasta convertirse en la red social de *microblogging* que es hoy en día. Millones de usuarios acceden cada día para compartir experiencias y opiniones convirtiéndose así en una herramienta ideal para la realización de encuestas y sondeos. No obstante para ello se hace necesario aplicaciones como la que aquí se desarrolla que compacte y de formato a la información haciéndola útil y clara.

Twitter se basa en una red de usuarios los cuales pueden leer y escribir contenido en unidades de hasta 140 caracteres. Cada una de estas unidades se denomina **tuit**. El conjunto de tuits que ha publicado un usuario aparecen en su perfil por orden cronológico inverso. Sobre un tuit pueden realizarse diversas acciones, a saber:

- **Favorito**: Un tuit es marcado como favorito por otro usuario, esto no tiene ningún efecto más que como contador de usuarios a los que les ha gustado el tuit.

- **Retuit**: Por otro lado, un usuario puede "retuitear" un tuit, lo cual añadirá ese tuit a su propia línea de publicaciones indicando siempre que se trata de una publicación de otro usuario (del que se retuiteó).
- **Respuestas**: Un tuit puede generarse como respuesta a otro tuit, lo que puede llegar a generar conversaciones o debates.

Cualquiera de estos tres factores puede emplearse como indicativo de la relevancia de un tuit. Un tuit que ha sido retuiteado por muchos usuarios aparecerá en los perfiles de más personas (y por lo tanto en la página principal o *feed*), con lo que será leído más veces y tendrá un potencial mayor de llegar a un número elevado de usuarios. Del mismo modo, un tuit muy marcado como favorito suele indicar que el tuit ha gustado a mucha gente, generalmente por concordancia del usuario con lo expresado en el tuit.

Los usuarios se relacionan mediante un sistema de suscripciones:

- Un usuario puede suscribirse al contenido que otro usuario publica, convirtiéndose así en su **seguidor**.
- El conjunto de usuarios a los que sigue un usuario concreto se denominan **amigos**.
- La página principal de Twitter de un usuario (que no su perfil) contiene una mezcla de los tuits y retuits recientes (por orden cronológico inverso) publicados por todos sus amigos.
- Un usuario que tenga un número elevado de seguidores logrará que su contenido llegue directamente a muchos más usuarios.

Este funcionamiento puede verse resumido en la Figura 1.1 en la que se representa la relación entre usuarios de manera gráfica. Las flechas con línea continua representan "sigue a" mientras que las punteadas representan "publica".

Figura 1.1: Relaciones entre Usuarios

Por otro lado la Figura 1.2 muestra los diferentes apartados de una misma cuenta de usuario.

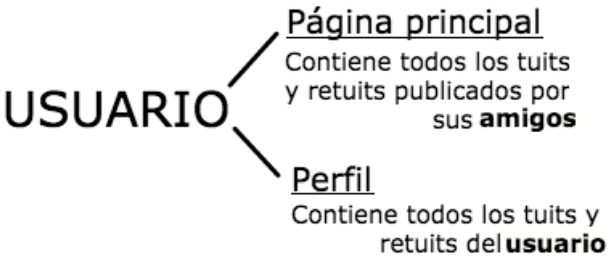


Figura 1.2: Contenido del usuario

Por lo tanto los tuits y retuits realizados por personas con muchos seguidores aparecerán en la página principal de un conjunto elevado de usuarios, logrando que esos tuits sean leídos por un número grande de usuarios, haciendo que estos tuits tengan un potencial mayor de convertirse en populares.

Por otro lado los tuits pueden contener, además de texto, una serie de "entidades" que se listan a continuación:

- **Menciones**: Una mención representa una citación a un usuario concreto en un tuit. Se representan mediante el símbolo "@" seguido del nombre del usuario que se quiere citar. Cuando un tuit contiene una mención el usuario citado recibe una notificación informándole de la existencia de dicho tuit.

- **Enlaces**: Un tuit puede contener también URLs o enlaces a cualquier contenido de la red, como imágenes o páginas web.

- ***Hashtags***: Los *hashtags* son una especie de etiquetas que se emplean para indicar o incluso complementar el contenido de un tuit. Se emplean generalmente para agrupar tuits por temáticas, de modo que si se visita la página de un *hashtag* concreto, podrán verse todos los tuits que se hayan publicado y lo contengan.

1.2. Objetivos del Proyecto

Como se ha comentado, el objetivo de este proyecto es el desarrollo de una aplicación web que permita al usuario realizar búsquedas sobre Twitter y obtener estadísticas de polaridad y una muestra del contenido representado en esos tuits. Para esto se plantea el desarrollo de una aplicación web basada en Django que:

- Permita al usuario realizar búsquedas sobre Twitter con una consulta concreta.

- Descargue un conjunto de tuits resultado de dicha búsqueda.

- Establezca la polaridad de cada uno de los tuits descargados y extraiga estadísticas al respecto.
- Resuma el contenido descargado para mostrar las opiniones más representativas de las que están produciendo los resultados de polaridad mencionados en el punto anterior.
- Almacene todos los resultados producidos en una base de datos de modo que sea posible realizar el análisis de tendencia, es decir, una comparativa temporal de la evolución de los resultados de polaridad para una búsqueda determinada.
- Muestre todos los resultados al usuario de manera atractiva y completa.

1.3. Antecedentes

Existen algunas aplicaciones *on-line* que realizan una tarea similar a la que se pretende desarrollar en este proyecto [1]. Una de las más destacables es Sentiment140. Esta herramienta emplea técnicas de aprendizaje automático [7] para clasificar un conjunto de tuits relativos a una búsqueda del usuario en simplemente positivos o negativos tanto en inglés como en español. Uno de los mayores inconvenientes de esta herramienta es que emplea solamente los tuits más recientes y trabaja con poca cantidad de contenido para extraer las estadísticas, lo cual resta validez a este análisis.

Por otra parte, la aplicación planteada en este proyecto emplea un clasificador de polaridad en seis clases (Muy positivo, positivo, neutro, negativo y muy negativo). Mediante el etiquetado que ofrece esta herramienta se calcularán una serie estadísticas empleando un número elevado de tuits que ofrezca unos resultados relevantes que representen la opinión de la comunidad de Twitter acerca de una búsqueda concreta. Además, el sistema de resumen desarrollado permite ofrecer al usuario una muestra de los tuits que representan esas opiniones, brindando al usuario un análisis más completo.

Otra herramienta ya existente que podía emplearse con un comportamiento ligeramente similar es Tweetmotif; no obstante, y pese a que se podía emplear para analizar resúmenes de sentimiento, Tweetmotif se centraba en el análisis temático de los tuits y no en la polaridad. Además, esta herramienta trabajaba principalmente en inglés, mientras que aquí se busca desarrollar esta herramienta para el idioma español. Tweetmotif ya no se encuentra en funcionamiento, mas se puede consultar su trabajo en [1].

1.4. Organización del documento

A continuación se va a describir brevemente el contenido de cada una de las secciones que contiene esta memoria.

En el Capítulo 2 se incluye un pormenorizado de todas las tecnologías empleadas para el desarrollo de la aplicación y sistema de resumen, así como las librerías de código auxiliares.

[1] Una lista de estas aplicaciones puede encontrarse en http://help.sentiment140.com/other-resources

En el Capítulo 3 se detalla cada una de las partes de las que consta la aplicación, tanto a nivel de interfaz como a nivel interno de Django.

En el Capítulo 4 se desglosa el funcionamiento de la aplicación. Para ello se han establecido los posibles flujos de información con cada una de las interacciones del usuario con la aplicación y se han detallado los pasos que se realizan internamente con cada uno de ellos.

En el Capítulo 5 de detalla la experimentación realizada para desarrollar el sistema de resumen del que dispone la aplicación y cada una de las pruebas que se llevaron a cabo para establecer cual era la configuración idónea en un sistema de resumen de tuits.

Por último, en el Capítulo 6 se concluye el trabajo realizado para este proyecto y se proponen algunos enfoques de trabajo futuro relacionado con el análisis de polaridad y con la aplicación desarrollada.

Al final de la memoria se incluye un glosario que contiene definiciones de los términos considerados relevantes al proyecto. También se ha incluido un anexo con la documentación[2] del código auxiliar que se desarrolló para la aplicación así como para las funciones de resumen.

[2]El código completo se encuentra disponible en https://github.com/javierselva/twitsentim

Capítulo 2

Tecnologías empleadas

En este capítulo se describirán el conjunto de tecnologías y librerías que fue necesario emplear para el desarrollo del proyecto. En primer lugar se comentarán las diferentes herramientas utilizadas para la creación de la aplicación web con la que interactúa el usuario. Estas herramientas fueron usadas principalmente para el desarrollo de la interfaz de usuario, pero también para algunas de las tareas de lógica subyacentes a la aplicación. Por otra parte, a continuación se describirán las librerías empleadas para el tratamiento de la información, tanto a nivel de la aplicación como para algunas de las tareas de tratamiento de la información como la preparación de estadísticas o la elaboración de los resúmenes.

El lenguaje principal subyacente a toda la aplicación es Python 3.4.3 [20] debido a la gran versatilidad de este lenguaje a la hora de trabajar con cadenas de texto (muy necesario cuando se trabaja en tareas de NLP, como es nuestro caso), además de muchas librerías que facilitan este trabajo, como *tokenizadores*, manejo de matrices o extracción automática de vectores de características de texto.

2.1. HTML

HTML [33] es Lenguaje de Marcado de Hipertexto (*HyperText Markup Language*). Es el lenguaje estándar de desarrollo web por excelencia. Este lenguaje es empleado para dar forma y estructura a una web. Permite no solo trabajar con texto, si no también insertar imágenes, tablas, formularios y enlaces que conecten con otro contenido web o multimedia, entre otros. La última versión (HTML5) permite trabajar con contenido multimedia de manera mucho más sencilla que versiones anteriores. No obstante, dadas las necesidades de la aplicación que se desarrolla en este proyecto, no se requiere el uso de esta última versión y se ha trabajado con la anterior (HTML4 [28]), la cual permite la inserción de imágenes, textos, formularios y enlaces, además de poderse trabajar con capas, que es todo lo que hace falta para interactuar con el usuario, tanto para que realice una búsqueda como para mostrarle toda la información de los resultados obtenidos.

Se ha escogido HTML como lenguaje de desarrollo de la aplicación por ofrecer gran sencillez a la hora del desarrollo y todo lo necesario para obtener y mostrar información.

2.2. CSS

CSS [31] es otro lenguaje estándar de desarrollo web que se emplea para definir y modificar los estilos de una página en HTML. Hojas de estilos en cascada o CSS (*Cascading Style Sheets*) se encarga de establecer el aspecto de una web. Entre otras cosas, marca el conjunto de colores, fuentes, tamaño de texto, alineamiento, dimensiones y demás características de los elementos que dispone HTML.

Mediante el empleo de CSS se consigue que la aplicación tenga un aspecto más ameno e intuitivo para el usuario. Para la aplicación se han empleado definiciones básicas de color y dimensiones para dotar a la aplicación de una apariencia sencilla sin dar demasiada importancia a este aspecto de la aplicación. En la Figura 2.1 se puede observar la diferencia entre la interfaz sin CSS (derecha) y la definitiva que si lo emplea (izquierda).

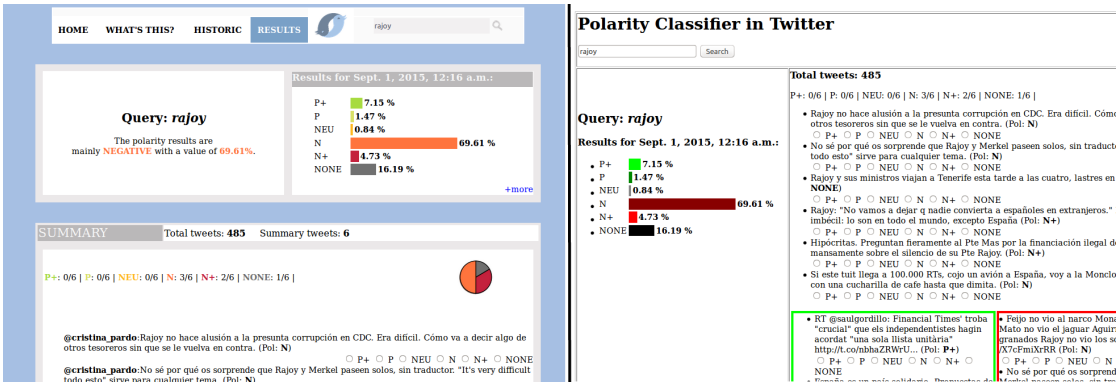


Figura 2.1: Comparativa de la interfaz con y sin CSS

Se ha empleado CSS por dar un gran juego a la hora de diseñar la web, y además permite separar de manera muy simple la estructura de la web (de la que se encarga HTML) de la apariencia, permitiendo realizar modificaciones en uno o el otro de manera independiente y sencilla.

2.3. JavaScript

JavaScript [34] es un lenguaje de *scripting* interpretado pensado para su utilización *on-line*. Se ejecuta por lo general de manera local al usuario para realizar tareas de preprocesado previo al envío de información al servidor. Junto a HTML y CSS es una de las tres tecnologías esenciales de la W3 (*World Wide Web*).

En la aplicación, JavaScript se ha empleado simplemente para el muestreo dinámico de alguna parte de la aplicación, con funciones insertadas en el mismo HTML que permiten modificar el contenido de manera dinámica mediante la interacción del usuario con parte de la interfaz; por ejemplo la capa que informa al usuario de que la aplicación está preparando la información mientras el usuario espera tras realizar la búsqueda.

2.4. Django

Django es un *framework* de código abierto basado en Python para el desarrollo de aplicaciones web. La elección de esta herramienta para el desarrollo de la aplicación web se debe a diversas razones. En primer lugar, permite integrar de manera muy simple la base de datos, la interfaz y la lógica de la aplicación de modo que ahorra al desarrollador el tener que preocuparse de cómo interaccionan las diversas capas entre si y encargándose solo del correcto funcionamiento de cada una de ellas. En segundo lugar, todo esto está implementado en Python, del mismo modo que el sistema de resumen y el clasificador de polaridad, de modo que nos facilita mucho la labor de integrar las funcionalidades internas con la aplicación. Para este proyecto se empleó la versión 1.8.2 de Django [5].

Pese a emplear su propia nomenclatura, el núcleo de Django se puede ver como MVC (*Model-View-Controller*)[32]. El patrón de arquitectura software Modelo-Vista-Controlador consiste precisamente en separar los datos y la lógica de la interfaz. En Django, estas tres partes se entienden como:

- **Modelo**: Un mapeador objeto-relacional que media entre los modelos de datos (definidos como clases de Python) y una base de datos relacional.
- **Vista**: Un sistema que procesa las peticiones HTTP y las devuelve al usuario mediante un sistema de plantillas web.
- **Controlador**: Un sistema basado en expresiones regulares que activa la vista adecuada según la URL solicitada.

En el Capítulo 3 se desglosa cada una de estas partes de manera detallada.

2.5. MongoDB

MongoDB [3] es un sistema de bases de datos NoSQL de código abierto orientado a documentos. En lugar de almacenar la información en tablas como se haría en una base de datos relacional, MongoDB guarda la información en forma de documentos JSON, haciendo que la integración de este sistema de información sea mucho más simple y rápido que otros sistemas de bases de datos.

MongoDB ha sido muy útil para almacenar de manera eficiente grandes conjuntos de tuits. Para trabajar con este sistema de bases de datos se ha empleado la librería PyMongo [19], que es la distribución para Python recomendada en la web oficial de MongoDB y que contiene todas las herramientas necesarias. Para el proyecto se ha empleado la versión 3.0.4 de MongoDB [4].

2.6. Librerías para Python

En este apartado se van a comentar las diferentes librerías de Python que se han empleado para el desarrollo de las distintas funcionalidades de la aplicación.

2.6.1. Tweepy

Tweepy [24] es una librería de código abierto para Python que incluye todo el conjunto de funciones necesarias para comunicar con Twitter mediante las API definidas por este. Las funciones definidas por Tweepy simplifican sobremanera la conexión y búsquedas con Twitter. Por ejemplo, toda conexión a Twitter debe estar certificada con OAuth, y mientras que por defecto habría que configurar esta conexión mediante otra librería como Python-OAuth y establecer cada conexión manualmente, Tweepy simplifica esto con un funciones que simplemente esperan los parámetros para configurar todo automáticamente. En el caso de OAuth, simplemente hay que pasarle los 4 *tokens* necesarios y en el caso de la búsqueda solo tenemos que indicarle los parámetros que solicita Twitter, toda la complejidad de las conexiones la trata internamente simplificando el trabajo inmensamente. Para el proyecto se ha empleado la versión 3.3 de Tweepy.

2.6.2. NumPy

Esta librería [18] es una extensión de Python de código abierto que añade soporte para trabajar con grandes arrays multidimensionales. Incluye la implementación de diversas operaciones matriciales de forma automática. NumPy permite realizar toda clase de operaciones complejas con matrices de manera eficiente y sencilla, por ello se convierte en la librería ideal siempre que haya que tratar con grandes cantidades de información en forma de matriz. Esta librería está destinada a un nivel inferior a Python, es decir sus funciones están dirigidas a trabajar sobre CPython, con una implementación que trata de sacar el máximo partido a las librerías en C de Python, buscando la máxima eficiencia.

En nuestro caso, tanto para el clasificador como para el sistema de resúmenes automáticos ha demostrado ser una librería de gran ayuda. En el caso del sistema de resumen, todas las matrices con las que trabaja scikit-learn (Subsección 2.6.3) están representadas como matrices de NumPy. Una función muy útil que incluye esta librería y que es primordial para el correcto funcionamiento del resumidor automático es la descomposición lineal SVD, la cual ha simplificado la implementación de los resúmenes por ser una operación matricial bastante compleja. Para el proyecto se ha empleado la versión 1.9.2 de NumPy.

2.6.3. Scikit-Learn

Es una librería de código abierto [27] para Python que añade funciones de aprendizaje automático. Implementa algoritmos de clasificación, regresión y *clustering* incluyendo algoritmos de Vectores de Máquinas Soporte o SVM (*Support Vector Machines*) que se emplean para el clasificador de polaridad [6].

Esta librería se ha empleado para crear automáticamente vectores de características (como veremos en el Capítulo 5), es decir, dado un conjunto de documentos, genera automáticamente matrices con características de las palabras en los documentos, agilizando mucho el tratamiento de los tuits a la hora de realizar los resúmenes. Para este proyecto se ha empleado la versión 0.16.1 de Scikit-Learn [26].

2.6.4. MatPlotLib

Es una librería de dibujo (código abierto) para Python. Se emplea para representar gráficas y demás muestras visuales de información. Se ha escogido por la gran cantidad de posibilidades de dibujo que ofrece, permitiendo representar gráficas de manera simple y sin perder completitud.

En nuestro caso, todas las gráficas que se muestran en la aplicación y en la memoria han sido dibujadas empleando esta librería. Para el proyecto se ha utilizado la versión 1.4.3 de MatPlotLib [11].

Capítulo 3
Estructura de la aplicación

En este capítulo se detalla la estructura de la aplicación. Esto comprende desde las diferentes partes que forman la interfaz hasta cómo se han implementado las diferentes partes del patrón MVC de Django. Después, en el Capítulo 4 se detallará el funcionamiento de cada una de estas partes y el "recorrido" que sigue la aplicación con cada acción del usuario.

3.1. Interfaz de usuario

En primer lugar se va a explicar brevemente la distribución de la interfaz de usuario. La aplicación web tiene cuatro apartados claramente diferenciados:

- **Inicio**: En esta sección hay un cuadro de búsqueda que permite al usuario realizar las búsquedas que desee y además se muestra una lista con las 10 búsquedas más realizadas en la web, de modo que el usuario pueda seleccionarlas para que se muestren las gráficas con los resultados históricos de esa búsqueda. En la Figura 3.1 se muestra la página de inicio con búsquedas de ejemplo que se realizaron durante la etapa de desarrollo de la aplicación.

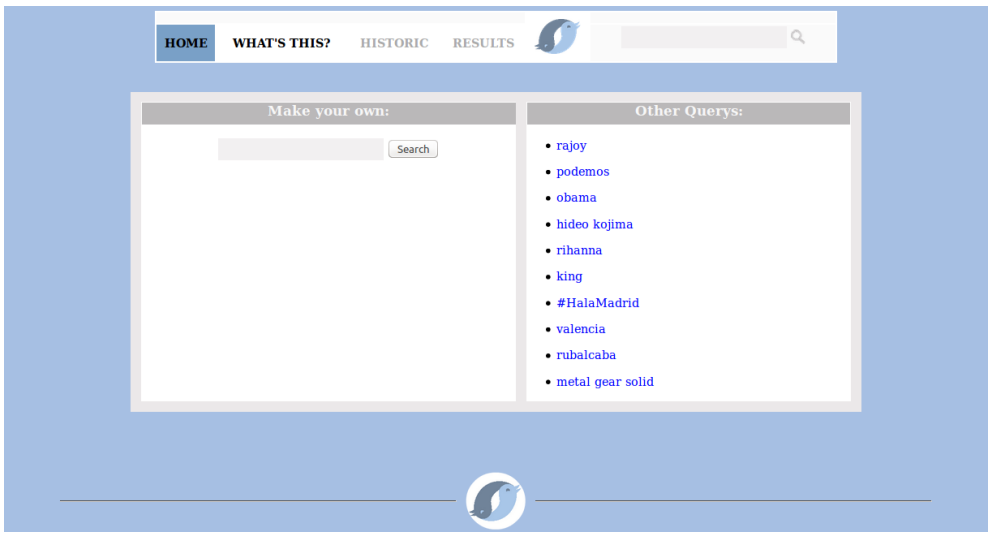


Figura 3.1: Muestra de la página principal de la aplicación

- **Acerca de**: Es una sección que simplemente ofrece algunos detalles acerca de la aplicación en si y su funcionamiento.

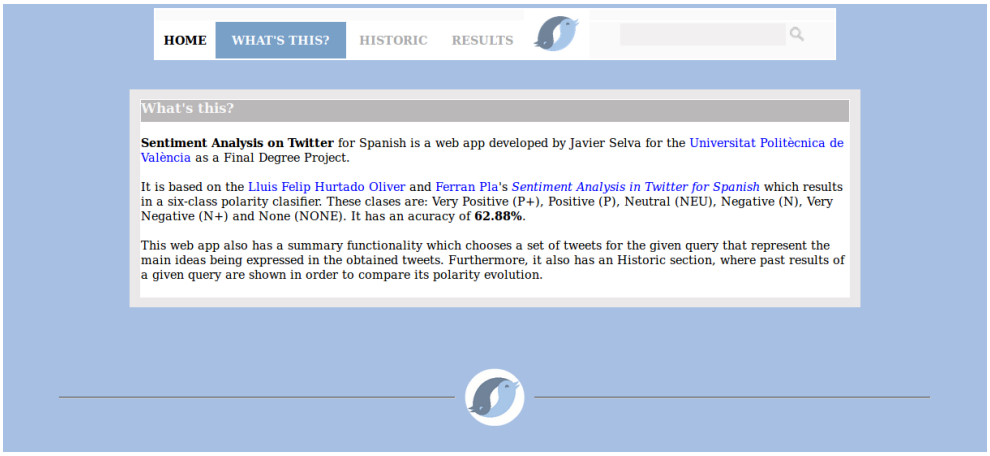


Figura 3.2: Muestra de la página "Acerca de" de la aplicación

- **Histórico**: Este apartado de resultados incluye dos gráficas y una tabla que muestran al usuario los resultados históricos de una búsqueda concreta. En particular, permite observar la evolución de polaridad de esa búsqueda. En la Figura 3.3 se muestra la página de resultados históricos con la búsqueda "podemos" que se realizó durante la etapa de desarrollo de la aplicación.

Figura 3.3: Muestra de la página de resultados históricos

- **Resultados**: Este apartado muestra al usuario los resultados de una búsqueda que haya realizado en ese momento. Incluye las estadísticas de polaridad y los resúmenes tanto generales como solo de los tuits positivos y negativos. Además añade la posibilidad de

que el usuario etiquete a mano los tuits resumen con su polaridad real correspondiente y enviarlos a modo de retroalimentación.

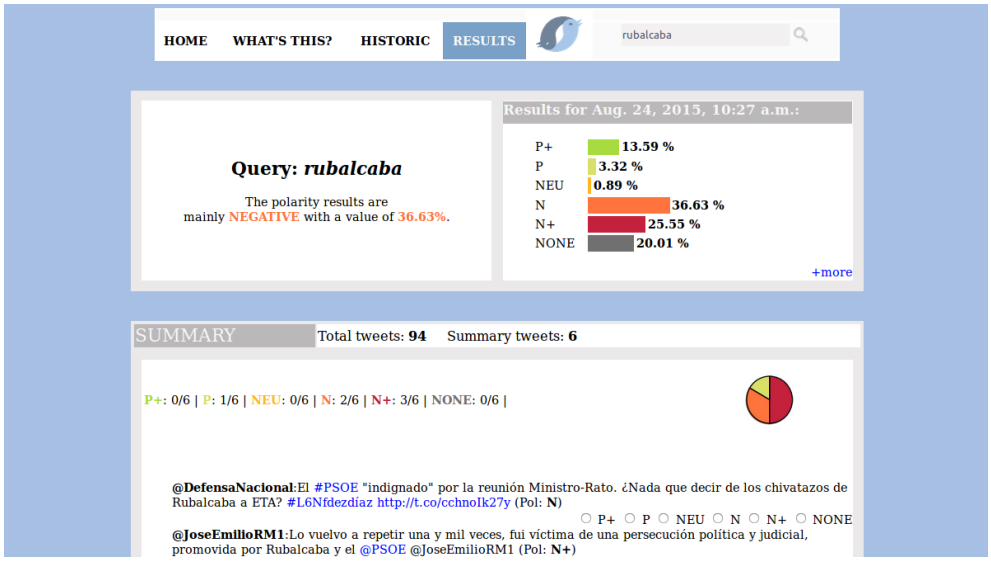


Figura 3.4: Muestra de la página de resultados concretos de una búsqueda

Todos los apartados de la aplicación incluyen en la parte superior un cuadro de búsqueda, de modo que desde cualquier lugar del sitio sea posible realizar una nueva búsqueda y acceder al apartado de resultados de la misma. Cada una de las secciones requirió una plantilla *.html* y un *.css* común a todas ellas para dar una apariencia uniforme. Además, se preparó una vista Django (como veremos a continuación) para cada una.

3.2. Django: MVC

Aquí se va a detallar la implementación de cada una de las tres partes del patrón MVC que sigue Django. En primer lugar se detallará la estructura de la base de datos (modelo), definida como clases de Python. En segundo lugar, se explicará cada una de las vistas necesarias para el funcionamiento de la aplicación. Y por último se comentarán las expresiones regulares que se utilizaron para que el controlador supiese asignar una vista a cada una de las posibles URLs solicitadas por la aplicación.

3.2.1. Modelos

El modelo de la base de datos se define como un conjunto de clases Python. Después, con una serie de comandos de Django, se crea la base de datos automáticamente con los campos que se hayan definido. En nuestro caso se han creado un total de cuatro clases:

- **`Query`**: Esta tabla contiene simplemente el texto de todas las búsquedas que se hayan realizado alguna vez. Se ha incluido en una tabla separada para simplificar algunos aspectos de la aplicación, como la necesidad de recuperar una lista de todas las búsquedas que

se hayan realizado alguna vez, lo cual sería más complejo si cada búsqueda genera una entrada diferente (con repeticiones). Consta de los siguientes campos:

  - **id**: Clave primaria, se genera automáticamente y es autoincremental.
  - **query_text**: Contenido textual de las búsquedas.

- **Query_data**: Contiene toda la información de una búsqueda concreta. Una misma búsqueda puede realizarse repetidas veces, de modo que esta tabla contendrá los resultados de polaridad de cada vez que se realice la búsqueda. Está formada por los siguientes campos[1]:

  - **id**: Clave primaria, se genera automáticamente y es autoincremental.
  - **query_id**: Clave ajena referencia a la tabla *Query* que señala de qué consulta son estos resultados.
  - **query_date**: Fecha en la que se realizó la búsqueda.
  - **p_pos_p**: Porcentaje de tuits con una polaridad muy positiva.
  - **p_pos**: Porcentaje de tuits con una polaridad positiva.
  - **p_neu**: Porcentaje de tuits con una polaridad neutra.
  - **p_neg**: Porcentaje de tuits con una polaridad negativa.
  - **p_neg_p**: Porcentaje de tuits con una polaridad muy negativa.
  - **p_none**: Porcentaje de tuits sin polaridad.
  - **hm_tweets**: Cantidad de total de tuits que se obtuvieron en esta búsqueda (tras eliminar retuits).

- **Summary_tweeet**: En esta tabla se almacenarán los tuits resumen de una búsqueda concreta. Los campos necesarios para almacenar un tuit resumen son:

  - **id**: Clave primaria, se genera automáticamente y es autoincremental.
  - **query_id**: Clave ajena referencia a la tabla *Query_data* que señala de qué búsqueda concreta es este tuit.
  - **tweet_id**: Esta campo contiene el identificador del tuit asignado por Twitter.
  - **tag**: Representa la etiqueta del resumen. La aplicación generará tres resúmenes diferentes, uno con todos los tuits descargados (sin retuits), otro solo con los tuits cuyo resultado en la clasificación de polaridad fuese positivo o muy positivo y otro con los tuits con resultados negativos o muy negativos. Así pues este campo de la tabla indicará si el tuit pertenece al resumen general (etiqueta "ALL"), al positivo ("POS") o al negativo ("NEG").
  - **tweet_text**: Contiene el texto del tuit con las entidades transformadas en enlaces HTML, de modo que al insertarlos en la web cada una enlace correctamente a donde debiera enlazar.
  - **tweet_pol**: Indica la polaridad del tuit.
  - **tweet_user**: Contiene el nombre del usuario que publicó el tuit.

[1] En esta clase todos los porcentajes están calculados sobre el total de los tuits, es decir, sin eliminar los retuits

- **`Test_tweet`**: Aquí se almacenan los tuits que se envían con una etiqueta correcta vía retroalimentación. Estos tuits se podrán emplear en un futuro para añadirlos al entrenamiento del clasificador de polaridad y mejorar así su funcionamiento. Tan solo contiene tres campos necesarios:
  - **`id`**: Clave primaria, se genera automáticamente y es autoincremental.
  - **`tweet_text`**: Contenido del tuit en texto plano.
  - **`tweet_pol`**: Polaridad enviada del tuit.

De este modo, una vez ejecutados los comandos Django adecuados, se crea una base de datos en SQLite (por defecto) que contiene todas las tablas definidas mediante estas clases. En el futuro, cada vez que se desee acceder al contenido de las tablas, se realizarán los accesos mediante funciones Python asociadas a estas clases.

Este contenido se encuentra en el archivo `models.py` del proyecto.

3.2.2. Vistas

En este apartado se van a detallar cada una de las vistas que se crearon. Cada una de ellas se activa con una solicitud concreta de una URL por parte del usuario, y realiza las tareas necesarias para preparar y devolver una plantilla con el contenido adecuado para mostrarla al usuario. En total se han creado seis plantillas, cuatro de ellas se corresponden con los cuatro apartados de la interfaz, y las otras dos son vistas que preparan y almacenan información y devuelven el control a otra vista. En este apartado simplemente se va a describir lo que hace cada una, mientras que en el Capítulo 4 veremos cada paso que siguen para detallar el cómo lo hacen.

- **`index`**: Se corresponde con el apartado de "Inicio" de la interfaz de usuario, simplemente obtiene una lista con cada una de las búsquedas realizadas alguna vez y manda la información a la plantilla de inicio.
- **`whats_this`**: Está ligada al apartado "Acerca De" de la página. Más sencillo que la vista anterior (pues la plantilla de este apartado carece de elementos variables), simplemente dirige a la plantilla que contiene los elementos de este apartado.
- **`query_page`**: Esta vista se llama cada vez que el usuario realiza una búsqueda. Se encarga de preparar todos los resultados necesarios antes de devolver el control a la vista que se encarga de "montarlos" para entregarlos al usuario a través de la plantilla de resultados.
- **`show_results`**: Esta vista es llamada a través de la vista "`query_page`" una vez los resultados están preparados. Esta vista se encarga de darles el formato adecuado antes de entregarlos a la plantilla para que los muestre al usuario.
- **`show_historic`**: Esta vista se llama cada vez que el usuario intenta acceder a los resultados históricos de una búsqueda, ya sea desde la página de inicio con uno de los enlaces a las búsquedas más realizadas o a través de la página de resultados de una búsqueda concreta. Basándose en los resultados que prepara "`query_page`" elabora los gráficos históricos que se mostrarán al usuario y los envía a la plantilla de resultados históricos.

- **`add_text`**: Esta vista se llama cada vez que el usuario envía retroalimentación. Simplemente almacena los tuits que se han etiquetado a mano en la tabla "`Test_tweet`".

Podríamos decir que en estas vistas en donde "sucede" la mayor parte de la aplicación. Es donde se realizan todas las acciones, como bajar los tuits, clasificarlos o resumirlos, es donde se accede a la base de datos y en definitiva es donde se generan los resultados. En otras palabras, es el esqueleto de la aplicación.

Este contenido se encuentra en el archivo `views.py` del proyecto.

3.2.3. URLs

En este apartado se van a comentar brevemente las expresiones regulares que representan a cada una de las vistas. Básicamente, en Django cada posible URL aceptada por una aplicación tiene una vista asociada que realiza las acciones pertinentes para devolver al usuario la información que ha solicitado. Las expresiones regulares que describen una URL en una aplicación Django consisten en el contenido de la misma que se encuentre a continuación del nombre de la aplicación; en nuestro caso la aplicación fue llamada **tweetclass**, así que la URL base sería `host.com/tweetclass/` [2] En este caso, las URLs asociadas a cada vista son:

- **`index`**: Es la dirección de inicio de la página, la que se entregará al usuario por defecto cuando acceda a la aplicación sin más detalle. De este modo a ULRL no incluye ninguna información adicional. La expresión regular representa la cadena vacía "^$".
  Ejemplo: `host.com/tweetclass/`

- **`whats_this`**: Casi del mismo modo que la anterior, esta es una página fija, de modo que su URL también lo es: "^`whats_this`/$"
  Ejemplo: `host.com/tweetclass/whats_this/`

- **`query_page`**: Esta página es a la que redirigen todos los formularios de búsqueda. Carga la vista con el mismo nombre. Dado que es una página solo de carga (no hay plantilla asociada) el usuario no llega a acceder a ella por que termina direccionando a la página de resultados, no obstante es de vital importancia, pues prepara toda la información necesaria para mostrar los resultados. Además, dado que la información de la búsqueda a realizar le llega mediante el método POST[3] de HTML, la URL será también genérica. La expresión regular se representa mediante el inicio de línea, la cadena "`query_text`" y el final de línea: "^`query_text`/$"
  Ejemplo: `host.com/tweetclass/query_page/`

- **`show_results`**: Este apartado es el encargado de preparar la vista que muestra los resultados al usuario. Toda la información relacionada con esta vista depende enteramente de una búsqueda concreta, en otras palabras, de un objeto "`Query_data`", así que para representar la URL y que se permita acceder a la página de resultados de una búsqueda específica, se ha incluido el identificador de la búsqueda en la URL. De este modo, además de la cadena "`show_results`" en la expresión regular se ha incluido un número representando este identificador, y una letra '*F*' que indica si el formulario de *feedback* ha

[2] Se ha empleado "host.com" como genérico. La aplicación podría estar alojada en cualquier lugar.
[3] El método POST es uno de los métodos de solicitud de HTTP, se encarga de enviar información

sido enviado ya o no, en el primer caso, el formulario no se mostraría al usuario: "^F?[0-9]+/`show_results`/$"
Ejemplo: `host.com/tweetclass/229/show_results/`

- **`show_historic`**: De modo muy similar al anterior, prepara los resultados históricos de una "`Query_data`" concreta, así que se incluye el identificador en al URL:
"^F?[0-9]+/`show_historic`/$"
Ejemplo: `host.com/tweetclass/174/show_historic/`

- **`add_text`**: En este caso, el comportamiento es muy parecido a "`query_page`" la cual solo se encarga de tratar información (recibida a través del método POST) y redirigir al usuario, de modo que la URL es muy similar a aquella: "^`add_test`/$"
Ejemplo: `host.com/tweetclass/add_test/`

De este modo, cuando al controlador le llega una URL busca cual de las expresiones regulares definidas encaja, y redirige el control a la vista adecuada.

Este contenido se encuentra en el archivo `urls.py` del proyecto.

Capítulo 4
Flujo de la aplicación

En este capítulo se detallan todos los pasos de funcionamiento de la aplicación. Desde la realización de la búsqueda de un usuario hasta que se muestran todos los resultados. El objetivo de este apartado es examinar con detenimiento cada uno de los pasos que se siguen para preparar los resultados y comentar los problemas que se encontraron en cada uno de ellos además de las soluciones planteadas.

Se ha dividido el capítulo en diversos apartados, cada uno de los cuales representando una posible acción del usuario partiendo de un estado concreto de la aplicación. En la Sección 3.1 puede verse el resultado de cada una de las acciones que será uno de los cuatro posibles apartados de la aplicación.

4.1. Acceso al inicio

Tanto al acceder a la aplicación inicialmente (accediendo a la URL inicial que se comentó en el apartado anterior "`host.com/tweetclass/`") como al presionar sobre el botón de inicio desde cualquier apartado de la aplicación, el controlador redireccionará a la vista "*index*" la cual realiza una consulta a la base de datos para obtener cada una de las búsquedas realizadas alguna vez y el número de veces que se ha solicitado cada una, de modo que enviará a la plantilla las 10 consultas más realizadas ordenadas de mayor a menor.

Así pues, esto resulta en una lista de enlaces a los resultados históricos de cada una de las diez búsquedas más realizadas en el apartado de inicio. En la Figura 3.1 podemos ver esta lista en la parte derecha de la interfaz.

Además, bajo la barra de búsqueda se ha añadido un enlace a la documentación de Twitter donde se especifican los operadores que se pueden emplear para construir una consulta[1], de modo que el usuario sea capaz de realizar búsquedas más exactas.

4.2. Acceso a resultados históricos desde el inicio

Una vez creada la lista en el apartado anterior, cada vez que se acceda a uno de los enlaces se accederá al apartado de resultados históricos para esa búsqueda. Como vimos en la Subsección 3.2.3, la vista "`show_historic`" espera recibir el identificador de un objeto "`Query_data`", es decir, de una búsqueda concreta. Sin embargo no se dispone de esta información en el inicio,

[1] `https://dev.twitter.com/rest/public/search`

pues tendría un coste temporal demasiado alto prepararla para cada uno de los enlaces cada vez que se carga la página inicial. Así pues, se optó por enviar a la vista de resultados históricos el texto de la búsqueda mediante un método POST oculto y que la propia vista se encargase de recuperar todos los resultados de esa búsqueda. En concreto, adquiere los porcentajes de polaridad obtenidos cada vez que se realizó esa consulta.

Además, como la vista necesita recibir un identificador a través de la URL, se solicita siempre la dirección `host.com/tweetclass/000/show_historic`; de este modo cuando la vista recibe el identificador "000" sabe que tiene que recuperar los resultados históricos mediante el texto de la búsqueda contenido en el campo "`real_id`" del método POST.

Se optó por esta solución en lugar de añadir otra vista por que la modificación es ínfima, la única diferencia entre las dos vistas habría sido el modo en que se obtienen la lista de resultados, mientras que el resto se mantendría igual; de esta forma y por limpieza, se añadió esa comprobación al inicio de la vista.

Una vez recuperados todos los resultados de la búsqueda, se dibujan las diferentes gráficas de evolución de la polaridad:

- **Gráfico genérico**: Muestra la evolución de los porcentajes de cada una de las polaridades. En la Figura 4.1 se puede ver un ejemplo de este tipo de gráfica para la búsqueda "podemos".

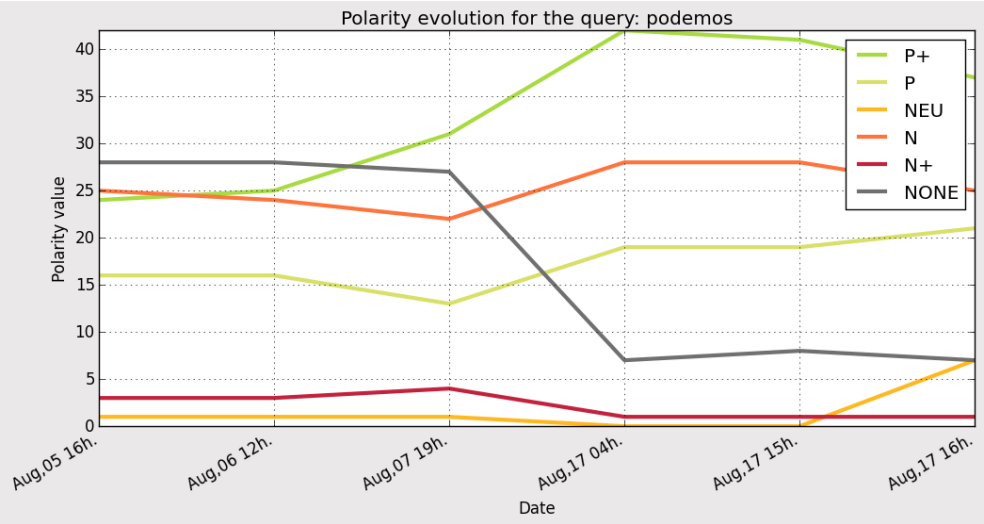


Figura 4.1: Ejemplo de gráfica genérica

- **Gráfico resumen**: Para este gráfico se calcula una puntuación de polaridad valorando de diferente manera cada uno de los porcentajes, en concreto, esta puntuación se calcula cómo $50 + 1{,}5 * P^{+} + P - N - 1{,}5 * N^{+}$ de modo que una valoración casi neutra es cercana a 50, positiva por encima de 50 y negativa por debajo. En la figura Figura 4.2 se puede ver un ejemplo de este tipo de gráfica para la búsqueda "podemos" equivalentes a los resultados de la gráfica anterior.

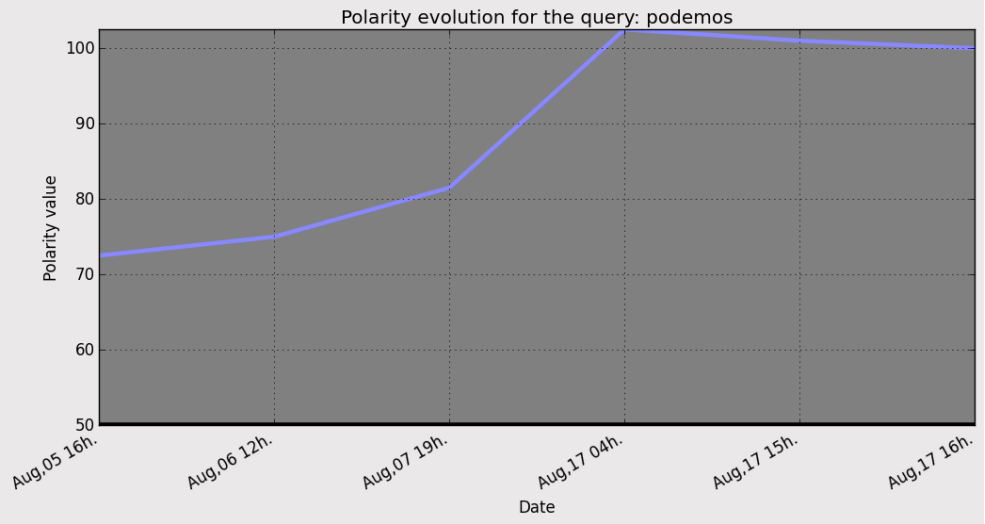
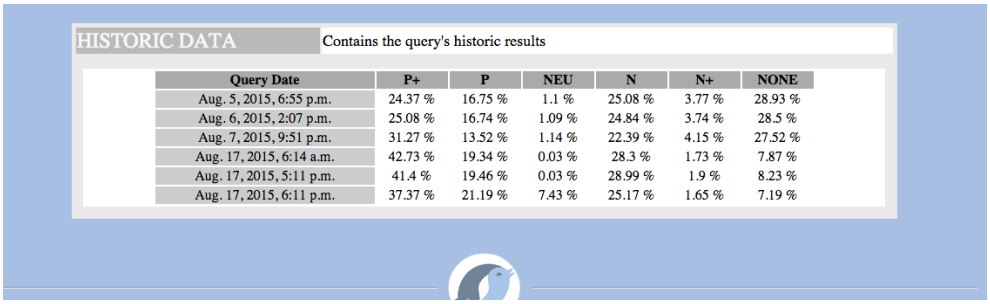


Figura 4.2: Ejemplo de gráfica resumen

Una vez generados y guardados ambos gráficos, se envía a la plantilla toda la información, tanto la ruta donde se han guardado las dos imágenes como la lista de todos los resultados para que se construya una tabla que muestre el historial de polaridades, tal y como se puede apreciar en la Figura 4.3.


HISTORIC DATA — Contains the query's historic results

Query Date	P+	P	NEU	N	N+	NONE
Aug. 5, 2015, 6:55 p.m.	24.37 %	16.75 %	1.1 %	25.08 %	3.77 %	28.93 %
Aug. 6, 2015, 2:07 p.m.	25.08 %	16.74 %	1.09 %	24.84 %	3.74 %	28.5 %
Aug. 7, 2015, 9:51 p.m.	31.27 %	13.52 %	1.14 %	22.39 %	4.15 %	27.52 %
Aug. 17, 2015, 6:14 a.m.	42.73 %	19.34 %	0.03 %	28.3 %	1.73 %	7.87 %
Aug. 17, 2015, 5:11 p.m.	41.4 %	19.46 %	0.03 %	28.99 %	1.9 %	8.23 %
Aug. 17, 2015, 6:11 p.m.	37.37 %	21.19 %	7.43 %	25.17 %	1.65 %	7.19 %

Figura 4.3: Ejemplo de tabla de evolución de polaridades

La primera columna de la tabla está formada, asimismo, por enlaces que redirigen a la página de los resultados concretos. Aquí el usuario podrá consultar, además de los resultados de polaridad, los resúmenes que se generaron. Con esto se permite realizar un estudio del contenido que dio lugar a esos valores y establecer la relación entre determinados sucesos y la polaridad obtenida.

4.3. Acceso a resultados históricos desde resultados concretos

La otra forma de acceder a la página de resultados históricos es desde la página de resultados de una búsqueda concreta, en la cual se dispone del identificador del elemento "`Query_data`" actual, y por lo tanto de su identificador. Así pues, dado que nos encontramos en la página de resultados concretos, disponemos del identificador que necesitamos, y simplemente tomando

la URL `host.com/tweetclass/xyz/show_results/`, acceder al apartado de resultados históricos es tan sencillo como acceder al URL `host.com/tweetclass/xyz/show_historic/` manteniendo el valor del identificador **xyz**. En esta ocasión, cuando el controlador redirige a la vista "`show_historic`" la lista de resultados de la búsqueda se extrae directamente con el identificador en lugar de hacerlo a través del texto de la búsqueda como en el apartado anterior. Salvo esta diferencia, el resto de flujo de información es idéntico al explicado arriba en la Sección 4.2.

4.4. Realización de una búsqueda

Desde cualquier apartado de la aplicación se puede realizar una nueva búsqueda de tuits sobre los que efectuar clasificación de polaridad y resúmenes. Así pues, cada vez que un usuario realiza una búsqueda se solicita la URL `host.com/tweetclass/query_page` y se le envía el texto de la consulta realizada mediante el método POST. El controlador activará la vista correspondiente (`query_page`) la cual se encarga de realizar el núcleo de trabajo de la aplicación.

El primer paso es comprobar si esa búsqueda se ha realizado alguna vez, y en caso contrario crearla. Para esto se realizan accesos a la base de datos sobre la tabla "`Query`". Una vez se tiene la referencia al objeto de la búsqueda, se piden a Twitter un total de 1000 tuits relativos a esta búsqueda. Twitter permite descargar tuits populares, recientes o ambos. En nuestro caso se decidió descargar tuits de ambos tipos, pues permite obtener más variedad no centrándose solo en tuits que hayan adquirido cierta popularidad o hayan sido emitidos muy recientemente. Esta parte del proceso (la descarga de los tuits) ha demostrado ser el cuello de botella de la aplicación, que abarca aproximadamente un 65 % del tiempo total de carga. Se han realizado pruebas con diferentes maneras de bajarlos, y la manera actual (mediante Cursores [23]) ha demostrado ser, pese a todo, la más eficiente.

Una vez descargados los tuits se genera una lista con los mismos (eliminando aquellos que sean retuits) representados como diccionarios. Se ha elegido esta representación en lugar de la representación en forma de clase por que permite añadir campos de manera dinámica en cualquier momento. Los campos que se extraen en la lista de tuits:

- **`id`**: El identificador del tuit en forma de cadena de caracteres.
- **`text`**: Contenido textual del tuit eliminando saltos de línea.
- **`date`**: Fecha en la que se publicó el tuit.
- **`retweet_count`**: Número de veces que el tuit ha sido retuiteado.
- **`favorite_count`**: Número de veces que el tuit ha sido marcado como favorito.
- **`user`**: Autor que publicó el tuit originalmente.
- **`followers`**: Número de seguidores que tiene el usuario que publicó el tuit.
- **`friends`**: Número de usuarios que son seguidos por el usuario que publicó el tuit (amigos).

- **`rt_corpus`**: Número de veces que el tuit ha sido retuiteado en el conjunto de tuits descargados.

En este paso, cuando hubo que eliminar los retuits (para el sistema de resumen) se planteó el problema de en qué momento eliminarlos. Para detectar si un tuit se trataba de un retuit había que buscar en el tuit descargado el campo "`retweeted_status`", si no lo contenía era un tuit original y en caso contrario se trataba de un retuit. En el segundo supuesto este campo contendría el tuit original. Si se eliminaban los retuits al momento de recibirlos, se perdía cantidad de información a la hora de calcular las estadísticas de polaridad, haciendo que esta perdiese relevancia. Por otro lado, si se pretendía eliminarlos después de realizar la clasificación había que mantener no solo información de qué tuits eran retuits, si no una lista con los tuits originales, incrementando la cantidad de información que había que manejar, haciendo más incómodo el tratamiento de los tuits. Además el "mismo" tuit se clasificaría varias veces debido a que el texto de los retuits es el mismo y la polaridad que obtendrá cada uno de ellos es también la misma, siendo redundante clasificar un número indeterminado de retuits y además el original. Así pues, se optó por una solución en la que se eliminaban los retuits pero se guardaban los tuits originales de los que procedían (asegurando que no existan repeticiones) y se contaba cuantas veces ese tuit había sido retuiteado en el conjunto de tuits descargados. De este modo se eliminaba el problema de la pérdida de información y de repetir la clasificación de los tuits, reduciendo la carga de trabajo y simplificando la estructura de datos.

Una vez se dispone de la lista de tuits se genera una lista solo con los textos de los mismos y se pasa al clasificador de polaridad. Éste devolverá una lista de polaridades en forma de cadenas de caracteres (una para cada tuit de la lista), a saber: **P+** (Muy positiva), **P** (Positiva), **NEU** (Neutra), **N** (Negativa), **N+** (Muy negativa) y **NONE** (Ninguna). A continuación se añade la polaridad de cada tuit como un nuevo campo de los diccionarios.

El siguiente paso es realizar los resúmenes. En total se realizan tres: el primero con todos los tuits de los que se disponen; el segundo solo con los tuits que han sido etiquetados como positivos o muy positivos; y el tercero solo con los tuits que han sido etiquetados como negativos o muy negativos. Uno de los problemas encontrados en este paso es debido al sistema que se emplea para realizar estos resúmenes. Como se verá en el Capítulo 5 este sistema emplea una descomposición matricial conocida como SVD que se encuentra implementada en la librería NumPy. No obstante tiene un número de iteraciones limitado que si se alcanza ocasiona que la descomposición no converja y por lo tanto que el resumen no pueda generarse. Por esta razón ha tenido que capturarse esta excepción e indicar al usuario que ha habido un fallo en la producción del resumen.

Una vez calculadas las polaridades y preparados los resúmenes ya está toda la información lista para mostrarse por la plantilla al usuario, no obstante, antes de esto hay que almacenar los resultados en la base de datos. En primer lugar se almacenan las polaridades, para ello se cuenta cuantos tuits se han clasificado en cada polaridad, aplicando los correspondientes multiplicadores de retuit almacenados en el campo `rt_corpus` y se calcula el porcentaje de cada una obtenida. A continuación se almacenan los tres resúmenes realizados. Por último, se almacena el conjunto total de tuits descargados.

Al principio esto último supuso un gran problema de tiempo. Almacenar aproximadamente

mil tuits suponía una gran parte del tiempo, siendo entonces el cuello de botella. Por esta razón se decidió almacenar los tuits en un segundo hilo, de modo que, dado que no se iba a acceder a esa tabla para el muestreo de los resultados podrían guardarse estos tuits por un lado mientras se accedía a los resultados de polaridad y resúmenes por el otro. De este modo, se decidió almacenar el conjunto de tuits masivos en otra base de datos empleando MongoDB. Una vez realizadas estas modificaciones, el almacenamiento del grueso de los tuits resulta mucho más rápido, dejando de ser un problema.

A continuación se redirige a la vista `show_results` la cual se encarga de extraer solo la información necesaria de la base de datos para preparar los resultados en el formato necesario antes de pasarlos a la plantilla HTML. El proceso almacena y luego extrae de la base de datos en lugar de pasarlo directamente por que en Django, transmitir información entre vistas es mucho más simple de esta manera. Así pues esta vista recibe el identificador de la `Query_data` actual y con él obtiene de la base de datos toda la información que necesita para preparar la interfaz: resultados de polaridad, el texto de la búsqueda y los tres resúmenes. Después se calcula el tamaño de cada una de las barras que indicarán el porcentaje de cada uno de los seis valores de polaridad. Estas barras se representarán como tablas con el color de la polaridad y diferentes tamaños (proporcionales al porcentaje correspondiente).

El siguiente paso es dibujar el pequeño gráfico circular que indicará la proporción de tuits de cada polaridad en el resumen general y por último calcular qué polaridad tiene el valor máximo, es decir en cuál de las polaridades han sido clasificada la mayoría de tuits.

Dado que a la plantilla de muestreo de resultados hay que enviarle el contenido con el formato en el que queremos insertarlo, las entidades ha habido que transformarlas en enlaces HTML antes de mandarlos al usuario. En un primer enfoque esta transformación se hacía mediante el campo "*entities*" que incluyen los tuits descargados y que contiene toda la información de todas las menciones, *hashtags* y enlaces contenidos en el tuit en cuestión. Esto requería almacenar esta información en los diccionarios para procesarla tras la realización de los resúmenes, no obstante se observó que en las ocasiones en que un tuit contenía una imagen, Twitter la entregaba como un enlace que no se incluía en la lista de entidades. Por eso y por simplificar el diccionario se optó por transformar todas las entidades mediante expresiones regulares que detecten las entidades y las conviertan en enlaces HTML.

Una vez transformadas las entidades, cuando se inserten en la plantilla los enlaces enlazarán a donde se suponga que deben enlazar, las menciones enlazarán con el perfil del usuario mencionado y los *hashtags* con la página del *hashtag* correspondiente.

Esta transformación no se ha realizado antes de almacenar los resúmenes en la base de datos por dos razones: la primera es que interesa tener los resúmenes limpios, es decir, sin código HTML que emborrone el contenido, y la segunda (relacionada con la primera) es que cuando se almacena la retroalimentación, se recupera el texto de los resúmenes para almacenarlos en la tabla `Test_tweet` y se necesita que ese texto sea puramente el contenido en el tuit, pues el objetivo es emplear ese *feedback* como herramienta para mejorar la precisión del clasificador.

El último paso es enviar toda la información a la plantilla, la cual estructurará la información en una web HTML y la mostrará al usuario un resultado similar al visto en la Figura 3.4.

4.5. Envío de retroalimentación

Esta última acción muy sencilla se realiza cuando el usuario ha etiquetado a mano todos los tuits resumen de una página de resultados y presiona sobre el botón "*Send feedback*", lo cual enviará la información mediante el método POST. En ese momento el controlador redirige a la vista `add_test` la cual simplemente extrae las nuevas polaridades y almacena el texto de los tuits junto a sus polaridades actualizadas en la tabla `Test_tweet`.

Una vez hecho esto, redirige a la vista `show_results` para que vuelva a mostrar al usuario la misma página en la que se encontraba antes de presionar el botón. No obstante una vez enviado el *feedback* la página de resultados ya no incluirá el formulario de envío de retroalimentación para evitar la repetición de tuits.

Capítulo 5
Sistema de Resumen

En este capítulo se detallan los pormenores del sistema de resumen que se emplea para seleccionar la información más representativa del conjunto de tuits descargados según las búsqueda del usuario. En primer lugar veremos las características principales del sistema de resumen que se utiliza tanto para resumir el conjunto total de tuits como para hacerlo con los tuits solo positivos o solo negativos; en segundo lugar se detallan todas las pruebas que se realizaron para encontrar la mejor configuración del sistema de resumen para tuits en español.

Uno de los objetivo de este proyecto es extraer la mayor cantidad de información posible acerca de la polaridad de los tuits. Así pues, no solo interesa cuántos han sido clasificados con una polaridad u otra si no también qué información están expresando. Buscamos pues un subconjunto de tuits que representen a la información contenida en un conjunto más grande, es decir, un resumen que incluya las ideas u opiniones más representativas.

La realización de resúmenes automáticos ha sido objeto de estudio desde hace aproximadamente 60 años [13]. A lo largo de este tiempo se han planteado diferentes maneras de enfocar la elaboración de resúmenes de grandes conjuntos de información, no obstante, los más significativos y que adquieren más relevancia en la práctica son la extracción y la abstracción [30]:

- El resumen automático mediante **extracción** consiste en la selección de una serie de frases del documento a resumir sin producir nuevo texto. La dificultad de este sistema consiste en decidir qué frases son realmente relevantes y cuáles deben formar parte del resumen.
- Por otra parte, los resúmenes mediante **abstracción** consisten en parafrasear el documento original, extrayendo los conceptos más importantes para crear nuevo texto que resuma el original, tal y como se haría en un resumen realizado a mano. El problema de esta técnica es la necesidad de emplear técnicas de Generación y Procesamiento de Lenguaje Natural, más sofisticadas y que están fuera de los objetivos de este proyecto.

En el caso del presente proyecto se ha optado por el uso de técnicas de extracción, por presentar unas características más apropiadas a las necesidades del problema, esto es, buscamos "extraer" los tuits más relevantes de un conjunto de tuits. Para ello se considera el conjunto total de tuits como el documento a resumir y cada tuit una frase de este.

Existen diferentes sistemas para establecer la importancia de una frase en el texto, como la posición de las frases en éste y la de las palabras dentro de la frase; otros sistemas emplean

análisis semántico de cadenas léxicas para determinar la importancia de las frases [14] y así extraer las más significativas. No obstante, un sistema más reciente emplea técnicas algebraicas para determinar la relevancia de una frase. Este sistema es conocido como Análisis Semántico Latente o LSA (*Latent Semantic Analysis*). El algoritmo LSA permite determinar la similitud semántica entre varios bloques de texto analizando la relación entre los documentos y los términos que contienen empleando información acerca del uso de las palabras en su contexto. En otras palabras, representa las palabras en función de las frases que las contienen y viceversa.

Así pues, el primer paso de LSA es construir una matriz cuyas columnas sean las frases y cuyas filas sean palabras contenidas en esas frases. El contenido de la matriz puede representar diversos valores que midan el peso de la palabra en la frase ponderado de diferentes maneras que veremos más adelante en la Sección 5.1; en otras palabras, cada columna sería un vector de características de esa frase. Supongamos que hay n frases y m términos diferentes, la matriz que se crea (en adelante "A") será de dimensiones m x n.

A continuación se factoriza dicha matriz empleando un sistema conocido como Descomposición en Valores Singulares o SVD (*Singular Value Decomposition*) que reduce significativamente el número de filas y ayuda a establecer la relación entre las frases y las palabras. Dada la matriz A, esta descomposición consiste en:

$$A = U\Sigma V^T \tag{5.1}$$

Donde U y V^T son dos matrices m x n ortonormales y Σ es una matriz diagonal n x n que contiene los valores singulares de A. Desde un punto de vista semántico, SVD obtiene la estructura semántica latente del documento representado en forma de matriz. La matriz U representa la relación entre palabras mientras que V representa relación entre palabras y frases.

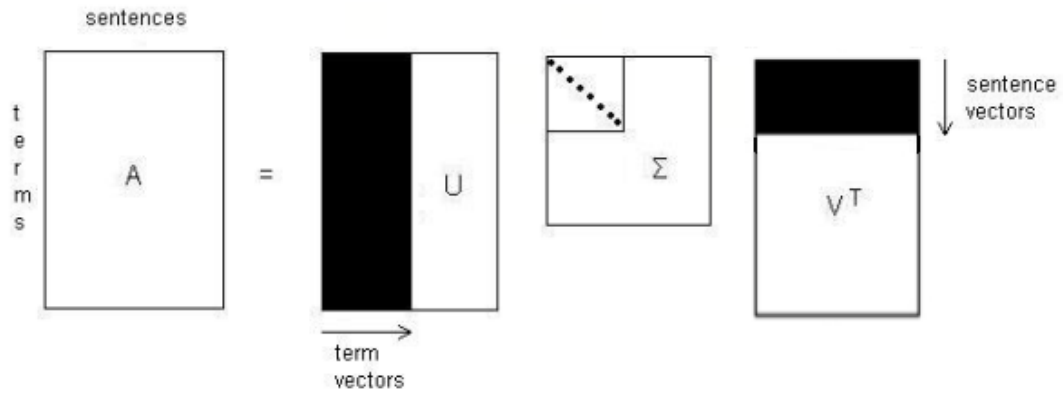


Figura 5.1: Esquema de funcionamiento del SVD [22]

En definitiva, obtenemos la matriz V^T cuyas columnas representan las frases y cuyas filas son los conceptos obtenidos del método SVD. Los conceptos están ordenados por relevancia (de mayor a menor) y el valor de las celdas expresa su relación con las frases en las que aparecen. Un valor mayor significa que el concepto está más relacionado con la frase (tiene más relevancia en

la frase). Una vez hecho esto, el resumen consiste en escoger las frases en las que esos términos más importantes aparecen de manera relevante.

Por otro lado, en nuestro caso, las frases serán tuits y en Twitter no solo es relevante el contenido per se. También hay que tener en cuenta otros factores como retuits, o el número de seguidores del usuario que enunció el tuit. De este modo será necesario encontrar una manera de ponderar esa información en la matriz resultante para tener en cuenta estas características especiales.

5.1. Creación de la matriz de pesos

Como se ha comentado anteriormente, el primer paso es crear la matriz de frases (en nuestro caso tuits) por palabras. En este paso hay que hacer hincapié en cuán importante resulta escoger adecuadamente la manera de ponderar las palabras en la matriz. En nuestro caso se han realizado pruebas con 9 maneras diferentes, las cuales se muestran y ejemplifican a continuación.

Imaginemos que nuestro corpus está formado por tres frases (extraídas de un poema de Juan Ramón Jiménez):

```
¡Rosa no presentida que quitara
a la rosa la rosa que le diera
a la rosa le diera la rosa!
```

Los sistemas probados y las diferentes matrices resultantes con cada uno de ellos serían:

- **Contadores**: El número de veces que la palabra aparece en la frase.

diera	la	le	no	presentida	que	quitara	rosa
0	0	0	1	1	1	1	1
1	2	1	0	0	1	0	2
1	2	1	0	0	0	0	2

- **Contadores binarios**: Valor binario si la palabra aparece o no en la frase.

diera	la	le	no	presentida	que	quitara	rosa
0	0	0	1	1	1	1	1
1	1	1	0	0	1	0	1
1	1	1	0	0	0	0	1

- **Contadores n-gramas**: Número de veces que un grupo de *n* palabras aparece en la frase. En este caso cada palabra de la matriz será en realidad un grupo de hasta *n* palabras.

diera	diera la	la	la rosa	le	le diera	que	que le	rosa	rosa la	
1	0	2	2	1	1	1	1	2	1	
1	1	2	2	1	1	0	0	2	0	...

(Se han empleado solo las dos segundas frases para reducir el tamaño de la tabla generada)

- **TF con normalización 'l1'**: Se calcula la frecuencia del término en la frase y se normalizan los contadores para sumar 1 (por ejemplo, suponiendo un vector $\overrightarrow{vec}$ de longitud n: $\sum_{i=0}^{n} \overrightarrow{vec}[i] = 1$).

diera	la	le	no	presentida	que	quitara	rosa
0	0	0	0.2	0.2	0.2	0.2	0.2
0.1429	0.2857	0.1429	0	0	0.1429	0	0.2857
0.1667	0.3333	0.1667	0	0	0	0	0.3333

- **TF con normalización 'l2'**: Igual que el anterior, pero aquí normalizando los contadores para que el módulo del vector sea 1 (por ejemplo, suponiendo un vector $\overrightarrow{vec}$ de longitud n: $\sqrt{\sum_{i=0}^{n} \overrightarrow{vec}[i]^2} = 1$).

diera	la	le	no	presentida	que	quitara	rosa
0	0	0	0.4472	0.4472	0.4472	0.4472	0.4472
0.3015	0.603	0.3015	0	0	0.3015	0	0.603
0.3162	0.6325	0.3162	0	0	0	0	0.6325

- **TF-IDF**: Se calcula el factor TF-IDF (*Term Frequency - Inverse Document Frequency*) de cada término.

diera	la	le	no	presentida	que	quitara	rosa
0	0	0	2.0986	2.0986	1.4055	2.0986	1.0
1.4055	2.8109	1.4055	0	0	1.4055	0	2.0
1.4055	2.8109	1.4055	0	0	0	0	2.0

- **TF-IDF con IDF suavizado**: Igual que el anterior pero aplicando suavizado al IDF [1].

diera	la	le	no	presentida	que	quitara	rosa
0	0	0	1.6931	1.6931	1.2877	1.6931	1.0
1.2877	2.5754	1.2877	0	0	1.2877	0	2.0
1.2877	2.5754	1.2877	0	0	0	0	2.0

- **TF-IDF con IDF suavizado y normalizado 'l1'**: Igual que el anterior normalizando los contadores para sumar 1.

diera	la	le	no	presentida	que	quitara	rosa
0	0	0	0.2298	0.2298	0.1748	0.2298	0.1357
0.1526	0.3052	0.1526	0	0	0.1526	0	0.237
0.1801	0.3602	0.1801	0	0	0	0	0.2797

- **TF-IDF con IDF suavizado y normalizado 'l2'**: Igual que el anterior pero normalizando los contadores para que el módulo del vector sea 1.

[1] El suavizado IDF se realiza sumando 1 a la frecuencia de los documentos, simulando un documento extra que contuviera todos los términos exactamente una vez.

diera	**la**	**le**	**no**	**presentida**	**que**	**quitara**	**rosa**
0	0	0	0.5046	0.5046	0.3838	0.5046	0.298
0.3259	0.6519	0.3259	0	0	0.3259	0	0.5063
0.3448	0.6896	0.3448	0	0	0	0	0.5355

5.2. Métodos de selección de los tuits resumen

Una vez creada la matriz de pesos y aplicado SVD, el siguiente paso es extraer de las matrices resultantes los tuits mas representativos del conjunto para formar el resumen. En [14] se pueden encontrar diversos métodos para escoger las frases más adecuadas. En este proyecto hemos probado dos de ellos obteniendo diferentes resultados (ver Sección 5.3):

- El más simple de ellos [8] consiste en escoger el concepto más importante y extraer la frase en la que ese concepto tenga más relevancia. Se sigue este proceso sistemáticamente hasta haber extraído un número determinado de frases que formarán el resumen.

  El principal problema de este método es que a la hora de seleccionar una frase para el resumen solo se tiene en cuenta el hecho de que contenga un concepto relevante, cuando podría ser el único que contenga, mientras que se podría dar la existencia de algunas frases que incluyeran un número mayor de conceptos medianamente importantes que se está descartando por no incluir ninguno de los conceptos más relevantes.

- Por otro lado, también en [14] se proponen dos nuevos métodos, uno de los cuales obtuvo resultados significativamente mejores que los otros propuestos. Este sistema denominado *cross* consiste en realizar una serie de postprocesados a la matriz V^T orientados a eliminar ruido.

  En primer lugar se calcula el valor medio de relevancia de cada concepto en todas las frases tal y como se muestra en la Tabla 5.1:

	sent0	sent1	sent2	sent3	average
con0	0.557	0.691	**0.241**	**0.110**	0.399
con1	**0.345**	0.674	0.742	**0.212**	0.493
con2	0.732	**0.232**	0.435	**0.157**	0.389
con3	**0.628**	**0.436**	0.783	0.865	0.678
con4	0.557	0.691	**0.241**	0.710	0.549

Tabla 5.1: Método Cross (I): Calcular el valor medio de cada concepto

A continuación todos los valores de cada fila inferiores a su media se ponen a cero (en negrita en la Tabla 5.1). Esto se hace porque hay frases que sin ser las más representativas para un concepto pueden estar relacionadas de alguna forma con él. Para eliminar ese ruido se pone a cero todo valor inferior a la media del concepto.

El siguiente paso es calcular la longitud de cada frase. Esto se realiza multiplicando los valores singulares contenidos en Σ por los valores de V y a continuación sumando los

valores de todos los términos para cada frase. No obstante, durante la implementación del sistema se realizó una pequeña modificación a este proceso que, como veremos en la Sección 5.3, ofrece mejores resultados para el caso de trabajar con tuits. Así pues, siguiendo con el ejemplo anterior, en la Tabla 5.2 se muestra la forma en la que se calcularía la longitud:

	sent0	sent1	sent2	sent3	average
con0	0.557	0.691	**0**	**0**	0.399
con1	**0**	0.674	0.742	**0**	0.493
con2	0.732	**0**	0.435	**0**	0.389
con3	**0**	**0**	0.783	0.865	0.678
con4	0.557	0.691	**0**	0.710	0.549
length	1.846	2.056	1.960	1.575	

Tabla 5.2: Método Cross (II): Calcular la longitud de cada frase

Por último, la selección de frases consiste simplemente en escoger aquellas cuya longitud sea mayor y formar con ellas el resumen. De este modo, no se tiene en cuenta solo el que una frase contenga el concepto más relevante, si no que entran en juego todos ellos.

5.3. Pruebas realizadas

Como hemos podido ver, existen diferentes métodos tanto para crear la matriz de pesos como para escoger las frases del resumen, así pues, se han realizado una serie de pruebas para encontrar el sistema que mejores resultados ofrezca para nuestro problema concreto.

Además del trabajo con matrices, Twitter tiene una serie de características que añaden variables a la ecuación. Por ejemplo, a la hora de realizar el resumen se pueden emplear retuits (frases repetidas) o no. Otro hecho a tener en cuenta es que determinadas entidades podrían generar ruido al ser tratados como conceptos si no se eliminan previamente, por ejemplo las URLs contenidas en los tuits pueden tomarse como palabras, cuando en realidad no añaden información semántica al tuit.

Por otro lado, para realizar las pruebas se ha trabajado con diferentes conjuntos de tuits, de los que se realizó un resumen a mano. Este resumen modelo se comparó de diferentes maneras con los resúmenes generados automáticamente. Entre ellas, se empleó el conjunto de métricas ROUGE [10] y se evaluaron ciertas características de los tuits extraídos que veremos a continuación.

Para las diferentes pruebas llevadas a cabo, se ha preparado un sistema automatizado que una vez configurados los parámetros de la prueba concreta, genere un documento *.html* que contenga tablas con los diferentes resultados, ya sea valores de precisión, *recall* y puntuación del ROUGE o características diversas de los tuits del resumen generado automáticamente. Además se implementó una herramienta que genere tablas resumen comparativas conjugando todos esos

documentos *html* generados de modo que la selección de un sistema u otro como sistema definitivo de resumen fuese más sencilla.

5.3.1. Primera prueba: Aproximación ingenua

Esta prueba ofreció unos resultados pésimos, de modo que solo se comentará brevemente. Es una primera aproximación ingenua, en la cual se descargaron **515** tuits con el texto "rajoy". Estos tuits se puntuaron a mano entre 0 y 10, siendo 0 un tuit nada relevante y 10 muy relevante. El criterio seguido para establecer la relevancia fue que el contenido del tuit estuviese relacionado con alguno de los temas de los que más se hablaba en el conjunto completo de tuits, por ejemplo:

- Explicar la posición de España en Grecia
- El envío de 4000 soldados a las fronteras con Rusia
- Pablo Iglesias defiende la patria
- Los cambios de Rajoy que parecen nulos
- Otros de inferior relevancia...

Además, se optó por eliminar tanto menciones como URLs (por no aportar contenido semántico al tuit), mientras que sin embargo se dejaron los *hashtags* dado que pueden añadir información al tuit. Por último, no se eliminaron los retuits, dejando todas las repeticiones. El resumen modelo se realizó extrayendo todos aquellos que tuvieran una puntuación de 6 o más y eliminando manualmente los repetidos. El resumen generado automáticamente es de un tamaño fijo de 10 tuits.

En la Tabla 5.3 se muestran dos valores de los resúmenes obtenidos: en primer lugar, la puntuación total es la suma de la puntuación de todos los tuits del resumen (como máximo 86), mientras que el número de tuits en el resumen modelo (tuits exitosos o de acierto) es la cantidad de tuits cuya puntuación es superior a 6. Cómo puede observarse en la tabla, la puntuación total no es demasiado alta (como veremos en pruebas posteriores) y el número de tuits obtenidos en el resumen generado de manera automática cuya puntuación sea superior a seis es como mucho 2 (de un máximo de 10).

Sistema empleado	Puntuación	Nº de tuits en el resumen modelo
Contadores	18	0
Contadores binarios	34	2
Contadores por N-gramas	30	1
TF-IDF: con idf suav. y norm 'l2'	18	0
TF normalizado 'l1'	36	1
TF normalizado 'l2'	28	0
TF-IDF	21	0
TF-IDF: con idf suavizado	33	1
TF-IDF: con idf suav. y norm.'l1'	33	0

Tabla 5.3: Resultados de puntuación (Prueba 1)

Por otro lado, los resultados obtenidos mediante la métrica ROUGE se muestran en la Tabla 5.4. Solo se han incluido los resultados de ROUGE-4, ROUGE-L, ROUGE-W y ROUGE-

SU* por claridad y espacio, aunque los valores obtenidos en estas son muy similares a los que se obtuvieron en el resto de métricas. Para más información acerca de la métrica ver [10].

SISTEMA EMPLEADO	ROUGE-4	ROUGE-L	ROUGE-W	ROUGE-SU*
Contadores	0.34464	0.56128	0.18577	0.28617
Contadores binarios	**0.37733**	0.58678	0.19362	0.31261
Contadores ngramas	0.34311	0.55173	0.18214	0.28195
TF-IDF: idf suav. y norm 'l2'	0.31217	0.54386	0.17773	0.28019
TF normalizado 'l1'	0.36861	0.58771	**0.19461**	**0.32192**
TF normalizado 'l2'	0.35165	0.56831	0.18191	0.29035
TF-IDF	0.32728	0.54250	0.17064	0.27019
TF-IDF con idf suavizado	0.35357	0.57904	0.18682	0.30938
TF-IDF: idf suav. y norm. 'l1'	0.37489	**0.59631**	0.19358	0.32178

Tabla 5.4: Resultados de ROUGE (Prueba 1)

Pese a los resultados obtenidos arriba, esta métrica ofrece resultados prometedores con valores de incluso 0.54 en ROUGE-L, no obstante el corpus de prueba es demasiado pequeño y los resultados no pueden considerarse significativos.

5.3.2. Segunda prueba: Extracción de frases

Esta prueba está destinada a escoger uno de los dos sistemas de selección de frases resumen: el sistema directo o el sistema *cross*. El objetivo es escoger el mejor entre ellos y a continuación emplear el ganador para realizarle una serie de modificaciones de modo que tenga en cuenta factores como retuits y seguidores a la hora de realizar los resúmenes.

Así pues, para las siguientes pruebas se descargó un nuevo corpus de 2000 tuits, también con la búsqueda "rajoy". Dado que los resultados obtenidos en la primera prueba fueron tan bajos y el objetivo es ponderar los retuits de otro modo, se decidió eliminar todos los tuits que fuesen retuit, conservando solo los originales. Tras este proceso, el corpus quedó reducido a un total **1010** tuits, los cuales se etiquetaron a mano por relevancia entre 0 y 10. De la misma manera que en la primera prueba, se crearon dos resúmenes modelo extrayendo los tuits cuyas puntuaciones fuesen superiores a 6: el primero sin ninguna entidad, mientras que el segundo conservaba los *hashtags*. Para esta prueba y las siguientes se emplearon ocho modelos diferentes, o en otras palabras, ocho maneras de "limpiar" los tuits de elementos que puedan distorsionar los resultados del resumen automático:

- **02**: Tuits sin entidades, tanto en el resumen modelo como en los automáticos.
- **04**: Tuits sin entidades salvo *hashtags*.
- **06**: Igual que #02 pero eliminando *stopwords* [2] previamente.
- **08**: Igual que #04 pero eliminando *stopwords* previamente.

[2]Se ha empleado la lista de *stopwords* en español que proporciona SnowBall: http://snowball.tartarus.org/algorithms/spanish/stop.txt

Los modelos #01, #03, #05 y #07 son equivalentes respectivamente a los cuatro de arriba pero sin realizar limpieza (borrado de entidades) alguna de los tuits.

En primer lugar, se realizó un intento con el modelo #02 con el cual se comprobó que el sistema *cross* no proporcionaba unos resultados tan buenos como los esperados, de hecho, ambos sistemas ofrecen resultados bastante similares:

	D	C	D	C	D	C	D	C	D	C
SIST. EMPLEADO	ROUGE-L		ROUGE-W		Puntuación[3]		Nº tuits >6		Media RT	
Contadores	**0.429**	0.417	**0.133**	**0.129**	42	46	**3**	1	**93.0**	28.8
Cont. binarios	0.377	0.385	0.120	0.117	32	33	0	1	12.4	1.5
Cont. ngramas	0.388	0.390	0.123	0.126	34	34	0	0	6.3	6.3
TF-IDF: idf s. y 'l2'	0.411	**0.435**	0.117	0.127	39	**53**	0	0	0.8	**40**
TF norm. 'l1'	0.426	0.127	0.119	0.121	**47**	41	1	**2**	6.3	0.4
TF norm. 'l2'	0.418	0.416	0.130	0.121	41	42	1	2	15.2	7.1
TF-IDF	0.408	0.401	0.121	0.120	38	37	0	0	3.1	0.6
TF-IDF: idf suav.	0.415	0.393	0.119	0.121	34	45	2	0	10.0	0.2
TF-IDF: idf s. y 'l1'	0.390	0.433	0.118	0.109	43	43	0	2	8.7	0.2

Tabla 5.5: Comparativa básica *cross* vs directo

Como se puede ver, el método *cross* no ofrece demasiada mejora respecto al sistema directo, y además es mucho más complejo y costoso. En un intento de solucionar esto, se ha cambiado un paso de este segundo sistema. En concreto, se ha eliminado la parte del proceso donde se multiplica la matriz Σ por V, con lo que se han conseguido mejorar notablemente los resultados.

	C2	C1	C2	C1	C2	C1	C2	C1	C2	C1
SIST. EMPLEADO	ROUGE-L		ROUGE-W		Puntuación		Nº tuits >6		Media RT	
Contadores	0.414	0.414	0.119	**0.129**	50	46	0	1	0.8	28.8
Cont. binarios	0.416	0.385	0.122	0.117	36	33	2	1	7.6	1.5
Cont. ngramas	0.389	0.390	0.110	0.126	34	34	0	0	16.1	6.3
TF-IDF: idf s. y 'l2'	0.444	**0.435**	0.117	0.127	51	**53**	2	0	**52.2**	**40**
TF norm. 'l1'	0.457	0.127	0.120	0.121	51	41	**5**	**2**	1.0	0.4
TF norm. 'l2'	**0.460**	0.416	**0.171**	0.121	52	42	4	2	0.7	7.1
TF-IDF	0.392	0.401	0.113	0.120	41	37	0	0	17.8	0.6
TF-IDF: idf suav.	0.391	0.393	0.117	0.121	41	45	0	0	2.3	0.2
TF-IDF: idf s. y 'l1'	0.457	0.433	0.124	0.109	**54**	43	4	2	0.6	0.2

Tabla 5.6: Comparativa básica *cross* original (1) vs *cross* modificado (2)

Se puede observar en la Tabla 5.6 que el número de tuits cuya puntuación está por encima de 6 es mucho mayor con el sistema *cross* modificado (en concreto, la media sube de 0.8 a 1.8), de la misma manera que la puntuación total de los resúmenes es también mayor en general. Así mismo puede verse que los valores de ROUGE-L son en general mayores con el *cross* modificado.

Por otro lado, puede verse una cierta predilección a obtener un número más bajo de retuits cuánto mayor es el número de aciertos (tuits en el resumen modelo), esta característica del sistema de resumen la exploraremos más adelante.

La mejora del método *cross* modificado sobre el original se debe a que en el producto de Σ por V se pierden algunos de los conceptos. A continuación se detalla este hecho:

Como se vió al principio de este capítulo, las dimensiones de V son n x m (donde n es la cantidad de frases (o tuits) y m es la cantidad de conceptos) y Σ es una matriz diagonal n x n. La longitud de cada frase i se calcula como $longitud[i] = \sum_{j=0}^{j=t} V_{ij} * \Sigma_{jj}$. El producto matricial que se realiza para el cálculo de la longitud multiplica cada uno de los conceptos por uno de los valores singulares contenidos en la matriz Σ. La variable t es un parámetro del método *cross* que indica cuantos conceptos deben tenerse en cuenta. Sin embargo, se puede decir sin pérdida de generalidad que la cantidad de conceptos será siempre superior o igual a la cantidad de tuits, y por lo tanto que $n \leq m$. De este modo, no se consideran todos los conceptos al calcular la longitud pues el valor de t será como máximo n.

Así pues, el método *cross* modificado, por eliminar este producto sí considera todos los conceptos a la hora de seleccionar las frases para el resumen. Esto deriva en que al trabajar con tuits (en lugar de artículos científicos con los que se probó el método *cross* original [14]), se obtengan mejores resultados. Debido a la reducida cantidad de conceptos que puede haber en un máximo de 1000 tuits (en comparación con un corpus de artículos científicos), considerar la mayor cantidad de conceptos obtiene mejores resúmenes.

En conclusión, el método escogido para realizar los resúmenes es el sistema *cross* modificado por ofrecer mejores resultados tanto en la métrica ROUGE como en la puntuación total y número de aciertos.

5.3.3. Tuits populares

Una vez escogido el sistema de extracción de frases resumen de la matriz, el siguiente paso es encontrar una forma de ponderar los factores de Twitter en la ecuación. En primer lugar se observó que no existía relación alguna entre la puntuación asignada a mano y el número de retuits o favoritos que pudiera tener el tuit. Cómo puede verse en la Figura 5.2, ordenada por número de retuits para 200 tuits del corpus de la sección anterior, existe cierta relación entre el número de RT y favoritos, no obstante la puntuación asignada manualmente a los tuits no guarda concordancia alguna.

El siguiente paso es, entonces, encontrar qué hace a un tuit relevante. En una entrevista [12] con Abdur Chowdhury, Director Científico de Twitter, se explica como cada tuit tiene una expectativa de interacciones, esto es, según el historial del usuario se espera un número concreto de retuits o respuestas. Según cuanto supere el tuit esas expectativas será más o menos popular. Así pues se decidió estudiar las características de los tuits populares descargando un corpus de 45 tuits con la búsqueda “rajoy”.

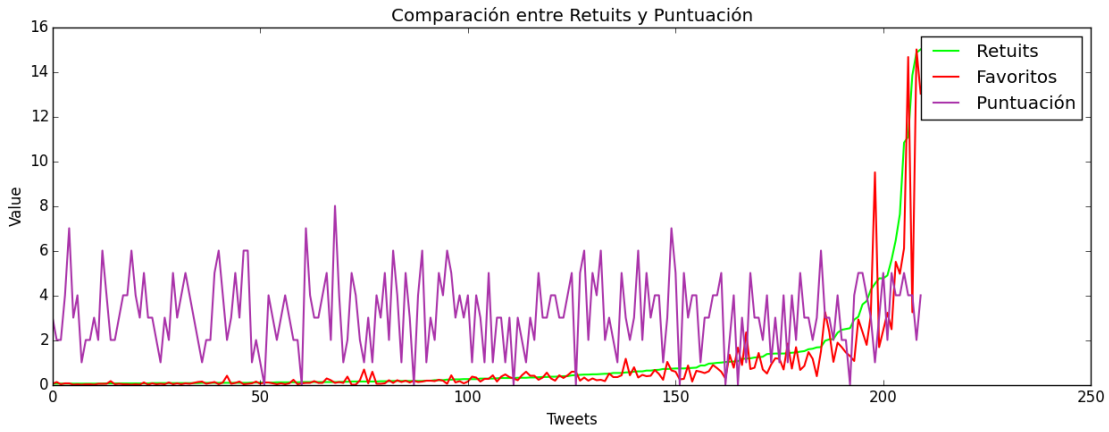


Figura 5.2: Comparativa entre los valores de puntuación y retuits

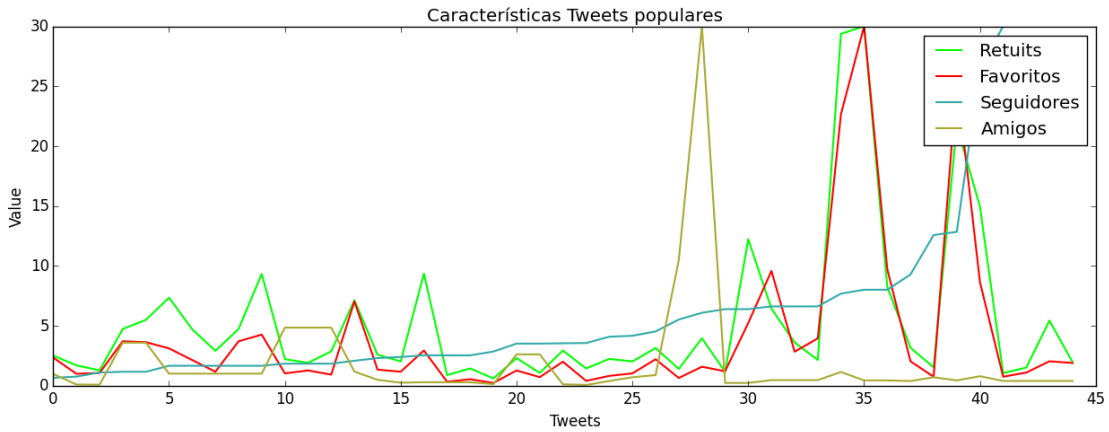


Figura 5.3: Características de los tuits populares

Como puede observarse en la Figura 5.3, ordenada por número de seguidores y con todos los valores reescalados situando el máximo en 30 para más claridad, los tuits populares tienen siempre un número elevado de retuits y favoritos, estando estos segundos mayoritariamente por debajo de los primeros. Además se comprobó que la relación entre seguidores y amigos es de 100 a 7 en general, es decir, los tuits populares son enunciados por usuarios cuyo número de amigos es de aproximadamente un 7 % del número de seguidores.

En resumen, para medir la popularidad de un tuit teniendo en cuenta estas características se han empleado las siguientes heurísticas:

- **Popularidad del contenido**: El número de retuits y favoritos debe ser elevado en relación con el número de seguidores, es decir, tienen más validez 30 retuits para un usuario con 5 seguidores que para otro con 1200.
- **Popularidad del usuario**: Se requiere un elevado número de seguidores en relación con el número de amigos. Además, para una misma proporción entre seguidores y amigos, se

valora más aquel que tenga más seguidores, por ejemplo: tiene mayor relevancia un tuit publicado por un usuario con 100 seguidores y 10 amigos, que otro con 50 seguidores y 5 amigos pese a que la proporción seguidores/amigos sea la misma.

Así pues, para valorar los retuits (RT) y favoritos (Fav) dando más relevancia a los primeros y valorándolos más cuantos menos seguidores (F) tenga, se ha planteado la siguiente ecuación:

$$contentPop = \frac{RT * 1'5 + Fav}{\log F} \tag{5.2}$$

Se divide entre el logaritmo del número de seguidores (en lugar de hacerlo sin el logaritmo) para no reducir demasiado el valor de $contentPop$. Por otro lado, para considerar la relación entre seguidores y amigos (f), en la ecuación Ecuación 5.3 se plantea un cociente que se hará mayor cuantos más seguidores y menos amigos tenga. Además se da un mayor relevancia al número de seguidores elevando el valor al cuadrado.

$$userPop = \frac{F^2}{f} \tag{5.3}$$

A lo largo del desarrollo del proyecto se plantearon otras maneras de ponderar las diversas características de un tuit, mas fueron desechadas por los resultados obtenidos.

Una vez establecidos los dos factores en forma de ecuación, se probaron tres maneras de conjugarlos en uno solo para formar tres métricas diferentes:

$$metric1 = \log\left(\log\frac{RT * 1'5 + Fav}{\log F} * \log\frac{F^2}{f}\right) \tag{5.4}$$

,

$$metric2 = \sqrt{\left(\log\frac{RT * 1'5 + Fav}{\log F} * \log\frac{F^2}{f}\right)} \tag{5.5}$$

y

$$metric3 = \left(\log\frac{RT * 1'5 + Fav}{\log F} * \log\frac{F^2}{f}\right) \tag{5.6}$$

Lo que se consigue empleando el logaritmo o la raíz cuadrada es un suavizado de los valores de la métrica obtenidos. Se observó que si se utilizaban los dos componentes en bruto causaban una valoración demasiado elevada de los factores de Twitter, reduciendo la cantidad de tuits con una puntuación alta que se obtienen en el resumen. Por ello se optó por probar suavizados de diferente orden que redujesen la manera en que la métrica afecta al sistema de resumen.

Como podemos observar en la Figura 5.4 (cuyos valores están todos reescalados poniendo el máximo en 50 por claridad), las tres métricas tienen un comportamiento similar, y la única diferencia aparente es la pendiente con la que aumentan. Aparentemente todas ellas crecen cuando lo hacen las características de los tuits observadas, mas se hacen necesario una serie de pruebas que veremos en la Subsección 5.3.4 para determinar cual de ellas es más eficaz a la hora de determinar los tuits que deberán formar parte del resumen.

También se puede observar que las tres métricas guardan una relación mas estrecha con retuits y favoritos que con seguidores o amigos, no obstante como veremos en la Subsección 5.3.4, también está ligado con esas características. Por otra parte, no solo tenemos que buscar la métrica que obtenga más tuits populares, si no aquella que encuentre un equilibrio entre tuits populares y tuits con una puntuación alta.

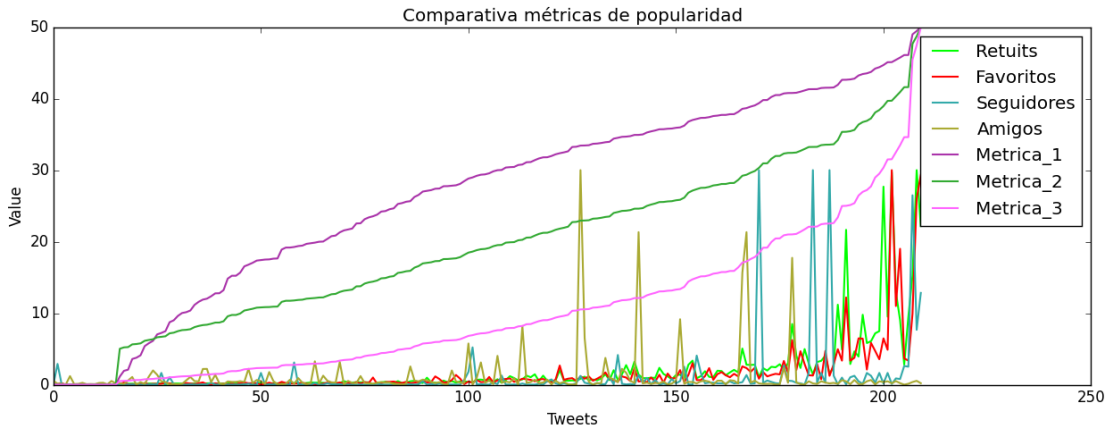


Figura 5.4: Comparativa entre las tres métricas planteadas

5.3.4. Tercera prueba: Popularidad

En este apartado se plantea la manera en la que la métrica escogida debe afectar a la matriz V de modo que a la hora de escoger los tuits para el resumen se valoren más los tuits populares.

Como vimos en la Sección 5.2, el método *cross* genera un vector con las longitudes de cada tuit, de modo que el tuit con una longitud mayor sea el primero en ser elegido para el resumen. Así pues, la métrica deberá modificar de alguna manera esos valores de modo que se de más relevancia a los tuits mas populares. A la longitud modificada por la métrica la denominaremos la puntuación (*score*) de un tuit.

Se han probado seis formas diferentes de ponderar la métrica, en todos ellos, el valor resultante se multiplica directamente por la longitud del tuit. Supongamos que $\overrightarrow{leng}$ es el vector de longitudes, $\overrightarrow{met}$ es un vector del mismo tamaño que contiene el valor de la métrica para cada tuit y $\overrightarrow{score}$ es el vector que contendrá las puntuaciones de los tuits. De este modo, las seis formas planteadas de calcular la puntuación son:

- **A**: El valor obtenido de la métrica de la Ecuación 5.4 se multiplica sin más al vector de longitudes, es decir, ambos vectores se multiplican elemento a elemento: $\overrightarrow{score} = \overrightarrow{leng}$ x $\overrightarrow{met}$.
- **B**: de manera similar a **A**, salvo que en este caso el vector $\overrightarrow{met}$ se divide por el valor máximo que tome la métrica en el vector y se suma 1 a cada uno de ellos, quedando valores entre 1 y 2.

  De este modo al multiplicar a $\overrightarrow{leng}$, afecta como un porcentaje que incrementa más o menos su valor (pero nunca disminuyendo). En resumen, para cada elemento i de los vectores, la puntuación se se calcula como: $\overrightarrow{score}[i] = \overrightarrow{vec}[i] * \left(1 + \frac{\overrightarrow{met}[i]}{\text{máx}(\overrightarrow{met})}\right)$.
- **C** y **D**: Igual que **A** y **B** respectivamente, pero empleando la métrica en la Ecuación 5.5.
- **E** y **F**: Igual que **A** y **B** respectivamente, pero empleando la métrica en la Ecuación 5.6.

Se ha probado el sistema de resúmenes con estas seis maneras de calcular la puntuación, obteniendo los resultados que se muestran en la Tabla 5.7. Para el conteo de tuits acertados

se ha modificado el criterio de >6 a ≥5 debido a que como hemos visto en la Figura 5.2, la relación entre las características del tuit con la puntuación es prácticamente nula, y el hecho de incrementar el número de factores a tener en cuenta en el momento de generar el resumen, hace que sea necesario relajar las restricciones empleadas para evaluar el sistema; esto es debido a que el número de tuits populares que además tengan una puntuación alta es menor, pues en general la cantidad de tuits populares es menor.

Los resultados de la Tabla 5.7 se han obtenido empleando el modelo #06 de los descritos en la Subsección 5.3.2, pero los resultados son muy similares a los obtenidos con otros modelos.

SIST.	Nº tuits ≥5						Media RT					
	A	B	C	D	E	F	A	B	C	D	E	F
Contadores	3	2	3	3	2	3	563	496	563	563	587	563
Cont. binarios	2	2	2	1	2	2	565	525	565	552	575	565
Cont. n-gramas	1	2	1	2	1	1	535	330	511	487	560	535
TF-IDF: idf s. y norm 'l2'	3	3	2	3	2	2	509	381	567	487	583	567
TF norm. 'l1'	3	4	3	3	2	3	440	244	535	423	575	440
TF norm. 'l2'	3	4	3	3	3	3	564	365	564	535	564	564
TF-IDF	2	2	3	2	2	2	539	405	564	539	684	544
TF-IDF: idf s.	3	2	3	2	2	3	564	437	564	555	684	564
TF-IDF: idf s. y norm 'l1'	4	5	4	5	3	4	348	220	351	220	543	341

Tabla 5.7: Comparativa de los diferentes sistemas para aplicar la métrica

Cómo podemos observar en las figuras 5.5 y 5.6, E obtiene muy buenos resultados en el número medio de retuits, pero es a costa de unos resultados bajos en número de tuits ≥5. Con B sucede justo lo contrario: obtiene buenos resultados en número de tuits ≥5 y por contra, pésimos en número medio de retuits. De los cuatro restantes, todos obtienen resultados muy similares en número de tuits, mientras que en número de retuits se desmarca D como inferior. De este modo, y dado que en general los resultados son muy similares (también en ROUGE), se ha escogido C como métrica ganadora por ser una de las más equilibradas y obtener buenos resultados en número medio de retuits.

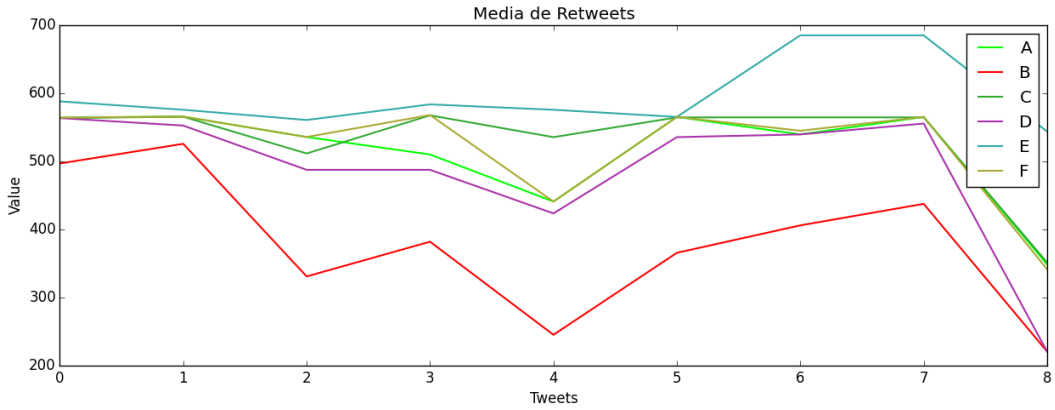


Figura 5.5: Valor medio de retuits para los diferentes sistemas de aplicación de la métrica

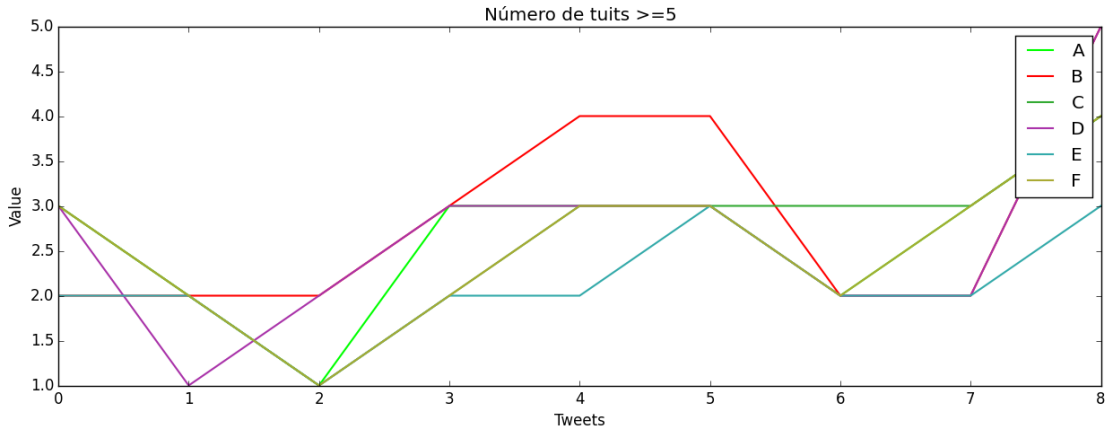


Figura 5.6: Número de tuits $\geq$5 para los diferentes sistemas de aplicación de la métrica

5.3.5. Cuarta prueba: Matriz de pesos

Esta última prueba pretende encontrar cual de los nueve sistemas vistos en la Sección 5.1 produce mejores resultados y si eliminando *hashtags* y/o *stopwords* se obtienen mejores resultados una vez establecidas todas las demás variables del sistema.

Para este apartado, primero se ha buscado cual de los ocho sistemas planteados en el apartado 5.3.2 obtenía mejor resumen. Para ello, se ha resumido el corpus de tuits con cada una de las combinaciones posibles de a) llenar la matriz de pesos y b) eliminar entidades o stopwords. Una vez probadas las 72 combinaciones, se ha calculado la media para cada una de las ocho combinaciones *hashtags/stopwords* (pues tratar de analizar directamente los resultados obtenidos de las 72 combinaciones resultaba innecesariamente complejo) y se han obtenido los resultados que se ilustran en la Figura 5.7. Los valores fueron reescalados situando el máximo en 50, pues mientras por ejemplo ROUGE ofrece valores entre 0 y 1, el número de seguidores puede llegar a miles o millones.

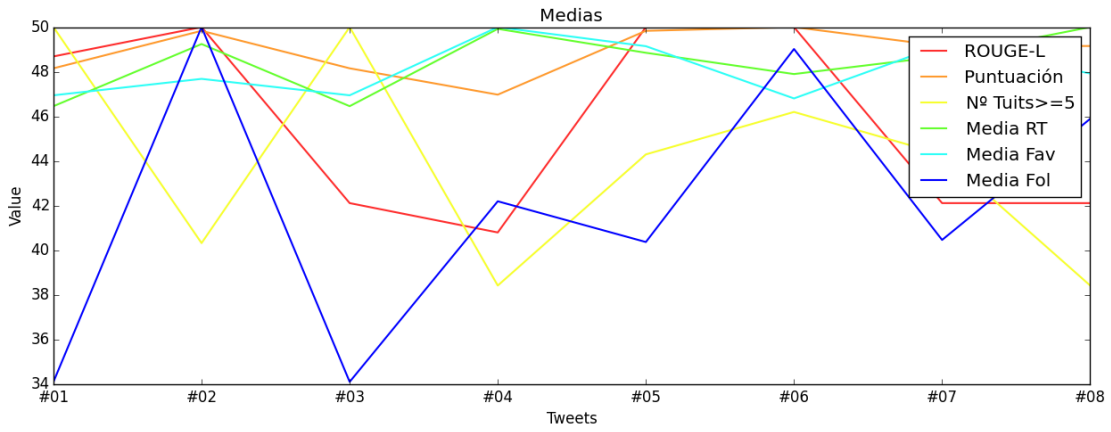


Figura 5.7: Valores medios de cada sistema

Como puede verse, algunos sistemas tienen valores muy altos en algunos campos pero muy bajos en otros, sin embargo el sistema #06 muestra unos resultados buenos en todas las características medidas, pese a no ser los más altos, esa consistencia es muy deseable y se le ha escogido como sistema ganador.

El siguiente paso es comparar los 9 sistemas de creación de matrices de pesos para el sistema #06. En la Tabla 5.8 podemos ver los resultados obtenidos:

SIST. EMPLEADO	ROUGE-L	ROUGE-W	Puntuación	Nº tuits ≥5	Media RT
Contadores	0.36256	0.11122	32	3	563.7
Contadores binarios	0.36673	0.11259	28	2	565.2
Contadores ngramas	**0.39899**	**0.12819**	30	1	511.7
TF-IDF: idf s. y 'l2'	0.35854	0.11072	29	2	**567.5**
TF norm. 'l1'	0.38183	0.12136	35	3	535.6
TF norm. 'l2'	0.38280	0.12174	**36**	3	564.7
TF-IDF	0.38280	0.12174	36	3	564.7
TF-IDF: idf s.	0.38280	0.12174	36	3	564.7
TF-IDF: idf s. y 'l1'	0.38605	0.12460	35	**4**	351.3

Tabla 5.8: Comparativa entre las 9 formas de llenar la matriz de pesos para el sistema #06

Del mismo modo que hemos venido observando a lo largo de este capítulo, un mayor número de tuits con puntuación alta implica una media de retuits más baja, tal y como podemos ver con el sistema TF-IDF con IDF suavizado y normalización 'l1', el cual tiene el número más alto de tuits acertados y el más bajo de retuits; por el contrario, el sistema equivalente pero con normalización 'l2' obtiene el número más alto de retuits y un valor muy bajo de tuits acertados. Así pues hemos buscado sistemas más equilibrados, por ejemplo, TF con normalización 'l2', el cual obtiene buenos resultados en todos los campos, y aunque no obtiene los mayores valores, sus valores son cercanos al máximo en todas las características medidas.

5.4. Conclusiones del sistema de Resumen

El sistema de resumen para la aplicación queda finalmente definido por los siguientes parámetros:

- El valor para llenar la matriz de pesos es TF-IDF con normalización 'l2'.
- El sistema para extraer los tuits resumen de la matriz es el *cross* modificado.
- Los tuits se pasan al sistema de resumen sin entidad alguna y eliminando *stopwords*.
- La métrica para ponderar los tuits populares es la definida en la Ecuación 5.5 sin normalización.

El resumen generado por el sistema una vez configurado con estos parámetros es el siguiente:

```
- Mucho hablar de cambios y, al final, solo cambiaron a Wert. Que Rajoy
pague el IVA Cultural por haber hecho teatro.
- Rajoy dice no a la independencia de Cataluña. Dijo lo mismo del IVA, el
rescate, etc... Técnicamente ha declarado la independencia.
- Rajoy no es tan tonto si lo comparas con la gente que lo volverá a
votar.
- Aznar critica a Rajoy en nombre de la vieja guardia del PP. Dice que
conduce al PP a la derrota
- Hace tanto calor que Rajoy es gilipollas
- Grecia puede alterar el discurso económico de Rajoy. Aznar, el discurso
interno. Maíllo, el discurso anticorrupción. La culpa, del martilleo
- La que se va a liar cuando Rajoy recrimine a Obama por pedir una quita
de la deuda griega. #YoVoyConGecia
- El New York Times tilda de "franquista" la #LeyMordaza del Gobierno de
Rajoy #PP #SinMordazas
- Si Rajoy les devuelve la extra de 2012 a los funcionarios, ¿por qué no
devuelve a los contribuyentes los impuestos extra pagados desde 2012?
- Presidente Rajoy, Colombia tiene el mayor cartel de cocaína y terrorismo
del mundo que aparece como actor político.
```

Como se puede observar trata de prácticamente todos los temas mayoritarios que se mencionaron en la Subsección 5.3.1.

Tras haber trabajado con la aplicación en el periodo de desarrollo de la misma, se ha concluido que los resúmenes realizados para búsquedas ambiguas obtenían resultados poco representativos. En otras palabras, si la búsqueda obtiene un conjunto de tuits cuyos contenidos no están al menos en parte relacionados entre si los resúmenes formados no parecen representar realmente las opiniones de un tema concreto, pues tratan de temáticas potencialmente muy dispares.

Capítulo 6
Conclusiones

En este apartado se realiza una recapitulación del trabajo realizado en el proyecto y se analiza la profundidad con la que se han alcanzado los objetivos planteados en la Sección 1.2.

En primer lugar se ha realizado una aplicación atractiva al usuario que le permite realizar búsquedas sobre Twitter. Los tuits obtenidos son clasificados y resumidos, de modo que el usuario puede apreciar los distintos resultados de polaridad que obtiene la búsqueda. Esto permite valorar qué opinión tiene el grueso de usuarios de Twitter acerca del ente buscado. Además, dado que estos resultados se almacenan, se ofrece la posibilidad de observar la evolución de estas opiniones a lo largo del tiempo, permitiendo asociar hechos de actualidad con reacciones al respecto por parte de los usuarios de Twitter.

Django ha demostrado ser una gran herramienta de trabajo para desarrollar aplicaciones web de manera efectiva y sencilla. Tiene un tiempo de aprendizaje muy reducido y adaptarse a la estructura MVC que plantea no ha sido complicado. La mayor dificultad ha sido transmitir información entre vistas y entre plantillas y vistas, no obstante estos obstáculos han podido salvarse sin que el aumento de complejidad fuese significativo.

Por otro lado, se ha incluido un sistema de resumen que permite extraer el contenido más representativo de los tuits. Seleccionar los tuits más importantes del conjunto descargado posibilita un análisis más completo de las opiniones que se plasman en los tuits descargados con la búsqueda. Esto permite al usuario hacerse una idea más acertada de lo que se está publicando en Twitter acerca del tema en cuestión. El sistema de resumen se ha creado probando las diferentes configuraciones que permitía modificar el algoritmo empleado. Desde la creación de la matriz de pesos (que finalmente contiene los valores TF-IDF normalizando los contadores para que el módulo del vector sea 1), hasta la cantidad de elementos a eliminar del tuit antes de realizar el resumen (en nuestro caso tanto *stopwords* como todas las entidades). Por otro lado, encontramos que la modificación del sistema *cross* (el cual había demostrado ofrecer muy buenos resultados a la hora de crear resúmenes por extracción mediante el sistema LSA) en la que se realiza un paso menos ofrecía mejores resultados a la hora de trabajar con tuits como documentos.

En particular, el sistema de resumen ha sido especialmente complicada de desarrollar por la ausencia de corpus de tuits puntuados que permitiesen evaluar de manera más completa los resúmenes obtenidos. En general, puntuar o etiquetar tuits manualmente resulta una tarea larga que consume una gran cantidad del tiempo del proyecto. Además la elección de una métrica que

valore la popularidad de un tuit es un tanto subjetiva y se basa en una serie de heurísticas que se ha definido y evaluado en este proyecto y que habría que contrastar más detalladamente en un conjunto de tuits más extenso.

6.1. Trabajo futuro

Este proyecto da pie a múltiples modificaciones futuras. Una de las principales sería adaptar la aplicación aquí desarrollada para permitir un mayor grado de paralelismo y permitir así su funcionamiento *on-line* en un servidor y atender a múltiples usuarios que pudiesen estar accediendo simultáneamente a la aplicación. En primer lugar sería necesario realizar las modificaciones pertinentes y que son requisito de Django para permitir a la aplicación funcionar en un servidor y en segundo lugar, cambiar la base de datos con la que se trabaja por defecto (SQLite) por otra que soporte mayor nivel de paralelismo.

Otra de las modificaciones principales que se podrían hacer es emplear un corpus mayor de tuits puntuados a mano para medir de manera más ajustada la precisión del sistema de resumen, así como la adecuación de la métrica para la valoración de tuits como más o menos populares.

También sería interesante considerar algunos de los siguientes añadidos:

- Añadir un clasificador de temáticas que permita afinar más las búsquedas a lo que realmente el usuario desea encontrar. Esto permitiría ofrecer unos resultados mucho más concretos dado que una misma búsqueda puede contener resultados en campos muy distintos no estando necesariamente unos relacionados con los otros. Un ejemplo claro es el partido político "Podemos", que puede ofrecer resultados en Twitter de cualquier tuit que contenga esta conjugación del verbo "poder". Un clasificador de temáticas ayudaría a filtrar este contenido no deseado.

- En el caso de que la aplicación se alojase en un servidor y dado que el cuello de botella es la descarga de los tuits, una característica interesante sería que la aplicación descargase semanalmente una cantidad grande de tuits para las búsquedas más populares contenidas en la base de datos. Esto permitiría disponer de resultados históricos mucho más significativos dado que se estaría trabajando con un número mucho más grande de tuits a cada vez, cosa que al trabajar en directo con el usuario no puede hacerse (por el gran tiempo que consume la descarga de los tuits).

Por último, la aplicación permite recoger retroalimentación de manera sencilla, por lo que es una herramienta muy útil a la hora de reunir un corpus mayor de tuits etiquetados a mano que permitan entrenar de nuevo el clasificador de polaridad, dando lugar a un clasificador con una precisión potencialmente mayor.

Bibliografía

[1] «TweetMotif: Exploratory Search and Topic Summarization for Twitter», 2010.
http://brenocon.com/oconnor_krieger_ahn.icwsm2010.tweetmotif.pdf

[2] «TASS 2013», 2013. [Online; accessed 13-July-2015].
http://www.daedalus.es/TASS2013/about.php

[3] 10GEN: «MongoDB», 2015. [Online; accessed 23-August-2015].
https://www.mongodb.org/

[4] ——: «MongoDB - 3.0 Documentation», 2015. [Online; accessed 19-August-2015].
http://docs.mongodb.org/v3.0/

[5] DJANGO SOFTWARE FOUNDATION: «Django 1.8.2 release notes», 2015. [Online; accessed 27-May-2015].
https://docs.djangoproject.com/en/1.8/releases/1.8.2/

[6] GIMÉNEZ, MAYTE; PLA, FERRAN y HURTADO, LLUÍS-F.: «ELiRF: A Support Vector Machine Approach for Sentiment Analysis Tasks in Twitter at SemEval-2015», 2015, p. 574–581.
http://users.dsic.upv.es/~fpla/ARTICLES13/fpla_semeval2015.pdf

[7] GO, ALEC; BHAYANI, RICHA y HUANG, LEI: «Twitter Sentiment Classification using Distant Supervision». Processing, 2009, pp. 1–6.
http://www.stanford.edu/~alecmgo/papers/TwitterDistantSupervision09.pdf

[8] GONG, YIHONG y LIU, XIN: «Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis». En: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '01, pp. 19–25. ACM, New York, NY, USA. ISBN 1-58113-331-6, 2001. doi: 10.1145/383952.383955.
http://www.cs.bham.ac.uk/~pxt/IDA/text_summary.pdf

[9] HURTADO, LLUÍS-F. y PLA, FERRAN: «ELiRF-UPV en TASS 2014: Análisis de Sentimientos, Detección de Tópicos y Análisis de Sentimientos de Aspectos en Twitter», 2014.
http://users.dsic.upv.es/~fpla/ARTICLES13/elirf_upv_tass2014_v2.pdf

[10] LIN, CHIN-YEW: «ROUGE: A Package for Automatic Evaluation of summaries». En: Proc. ACL workshop on Text Summarization Branches Out, p. 10, 2004.
http://research.microsoft.com/~cyl/download/papers/WAS2004.pdf

[11] MATPLOTLIB: «MatPlotLib 1.4.3 Documentation», 2015. [Online; accessed 11-August-2015].
http://matplotlib.org/1.4.3/contents.html

[12] MCGEE, MATT: «Twitter: How Our New 'Top Tweets' Works». http://searchengineland.com/twitter-how-our-new-top-tweets-works-39115, 2010. [Online; accessed 15-August-2015].

[13] NEVOKA, ANI y MCKEOWN, KATHLEEN: «Automatic Summarization». Foundations and Trends in Information Retrieval, 2011, pp. 103–233. doi: 10.1561/1500000015.
http://www.cis.upenn.edu/~nenkova/1500000015-Nenkova.pdf

[14] OZSOY, MAKBULE GULCIN; CICEKLI, ILYAS y ALPASLAN, FERDA NUR: «Text Summarization of Turkish Texts Using Latent Semantic Analysis». En: Proceedings of the 23rd International Conference on Computational Linguistics, COLING '10, pp. 869–876. Association for Computational Linguistics, Stroudsburg, PA, USA, 2010.
http://dl.acm.org/citation.cfm?id=1873781.1873879

[15] PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M. y DUCHESNAY, E.: «Scikit-learn: Machine Learning in Python». Journal of Machine Learning Research, 2011, **12**, pp. 2825–2830.
http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

[16] PLA, FERRAN y HURTADO, LLUÍS-F.: «ELiRF-UPV en TASS-2013: Análisis de Sentimientos en Twitter», 2013.
http://users.dsic.upv.es/~fpla/ARTICLES13/elirf_TASS2013.pdf

[17] ——: «Sentiment Analysis in Twitter for Spanish», 2014, pp. 208–213.
http://users.dsic.upv.es/~fpla/ARTICLES13/sentiment_analisisNLDB_v19_final_SHORTPAPER.pdf

[18] PROJECT, COMMUNITY: «NumPy 1.9.2», 2015. [Online; accessed 29-May-2015].
http://www.numpy.org/

[19] PYMONGO: «PyMongo Documentation - MongoDB», 2015. [Online; accessed 23-August-2015].
https://api.mongodb.org/python/current/

[20] PYTHON SOFTWARE FOUNDATION: «Python 3.4.3», 2015. [Online; accessed 27-May-2015].
https://www.python.org/downloads/release/python-343/

[21] RUSSELL, MATTHEW A.: Mining the Social Web (2nd Edition). O'Reilly Media, 2013. ISBN 978-1449367619.

[22] STEINBERGER, JOSEF y JEŽEK, KAREL: «Using Latent Semantic Analysis in Text Summarization and Summary Evaluation», 2004.
http://www.kiv.zcu.cz/~jstein/publikace/isim2004.pdf

[23] TWEEPY: «Tweepy Cursors», 2014. [Online; accessed 16-August-2015].
http://docs.tweepy.org/en/latest/cursor_tutorial.html

[24] ——: «Tweepy 3.3», 2015. [Online; accessed 28-May-2015].
https://github.com/tweepy/tweepy

[25] TWITTER, INC.: «Twitter: About the company», 2015. [Online; accessed 25-August-2015].
https://about.twitter.com/en/company

[26] VARIOUS AUTHORS: «Scikit-Learn 0.16.1», 2015. [Online; accessed 01-June-2015].
http://scikit-learn.org/0.16/

[27] ——: «Scikit-Learn github repository», 2015. [Online; accessed 01-June-2015].
https://github.com/scikit-learn/scikit-learn

[28] W3C: «HTML 4.01 Specification», 1999. [Online; accessed 16-August-2015].
http://www.w3.org/TR/1999/REC-html401-19991224/

[29] W3 SCHOOLS: «JavaScript and HTML DOM Reference», 2015. [Online; accessed 13-August-2015].
http://www.w3schools.com/jsref/

[30] WIKIPEDIA: «Automatic summarization — Wikipedia, The Free Encyclopedia», 2015. [Online; accessed 11-August-2015].
https://en.wikipedia.org/w/index.php?title=Automatic_summarization&oldid=676307597

[31] ——: «Cascading Style Sheets — Wikipedia, The Free Encyclopedia», 2015. [Online; accessed 13-August-2015].
https://en.wikipedia.org/w/index.php?title=Cascading_Style_Sheets&oldid=675622747

[32] ——: «Django (web framework) — Wikipedia, The Free Encyclopedia», 2015. [Online; accessed 15-August-2015].
https://en.wikipedia.org/w/index.php?title=Django_(web_framework)&oldid=673849585

[33] ——: «HTML — Wikipedia, The Free Encyclopedia», 2015. [Online; accessed 13-August-2015].
https://en.wikipedia.org/w/index.php?title=HTML&oldid=677481496

[34] ——: «JavaScript — Wikipedia, The Free Encyclopedia», 2015. [Online; accessed 13-August-2015].
https://en.wikipedia.org/w/index.php?title=JavaScript&oldid=677496488

[35] ——: «Latent semantic analysis — Wikipedia, The Free Encyclopedia», 2015. [Online; accessed 12-August-2015].
https://en.wikipedia.org/w/index.php?title=Latent_semantic_analysis&oldid=670683490

[36] ——: «Singular value decomposition — Wikipedia, The Free Encyclopedia», 2015. [Online; accessed 12-August-2015].
https://en.wikipedia.org/w/index.php?title=Singular_value_decomposition&oldid=677053296

Lista de Abreviaturas

CSS	Cascading Style Sheets
HTML	HyperText Markup Language
IDF	Inverse Document Frequency
LSA	Latent Semantic Analysis
MVC	Model-View-Controller
NLG	Natural Language Generation
NLP	Natural Language Processing
SEPLN	Sociedad Española para el Procesamiento del Lenguaje Natural
SVD	Singular Value Decomposition
SVM	Support Vector Machines
TASS	Taller de Análisis de Sentimientos en la SEPLN
TF	Term Frequency
W3	World Wide Web

Glosario

Amigo Conjunto de usuario a los que está suscrito un usuario concreto.

Análisis de Sentimientos Es una tarea de NLP que busca extraer contenido subjetivo (opiniones y polaridad) de uno o más textos.

Favorito Un tuit marcado como favorito indica que al usuario le ha gustado.

Framework Son el conjunto de herramientas y técnicas que sirven de soporte para construir una aplicación.

Hashtag Son etiquetas que se emplean para indicar o incluso complementar el contenido de un tuit.

Mencion Citación a un usuario contenida en un tuit.

Retuit Es el reenvio de un tuit por parte de un usuario. Cuando un tuit es retuiteado se añade al conjunto de publicaciones del usuario.

Seguidor Conjunto de los usuario suscritos a las publicaciones de un usuario concreto.

Tuit Es la entidad básica de publicación en Twitter. Consta de un máximo de 140 caracteres.

Twitter Es una red social de *microblogging* que permite a sus usuarios publicar opiniones e ideas en forma de textos breves..

Apéndice A

Apéndice A

Este apéndice contiene la documentación (en inglés) del código auxiliar que se desarrolló para servir de soporte a la aplicación Django. Cada uno de los apartados incluye las funciones de un archivo concreto.

A.1. `tweetclass/database_connector.py`

This file contains all the necessary functions for accessing the database. Everytime some function needs to retreive or store information, it is through this file's functions.
date: 2015
author: Javier Selva Castello

`obtain_query`(`query_text_search`)

- **desc**: Returns a Query object with the requested query text. If it doesn't exist, it is created.
- **params**:
  - `query_text_search` - a string containing the text of the query to be found.
- **return**: Query object with the text in "`query_text_search`".

`store_polarity`(`requested_query`,`raw_tweets`,`clas_tweets=[]`)

- **desc**: Creates a `Query_data` with the polarity results for a given Query
- **params**:
  - `requested_query` - the Query on which to add the `Query_data` object
  - `raw_tweets` - a list of tweets (dictionaries) the polarity of which has to be stored. It has to contain at least the "polarity" and "`retweet_count`" fields
  - `clas_tweets` - a list of strings containing the polarity of each tweet contained in the list "`raw_tweets`". If not provided, the polarities will be extracted from `raw_tweets`
- **return**: `Query_data` object created with the percentage value of each polarity and its date

store_tweets(requested_query,raw_tweets)

- **desc**: Stores all the downloaded tweets for a given query
- **params**:
  - requested_query - the Query on which to add the tweets in the database
  - raw_tweets - a list of tweets (dictionaries) to be stored.
- **return**: nothing

get_last_query_data(q_text)

- **desc**: Retreives the last Query_data id from a given Query
- **params**:
  - q_text - string containing the query text
- **return**: integer containing the last Query_data id

retrieve_query(requested_query_data_id)

- **desc**: Given a Query_data id, returns all the information available in the database for that query
- **params**:
  - requested_query_data_id - integer containing the id for which the information has to be retrieved
- **return**:
  - current_query - the actual Query_data which id is the requested by parameter
  - query - the Query object referenced by the Query_data
  - all_results - a list of all the Query_datas for the Query "query"
  - sum_tweets - the three sumaries (generic, positive and negative) as a lists of Summary_tweets

store_summary(requested_query, tweets, tweet_tag)

- **desc**: Given a summary and its type (generic, positive or negative), it's stored on the database
- **params**:
  - requested_query - Query object related to which the summary has to be stored
  - tweets - a list of tweets (dictionaries) containing the summary tweets to be stored

  - `tweet_tag` - string containgin the type of summary being stored. The different values expected are:
    - "*ALL*" - for a generic summary
    - "*POS*" - for a summary of the positive tweets
    - "*NEG*" - for a summary of the negative tweets

- **return**: nothing

`retrieve_query_list`()

- **desc**: Generates a list of all the Querys ever made
- **params**: none needed
- **return**: a list of Query objects sorted by the amount of times it has been made

`store_feedback`(`tweets,polarity`)

- **desc**: Stores feedback tweets in the database
- **params**:
  - `tweets` - a list of tweets (as `Summary_tweets`) to be stored
  - `polarity` - the feedbacked polarity as a list of strings
- **return**: nothing

A.2. tweetclass/graph_data_generator.py

The main idea for this file is transforming the raw Query_datas into a list of polarities so the plotter (generate_graph) only has to draw it.
author: Javier Selva Castello
date: 2015

`generate_data`(`name,query_list`)

- **desc**: This function creates a list of lists of polarity values (floats), and then calls the draw function in `code/generate_graph.py` to plot the graphics
- **params**:
  - `name` - string containing the query text
  - `query_list` - a list of `Query_datas` with all the polarity results for every time the query "name" was made
- **return**: nothing, it will create the graphics calling the plotter

A.3. tweetclass/code/get_polarity.py

This file contains the necessary code to call the clasificator given a list of tweets (as plain text) **author**: Javier Selva Castello
date: 2015

get_polarity(tweets)

- **desc**: This function obtains the polarity out of a list of tweets.
- **params**:
  - tweets - a list of strings containing the text of the tweets to be classified
- **return**: a list of strings containing the polarity of each tweet given

A.4. tweetclass/code/get_tweets.py

This file contains all the necessary functions to download and procces the tweets from Twitter. To connect, the tweepy library is used.
author: Javier Selva Castello
date: 2015

get_tweet_api()

- **desc**: This function sets the authentification process and returns an api object with the OAuth configured.
- **params**: none needed, the keys are global variables
- **return**: api object on which make all the Twitter connections needed

transform_links_regex(tweets)

- **desc**: This function transforms all the entities within a tweet to make them "html" links.
  - The mentions will link to the mentioned user's profile
  - The hashtags will linkt the hashtag's page
  - The urls will link wherever they are suppoused to link
- **params**:
  - tweets - a list of tweets (Summary_tweet objects) which entities are to be transformed The tweets should at least contain the key "text"
- **return**: nothing, the list of tweets is modified through the reference passed by param

extract_tweet_info(tweet)

- **desc**: Given a tweet as an object it returns a dictionary with the tweet content extracted
- **params**:
  - `tweet` - tweet object as it comes when doing a request to Twitter
- **return**: dictionary containing all the tweet necessary info

`clear_retweets`(`raw_tweets`)

- **desc**: Given a Cursor, the function iterates extracting every tweet info and returns a list of tweets. It also removes any tweet that is a retweet of other tweet and places the original one in the list instead.
- **params**:
  - `raw_tweets` - Cursor containing the tweets of a given query
- **return**: list of tweets (dictionaries) containing all the tweet processed data necessary for the app

`get_tweets`(`query,types="mixed",MAX_TWEETS = 1000`)

- **desc**: Given a query, it searches twitter for that query and returns a list of tweets (dictionaries) containing all the tweet info
- **params**:
  - `query` - string containing the query to be made
  - `types` - string indicating the type of tweets to look for Expected inputs:
    - "mixed": popular and real time tweets
    - "popular": only popular tweets
    - "recent": only the most recent results
  - `MAX_TWEETS` - integer containing the amount of tweets to look for
- **return**: list of tweets (dictionaries) with each field being one relevant feature of the tweet such as "`text`", "`id`" or "`retweet_count`"

A.5. `tweetclass/code/generate_graph.py`

This file contains all the necessary functions to generate the application graphics.
author: Javier Selva Castello
date: 2015

`draw_things`(`things,colors,labels,text,name=""`)

- **desc**: Generic function to draw a graphic given a set of data

- **params**:
  - `things` - a list of lists of integers containing the data to be plotted All the lists inside "`things`" should have the same length.
  - `colors` - a list of strings representing the color that each thing in things will have on the plot. The colors may be in hexadecimal (`#AF33E2`) or follow the matplotlib predefined colors (`http://matplotlib.org/api/colors_api.html`) The length of this list should be greater or equal to the lenght of "`things`".
  - `labels` - a list of strings containing the label for each thing to be plotted. The length of this list should be greater or equal to the lenght of "`things`".
  - `text` - a string containing the title of the plotting.
  - `name` - a string containing the name of the file where the plot will be stored. If the default value is recieved, the file name will be the same as the "`text`"
- **return**: nothing, a "*.png*" image with the resulting graphic will be saved to disk

`general_graph`(`polarity,name`)

- **desc**: Draws the generic graph (a line for each polarity evolution) for a query
- **params**:
  - `polarity` - a list of lists of integers (length = 6) representing the evolution of each polarity through the different times the query was made
  - `name` - the name of the query being plotted
- **return**: nothing, a "*.png*" image with the resulting graphic will be saved to disk

`radial_summary`(`values,name`)

- **desc**: Draws a pie graph representing the percentage of each polarity in the total amount of tweets in the generic summary
- **params**:
  - `polarity` - a list of integers representing the percentage of each polarity
  - `name` - the name of the query being plotted
- **return**: nothing, a "*.png*" image with the resulting graphic will be saved to disk

`summary_graph`(`polarity,name`)

- **desc**: Draws the summary graph (one line representing the summed up polarity) for a query
- **params**:

  - `polarity` - a list of integers representing the evolution of the summed up polarity of the query through the different times the query was made
  - `name` - the name of the query being plotted
- **return**: nothing, a "*.png*" image with the resulting graphic will be saved to disk

A.6. tweetclass/code/tweet_summary.py

This file contains all the necessary functions to produce a tweet's summary.
author: Javier Selva Castello
date: 2015

add_field(field_name,tweets,field_content)

- **desc**: Adds a (key,value) tuple to each dictionary contained in a list of dics
- **params**:
  - `field_name` - a string containing the name of the field to be added (the key)
  - `tweets` - a list of diccionaries on which to add the field
  - `field_content` - a list containing the values to be added to each dictionary
- **return**: nothing, as it modifies the list referenced in the param "`tweets`"

clean_tweets(tweets,to_clean=["urls","hashtags","mentions"])

- **desc**: Ereases different twitter entities (urls, hashtags and/or mentions) from a list of tweets
- **params**:
  - `tweets` - a list of dictionaries, each of which represeinting a tweet and containing at least the key "`text`"
  - `to_clean` - a list with the one to three entities to remove from the tweets. by default it is all of them
- **return**: list of strings - a list of tweets (just the text) whitout the entities requested

prepare_metric(original_tweets)

- **desc**: Calculates the popularity metric value for each tweet
- **params**:
  - `original_tweets` - a list of tweets (in the form of dictionaries) containing at least the fields "`followers`", "`friends`", "`retweet_count`" and "`favorite_count`"
- **returns**: nothing, as it modifies the list referenced in the param "`original_tweets`"

remove_stopwords(tweet)

- **desc**: Removes the stopwords in a tweet
- **params**:
  - tweet - a string containing the tweet text from where to remove the stopwords
- **returns**: string - the string without stopwords

summarize(tweets,system=5,wrange=4,MAX_RES_TWEETS = 10,use_retweets=True, remove_stop=True,use_cross=True,cleaning=1)

- **desc**: Creates a tweet summary out of a list of tweets. All the default params where proved to output better summaries.
- **params**:
  - tweets - a list of dictionaries containing all the tweet info necessary to generate the summary (at least the "text") If use_retweets is enabled, the tweets should also contain the fields "followers", "friends", "retweet_count" and "favorite_count"
  - system - integer value between 0 and 8. The system indicates the values that the matrix will contain to represent the importance of a concept in a sentence:
    - 0 - Counters
    - 1 - Binary Counters
    - 2 - N-gram Counter
    - 3 - TF-IDF: with smoothen IDF and 'l2' normalization
    - 4 - TF with 'l1' normalization (Normalized counters to sum up to 1)
    - 5 - TF with 'l1' normalization (Normalized counters in order to the vector's module to be 1)
    - 6 - TF-IDF
    - 7 - TF-IDF with smoothen IDF
    - 8 - TF-IDF: with smoothen IDF and 'l2' normalization
  - wrange - integer value greater or equal to 1. It is only used if system = 2, is the N in N-grams
  - MAX_RES_TWEETS - integer value containing the amount of tweets that will be in the summary result
  - use_retweets - boolean indicating whether the popularity values should be used or not for the summary generation
  - remove_stop - boolean indicating whether or not the stopwords should be removed from the tweets
  - use_cross - boolean indicating whether or not the cross method for tweet selection should be used

  - `cleaning` - integer containging the amount of "cleaning" (removing of entities) that should be made:
    - 0 - No cleaning at all
    - 1 - All the entities will be removed
    - 2 - All the entities but hashtags will be removed

- **returns**: a list of dicctionaries (tweets) of length `MAX_RES_TWEETS` containing the summary tweets If SVD doesn't converge, it returns an empty list