Proyecto Fin de Carrera

Ingeniería de Telecomunicación

Analítica de datos en Twitter

Dani Mir Montserrat

Director: Jose López Vicario

Departament de Telecomunicació i d'Enginyeria de Sistemes

Escola Tècnica Superior d'Enginyeria (ETSE)
Universitat Autònoma de Barcelona (UAB)

Febrero 2015

El abajo firmante, *Jose López Vicario*, Profesor de l'Escola Tècnica Superior d'Enginyeria (ETSE) de la Universitat Autònoma de Barcelona (UAB),

CERTIFICA:

Que el proyecto presentado en esta memoria de Proyecto Fin de Carrera ha estado realizado bajo su dirección por el alumno *Dani Mir Montserrat*.

Y, para que conste a todos los efectos, firma el presente certificado.

Bellaterra, *27 de Enero del 2015*.

Signatura: *Jose López Vicario*

"A todos los familiares y amigos que me han apoyado,
en especial a María Cortés Gimeno."
Bellaterra, Febrero de 2015

Índice

Índice de figuras

Índice de tablas

1 Introducción

1.1 Motivación

Las redes sociales (rrss o *social media* en inglés) se pueden definir como sitios en Internet que permiten la creación de comunidades de personas entre las que se establece un intercambio dinámico de información y contenidos generados por los propios usuarios.

En los últimos 5 años han ido apareciendo diferentes redes sociales. La figura siguiente permite observar algunas de ellas por año de aparición, donde quizás se podrían destacar teniendo en cuenta el número de usuarios registrados: Facebook, Twitter, Youtube, Google+ y Linkedin.

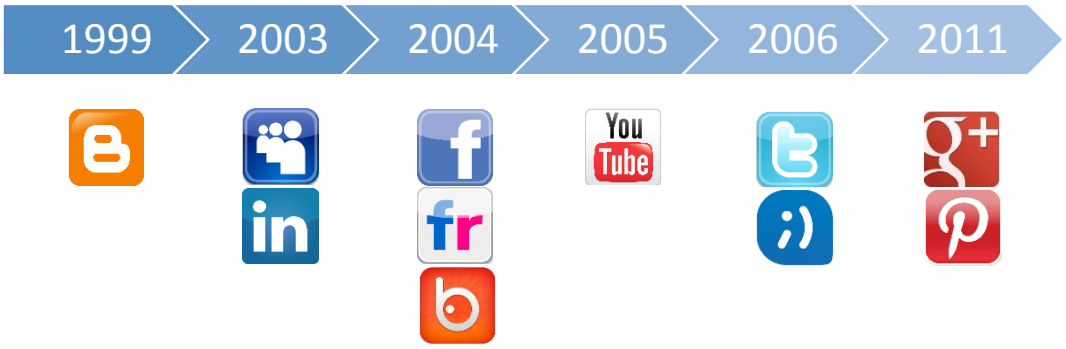


Figura 1.1: Línea de tiempo del social media.

Desde la aparición de las rsss, estas se han convertido en un canal idóneo de comunicación directa para las empresas, una forma de interactuar y conversar con el consumidor final, y así conseguir la fidelización de estos. Con el paso del tiempo,

el crecimiento y expansión que estas han experimentado y debido a la gran competitividad requerida por los mercados, la fidelización de los clientes ya no se consigue solamente con el uso de las rrss como medio de comunicación sino que además se hace imperativo el realizarlo de una forma eficiente.

Así pues, debido a la necesaria información de interés con la que conocer mejor a los usuarios, el presente proyecto pretende realizar *analytics* mediante técnicas *Machine Learning*, procesado del lenguaje, procesado de señal y demás campos de la ingeniería de telecomunicaciones para así ofrecer datos de valor a la hora de llevar a cabo un plan de *marketing* en rrss.

1.2 Antecedentes

Sobre los distintos *social media* que han ido surgiendo, a nivel mundial cabe destacar Facebook y Twitter como las dos principales redes sociales. Existen multitud de diferencias entre ambas y distintas formas de compararlas, aunque quizás, la gran diferencia radica en el tipo de información que estas comparten.

En este sentido, Facebook es una red social basada en las relaciones que se establecen entre los usuarios, es decir, se basa en el entramado social que rodea a cada individuo. Mientras que Twitter, al ser algo más parecido a un medio de comunicación donde la mayoría de la información es pública, se basa en la transmisión de contenidos que a los propios usuarios interesan, generando así nuevas tendencias a escala global.

Es por esto último, junto con la diferencia en el tema de la privacidad de datos entre ambas redes sociales, que en el presente proyecto se decide trabajar con la información que ofrece **Twitter**, aunque el potencial de las dos plataformas es obvio.

Junto a la evolución experimentada por las redes sociales, han surgido multitud de herramientas destinadas a la analítica, y en el caso de Twitter esto no es una excepción. En la breve historia de este han aparecido una gran variedad de herramientas enfocadas a la influencia en la red, otras a la gestión de la cuenta y otras tantas a la gestión de los seguidores.

Algunas de las herramientas más destacadas son Twitonomy[1], Twitalyzer[2], Twitter counter[3], SocialBro[4], Kloud[5] y PeerIndex[6], aunque quizás, la de mayor veracidad sería Twitter Analytics[7], puesto que es la herramienta oficial de Twitter.

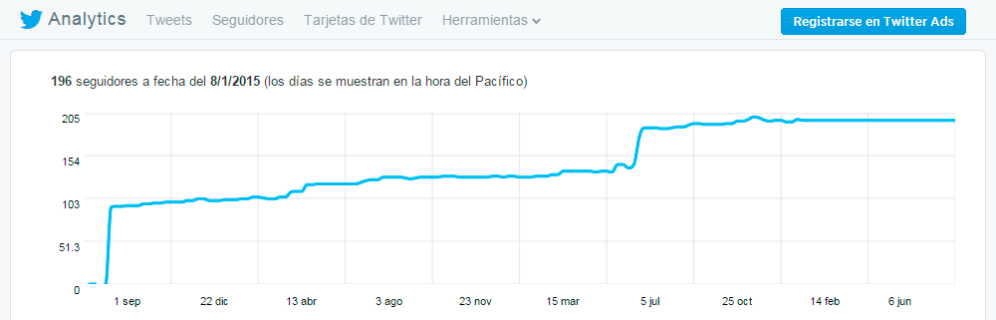


Figura 1.2: Evolución de los seguidores en Twitter Analytics.

Entre la información que estas ofrecen, se pueden encontrar analíticas relacionadas con la evolución del número de seguidores, la repercusión de los tuits en la cuenta, los *hashtags* más utilizados, la geolocalización de los usuarios, los seguidores *celebrity* y demás estadísticas sobre los tuits. [8].

Aunque mucha de la información que proponen es completa, esta tiene algunos problemas o limitaciones:

[1] URL: http://www.twitonomy.com/
[2] URL: http://twitalyzer.com/
[3] URL: http://twittercounter.com/
[4] URL: http://es.socialbro.com/
[5] URL: https://klout.com
[6] URL: http://www.peerindex.com/
[7] URL: https://analytics.twitter.com

- Información bajo pago.
- Información reducida en el tiempo.
- Necesidad de ser propietario de la cuenta a analizar.
- Dudosa veracidad en las métricas (Ej.: análisis de sentimiento en tuits).
- Falta de análisis comparativos entre la competencia.

Así pues, el presente trabajo pretende aportar un estudio alternativo y complementario a los problemas citados y elaborar un análisis más personalizado y comparativo entre diferentes marcas competidoras.

1.3 Objetivos

Al pensar en marcas con las que realizar el análisis comparativo comentado en el punto anterior, se antoja más interesante un análisis entre grandes compañías multinacionales ya que las posibilidades de estas en *marketing* serán mucho mayores:

- Elevada cantidad de consumidores, seguidores y simpatizantes.
- Popularidad a escala internacional.
- Diferentes tipos de productos.
- Gran competencia en los medios.
- Recursos elevados para realizar todo tipo de publicidad.

Hay muchas compañías que puedan cumplir con todo el potencial listado, pero en el presente trabajo se analizarán las grandes multinacionales: adidas y Nike.

Por tanto, el principal objetivo del proyecto, es estudiar la competencia de adidas y Nike en la red social de Twitter. Realizando diferentes analíticas para la extracción de patrones, así como segmentando los seguidores mediante la utilización de técnicas *Machine Learning* y extrayendo tendencias a partir de métodos de *Natural Languaje Processing*, se busca encontrar similitudes y

diferencias entre ambas marcas, todo ello con el propósito de experimentar los campos de la ingeniería citados y abrir un camino de investigación en el mundo del *social media.*

Este estudio se pretende implementar mediante el lenguaje de programación Python, un lenguaje de alto nivel y actualmente en auge. Es por ello, que al primer objetivo se le añade el del aprendizaje de Python.

1.4 Memoria

La presente memoria está dividida en las siguientes partes:

En el capítulo 2 se realiza una introducción a los temas que involucran el desarrollo del trabajo, confeccionando así un análisis sobre las posibles herramientas necesarias.

En el capítulo 3 se describe la propuesta a resolver, metodología e implementaciones elaboradas.

El capítulo 4 muestra los resultados experimentales realizados y su interpretación.

Y finalmente, en el capítulo 5 se exponen las conclusiones finales y el posible trabajo futuro.

2 Estudio de las herramientas

En este capítulo se introducen conceptos básicos para el desarrollo del proyecto y las herramientas utilizadas. Esto es el lenguaje de programación Python, la red social de Twitter y los campos de la inteligencia artificial *Machine learning* y *Natural languaje processing*. Finalmente se comenta el estudio previo realizado para la obtención de los conocimientos citados y las herramientas que en definitiva se usarán.

2.1 Python

La motivación por la que usar el lenguaje de programación Python es debida a la evolución que este ha experimentado en los últimos años, y aunque el lenguaje es de propósito general, este es ampliamente utilizado en ciencias e ingeniería y dispone de una gran cantidad de bibliotecas para multitud de aplicaciones relacionadas con los campos de *Machine Learning* y *Natural Language Processing*, al igual que una librería con la que usar la API de Twitter. Además dispone de una gran cantidad de documentación online.

Así pues, se trata de un lenguaje de programación relativamente moderno, creado en el año 1991 por el Holandés Guido Rossum al conseguir unir las mejores ideas de otros lenguajes de una forma sencilla, intuitiva y fácil de aprender. Como se ha comentado, en los últimos años ha conseguido aumentar su popularidad gracias al auge de las aplicaciones web, hasta tal punto que se ha convertido en uno de los lenguajes de programación oficiales de Google. [3]

Python es un lenguaje **dinámico**, es decir, no necesita declarar las variables previamente a la asignación del valor, si no que cada variable toma directamente el tipo del valor que se le esté asignando. Así pues la mayoría de las asignaciones resuelven el tipo en tiempo de ejecución y no en tiempo de compilación.

```
In [1]: a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
In [2]: type(a)
Out[2]: list
In [3]: b = "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
In [4]: type(b)
Out[4]: str
```

Figura 2.1: Asignación dinámica del tipo list y str.

Por tanto, se puede añadir que también es un lenguaje **interpretado**, lo que quiere decir que el código no llega a traducirse a algo que el sistema operativo entienda, si no que este es interpretado por una máquina virtual que es capaz de entender y ejecutar el código, consiguiendo así tiempos de trabajo mucho más rápidos al no tener que realizar el ciclo de compilación, ejecución y depuración.

Python es un lenguaje de programación **orientado a objetos**, de hecho en Python prácticamente todo son objetos. Los objetos son entidades o instancias de una clase. Las clases están formadas por atributos, las variables que definen el objeto, y por métodos, las funciones que operan con estas variables.

```
In [1]: a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
In [2]: a.append(10)
In [3]: a
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Figura 2.2: Método append en el objeto list.

Python dispone de muchísimos entornos de desarrollo diferentes, algunos libres, otros comerciales y otros enfocados al contexto científico. En este sentido, uno de los entornos más completos y el utilizado en el desarrollo del proyecto es **Spyder**.

Spyder goza de un potente editor con grandes posibilidades de personalización, de un explorador de variables y un inspector de objetos entre otras muchas características. También te permite abrir tantas consolas Python o Ipython como desees. Y además de ser un entorno *open source* puede instalarse en la mayoría de las plataformas (Windows, Mac y Linux).

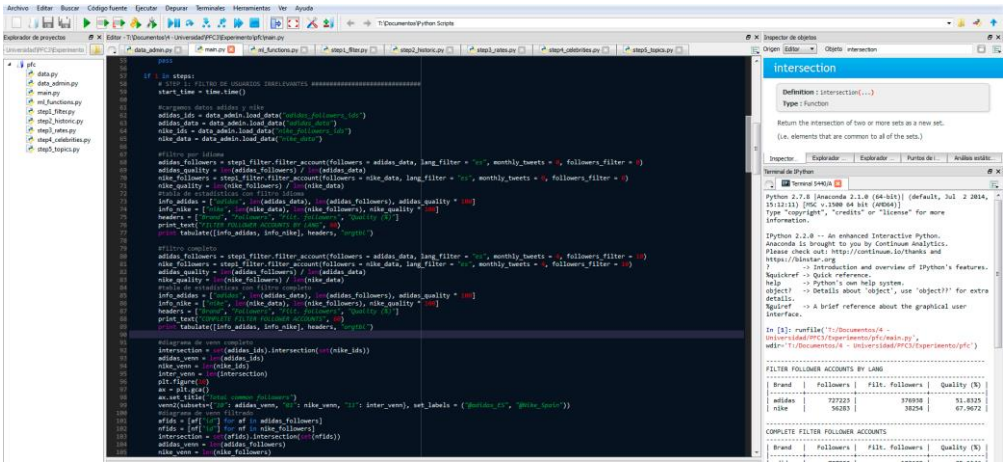


Figura 2.3: Entorno de trabajo Spyder.

La distribución de Spyder utilizada en el proyecto trae incorporada las mejores herramientas científicas de Python, algunas se listan a continuación:

NumPy: biblioteca que permite trabajar con arrays multidimensionales, pilares básicos en el mundo científico, y funciones universales con las que operar a un alto nivel de eficiencia. [14]

Matplotlib: una de las bibliotecas más usadas de Python. Pretende ser similar a las funciones de visualización de MATLAB y por lo tanto nos permite representar los datos y producir gráficas. [6]

ScyPy: Conjunto de paquetes que contienen funciones para computación científica en general sobre diversos temas como la integración, ecuaciones diferenciales, etc. [5]

2.1.1 IPython y Notebook

IPython es un componente de la librería SciPy que añade funcionalidades interactivas a Python. Se puede decir que es la forma interactiva y científica de Python. [4]

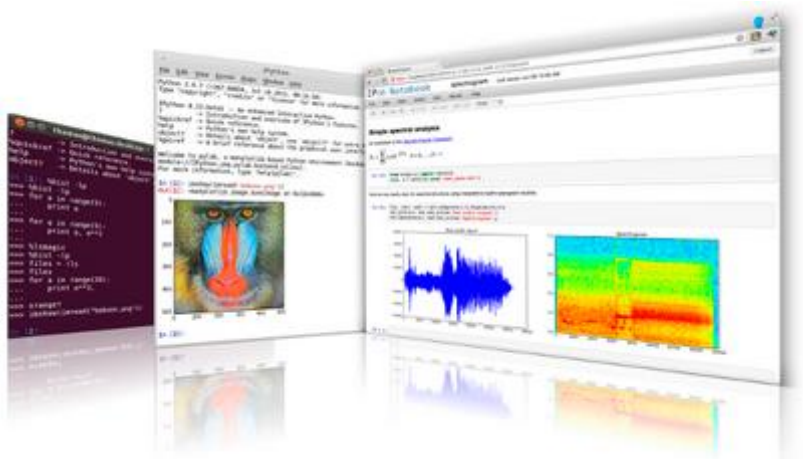


Figura 2.4: Entorno IPython y Notebook.

El Notebook de IPython es una interfaz para Python mejorada, un sistema interactivo web que permite a los usuarios crear documentos que incluyen código, texto, ecuaciones LaTeX, HTML, imágenes y video, tal y como se puede intuir en la imagen[8] anterior. Estos documentos están organizados en celdas que pueden ejecutarse independientemente y que permiten separar de manera lógica cada una de las partes del programa y almacenar sus resultados. Además, los documentos del Notebook de IPython pueden compartirse por correo electrónico, Dropbox o mediante **nbviewer**[9].

[8] Imagen de http://ipython.org/

[9] IPython Notebook Viewer es un servicio web gratuito que te permite compartir versiones HTML estáticas de archivos notebook.

Figura 2.5: Compartir documentos en nbviewer.

2.2 Twitter

Como se ha comentado en el apartado 1.2, aunque bien se hubiera podido realizar el estudio sobre otra red social, finalmente se elige a Twitter debido a que ofrece ciertas ventajas que la hacen muy interesante desde el punto de vista de *analytics*.

Twitter es una aplicación de microblogging que reúne las ventajas de los blogs, las redes sociales y la mensajería instantánea. Podría entenderse como una nueva forma de comunicación que permite a los usuarios estar en contacto con personas o entidades de su interés en tiempo real a través de mensajes breves de texto.

Twitter fue fundado en el 2006 en California por los estadounidenses Evan Williams y Biz Stone y con la colaboración de Jack Dorsey y Noah Glass. La idea surgió en la compañía Odeo aunque existe cierta controversia acerca de cómo se gestó. En el inicio el proyecto tuvo distintos nombres hasta que finalmente quedó el actual y en 2009 se independizó convirtiéndose en el actual Twitter, Inc. [16]

Figura 2.6: Página de inicio de Twitter.

Los mensajes a través de los que se realiza la comunicación son conocidos como tuits y son la esencia de Twitter. Estos están formados por un máximo de 140 caracteres de texto plano y pueden contener básicamente imágenes, enlaces y vídeos. Estos mensajes pueden ser clasificados mediante etiquetas, palabras precedidas por el carácter "#" (almohadilla) conocidas como *hashtags*. También es posible mencionar y contestar a otros usuarios mediante *handles* que, de forma parecida a los *hashtags,* van precedidas de un carácter especial, en este caso la "@" (arroba). Asimismo es posible compartir mensajes de otros usuarios, lo que se conoce como retuit, y también citar un tuit lo que no es más que el contenido del tuit original entre "" (comillas). La imagen siguiente muestra un retuit y algunos de los elementos citados.

Figura 2.7: Retuit de la UAB Barcelona sobre la tragedia en Charlie Hebdo.

Por otro lado Twitter permite a los usuarios leer en su propia página de inicio (o *timeline*) las publicaciones de otros usuarios sin la necesidad de tener que acceder a cada una de las páginas de usuario. Para ello los usuarios deben "seguir" a aquellos otros de los cuales quieren leer sus publicaciones.

Extrapolando este tipo de relación puede entenderse a Twitter como una red de usuarios que siguen y a su vez son seguidos por otros usuarios. Los usuarios a los que sigue uno se conocen como *friends* y en cambio, aquellos usuarios que siguen la cuenta de uno se les conocen como *followers*. Los usuarios que disponen de una gran cantidad de seguidores y que por tanto pueden tener un impacto muy grande en la red se conocen como *celebrities*, aunque no tienen ningún tipo de distintivo o trato especial dentro de Twitter. También existen usuarios verificados, una forma de distinguir marcas o personajes famosos con un pequeño *tick* azul.

Twitter puede ser usado con diferentes propósitos y situaciones. Por ejemplo con el fin de divulgar noticias o crear hilos de discusión sobre temas de actualidad, organizar protestas, publicitar productos, etc.

Y aunque el uso más común, pensando en el listado anterior, implica que las cuentas de usuario sean públicas (es la opción más lógica desde el punto de vista de la divulgación de la información), también existen cuentas privadas las cuales son muy útiles en la comunicación privada de grupos de personas sin el objetivo de recibir lectores.

2.2.1 Twitter API

Twitter pone a disposición de los desarrolladores una gran variedad de documentación, herramientas y APIs. Se puede encontrar desde *widgets*[10] para incorporar en páginas web, o información sobre Twitter Cards[11], hasta soporte sobre Apps para móvil, la API de Streaming[12], o la API REST. [7]

La API REST o API pública es la utilizada en el proyecto para la extracción de información de las cuentas de Nike y adidas. Ésta proporciona acceso de lectura sobre los datos públicos en Twitter, y por lo tanto no requiere disponer datos de

[10] Twitter para Sitios Web es un conjunto de widgets embebidos, botones y herramientas para integrar Twitter en su web o aplicación.

[11] Tarjetas de Twitter permiten mostrar contenido adicional junto a un tuit.

[12] La API Streaming permite realizar consultas de la API REST de forma continuada a través de conexión HTTP largas con las que así extraer información en tiempo real.

ingreso de las cuentas a las que se pretende acceder, por lo que se pueden extraer datos como los tuits de un usuario, información de perfiles usuario y datos de los seguidores entre otras opciones de una forma muy sencilla y para cualquier cuenta.

Para configurar la API, primero hay que registrarse en Twitter y crear una App en Twitter Apps[13], luego generar las credenciales o el OAuth con el cual la API identifica las peticiones y así crear las respuestas JSON.

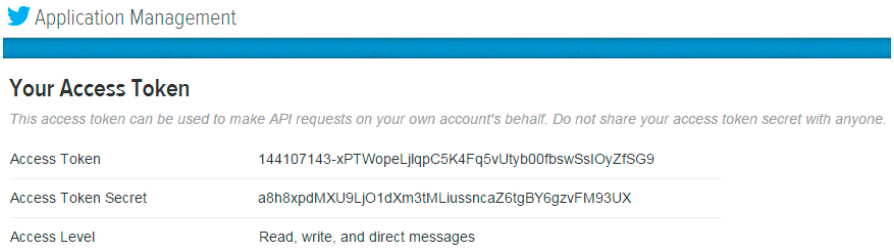


Figura 2.8: Ejemplo credenciales para la API de Twitter.

Algunas de las peticiones o métodos más comunes que se pueden realizar mediante la API son *GET followers/ids*, *GET friends/ids*, *GET followers/list*, *GET users/show*, *GET users/lookup* o *GET statuses/user_timeline*, aunque hay muchos más [7]. En el uso de las diferentes peticiones existentes hay que tener en cuenta algunos factores que limitan las posibilidades de estas. Los dos factores más importantes son:

1. Para aquellos métodos que devuelven una línea de tiempo de tuits, hay que tener en cuenta que la paginación de la información puede generar problemas debido a la naturaleza de tiempo real de los propios tuits. Por lo que lo más común será utilizar el parámetro *max_id* a partir del cual se cargaran los tuits, evitando así duplicidades en caso de haber actualizaciones entre petición y petición.
2. Otro factor a tener en cuenta es el límite en la tasa de peticiones. La tasa límite de la API se define en intervalos (o ventanas) de 15 minutos en las que según el

[13] URL: https://apps.twitter.com/

método utilizado se pueden realizar un número determinado de peticiones u otro. La tabla siguiente muestra los límites para algunos de los métodos.

Título	Familia	Peticiones 15-min (user auth)	Peticiones 15-min (app auth)
GET followers/ids	followers	15	15
GET followers/list	followers	15	15
GET friends/ids	friends	15	15
GET users/lookup	users	180	60
GET users/show	users	180	180
GET statuses/user_timeline	statuses	180	300

Tabla 2.1: Límite de peticiones por método cada 15 minutos.

2.2.2 Análisis previo

Después un primer análisis en Twitter, se ha detectado como los perfiles de usuario en la red social no contienen información relevante como la edad, el sexo, características físicas o la localización. Aunque esta última sí existe en el formulario de perfil, este es un campo de texto que muchos usan de forma equivocada o engañosa, por lo que no puede tomarse en cuenta. En este sentido, también cabe destacar como aunque los tuits pueden localizarse, se ha podido constatar cómo solamente unos pocos del total de seguidores de las marcas de estudio emplean este modo.

Por lo tanto, el análisis en Twitter se enfoca desde el punto de vista de la actividad de los seguidores, la calidad de estos y sus intereses, datos existentes para cada usuario y accesibles públicamente mediante la API para prácticamente todos ellos.

Al concretar este primer análisis, se revela como ambas marcas disponen de una multitud de cuentas verificadas según sector o finalidad. La tabla 2.2 muestra 10 cuentas del total existente, con el fin de dar a conocer las posibilidades en el análisis, y destacando las dos cuentas escogidas para el estudio en el proyecto: **@adidas_ES** y **@Nike_Spain**, elegidas por ser dos cuentas generales destinadas al mercado español.

Cuentas verificadas de adidas	Cuentas verificadas de nike
@adidas	@Nike
@adidasUS	@NikeNYC
@adidasUK	@NikeLasVegas
@adidasFR	@NikeDallas
@adidas_ES	@NikeLA
@adidasAR	@Nike_Spain
@adidaslatam	@NikeSupport
@adidasoriginals	@nikefootball
@adidasrunning	@nikebasketball
@adidastennis	@nikegolf

Tabla 2.2: Extracto de las diferentes cuentas en Twitter para adidas y Nike.

2.3 Machine Learning

Las técnicas de aprendizaje automático tienen multitud de aplicaciones distintas, y consiguen afrontar una variedad de problemas que tratan desde temas relacionados con el diagnóstico médico, motores de búsqueda y robótica, hasta temas como la segmentación de mercados, análisis de redes sociales y minería de datos. Las posibilidades de estas últimas son las que motivan el uso de *Machine Learning* en el presente trabajo, herramientas con las que extraer la información de interés y de valor comentada en el primer capítulo.

Existen distintas definiciones de *Machine Learning*, aunque quizás las dos más simbólicas fueron la de Arthur Samuel en 1959, cuando al diseñar el primer programa capaz de jugar a damas consigo mismo y aprender de su experiencia, dijo que "*ML era el campo de estudio que hacía que las computadoras adquirieran la habilidad de aprender sin una programación explícita*".

Y en 1998, cuando Tom Mitchell hizo una definición algo más técnica y moderna al decir que "*un programa informático aprende de una experiencia E, con respecto alguna tarea T y de la medida del rendimiento P (en inglés performance), si su rendimiento en T medido por P, mejora con la experiencia E*".

Pensando en el ejemplo del juego de las damas se puede entender la definición de Tom Mitchell, donde la experiencia E será la experiencia del programa por jugar decenas de miles de veces al juego contra ella misma, la tarea T será la tarea de jugar a damas y el rendimiento R, será la probabilidad de que gane el siguiente juego de damas contra algún oponente nuevo.

Los diferentes algoritmos de *Machine Learning* pueden clasificarse en dos grandes grupos o tipos de aprendizaje:

- Aprendizaje supervisado.
- Aprendizaje no supervisado.

2.3.1 Aprendizaje supervisado

En el aprendizaje supervisado se dispone de un conjunto de datos o *dataset* el cual está formado por un seguido de objetos o valores de entrada y en el cual se conocen los valores de salida que estos producen. Así pues, estas técnicas tratan de crear una función llamada hipótesis que represente los datos del *dataset* y a partir de estos ser capaz de predecir datos de salida a partir de nuevos datos de entrada.

El aprendizaje supervisado puede dividirse a su vez en dos categorías más según la naturaleza de los datos de salida.

- Regresión.
- Clasificación.

Se tiene un problema de **regresión** cuando por ejemplo se trata de predecir el precio de casas según el tamaño de estas. Se puede deducir que cuanto más grande sea una casa mayor será su precio, pero este valor estará siempre en un rango continuo y será un valor numérico.

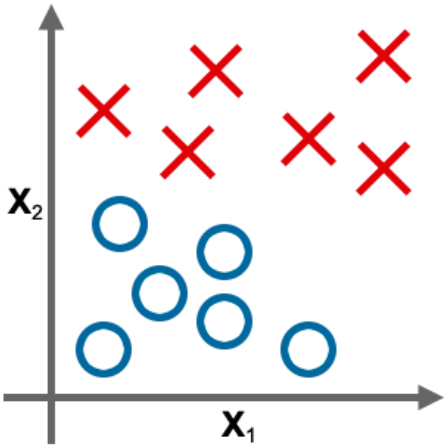


Figura 2.9: Representación de regresión.

En cambio, para el caso de la **clasificación**, un ejemplo puede ser cuando se trata de predecir si un tumor es cancerígeno o no según su tamaño. Como se puede ver en la siguiente figura, los valores de salida siempre formaran parte de un grupo discreto, en este caso toman dos valores {*Sí*, *No*}.

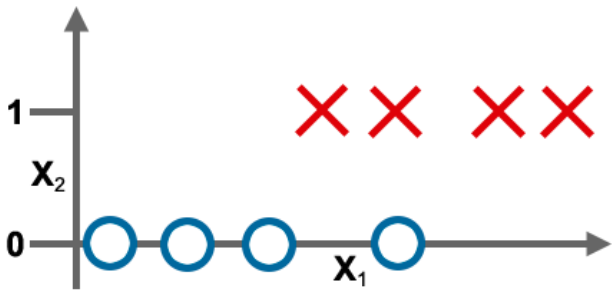


Figura 2.10: Representación de clasificación.

Algunos de los algoritmos más populares son las redes neuronales y las máquinas de soporte vectorial (SVM), aunque el listado es amplio y podrían destacarse también los árboles de decisión o la regresión gaussiana entre muchos otros.

Las **redes neuronales** se basan en los sistemas nerviosos de los animales, su estructura en red simula la interconexión de las neuronas en un cerebro. La red se divide en capas y estas a su vez en neuronas donde cada una de las neuronas está

conectada a las neuronas de la capa siguiente, es decir, cada una de las neuronas genera una o varias respuestas (número de neuronas de la capa siguiente) según las distintas entradas que recibe de las neuronas de la capa anterior. Según la estructura de la red, se pueden tratar problemas de clasificación binaria (una única neurona de salida) o de clasificación multiclase (varias neuronas en la última capa).

La imagen siguiente muestra un ejemplo de una red neuronal binaria, dividida en 3 capas, con 3 neuronas en la capa de entrada y la intermedia.

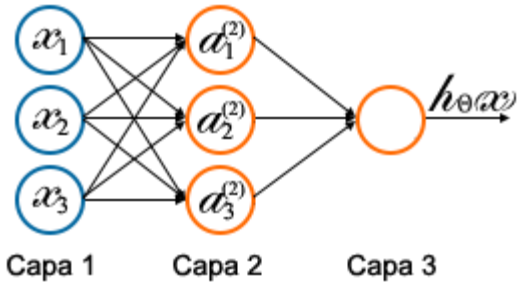


Figura 2.11: Representación de una red neuronal.

$$a_1^{(2)} = g\left(\Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) \tag{2.1}$$

$$a_2^{(2)} = g\left(\Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) \tag{2.2}$$

$$a_3^{(2)} = g\left(\Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) \tag{2.3}$$

$$h\Theta(x) = a_1^{(3)} = g\left(\Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) \tag{2.4}$$

donde $a_i^{(j)}$ es la neurona *i* de la capa *j* y $\Theta^{(j)}$ es la matriz de pesos que forman la capa *j* y la *j+1*.

Las **máquinas de soporte vectorial** buscan básicamente maximizar el margen que separa cada una de las diferentes clases en las que se divide el espacio mediante uno o varios hiperplanos *h*, para ello proyectan los datos en un espacio de dimensionalidad superior o espacio de Hilbert.

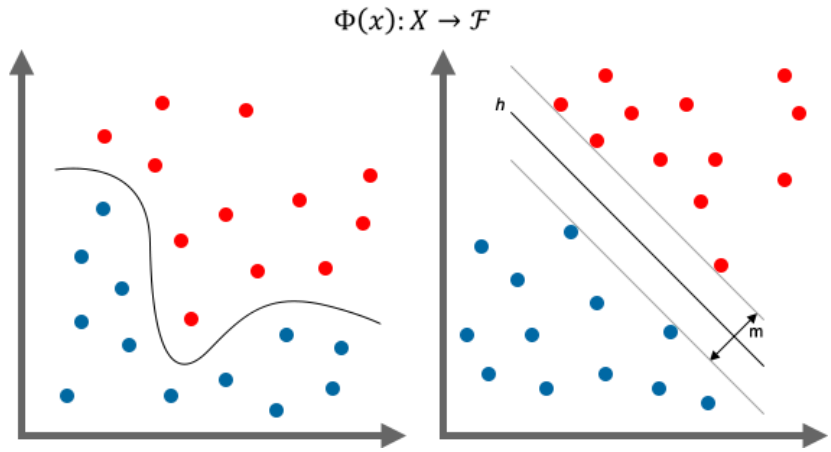


Figura 2.12: (a) Espacio de entrada X con muestras no linealmente separables. (b) Espacio de características F con muestras linealmente separables y margen máximo.

Los hiperplanos que separan los datos pueden tomar diferentes formas, lo que se conoce como funciones Kernel. Entre las más conocidas destacan:

- Lineal: $K(x, y) = x^T y + c$ (2.5)
- Polinómica: $K(x, y) = (\alpha x^T y + c)^n$ (2.6)
- Gaussiana (*RBF*): $K(x, y) = exp\left(\frac{-\|x-y\|^2}{2\sigma^2}\right)$ (2.7)
- Sigmoidal: $K(x_i, x_j) = \tanh(\alpha x^T y + c)$ (2.8)

2.3.2 Aprendizaje no supervisado

A diferencia del supervisado, en el aprendizaje no supervisado no se disponen de los datos de salida o resultados que se quieren predecir, por lo que entonces estas técnicas tratan de relacionar, buscar estructuras o agrupar, los objetos de entrada según sus características.

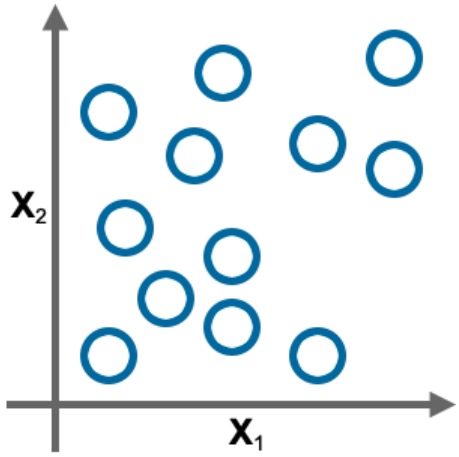


Figura 2.13: Representación de aprendizaje no supervisado.

Un ejemplo claro es el de Google News, donde las noticias son agrupadas por temas o categorías similares. El aprendizaje no supervisado tiene muchas otras aplicaciones como por ejemplo en el análisis de las redes sociales al agrupar amigos con características similares, también en marketing al tratar de segmentar los clientes según sus datos para poder venderles productos más afines, o incluso en el análisis de datos astronómicos donde llegan a conseguir explicar cómo nacen las galaxias y se agrupan las estrellas.

A los algoritmos más populares en el aprendizaje no supervisado se les conoce como algoritmos de agrupamiento o *clustering*, y entre todos ellos, quizás el más famoso sea k-means, aunque también cabe destacar otros algoritmos como por ejemplo *hierarchical clustering*.

Hierarchical clustering o en castellano agrupamiento jerárquico, tiene el fin, como su propio nombre indica, de construir una jerarquía de grupos, la cual, por su similitud en la forma estructural, puede entenderse como las ramificaciones de un árbol. La jerarquía comúnmente suele representarse mediante un dendograma (ver ejemplo en la figura 2.14) y este puede ser calculado mediante multitud de métricas, lo que sería una ventaja del algoritmo, aunque por el contrario estos son demasiado lentos para grandes conjuntos de datos. Algunas de las métricas más populares son:

- Distancia euclídea: $\|p - q\|_2 = \sqrt{\sum(p_i - q_i)^2}$ (2.9)
- Distancia Manhattan: $\|p - q\|_1 = \sum|p_i - q_i|$ (2.10)
- Similitud coseno: $\cos(p, q) = \frac{\sum(p_i \cdot q_i)}{\sqrt{\sum(p_i)^2} \cdot \sqrt{\sum(q_i)^2}}$ (2.11)

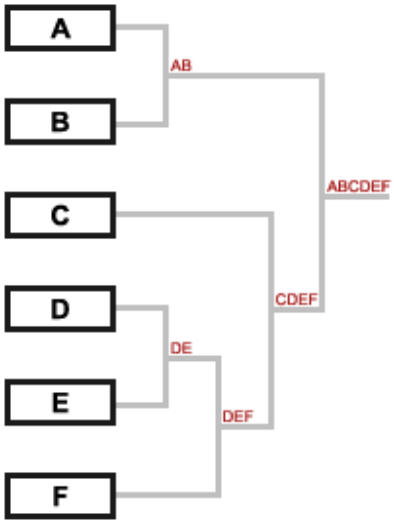


Figura 2.14: Representación dendograma.

K-means trata de dividir el espacio en el que se representan los datos en *K* grupos. A cada muestra se le asigna el grupo más cercano mediante el cálculo de la distancia euclídea (3.7) entre la muestra y el centro de gravedad de cada *cluster*. Al inicio del algoritmo la división espacial se realiza de forma aleatoria, luego se calculan los centros de gravedad, se asignan los nuevos grupos y se repite el proceso hasta converger.

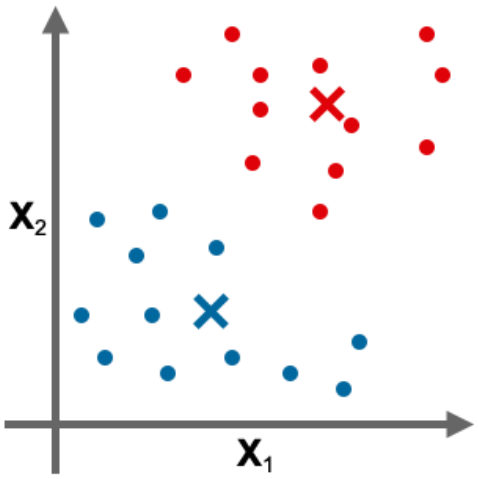


Figura 2.15: Representación del agrupamiento de k-means.

Entre las diferentes herramientas ML comentadas, en el presente proyecto se decide implementar k-means debido a su gran uso, robustez, sencillez y lógicamente al hecho de tener que afrontar un problema de aprendizaje no supervisado.

2.4 Natural Language Processing

En el estudio de la red social Twitter, gran parte del análisis se enfoca en los tuits y la información que los textos de estos contienen. Es por ello que el presente proyecto requerirá de técnicas sobre el procesado del lenguaje natural para extraer los temas más usados en los tuits.

El procesamiento de lenguajes naturales, en inglés *Natural lenguaje processing* o NLP, es uno de los campos más importantes en la Inteligencia Artificial, basándose en la comunicación entre el hombre y la computadora mediante el lenguaje humano a través de sistemas informáticos, por medio de la voz o del texto. En la actualidad, con el aumento en las comunicaciones a través del *social media* y la gran cantidad de información que se encuentra en la red, el NLP posibilita una variedad interesantísima de aplicaciones.

Algunas de estas aplicaciones son la **extracción de la información,** por ejemplo al comprender un email que contiene una citación, extrayendo así los datos de fecha, lugar y motivo, con el fin de poder programar un calendario automáticamente. El **análisis del sentimiento**, por ejemplo para conocer el impacto que produce un anuncio en un blog a partir de los comentarios que este genera. La **traducción automática de textos** donde el traductor de Google es un ejemplo claro. **Conversaciones automatizadas**, un ejemplo serían algunos de los contestadores automáticos de las compañías telefónicas. Y finalmente la **respuesta a preguntas**, responder preguntas de interés general a través del análisis de la red.

En contra, todas estas aplicaciones encuentran diferentes problemas puesto que existen dificultades intrínsecos al lenguaje complejas de tratar tales como, la detección del idioma del texto a analizar, la ironía con la que se comunica la información, la ambigüedad de textos segmentados con símbolos de puntuación, la aparición de palabras nuevas o neologismos, textos abreviados, etc.

2.4.1 Bag of Words

Bag of Words o BoW, es uno de los métodos más conocidos y usados para la representación de documentos en NLP. Tal y como Baeza-Yates y Ribeiro-Neto [11] describen, este método modela los documentos mediante un histograma de términos relevantes, es decir, representa cada documento mediante la frecuencia de aparición o número de veces que aparecen las palabras con un peso mayor, sin tener en cuenta el orden de aparición de las mismas.

Para su representación, se construye una matriz de *m* documentos y *n* palabras, donde cada documento representa una fila en la matriz y cada columna corresponde a una palabra, resultando una matriz de $m \cdot n$. donde cada fila de la matriz representa un documento y las frecuencias de las palabras que aparecen en este. Al conjunto de las palabras más relevantes se le conoce como diccionario.

A continuación se muestra un ejemplo muy sencillo con 4 documentos, donde después de eliminar los *stop-words* (palabras que no son relevantes, como por ejemplo los artículos), la matriz de frecuencias resultante queda representada en la tabla 2.3:

- **D1:** los planetas giran alrededor del sol.
- **D2:** las agujas del reloj giran.
- **D3:** las peonzas giran al igual que giran los planteas.
- **D4:** los planetas y el sol son astros.

	planetas	giran	alrededor	sol	agujas	reloj	peonzas	igual	astros
D1	1	1	1	1	0	0	0	0	0
D2	0	1	0	0	1	1	0	0	0
D3	1	2	0	0	0	0	1	1	0
D4	1	0	0	1	0	0	0	0	1

Tabla 2.3: Tabla de frecuencias de los documentos.

2.4.2 Term frequency-Inverse document frequency

Los pesos de los términos, como se ha visto en el punto anterior y se comenta en [13], son las cuentas del número de apariciones de cada término en cada uno de los documentos. Esta medida hace que la importancia de cada término quede desproporcionada por lo que suele representarse en escala logarítmica.

$$W_{t,d} = \begin{cases} 1 + \log tf_{t,d}, & tf_{t,d} > 0 \\ 0, & tf_{t,d} \leq 0 \end{cases} \tag{2.12}$$

En documentos extensos sucede que las frecuencias de los términos *tf* pueden ser fácilmente mayores que en documentos mucho más breves lo cual distorsiona la importancia real de las palabras, motivo por el que la frecuencia del término suele normalizarse mediante el número total de documentos *N*.

Aunque se normalicen y escalen las frecuencias de los términos, la importancia de cada palabra aumenta de forma proporcional al número de veces que esta aparece en un documento por lo que se busca compensar este efecto teniendo en cuenta la

frecuencia en el total de documentos en los que la palabra aparece. La idea de G. Salton es dar importancia a aquellos términos que aparecen en menos documentos, delante de los que lo hacen en prácticamente todos, dado que estos discriminan poco o nada a la hora de la representación del conjunto. Este factor se conoce como *term frequency-inverse document frequency*. [12]

Así pues, para el cálculo de *tf-idf*, se propone el *inverse document frequency*. Este se obtiene dividiendo el número total de documentos entre el número de documentos que contienen el término y aplicando el logaritmo:

$$idf_t = \log \frac{N}{df_t} \tag{2.13}$$

donde *N* es el número total de documentos y df_t es la frecuencia de documentos que contienen el término *t*.

Finalmente, el cálculo del peso *tf-idf* resulta en una combinación de ambos factores:

$$W_{t,d} = tf_{t,d} \cdot idf_t \tag{2.14}$$

donde ambos factores se miden en (tabla 2.4) y (tabla 2.5).

Siguiendo con el ejemplo del punto anterior, en la tabla 2.4 se muestra el cálculo del *idf* a partir del cual se calculan los pesos *tf-idf* proporcionados en la tabla 2.5.

	planetas	giran	alrededor	sol	agujas	reloj	peonzas	igual	astros
idf	1.333	1.333	4	2	4	4	4	4	4
log	0.124	0.124	0.602	0.301	0.602	0.602	0.602	0.602	0.602

Tabla 2.4: Cálculo de idf.

	planetas	giran	alrededor	sol	agujas	reloj	peonzas	igual	astros
D1	0.124	0.124	0.602	0.301	0	0	0	0	0
D2	0	0.124	0	0	0.602	0.602	0	0	0
D3	0.124	0.248	0	0	0	0	0.602	0.602	0
D4	0.124	0	0	0.301	0	0	0	0	0.602

Tabla 2.5: Pesos tf-idf.

2.5 Estudio previo

Para adquirir los conocimientos básicos necesarios desarrollados en los puntos anteriores, se ha realizado un estudio previo dividido en 4 temas distintos: Python, Twitter, *Machine learning* y *Natural Language Processing*:

El estudio previo sobre **Python** comienza con la realización del curso "Una introducción a la programación interactiva en Python" [1] en Coursera y este es reforzado mediante la información expuesta en [3] por Mark Lutz. El curso consta de una duración de 12 semanas en la que se han desarrollado diferentes miniproyectos:

- El juego Pong[14].
- El juego Memory[15].
- El juego Blackjack[16].
- El juego RiceRocks[17].

Un primer estudio sobre las principales herramientas de analítica citadas en el punto 1.2, junto con lo comentado por Guy Kawasaki [8] y la información pública que ofrece la API de **Twitter**, servirá para adquirir una base sobre *analytics* y las posibilidades a implementar en el presente proyecto sobre la red social.

En cuanto al análisis inicial sobre **Machine Learning**, este se fundamenta a partir de las video-lecturas del famoso curso "Machine Learning" [9], también en Coursera y los conocimientos adquiridos son contrastados con el libro [10] de Peter Harrington.

Del mismo modo que en el caso de ML, el estudio preliminar de **Natural Language Processing** se basa en las video-lecturas de un curso en Coursera,

[14] Jugar a Pong en URL: http://www.codeskulptor.org/#user38_tn5AsJEPWtT1lvy.py
[15] Jugar a Memory en URL: http://www.codeskulptor.org/#user38_O6UZaAKt52O74PJ.py
[16] Jugar a Blackjack en URL: http://www.codeskulptor.org/#user38_Ot7VHohOrUtP7My.py
[17] Jugar a RiceRocks en URL: http://www.codeskulptor.org/#user38_CFdKUJSBoTha1yq.py

concretamente del curso “Procesamiento del lenguaje natural” [12], y se respalda mediante el libro “Natural Language Processing with Python” [13].

Finalmente, una vez terminado el estudio previo y concluido el desarrollado de los temas principales con los que trabajar, se decide utilizar y/o implementar las herramientas siguientes:

- Python (y el Notebook de IPython) como lenguaje de programación con el que implementar la solución propuesta.
- La API de Twitter con la que acceder a la información de las cuentas.
- El algoritmo k-means con el que segmentar la información de interés. Con apoyo del método Elbow para conocer el número de *clusters* óptimo.
- *Term frequency-Inverse document frequency* con el que dar unos pesos a los topics y así conocer los más relevantes de una forma más certera.

3 Metodología desarrollada

En este capítulo se busca identificar carencias o necesidades que puedan requerir las dos grandes marcas de estudio, a partir de las cuales surge la propuesta a implementar y finalmente la solución elaborada mediante las técnicas y herramientas del ámbito de la Ingeniería de Telecomunicaciones señaladas en el punto anterior.

3.1 Propuesta a resolver

Después de varios años de experiencia profesional en una empresa de marketing online en Badalona y tras contrastar con los compañeros algunas de las problemáticas existentes, junto con la oportunidad de haber hablado con expertos en marketing digital (personas de mi entorno social) de una pequeña empresa y otra gran compañía, ambas de Zaragoza, han surgido las siguientes problemáticas a resolver, propuestas que podrían interesar a las dos marcas de estudio.

1. Uno de los problemas que puede padecer una compañía internacional y de renombre es la de disponer seguidores que no interesan de forma directa y que por tanto, en caso de ser un volumen importante, desvirtúan las estadísticas y datos que pueden extraer mediante otras herramientas. Este tipo de seguidores pueden ser:

- Seguidores *fakes* creados mediante "granjas".
- Seguidores con muy poca o ninguna actividad.
- Seguidores con muy pocos *followers*.
- Seguidores extranjeros, no son *target* directo de las publicaciones de la marca.

2. Al crear una cuenta en Twitter, la cantidad de seguidores de la misma es muy baja y por tanto, el objetivo de toda empresa al inicio es buscar ganar cuantos más seguidores mejor, sin importar la calidad de estos, pero pasado cierto umbral, y más pensando en compañías importantes con grandes cuentas, la finalidad se convierte en ganar seguidores de calidad, *followers* con valor desde el punto de vista del impacto que estos pueden generar en la cuenta. Así pues se hace interesante detectar a este tipo de seguidores.

3. Una vez detectados los seguidores más influyentes, es importante conocerlos más a fondo desde el punto de vista del impacto que estos pueden generar y descubrir así la importancia que pueden representar en la cuenta, e incluirlos dentro de un plan de *marketing*:

- Conocer el volumen que estos representan.
- Conocer la contribución que realizan a la marca.
- Detectar cuales son los que más menciones realizan para comprobar si estos tienen algún tipo de contrato y así repensar la relación.
- Segmentar este tipo de seguidores con la finalidad de buscar diferentes tipos de relaciones: contratos *premium*, contratos básicos, embajadores de marca (sin contrato), etc.

4. Con la finalidad de seguir conociendo a los seguidores más relevantes, también llamados en este proyecto como seguidores *celebrities* aunque realmente no sean personajes famosos, es importante detectar cuales son los temas más comunes que tratan:

- Extraer los temas de interés con los que conocer mejor la cuenta.
- Buscar relaciones entre los *celebrities* y los distintos temas sobre los que hablan con la finalidad de agrupar *topics* y conocer aquellos seguidores relevantes que pueden promocionar la marca en distintos ámbitos.

3.2 Implementación realizada

Para implantar la propuesta expuesta en el apartado anterior, se ha procedido a asociar cada uno de los 4 puntos descritos a un archivo Python distinto y este a su vez corresponde a cada uno de los 4 experimentos realizados. Así pues, en cada fichero se han desarrollado las funciones vinculadas a las cuestiones que integra cada uno de los puntos.

Además de estos 4 archivos correspondientes a los 4 experimentos, se crean 2 ficheros más. Uno contiene el desarrollo de los algoritmos ML (utilizados en los puntos 3 y 4), y otro dispone de las funciones destinadas a la obtención y descarga de la información de las cuentas en Twitter.

Finalmente se crea el archivo principal o *main* desde el cual se realiza el desarrollo completo del trabajo a partir de las llamadas a las funciones implementadas en el resto de ficheros. La figura siguiente muestra la estructura descrita y el orden en el que se ha implementado el proyecto.

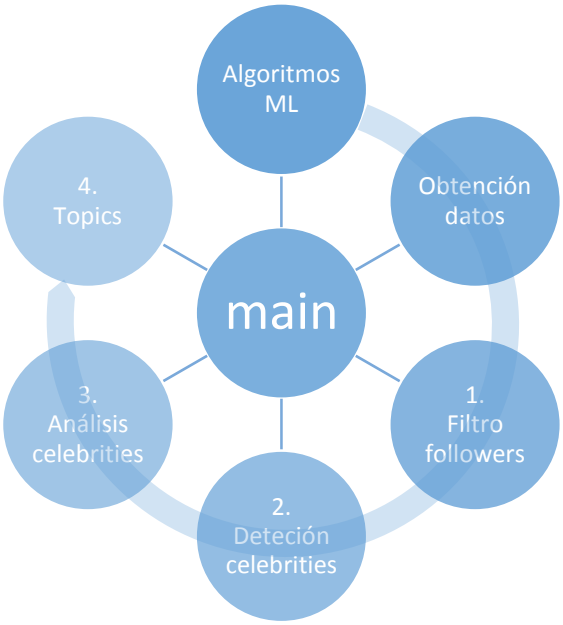


Figura 3.1: Estructura de ficheros del proyecto.

3.2.1 Algoritmos ML

El desarrollo del proyecto comienza con la implementación de los algoritmos ML necesarios en el experimento. Es por ello, que en el fichero *ml_functions.py* se recogen básicamente dos funcionalidades:

- Algoritmo PCA. [10]
- Algoritmo k-means original[18] (y método Elbow). [9]

Algoritmo PCA

El algoritmo PCA (*Principal Component Analysis*) tiene el objetivo de reducir la dimensionalidad de los datos o muestras a analizar con la finalidad de minimizar costes computacionales, reduciendo espacio de memoria en disco y aumentando así la velocidad del algoritmo de aprendizaje utilizado, consiguiendo paralelamente eliminar aquellas características que sean irrelevantes en el proceso de segmentación.

Primero se calcula la matriz de covarianza $\sum$ (expresión 3.1) del conjunto de datos *x*.

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}\left(x^{(i)}\right)\left(x^{(i)}\right)^{T} \tag{3.1}$$

donde $x^{(i)}$ representa cada una de las *m* muestras del conjunto $\left\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\right\}$ y *n* el número de características de cada muestra.

A partir de esta matriz resultante de *n·n* dimensiones pueden calcularse los autovectores y autovalores de forma muy sencilla mediante la función svd (*singular value decomposition*) resultando una matriz *U* con dichos autovectores y un vector

[18] Existe un abundante número de versiones y variantes del algoritmo k-means, com por ejemplo: k-means++ y Spherical k-means entre otros.

s con los autovalores ordenados todos ellos de mayor información a menor (expresiones 3.2 y 3.3 respectivamente).

$$U = \begin{bmatrix} | & | & | \\ u^{(1)} & u^{(2)} \cdots & u^{(n)} \\ | & | & | \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{3.2}$$

$$v = \begin{bmatrix} v^{(1)} & v^{(2)} \cdots & v^{(n)} \end{bmatrix} \in \mathbb{R}^{1 \times n} \tag{3.3}$$

Después se escogen los *k* autovectores con mayor información y seguidamente se proyectan los datos en este nuevo espacio Z de dimensionalidad reducida (expresión 3.4 y 3.5 respectivamente):

$$Ureducida = \begin{bmatrix} | & | & | \\ u^{(1)} & u^{(2)} \cdots & u^{(k)} \\ | & | & | \end{bmatrix} \in \mathbb{R}^{n \times k} \tag{3.4}$$

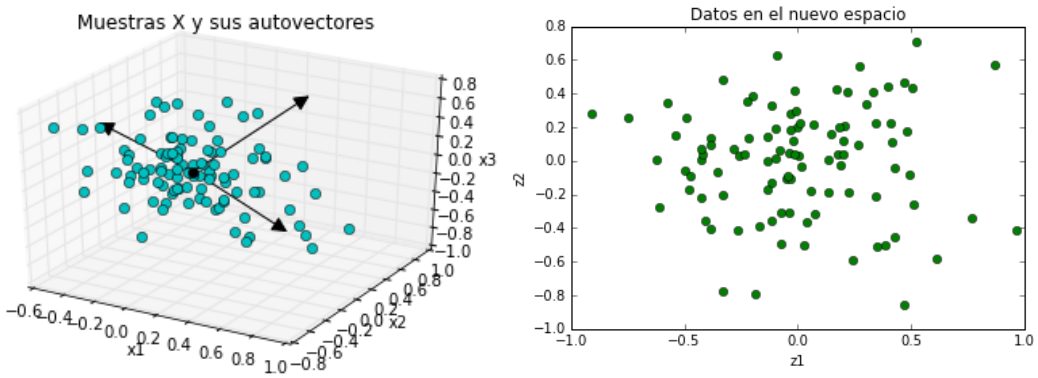
$$Z = Ureducida^T * x \tag{3.5}$$

Finalmente, añadir que en la elección de las *k* dimensiones del nuevo espacio, se computa la varianza a partir de los autovalores *s* (expresión 3.3) para cada una de las *k* posibles dimensiones, escogiendo así la dimensionalidad menor que retenga la varianza deseada.

$$var \leq \frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \tag{3.6}$$

La figura siguiente muestra el desarrollo inicial del algoritmo PCA implementado en el Notebook de IPython, en el que se presenta una reducción de dimensinalidad de 3D a 2D para un conjunto de datos aleatorios. Para ver el ejemplo completo visitar la publicación[19].

[19] Ejemplo 2D URL: http://nbviewer.ipython.org/gist/anonymous/b874b826579d6b89b01a
Ejemplo 3D URL: http://nbviewer.ipython.org/gist/anonymous/e226f575956dd86ee867



(a) (b)

Figura 3.2: (a) Muestras del conjunto datos en 3D (b) Proyección de las muestras en un espacio reducido de 2D tras usar PCA.

Algoritmo k-means

El algoritmo k-means descrito en la sección 2.3.2 queda implementado en la función `kmeans`, la cual se divide en 4 funciones más, cada una con diferentes propósitos:

1. `cluster_assignment`
2. `move_centroids`
3. `cluster_data`
4. `cost_function`

La primera función se encarga de la asignación de las muestras a los *clusters* mediante el cálculo de la distancia euclídea (3.7) entre ambos.

$$c^{(i)} = \min_{k} \sum_{k=1}^{K} \sum_{i=1}^{m} \left\| x^{(i)} - \mu_k \right\|^2 \tag{3.7}$$

donde μ_k son los centros de los K grupos o *cluster centroids* $\{\mu_1, \mu_2, \dots, \mu_K\}$.

La segunda función calcula los centros de gravedad de cada uno de los distintos grupos a partir de la media de cada una de las muestras del *cluster* en cuestión.

$$\mu_k = \frac{1}{n_k} \sum_{x^{(i)} \in c^{(k)}} x^{(i)} \tag{3.8}$$

donde n_k es el número de muestras pertenecientes al *cluster* $c^{(k)}$.

En la tercera función se inicializa el algoritmo asignando los *centroids* μ_k a k muestras del conjunto al azar. Posteriormente se repiten las funciones 1 y 2 hasta que la asignación del *cluster* en la primera función no varía, es decir, el algoritmo converge. Para evitar que k-means converja en una solución local, debido a la inicialización aleatoria de los centros, este debe optimizarse. Para ello se aplica el algoritmo varias veces y se toma como resultado final el que minimice la función de costes:

$$J = \frac{1}{m} \sum_{i=1}^{m} \left\| x^{(i)} - \mu_{c^{(i)}} \right\|^2 \tag{3.9}$$

donde $\mu_{c^{(i)}}$ son los centros del cluster asignada a la muestra $x^{(i)}$.

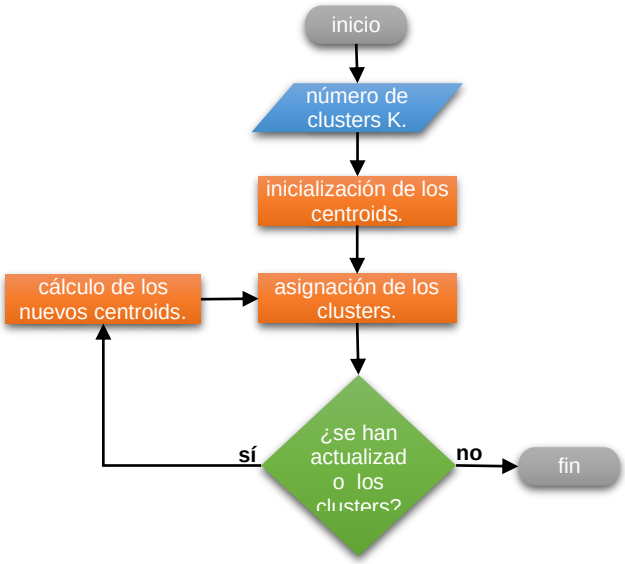


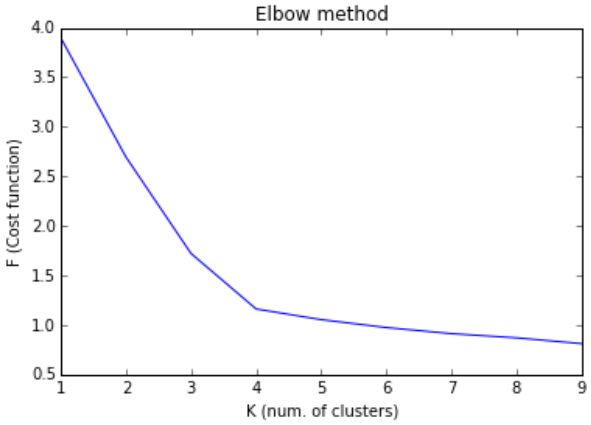
Figura 3.3: Diagrama bloques del algoritmo k-means.

Método Elbow

El **método Elbow** ayudará a escoger el número de grupos óptimo con el que dividir el conjunto de muestras. Este método trata de aplicar el algoritmo k-means para distintos valores de K y a partir de la representación de la función de costes *J* de cada uno de los resultados con respecto a las distintas K, se determina el número de grupos que mejor segmenten los datos.

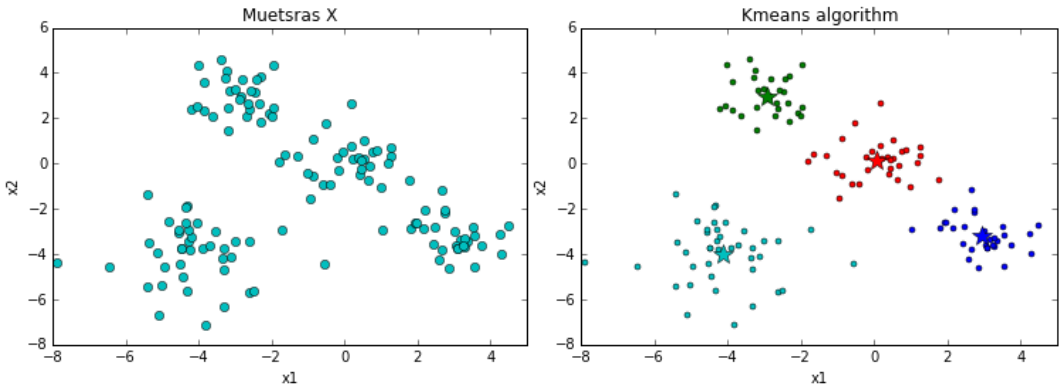
Así pues el valor de K óptimo será aquel en el que el gráfico dibuja un pico o cambio brusco en la evolución de *J*, dibujando una forma parecida a la de un brazo y su codo, de ahí el nombre.

Las figuras siguientes muestran el desarrollo inicial del algoritmo k-means y el método Elbow, ambos modelos implementados en el Notebook de Ipython. Para ver el ejemplo completo visitar la publicación [20].



(a)

[20] URL: http://nbviewer.ipython.org/gist/anonymous/e249a324da4dc184cb51



(b) (c)

Figura 3.4: (a) Método Elbow con solución K=4 (b) Muestras del conjunto de datos (c) Datos segmentados mediante k-means tras el análisis del método Elbow.

3.2.2 Obtención de datos

En la fase de descarga de datos es importante destacar el uso de la librería *Twython* [15] mediante la cual Python se comunica a la API de Twitter. Una vez obtenidas las credenciales descritas en el capítulo 2.2.1, se crea el objeto `twitter` el cual dispone de todos los métodos de la API con los que realizar las consultas.

También es importante señalar que los datos descargados para el estudio corresponden al período de la segunda quincena de noviembre, y por tanto hay que tener presente que desde entonces ambas cuentas han visto aumentado su número de seguidores.

Retomando el desarrollo descrito en la figura 3.1, en el fichero *data_admin.py* se implementan un seguido de funciones con la finalidad de descargar los datos de estudio de las cuentas de Twitter y almacenarlos para su posterior análisis. Las funciones destinadas a tal propósito son:

1. `friends_ids`
2. `followers_ids`
3. `twitter_user_data`

4. `twitter_user_status`

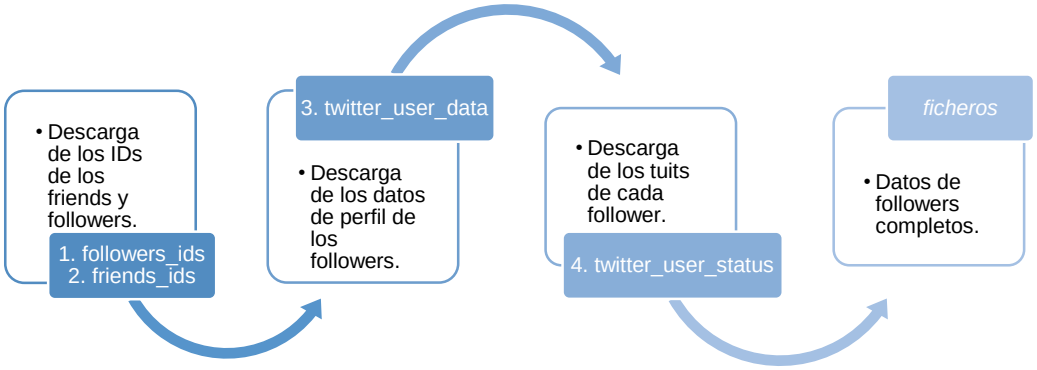


Figura 3.5: Proceso de descarga de los datos.

Las funciones 1 y 2 tienen la finalidad de descargar y almacenar por separado los IDs de los usuarios *friends* y *followers* de ambas marcas, a partir de los métodos *GET friends/ids* y *GET followers/ids*.

La función 3 reúne los datos de perfil de cada usuario mediante el método *GET users/lookup*. Algunos de los datos de perfil más relevantes son: `name`, `screen_name`, `created_at`, `description`, `url`, `favourites_count`, `followers_count`, `friends_count`, `lang`, `location` y `protected`.

Finalmente, la función 4 se encarga de la descarga y almacenamiento de los últimos 200 tuits[21] de cada uno de los *followers*, a partir de los IDs obtenidos en la función 2 y esta vez mediante el método *GET statuses/user_timeline*.

Todo el proceso de obtención de datos queda limitado en el tiempo debido a las restricciones en el número de consultas que la API permite (ver tabla 2.1). A continuación se recogen los tiempos mínimos necesarios por la API en la descarga de los datos completos. Como se puede observar en la tabla 3.1 los tiempos son muy considerables.

[21] Límite de tuits permitidos por la API de Twitter.

Título *método*	Peticiones en 15-min	Usuarios / petición	Tiempo espera / petición	Tiempo adidas*	Tiempo Nike*
GET friends/ids *get_friends_ids*	15	5000	60s	2.42h	0.18h
GET followers/ids *get_followers_ids*	15	5000	60s	2.42h	0.18h
GET users/lookup *lookup_user*	60	100	15s	30.35h	2.34h
GET statuses/user_timeline *get_user_timeline*	300	1	3s	606.96h	46.8h
			Total marca:	642.15h	49.5h
			TOTAL:	**28.81 días**	

* tiempos calculados a partir del número de friends y followers en la tabla 3.2.

Tabla 3.1: Tiempos teóricos en la obtención de datos.

	Núm. *friends*	Núm. *followers*
adias	188	728504
Nike	203	56289

* datos a fecha 22/12/2014.

Tabla 3.2: Número de friends y followers por marca.

3.2.3 Solución a la propuesta

A continuación se describe la solución realizada a cada uno de los puntos propuestos al inicio del capítulo y las herramientas utilizadas para tal efecto.

1. Filtro de followers

El fichero *step1_filter.py* tiene la función de eliminar aquellos usuarios que no cumplan con ciertas variables. Así pues se filtran aquellos seguidores inactivos, irrelevantes y extranjeros a partir de los parámetros siguientes:

- `followers_filter`: número de seguidores mínimo para cada *follower*.
- `monthly_tweets`: cantidad media mínima de tuits por mes desde la creación de la cuenta.
- `lang_filter`: tener configurado en el perfil el idioma español.

Además de descartar a los seguidores no deseados, se analiza la densidad de estos sobre el total de *followers*, se calcula el alcance potencial de las cuentas (suma

de los *followers* y de los seguidores de los *followers*) y se muestra el número de seguidores en común entre ambas marcas mediante un diagrama de Venn.

2. Detección de celebrities

El archivo *step2_detection.py* obtiene aquellos usuarios más relevantes. En lugar de únicamente tomar en cuenta el número total de seguidores que tiene cada *follower* (esta métrica es errónea por el hecho de que muchos usuarios en Twitter utilizan la técnica del *followback* [8]), estos son representados según el número de seguidores y el número de *friends*. Debido a los grandes rangos entre los valores de cada usuario, se representan en escala logarítmica. Seguido se extrae un valor umbral, el cociente entre el número de *followers* y el número de *friends*, a partir del cual se consideraran los seguidores como *celebrities*. Finalmente se usa el filtro implementado en el punto 1 de forma más restrictiva, forzando así a que los *followers* finales tengan una gran cantidad de seguidores y alta actividad en Twitter:

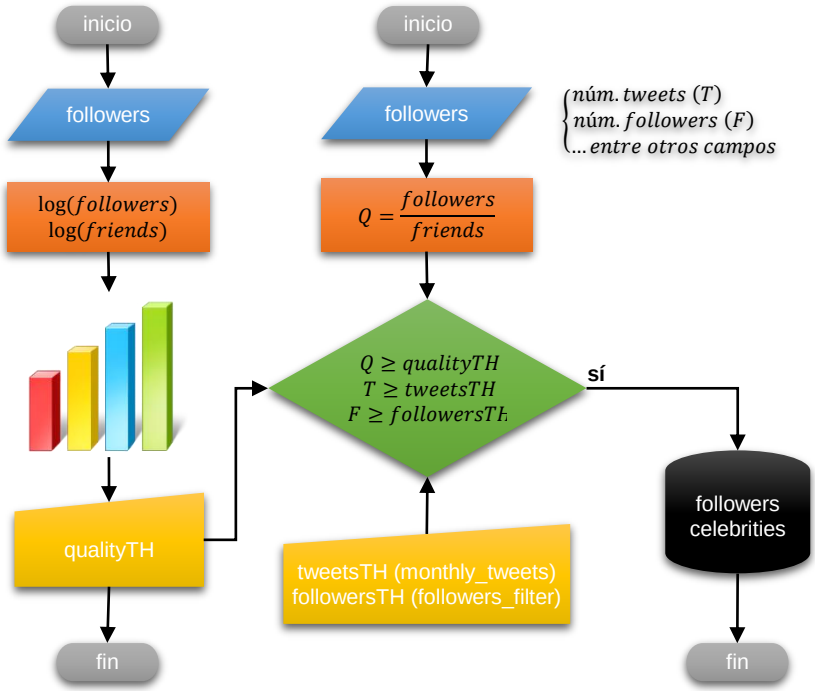


Figura 3.6: Proceso en la detección de los followers celebrity.

3. Análisis de celebrities

En *step3_celebrities.py* se crean las funciones necesarias para tratar los diferentes temas propuestos en el punto 3. A partir de los *celebrities* detectados en el paso anterior, se extraen diferentes estadísticas como la densidad de los mismos por marca, la actividad de estos en Twitter y el número de menciones a las marcas. También muestra los *celebrities* más afines a la marca teniendo en cuenta el número de menciones de estos. Y finalmente, mediante el **algoritmo k-means implementado** en el punto anterior, se segmentan estos seguidores según la cantidad de *friends*, *followers* y actividad, con el objetivo de encontrar diferentes formas de relación entre marca y *celebrity*.

4. Topics

En el último punto se implementa la función `followers_topics` en *step4_topics.py*, la cual recorre todos los tuits de los *celebrities* "tokenizando" los *hashtags* y menciones en los textos mediante la librería NLTK de Python. Posteriormente se continúa con el procesamiento del lenguaje natural eliminando los *stopwords* desde la función `tokens_cleaner` y con la misma librería de NLTK se extraen las frecuencias[22] de cada una de las palabras por usuario. Seguidamente la función `term_weights` calcula los **tf-idf** y a partir de estos se crean los datos finales. La figura siguiente muestra esta secuencia:

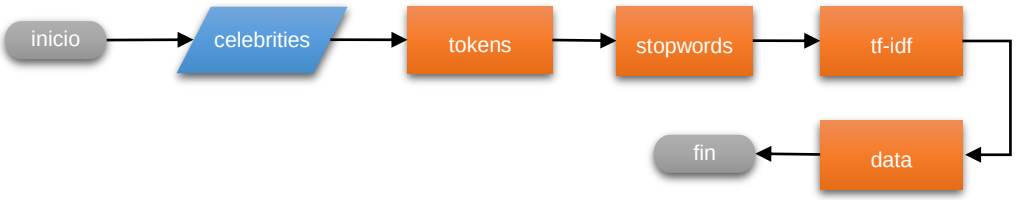


Figura 3.7: Proceso en la extracción de topics.

[22] Número de repeticiones de cada palabra.

Una vez obtenidos los datos, se presenta un histograma con las frecuencias de los *topics* más usados con los que poder extraer conclusiones sobre los temas más relevantes en las dos marcas.

Posteriormente se segmentan los *celebrities* según algunos de estos temas más relevantes, mediante el **algoritmo k-means implementado**, con el objetivo de encontrar relaciones con las que agrupar *topics* y conocer aquellos seguidores relevantes que pueden promocionar la marca en distintos ámbitos.

Finalmente, con el objetivo de reforzar la búsqueda de temas de interés en las cuentas, sin la necesidad de analizar todos los tuits generados por todos los *followers*, se crea un histograma con el total de cuentas registradas por fecha presentando ambas marcas en el mismo gráfico para un mejor análisis.

4 Resultados experimentales

En este capítulo se presentan los resultados experimentales de aplicar la propuesta desarrollada en el capítulo anterior, así como el análisis realizado a partir de los mismos resultados.

4.1 Experimento 1: Filtro de followers

Antes de filtrar los seguidores que se consideran irrelevantes, es interesante representar el total de *followers* de ambas cuentas de forma visual y constatar así el número de seguidores que comparten. Para ello servirá el siguiente diagrama de Venn:

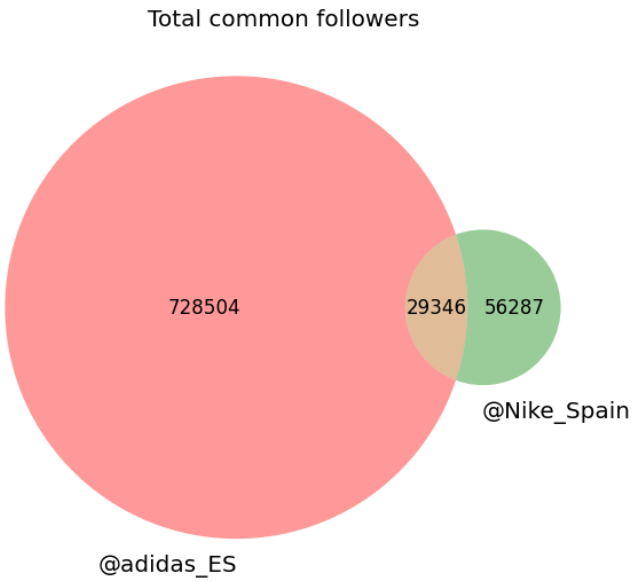


Figura 4.1: Diagrama de Venn sobre el número followers en adidas y Nike.

Este primer análisis sugiere un uso del medio Twitter muy superior por parte de adidas ya que la marca alemana dispone del orden de 13 veces más seguidores, aunque este dato puede ser debido a multitud de factores diferentes a estudiar. Además prácticamente la mitad de los seguidores de Nike son *followers* de adidas, esto último no es algo extraño puesto que la comparativa corresponde a dos marcas muy semejantes. Aclarar como la cuenta de Nike es 9 meses anterior a la de adidas por lo que esta enorme diferencia no es debido a la antigüedad en Twitter.

A continuación se realiza un primer filtrado mediante el parámetro `lang_filter=es`. En la tabla siguiente se puede observar el resultado con el total de seguidores antes y después de aplicar el filtro, y el porcentaje de *followers* que se mantienen.

Brand	Followers	Followers after filter	Followers (%)
adidas	727223	376938	51.83
Nike	56283	38254	67.96

Tabla 4.1: Followers después del filtro idioma.

Seguidamente se reúnen los resultados de aplicar el filtro con los parámetros `followers_filter=10` y `monthly_tweets=4` en la siguiente tabla y se añade el alcance potencial del total de cada una de las cuentas:

Brand	Followers	Followers after filter	Followers (%)	Potential reach
adidas	727223	182639	25.11	245 millones
Nike	56283	23832	42.34	71 millones

Tabla 4.2: Followers después del filtro de seguidores y actividad.

Después de aplicar los filtros de idioma, actividad y seguidores mínimos representados en las tablas 4.1 y 4.2, se observa como existe un volumen enorme de *followers* irrelevantes desde el punto de vista del mercado de las cuentas, donde prácticamente la mitad de los *followers* son extranjeros y una cuarta parte son seguidores inactivos o intrascendentes.

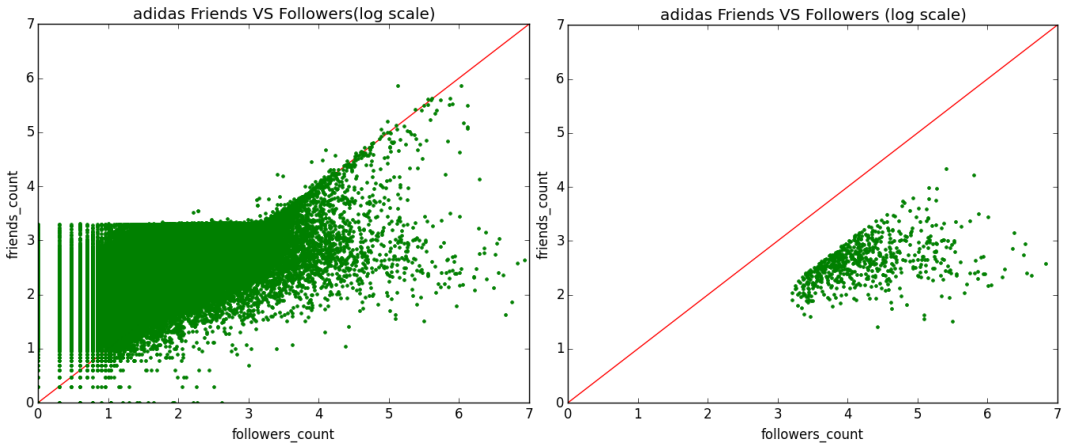
Se puede concluir señalando que el gran volumen de seguidores irrelevantes detectado, mostrado en las tablas anteriores, puede ser debido al carácter internacional de las dos marcas. Esto puede ser positivo desde el punto de vista de relevancia mundial, pero también tiene una lectura negativa desde el punto de vista de la tasa de rebote, ya que muchas de las publicaciones no causarán ningún impacto en gran parte de las cuentas. También se puede constatar cómo aunque el porcentaje de usuarios irrelevantes es mayor en adidas, el alcance potencial de la cuenta alemana es muy superior.

4.2 Experimento 2: Detección de celebrities

El objetivo del experimento 2 es detectar aquellos seguidores más relevantes, así pues tal y como se ha comentado en el capítulo 3, se representan los seguidores mediante el gráfico *friendsVSfollowers* en escala logarítmica.

Del gráfico 4.2.a se consideran seguidores *celebrities* aquellos que quedan en la zona inferior derecha, es decir usuarios con un gran número de *followers* y a su vez pocos *friends*. También se puede observar el gran número de seguidores poco influyentes en la zona superior izquierda, y en general todos los que están situados por encima de línea roja, puesto que no disponen de prácticamente *followers* y en cambio siguen a muchos otros. Aquellos *followers* situados en la esquina superior derecha y cercanos a la línea roja, usuarios con una relación *followers/friends* muy cercana a 1, se pueden considerar seguidores que han realizado acciones *followback* muy agresivas.

Y en el gráfico 4.2.b se muestra el resultado de extraer a los usuarios *celebrities*, considerados como aquellos seguidores que disponen de más de 3,2 *followers* (escala logarítmica), que han publicado una media de 20 tuits al mes y que a su vez disponen de un ratio *followers/friends* mayor de 10.



(a) (b)

Figura 4.2: (a) Representación friendsVSfollowers (escala logarítmica) de los seguidores de adidas. (b) Detección de los seguidores celebrities.

Ver el total de resultados (*celebrities* Nike) en el Anexo B.

4.3 Experimento 3: Análisis de celebrities

Una vez disponemos de los *followers celebrities*, seguidores que generan un mayor impacto en el mercado, se procede a extraer diversas estadísticas entre este tipo de usuarios. En este sentido, la tabla 4.3 muestra la cantidad y densidad de *celebrities* en ambas cuentas, la media de seguidores que a su vez estos disponen, la actividad media que realizan y finalmente las menciones que generan a la marca y la media de estas.

Se puede observar cómo aunque Nike dispone de una mayor densidad de *celebrity*, adidas dispone de muchos más y estos también son algo más activos y un poco más mediáticos desde el punto de vista de los *followers* que les siguen. Además los *celebrities* de adidas mencionan con una mayor frecuencia a la marca, así que podría ser este uno de los factores que hacen que la marca alemana disponga de esa cantidad superior de seguidores vista en el primer experimento.

Brand	Celebrities (%)	Av. activity	Av. followers	Mentions (Av.)
adidas	650 (0.35)	13402	125894	503 (0.77)
Nike	260 (1.09)	11787	114301	164 (0.63)

Tabla 4.3: Estadísticas sobre los celebrities de adidas y Nike.

El siguiente gráfico muestra el número de menciones por mes. Se pueden detectar un par de picos en los meses de noviembre y diciembre, donde además de ser fechas vacacionales coincide para el caso de Nike, con la carrera de San Silvestre de Vallecas (patrocinada por la marca americana) y para el caso de adidas, con un nuevo lanzamiento de botas de fútbol y un par de eventos, uno en Madrid y otro la ciudad condal con los jugadores del FC. Barcelona.

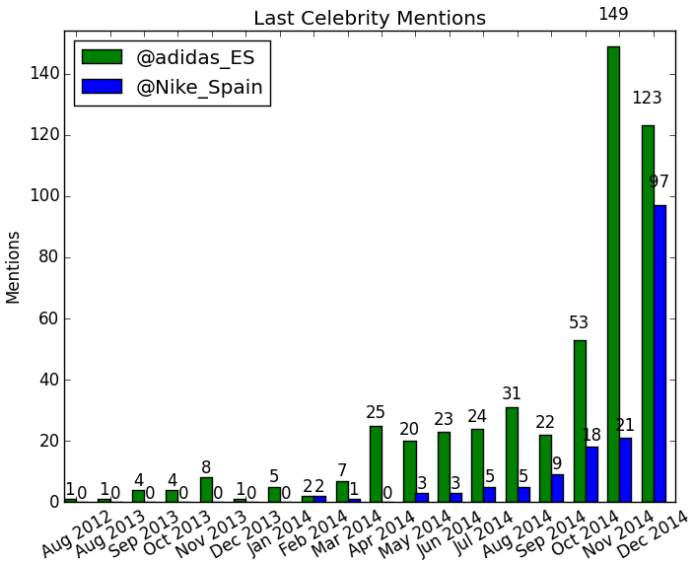


Figura 4.3: Historial de menciones de los celebrities.

A continuación se muestran los *celebrities* más afines a las marcas, desde el punto de vista de las menciones realizadas por estos, con la finalidad de que ambas compañías puedan conocerlos y plantearse nuevas relaciones con ellos, además de conocer también a los de la competencia. Del top10 listado, se puede observar como los corredores son los que realizan más menciones en general a su marca, aunque también aparece algún futbolista conocido.

Top10 adidas celebrities	Top10 Nike celebrities
Luís A. Hernando (TrailHernando) / Mentions: 57 Rafael Iglesias (RafaelIglesiasB) / Mentions: 31 Sergio Rodriguez (SergioRodriguez) / Mentions: 17 Koke Resurrección (Koke6) / Mentions: 17 vanessa veiga (vaneveiga79) / Mentions: 16 Races Trail Running (RTrailrunning) / Mentios: 15 Carles Castillejo (C_Castillejo) / Mentions: 15 dARIO bARRIO (dariobarrio_db) / Mentions: 13 Álvaro Arbeloa (aarbeloa17) / Mentions: 13 Mariam hernandez (Mariam_Hernan) / Mentions: 12	Joan Pedrero (JoanPedrero) / Mentions: 36 Mónica Martínez (Monicamtinez) / Mentions: 15 Universia España (Universia) / Mentions: 9 Kevin López Yerga (kevinlopezyerga) / Mentions: 8 Ruben Rochina Naixes (rubenrochina23) / Mentions: 7 EVASION RUNNING (evasionrunning) / Mentions: 6 Luis Alberto Marco (Marco800) / Mentions: 6 El Corte Inglés ÉL (elcorteinglésel) / Mentions: 5 Iñaki Cano Martínez (ICano14) / Mentions: 5 Run Academy (runacademy) / Mentions: 4

Tabla 4.4: Top10 celebrities por marca según número de menciones.

A modo de ejemplo sobre contratos/relaciones ente marcas y *celebrities* se presenta el caso del corredor Luís Alberto para adidas, el cual a través del *hashtag* #aquesitioostraigo publicita material en cada salida. Y el caso de la periodista Mónica Martínez para la marca americana, la cual ha recibido material para publicitar Nike en la reciente carrera San Silvestre en Vallecas.

(a) (b)

Figura 4.4: (a) Publicidad adidas por parte del corredor Luis Alberto. (b) Publicidad Nike por parte de la periodista Mónica Martínez.

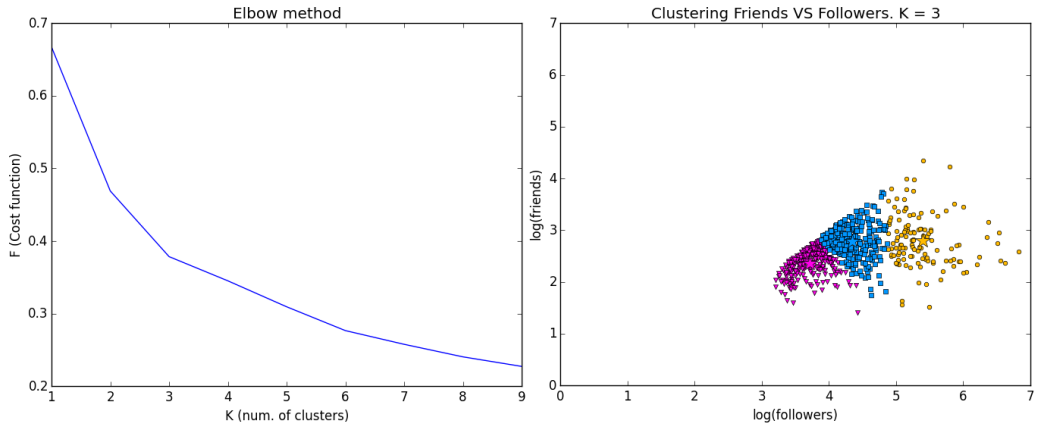
A continuación se lista el top10 de seguidores *celebrities* más mediáticos y que a su vez las marcas todavía no siguen, con el objetivo de que estos sean personajes potenciales a contratar. En el listado de adidas se observan cantantes, actores y medios de comunicación, todos ajenos al deporte, lo que lleva a pensar que la marca

alemana ya sigue a sus *assets*, en cambio la marca americana parece no realiza esta práctica puesto que encontramos varios futbolistas Nike.

Top10 adidas potential friends	Top10 Nike potential friends
Pablo Alborán (pabloalboran) / Followers: 2381744 mario (Mario_casas_) / Followers: 1601593 El Mundo en Imágenes (NaturPictures) / Followers: 1026243 **Rafa Mora Garcia (RAFAMORATETE) / Followers: 735787** Abel Aguilar (abelaguilart) / Followers: 724527 ABC.es (abc_es) / Followers: 643239 **ELLE España (elle_es) / Followers: 561232** Mundo al Revés (Paralelito) / Followers: 424536 Sara (Sarinha_3) **/** Followers: 404148 Futbol_Mercado (Futbol_Mercado) / Followers: 347079	ISCO ALARCON (isco_alarcon) / Followers: 2461473 **Pablo Alborán (pabloalboran) / Followers: 2381744** Marc Bartra (MarcBartra) / Followers: 1521032 Carlos Latre (Carlos_Latre) / Followers: 832290 **Rafa Mora Garcia (RAFAMORATETE) / Followers: 735787** Iker Muniain (IkerMuniain19) / Followers: 726303 **ELLE España (elle_es) / Followers: 561232** Mario Suárez (MarioSuarez4) / Followers: 489679 OTROGUAPOSUELTO (OTROGUAPOSUELTO) / Followers: 406939 Borja Valero (bvalero20) / Followers: 254589

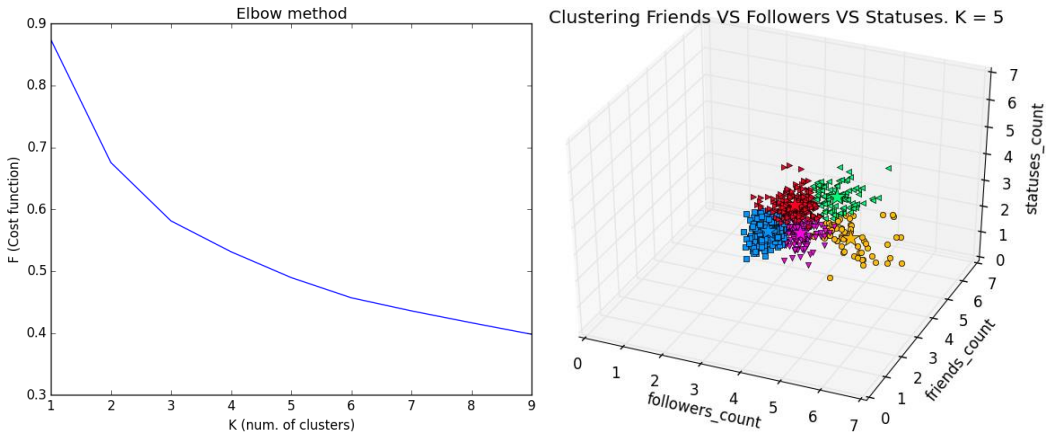
Tabla 4.5: Friends potenciales para adidas y Nike.

Se prosigue con la segmentación de *celebrities* mediante el algoritmo k-means implementado, primero según el gráfico *friendsVSfollowers* y posteriormente añadiendo una tercera variable actividad `statuses_count`. Las figuras siguientes muestran el resultado de aplicar el algoritmo k-means y el método Elbow para el caso de adidas (ver el resto de gráficos en el Anexo C).



(a) (b)

Figura 4.5: Clustering de los celebrities de adidas friendsVSfollowers mediante el método Elbow (a) y el algoritmo k-means (b).



(a) (b)

Figura 4.6: Clustering de los celebrities de adidas friendsVSfollowersVSstatuses mediante el método Elbow (a) y el algoritmo k-means (b).

Como se puede observar en las dos figuras anteriores, el resultando en el número de clústeres es un tanto ambiguo ya que no parece haber un número óptimo claro de categorías con las que tratar los *celebrities*, entendiendo el trato como el tipo de contrato (sueldo, tipo relación, material, etc.). Aunque una posible opción, a modo de ejemplo, sería la de generar 3 categorías:

1. Seguidores muy famosos con sueldo elevado.
2. Seguidores menos famosos con un sueldo bajo.
3. Seguidores con producto, conocidos como embajadores, los cuales solamente reciben género.

Aunque lógicamente cuantas más categorías se crearan mejor sería la relación *celebrity*-marca puesto que estarían ofreciendo un trato más personalizado.

4.4 Experimento 4: Topics

Primero se presenta un análisis sobre las palabras más importantes para cada una de las marcas, importancia calculada a partir de la media entre los pesos tf-idf y la cantidad de *celebrities* de cada una. Los gráficos creados a tal efecto muestran las

30 palabras más relevantes para cada una de las dos marcas (ver Anexo D) y de forma conjunta para aquellas palabras comunes en ambas cuentas (ver figura 4.7).

Del análisis de estos primero gráficos se observa como el tema por excelencia en ambas cuentas es el fútbol, aparecen una gran cantidad de futbolistas, equipos y periódicos deportivos relacionados. También se observa una clara predisposición a temas relacionados con el Real Madrid para la cuenta de adidas, lógico puesto que es al equipo que patrocina. En cambio los términos en los que Nike se diferencia más de la marca alemana no están relacionados con el FC Barcelona, si no con temas totalmente ajenos al fútbol como por ejemplo: *running*, *selfie* o *navidad*.

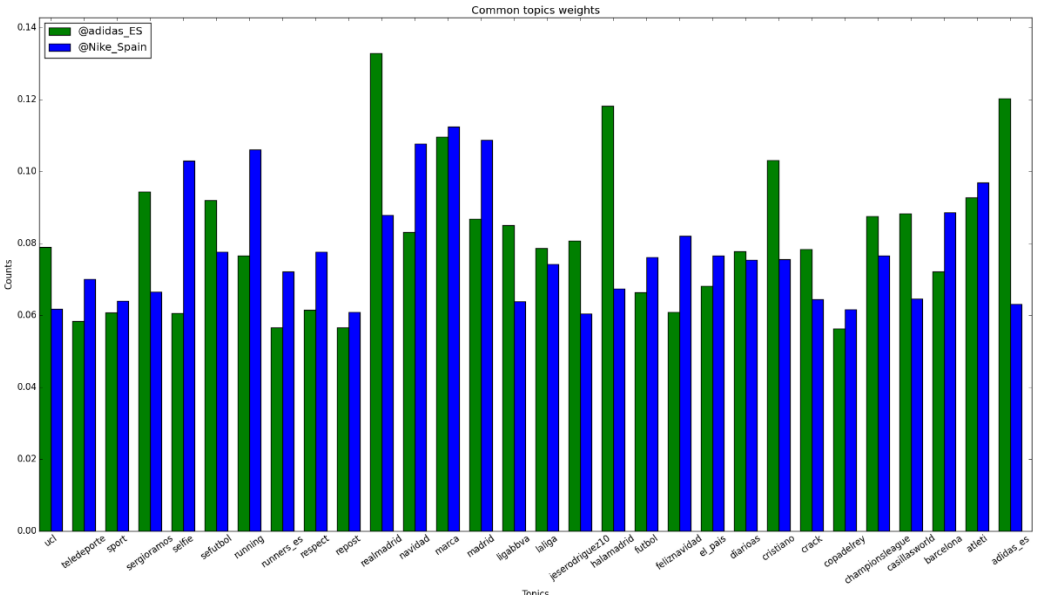
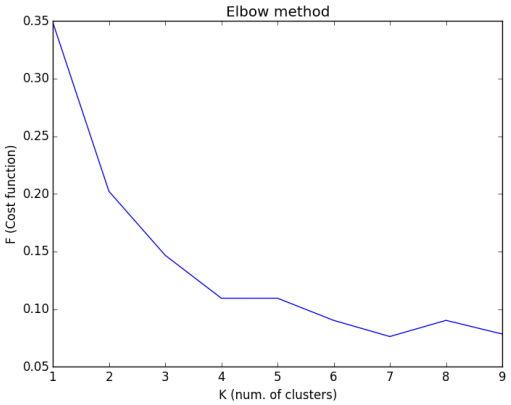


Figura 4.7: Top30 términos comunes más frecuentes.

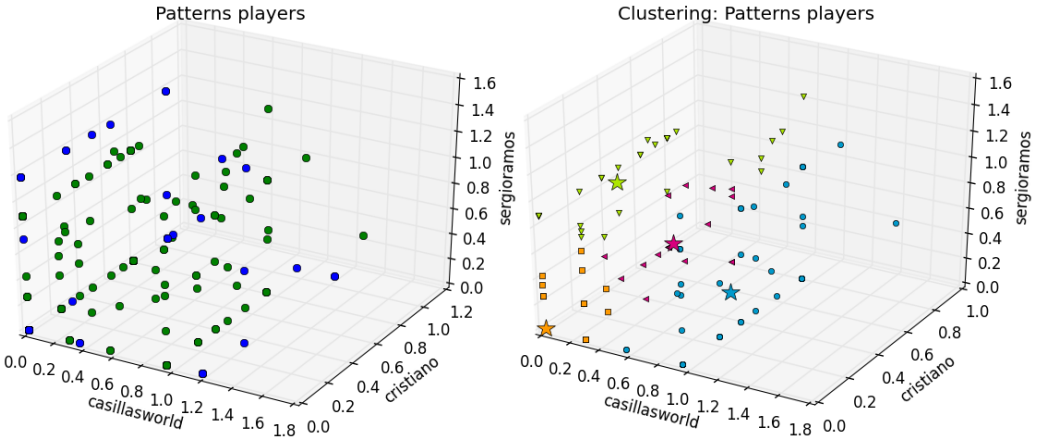
A continuación se procede a la representación de los *celebrities* a partir de los *topics* más relevantes y aplicando k-means con el método Elbow con la finalidad de encontrar la relación entre *topics* y *celebrities* comentada en el capítulo 3.

La figura 4.8 muestra la relación entre términos vinculados con jugadores de fútbol del Real Madrid. Se puede constatar como hay muchos más seguidores por parte de adidas (verde) relacionados con los tres jugadores ya que comentan sobre

todos en general, en cambio el hecho de que los seguidores de Nike (azul) aparezcan mucho más cercanos a los ejes puede indicar un interés menor por el Real Madrid y sus jugadores, generando así únicamente comentarios puntuales.



(a)



(b) (c)

Figura 4.8: (a) Método Elbow con solución K=4 (b) Muestras del conjunto de datos (c) Datos segmentados mediante k-means tras el análisis del método Elbow.

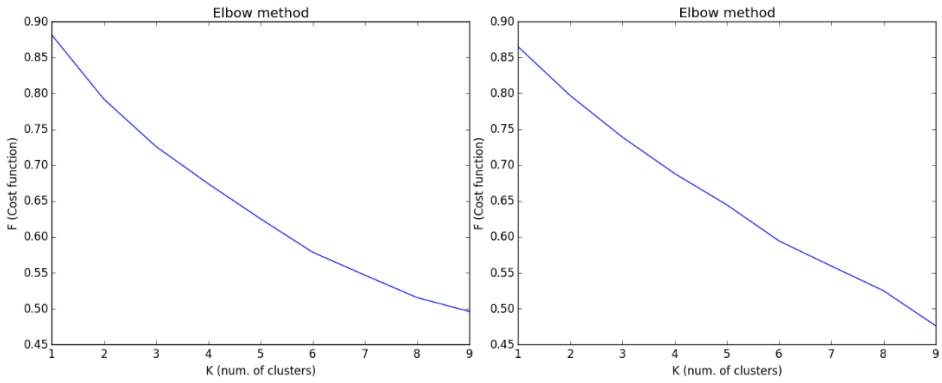
La relación entre estos tres jugadores podría segmentarse en 4 grupos distintos:

1. Celebrities con poco interés sobre estos jugadores (amarillo).
2. Celebrities muy afines a casillas pero que también comentan sobre "cristiano" y "sergioramos" (azul claro).

3. Celebrities muy afines a “sergioramos” pero que también comentan sobre “cristiano” (verde claro).
4. Celebrities interesados en los tres jugadores (magenta).

Ver más segmentaciones de *topics* relacionados en el Anexo D.

Seguidamente se segmentan todos los *celebrities*, por marca, a partir de los 10 *topics* más frecuentes, aplicando el algoritmo PCA con la finalidad de reducir las dimensiones y el método Elbow con el objetivo de encontrar un número de grupos óptimo con los que clusterizar los *celebrities*. Aunque se consigue una mínima reducción en el número de dimensiones con el 90% de la varianza, se puede observar cómo no se alcanza a obtener un número de grupos claro (ver figura 4.9), es decir, no se logra relacionar los *celebrities* a partir de los 10 términos que más usan.



(a) (b)

Figura 4.9: Elbow tras la segmentación según topics para adidas (a) y Nike (b).

Finalmente se presenta el histograma con las fechas en las que los seguidores han sido creados. Con el objetivo de hallar gustos e intereses deportivos en los *followers*, se pretende encontrar eventos que pudieran suscitar la creación de las cuentas. En este sentido, después de analizar las fechas en las que el histograma presenta los picos más abruptos, algunos de los eventos más destacados son:

- **Octubre 2012:** Red Bull Stratos. Salto en caída libre desde la estratosfera.
- **Noviembre 2012:** Sebastian Vettel, tricampeón del mundial de Fórmula 1.
- **Agosto 2013:** Campeonato Mundial de Natación en Barcelona, Mireia Belmonte gana 2 platas y un bronce. Y la selección femenina de Waterpolo campeona.
- **Junio y julio 2014:** Copa Mundial de la FIFA en Brasil.

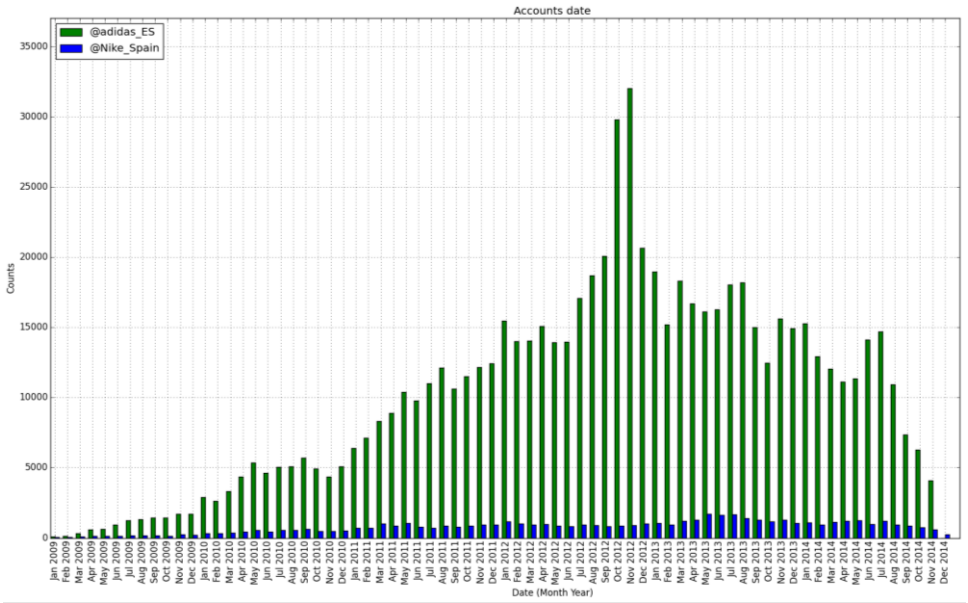


Figura 4.10: Histograma con las fechas de creación de cuentas.

Por lo que se puede concluir como el estar presente en eventos deportivos mundiales, aprovechando la gran actividad que estos generan, puede ser una acción de interés a tomar por parte de las marcas de estudio.

5 Conclusiones

En este proyecto se ha marcado el objetivo de estudiar el *social media analytics* realizando diferentes analíticas para la extracción de patrones, así como segmentando los seguidores mediante la utilización de técnicas *Machine Learning* y extrayendo tendencias a partir de métodos de *Natural Languaje Processing*. A continuación se analizan en detalle las diferentes partes del proyecto:

Respeto al estudio previo, se ha finalizado el curso [1] sobre Python con una muy buena cualificación, aprendiendo así las bases sobre el lenguaje de programación, el cual formaba una pequeña parte dentro de los objetivos. Relacionado con Python también ha sido muy interesante el aprendizaje de la herramienta Notebook a través de las video-lecturas del curso [2]. También se han reforzado conocimientos referentes a ML y NLP mediante los cursos [9] y [12] respectivamente.

En cuanto a la implementación, se ha desarrollo el algoritmos k-means con el que segmentar los datos de estudio, el método Elbow con el que hallar el número de *clusters* óptimo, el algoritmo PCA para reducir las variables de las muestras de entrada, y realizando técnicas de procesamiento del lenguaje como la tf-idf. Con todo ello se ha conseguido mejorar las competencias adquiridas en la etapa de estudio previo a partir de la puesta en práctica de los anteriores conocimientos adquiridos. El desarrollo del trabajo también ha servido para conocer las posibilidades de la analítica en Twitter y el potencial de la API.

En la fase experimental se han conseguido resultados interesantes como la detección y clasificación de los seguidores más relevantes e influyentes, la

extracción de los temas tuiteados más frecuentes y la gran cantidad de seguidores *fake* desde el punto de vista del mercado objetivo de ambas cuentas.

Por otro lado también han surgido problemas que afrontar, el gran número de seguidores en ambas cuentas (sobretodo adidas con más de 700.000 *followers*) y el total de información que estos representan, han dilatado muchísimo los tiempos en las descargas de datos, debido principalmente a dos factores: por un lado las limitaciones de la API en el número de peticiones permitidas, y por otro el proceso *open-save-close* de ficheros durante la descarga de datos, donde estos han llegado a pesar más de 1,5 Gigabytes al intentar obtener los últimos 200 tuits de todos los seguidores, finalmente limitado a únicamente seguidores *celebrity*.

Por lo tanto, se han cumplido los objetivos marcados aunque todavía hay mucho margen de mejora y unas posibilidades enormes en materia de *analytics*, propuestas para un trabajo futuro.

5.1 Trabajo futuro

Relacionado con el problema de los amplios tiempos necesarios en la descarga de la información y el peso de los ficheros por el gran volumen de datos, una posible solución sería crear una cantidad importante de cuentas con las que realizar consultas a Twitter de forma paralela y también dividir/organizar los ficheros donde se almacenan los datos para acceder más ágilmente a la información, todo ello implementando técnicas **Big Bata** con las que reducir estos altos tiempos.

Una vez solucionado el tema de los dilatados tiempos, una futura tarea interesante sería la **creación de una aplicación** con la que poder realizar *analytics* sobre cualquier cuenta y ofrecer los resultados de la analítica en la misma interfaz.

Por otro lado, se ha constatado como el número de usuarios que utilizan la **geolocalización en sus tuits** todavía es muy bajo, por lo que el analizar las cuentas en particular a partir de este criterio se convierte en algo no representativo. De todos

modos es interesante conocer este factor ya que en un futuro próximo el contexto actual podría variar y la analítica sobre este elemento podría ganar importancia.

En este sentido, debido al bajo número de seguidores que geolocalizan sus tuits junto con el uso erróneo por parte de los usuarios en relación con el campo "Ubicación" en el perfil de cuenta, ha sido un problema **localizar a los seguidores**, un dato que hubiera abierto cuantitativamente el abanico de posibilidades en *analytics*.

Otro trabajo futuro interesante sería explotar más técnicas de NLP, por ejemplo investigando sobre el **análisis de sentimiento**, aunque este es un tema complejo debido a la gramática del lenguaje castellano, la ironía con la que nos comunicamos y las abreviaciones con las que escribimos, más todavía en Twitter debido al reducido número de caracteres que admite un tuit. Y relacionado con esto, poder extraer **información demográfica** sobre los seguidores.

Finalmente, aunque una parte negativa sobre la API de Twitter ha sido lo comentado sobre sus limitaciones en el número de peticiones, Twitter ofrece otra API destinada a la transmisión en vivo. Así pues, un trabajo futuro sería la exploración de esta API y las posibilidades que esta ofrece en *analytics* al poder monitorizar los **tuis en tiempo real**.

Bibliografía

[1] Joe Warren, Scott Rixner, Johm Greiner and Stephen Wong, *An Introduction to Interactive*, Fundamentals of Computing, Rice University. [Curso en línea]. Coursera. https://www.coursera.org/

[2] UAVideoTube, Pybonacci. (2014, March 17). *Curso Python para científicos e ingenieros* [video]. Recuperado de http://youtu.be/ox09Jko1ErM

[3] Mark Lutz, *Learning Python, 5th Edition*, O'Reilly Media, June 2013.

[4] Fernando Pérez and Brian E. Granger, *IPython: A System for Interactive Scientific Computing*, Computing in Science & Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, DOI:10.1109/MCSE.2007.53, http://ipython.org/

[5] Jones E, Oliphant E, Peterson P, et al. *SciPy: Open Source Scientific Tools for Python*, 2001, http://www.scipy.org/

[6] Hunter, J. D. *Matplotlib: A 2D Graphics Environment*, Computing In Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007, DOI:10.5281/ZENODO.12400, http://matplotlib.org/

[7] Twitter Developers [Web en línea]. https://dev.twitter.com/

[8] Guy Kawasaki and Peg Fitzpatrick, *The Art of Social Media: Power Tips for Power Users*, Penguin Books, December 2014.

[9] Andrew Ng, *Machine Learning*, Stanford University. [Curso en línea]. Coursera. https://www.coursera.org/

[10] Peter Harrington, *Machine Learning in Action*, Manning Publications, April 2012.

[11] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., 1999.

[12] Dan Jurafsky and Christopher Manning, *Natural Language Processing*, Stanford University. [Curso en línea]. Coursera. https://www.coursera.org/

[13] Bird, Steven, Edward Loper and Ewan Klein, *Natural Language Processing with Python,* O'Reilly Media Inc., 2009.

[14] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux, *The NumPy Array: A Structure for Efficient Numerical Computation*, Computing in Science & Engineering, 13, pp. 22-30, 2011, DOI:10.1109/MCSE.2011.37, http://www.numpy.org/

[15] Ryan McGrath, *Twython*, https://twython.readthedocs.org

[16] Wikipedia, *Twitter*, http://es.wikipedia.org/wiki/Twitter

Anexo A

Resultados del experimento 2

Brand	Followers	F.ollowers after filter	Followers (%)
adidas	727223	376938	51.83
Nike	56283	38254	67.96

Tabla A.1: Followers después del filtro idioma.

Brand	Followers	F.ollowers after filter	Followers (%)
adidas	727223	212292	29.19
Nike	56283	26742	47.51

Tabla A.2: Followers después del filtro idioma y monthly_tweets=2.

Brand	Followers	Followers after filter	Followers (%)
adidas	727223	182639	25.11
Nike	56283	23832	42.34

Tabla A.3: Followers después del filtro idioma y monthly_tweets=4.

Brand	Followers	F.ollowers after filter	Followers (%)
adidas	727223	152115	20.91
Nike	56283	20551	36.51

Tabla A.4: Followers después del filtro idioma y monthly_tweets=8.

Brand	Followers	F.ollowers after filter	Followers (%)
adidas	727223	121810	16.75
Nike	56283	16941	30.09

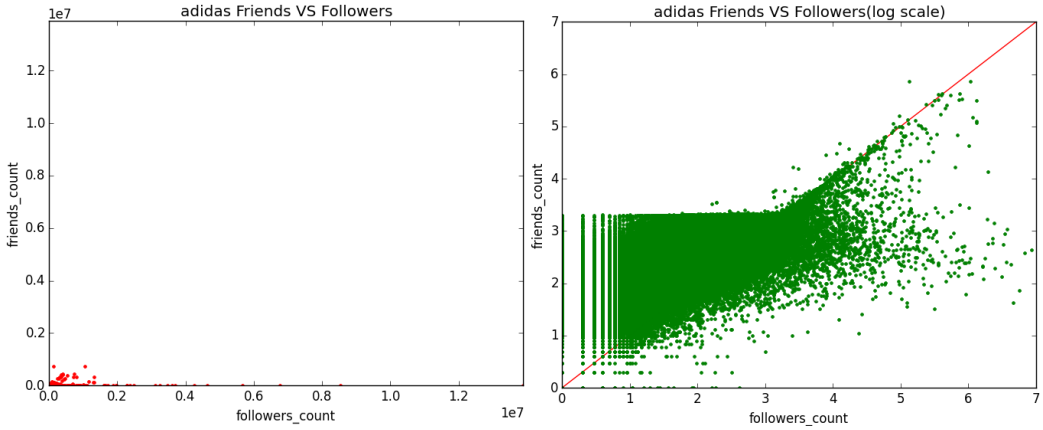
Tabla A.5: Followers después del filtro idioma y monthly_tweets=16.

Brand	Followers	F.ollowers after filter	Followers (%)
adidas	727223	92145	12.67
Nike	56283	13149	23.36

Tabla A.6: Followers después del filtro idioma y monthly_tweets=32.

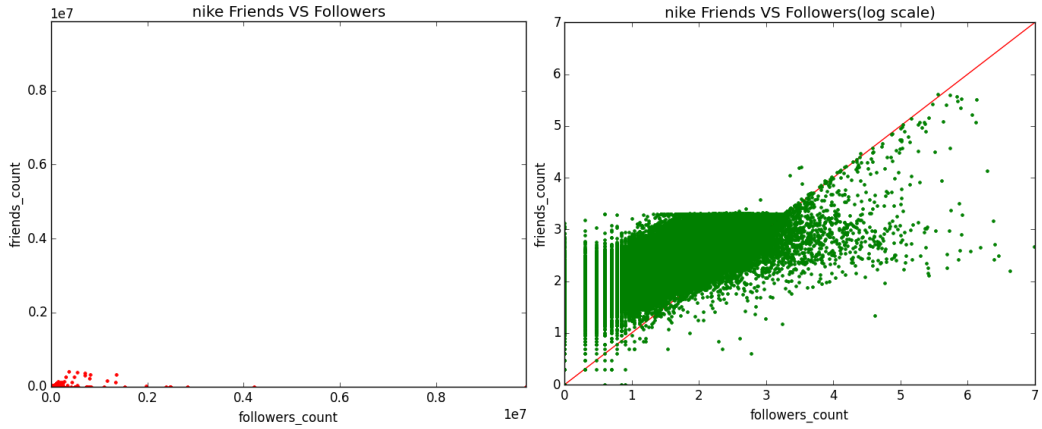
Anexo B

Resultados del experimento 3



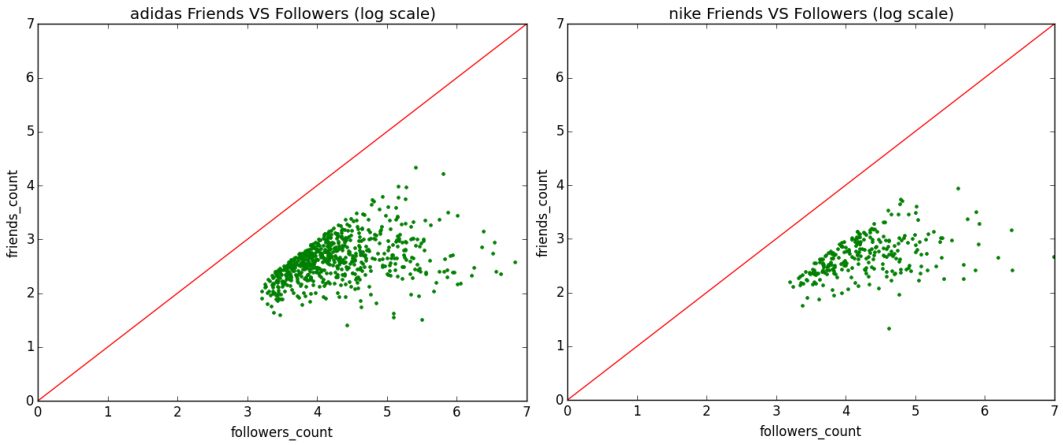
(a) (b)

Figura B.1: Representación de los followers de adidas friendsVSfollowers (a) y en escala logarítmica (b).



(a) (b)

Figura B.2: Representación de los followers de Nike friendsVSfollowers (a) y en escala logarítmica (b).

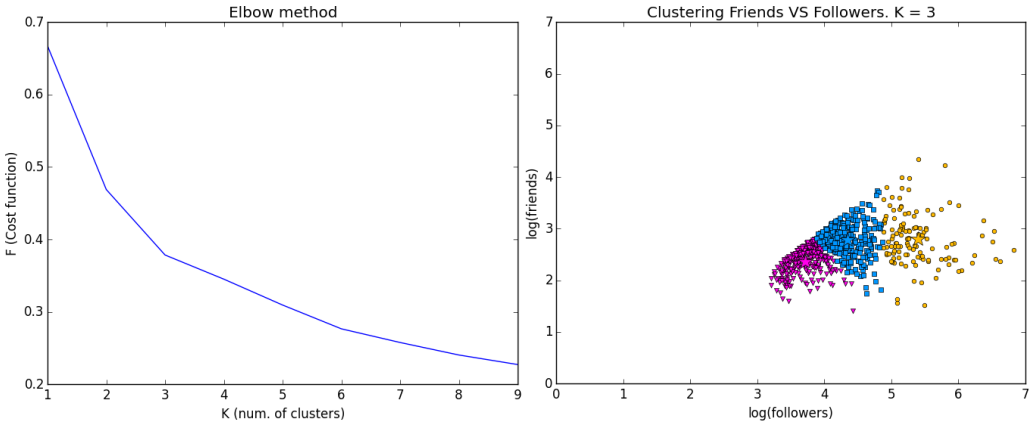


(a) (b)

Figura B.3: (a) Celebrities de adidas (b) Celebrities de Nike.

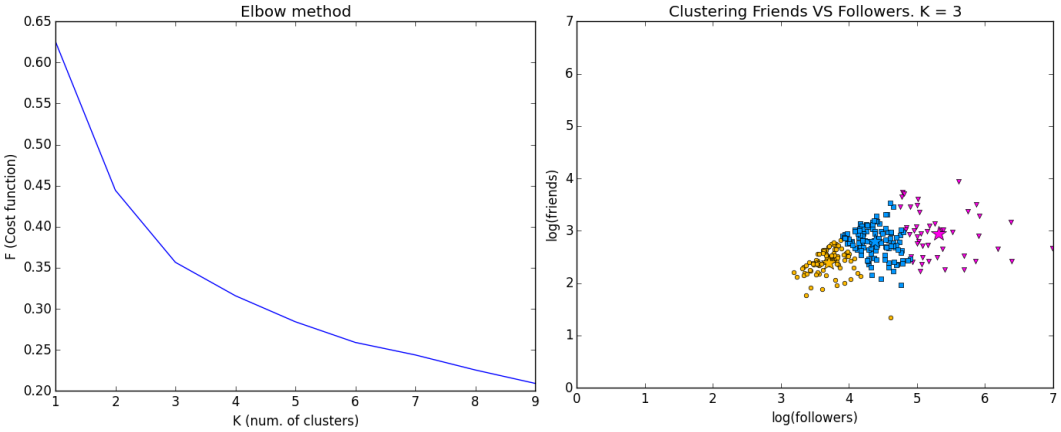
Anexo C

Resultados del experimento 4



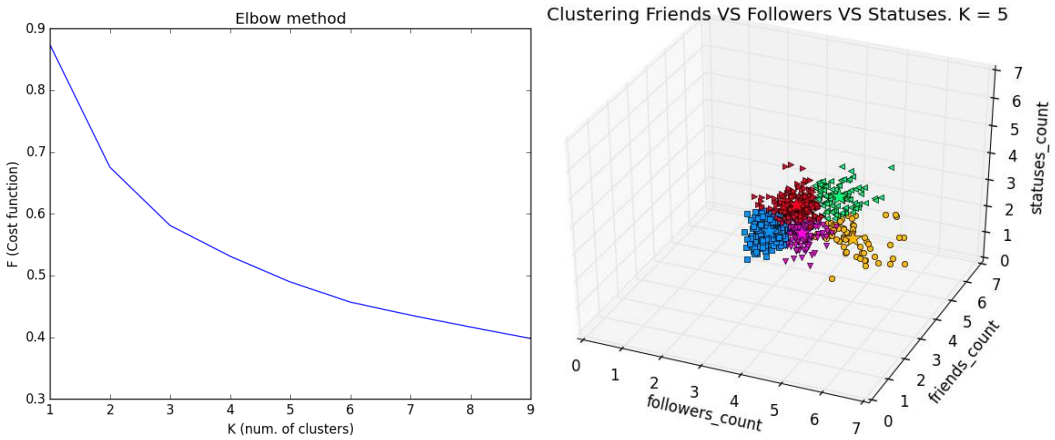
(a) (b)

Figura C.1: Clustering de los celebrities de adidas friendsVSfollowers mediante el método Elbow (a) y el algoritmo k-means (b).



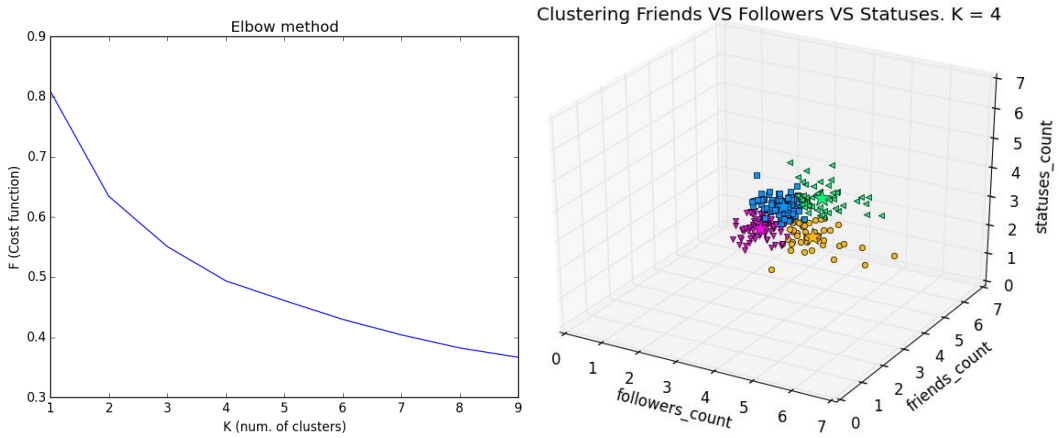
(a) (b)

Figura C.2: Clustering de los celebrities de Nike friendsVSfollowers mediante el método Elbow (a) y el algoritmo k-means (b).



(a) (b)

Figura C.3: Clustering de los celebrities de adidas friendsVSfollowersVSstatuses mediante el método Elbow (a) y el algoritmo k-means (b).



(a) (b)

Figura C.4: Clustering de los celebrities de Nike friendsVSfollowersVSstatuses mediante el método Elbow (a) y el algoritmo k-means (b).

Anexo D

Resultados del experimento 5

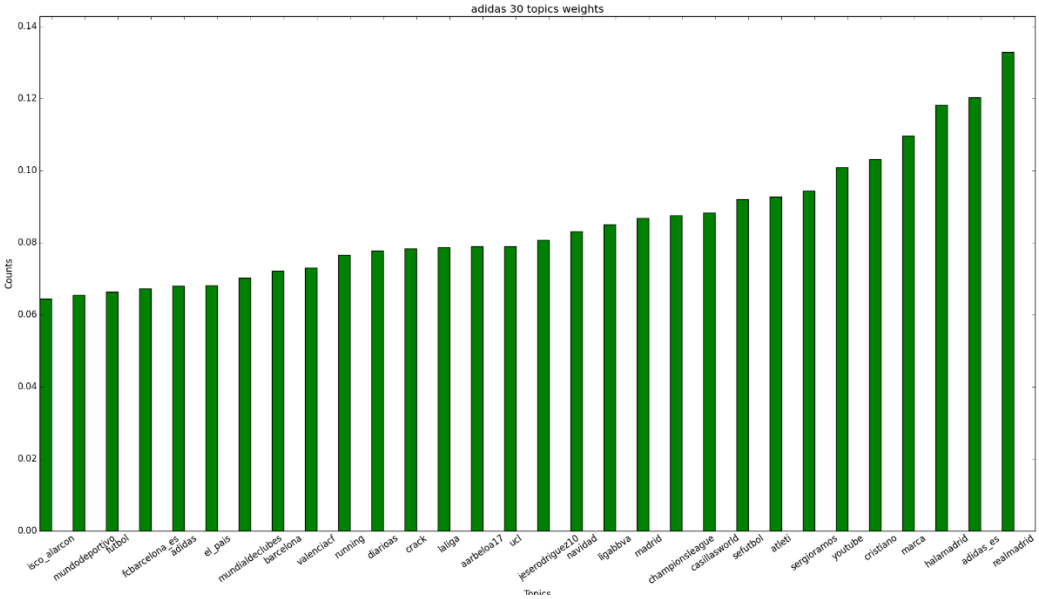


Figura D.1: Top30 términos más frecuentes para adidas.

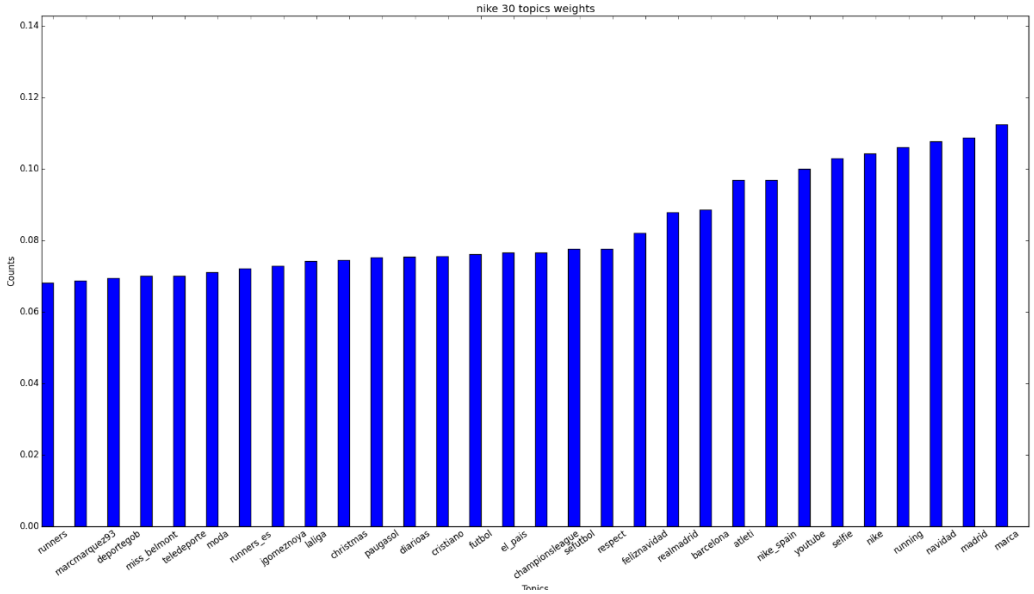


Figura D.2: Top30 términos más frecuentes para Nike.

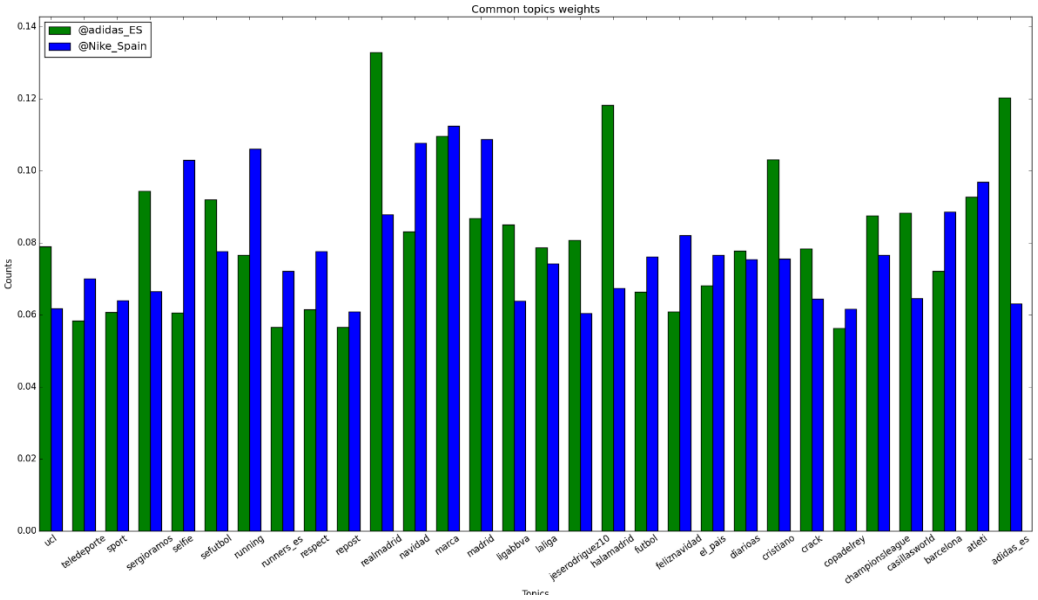
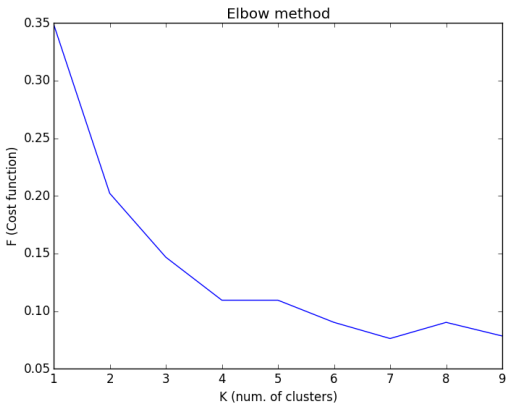
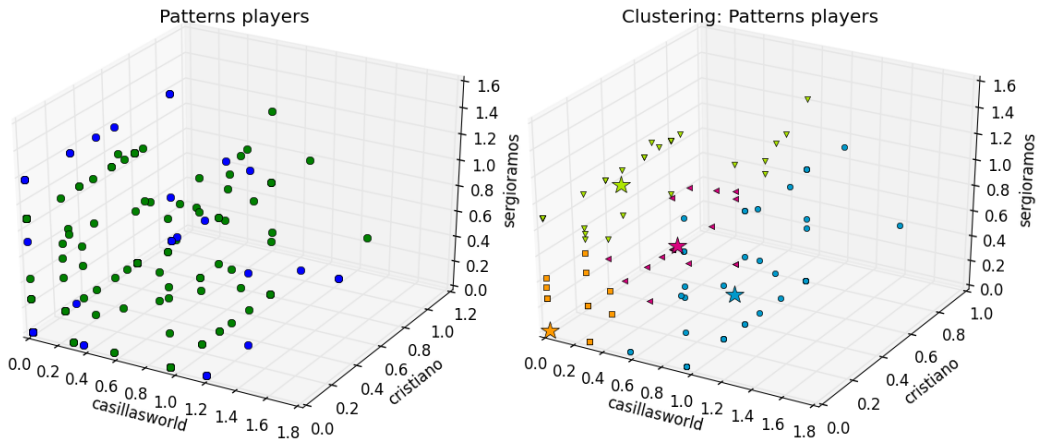


Figura D.3: Top30 términos más frecuentes comunes.

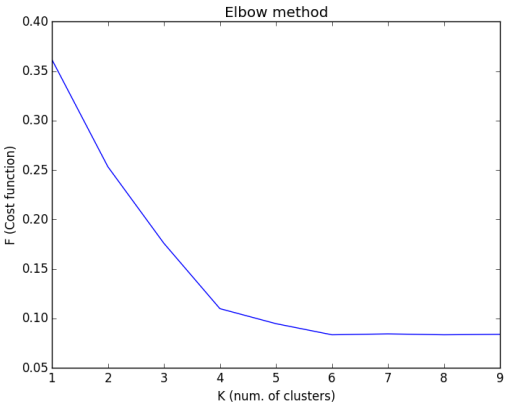


(a)

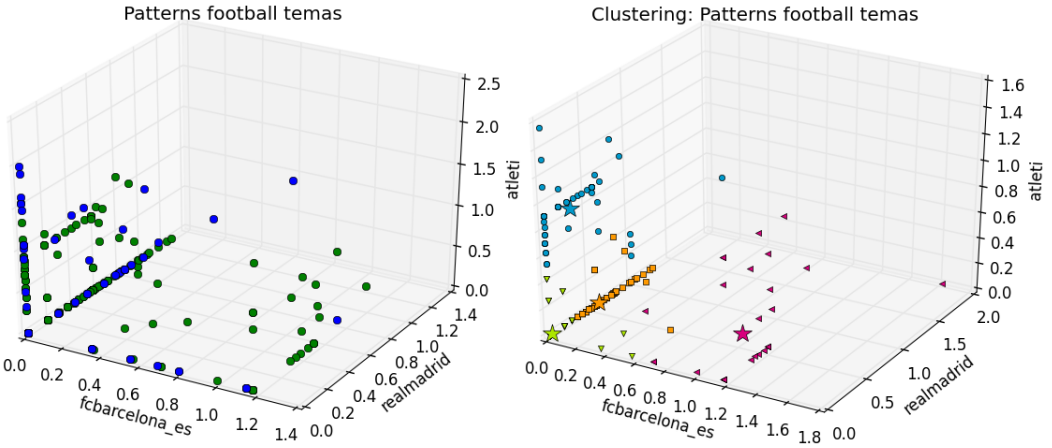


(b) (c)

Figura D.4: (a) Elbow (b) Topics sobre jugadores (c) Datos segmentados.

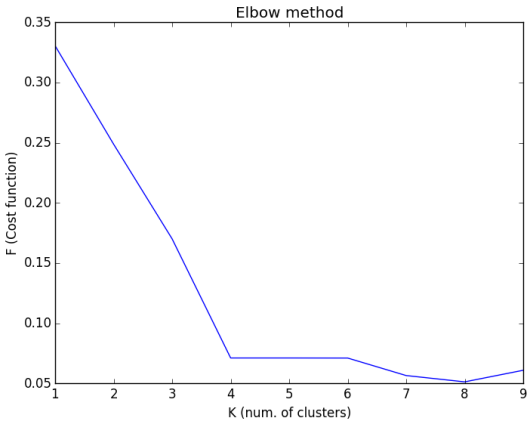


(a)

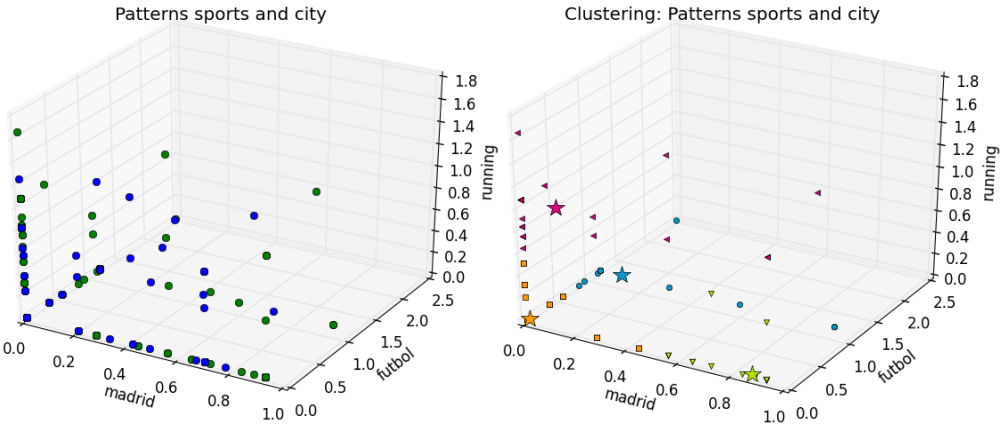


(b) (c)

Figura D.5: (a) Elbow (b) Topics sobre equipos de fútbol (c) Datos segmentados.

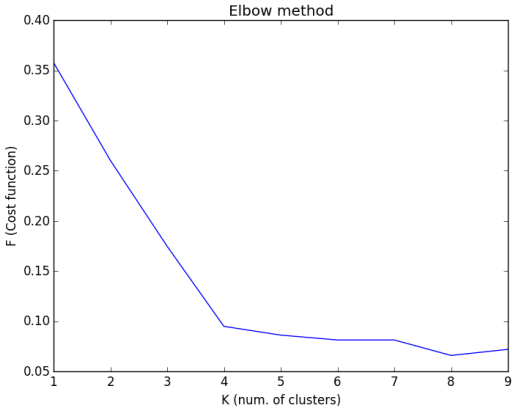


(a)

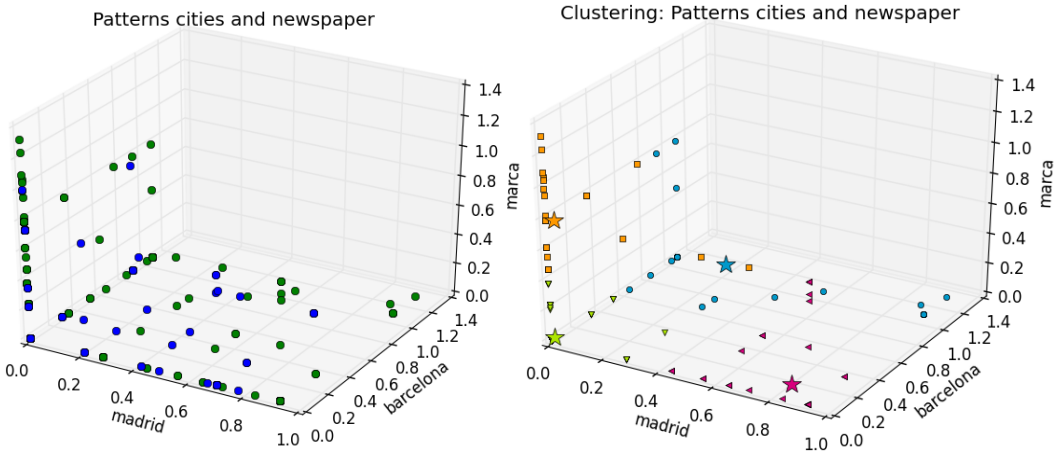


(b) (c)

Figura D.6: (a) Elbow (b) Topics sobre deportes y ciudad (c) Datos segmentados.



(a)



(b) (c)

Figura D.7: (a) Elbow (b) Topics sobre ciudades y peródico (c) Datos segmentados.

Anexo E

Código Python fichero main.py

```
# -*- coding: utf-8 -*-

"""
MACHINE LEARNING. CLUSTERING TWITTER DATA
"""
# Librerías utilizadas
from __future__ import division
from tabulate import tabulate
from matplotlib_venn import venn2
from datetime import datetime
import time
import numpy as np
import matplotlib.pyplot as plt

# Funciones desarrolladas
import ml_functions
import data_admin
import step1_filter
import step2_detection
import step3_celebrities
import step4_topics

# Variables globales
STEPS = [1] #options = [0, 1, 2, 3, 4]

# Funciónes del main
def print_text(text, size, top = True, bottom = True, t_brake = True, b_brake = False):
if t_brake:
print ""
if size > 0 and top:
print "-" * size
print text
if size > 0 and bottom:
print "-" * size
if b_brake:
print ""
# Función MAIN
def main(steps):
if 0 in steps:
# STEP 0: DOWNLOAD DATA FROM ADIDAS AND NIKE TWITTER ACCOUNTS ##########
#exemple adidas data
#adidas_friends_ids = data_admin.friends_ids("@adidas_ES")
#adidas_followers_ids = data_admin.followers_ids("@adidas_ES")
#data_admin.save_data(adidas_followers_ids, "adidas_users_ids")
#adidas_data = data_admin.twitter_user_data("adidas_users_ids", "adidas_data")
#data_admin.twitter_user_status("adidas_data", "adidas_status_data")
#exemple nike data
#nike_friends_ids = data_admin.friends_ids("@Nike_Spain")
#nike_followers_ids = data_admin.followers_ids("@Nike_Spain")
#data_admin.save_data(nike_followers_ids, "nike_users_ids")
#nike_data = data_admin.twitter_user_data("nike_users_ids", "nike_data")
#data_admin.twitter_user_status("nike_data", "nike_status_data")
pass
if 1 in steps:
# STEP 1: FILTRO DE USUARIOS IRRELEVANTES ##############################
start_time = time.time()
#cargamos datos adidas y nike
adidas_ids = data_admin.load_data("adidas_followers_ids")
adidas_data = data_admin.load_data("adidas_data")
nike_ids = data_admin.load_data("nike_followers_ids")
nike_data = data_admin.load_data("nike_data")

#filtro por idioma
adidas_followers = step1_filter.filter_account(followers = adidas_data, lang_filter = "es",
monthly_tweets = 0, followers_filter = 0)
adidas_quality = len(adidas_followers) / len(adidas_data)
nike_followers = step1_filter.filter_account(followers = nike_data, lang_filter = "es",
monthly_tweets = 0, followers_filter = 0)
nike_quality = len(nike_followers) / len(nike_data)
#tabla de estadísticas con filtro idioma
info_adidas = ["adidas", len(adidas_data), len(adidas_followers), adidas_quality * 100]
info_nike = ["nike", len(nike_data), len(nike_followers), nike_quality * 100]
headers = ["Brand", "Followers", "Filt. followers", "Quality (%)"]
print_text("FILTER FOLLOWER ACCOUNTS BY LANG", 60)
print tabulate([info_adidas, info_nike], headers, "orgtbl")
#filtro completo
```

```
adidas_followers = step1_filter.filter_account(followers = adidas_data, lang_filter = "es",
monthly_tweets = 4, followers_filter = 10)
nike_followers = step1_filter.filter_account(followers = nike_data, lang_filter = "es",
monthly_tweets = 4, followers_filter = 10)
adidas_quality = len(adidas_followers) / len(adidas_data)
nike_quality = len(nike_followers) / len(nike_data)
#tabla de estadísticas con filtro completo
adidas_pr = step1_filter.potential_reach(adidas_data)
nike_pr = step1_filter.potential_reach(nike_data)
info_adidas = ["adidas", len(adidas_data), len(adidas_followers), adidas_quality * 100,
adidas_pr]
info_nike = ["nike", len(nike_data), len(nike_followers), nike_quality * 100, nike_pr]
headers = ["Brand", "Followers", "Filt. followers", "Quality (%)", "Potential reach"]
print_text("COMPLETE FILTER FOLLOWER ACCOUNTS", 60)
print tabulate([info_adidas, info_nike], headers, "orgtbl")
#diagrama de venn completo
intersection = set(adidas_ids).intersection(set(nike_ids))
adidas_venn = len(adidas_ids)
nike_venn = len(nike_ids)
inter_venn = len(intersection)
plt.figure(10)
ax = plt.gca()
ax.set_title("Total common followers")
venn2(subsets={"10": adidas_venn, "01": nike_venn, "11": inter_venn}, set_labels =
("@adidas_ES", "@Nike_Spain"))
#diagrama de venn filtrado
afids = [af["id"] for af in adidas_followers]
nfids = [nf["id"] for nf in nike_followers]
intersection = set(afids).intersection(set(nfids))
adidas_venn = len(adidas_followers)
nike_venn = len(nike_followers)
inter_venn = len(intersection)
plt.figure(11)
ax = plt.gca()
ax.set_title("Common followers after filter")
venn2(subsets={"10": adidas_venn, "01": nike_venn, "11": inter_venn}, set_labels =
("@adidas_ES", "@Nike_Spain"))
print_text("Total Step 1 execution time: " + str(time.time() - start_time) + " seconds", 60)
if 2 in steps:
# STEP 2: CELEBRITY DETECTION ##########################################
start_time = time.time()
#cargamos datos adidas y nike
#adidas_data = data_admin.load_data("adidas_data")
#nike_data = data_admin.load_data("nike_data")
keys = ["followers_count", "friends_count"]
step2_detection.cluster_followers(adidas_data, keys, title = "adidas", method = False, fig =
[31, 32])
step2_detection.cluster_followers(nike_data, keys, title = "nike", method = False, fig = [33,
34])
#extraemos celebrities, calculamos sus tweets y los guardamos
#require step1
adidas_celebrities = step1_filter.filter_account(followers = adidas_data, monthly_tweets = 20,
followers_filter = 1584, quality_filter = 10)
nike_celebrities = step1_filter.filter_account(followers = nike_data, monthly_tweets = 20,
followers_filter = 1584, quality_filter = 10)
#data_admin.save_data(adidas_celebrities, "adidas_celebrities")
#data_admin.save_data(nike_celebrities, "nike_celebrities")
#data_admin.twitter_user_status("adidas_celebrities", "adidas_celebrities_status")
#data_admin.twitter_user_status("nike_celebrities", "nike_celebrities_status")
print_text("Total Step 2 execution time: " + str(time.time() - start_time) + " seconds", 60)
if 3 in steps:
# STEP 3: ANÁLISIS DE LOS CELEBRITIES ##################################
start_time = time.time()
#require step1
#cargamos datos adidas y nike
adidas_celebrities = data_admin.load_data("adidas_celebrities_status")
nike_celebrities = data_admin.load_data("nike_celebrities_status")
adidas_stats, adidas_timeline, adidas_top10 =
step3_celebrities.celebrity_statistics(adidas_celebrities, "@adidas_ES")
nike_stats, nike_timeline, nike_top10 =
step3_celebrities.celebrity_statistics(nike_celebrities, "@Nike_Spain")
#tabla de estadísticas
adidas_celebrities_density = str(round(len(adidas_celebrities) / len(adidas_followers) * 100,
2))
nike_celebrities_density = str(round(len(nike_celebrities) / len(nike_followers) * 100, 2))
aux = 200 * len(adidas_celebrities)
adidas_m = str(round(adidas_stats[2] / aux * 100, 2))
aux = 200 * len(nike_celebrities)
nike_m = str(round(nike_stats[2] / aux * 100, 2))
info_adidas = ["adidas", str(len(adidas_celebrities)) + " (" + adidas_celebrities_density +
")", str(adidas_stats[0]), str(adidas_stats[1]), str(adidas_stats[2]) + " (" + adidas_m + ")"]
info_nike = ["nike", str(len(nike_celebrities)) + " (" + nike_celebrities_density + ")",
str(nike_stats[0]), str(nike_stats[1]), str(nike_stats[2]) + " (" + nike_m + ")"]
headers = ["Brand", "Celebrities (%)", "Av. activity", "Av. followers", "Mentions (%)"]
print_text("CELEBRITIES STATISTICS", 60)
print tabulate([info_adidas, info_nike], headers, "orgtbl")
print "* Mentions in last 200 tweets"
#top10 de celebrities que más mencionan la marca
```

```
print_text("TOP10 ADIDAS CELEBRITIES", 60)
print "\n".join([top10a["celebrity"] + " / Followers: " + str(top10a["followers"]) + " /
Mentions: " + str(top10a["mentions"]) for top10a in adidas_top10 if top10a["mentions"] > 0])
print_text("TOP10 NIKE CELEBRITIES", 60)
print "\n".join([top10n["celebrity"] + " / Followers: " + str(top10n["followers"]) + " /
Mentions: " + str(top10n["mentions"]) for top10n in nike_top10 if top10n["mentions"] > 0])
print "* order by mentions"
#lista de potenciales friends
adidas_friends_ids = data_admin.load_data("adidas_friends_ids")
adidas_potential_friends = [celebrity for celebrity in adidas_celebrities if celebrity["id"]
not in adidas_friends_ids]
adidas_potential_friends = sorted(adidas_potential_friends, key = lambda k:
k["followers_count"], reverse = True)[0:10]
print_text("TOP10 ADIDAS POTENTIAL FRIENDS", 60)
print "\n".join([apf["name"] + " (" + apf["screen_name"] + ") / Followers: " +
str(apf["followers_count"]) for apf in adidas_potential_friends])
nike_friends_ids = data_admin.load_data("nike_friends_ids")
nike_potential_friends = [celebrity for celebrity in nike_celebrities if celebrity["id"] not
in nike_friends_ids]
nike_potential_friends = sorted(nike_potential_friends, key = lambda k: k["followers_count"],
reverse = True)[0:10]
print_text("TOP10 NIKE POTENTIAL FRIENDS", 60)
print "\n".join([npf["name"] + " (" + npf["screen_name"] + ") / Followers: " +
str(npf["followers_count"]) for npf in nike_potential_friends])
print_text("COMMON POTENTIAL FRIENDS", 60)
intersection = [apf for apf in adidas_potential_friends if apf["id"] in [npf["id"] for npf in
nike_potential_friends]]
print "\n".join([inter["name"] + " (" + inter["screen_name"] + ") / Followers: " +
str(inter["followers_count"]) for inter in intersection])
#diagrama de venn
adidas_venn = len(adidas_potential_friends)
nike_venn = len(nike_potential_friends)
inter_venn = len(intersection)
plt.figure(40)
ax = plt.gca()
ax.set_title("Common potential friends")
venn2(subsets={"10": adidas_venn, "01": nike_venn, "11": inter_venn}, set_labels =
("@adidas_ES", "@Nike_Spain"))
#gráfico timeline
plt.figure(41)
ax = plt.gca()
width = 0.35
xticks = tuple(set(adidas_timeline[1]) | set(nike_timeline[1]))
xticks = tuple(sorted(xticks, key = lambda dates:datetime.strptime(dates,"%b %Y")))
x = np.arange(len(xticks))
dif_timeline = tuple(set(xticks) - set(adidas_timeline[1]))
adidas_timeline = adidas_timeline[0] + [(t, 0) for t in dif_timeline]
adidas_timeline = sorted(adidas_timeline, key = lambda dates: datetime.strptime(dates[0],"%b
%Y"))
adidas_values = tuple(value[1] for value in adidas_timeline)
dif_timeline = tuple(set(xticks) - set(nike_timeline[1]))
nike_timeline = nike_timeline[0] + [(t, 0) for t in dif_timeline]
nike_timeline = sorted(nike_timeline, key = lambda dates: datetime.strptime(dates[0],"%b %Y"))
nike_values = tuple(value[1] for value in nike_timeline)
adidas_bars = ax.bar(x, adidas_values, width, color = "g")
nike_bars = ax.bar(x + width, nike_values, width, color = "b")
ax.set_title("Last Celebrity Mentions")
ax.set_xlabel("Date (Month Year)")
ax.set_ylabel("Mentions")
ax.set_xticks(x + width)
ax.set_xticklabels(xticks, rotation = 30)
ax.legend((adidas_bars[0], nike_bars[0]), ("@adidas_ES", "@Nike_Spain"), loc = 2)
def autolabel(bars):
for bar in bars:
height = bar.get_height()
ax.text(bar.get_x() + bar.get_width() / 2., 1.05 * height, "%d" % int(height), ha = "center",
va = "bottom")
autolabel(adidas_bars)
autolabel(nike_bars)
plt.ylim((0, max(adidas_values + nike_values) + 5))
plt.show()
#cluster adidas and nike celebrities
keys = ["followers_count", "friends_count"]
step2_detection.cluster_followers(adidas_celebrities, keys, title = "adidas", method =
"elbow", fig = [0, 42, 43])
step2_detection.cluster_followers(nike_celebrities, keys, title = "nike", method = "elbow",
fig = [0, 44, 45])
keys = ["followers_count", "friends_count", "statuses_count"]
step2_detection.cluster_followers(adidas_celebrities, keys, title = "adidas", method =
"elbow", fig = [0, 46, 47])
step2_detection.cluster_followers(nike_celebrities, keys, title = "nike", method = "elbow",
fig = [0, 48, 49])
print_text("Total Step 3 execution time: " + str((time.time() - start_time) / 60) + "
minutes", 59)
if 4 in steps:
# STEP 4: CELEBRITIES TOPICS ##############################################
start_time = time.time()
```

```
#cargamos datos adidas y nike, extraemos topics y calculamos pesos
adidas_celebrities = data_admin.load_data("adidas_celebrities_status")
nike_celebrities = data_admin.load_data("nike_celebrities_status")
adidas_X, adidas_topics, adidas_weights = step4_topics.followers_topics(adidas_celebrities,
interval = (-10, None))
nike_X, nike_topics, nike_weights = step4_topics.followers_topics(nike_celebrities, interval =
(-10, None))
#topics comunes
common_topics = set(adidas_topics).intersection(set(nike_topics))
print_text("COMMON TOPICS", 60)
print "\n".join(common_topics)
#diagrama de venn
n_topics = len(adidas_topics)
n_common_topics = len(common_topics)
plt.figure(50)
ax = plt.gca()
ax.set_title("Common topics")
v = venn2(subsets={"10": n_topics, "01": n_topics, "11": n_common_topics}, set_labels =
("@adidas_ES", "@Nike_Spain"))
v.get_patch_by_id("10").set_color("b")
v.get_patch_by_id("11").set_color("g")
v.get_patch_by_id("01").set_color("c")
#gráficos de frecuencias
adidas_freqs = [atw[1] for atw in adidas_weights]
nike_freqs = [ntw[1] for ntw in nike_weights]
common_topics = sorted(common_topics, reverse = True)
adidas_weights = sorted(adidas_weights, key = lambda k: k[0], reverse = True)
nike_weights = sorted(nike_weights, key = lambda k: k[0], reverse = True)
adidas_common_freqs = [atw[1] for atw in adidas_weights if atw[0] in common_topics]
nike_common_freqs = [ntw[1] for ntw in nike_weights if ntw[0] in common_topics]
maxfreq = max(adidas_freqs + nike_freqs)
maxcomfreq = max(adidas_common_freqs + nike_common_freqs)
step4_topics.plot_topics_bars(freqs = adidas_freqs, xticks = adidas_topics, ymax = maxfreq,
title = "adidas 30 topics weights", color = "g", fig = 51)
step4_topics.plot_topics_bars(freqs = nike_freqs, xticks = nike_topics, ymax = maxfreq, title
= "nike 30 topics weights", color = "b", fig = 52)
step4_topics.plot_topics_bars(freqs = [adidas_common_freqs, nike_common_freqs], xticks =
common_topics, ymax = maxcomfreq, title = "Common topics weights", fig = 53)

#buscamos patrones entre temas distintos (de los últimos 70 terms)
#casillasworld, cristiano, sergioramos
X1 = np.array([adidas_X[:,60], adidas_X[:,65], adidas_X[:,63]])
X2 = np.array([nike_X[:,36], nike_X[:,52], nike_X[:,37]])
step4_topics.celebrities_topics(X1, X2, ["Patterns players", "casillasworld", "cristiano",
"sergioramos"])
#fcbarcelona_es, realmadrid, atleti
X1 = np.array([adidas_X[:,43], adidas_X[:,69], adidas_X[:,62]])
X2 = np.array([nike_X[:,10], nike_X[:,59], nike_X[:,61]])
step4_topics.celebrities_topics(X1, X2, ["Patterns football temas", "fcbarcelona_es",
"realmadrid", "atleti"])
#madrid, barcelona, marca
X1 = np.array([adidas_X[:,58], adidas_X[:,47], adidas_X[:,66]])
X2 = np.array([nike_X[:,68], nike_X[:,60], nike_X[:,69]])
step4_topics.celebrities_topics(X1, X2, ["Patterns cities and newspaper", "madrid",
"barcelona", "marca"])
#madrid, futbol, running
X1 = np.array([adidas_X[:,58], adidas_X[:,42], adidas_X[:,49]])
X2 = np.array([nike_X[:,68], nike_X[:,53], nike_X[:,66]])
step4_topics.celebrities_topics(X1, X2, ["Patterns sports and city", "madrid", "futbol",
"running"])
#marca, diarioas, sport
#X1 = np.array([adidas_X[:,66], adidas_X[:,50], adidas_X[:,32]])
#X2 = np.array([nike_X[:,69], nike_X[:,51], nike_X[:,33]])
#futbol, running, nba
#X1 = np.array([adidas_X[:,42], adidas_X[:,49], adidas_X[:,17]])
#X2 = np.array([nike_X[:,53], nike_X[:,66], nike_X[:,24]])
#valencia, madrid, barcelona
#X1 = np.array([adidas_X[:,37], adidas_X[:,58], adidas_X[:,47]])
#X2 = np.array([nike_X[:,20], nike_X[:,68], nike_X[:,60]])

#segmentamos celebrities según topics
adidas_Z = ml_functions.pca(adidas_X, .90)
nike_Z = ml_functions.pca(nike_X, .90)
adidas_Z = adidas_Z.T
nike_Z = nike_Z.T
print np.shape(adidas_Z)
print np.shape(nike_Z)
adidas_centroids, adidas_clusters, adidas_K = ml_functions.kmeans_elbow(adidas_Z, 10, 40)
nike_centroids, nike_clusters, nike_K = ml_functions.kmeans_elbow(nike_Z, 10, 40)
#gráfico timeline
adidas_data = data_admin.load_data("adidas_data")
nike_data = data_admin.load_data("nike_data")
adidas_timeline = step4_topics.get_follower_timeline(adidas_data)
nike_timeline = step4_topics.get_follower_timeline(nike_data)
plt.figure(20)
ax = plt.gca()
width = 0.35
adidas_timeline_dates = [adidas_t[0] for adidas_t in adidas_timeline]
```

```
nike_timeline_dates = [nike_t[0] for nike_t in nike_timeline]
xticks = tuple(set(adidas_timeline_dates) | set(nike_timeline_dates))
xticks = tuple(sorted(xticks, key = lambda dates:datetime.strptime(dates,"%b %Y")))
x = np.arange(len(xticks))
dif_timeline = tuple(set(xticks) - set(adidas_timeline_dates))
adidas_timeline = adidas_timeline + [(t, 0) for t in dif_timeline]
adidas_timeline = sorted(adidas_timeline, key = lambda dates: datetime.strptime(dates[0],"%b
%Y"))
adidas_values = tuple(value[1] for value in adidas_timeline)
dif_timeline = tuple(set(xticks) - set(nike_timeline_dates))
nike_timeline = nike_timeline + [(t, 0) for t in dif_timeline]
nike_timeline = sorted(nike_timeline, key = lambda dates: datetime.strptime(dates[0],"%b %Y"))
nike_values = tuple(value[1] for value in nike_timeline)
adidas_bars = ax.bar(x, adidas_values, width, color = "g")
nike_bars = ax.bar(x + width, nike_values, width, color = "b")
ax.set_title("Accounts date")
ax.set_xlabel("Date (Month Year)")
ax.set_ylabel("Counts")
ax.set_xticks(x + width)
ax.set_xticklabels(xticks, rotation = 90)
ax.legend((adidas_bars[0], nike_bars[0]), ("@adidas_ES", "@Nike_Spain"), loc = 2)
plt.ylim((0, max(adidas_values + nike_values) + 5000))
plt.grid()
plt.show()

print_text("Total Step 4 execution time: " + str(time.time() - start_time) + " seconds", 60)
if __name__ == "__main__":
main(STEPS)
```

Código Python fichero data_admin.py

```
# -*- coding: utf-8 -*-

"""
FUNCIONES DE CARGA Y GUARDADO DE DATOS

Conjunto de funciones que mediante la API de Twitter extraen los datos y se encargan de
guardarlos en ficheros para el posterior procesado:

* save_data: guarda datos en json
* load_data: carga datos desde json
* shape_data: extrae las variables deseadas de los datos del json.
* friends_ids: carga los ids de los friends.
* followers_ids: carga los ids de los followers.
* twitter_user_data: carga los datos de perfil de cada seguidor.
* twitter_user_status: carga los topics de cada usuario.
"""

# Librerías
import json
import time
import numpy as np
from blist import blist
from twython import Twython

# Variables globales
APP_KEY = "dpyWQCt9Yq2VfrU515NCVc5ml"
APP_SECRET = "HPodBdA9pzKHnfroV7bo99oFgIkwM7ugvClYYn2YE537FK4fUe"
twitter = Twython(APP_KEY, APP_SECRET, oauth_version=2)
ACCESS_TOKEN = twitter.obtain_access_token()
twitter = Twython(APP_KEY, access_token=ACCESS_TOKEN)
DIR_PATH = "T:/Documentos/4 - Universidad/PFC3/Experimento/dbfiles/"

# Guardar datos en json ##############################################################
def save_data(data, file_name):
"""
Guardar datos json de 'data' en el fichero 'file_name'.
"""
#crea/abre fichero
jsonfile = open(DIR_PATH + file_name, "wb")
#guarda los datos en el fichero
json.dump(data, jsonfile)
#cierra el fichero
jsonfile.close()

# Cargar datos desde json ############################################################
def load_data(file_name):
"""
Carga los datos json del fichero 'file_name' en 'data'.
"""
#abre fichero
jsonfile = open(DIR_PATH + file_name, "rb")
#guarda los datos del fichero en data
```

```
data = json.load(jsonfile)
#cierra el fichero
jsonfile.close()
return data

# Extrae las variables deseadas de los datos del json ##########################
def shape_data(data, extract_keys):
"""
Extra las variables en 'extract_keys' del conjunto completo de datos en 'data' y convierte los
datos en una array de numpy.
"""
X = []
#para cada muestra en los datos
for row in data:
#únicamente extrae los datosque hay en extract_keys
new_row = [value for key, value in row.iteritems() if key in extract_keys]
#guarda datos reducidos en X
X.append(new_row)
#devuelve los nuevos datos convertidos en un array
return np.array(X)

# Carga los ids de los friends #################################################
def friends_ids(account_name, limit = 5000):
"""
Carga en 'friends_ids' los ids de los friends de la cuenta 'account_name'.
"""
#inicializamos cursor
cursor = -1
friends_ids = blist([])
#mientras hayan datos
while cursor != 0:
#aplica api para cargar ids
query = twitter.get_friends_ids(screen_name = account_name, cursor = cursor, count = limit)
#únicamente guardamos ids
friends_ids.extend(query["ids"])
#actualizamos paginación
cursor = query["next_cursor"]
#pausa de 60 segundos por la api
time.sleep(60)
return list(friends_ids)
# Carga los ids de los followers ###############################################
def followers_ids(account_name, limit = 5000):
"""
Carga en 'followers_ids' los ids de los followers de la cuenta 'account_name'.
"""
#inicializamos cursor
cursor = -1
followers_ids = blist([])
#mientras hayan datos
while cursor != 0:
#aplica api para cargar ids
query = twitter.get_followers_ids(screen_name = account_name, cursor = cursor, count = limit)
#únicamente guardamos ids
followers_ids.extend(query["ids"])
#actualizamos paginación
cursor = query["next_cursor"]
#pausa de 60 segundos por la api
time.sleep(60)
return list(followers_ids)

# Carga los datos de perfil de cada seguidor ###################################
def twitter_user_data(file_ids, save_file):
"""
Carga en 'save_file' los datos de perfil de cada usuario en 'file_ids'.
"""
data = blist([])
#según los datos ya calculados sigue calculando
total_uids = load_data(file_ids)
try:
prev_data = load_data(save_file)
except:
prev_data = []
prev_data_ids = [user["id"] for user in prev_data]
followers_ids = list(set(total_uids) - set(prev_data_ids))
grouped_followers_ids = [followers_ids[n : n + 100] for n in range(0, len(followers_ids),
100)]
#para cada grupo de ids
for grouped_ids in grouped_followers_ids:
try:
#calcula los datos de usuario
followers_obj = twitter.lookup_user(user_id = grouped_ids, include_entities = False)
#para cada usuario
for user in followers_obj:
#únicamente guardamos las características deseadas
follower = {}
follower["id"] = user["id"]
follower["name"] = user["name"]
follower["screen_name"] = user["screen_name"]
```

```
follower["created_at"] = user["created_at"]
follower["description"] = user["description"]
follower["url"] = user["url"]
follower["favourites_count"] = user["favourites_count"]
follower["followers_count"] = user["followers_count"]
follower["friends_count"] = user["friends_count"]
follower["language"] = user["lang"]
follower["location"] = user["location"]
follower["protected"] = user["protected"]
follower["statuses_count"] = user["statuses_count"]
data.append(follower)
except Exception as e:
if e.error_code == 401:
print "error: ", e.error_code
else:
print str(e)
print e.error_code
break
#pausa de 15 segundos por la api
time.sleep(15)
#guardamos datos calculados
save_data(prev_data + list(data), save_file)
return data

# Carga los topics de cada usuario ##########################################
def twitter_user_status(file_data, save_file):
"""
Carga en 'save_file' los status (tweets y retweets) de los usuarios en 'file_data'.
"""
data = blist([])
#según los datos ya calculados sigue calculando
total_data = load_data(file_data)
try:
prev_data = load_data(save_file)
except:
prev_data = []
prev_data_ids = [user["id"] for user in prev_data]
followers_obj = [user for user in total_data if user["id"] not in prev_data_ids]
#para cada usuario
for follower in followers_obj:
follower["status"] = []
#si todavía existe y no está protegido
if not follower["protected"] and follower["statuses_count"] > 0:
try:
#calcula con la api los últimos 200 tweets y retweets
user_timeline = twitter.get_user_timeline(user_id = follower["id"], count = 200)
#para cada tweet/retweet
for status in user_timeline:
#guardamos únicamente las variables deseadas
user_status_dict = {}
user_status_dict["text"] = status["text"]
user_status_dict["hashtags"] = [hashtag["text"] for hashtag in status["entities"]["hashtags"]]
user_status_dict["created_at"] = status["created_at"]
user_status_dict["coordinates"] = status["coordinates"]
follower["status"].append(user_status_dict)
except Exception as e:
if e.error_code == 401:
print "error: ", e.error_code
follower["status"].append("Error 401: protected user")
else:
print str(e)
print e.error_code
break
data.append(follower)
#pausa de 3 segundos por la api
time.sleep(3)
#guardamos datos calculados
save_data(prev_data + list(data), save_file)
return data
```

Código Python fichero ml_functions.py

```
# -*- coding: utf-8 -*-
"""
FUNCIONES RELACIONADAS CON ML: CLUSTERING DATA

Conjunto de funciones que implementan diversas funciones relacionadas con Machine Learning:

* normalize_data: escala y normaliza el conjunto de datos.
* pca: implementa el algoritmo pca para la reducción de dimensiones de los datos.
* cluster_assignment: asigna cada muestra a un cluster.
* move_centroids: recalcula los centros de los nuevos clusters.
* cluster_data: calcula repetidamente los clusters hasta que converge.
```

```
* cost_function: calcula la función de costes.
* kmeans: repite el algoritmo k-means para minimizar mínimos locales.
* kmeans_elbow: permite aplicar el método elbow para escoger el número de clusters.
"""

#Librerías
import numpy as np
import random as rand
import matplotlib.pyplot as plt

# Normaliza y escala los datos para optimizarlos.
def normalize_data(X):
"""
Obtimiza los datos al escalarlos y normalizarlos.
"""
#restamos cada característica entre su media
X -= np.mean(X, 0)
#dividimos cada característica entre su deviación standard
X /= np.std(X, 0)
#escalamos los datos a un max de 1
X /= np.abs(X).max()
return X

# Principal Component Analysis PCA. Reduce la dimensionalidad.
def pca(X, var):
"""
Algoritmo Principal Component Analysis, PCA que reduce la dimensionalidad de los datos
manteniendo la varianza deseada.
"""
#matriz covarianza
C = np.cov(X.T)
#calculamos autovectores y autovalores
U, s, V = np.linalg.svd(C)
#buscamos el número de dimensiones que mantienen la varianza señalada
for k in range(1, len(s)):
variance_retaines = sum(s[0:k]) / sum(s)
if variance_retaines >= var:
break
#reducimos el espacio manteniendo los autovectores con mayor información
U_reduced = U[:,:k]
#proyectamos los datos en el nuevo espacio de dimensión reducida
Z = U_reduced.T.dot(X.T)
return Z
# Asigna cada muestra a un cluster u otro según la distancia a este.
def cluster_assignment(X, centroids):
"""
Asigna cada muestra en 'X' a un cluster u otro según la distancia al centro 'centroid' de
este.
"""
clusters = {}
#para cada muestra
for x in X:
norms = []
#recorremos cada centro
for centroid in enumerate(centroids):
#calculamos normas entre la muestra y los centro de los clusters en norms
norms.append([np.linalg.norm(x-centroid[1]), centroid[0]])
#escogemos el cluster según distanica
cluster_key = min(norms)[1]
#asignamos el punto al cluster escogido
if clusters.has_key(cluster_key):
clusters[cluster_key].append(x)
else:
clusters[cluster_key] = [x]
return clusters

# Recalcula los centros de los nuevos clusters.
def move_centroids(clusters):
"""
Recalcula los 'centroids' de los nuevos clusters calculados.
"""
new_centroids = []
#recorremos cada cluster
for k in clusters.keys():
#calcula el centro del cluster
new_centroids.append(np.mean(clusters[k], 0))
return new_centroids

# Algoritmo k-means.
def cluster_data(X, K):
"""
Implementa el algoritmo kmeans con 'K' clusters...segmenta datos hasta converger.
"""
#inicializa los centros de forma aleatoria
centroids = rand.sample(X, K)
prev_centroids = np.array(0)
#mientras vaya recalculando nuevos centro: no converge
while not np.array_equal(centroids, prev_centroids):
```

```
prev_centroids = centroids
#asigna muestras a clusters
clusters = cluster_assignment(X, centroids)
#recalcula los centros de los clusters asignados
centroids = move_centroids(clusters)
return (centroids, clusters)

# Calcula la función de costes.
def cost_function(centroids, clusters):
"""
Calcula la función de costes o distorsión.
"""
J, m = 0, 0
#para cada cluster
for k in clusters.keys():
#número de muestras en el cluster
m += len(clusters[k])
#recorremos las muestras del cluster
for x in clusters[k]:
#calculamos las distancias o norma de las muestras al centro
J += np.linalg.norm(x-centroids[k])
return J / m

# Repite el algoritmo k-means y se queda con el mejor resultado.
def kmeans(X, K, repeat = 10):
"""
Repite el algoritmo kmeans varias veces para evitar soluciones locales y devuelve la mejor
solución según la función de costes.
"""
#inicializa la función de costes a un valor elevado
prev_F = 1000
#número de veces que recalculamos kmeans
for i in range(repeat):
centroids, clusters = cluster_data(X, K)
F = cost_function(centroids, clusters)
#si mejoramos la figura de ruido
if F < prev_F:
prev_F = F
best_kmeans = [centroids, clusters, F]
return (best_kmeans[0], best_kmeans[1], best_kmeans[2])

#K-means con el método elbow para escoger la K
def kmeans_elbow(X, Kmax = 6, repeat = 10):
"""
Aplica el método Elbow para escoger manualmente la mejor K.
"""
data = []
#para distintos números de agrupaciones
for k in range(1, Kmax):
#calcula kmeans
centroids, clusters, F = kmeans(X, k, repeat)
data.append([k, F, centroids, clusters])
data_array = np.array(data)
#dibujamos figura de ruido según el número de clusters
elbow_data = data_array[:,:2]
plt.figure()
ax = plt.gca()
ax.plot(elbow_data[:,0], elbow_data[:,1], "b-")
ax.set_title("Elbow method")
ax.set_xlabel("K (num. of clusters)")
ax.set_ylabel("F (Cost function)")
plt.show()
#escogemos manualmente el número de clusters K
K = raw_input("Enter clusters number K (max: 6): ")
K = int(K)
if 0 < K < 7:
print "Correct input. Cluster number is " + str(K)
else:
print "Invalid input. Default K = 4"
K = 4
return (data[K - 1][2], data[K - 1][3], K)
```

Código Python fichero step1_filter.py

```
# -*- coding: utf-8 -*-

"""
EXPERIMENTO 1:

USUARIOS 'FAKE'

Filtra los usuarios según idioma, actividad y/o 'calidad'.
"""
```

```
# Librerías
import random as rand
import data_admin
from blist import blist
from datetime import date

# Filtra la cuenta de usuarios fake
def filter_account(followers, lang_filter = "es", monthly_tweets = 4, followers_filter = 10,
quality_filter = False, limit = False):
"""
Extrae de 'followers' aquellos usuarios que no cumplan con los parámetros deseados:
lang_filter, monthly_tweets, followers_filter, quality_filter
"""
if type(followers) == str:
followers = data_admin.load_data(followers)
filtered_followers = blist([])
if limit:
followers = rand.sample(followers, limit)
#para cada seguidor
for follower in followers:
#cargamos filtro actividad
statuses_filter = (1 + date.today().year / int(follower["created_at"][-4:])) * 12 *
monthly_tweets
#filtramos por idioma, por número de seguidores y por actividad
if follower["language"] == lang_filter and follower["followers_count"] > followers_filter and
follower["statuses_count"] > statuses_filter:
#filtramos por calidad según rate
if quality_filter:
friends = follower["friends_count"]
if follower["friends_count"] == 0:
friends = 1
rate = follower["followers_count"] / friends
if rate >= quality_filter:
filtered_followers.append(follower)
else:
filtered_followers.append(follower)
return list(filtered_followers)
# alcande potencial de la cuenta
def potential_reach(followers):
"""
Calcula el aclance potencial de la cuenta a partir de los seguidores de los
followers de la cuenta.
"""
pot = 0
for follower in followers:
pot += follower["followers_count"]
return pot + len(followers)
```

Código Python fichero step2_detection.py

```
# -*- coding: utf-8 -*-

"""
EXPERIMENTO 3:

REPRESENTA LOS SEGUIDORES SEGÚN FOLLOWERS/FRIENDS

Representa los seguidores según el índice de followers/friends el cual es un indicador
de la calidad del seguidor. Para así identificar los celebrities.
"""

# Librerías
import data_admin
import matplotlib.pyplot as plt
import numpy as np
import random as rand
import ml_functions
from mpl_toolkits.mplot3d import Axes3D

# Variables globales
MARKERS = ["o", "o", "o", "o", "o", "o"]
#COLORS = ["#ffba00", "#ff00ea", "#0096ff", "#00ff78", "#ff001e", "#ff5a00", "#00ffea"]
COLORS = ["b", "g", "r", "c", "m", "y"]

# Aplica kmeans a los datos 2D y 3D
def cluster_followers(followers, keys, title = "", method = False, K = 4, fig = [1, 2, 3],
limit = False):
"""
Representa los seguidores mediante sus friends y followers, en escala logarítmica.
Y permite segmentar los seguidores a partir del algoritmo k-means implementado por lo que
se usa en el paso 3.
"""
clusters = []
```

```
#carga de followers, según fichero o variable y con límite o con todos
if type(followers) == str:
followers = data_admin.load_data(followers)
if limit:
followers = rand.sample(followers, limit)
#clustering 2D
if len(keys) == 2:
if fig[0] != 0:
plot_followers_2d(followers, keys, False, title + " Friends VS Followers", "r", fig[0])
logX = plot_followers_2d(followers, keys, True, title + " Friends VS Followers", "g", fig[1])
if method == "elbow":
centroids, clusters, K = ml_functions.kmeans_elbow(logX, 10, 30)
elif method == "kmeans":
centroids, clusters, F = ml_functions.kmeans(logX, K, 30)
#dibujamos la segmentación de los datos
if method:
plt.figure(fig[2])
plt.axis([0, 7, 0, 7])
ax = plt.gca()
for k, clr, mkr in zip(range(K), COLORS, MARKERS):
cluster = np.squeeze(clusters[k])
if len(cluster) == 2: # grupo de únicamente un follower
ax.plot(cluster[1], cluster[0], linestyle="None", markerfacecolor=clr, marker=mkr,
markersize=4)
else:
ax.plot(cluster[:,1], cluster[:,0], linestyle="None", markerfacecolor=clr, marker=mkr,
markersize=4)
#ax.plot(centroids[k][1], centroids[k][0], linestyle="None", markerfacecolor=clr, marker="*",
markersize=15)
ax.set_title("Clustering Friends VS Followers. K = " + str(K))
ax.set_xlabel("log(followers)")
ax.set_ylabel("log(friends)")
ax.set_autoscale_on(False)
plt.show()
#clustering 3D
elif len(keys) == 3:
if fig[0] != 0:
plot_followers_3d(followers, keys, False, title + " Friends VS Followers VS Statuses", "r",
fig[0])
logX = plot_followers_3d(followers, keys, True, title + " Friends VS Followers VS Statuses",
"g", fig[1])
if method == "elbow":
centroids, clusters, K = ml_functions.kmeans_elbow(logX, 10, 20)
elif method == "kmeans":
centroids, clusters, F = ml_functions.kmeans(logX, K, 20)
#dibujamos la segmentación de los datos
if method:
plt.figure(fig[2])
ax = plt.gca(projection="3d")
for k, clr, mkr in zip(range(K), COLORS, MARKERS):
cluster = np.squeeze(np.asarray(clusters[k]))
ax.plot(cluster[:,1], cluster[:,0], cluster[:,2], linestyle="None", markerfacecolor=clr,
marker=mkr, markersize=5)
#ax.plot([centroids[k][1]], [centroids[k][0]], [centroids[k][2]], linestyle="None",
markerfacecolor=clr, marker="*", markersize=15)
ax.set_title("Clustering Friends VS Followers VS Statuses. K = " + str(K))
ax.set_xlabel(keys[0])
ax.set_ylabel(keys[1])
ax.set_zlabel(keys[2])
ax.set_xlim([0,7])
ax.set_ylim([0,7])
ax.set_zlim([0,7])
plt.show()
else:
pass
print "Too many dimensions"
return clusters
# Dibuja los followers en 2D
def plot_followers_2d(followers, keys, logdata = False, title = "Plotting followers 2D", color
= "b", fig = 1, limit = False):
if limit:
followers = rand.sample(followers, limit)
X = data_admin.shape_data(followers, keys)
axislim = max(X[:,1])
if max(X[:,1]) < max(X[:,0]):
axislim = max(X[:,0])
plt.figure(fig)
plt.axis([0, axislim, 0, axislim])
ax = plt.gca()
if logdata:
title = title + " (log scale)"
X[X == 0] = 1
X = np.log10(X)
plt.axis([0, 7, 0, 7])
ax.plot([0,7], [0,7], "r-")
ax.plot(X[:,1], X[:,0], color + ".")
ax.set_title(title)
```

```
ax.set_xlabel(keys[0])
ax.set_ylabel(keys[1])
ax.set_autoscale_on(False)
plt.show()
return X

# Dibuja los followers en 3D
def plot_followers_3d(followers, keys, logdata = False, title = "Plotting followers 3D", color
= "b", fig = 1, limit = False):
if limit:
followers = rand.sample(followers, limit)
X = data_admin.shape_data(followers, keys)
plt.figure(fig)
ax = plt.gca(projection="3d")
ax.set_xlim([0, max(X[:,0])])
ax.set_ylim([0, max(X[:,1])])
ax.set_zlim([0, max(X[:,2])])
if logdata:
title = title + " (log scale)"
X[X == 0] = 1
X = np.log10(X)
ax.set_xlim([0,7])
ax.set_ylim([0,7])
ax.set_zlim([0,7])
ax.plot(X[:,1], X[:,0], X[:,2], color + ".")
ax.set_title(title)
ax.set_xlabel(keys[0])
ax.set_ylabel(keys[1])
ax.set_zlabel(keys[2])
plt.show()
return X
```

Código Python fichero step3_celebrities.py

```
# -*- coding: utf-8 -*-

"""
EXPERIMENTO 3:

ANÁLISIS DE LOS SEGUIDORES

Extrae información de los seguidores para analizar su contribución a la marca.
"""
# Librerías
from datetime import datetime

# Extrae estadísticas de los followers celebrity
def celebrity_statistics(celebrities, mention):
#inicializamos variables
total_activity, total_followers, total_mentions = 0, 0, 0
timeline_counts = {}
top10celebrities = []
#para cada celebrity
for celebrity in celebrities:
user_count_mentions = 0
#para cada tweet o retweet del celebrity
for status in celebrity["status"]:
if type(status) == dict and mention in status["text"]:
user_count_mentions += 1
month = status["created_at"][4:7]
year = status["created_at"][-4:]
key = " ".join([month, year])
#contabilizamos según fecha del tweet
if key in timeline_counts:
timeline_counts[key] += 1
else:
timeline_counts[key] = 1
top10celebrities.append({"celebrity": celebrity["name"] + " (" + celebrity["screen_name"] +
")", "mentions": user_count_mentions, "followers": celebrity["followers_count"]})
total_activity += celebrity["statuses_count"]
total_followers += celebrity["followers_count"]
#menciones totales
total_mentions = sum(timeline_counts.itervalues())
total_celebrities = len(celebrities)
#media del número de tweets de cada celebrity
av_activity = total_activity / total_celebrities
#media del número de followers de cada celebrity
av_follower = total_followers / total_celebrities
statistics = [av_activity, av_follower, total_mentions]
#preparamos timeline
timeline_counts = sorted(timeline_counts.items(), key = lambda w:datetime.strptime(w[0],"%b
%Y"))
timeline_dates = [t[0] for t in timeline_counts]
```

```
timeline = [timeline_counts, timeline_dates]
#ordenamos y sacamos 10
top10celebrities = sorted(top10celebrities, key = lambda k: k["mentions"], reverse =
True)[0:10]
return (statistics, timeline, top10celebrities)
```

Código Python fichero step4_topics.py

```
# -*- coding: utf-8 -*-

"""
EXPERIMENTO 4:

EXTRAE LOS TOPICS

Extrae menciones y hashtags de los usuarios para analizar los últimos topics más
importantes.
"""

#Librerías
import numpy as np
import string as st
import nltk
import ml_functions
import matplotlib.pyplot as plt
import data_admin
from datetime import datetime

# Extrae línea de tiempo de las altas de los followers
def get_follower_timeline(followers):
"""
Acumula los usuarios según su fecha de creación.
"""
followers_timeline = {}
if type(followers) == str:
followers = data_admin.load_data(followers)
#para cada usuario
for follower in followers:
#extraemos mes y año de su fecha de creación
month = follower["created_at"][4:7]
year = follower["created_at"][-4:]
key = " ".join([month, year])
#actualiza contador según fecha
if int(year) > 2008:
if key in followers_timeline.iterkeys():
followers_timeline[key] += 1
else:
followers_timeline[key] = 1
followers_timeline = sorted(followers_timeline.items(), key = lambda
w:datetime.strptime(w[0],"%b %Y"))
return followers_timeline
# Limpia tokens
def tokens_cleaner(tokens):
"""
Elimina los stopwords.
"""
stop_words = nltk.corpus.stopwords.words("spanish")
stop_words += list(st.punctuation)
stop_words += ["“".decode("utf8"), "…".decode("utf8"), "¡".decode("utf8"), "''", "``", "...",
"ff", "rt", "s", "l", "tod"]

#stop_words += ["''", "``", "´´", "“".decode("utf8"), "…".decode("utf8"), "¡", "http",
"https", "rt", "te", "si", "_", "//t.", "va", "the", "in", "da", "s", "...", "tod"]
tokens_cleaned = [t for t in tokens if t not in stop_words]
return tokens_cleaned

# calcula los pesos mediante tf-idf
def term_weights(X):
"""
Calcula los pesos tf-idf.
"""
d = float(np.shape(X)[0])
#term frequency con escala log
tf = np.log10(X)
tf[tf == 0] = 1
tf[np.isinf(tf)] = 0
#inverse document frequency
df = (X != 0).sum(0)
idf = d / df
idf = np.log10(idf)
#pesos finales
tf_idf = tf * idf
weights = np.around(tf_idf, decimals = 3)
```

```
return weights
# Adaptamos los datos: filtramos palabras, tokenizamos y extraemos frecuencias
def followers_topics(followers, interval = (-50, None), limit = False):
"""
Extracción de los topics.
"""
followers_bag_words = []
if type(followers) == str:
followers = data_admin.load_data(followers)
if limit:
followers = followers[0:limit]
#extraemos tokens y sus freq
for follower in followers:
tokens = []
for status in follower["status"]:
if type(status) == dict:
words = status["text"].split()
tokens += [w for w in words if "@" in w]
tokens += status["hashtags"]
tokens = nltk.tokenize.word_tokenize(" ".join(tokens).lower())
tokens = tokens_cleaner(tokens)
count_words = dict(nltk.FreqDist(tokens))
followers_bag_words.append(count_words)
#creamos matriz con los topics y sus freq
topics = [list(fbw.keys()) for fbw in followers_bag_words]
topics = {topic for union in topics for topic in union}
indextopics = dict(enumerate(topics))
topics = np.array(list(indextopics.values()))
keys = list(indextopics.keys())
X = np.empty((len(followers), len(indextopics.keys())))
for i, follower in enumerate(followers_bag_words):
X[i,:] = [follower.get(indextopics[k], 0) for k in keys]
#calculamos pesos reales mediante tf-idf
X = term_weights(X)
#ordenamos y extremos los mejores topics
topics_counts = X.sum(0) / float(np.shape(X)[0])
Xaux = np.vstack([topics_counts, X])
idx_order = Xaux[0,:].argsort()
X = X[:, idx_order]
X = X[:, interval[0]:interval[1]]
mask = np.all(X == 0, axis=1)
X = X[-mask]
topics = topics[idx_order]
topics = topics[interval[0]:interval[1]]
topics_weights = topics_counts[idx_order]
topics_weights = topics_weights[interval[0]:interval[1]]
topics_weights = list(zip(topics, topics_weights))
return (X, topics, topics_weights)

# Dibuja las frecuencias de los topics con stackplot
def plot_topics_area(freqs, xticks, ymax, title = "Topics freq.", color = "b", fig = 1):
c = ["b", "c"]
x = np.arange(len(xticks))
plt.figure(fig)
plt.xticks(x, xticks)
ax = plt.gca()
if len(freqs) == 2:
ax.stackplot(x, freqs[0], colors = c[0])
ax.stackplot(x, freqs[1], colors = c[1])
p1 = plt.Rectangle((0, 0), 1, 1, fc="b")
p2 = plt.Rectangle((0, 0), 1, 1, fc="c")
ax.legend([p1, p2], ["@adidas_ES", "@Nike_Spain"], loc = 2)
else:
ax.stackplot(x, freqs, colors = color)
ax.set_title(title)
ax.set_xlabel("Topics")
ax.set_ylabel("Counts")
ax.set_xticks(x)
ax.set_xticklabels(xticks, rotation = 35)
plt.ylim((0, ymax))
plt.grid()
plt.show()
# Dibuja las frecuencias de los topics con bar
def plot_topics_bars(freqs, xticks, ymax, title = "Topics freq.", color = "b", fig = 1):
c = ["g", "b"]
plt.figure(fig)
ax = plt.gca()
width = 0.35
x = np.arange(len(xticks))
if len(freqs) == 2:
bars1 = ax.bar(x, freqs[0], width, color = c[0])
bars2 = ax.bar(x + width, freqs[1], width, color = c[1])
ax.legend((bars1[0], bars2[0]), ("@adidas_ES", "@Nike_Spain"), loc = 2)
else:
ax.bar(x, freqs, width, color = color)
ax.set_title(title)
ax.set_xlabel("Topics")
ax.set_ylabel("Counts")
```

```
ax.set_xticks(x + width)
ax.set_xticklabels(xticks, rotation = 35)
plt.ylim((0, ymax + .01))
plt.show()

def celebrities_topics(X1, X2, keys):
plt.figure()
ax = plt.gca(projection="3d")
ax.plot(X1[0], X1[1], X1[2], "go")
ax.plot(X2[0], X2[1], X2[2], "bo")
ax.set_title(keys[0])
ax.set_xlabel(keys[1])
ax.set_ylabel(keys[2])
ax.set_zlabel(keys[3])
plt.show()
x1 = np.concatenate((X1[0], X2[:,0]), axis=0)
x2 = np.concatenate((X1[1], X2[:,1]), axis=0)
x3 = np.concatenate((X1[2], X2[:,2]), axis=0)
X = np.array([x1, x2, x3]).T
centroids, clusters, K = ml_functions.kmeans_elbow(X, 10, 40)
colors = ["#0099cb", "#ade601", "#fe9900", "#d8007d", "#8800cc", "#2201cc"]
markers = ["o", "v", "s", "<", ">", "."]
fig = plt.figure()
ax = fig.gca(projection="3d")
for k, clr, mkr in zip(range(K), colors, markers):
cluster = np.squeeze(np.asarray(clusters[k]))
ax.plot(cluster[:, 0], cluster[:, 1], cluster[:, 2], linestyle="None", markerfacecolor=clr,
marker=mkr, markersize=5)
ax.plot([centroids[k][0]], [centroids[k][1]], [centroids[k][2]], linestyle="None",
markerfacecolor=clr, marker="*", markersize=15)
ax.set_title("Clustering: " + keys[0])
ax.set_xlabel(keys[1])
ax.set_ylabel(keys[2])
ax.set_zlabel(keys[3])
plt.show()
```

Resum

Als últims anys, les xarxes socials han esdevingut una eina empresarial indispensable amb la que, en definitiva, augmentar les vendes a les companyies. El present treball fa front al repte de realitzar analítica a Twitter, mitjançant la comparativa entre dues grans marques com son adidas i Nike, i experimentant amb l'algorisme k-menas i tècniques *Natural Language Processing* amb les que extraure patrons d'interès. Entre els resultats obtinguts destacar 3 punts: la gran quantitat de seguidors irrellevants des del punt de vista del mercat dels dos comptes, la detecció i classificació dels seguidors més rellevants i influents, i l'extracció dels temes tuitejats més freqüents.

Resumen

En los últimos años, las redes sociales se han convertido en una herramienta empresarial indispensable con la que, en definitiva, aumentar las ventas en las compañías. El presente trabajo afronta el reto de realizar analítica en Twitter, mediante la comparativa entre dos grandes marcas como son adidas y Nike, y experimentando con el algoritmo k-means y técnicas *Natual Language Processing* con las que extraer patrones de interés. Entre los resultados obtenidos destacar 3 puntos: la gran cantidad de seguidores irrelevantes desde el punto de vista del mercado de ambas cuentas, la detección y clasificación de los seguidores más relevantes e influyentes, y la extracción en los temas tuiteados más frecuentes.

Abstract

In recent years, social media have become an indispensable business tool to ultimately increase sales companies. This paper addresses the challenge of performing analytics on Twitter, by comparing two great brands such as adidas and Nike, and experimenting with k-means algorithm and Natual Language Processing techniques to extract interesting patterns. Among the results, it should be highlighted three points: the large number of followers irrelevant from the point of view of the both market accounts, the detection and classification of the most important and influential followers, and extraction of the most frequent topics tweeted.