UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA INFORMÀTICA

Universitat Politècnica de València

Recopilación de *tweets* y medición de la relevancia basada en la topología

Trabajo Fin de Grado

Grado en Ingeniería Informática

Alumno: Daniel Duato Catalán

Tutores: Lluís Felip Hurtado Oliver
y Ferran Pla Santamaría

2014-2015

Resumen

En este proyecto se ha creado una base de datos MongoDB y una aplicación en Python que utiliza la API de Twitter para obtener tuits e información de las cuentas de usuarios. Se ha obtenido una colección de tuits relacionados con política española. Se han implementado los algoritmos PageRank y HITS usando matrices obtenidas a partir tanto de las relaciones de seguimiento entre cuentas como de las relaciones de retuits y se han empleado para ordenar a los autores de los tuits recogidos según su relevancia. La calidad de las distintas ordenaciones se ha estudiado y comparado con métricas más triviales. El algoritmo PageRank ha dado los mejores resultados. Se ha empleado el algoritmo de clusterización k-medoids y se ha estudiado la relevancia y la polaridad con respecto a distintos partidos políticos de los diferentes clústeres creados.

Palabras clave relevancia, Twitter, PageRank, HITS, clúster

Resum

En aquest projecte s'ha creat una base de dades MongoDB i una aplicació en Python que utilitza la API de Twitter per a obtenir tuits i informació dels comptes d'usuaris. S'ha obtingut una col·lecció de tuits relacionats amb política espanyola. S'han implementat els algorismes PageRank i HITS usant matrius obtingudes a partir tant de les relacions de seguiment entre comptes com de les relacions de retuits i s'han emprat per a ordenar als autors dels tuits recollits segon la seua rellevància. La qualitat de les diferents ordenacions s'ha estudiat i comparat amb mètriques més trivials. L'algorisme PageRank ha donat els millors resultats. S'ha emprat l'algorisme de clusterització k-medoids i s'ha estudiat la rellevància i la polaritat pel que fa a diferents partits polítics dels diferents clusters creats.

Paraules clau rellevància, Twitter, PageRank, HITS, cluster

Abstract

In this project a MongoDB database and a Python application which uses Twitter's API to obtain tweets and information of the user accounts have been created. A collection of tweets related with spanish politics has been obtained. The algorithms PageRank and HITS have been implemented using matrixes created using both the following and the retweeting relationships between user accounts. The results have been used to sort the authors of the obtained tweets according to their relevance. The quality of the different sortings has been studied and compared with more trivial metrics. The PageRank algorithm has showed the best results. The clustering algorithm k-medoids has been applied. The relevance and polarities relating various political parties of the clusters created have been studied.

Keywords relevance, Twitter, PageRank, HITS, cluster

Índice general

Capítulo 1

Introducción

Estamos rodeados de pruebas de la importancia de Internet en el mundo moderno, una importancia que cada día crece más. El número de usuarios de Internet sigue subiendo, actualmente se estima que el 40.4 % [27] de la población mundial lo son. Una parte de estos usuarios son creadores de contenido, que incrementan la información existente en la web añadiendo contenido a páginas webs o creando páginas nuevas. El presidente ejecutivo de Google, Eric Schmidt, dijo en 2010 que cada dos días se está creando tanta información como se creó desde el inicio de la civilización hasta el 2003 [8]. Hoy esta velocidad es aún mucho mayor.

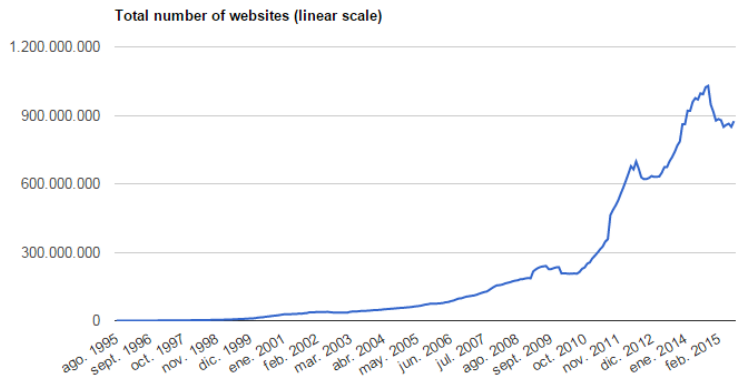


Figura 1.1: Crecimiento en el número de páginas web[17].

Con tanta información a nuestro alcance, ha sido fundamental la existencia de medios capaces de dar orden y fácil acceso a tantas páginas y datos. Es por eso que directorios de enlaces y motores de búsqueda que pretendían hacer la web más accesible a todo el mundo fueron una parte esencial de Internet. Empezando por Archie ya en el 1990, antes aún de la World Wide Web, y continuando con otros servicios como Yahoo!, Lycos, AltaVista o Google. Además de estas empresas, muchas otras aparecieron y desaparecieron [31]. Los motores de búsqueda han ido perfeccionándose rápidamente, tratando de seguir el ritmo al que crecía la Web. Finalmente hemos llegado a gigantes como Google, siendo la página web

más visitada, que en el año 2014 ha realizado aproximadamente 2,1 billones de búsquedas [12].

Otra parte importante de Internet durante los últimos años han sido las redes sociales. Estas permiten a las personas conectar entre ellas y compartir información de manera sencilla. Son servicios como Facebook, LinkedIn, Twitter o Google+, entre otros. Por supuesto, como en el caso de los motores de búsqueda, muchos de estos han sido olvidados. Sus usuarios cada vez comparten más datos en estas páginas, convirtiéndose ellas también en una sustancial fuente de datos.

Toda la información disponible gracias a estas redes sociales es abrumadora e inabarcable para una persona. Tratar de alcanzarla y comprenderla toda es una ambición imposible, por lo que será necesario poder distinguir qué información de ésta es realmente útil o relevante y cual no le interesa al usuario, por ello es natural tratar de automatizar esta selección de información esencial y delegarla a algoritmos y mecanismos que pueden ser mucho más eficientes que nosotros. Para ello, se han usado técnicas de minería de datos o han sido adaptadas técnicas propias de los motores de búsqueda, dadas las similitudes existentes entre las topologías de estas redes sociales con la de Internet. Por ejemplo, en el caso de Twitter, la relación de una cuenta que sigue a otra cuenta equivale a un enlace que apunta desde una página web a otra.

1.1. Motivación

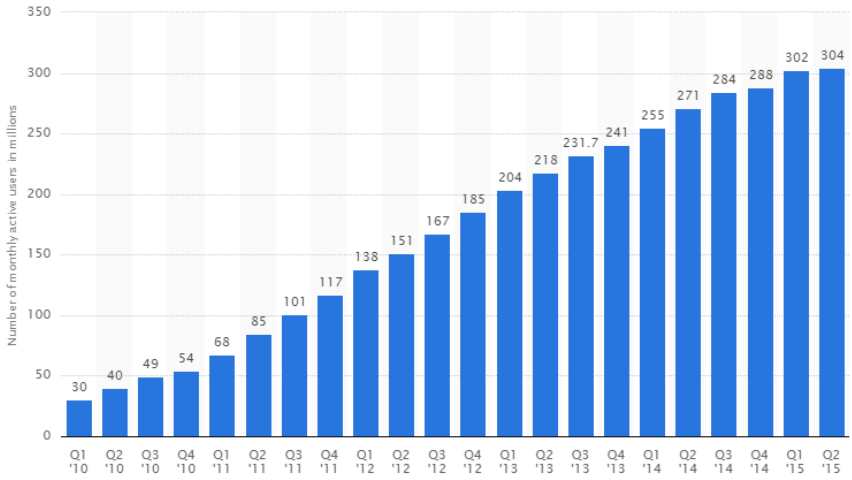


Figura 1.2: Crecimiento de usuarios activos de Twitter [26].

La discusión creada en Twitter puede llegar a tener un gran impacto en diversos sectores de la vida real, como se ha podido comprobar en múltiples ocasiones [23][5][16]. La inmediatez de las comunicaciones por Twitter hace de éste un mecanismo ideal para hacer llegar

información de manera veloz a grandes cantidades de personas. Esto hace que muchas empresas y organizaciones se interesen en Twitter como herramienta de mercadotecnia, creando profesiones tales como *community manager*. También se puede ver en Twitter un reflejo de la sociedad, aunque no exacto, sí bastante aproximado. Por ello interesa utilizar Twitter como una gran fuente de opiniones que pueden ser analizadas para poder conocer la reputación general de una persona, producto o entidad.

Para facilitar estas funciones son necesarios sistemas que puedan analizar el contenido de los tuits. Una medida posible para lograr esto es analizar la relevancia de las cuentas que publiquen los tuits, que es lo que se tratará de calcular en este proyecto, aprovechando la existencia de algunas técnicas de medición de relevancia ya existentes para su uso en la red.

1.2. Situación y problemática actual

Actualmente existen múltiples herramientas que tratan de asignar relevancia a distintos elmentos de las redes sociales. Se han investigado métodos de recomendación de usuarios para las redes sociales [25] y muchas de ellas ya tienen sistemas de este estilo puestos en práctica: Facebook y Google+ tiene sistemas de recomendación de personas que quizás conozcas, mientras que Twitter tiene uno que recomienda qué cuentas seguir (*Who To Follow*, ver figura 1.3).

Figura 1.3: Recomendación de cuentas de Twitter

Para medir la relevancia de una cuenta de Twitter no basta con observar atributos triviales como el número de seguidores, debido a cuentas *follow-back* ("sígueme y te sigo"), que crean comunidades con usuarios que se siguen mutuamente entre ellos pudiendo llegar a números muy grandes de seguidores sin tener ninguna relevancia real, y a prácticas como las de las llamadas granjas de clicks (*click farms*), que han alcanzado las redes sociales [1]. Las granjas de clicks son empresas que venden (en el caso de las redes sociales) grandes cantidades de *likes*, amigos o seguidores desde cuentas falsas, para aumentar la popularidad de una cuenta. Por ello es necesario el uso de técnicas más sofisticadas para poder calcular la relevancia de cuentas de Twitter. Si se tienen en cuenta los seguidores de una cuenta, habría que comprobar también la relevancia de estos seguidores, para asegurarnos de que no sean cuentas falsas y de que estas relaciones sean significativas.

Algoritmos diseñados para motores de búsqueda como HITS y PageRank, que pueden aprovechar la topología de seguidores de Twitter, tienen esto en cuenta. Estos algoritmos ya se han usado con anterioridad para medir relevancia de cuentas de Twitter en [3] y una variante de un algoritmo basado en ellos, SALSA (del inglés: *Stochastic Approach for Link-*

Structure Analysis), es empleado por Twitter en su sistema *Who To Follow* [9].

Twitter se está usando como fuente de análisis de la sociedad en todo tipo de áreas: desde opiniones sobre el cambio climático [13] o deportivas [32], hasta opiniones gastronómicas [30], incluyendo análisis de opiniones de la política española [4], tema con el que se trabajará durante el proyecto. Dadas tantas investigaciones realizadas a partir de datos de Twitter, es innegable la importancia de utilizar buenos métodos de extracción de información relevante.

1.3. Objetivos del proyecto

Como hemos visto, ya existen estudios que calculan la relevancia de Twitter usando algoritmos propios de motores de búsqueda, sin embargo, existen muchos aspectos a investigar referentes a estos o nuevos algoritmos para determinar la relevancia de los tuits. En este proyecto se tratará de poner a prueba estos algoritmos de nuevas maneras. Además, se empleará un nuevo enfoque no muy estudiado: se analizarán clústeres de usuarios y las similitudes en relevancia u opiniones dentro de ellos.

Será necesario crear una aplicación que, usando la API (del inglés: *Application Programming Interface*) proporcionada por Twitter, recopile tuits según una palabra o palabras clave y los almacene en una base de datos. Usando esta aplicación, se obtendrá una cantidad de tuits suficientemente grande relacionados con un tema en concreto: en nuestro caso, partidos políticos españoles.

Se implementarán los algoritmos PageRank y HITS. A partir de las cuentas obtenidas, se crearán grafos según las relaciones direccionales de qué usuario sigue a quién y de qué usuario retuitea a quién. Los algoritmos implementados se ejecutarán sobre estos grafos y se asignarán puntuaciones de relevancia a cada una de las cuentas.

Estas puntuaciones se compararán con una métrica más trivial (el número de seguidores de cada cuenta), y se evaluarán de distintas maneras:

- Usando los coeficientes de correlación de Spearman.
- Etiquetando una muestra de las cuentas con un valor de relevancia y calculando la ganancia acumulada descontada (DCG).
- Se usará un detector de polaridad para calcular la polaridad de toda la colección de cuentas obtenidas y se estudiará la representatividad de los usuarios más relevantes.

Se espara obtener mejores puntuaciones tanto en el cálculo del DCG como en el análisis de representatividad para los algoritmos HITS y PageRank que para los números de seguidores.

Finalmente se analizarán los usuarios usando técnicas de clusterización. Primero se comprobará la conectividad y luego se empleará el algoritmo k-medoids. Se analizarán las diferencias entre la alineación política y la relevancia entre los distintos clústeres.

Se espera ver diferencias entre clústeres en estos campos. Esto podría ayudar a seleccionar mejor aquellas cuentas que sean más representativas y a inferir la relevancia o alineación política de una cuenta según el clúster al que pertenezca.

1.4. Estructura del Trabajo

Este documento estará dividido en 8 capítulos:

- **Introducción** - exposición de la situación actual, algunos trabajos relacionados, la motivación y los objetivos del proyecto.
- **Conceptos previos** - explicación de algunos conceptos que serán necesarios entender.
- **Tecnologías usadas** - se explorarán las distintas herramientas y tecnologías usadas durante el proyecto.
- **Recopilación de datos** - se detallará el proceso de adquisición de tuits e información de las cuentas de Twitter.
- **Algoritmos de ordenación** - explicación de los algoritmos y su implementación.
- **Evaluación de los resultados** - se evaluarán de distintas maneras los resultados obtenidos en el capítulo anterior.
- **Análisis de clústeres** - se detallará la creación de clústeres y los resultados obtenidos a partir de ellos.
- **Conclusiones** - repaso del trabajo realizado y los resultados obtenidos y propuestas para trabajos futuros.

- **Apéndice** - listas de las primeras 30 cuentas según cada ordenación de relevancia.

Capítulo 2

Conceptos previos

2.1. Twitter

Twitter es una red social creada en 2006 que permite publicar cortos mensajes de hasta 140 caracteres. Es una de las webs que han vivido un mayor crecimiento en su número de usuarios, llegando a tener actualmente más de 500 millones de usuarios. Tal ha sido su éxito que hasta la Real Academia Española (RAE) ha incorporado palabras como 'tuit' o 'tuitero' a su diccionario (ver figura 2.1)[22].

Funciona de manera similar a un servicio de blogs personales, donde cada usuario tiene una página con un muro en el que puede publicar estos mensajes cortos, llamados tuits (del inglés: *tweet*). Los usuarios pueden suscribirse, a lo que también se le llama "seguir", a otras cuentas (cuenta y usuario se usarán de manera prácticamente intercambiable) para que así aparezcan los tuits de estas cuentas en su página principal, convirtiéndose así en su seguidor (del inglés: *follower*). En el caso opuesto, las cuentas a las que sigue un usuario serán llamadas sus amigos (del inglés: *friends*); este término está sacado de la documentación de la API de Twitter, pero a diferencia del resto, no se suele usar en conversaciones normales sobre Twitter.

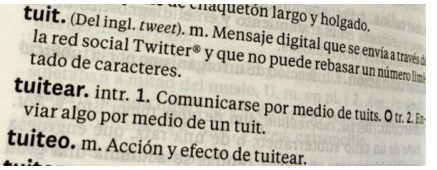

chaquetón largo y holgado.
tuit. (Del ingl. *tweet*). m. Mensaje digital que se envía a través d
la red social Twitter® y que no puede rebasar un número lim
tado de caracteres.
tuitear. intr. 1. Comunicarse por medio de tuits. O tr. 2. En
viar algo por medio de un tuit.
tuiteo. m. Acción y efecto de tuitear.

Figura 2.1: Definiciones de las palabras 'tuit', 'tuitear' y 'tuiteo' en la RAE.

Un usuario puede compartir en su muro y a la vez con todos sus seguidores cualquier tuit de otra cuenta. A esta acción se le llama retuitear y al tuit compartido retuit.

Existen más funcionalidades en Twitter, como enviar mensajes directos, responder a otro tuit, el uso de *hashtags*, etc. Pero para este proyecto solo será necesario comprender los conceptos ya explicados.

La topología de Twitter se puede ver como un gran grafo dirigido, siendo los nodos las cuentas de usuario y las aristas dirigidas las relaciones de un usuario siguiendo a otro. Otra manera manera menos trivial de construir un grafo a patir de Twitter será usando los retuits,

en este caso las aristan indicarán que un usuario retuitea tuits de otro. Ambos tipos de grafo se han usado durante este trabajo.

2.2. Relevancia

La relevancia es una propiedad que interesa a la hora de recoger información. De acuerdo a Hjørland y Sejer Christensen, un documento es relevante si ayuda en cierta tarea o investigación a completar cierto objetivo [10]. Por lo tanto algo únicamente puede ser relevante si es en relación a un objetivo dado. Es decir, un documento muy relevante dentro de cierto campo, puede no ser nada relevante en otro. Se puede diferenciar también entre relevancia tópica y situacional. Si algo tiene relevancia tópica, significa que está relacionado de alguna manera con la tarea, por ejemplo, se engloba dentro del mismo campo que se estaba investigando en la tarea. Pero sin embargo puede no tener ninguna relevancia situacional, es decir, no ser útil para la concreta situación y el concreto objetivo que se deseaba cumplir.

La medición de la relevancia puede ser algo muy complicado. Para dos personas distintas, aunque estén realizando la misma investigación con el mismo objetivo, debido a los distintos conocimientos e ideas de cada uno, un mismo documento puede tener distinta relevancia percibida.

Esto hará muy difícil la comparación de los distintos métodos de asignación de relevancia, ya que no existe una solución completamente objetiva, pero más adelante explicaremos las diferentes maneras en las que este problema puede ser abordado.

Puesto que es necesario un contexto para la medición de la relevancia, se ha decidido escoger el de la política española, coincidiendo con las elecciones autonómicas y municipales de España. Se tomarán dos enfoques ligeramente distintos a la hora de realizar las distintas pruebas:

En la sección 6.2 se tomará el papel de un votante indeciso, cuya tarea será la de obtener la mejor información posible sobre política española desde Twitter con el objetivo de realizar la mejor elección posible a la hora de votar en las elecciones autonómicas y municipales de 2015. Para obtener esta información el votante querrá seguir a un número limitado de cuentas de Twitter, por lo que la función de los sistemas de asignación de relevancia que se usarán será la de poder indicar a este votante qué cuentas son las más relevantes en cuanto a información sobre política española.

En la sección 6.3 de este trabajo, el objetivo será otro: conocer la opinión general sobre política de los usuarios de Twitter. Para ello las cuentas más relevantes serán aquellas más representativas de las opiniones de todos los usuarios de Twitter.

La relevancia tópica de cada cuenta debería ser la misma en ambos casos (política española), mientras que la situacional puede ser diferente, ya que el objetivo de la investigación es distinto.

2.3. Análisis de sentimiento

Para las pruebas que se realizarán en la sección 6.3 será necesario entender este concepto. Análisis de sentimiento es el uso de técnicas como las del procesamiento de lenguaje natural para poder extraer a partir de algún recurso información acerca de la actitud de su autor. De esta actitud se pueden detectar opiniones o estados de ánimo.

Se puede considerar este problema como uno de clasificación, en el que se trata de clasificar cada documento en una categoría de acuerdo a la opinión o estado de ánimo detectado. Por ejemplo, aquellos sistemas que detectan opiniones a menudo las clasifican como positivas, neutrales o negativas. A esta clasificación, en una única dimensión, se le llama de polaridad y es la que ha sido empleada para este trabajo.

Dada la creciente popularidad de las redes sociales, métodos de este tipo se están empleando cada vez más sobre estas redes como modo alternativo de sondeo a la población. Sobre todo Twitter es una gran fuente de recursos para estas investigaciones, ya que dispone de una gran cantidad de usuarios que publican opiniones de manera pública a las que es muy fácil acceder.

Capítulo 3

Tecnologías usadas

En este capítulo se describirán brevemente las diferentes tecnologías que se usarán durante el proyecto.

3.1. Twitter REST API

A los pocos meses de su creación, Twitter lanzó una interfaz de programación de aplicaciones o API (del inglés: *Application Programming Interface*) oficial al ver que desarrolladores independientes empezaban a idear otras maneras de acceder a sus datos y así poder crear distintos tipos de aplicaciones [28]. Esta nueva API permitía acceder a una lista de tuits públicos o, si uno se autentificaba mediante una autenticación de acceso básica, a los tuits propios, de tus amigos o de tus seguidores. Estas operaciones seguían una arquitectura REST (Transferencia de Estado Representacional), intercambiando los datos directamente mediante HTTP. Esta API rápidamente fue creciendo añadiendo más opciones y mejorando su seguridad.

A la vez se fueron ofreciendo más recursos para desarrolladores que quisieran aprovecharse de la tecnología de Twitter; como la Streaming API, que permite recibir tuits en tiempo real, o más recientemente Fabric, un entorno de desarrollo para aplicaciones móviles desarrollado por Twitter. Pero a nosotros nos interesará únicamente la REST API: trabajaremos siempre sobre la misma colección de tuits, sobre la que haremos distintas pruebas, por lo que obtener tuits en tiempo real no nos será de mucha utilidad; y Fabric es algo que no tiene ninguna relación con el objetivo de este proyecto. A continuación se explicará como obtener acceso a la REST API de Twitter (versión 1.1), profundizaremos en su funcionamiento y mencionaremos brevemente las operaciones que nos serán útiles durante el proyecto.

Para realizar cualquier operación con la API se requiere crear una aplicación que usará una autentificación de tipo OAuth. Estas aplicaciones pueden ser de dos tipos: conectada a una única cuenta de usuario o independiente de ella, de manera que varios usuarios puedan usar la misma aplicación. Las aplicaciones se crean desde la página *https://apps.twitter.com/*, para lo que es necesario tener una cuenta de Twitter. Una vez creada, tendremos acceso a la pági-

na de la aplicación, en la que obtendremos una clave de autentificación Oauth (*Consumer Key*) y su clave secreta asociada (*Consumer Secret*). Estas claves son las que usará la aplicación para conectarse con la API y, en el caso de que la aplicación a crear vaya a ser pública y usada por múltiples cuentas, nunca debería ser visible a dichos usuarios. Sin embargo la aplicación no está asociada todavía a ninguna cuenta y es incapaz de realizar peticiones a la API, para ello es necesario un token de acceso (*Access Token*) con su respectivo secreto (*Access Token Secret*). El token de acceso se puede obtener por alguno de los siguientes métodos de autenticación:

- **Inicio de sesión con Twitter** - pensado para aplicaciones basadas en web. El usuario podrá iniciar sesión en la aplicación con su cuenta de Twitter por medio de un botón, que indicará los permisos que está concediendo, tras lo cual la aplicación obtendrá un token de acceso y podrá realizar peticiones a la API en su nombre.
- **3-legged Oauth** - similar al método anterior, pero se pedirá permiso al usuario antes de hacer cada petición, aún cuando ya se le hubiese dado antes.
- **PIN-based Oauth** - pensado para aplicaciones que no tengan acceso a la web. La aplicación muestra al usuario un enlace al que podrá acceder desde cualquier explorador web. Este enlace devolverá un código PIN que el usuario deberá introducir en la aplicación para que ésta obtenga el token de acceso.
- **Tokens de dev.twitter.com** - si solo se necesita que el desarrollador tenga acceso a la aplicación, se podrá conseguir un token de acceso para la cuenta de Twitter que ha creado la aplicación desde su propia página.
- **xAuth** - este método permite obtener un token de acceso a partir del nombre y contraseña del usuario, de manera que no sea necesario el uso de la web. Puesto que requiere el uso de información sensible, no es muy común y para poder ser usado es necesario obtener permisos privilegiados de Twitter. Para obtener estos permisos hay que justificar los motivos técnicos por los que ningún otro de los métodos de autenticación es posible para la aplicación.
- **OAuth Echo** - cuando haya un tercero involucrado en la interacción con la API, se utilizará OAuth Echo para comunicar los datos de acceso de una parte a la otra.
- **Autenticación solo de la aplicación** esto se usará cuando no sea necesario asociar la aplicación a ninguna cuenta. Por lo tanto, habrá varias operaciones de la API relacionadas, como publicar nuevos tuits, que no se podrán realizar.

Se ha decidido obtener el token de acceso directamente desde dev.twitter.com, asociando así la aplicación con la cuenta del desarrolador, puesto que no es necesario que más usuarios utilicen la API y es probablemente el método más sencillo.

Una vez se tiene el token de acceso, se pueden realizar peticiones HTTP a los distintos *endpoints* ofrecidos por la REST API. Estas peticiones requieren una cabecera *Authoritation* que incluirá la clave de autentificación de la aplicación y el token de acceso, así como un *nonce* (un valor generado aleatoriamente incluído para poder determinar si dos peticiones son o no idénticas), una firma (generada codificando por HMAC-SHA1 todos los parámetros de la petición con el secreto de la clave de autentificación y el del token de acceso), así como otros parámetros de control. Además de esta cabecera, la petición incluirá todos los parámetros requeridos para cada operación.

Los *endpoints* de la REST API de twitter son enlaces a los que se pueden mandar peticiones para que realicen alguna operación, que puede ser de dos tipos: solicitud de datos, como obtener una lista de seguidores de una cuenta, o subida de datos, como subir una foto a Twitter. A los *endpoints* del primer tipo se les enviarán peticiones HTTP de tipo GET, sus parámetros se enviarán incluídos dentro de la propia url y las respuestas serán devueltas en formato JSON; mientras que a los del segundo se les enviarán peticiones de tipo POST, que podrán contener información en el cuerpo del mensaje. Por cada operación hay un *endpoint* distinto y en total existen 96 distintos (sin contar aquellos usados para procedimientos como la autentificación y los pertenecientes a otras APIs).

Existen limitaciones en el número de veces que se puede usar cada *endpoint* con un mismo token de acceso. Esta limitación se mide en intervalos de 15 minutos. Por lo tanto, si un *endpoint* tiene un límite de 15 peticiones por intervalo, significa que únicamente se podrán realizar hasta 15 peticiones a ese *endpoint* por cada 15 minutos con el mismo token de acceso. Una vez superado ese límite, no se podrán realizar más peticiones al *endpoint* hasta que hayan pasado los 15 minutos del intervalo de tiempo.

Para este proyecto serán necesarios los siguientes *endpoints*:

- **search/tweets** - se le pasa como argumento una cadena de carácteres *q* y devolverá hasta 100 tuits que contengan *q*. Se pueden elegir otras opciones, como el idioma de los tuits, su fecha de publicación máxima, etc. Los tuits devueltos tendrán una antigüedad máxima de una semana, para obtener tuits más antiguos es necesario usar otros métodos. También merece la pena mencionar que la búsqueda no es exhaustiva y no todos los tuits serán devueltos. Sobre todo tuits publicados por cuentas recientes que apenas tengan seguidores a menudo no estarán correctamente indexados y no serán devueltos por la API. Tiene un límite de 180 peticiones por intervalo.

- **users/show** - al pasarle como argumento el nombre o identificador de una cuenta de Twitter, devuelve mucha información sobre esa cuenta, como número de seguidores, descripción, último tuit, etc. Tiene un límite de 180 peticiones por intervalo.

- **friends/list** - al pasarle como argumento el nombre o identificador de una cuenta de Twitter, devuelve una lista de hasta 200 amigos de esa cuenta. Tiene un límite de 15 peticiones por intervalo.

- **statuses/user_timeline** - al pasarle como argumento el nombre o identificador de una cuenta de Twitter, devuelve una lista de hasta 200 tuits publicados por esa cuenta. De esta manera se pueden obtener únicamente hasta los 3 200 tuits más recientes de cada cuenta. Tiene un límite de 180 peticiones por intervalo.

Tanto a *search/tweets* como a *statuses/user_timeline* se les puede pasar como argumento *since_id* o *max_id*, que permiten indicar respectivamente a partir de qué tuit y hasta qué tuit se desea que se devuelvan los resultados. Esto permite obtener colecciones de tuits mayores que las que podrían ser devueltas por una sola petición. Por ejemplo, si se quieren obtener 200 tuits por medio de *search/tweets* (cuyo límite es 100 por petición), se podría lanzar primero ua petición de manera normal y a continuación una segunda que incluyese el parámetro *max_id*, con un valor igual al identificador del tuit más antiguo devuelto por la primera petición menos uno (para que el mismo tuit no sea devuelto de nuevo).

3.2. MongoDB

Puesto que todos los resultados devueltos por la REST API de Twitter se encuentran en formato JSON, se ha decidido usar MongoDB como sistema de base de datos. Es un sistema NoSQL que almacena la información en documentos tipo JSON, lo que simplifica mucho el almacenamiento de tuits y demás información devuelta por Twitter. Al no ser una base de datos relacional, no se utilizarán tablas, si no que los documentos son guardados en distintas colecciones. Se pueden realizar múltiples tipos de consultas muy flexibles, ya que incluso se pueden definir funciones JavaScript dentro de ellas. La fácil creación de índices en cualquier campo hace muy sencillo y rápido ordenar los documentos.

3.3. Python

Se ha optado por usar Python como lenguaje de programación durante todo el proyecto. Este lenguaje será muy apropiado para el proyecto, ya que es muy sencillo y flexible, lo que facilita una rápida y clara implementación de los algoritmos, así como la fácil creación de pequeñas funciones de tipo scripting, que serán necesarias durante el proyecto. Es un lenguaje muy portable en comparación con otros, por lo que se podrá usar en varias máquinas distintas sin ninguna dificultad. Además dispone de librerías fáciles de instalar que facilitarán muchos aspectos del trabajo. Ahora se describirán brevemente las librerías y paquetes usadas más relevantes.

- **Tweepy** - Esta librería funciona como un sencillo *wraper* de la API de Twitter. Incluye una clase *API* con funciones que acceden a todos los métodos de la Twitter REST API, de manera que el desarrollador no ha de preocuparse de los aspectos de menor nivel,

como la creación del *nonce* y la firma en las cabeceras de cada petición o el envío de varias peticiones con distintos *max_id* cuando se quieran obtener más resultados que los permitidos por una única petición.

- **PyMongo** ofrece herramientas para trabajar desde Python con una base de datos MongoDB. Todas las funciones de acceso a la base de datos y la sintaxis de las peticiones es idéntica a la de MongoDB, por lo que su manejo resulta trivial una vez se conoce como usar este tipo de bases de datos.
- **PyTables** es un paquete que permite administrar de manera eficaz grandes cantidades de datos. Está basado en ficheros de tipo HDF5 (Hierarchical Data Format). Nosotros lo usaremos para poder almacenar en el disco duro matrices que sean demasiado grandes para el tamaño de la memoria RAM.
- **NumPy** - Esta librería incluye todo tipo de operaciones matemáticas. Será de utilidad sobre todo a la hora de trabajar con matrices.

Capítulo 4

Recopilación de datos

En este capítulo se describirá el proceso de recopilación de tuits e información de las distintas cuentas. Como ya ha sido mencionado, se recogerán tuits pertinentes a la política española. Estos tuits han sido recogidos durante la jornada de campaña electoral de las elecciones autonómicas y municipales de España de 2015. Específicamente entre el 17 y el 23 de mayo.

Se ha elegido este tema en este momento porque es capaz de generar mucha discusión que llegará a Twitter. Se esperan usuarios con distintas opiniones lo que permitirá realizar también alálisis de polaridad.

4.1. Recopilación de tuits

El primer paso es obtener los tuits que interesa analizar. Para esto se ha creado una aplicación en Python que utiliza la librería *tweepy* para comunicarse con la REST API de Twitter, cuyo funcionamiento ya ha sido explicado en el capítulo anterior. También es necesario crear una base de datos MongoDB en la que se almacenarán los tuits.

Para tuits relacionados con la política española, se ha utilizado el endpoint *search/tweets*, usando como términos de búsqueda los nombres de partidos políticos, usando tanto sus nombres completos como sus siglas cuando fuese posible. Se han recuperado como máximo 10 000 tuits por partido político, aunque en la mayoría de los casos no había tantos en una semana. Estos tuits se obtienen en formato JSON y son almacenados en distintas colecciones de la base de datos MongoDB según el partido político que mencionen, para posteriormente poder hacer más sencillo el análisis.

Los partidos políticos cuyos tuits han sido almacenados y para los que se ha creado una colección en la base de datos son los siguientes: PP, PSOE, Podemos, Ciudadanos, UPyD. Se han elegido estos partidos porque son los que más tuits han devuelto. Al probar con otros partidos, el número de tuits que en los que eran mencionados durante la semana era demasiado pequeño.

Para hacer más rápida la recolección de estos tuits y no estar limitados por las restricciones de la API de Twitter, que permite hasta 180 llamadas a *search/tweets* por cada token de acceso, se crearon dos aplicaciones de manera que cada una tuviese un token distinto para que se ejecutaran en paralelo y así duplicar la velocidad de recogida de tuits.

Una vez obtenidos los tuits, se han pasado por algunos simples filtros, que, aunque no sean perfectos, ayudan a eliminar varios tuits que aunque contengan las palabras clave no están relacionados con política española. Por ejemplo, se descubrió que en los casos de los partidos Podemos o Ciudadanos, prácticamente todas las veces que estaban escritos en minúsculas se referían a los significados habituales de esas palabras, mientras que cuando aparecían escritos en mayúsculas, se referían a los partidos políticos. Otro caso es el de siglas como PP o PSOE, que pueden aparecer en muchos contextos, pero sobre todo se encontraban entre los caracteres aleatorios de links acortados. En este caso con simplemente exigir que las siglas no se encuentre dentro de otra palabra ha sido suficiente.

En cada tuit recogido, además del contenido del tuit, se incluyen metadatos con información como el idioma en el que está escrito, la fecha en la que se escribió, la cuenta de su autor, si es un retuit, etc. Para almacenar la información de los autores se creó una nueva colección en la base de datos, donde se insertan todos los autores distintos de los tuits almacenados. A partir de los metadatos incluídos en estos tuits podemos obtener el nombre y número de identificación (*id*) del autor del tuit y su número de seguidores entre otras cosas; pero no incluyen información de cada seguidor o la gente que sigue el autor del tuit, que es la información que será de mayor interés a la hora de calcular la relevancia asociada a cada cuenta o cada tuit.

4.2. Recopilación de información de los usuarios

Esta información se puede obtener usando distintos *endpoints* de la API, entre los cuales se ha decidido usar el endpoint *user/friends*, que obtiene los amigos de una cuenta. Se ha decidido esto en lugar de obtener los seguidores de una cuenta, ya que hay cuentas con números tan grandes de seguidores que harían inpracticable obtener la información de todos ellos. También existen usuarios que siguen a grandes números de cuentas, pero suele ser en menor medida.

La API es más restrictiva con este *endpoint* que con el de búsqueda de tuits, limitando su uso a únicamente 15 peticiones por intervalo de 15 minutos. Para acelerar el proceso aún más, se crearon otras seis aplicaciones de Twitter de manera que, sumadas a las dos ya creadas, se pudiesen realizar ocho veces más peticiones que el límite de 15, es decir, 120 peticiones cada 15 minutos.

A la vez se realizarán peticiones para obtener los últimos 1 000 tuits de cada cuenta utilizando *statuses/user_timeline*, comprobar cuales de ellos son retuits y guardar en la base de datos las cuentas a las que ha retuiteado. Esto servirá para más tarde poder calcular la

relevancia de las cuentas basándonos en quién retuitea a quién en lugar de en quién sigue a quién. Es importante mencionar que la API de Twitter, en el caso de un retuit, los metadatos no incluyen información de a qué cuenta ha retuiteado directamente, sino de la cuenta que originalmente publicó el tuit. Por ejemplo, dadas tres cuentas de Twitter A, B y C; en el caso de que A publique un tuit t, B retuitee t desde A y a continuación C retuitee t desde B; si se obtiene t desde la cuenta C, este tuit contendrá metadatos que indicarán que su autor original es A, pero no habrá ningún rastro de que ha pasado por B. Como esta información será imposible de conseguir, cada vez que se diga que una cuenta A retuitea a una cuenta B, deberá entenderse que A retuitea un tuit cuyo autor original es B, sin tener en cuenta de qué cuenta lo ha retuiteado realmente.

En total se habían recogido tuits procedentes de 75 502 usuarios, pero dado que algunas cuentas son privadas y no se les pueden consultar los seguidores, el número final quedó reducido a 73 540 usuarios. También hay que tener en cuenta que si una cuenta sigue a mucha gente, no cabrán todos estos amigos en una sola respuesta, por lo que habrá que hacer más de una petición por usuario en estos casos. Esta recolección de información duró más de una semana, pero permite crear un grafo completo de las relaciones entre los 73 540 usuarios.

Además de esto, para las pruebas realizadas la sección 6.3, interesará almacenar más tuits de algunas cuentas para poder analizarlos mejor. Hacer esto con todas las cuentas no sería muy práctico, así que se intentará obtener los tuits de aquellas cuentas más relevantes. Por lo tanto esperaremos a obtener estos tuits hasta el capítulo 6, una vez ya se hayan calculado algunos datos de relevancia.

Capítulo 5

Algoritmos de ordenación

Una vez se han obtenido todos los datos de los usuarios necesarios, se puede empezar a calcular la relevancia. Para ello se usarán los algoritmos HITS y PageRank, que serán explicados a continuación junto con su implementación. Estos algoritmos han sido creados para su uso con páginas web de Internet, siendo los enlaces entre páginas web una parte fundamental de su funcionamiento. Para poder emplearlo en Twitter es necesario definir un elemento equivalente a estos enlaces. Se utilizarán dos métodos distintos:

- relaciones de seguimiento entre cuentas de Twitter, según las cuales un enlace desde una cuenta *A* a una cuenta *B* indica que la cuenta *A* sigue a la cuenta *B*
- relaciones de retuiteo entre cuentas de Twitter, según las cuales un enlace desde una cuenta *A* a una cuenta *B* indica que la cuenta *A* ha retuiteado un tuit publicado por la cuenta *B*

5.1. El algoritmo PageRank

PageRank es un algoritmo creado en 1996 por Larry Page y Sergey Brin. Este algoritmo se usaría dentro de un motor de búsquedas de Internet y su objetivo es el de ordenar las páginas según su relevancia o popularidad.

El algoritmo trata de simular la experiencia de un usuario de Internet, que navega por la red cambiando de página en página siguiendo los enlaces que se encuentre. Se asumirá que este usuario elije de manera aleatoria qué enlace de los que encuentre en una página elegirá para seguir navegando. Los resultados del algoritmo será una distribución de probabilidad que asignará a cada página la probabilidad de que el usuario acabe en ella, valor al que se conoce directamente como PageRank. El algoritmo se basa en los enlaces entre páginas. Intuitivamente se entiende que cuantos más enlazes apunten a una página, más probabilidades habrá de que un usuario acabe en ella y por lo tanto más popular o relevante será. También se puede asumir que cuanto más popular sea una página (y por lo tanto más probabilidades haya de que un usuario pase por ella), más probabilidades habrá de que un usuario

pase por aquellas páginas que esta enlaza. De esta manera, siendo p_i, $1 \leq i \leq N$, una página cualquiera, B_i el conjunto de páginas que enlazan a una página p_i y $PR(i)$ el PageRank de una página p_i, obtendríamos la siguiente fórmula:

$$PR(i) = \sum_{x \in B_i} PR(x) \tag{5.1}$$

También tiene sentido asumir que los enlaces de una página con únicamente un par de ellos tendrán más valor que los de una página con cientos de ellos, por lo que se puede tener en cuenta el número total de enlaces de una página de la siguiente manera, en la que $L(i)$ es el número de enlaces saliente de una página p_i:

$$PR(i) = \sum_{x \in B_i} \frac{PR(x)}{L(x)} \tag{5.2}$$

Además de moverse a base de enlaces, un usuario podría cambiar de web escribiendo la dirección de otra en la barra de direcciones o por algún otro medio independiente de la página en la que se encuentre actualmente. Al igual que con los enlaces, se asumirá que cuando el usuario decida no usar enlaces, la nueva página a la que llegue será elegida de manera aleatoria. Este comportamiento se modelará utilizado un *damping factor* (factor de amortiguamiento) d, que mide la probabilidad de que un usuario haya alcanzado una página por medio de un enlace, siendo $1-d$ la probabilidad de que la haya alcanzado por medio de la barra de direcciones. Así, siendo N el número total de páginas webs contempladas, obtendríamos:

$$PR(i) = \frac{1-d}{N} + d \sum_{x \in B_i} \frac{PR(x)}{L(x)} \tag{5.3}$$

En el caso especial de que una página no tuviese ningún enlace saliente, la única opción del usuario será saltar a otra página al azar, por lo tanto:

$$PR(i) = \begin{cases} \frac{1-d}{N} + d \sum_{x \in B_i} \frac{PR(x)}{L(x)} & \text{si } L(i) > 0 \\ \frac{1}{N} & \text{si } L(i) = 0 \end{cases} \tag{5.4}$$

Para calcular todos los valores PageRank sería necesaria una matriz de adyacencia modificada que se explicará a continuación. Primero empezaremos por una matriz de conectividad G, será una matriz cuadrada de tamaño N en la que cada elemento g_{ij} será 1 si existe un enlace de la página p_i a la página p_j y 0 si no existe dicho enlace.

$$g_{ij} = \begin{cases} 1 & \text{si existe enlace de } p_i \text{ a } p_j \\ 0 & \text{si no existe enlace de } p_i \text{ a } p_j \end{cases} \tag{5.5}$$

A partir de aquí crearemos la matriz de transición del usuario de Internet P, cuyos elementos r se definirán a partir de la fórmula 5.4 como sigue:

$$r_{ij} = \begin{cases} \frac{1-d}{N} + d\frac{g_{ij}}{L(i)} & \text{si } L(i) > 0 \\ \frac{1}{N} & \text{si } L(i) = 0 \end{cases} \tag{5.6}$$

A partir de esta matriz de transiciones, el vector de valores PageRank se puede calcular de distintas maneras [19]. En nuestro caso se usará el método iterativo de la potencia. Este método consiste en multiplicar la matriz un vector inicial x^0 y volviendo a multiplicar la matriz con el resultado de manera iterativa, como se muestra en el algoritmo 1.

Algoritmo 1 Computación de vector PageRank siguiendo el método de las potencias

Entrada: Matriz P, precisión ε, vector x^0 con valores inicializados a $\frac{1}{N}$, $i = 0$

1: **repetir**
2: $\quad x^{i+1} = Px^i$
3: $\quad i = i + 1$
4: **hasta que** $|x^i - x^{i-1}| < \varepsilon$
5: **devolver** x^i

Cuando el vector x converja se habrá dado con el vector de valores PageRank 5.7.

$$\mathbf{x} = \begin{bmatrix} PR(1) \\ PR(2) \\ \vdots \\ PR(N) \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{,N} \\ r_{21} & \ddots & & \vdots \\ \vdots & & r_{ij} & \\ r_{N1} & \cdots & & r_{NN} \end{bmatrix} \mathbf{x} \tag{5.7}$$

5.2. Implementación del algoritmo

La interpretación del usuario de Internet que viaja de manera aleatoria siguiendo los enlaces que encuentra en la web no se puede aplicar para el caso de Twitter, pero tampoco es necesario entenderlo de esta manera. Bastará con entender la equivalencia de la relación de una cuenta de Twitter que sigue a otra con la de un enlace que apunta de una página de Internet a otra.

Primero se creará la matriz de transiciones. De la base de datos se obtiene las listas de amigos de cada uno de los usuarios y se itera sobre ellas para crear la matriz usando el algoritmo 2, en el que la columna i de una matriz P se denota como $P_{\bullet,i}$. De esta manera se crea una matriz que coincide con la fórmula 5.6.

La matriz creada estará formada por $73\,540^2$ elementos, lo que es una cantidad importante. No es posible guardarla como una matriz dispersa, ya que, debido al factor de amor-

Algoritmo 2 Creación de la matriz de transiciones para PageRank

Entrada: Matriz P, número de cuentas N, factor de amortiguamiento d, lista de *usuarios* con sus *amigos*

para todo *usuario* **hacer**
  $n = 0$
  para todo *amigo* de *usuario* **hacer**
    $P_{amigo,usuario} = d$
    $n = n + 1$
  fin para
  si $n > 0$ **entonces**
    $P_{\bullet,usuario} = P_{\bullet,usuario}/n$
    $P_{\bullet,usuario} = P_{\bullet,usuario} + \frac{1-d}{N}N$
  si no
    $P_{\bullet,usuario} = \frac{1}{N}$
  fin si
fin para
devolver P

tiguamiento, no existen elementos nulos. El ordenador usado para almacenar todos los tuits dispone únicamente de 8 gigas de RAM, que demostró no ser suficiente. Se usó la librería *pickle* de Python para almacenar la información de los usuarios necesaria y poder transportarla fácilmente a otro equipo con mayor memoria. Aquí fue probada la gran portabilidad de Python. A pesar de ello, un equipo de 64 gigas de RAM tampoco fue capaz de almacenar toda la matriz en memoria, dado esto se decidió usar la librería *PyTables* para poder almacenar datos en el disco duro en lugar de en la memoria RAM.

Puesto que en ningún momento (ni en la creación de la matriz ni durante la computación del PageRank) se necesita acceder a varias columnas distintas a la vez, se ha podido dividir la matriz en bloques de varias columnas, que permanecen almacenados en disco duro mientras no sean usados. Para calcular el vector de valores PageRank según el algoritmo 1 se ha multiplicado cada bloque por separado, usano las funciones de la librería *NumPy*.

Usando la matriz de retuits

Este proceso es casi idéntico al del PageRank usando la red de seguidores. En lugar de obtener la lista de los amigos de cada usuario de la base de datos, se obtendrá la lista de las cuentas a las que ha retuiteado. Serán estas listas las que se usarán en la creación de la matriz, de igual manera que se usaron las de los amigos de los seguidores. Una vez creada la matriz, el cómputo de los valores es idéntico. Es por esto que, a pesar de que al inicio se creó una función distinta para cada caso, al final se unieron ambas funcionalidades en una sola función.

Ejecución

Le ejecución transcurrió de manera muy similar usando las dos matrices distintas. Para calcular los PageRank de todos los usuarios se utilizaron 100 bloques de PyTables (de como mucho 736x73 540 elementos), lo que permitió usar únicamente 6 gigas de memoria RAM. Una vez creada la matriz, los vectores convergieron hasta una precisión de 0,00001 tras 72 y 85 iteraciones respectivamente, lo que en ambos casos duró 4 horas ± 15 minutos. Listas de las 30 cuentas más relevantes según estos valores PageRank se pueden encontrar en el apéndice.

5.3. El algoritmo HITS

El algoritmo HITS (del inglés: *Hypertext Induced Topic Selection*) también permite evaluar la relevancia de elementos. Este algoritmo, diseñado por Jon Kleinberg, fue ideado para analizar la importancia de páginas webs durante los primeros años de vida de Internet. Las páginas parecián dividirse en dos funciones distintas: unas son las creadoras de contenido y otras cuya función es compilar enlaces a estas webs creadores de contenidos. Por lo tanto, en lugar de asignar una sola puntuación a cada página, como en el caso de PageRank, se definirán dos puntuaciones distintas; el primero, *authority* o *auth*, indicará el valor de la página como creadora de contenido, mientras que el segundo, *hub*, indicará su valor como colección de enlaces a otras páginas. Estos valores se calcularán por medio de los enlaces entre páginas como con el algoritmo PageRank. Una importante diferencia entre ambos algoritmos es que HITS atribuye a las páginas importancia, no solo según el número y calidad de los enlaces entrantes, si no contemplando también a los enlaces salientes.

Las puntuaciones authority y hub se definen en términos de recursión mutua: la puntuación *authority* de una página es la suma de las puntuaciones *hub* de todas las páginas que enlacen a ella y la puntuación *hu*b de una página es la suma de todas las puntuaciones *authority* de todas las páginas a las que enlace. Es decir:

$$\begin{aligned} auth(p_i) &= \sum_{j \in B_i} hub(p_j) \\ hub(p_i) &= \sum_{j \in O_i} auth(p_j) \end{aligned} \tag{5.8}$$

donde B_i es el conjunto de páginas que enlazan a la página p_i y O_i es el conjunto de páginas que p_i enlaza.

Para calcular los valores primero se inicializarán todos hub y auth a 1. A continuación se iterarán las siguientes instrucciones hasta que los valores convergan:

- Se actualizan todos los valores *auth* a partir de los valores *hub* según la fórmula 5.8.
- Se actualizan todos los valores *hub* a partir de los valores *auth* según la fórmula 5.8.

- Los valores *hub* se normalizan dividiéndolos por la raíz cuadrada de la suma de los cuadrados de todos los valores *hub* y los valores *auth* se normalizan dividiéndolos por la raíz cuadrada de la suma de los cuadrados de todos los valores *auth*.

5.4. Implementación del algoritmo

Al igual que con el algoritmo PageRank, se probará HITS tanto usando la información de los seguidores, como la de direcciones de los retuits. En el primer caso se espera obtener puntuaciones *authority* altas en las cuentas que tengan muchos seguidores y puntuaciones *hub* altas en aquellas con muchos amigos. En el segundo caso serán aquellas cuentas que sean retuiteadas a menudo las que se espera que tengan mayor puntuación *authority* y la puntuación *hub* será mejor en las que más retuiteen.

Primero se crean listas de enlaces entrantes y salientes para cada nodo a partir de las listas de amigos o cuentas retuiteadas. A partir de estas listas se pueden ir calculando los valores *authority* y *hub* de cada página iterando las instrucciones explicadas anteriormente. Como condición de parada se comprobará que la norma de la diferencia entre los valores de una iteración y los valores de la anterior sea menor que una variable ε.

Para calcular las puntuaciones HITS de todas las cuentas se ha usado un $\varepsilon = 0{,}00001$. La computación ha durado apenas 3 minutos en ambos casos, convergiendo hasta una precisión ε tras 20 o 19 iteraciones. El algoritmo es mucho más eficiente en cuanto a espacio que PageRank, ya que no requiere utilizar ninguna matriz. En el apéndice se pueden ver las listas de los 30 usuarios más relevantes de acuerdo a cada puntuación.

Capítulo 6

Evaluación de los resultados

A continuación se analizan de distintas maneras los resultados ofrecidos por los algoritmos ya comentados y se tratará de sacar alguna conclusión de este tema que en principio es tan subjetivo.

6.1. Coeficiente de correlación de Spearman

Primero se comparan las ordenaciones que proporcionan los distintos algoritmos, comparándose también con una medida más obvia como el número de seguidores de las cuentas. Si el orden proporcionado por el algoritmo de PageRank fuese similar al orden por números de seguidores, todo este análisis no habría tenido mucho sentido. Para comparar las distintas ordenaciones se usará el coeficiente de correlación de Spearman. Este coeficiente se utiliza para calcular la correlación entre dos variables aleatorias continuas. Estas variables aleatorias serán en nuestro caso las ordenaciones que se quieran comparar.

La fórmula del coeficiente de correlación Spearman es la siguiente:

$$\rho = 1 - \frac{6 \sum\limits_{1 \leq i \leq N} d_i^2}{N(N^2 - 1)} \tag{6.1}$$

donde d_i es la diferencia de las posiciones del elemento i en ambas listas.

El valor resultante se encontrará entre 1 y -1, valores positivos indican una correlación directa (cuanto más alto se encuentre un valor en una lista, más alto se encontrará tambien en la otra), mientras que valores negativos indican una correlación inversa (cuanto más alto se encuentre un valor en una lista, más bajo se encontrará en la otra). Valores cercanos a 0 indican que no existe mucha correlación entre ambas listas.

En nuestro caso primero será necesario obtener la posición de cada cuenta según los distintos ordenes, para lo que se ha creado una función que ordenará las cuentas según los distintos resultados y asignará un atributo a cada cuenta con su posición. Una vez hecho esto

se puede ejecutar una sencilla función que implemente el algoritmo. Comparando las puntuaciones de los distintos algoritmos y los números de seguidores obtenemos los siguientes resultados:

Cuadro 6.1: Resultados coeficiente de correlación Spearman

	PR	PR-RTs	HITS (*auth*)	HITS (*hub*)	HITS-RTs (*auth*)	HITS-RTs (*hub*)
Seguidores	0.740	0.531	0.533	0.260	0.455	0.077
PR		0.701	0.780	0.532	0.673	0.338
PR-RTs			0.675	0.495	0.885	0.414
HITS (*auth*)				0.835	0.796	0.641
HITS (*hub*)					0.650	0.740
HITS-RTs (*auth*)						0.572

Como se puede comprobar, todos los coeficientes de correlación son positivos, por lo que se puede asumir que existe una relación positiva entre todas las métricas. En algunos casos esta relación es bastante intuitiva. Por ejemplo, si nos centramos en los coeficientes que comparan las distintas métricas con los números de seguidores, es de esperar que por lo general cuentas con un gran número de usuarios tengan números altos de PageRank y de *authority* según HITS, tanto usando el grafo de seguidores (en este caso la relación es trivial) como usando el grafo de retuits (usuarios con gran número de seguidores son retuiteados por más usuarios). Y es en estos casos cuando los ordenes proporcionados por estas métricas más se parecen al proporcionado según el número de seguidores, se aprecia también que las métricas basadas en los retuits difieren más del orden de los seguidores que las basadas en los seguidores.

Las listas ordenadas según las puntuaciones *hub* del algoritmo HITS son las que más se diferencian, llegando casi hasta no mostrar ninguna relación en el caso de la basada en retuits (0.077). Esto es de esperar, ya que no tendría por qué haber ninguna relación directa entre el número de seguidores y el número de cuentas a las que sigue o a las que retuitea una cuenta. Sin embargo la ligera tendencia que se observa (puesto que los coeficientes son positivos) también se puede explicar: si calculamos la media del ratio amigos/seguidores obtenemos una media de 1.1306 con una desviación estandar de 1.4232. Por ello se puede afirmar que existe una relación entre el número de amigos y el de seguidores en una cuenta, que explicaría el valor positivo en la puntuación *hub* según los seguidores. En el caso basado en retuits también se puede explicar argumentando que las cuentas con más seguidores serán por lo general más activas y retuitearán más, pero el coeficiente de correlación es tan pequeño que no merece la pena profundizar mucho más.

Además de los coeficientes que se comparan con el orden según los seguidores, los que comparan el resto de métricas también tienen bastante sentido: el PageRank basado en seguidores es similar a la puntuación *authority* del algoritmo HITS basado en seguidores, lo mismo ocurre con los mismos algoritmos basados en retuits, ambas puntuaciones *hub* se

parecen, etc.

En conclusión, estos resultados han permitido comprobar dos cosas: los algoritmos se han comportado de la manera esperada. Resultados inesperados podrían indicar que hay un error en el código. Los resultados de las métricas difieren de aquellos proporcionados por métricas más triviales como es el caso del número de seguidores. Por lo tanto el uso de estos algoritmos no ha sido en vano. Aunque falta demostrar si realmente ofrecen alguna mejoría en cuanto a la relevancia. Este problema trataremos de resolverlo a continuación.

6.2. Ganancia Acumulada Descontada (*DCG*)

Para realizar un análisis de la calidad de los resultados según cada algoritmo una buena métrica de evaluación es la Ganancia Acumulada Descontada (*DCG*, del inglés: *Discounted Cumulative Gain*), que calcula la utilidad de una lista ordenada de elementos según la relevancia de estos. Para ello cada elemento tiene que estar etiquetado a priori con un valor numérico que indique su relevancia. Una versión más primitiva de esta fórmula es la ganancia acumulada, que consiste en sumar directamente los valores de relevancia de cada elemento obtenido. Esta fórmula no tiene en consideración la posición de cada elemento, pero sirve para calcular la utilidad de un resultado cuando únicamente una fracción del número total de elementos existentes es devuelta. Se asume que esta utilidad está directamende ligada con la relevancia de cada documento.

$$CG = \sum_{i=1}^{N} rel_i \tag{6.2}$$

Si asumimos que la utilidad también depende de la posición de cada elemento en la lista (es decir, cuanto antes en la lista aparezca un documento relevante, más útil será), se puede utilizar la fórmula de Ganancia Acumulada Descontada, en la que se penaliza de manera logarítmica la posición de cada elemento.

$$DCG = rel_1 + \sum_{i=2}^{N} \frac{rel_i}{\log_2(i)} \tag{6.3}$$

Los resultados de esta fórmula se pueden normalizar para que su valor se encuentre entre 1 y 0. Esto se hará dividiendo el resultado DCG por el resultado ideal (*IDCG*). Este resultado se obtiene al aplicar la fórmula DCG a una lista de los documentos ordenados según su relevancia. El DCG normalizado (*NDCG*) servirá para comparar distintas listas cuando contengan los mismos elementos y difiera únicamente su orden, ya que si los elementos son distintos en cada lista, el valor IDCG podría diferir también.

$$NDCG = \frac{DCG}{IDCG} \tag{6.4}$$

Para realizar cualquiera de estas fórmulas son necesarias unas mediciones de relevancia de las cuentas con las que poder comparar los resultados de nuestros algoritmos. Lo ideal sería utilizar colecciones para test o benchmark con elementos ya marcados con su relevancia. Estos tipos de benchmark existen con documentos de muchas áreas distintas, como los ofrecidos por la Text Retrieval Conference (TREC), pero no existe ninguno enfocado a la relevancia de cuentas en Twitter. La única opción es etiquetar manualmente las cuentas que hayamos considerado.

Etiquetar manualmente las 73 540 cuentas sería un trabajo demasiado laborioso y quizás innecesario. Puesto que en ordenaciones de relevancia lo más importante son los primeros resultados, debería de bastar con analizar los primeros resultados de cada algoritmo. Por lo tanto etiquetaremos únicamente las 100 cuentas con mayor puntuación según cada una de las métricas.

En lugar de asignar directamente un valor de relevancia a cada cuenta se ha decidido clasificar las cuentas en distintas categoría y asignar a posteriori a cada categoría una relevancia. El objetivo de esta decisión es hacer más sencilla (no es necesario conocer cuan relevante es realmente cada usuario, sólo a qué categoría pertenece) la tarea de clasificación, así como más imparcial (estas categorías, aunque también distan bastante de ser completamente objetivas, sí que lo son bastante más que un valor de relevancia arbitrario). Por supuesto, como son categorías bastantes generales, en cada una habrá usuarios que realmente son muy influyentes y otros que no lo son en absoluto. Las distintas categorías y sus relevancias asignadas serán las siguientes:

- **Políticos** - Partidos políticos, políticos, entidades del gobierno, etc. (Relevancia: 4)
- **Medios de comunicación (MC) especializados en política** - Programas de televisión o periodistas especializados en política o economía, etc. (Relevancia: 4)
- **Otros MC** - Medios de comunicación generalistas o especializados en otras áreas. (Relevancia: 3)
- **Personalidad** - Famosos y celebridades de distintas índoles. (Relevancia: 2)
- **Personalidad en Twitter** - Cuentas de Twitter muy influyentes, gran ratio seguidores/amigos, pero poca influencia fuera de Twitter (Relevancia: 2)
- **Otros** - Cualquier usuario corriente de Twitter sin demasiada influencia (Relevancia: 1)
- **Irrelevante** - Cuentas de spam, otros países, etc. Cuentas que con muy poca probabilidad vayan a hablar de política española. (Relevancia: 0)

Para etiquetar las cuentas se ha decidido crear una aplicación online en Django, para hacer más sencillo el acceso a las páginas de Twitter de las cuentas, desde las que se obtiene

la información necesaria para clasificar la cuenta o al menos para continuar la investigación en la web. En muchos casos la clasificación ha sido arbitraria, ya que la línea de separación entre campos como medios de comunicación especializados y medios de comunicación generales o entre cuentas de Twitter corrientes y cuentas de Twitter muy influyentes puede ser algo difusa; por lo que hay que tomar estos resultados como medidas aproximadas, cosa que siempre será el caso si se trata de medidas subjetivas como la relevancia.

En total han sido 485 cuentas las clasificadas. La distribución según la categoría ha sido la siguiente:

Cuadro 6.2: Distribución de cuentas clasificadas

Políticos	40
MC política	70
Otros MC	78
Personalidades	7
Personalidades Twitter	32
Otros	211
Irrelevante	47

Estos resultados podrán compararse usando CG o DCG, ya que son una pequeña muestra ordenada de la colección total de cuentas. En cambio para calcular el NDCG sería necesario que los resultados devueltos fuesen los mismos. Para poder calcular los valores normalizados, podemos usar como resultado, en vez de las listas ordenadas de las 100 cuentas más relevantes según cada métrica, la lista de todas las cuentas etiquetadas ordenadas según cada métrica.

Cuadro 6.3: Resultados de Ganancia Acumulada Descontada

	CG	DCG	NDCG
PR	351	76.933	0.965
PR-RTs	309	69.429	0.957
HITS (*auth*)	296	68.038	0.949
HITS-RTs (*auth*)	283	61.504	0.924
Seguidores	172	36.210	0.855
HITS (*hub*)	137	31.884	0.834
HITS-RTs (*hub*)	99	20.788	0.797

CG y DCG calculados a partir de las primeras 100 cuentas de cada lista, NDCG calculado a partir de la lista de todas las cuentas clasificadas ordenada según cada métrica.

En estos resultados se puede ver que, según tanto CG como DCG como NDCG, existe un claro orden en la calidad de la ordenación de cada algoritmo o métrica:

$$PageRank > PageRank\text{-}Retuits > HITS(auth) > HITS\text{-}Retuits(auth) > Seguidores$$

Cualquiera de los algoritmos ofrece mejores resultados que la simple ordenación por seguidores, excepto las puntuaciones *hub*, pero esto no debería ser sorpresa, ya que ellas no aspiran a indicar los elementos más influyentes. De ahora en adelante no se tendrán en cuenta las puntuaciones *hub*.

6.3. Análisis de representatividad de polaridad de los tuits

Asumiendo que una muestra de los elementos más relevantes de una colección son representativos del contenido de toda la colección, es posible realizar otra prueba para evaluar la calidad de las métricas de relevancia. Esta prueba consistirá en, primero, analizar la polaridad media de toda la colección de tuits recogidos; a continuación se volverá a analizar esta polaridad media, pero únicamente de los tuits pertenecientes a los cuentas más relevantes según alguna de estas métricas y se compararán los resultados.

Será necesario un detector de polaridad, para ello se ha usado uno desarrollado en la Universitat Politècnica de València por F. Pla y L. Hurtado y que ya ha sido utilizado en varios estudios [20][11][21].

Antes de comenzar, probaremos el detector para comprobar que funcione correctamente. Para ello utilizaremos las 100 cuentas más relevantes según su puntuación PageRank. Estas cuentas se etiquetarán con el partido político con el que estén afiliadas, en el caso de que sean cuentas de políticos o partidos políticos. Se espera que estas cuentas muestren una mejor polaridad con respecto a su propio partido que al resto. En total 23 de las cuentas han sido etiquetadas.

A continuación se obtendrán los 1 000 tuits más recientes de cada una de estas cuentas para tener una muestra más amplia de cada cuenta y poder predecir su polaridad con mayor precisión. El detector de polaridad analizará cada uno de esos tuits, asignándo a cada uno una etiqueta de polaridad. A estas etiquetas de polaridad les asignaremos una puntuación para posteriormente poder realizar la media:

- **PP** - Muy positivo (Puntuación: 2)
- **P** - Positivo (Puntuación: 1)
- **NEU** - Neutral (Puntuación: 0)
- **N** - Negativo (Puntuación: -1)
- **NN** - Muy negativo (Puntuación: -2)
- **NONE** - Ninguna polaridad, puede confundirse con NEU (Puntuación: 0)

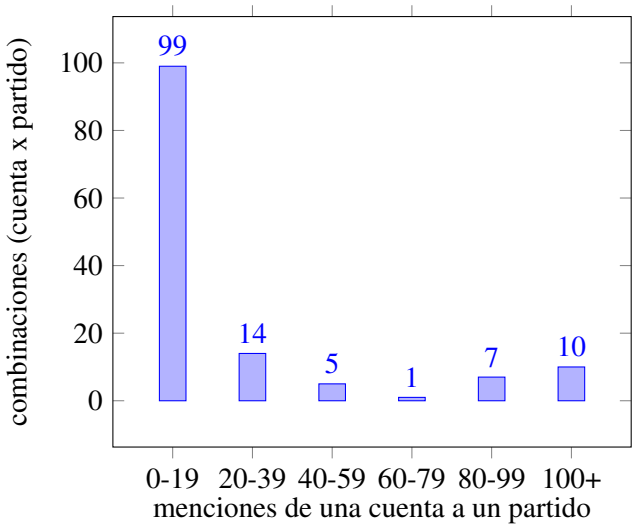
Usando expresiones regulares se analizarán todos los tuits obtenidos y se etiquetarán también por las menciones a los distintos partidos políticos. En este caso se ha buscado también por el partido Izquierda Unida. En la obtención inical de tuits, no había suficientes menciones del partido como para justificar su inclusión en esa búsqueda, pero en este caso varias de las cuentas etiquetadas pertenecen al partido Izquierda Unida, por lo que merece la pena incluírlo. Se buscarán tanto los nombres enteros de los partidos como sus siglas. De esta manera sabremos qué partido, o partidos, menciona cada tuit.

Entonces, siendo $M(i)$ el conjunto de tuits que mencionan al partido i, T_a el conjunto de tuits escritos por la cuenta a y $p(t)$ la puntuación asignada por el detector de polaridad al tuit t, la media de la polaridad $P_a(i)$ de la cuenta a con respecto al partido i se calcula de la siguinte manera:

$$P_a(i) = \frac{\sum_{t \in M_i \cap T_a} p(t)}{|M_i \cap T_a|} \tag{6.5}$$

Calculado esto para todas las cuentas etiquetadas, obtenemos una serie de puntuaciones por cada cuenta y cada partido. El objetivo será que la puntuación mayor de cada cuenta sea la del partido político al que está afiliado. Los resultados se encuentran en la tabla 6.4.

Se puede observar en estos resultados, que el detector funciona de manera aceptable. Únicamente 5 cuentas de 23 han sido incorrectamente interpretadas. Este error probablemente se deba en la mayoría de los casos al pequeño tamaño de la muestra, ya que el número de menciones que hacía una cuenta a cada partido por lo general es muy bajo, como se puede ver en la siguiente gráfica:



Los resultados son aceptables, así que emplearemos el detector de polaridad para poner

Cuadro 6.4: Resultados prueba de detector de polaridad

	Esperado	PP	PSOE	Podemos	Cs	UPyD	IU
ahorapodemos	Podemos	-0.29	-0.45	0.6	0	-	-
agarzon	IU	-0.68	-0.69	0.45	0.67	-	-
PSOE	PSOE	-0.15	0.69	0	0	-	-
sanzhezcastejon	PSOE	-0.35	0.84	-	-	-	-
GLlamazares	IU	-0.28	-0.46	-0.45	-0.33	-	0.5
Albert_Rivera	Cs	-0.04	-0.15	-0.1	0.37	-0.33	-
PPopular	PP	0.53	-0.68	0.22	0.5	-	-
ierrejon	Podemos	-0.54	-0.4	0.6	-1	-	-
cayo_lara	IU	-0.1	-0.83	0.5	-	-	0
JoseantonioJun	PSOE	-1.09	-0.5	-1.2	-1.17	-	-
MonederoJC	Podemos	-0.96	-0.21	0.07	-1.5	-	-
UPyD	UPyD	-0.96	-1.08	0.67	-	0.37	-
Pedro_Zerolo	PSOE	-0.57	0.72	0.33	-	-	-
Tonicanto1	UPyD	-0.09	-1.14	0.13	-0.29	0.43	-
iunida	IU	-0.52	-0.75	-1	-	-	0.63
ccifuentes	PP	1.11	0	0.5	0.5	-	-
CiudadanosCs	Cs	-0.64	-0.56	0.4	0.96	-	-
pnique	Podemos	0.11	0.08	0.55	-0.6	-	-
TeresaRodr_	Podemos	-0.15	-0.28	0.67	-	-	-
mdcospedal	PP	0.77	-1.09	1.5	-	-	-
LuisSalvador	Cs	-0.16	-0.08	-0.17	1.17	-	-
lozanoirene	UPyD	-0.53	-1.17	-	0	0.66	-
sevillajordi	PSOE	-0.6	0	-0.73	0.25	-	-

En la tabla se han marcado la mejor polaridad de cada cuenta. Cuando esta polaridad no coincide con la esperada, se ha marcado en rojo. No se han tenido en cuenta y se han indicado con "-" los casos en los que un partido era mencionado menos de dos veces.

a prueba la utilidad de nuestras puntuaciones de relevancia por medio de un posible uso práctico. Si tiene éxito, significará que para obtener una idea general de la opinión de los usuarios de Twitter sobre un tema, en lugar de analizar las opiniones de todos los usuarios, bastaría con analizar la de una pequeña muestra de ellos, elegida según una puntuación de relevancia y que sería más representativa de la opinión general cuanto mejor fuese la métrica de relevancia usada.

Tras obtener las puntuaciones de polaridad de toda la red de cuentas obtenidas y posteriormente las de las cuentas más significativas según cada métrica por separado, se calcula la media de la diferencia entre las obtenidas de las muestras parciales y de la colección completa. Para simplificar la comparación, se combinarán las etiquetas de polaridad en 3 grupos:

Las positivas y muy positivas bajo el grupo general positivo *p*, las de polaridad neutral y las que no muestren polaridad bajo el grupo general neutral *neu*, y las negativas y muy negativas bajo el grupo general negativo *n*.

La siguiente ecuación detalla la obtención de esta media, en la que $\{p;n;neu\}$ es el conjunto de grupos de etiquetas; P es el conjunto de los partidos políticos; $pol_{ai}(et)$ es la cantidad de tuits con polaridad et que mencionan al partido político i encontrados en la muestra de tuits a; I es la colección completa de tuits, que proporcionará la distribución de polaridad ideal que se tratará de aproximar con las otras muestras; $dif_a(i)$ es la diferencia media calculada de la polaridad entre la muestra a y la ideal con respecto al partido político i; y dif_a es la media de las diferencias medias calculadas de la polaridad entre la muestra a y la ideal para todos los partidos políticos.

$$\begin{aligned} dif_a(i) &= \sqrt{\sum_{et\in\{p;n;neu\}} \left(\frac{pol_{Ii}(et)}{|I|} - \frac{pol_{ai}(et)}{|a|}\right)^2} \\ dif_a &= \sqrt{\sum_{i\in\{P\}} dif_a(i)} \end{aligned} \tag{6.6}$$

Usando muestras de tanto los 100 como los 1 000 usuarios más relevantes, se obtienen los siguientes resultados:

Cuadro 6.5: Diferencia en las polaridades medidas en cada muestra con respecto a las polaridades de la colección completa

Tamaño muestra	100	1000
Seguidores	0.086	0.089
PageRank	0.08	0.135
PageRank-RTs	0.061	0.108
HITS (*auth*)	0.105	0.077
HITS-RTs (*auth*)	0.08	0.115

Las diferencias con respecto a la polaridad ideal son similares para todas las muestras y no se puede determinar que técnica es mejor para elegir la muestra, ya que también depende de la cantidad de la muestra. Para una muestra de 1 000, la mejor aproximación se obtiene con el algoritmo HITS según el grafo de seguidores (0.077), mientras que este algoritmo ofrece el peor resultado cuando se obtiene una muestra de 100 cuentas (0.105). En esta caso el mejor algoritmo es el de PageRank según el grafo de retuits (0.061).

Dados estos resultados no se puede concluir que la puntuaciones de relevancia implementadas sirvan para poder obtener una muestra más representativa del total de usuarios de Twitter. Esto puede ser debido a múltiples motivos: por ejemplo a que la representatividad y la relevancia de una muestra puedan no ser variables relacionadas en este caso, a los fallos de precisión del detector de polaridades, etc.

Uno de los motivos por los que se decidió usar esta muestra de tuits, de política española en tiempos de campaña electoral, fue para poder ser luego comparada con los resultados de encuestas y sondeos políticos realizados por distintas organizaciones. A la polaridad obtenida por cada cuenta se le asignaría un peso calculado a partir de su relevancia, de esta manera las opiniones de cuentas relevantes tendrían más importancia al calcular la opinión media. Se esperaba que esta opinión general ponderada se aproximase más a la opinión general observada en los resultados de los distintos sondeos que la no ponderada. Se parte por lo tanto de que los usuarios de Twitter no son a priori una muestra representativa de la opinión pública [15]. Sin embargo este enfoque tiene bastantes problemas, ya que no hay ninguna razón para asumir que las cuentas que sean consideradas más relevantes vayan a ser más representativas que el resto. Por lo tanto se decidió abandonar este enfoque.

Capítulo 7

Análisis de clústeres

En este capítulo se explicará cómo se han analizado las cuentas de Twitter obtenidas por medio de la creación de clústeres. Se ha estudiado la relevancia y la alineación política (polaridades con respecto a cada partido) de las cuentas dentro de cada clúster para comprobar si se puede usar esta técnica a la hora de considerar opiniones. Los clústeres se han creado por medio del algoritmo *k*-medoids, que será explicado más adelante. Para este algoritmo es necesaria una función o matriz de distancias, se utilizarán dos distintas: una matriz creada a partir del grafo de seguidores y otra creada a partir del grafo de retuits.

7.1. Estudio de la conectividad

Para conseguir las matrices de distancia a partir de los grafos de seguidores o retuits, será recomendable partir de un grafo conexo, es decir, que para cualquier par de vértices a y b exista un camino de a hasta b o de b hasta a. Si el grafo no es conexo, podría haber pequeños grupos de elementos separados del resto (o elementos complétament solos). Estos componentes tendrían una distancia infinita con respecto a los elementos de otros componentes del grafo, lo que haría necesario que, a la hora de crear clústeres, se tuviese que asignar un clúster a cada uno de ellos. En el algoritmo *k*-medoids el número de clústeres usado es determinado a priori y su número aumenta el coste de computación del algoritmo, por ello es importante ahorrar clústeres cuando sea posible y evitar componentes del grafo inconexos.

Las matrices usadas para el algoritmo PageRank, al no tener ningún elemento nulo, también definían un grafo conexo, según el cual siempre existía al menos una pequeña posibilidad de saltar desde cualquier página a cualquier otra página. Pero en este caso se debe usar otro enfoque, ya que no se busca crear una matriz estocástica.

Basándonos en el modelo $G(n,p)$ de generación de grafos de Erdös-Rényi [7], se puede predecir la existencia de un único componente gigante en el que se encuentren conectadas la mayoría de las cuentas. Este modelo se usa para la generación de grafos aleatorios y, si bien es cierto que existen modelos más apropiados para redes sociales como Twitter (por ejem-

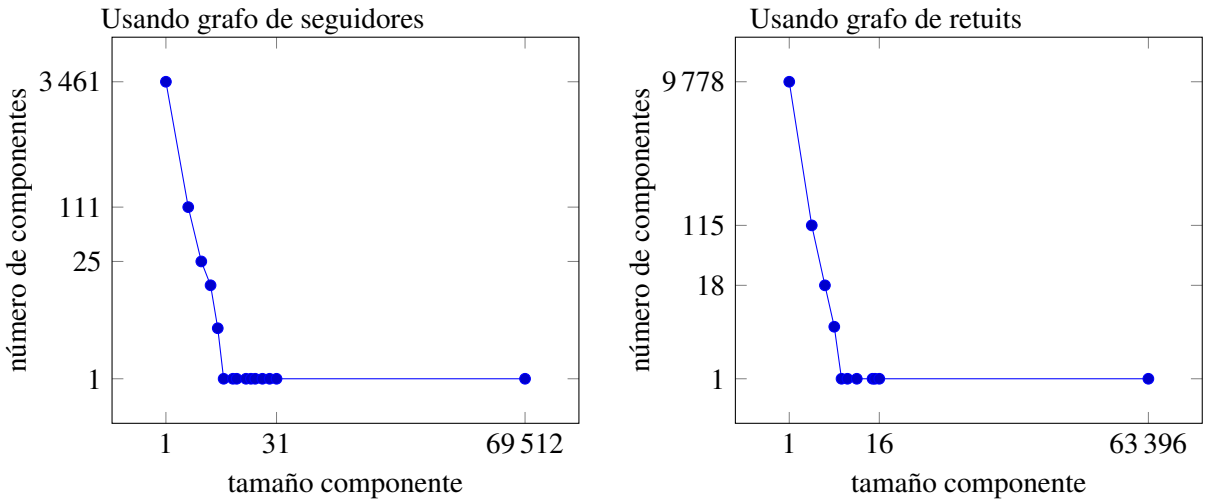
plo el modelo Barabási–Albert o el Watts–Strogatz), este modelo ofrece una aproximación aceptable, pues solo nos interesan los aspectos relacionados con un componente gigante, lo cual es una conclusión importante del modelo Erdös-Rényi. Según este, siendo n el número de nodos del grafo y p la probabilidad de entre dos nodos cualesquiera exista un vértice, si $np > 1$ (o lo que sería equivalente: si existe de media más de un vértice por nodo, lo cual es cierto en nuestro caso) entonces es altamente probable que exista un único componente gigante y que ningún otro componente tendrá un tamaño mayor que $log(n)$. Esta hipótesis se pondrá a prueba en nuestros grafos al analizar su conectividad.

Dada una matriz booleana M que indique los vértices existentes entre cada nodo sean en la dirección que sean (la dirección de los vértices no es relevante, ya que interesa obtener componentes conexos, no es necesario que sean fuertemente conexos), se utilizará un algoritmo iterativo para obtener los distintos componentes de nuestro grafo.

En los gráficos 7.1 se pueden apreciar los tamaños de los distintos componentes tanto del grafo de seguidores como del de retuits. Ambas topologías parecen ser muy similares. Se aprecia claramente que tienen un único componente gigante (compuesto por 69 512 elementos en el caso de los seguidores y por 63 396 elementos en el caso de los retuits), lo que corroborá lo predicho según el modelo Erdös-Rényi. Sin embargo el mayor componente no gigante está compuesto por 31 o 16 elementos según el grafo, que es mayor que el previsto límite, cuyo valor es $log(n)$ (en nuestro caso 4).

Estos componentes son lo suficientemente pequeños como para poder ser ignorados, puesto que se puede asumir que las cuentas pertenecientes a estos componentes no tendrán mucha relevancia al no tener ninguna relación con elementos del componente principal. Ahora se pasará a la creación de clústeres dentro de estos componentes principales conexos.

Figura 7.1: Tamaño de los componentes conexos de los grafos



7.2. Algoritmo de Floyd-Warshall

Será necesario definir una función de distancias entre elementos para poder calcular los clústeres. Puesto que las cuentas de Twitter no se encuentran en un espacio euclídeo definido por algún número de dimensiones, ha sido necesario utilizar un método menos convencional. Se han creado matrices de distancias, que indican la distancia entre dos elementos cualesquiera, a partir de las matrices de transición que indiquen los enlaces ya existentes entre dos cuentas, tanto en el caso de los seguidores como en el de los retuits.

Puesto que muchas cuentas no están directamente conectadas, la distancia entre ellas inicalmente será infinita. Pero, si utilizamos únicamente aquellas que se ha demostrado que forman un componente conexo, se podrá llegar de cualquier cuenta a cualquier otra saltando por distintos enlaces y creando un camino entre ellas. La distancia entre distintas cuentas será entonces la distancia del camino más corto entre ellas.

Para calcular esta distancia de los caminos más cortos se ha usado el algoritmo de Floyd-Warshall, que parte de la matriz de las distancias de las transiciones inciales entre los distintos elementos. Estas transiciones iniciales tendrán un valor infinito cuando no exista conexión entre los nodos y en caso contrario, un valor correspondiente a la distancia entre ellos que se definirá más adelante. Cabe destacar que el algoritmo de Floyd-Warshall devuelve únicamente las distancias de los caminos más cortos, no los caminos en sí, pero para nuestro objetivo bastará con esto. A continuación se muestra el algoritmo:

Algoritmo 3 Algoritmo de Floyd-Warshall

Entrada: Matriz de transiciones iniciales D, con valores d_{ij} $(1 \leq i \leq n, 1 \leq j \leq n)$

```
1: para todo 1 ≤ i ≤ n hacer
2:     para todo 1 ≤ j ≤ n hacer
3:         para todo 1 ≤ k ≤ n hacer
4:             si d_jk > d_ji + d_ik entonces
5:                 d_jk = d_ji + d_ik
6:             fin si
7:         fin para
8:     fin para
9: fin para
```

Mantener la dirección de los enlaces complicaría el cálculo de los clústeres y haría muy difícil su interpretación, por lo que se ha decidido convertir todos los enlaces en bidireccionales. Para poder expresar la diferencia entre aquellas relaciones que originalmente eran bidireccionales y las que no (por ejemplo, la diferencia entre la relación de dos cuentas que se siguen mutuamente y la de una cuenta que sigue a otra, pero no en el caso inverso) se les asignarán valores de distancia distintos a ambas relaciones. Cuando la relación fuese bidireccional, se le asignará a esta una distancia de 0,5, mientras que si era unidireccional se le asignará 1. De esta manera un par de cuentas estarán más próximas entre ellas si su relación

es mutua (ya sea que ambas se siguen mutuamente o que ambas se retuitean mutuamente).

Definidas estas distancias, las matrices de transiciones iniciales son simétricas, por lo que, con realizar unas pequeñas modificaciones al algoritmo 3, será suficente recorrer únicamente la mitad de la matriz.

7.3. Algoritmo k-medoids

Como ya se ha dicho, se utilizará al algoritmo de clusterización k-medoids, el cual está inspirado en el algoritmo k-means. k-means es uno de los algoritmos de clusterización más populares, el cual divide una colección de n elementos en k clústeres (el número de clústeres k debe de ser escogido a priori), de tal manera que cada elemento pertenezca al clúster cuya media sea más cercana y tratando de minimizar la suma de las distancias de cada elmento a la media de su clúster.

Para poder obtener la media de un clúster es necesario que se esté usando una distancia euclídea. En nuestro caso se está usando una función de distancia no euclídea (la matriz de distancias obtenida en la sección anterior), que haría imposible obtener una media, al menos de la manera tradicional. Es por este motivo que se ha optado por usar el algoritmo k-medoids.

El funcionamiento del algoritmo k-medoids es prácticamente idéntico al de k-means, pero en lugar de usar como centro del clúster una media, se escogerá uno de los elementos pertenecientes a cada clúster que actuara como medoide (del inglés: *medoid*) de éste. Un medoide es un elemento representativo de un conjunto de datos, que actúa como la media de éste cuando sea imposible obtener una media verdadera. Las distancias de los elementos se calcularán por lo tanto con respecto al medoide de su clúster, el cual es escogido tratando de minimizar estas distancias.

El método más común para la creación de estos clústeres es la repartición alrededor de medoides o PAM (del inglés: *Partitioning Around Medoids*) el cual selecciona unos medoides iniciales (de manera aleatoria o según cualquier otro método) y forma los clústeres al rededor de ellos. A continuación, se probará a intercambiar cada medoide con cada elemento no medoide y se calculará el coste total de la nueva configuración, deshaciendo el cambio en caso de que el coste haya aumentado. Este proceso se repetirá mientras el coste total descienda. Ya que en cada iteración hay que calcular el coste total (calcular este coste requerirá realizar $k-n$ operaciones) por cada uno de los k medoides y por cada uno de los $k-n$ elementos no medoides, su complejidad será $O(k(n-k)^2)$.

Para el trabajo se ha usado un algoritmo distinto más eficiente, diseñado por H. Park y C. Jun en [18]. Este algoritmo logra una complejidad $O(kn)$ por iteración y propone un método de selección de medoides iniciales que a menudo reduce el número de iteraciones. El algoritmo junto con el método propuesto de selección de medoides iniciales es el siguiente:

Algoritmo 4 Cómputo de clústeres por k-medoids

Entrada: Matriz de distancias D, con distancias d_{ij} $(1 \leq i \leq n, 1 \leq j \leq n)$; número de clústeres k

1: crear vector v
2: **para todo** $1 \leq i \leq n$ **hacer**
3: $\quad v_i = \sum_{j=1}^{n} \frac{d_{ji}}{\sum_{k=1}^{n} d_{jk}}$
4: **fin para**
5: ordenar los elementos según los valores de v en orden ascendente y seleccionar los k primeros como medoides iniciales
6: asignar cada elemento al clúster con el medoide más cercano
7: calcular coste total
8: **repetir**
9: $\quad$ **para todo** *clúster* **hacer**
10: $\quad\quad$ **para todo** *elemento* del *clúster* **hacer**
11: $\quad\quad\quad$ calcular suma de distancias desde *elemento* hasta cada otro miembro del *clúster*
12: $\quad\quad$ **fin para**
13: $\quad\quad$ asignar como nuevo medoide del *clúster* al elemento con la menor suma de distancias
14: $\quad$ **fin para**
15: $\quad$ asignar cada elemento al clúster con el medoide más cercano
16: $\quad$ calcular nuevo coste total
17: **hasta que** coste actual = coste anterior

Para calcular los clústeres es necesaria una matriz de distancias de gran tamaño, como en el caso de PageRank. Por desgracia en este caso no se podrá dividir la matriz en bloques tan fácilmente, ya que el acceso a la memoria no es tan ordenado. Es por ello que se ha decidido no usar todos las cuentas para crear los clústeres, si no únicamente aquellas más relevantes. Para ello se usará la puntuación PageRank de acuerdo a las relaciones de seguidores, ya que es la mejor métrica considerando los resultados obtenidos en la sección 6.2.

Para que haya únicamente un componente conexo, es necesario realizar la comprobación de conectividad con únicamente las cuentas que se vayan a usar para la creación de clústeres. Si no se hiciese esto, nodos esenciales para conectar varias partes del grafo podrían no estar incluídos entre los escogidos para crear los clústeres, dejando entonces distintas partes inconexas.

Una vez creados los clústeres, calcularemos para cada clúster la relevancia media de sus usuarios y las polaridades medias con respecto a cada uno de los partidos políticos según los tuits publicados por sus usuarios. Para calcular la relevancia, se ha utilizado la puntuación de PageRank. Para calcular las polaridades medias, se aprovecharán las polaridades medidas para cada tuit en la sección Análisis de representatividad de polaridades 6.3. Por cada clúster

y partido político se calculará la polaridad media a partir de los tuits publicados por cuentas pertenecientes al clúster y que mencionen al partido político, de manera similar a como se hizo en la sección 6.3.

Los resultados obtenidos para una muestra de $n = 1\,000$ y usando $k = 6$ clústeres son los siguientes:

Cuadro 7.1: Tamaño y relevancia media de los clústeres

Seguidores		Retuits	
Tamaño	Relevancia	Tamaño	Relevancia
618	0.006 37	546	0.002 20
168	0.000 72	144	0.002 27
85	0.001 43	128	0.001 19
65	0.000 53	64	0.000 61
39	0.000 94	24	0.000 68
24	0.001 29	49	0.000 61

Cuadro 7.2: Polaridades de los clústeres con respecto a cada partido político (grafo de seguidores)

Tamaño	PP	PSOE	UPyD	Podemos	Ciudadanos
618	−0.294	−0.264	0.015	0	−0.232
168	−0.339	−0.335	0.057	−0.007	−0.287
39	−0.239	−0.212	−0.194	−0.667	−0.261
65	−0.442	−0.279	−0.383	0.071	−0.067
85	−0.266	−0.343	−0.198	0.257	−0.333
24	0.200	−0.279	−0.383	0.071	0.257

Cuadro 7.3: Polaridades de los clústeres con respecto a cada partido político (grafo de retuits)

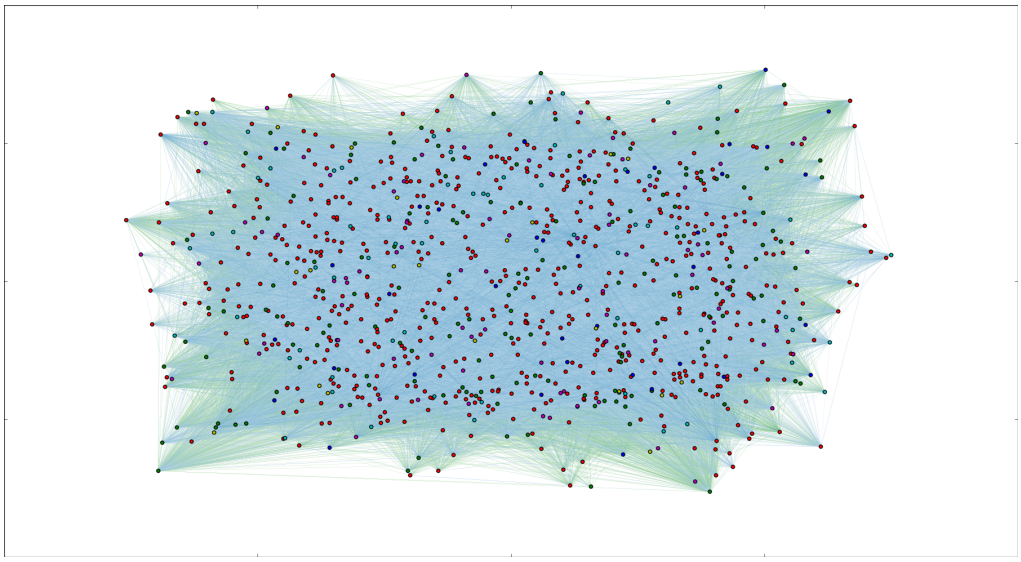
Tamaño	PP	PSOE	UPyD	Podemos	Ciudadanos
546	−0.257	−0.349	0.019	−0.042	−0.309
144	−0.449	−0.167	−0.179	−0.091	0.047
128	−0.248	−0.343	−0.187	0.340	−0.092
64	−0.419	−0.192	−0.178	−0.061	−0.463
49	−0.432	−0.380	−0.248	−0.136	0.052
24	−0.204	−0.424	0.013	−0.100	−0.388

A partir de los resultados de la tabla 7.1 se puede observar que existe un clúster de tamaño significativamente mayor. Este clúster por lo general tiene más relevancia media que el resto; excepto en el caso del segundo mayor clúster del grafo de retuits, que tiene menor relevancia que el mayor. La desviación típica no se muestra en la tabla, pero es en todos los casos menor que $4.4e^{-19}$. Es tan pequeña que se puede considerar que sí que existen diferencias entre las relevancias de los distintos clústeres. Aunque, dado que las diferencias entre los valores de las medias de relevancia no son muy significativas y que los tamaños de los clústeres difieren tanto, no parece que esta técnica de clusterización pueda ser muy útil a la hora de predecir la relevancia de las cuentas.

En los resultados de polaridad para ambas clusterizaciones no se pueden observar preferencias claras de partidos políticos. En todos los clústeres las puntuaciones de los mismos partidos son similares y es en los clústeres más pequeños donde más diferencias en las polaridades hay con el resto. Estas diferencias tampoco son muy significativas debido a la disparidad de tamaños de los clústeres, ya que las diferencias en los grupos más pequeños podrían deberse a anomalías como errores en el detector de polaridad.

Se ha tratado de visualizar los grafos y sus clústeres empleando la técnica de escalado multidimensional o MDS (del inglés: *Multidimensional scaling*). Esta técnica consiste en asignar coordenadas espaciales a los elementos de una colección, tratando de representar la similitud de los distintos elementos con la distancia que hay entre ellos. Elementos similares se encontrarán próximos según las coordenadas resultantes. De esta manera, usando las matrices de distancias ya creadas como matrices de disimilitud (que indica cuán distintos son los elementos), se podría utilizar MDS para visualizar los gráficos.

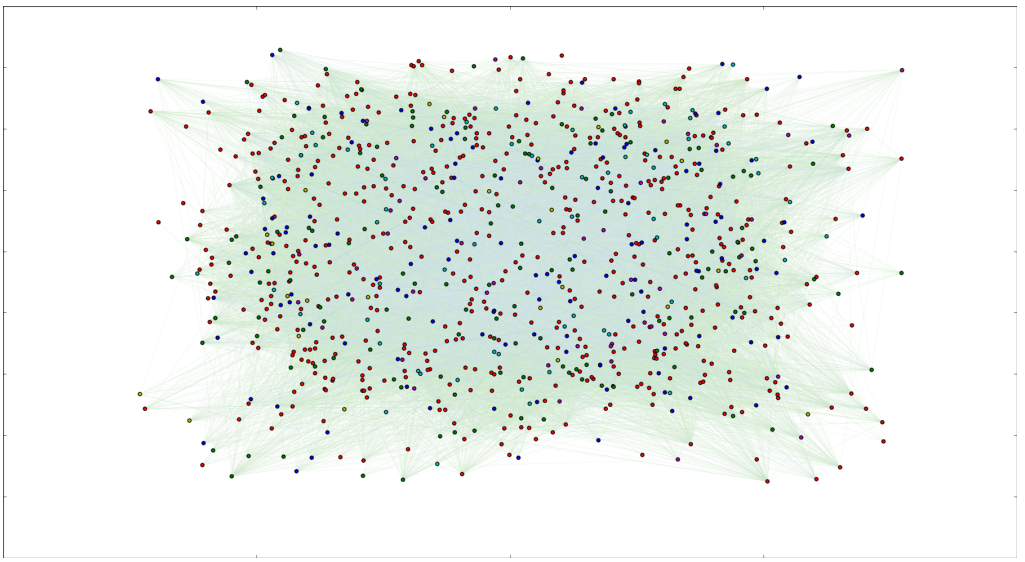
Figura 7.2: Visualización del grafo de seguidores, usando $n = 1\,000$ y $k = 6$



Para la visualización se ha utilizado la clase *manifold.MDS* de la librería *scikit-learn* de Python, que emplea el algoritmo SMACOF (*Scaling by Majorizing a Complicated Function*). Este algoritmo trata de minimizar la diferencia existente entre las distancias ideales (las indicadas por la matriz de disimilitud) y las distancias euclideas del resultado.

En las visualizaciones 7.2 y 7.3, cada punto representa una cuenta y su color indica el clúster al que pertenece. En ambos casos se aprecia que la mayoría de los puntos son de color rojo, lo que significa que la mayoría de las cuentas han sido asignadas al mismo clúster. Por desgracia no se puede observar ninguna agrupación según los clústeres: los colores de los nodos están muy desordenados y sus posiciones no parecen tener ninguna relación con los clústeres a los que pertenecen.

Figura 7.3: Visualización del grafo de retuits, usando $n = 1\,000$ y $k = 6$



Los ejes azules indican relaciones bidireccionales, mientras que los verdes indican relaciones unidireccionales. En el grafo de seguidores abundan las relaciones bidireccionales, se puede asumir que también hay muchas unidireccionales aunque no sean visibles, ya que las bidireccionales han sido dibujadas encima de las otras. En cambio resulta interesante que en el caso de los retuits apenas aparecen relaciones bidireccionales. Esto se puede entender si tenemos en cuenta lo explicado en el tercer párrafo de la sección Recopilación de información de los usuarios 4.2: cuando aparece que una cuenta reuitea a otra, es probable que no le haya retuiteado directamente. Esto podría indicar que no existe una relación de seguimiento entre la cuenta que ha retuiteado y el autor del tuit, lo que hace poco probable una relación bidireccional de retuits.

Otra explicación, más sencilla y quizás más relevante, es que la cantidad de retuits leídos es muy pequeña. Cada cuenta ha retuiteado de media tuits creados por 140 cuentas distintas,

mientras que de media cada usuario sigue a 971 cuentas (ambas cifras incluyen cuentas que no han sido recogidas en nuestra colección). Dados estas cifras y suponiendo que un usuario solo pudiese retuitear a cuentas que sigue, se puede calcular que la media de relaciones de retuits mutuas de un usuario sería 20. Por supuesto, puesto que un usuario no sólo retuiteará a sus amigos (como se ha explicado en el párrafo anterior), que en nuestra colección no están recogidos ni todos los amigos ni todas las cuentas a la que ha retuiteado un usuario y que el tamaño de la muestra que hemos cogido para calcular los clústeres es aún mucho más reducido. el número de estas relaciones mutuas de retuits que podemos observar en la visualización será bastante inferior que 20.

Se han repetido estas análisis para muestras de otros tamaños ($100 \leq n \leq 3\,000$) y usando distintos números de clústeres ($3 \leq k \leq 6$), pero en todos los casos los resultados han sido muy similares: un clúster significativamente más grande que el resto, diferencias en las polaridades no muy significativas y ningún patrón reconocible en la visualización.

Capítulo 8

Conclusiones

Durante este proyecto se ha creado una base de datos MongoDB y una aplicación en Python que utiliza la API de Twitter para obtener tuits e información de las cuentas de usuarios. Se ha tenido que estudiar la documentación tanto de MongoDB (de lo que no se tenían conocimientos previos al ser un sistema de bases de datos NoSQL) como de la API de Twitter.

A continuación, se han implementado los algoritmos PageRank y HITS. Se encontraron problemas debido al tamaño de las matrices, pero que fueron resueltos gracias al uso de la librería PyTables. Las ordenaciones proporcionadas por estos algoritmos fueron puestas a prueba y se obtuvieron resultados muy satisfactorios: los resultados de los distintos algoritmos eran claramente mejores que una ordenación según el número de seguidores de las cuentas. La mejor ordenación ha sido la proporcionada por el algoritmo PageRank a partir del grafo de seguidores.

El análisis de representatividad de polaridades de los tuits no ha proporcionado los resultados esperados. Sin embargo, este experimento tampoco ha sido en vano, puesto que la prueba enfocada a calcular la precisión del detector de polaridad usado sí que ha sido exitosa, demostrando así su correcto funcionamiento.

Se ha implementado un algoritmo de clusterización que ha sido aplicado a las cuentas de Twitter recogidas. No ha sido posible detectar entre los resultados ningún patrón significativo, pero es un estudio que merecía la pena realizar y que ha servido para adquirir nuevos conocimientos de librerías de Python y de distintas formas tanto de estudiar como de visualizar datos.

Durante toda la realización de este trabajo se ha adquirido también mucha experiencia a la hora de investigar artículos ciéntificos y demás bibliografía. Por último, para la redacción de este documento se ha utilizado LaTeX, herramienta que ha resultado ser muy flexible y cómoda.

Trabajos futuros

Como trabajos futuros hay múltiples posibilidades. Aún se podrán realizar más pruebas con los algoritmos HITS y PageRank: usando una colección más grande de tuits o tuits enfocados a otro campo específico, probar otras variaciones de estos algoritmos (como las propuestas en [6]), contrastar la calidad de sus ordenaciones por medio de otras métricas, etc.

En cuanto a los clústeres, es posible que, con otro tipo de clusterización, utilizando una función de distancia distinta o con una mayor cantidad de muestras se pudiese obtener resultados más concluyentes, por lo que es un área en la que aún merece la pena seguir investigando.

Como se ha comentado en la introducción, existe un gran interés en el procesamiento del contenido de las redes sociales, y en particular en Twitter, por lo que se podría seguir trabajando en la búsqueda de otros posibles usos prácticos de estas tecnologías.

Bibliografía

[1] ARTHUR, CHARLES: «How low-paid workers at 'click farms' create appearance of online popularity». The Guardian, 2013.
http://www.theguardian.com/technology/2013/aug/02/click-farms-appearance-online-popularity

[2] BRIN, SERGEY y PAGE, LAWRENCE: «The Anatomy of a Large-Scale Hypertextual Web Search Engine». Computer Networks and ISDN Systems, 1998.
http://infolab.stanford.edu/~backrub/google.html

[3] CHIEN, ONG KOK; HOONG, POO KUAN y HO, CHIUNG CHING: «A Comparative Study of HITS vs PageRank Algorithms for Twitter Users Analysis». Proceedings of the International Conference on Computation Science and Technology 2014, 2014.
http://www.academia.edu/8108166/A_Comparative_Study_of_HITS_vs_PageRank_Algorithms_for_Twitter_Users_Analysis

[4] CONGOSTO, M. LUZ y ARAGÓN, PABLO: «Twitter, del sondeo a la sonda nuevos canales de opinión: nuevos métodos de análisis». Más poder local, 2012.
http://www.researchgate.net/publication/259331406_Twitter_del_sondeo_a_la_sonda_nuevos_canales_de_opinin_nuevos_mtodos_de_anlisis

[5] COSSU, JEAN-VALÈRE; DUGUÉ, NICOLAS y LABATUT, VINCENT: «Detecting Real-World Influence Through Twitter». 2nd European Network Intelligence Conference, 2015.
http://arxiv.org/pdf/1506.05903v2.pdf

[6] DING, CHRIS; HE, XIAOFENG; HUSBANDS, PARRY; ZHA, HONGYUAN y SIMON, HORST D.: «PageRank, HITS and a Unified Framework for Link Analysis». Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, 2002.
http://delab.csd.auth.gr/~dimitris/courses/ir_spring06/page_rank_computing/p353-ding.pdf

[7] GILBERT, EDGAR N.: «Random Graphs». Annals of Mathematical Statistics, 1959.
http://projecteuclid.org/download/pdf_1/euclid.aoms/1177706098

[8] GOOGLE: «Eric Schmidt at Techonomy», 2010.
https://www.youtube.com/watch?v=UAcCIsrAq70

[9] GUPTA, PANKAJ; GOEL, ASHISH; LIN, JIMMY; SHARMA, ANEESH; WANG, DONG y ZADEH, REZA: «WTF: The Who to Follow Service at Twitter». En: Proceedings of the 22nd International Conference on World Wide Web, , 2013.

[10] HJØRLAND, BIRGER y CHRISTENSEN, FRANK SEJER: «Work tasks and socio-cognitive relevance: A specific example». Journal of the American Society for Information Science and Technology, 2002.

[11] HURTADO, LLUÍS F. y PLA, FERRAN: «Political Tendency Identification in Twitter using Sentiment Analysis Techniques». En: Coling 2014, , 2014.

[12] INSTITUTE, STATISTIC BRAIN RESEARCH: «Google Annual Search Statistics», 2015. http://www.statisticbrain.com/google-searches/

[13] JANGA, S. MO y HART, P. SOL: «Polarized frames on "climate change" and "global warming" across countries and states: Evidence from Twitter big data». Global Environmental Change, 2015. http://www.sciencedirect.com/science/article/pii/S0959378015000291

[14] KWAK, HAEWOON; LEE, CHANGHYUN; PARK, HOSUNG y MOON, SUE: «What is Twitter, a social network or a news media?» WWW '10 Proceedings of the 19th international conference on World wide web, 2010.

[15] MITCHELL, AMY y HITLIN, PAUL: «Twitter Reaction to Events Often at Odds with Overall Public Opinion». Pew Research Center, 2013. http://www.pewresearch.org/2013/03/04/twitter-reaction-to-events-often-at-odds-with-overall-public-opinion/

[16] NABEL, LUIS CÉSAR TORRES: «Social networks and political effects. Reflections about twitter impact in México». Sociología y tecnociencia. Revista digital de sociología del sistema tecnocientífico, 2009. http://www.academia.edu/1049146/Redes_sociales_y_efectos_pol%C3%ADticos._Reflexiones_sobre_el_impacto_de_twitter_en_M%C3%A9xico

[17] NETCRAFT: «Netcraft Web Server Survey», 2015. http://news.netcraft.com/archives/category/web-server-survey/

[18] PARK, HAE-SANG y JUN, CHI-HYUCK: «A simple and fast algorithm for K-medoids clustering». Expert Systems with Applications, 2014. http://www.sciencedirect.com/science/article/pii/S095741740800081X

[19] PEDROCHE, FRANCISCO: «Métodos de cálculo del vector PageRank». Boletín de la Sociedad Española de Matemática Aplicada, 2007.

[20] PLA, FERRAN y HURTADO, LLUÍS F.: «Análisis de Sentimientos, Detección de Tópicos y Análisis de Sentimientos de Aspectos en Twitter». En: TASS 2014, , 2014.

[21] PLA, FERRAN y HURTADO, LLUÍS F: «Sentiment Analysis in Twitter for Spanish». En: NLDB 2014, , 2014.

[22] RAE, (@RAEINFORMA): «El director de la RAE, José Manuel Blecua, anuncia la incorporación de "tuitear", "tuit", "tuiteo" y "tuitero" al Diccionario académico.», 2012.
https://twitter.com/raeinforma/status/248735186592870401

[23] ROGERS, SIMON: «What fuels a Tweet's engagement?», 2014.
https://blog.twitter.com/2014/what-fuels-a-tweets-engagement

[24] RUSSELL, MATTHEW A.: Mining the Social Web. O'Reilly Media, 2013.

[25] SCHALL, DANIEL: «Who to follow recommendation in large-scale online development communities». Information and Software Technology, 2014.
http://www.sciencedirect.com/science/article/pii/S0950584913002322

[26] STATISTA: «Number of monthly active Twitter users worldwide from 1st quarter 2010 to 2nd quarter 2015», 2015.
http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

[27] STATS, INTERNET LIVE: «Internet Users», 2014.
http://www.internetlivestats.com/internet-users/

[28] STONE, BIZ: «Introducing the Twitter API», 2006.
https://blog.twitter.com/2006/introducing-twitter-api

[29] TWITTER, INC:. «REST APIs».
https://dev.twitter.com/rest/public

[30] VIDALA, LETICIA; ARESA, GASTÓN; MACHÍNB, LEANDRO y JAEGERC, SARA R.: «Using Twitter data for food-related consumer research: A case study on "what people say when tweeting about different eating situations"». Food Quality and Preference, 2015.
http://www.sciencedirect.com/science/article/pii/S0950329315001263

[31] WALL, AARON:. «Search Engine History».
http://www.searchenginehistory.com/

[32] YU, YANG y WANG, XIAO: «World Cup 2014 in the Twitter World: A big data analysis of sentiments in U.S. sports fans' tweets». Computers in Human Behaviour, 2015.
http://www.sciencedirect.com/science/article/pii/S074756321500103X

Apéndice A

Ordenaciones

Cuadro A.1: PageRank-Seguidores

Nombre	PageRank
el_pais	0.003 23
iescolar	0.002 27
anapastor	0.002 20
elmundoes	0.002 05
EFEnoticias	0.001 92
La_SER	0.001 91
eldiarioes	0.001 79
ahorapodemos	0.001 70
agarzon	0.001 60
europapress	0.001 60
publico_es	0.001 56
Albert_Rivera	0.001 53
rtve	0.001 47
abc_es	0.001 43
pedroj_ramirez	0.001 36
elconfidencial	0.001 31
muyinteresante	0.001 29
GLlamazares	0.001 29
20m	0.001 29
ierrejon	0.001 28
PSOE	0.001 27
sanchezcastejon	0.001 23
ElHuffPost	0.001 21
gerardotc	0.001 19
norcoreano	0.001 18
jesusmarana	0.001 17
carlosecue	0.001 14
PPopular	0.001 08
JotDownSpain	0.001 07
24h_tve	0.001 06

Cuadro A.2: PageRank-Retuits

Nombre	PageRank
eldiarioes	0.006 30
eldiarioAnd	0.005 88
EFEempresas	0.005 46
el_pais	0.005 01
europapress	0.004 35
carnecrudaradio	0.004 13
EFEnoticias	0.003 46
gerardotc	0.003 17
La_SER	0.003 04
iescolar	0.002 90
abc_es	0.002 79
anapastor	0.002 69
EFEemprende	0.002 50
elconfidencial	0.002 23
diarioas	0.002 14
elmundoes	0.002 13
AntonioMaestre	0.002 03
diostuitero	0.001 95
dalealplay	0.001 94
ElHuffPost	0.001 81
elespanolcom	0.001 77
Albert_Rivera	0.001 66
subversivos_	0.001 64
cristina_pardo	0.001 60
RubenSanchezTW	0.001 58
norcoreano	0.001 57
publico_es	0.001 49
ahorapodemos	0.001 48
MonicaCarrillo	0.001 47
masaenfurecida	0.001 36

Cuadro A.3: HITS-Seguidores (*authority*)

Nombre	*authority*
iescolar	0.113 42
el_pais	0.107 94
anapastor	0.106 42
eldiarioes	0.102 00
publico_es	0.101 66
agarzon	0.098 10
ahorapodemos	0.090 51
La_SER	0.087 14
GLlamazares	0.086 80
JoseantonioJun	0.086 77
europapress	0.083 04
20m	0.080 95
elmundoes	0.079 50
jesusmarana	0.079 34
elconfidencial	0.075 56
El_Intermedio	0.075 38
ierrejon	0.075 11
ElHuffPost	0.073 86
EFEnoticias	0.073 78
MonederoJC	0.069 98
cayo_lara	0.068 15
DebatAlRojoVivo	0.067 60
Agus_Martinez58	0.067 60
PSOE	0.067 24
RevillaMiguelA	0.066 13
cristina_pardo	0.066 05
caval100	0.066 03
gerardotc	0.065 78
Ainhat	0.064 96
radiocable	0.064 78

Cuadro A.4: HITS-Seguidores (*hub*)

Nombre	*hub*
caval100	0.052 68
Agus_Martinez58	0.052 46
atlante83	0.048 49
JoseantonioJun	0.046 95
Pedro_Zerolo	0.045 68
SobresEnB	0.040 12
mariano9605	0.039 69
David_Calvo	0.039 18
carlosgomezgil	0.038 79
AnaGarridoRamos	0.038 50
nefeerr	0.038 31
twrboleta	0.036 97
curroflores1952	0.036 81
LuisSalvador	0.036 69
Zurine3	0.036 24
MorenoG_Agustin	0.035 99
ecorepublicano	0.035 40
ARMAKdeODELOT	0.035 31
diegocruzblog	0.035 14
7Josean7	0.033 47
FonsecAsturias	0.033 34
javierjuantur	0.033 21
JosePerLo	0.031 98
ManuelHuertas75	0.031 67
gsemprunmdg	0.030 89
sindi584	0.030 81
JLMagaz	0.030 66
cyberalex75	0.030 59
JuanMalaga2011	0.030 59
luisbeltri	0.030 41

Cuadro A.5: HITS-Retuits (*authority*)

Nombre	*authority*
gerardotc	0.176 08
eldiarioes	0.165 53
iescolar	0.156 49
el_pais	0.152 99
subversivos_	0.139 07
SiPeroNo1	0.127 86
EsppeonzAguirre	0.123 48
ecorepublicano	0.122 45
AntonioMaestre	0.121 77
Cazatalentos	0.117 66
rcabrero75	0.115 95
cristina_pardo	0.115 31
diostuitero	0.115 12
ahorapodemos	0.114 23
publico_es	0.113 84
isaranjuez	0.109 36
elpadrecorajede	0.108 38
masaenfurecida	0.103 97
La_SER	0.102 82
ElHuffPost	0.102 01
anapastor	0.100 71
Famelica_legion	0.098 46
europapress	0.096 82
rosamariaartal	0.094 60
RubenSanchezTW	0.094 57
agarzon	0.093 42
moedetriana	0.093 27
Juanmi_News	0.092 67
BobEstropajo	0.090 58
norcoreano	0.089 31

Cuadro A.6: HITS-Retuits (*hub*)

Nombre	*hub*
Otaru_CC	0.018 64
nabilu2	0.018 50
grigrigrillito	0.018 27
Observador2507	0.017 98
J_AnguloMestre	0.017 72
onty25	0.017 64
MariQueroGra	0.017 50
CU15M	0.017 41
FjRamiz81	0.017 38
yankeegohome5	0.017 28
espedompla	0.017 24
atletico47	0.017 20
malacate09	0.017 15
emiliocurt	0.017 02
jsaura58	0.016 93
Eml59Ruiz	0.016 88
pacopilP	0.016 69
jumagaka	0.016 69
gisesille	0.016 63
Danteboxs	0.016 62
PartidoPodrido	0.016 62
JrealJavireal	0.016 61
Juancontecho	0.016 59
Mylla60	0.016 59
AntonioX92	0.016 53
incomodo0	0.016 52
Chica_Gilmore	0.016 51
Vallekasxlakara	0.016 49
antonmiragaya	0.016 48
cmarinmunoz	0.016 44

Cuadro A.7: Número de seguidores

Nombre	seguidores
muyinteresante	5513709
NoticiasCaracol	5390902
lopezdoriga	4824328
el_pais	4255520
globovision	3535606
ElUniversal	3484306
El_Universal_Mx	3014002
ElNacionalWeb	3007426
MariaCorinaYA	2529908
noticierovv	2338825
noticias24	2300102
CaracolRadio	2167229
vickydavilalafm	2048904
CNNMex	2030804
PedroFerriz	1954103
HayQueDecirQue	1898071
CNNChile	1860584
rcnlaradio	1824038
NTelevisa_com	1750877
El_Hormiguero	1733709
elmundoes	1729866
liliantintori	1618535
malditaternura	1590425
NTN24ve	1562946
mundodeportivo	1551304
CF_America	1537514
TodoReflexion	1528374
PepeAguilar	1446311
diarioas	1429826
anapastor	1277594