UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Crowdsensing en Ciudades Intelientes

Sheila Bustos Jiménez

Junio, 2015

CROWDSENSING EN CIUDADES INTELIENTES

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de Información

TECNOLOGÍA ESPECÍFICA DE
INGENIERÍA DE COMPUTADORES

TRABAJO FIN DE GRADO

Crowdsensing en Ciudades Intelientes

Autor: Sheila Bustos Jiménez

Director: Félix Jesús Villanueva Molina

Director: David Villa Alises

Junio, 2015

Sheila Bustos Jiménez

Ciudad Real – Spain

E-mail: sheila.bustos@uclm.es

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

En este proyecto se estudian las redes sociales como elemento fundamental de cara a conocer la actividad de una comunidad dentro de una Ciudad Inteligente que vive conectada entre sí gracias a las mismas. Una red social puede ser una herramienta muy útil para empresas u organizaciones que quieren conocer la aceptación que tienen en una comunidad, o para poder conocer qué intereses mueven a un colectivo determinado.

Así pues, desarrollaremos una plataforma capaz de extraer, almacenar, analizar y visualizar la información que las redes sociales pueden ofrecernos a cerca de una ciudad concreta, en este caso Ciudad Real. Un elemento fundamental para conseguir recoger y mantener los datos objetivo, es el uso de una tecnología de Base de Datos poco habitual en los desarrollos actuales, que no es otra que la tecnología de *Base de Datos Orientada a Grafos*, que nos permitirá almacenar las relaciones entre los elementos de la red social de manera eficiente y natural. Además, para conseguir que la plataforma sea capaz de detectar sucesos anormales en la ciudad, se desarrolla una técnica de *Detección de Anomalías* basada en contabilizar el volumen de publicaciones realizadas durante un periodo de tiempo, que es capaz de detectar en qué momentos la actividad de la red social es superior a la actividad normal, hecho que se extrapola directamente a un suceso real en la ciudad. Todos los resultados obtenidos pueden recuperarse gracias a una aplicación Web que permiten conocer al ciudadano tanto la actividad pasada como presente de la red social, además de permitir a ciudadanos, compañías y gobiernos conocer todos y cada uno de los eventos que se han dado en su ciudad y que, han tenido gran repercusión en las redes sociales.

Por tanto, Crowdsensing en Ciudades Inteligentes ofrece una visión diferente de las redes sociales, más allá de su uso «social», y que permitirá conocer y estudiar de manera directa la actividad de diferentes comunidades en sus entornos.

Abstract

In this proyect, we study social networks as a key element in order to know the activity of a community which belongs to a smart city that lives connected to each other through them. A social network could be a very useful tool for companies or organizations which want to know the acceptance that they have in a community, or to know what interests move to a particular group.

So, we develop a platform with the capability to extract, store, analyze and display the information that social networks can offer about a particular city, in this case, we focus in Ciudad Real. A key to achieving the objective to collect and make data persistent is the use of a unusual database technology on current developments. This technology is Graph Databases, which allows store the relationship between the elements of the social network on a efficient and natural way. Furthermore, to ensure the capability of the platform to detect abnormal events in the city, we develop an Anomaly Detection technique based on counting the number of publications over a period of time, which is able to detect if the activity of the social network is higher than normal activity, a fact which is directly extrapolated to a real event in the city. All the results can be retrieved through a Web application that allows the citizen to know both past an present activity of the social network, and let citizens, companies and goverments to know all the events that have occurred in the city and have had a big impact on social networks.

Therefore, Crowdsensing en Ciudades Inteligentes offers a different view of social networks beyond their «social» use, and that will show and study directly the activity of different communities in their environments.

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor, Félix Jesús Villanueva Molina la oportunidad que me dio aceptándome como alumna para la realización de este proyecto, del que tanto he aprendido. También me gustaría agradecer nuevamente a Félix y a David Villa Alises su disposición a la hora de ayudarme y ofrecerme sus consejos y conocimientos cuando el camino se complicaba.

A mis padres y a mi hermana, les agradezco la paciencia de la que han tenido que armarse para tratar conmigo, mi mal humor y mis agobios durante estos cuatro años, y más especialmente en este último año donde los nervios se han apoderado de mi en muchas situaciones.

También tengo que agradecer a Aurelio y a Mila el gran apoyo que han supuesto para mi todo este tiempo. A Mila por ser incondicional, en el momento justo, en el lugar oportuno; y a Aurelio por enseñarme a afrontar este proyecto – y la vida – con optimismo y tranquilidad, has hecho que este último año sea más fácil.

Gracias a todos. Un trocito de todos mis logros, y los que vendrán, os pertenecen.

Sheila

Llegarás más lejos cuanto más altas sean tus metas.

Índice general

Índice de cuadros

Índice de figuras

Índice de listados

Listado de acrónimos

CIS	Centro de Investigaciones Sociológicas
API	Application Programming Interface
IoT	Intertet of Things
GPS	Global Positioning System
TIC	Tecnologías de la Información y la Comunicación
RECI	Red Española de Ciudades Inteligentes
POS	Part Of Speech
ARM	Association Rule Mining
RMT	Rule Matching Threshold
CRUD	Create, Read, Update, Delete
GNU	GNU is Not Unix
GPL	General Public License
AGPL	Affero General Public License
IAB	Interactive Advertising Bureau
REST	Representational State Transfer
HTML	HyperText Markup Language
PHP	PHP Hypertext Pre-processor
CSS	Cascading Style Sheets
XP	eXtreme Programming
ASD	Adaptative Software Development
JSON	JavaScript Object Notation
MVT	Model View Template
MVC	Model View Controller
XML	eXtensible Markup Language
WSGI	Web Server Gateway Interface
SATA	Serial Advanced Technology Attachment

SSD	Solid-State Drive
ext4	Fourth Extended filesystem
ZFS	Zettabyte File System

Capítulo 1

Introducción

DESDE el nacimiento de lo que se conoce como *Web 2.0*, las redes sociales se están convirtiendo en una gran fuente de información en tiempo real sobre el comportamiento y pensamiento de una sociedad. La percepción del gran potencial que tiene esta información es percibido por investigadores, empresas y organismos públicos por lo que han aparecido numerosas plataformas que pretenden extraer toda esta información de las redes sociales. La visualización del lugar donde los usuarios publican contenido (*Geolocalización*), vigilancia de la imagen de la marca de una empresa, o la posibilidad de conocer tendencias políticas, son ejemplos de información que puede obtenerse a través de redes sociales como Facebook, Twitter, Instagram, además de poder conocer, por supuesto, la relación existente entre los usuarios de la misma.

Normalmente, esta información se analiza con una perspectiva global atendiendo más al volumen de información que proporcionan que al origen de la misma. Sin embargo, cuando nos centramos en un área geográfica concreta, por ejemplo, una ciudad, podemos obtener información de primera mano de dicho ámbito. Así, en este proyecto se pretenden estudiar las redes sociales y sus usuarios como sensores inteligentes que monitorizarán la actividad cotidiana de su ciudad. Esta utilización de las personas como sensores se denomina *Crowdsensing*. El objetivo de este fenómeno es obtener información para contribuir a mejorar un objetivo social o conocer la relevancia de un suceso o acontecimiento de interés para una sociedad concreta [GYL11], todo ello gracias a la recolección de la información que ofrecen los individuos de la comunidad mediante el uso de diferentes dispositivos (smartphones, sensores vehiculares...) o herramientas, como las redes sociales, en nuestro caso.

Este término se encuadra dentro de lo que se conoce como Ciudad Inteligente o*Smart City*. Si nos referimos a este concepto en términos gubernamentales y políticos, una Ciudad Inteligente se refiere al conjunto de medidas e infraestructuras desplegadas sobre la ciudad, que contribuyen a mejorar y conseguir el desarrollo sostenible de la misma. Sin embargo, y desde una perspectiva tecnológica, nos referimos a Ciudad Inteligente como ciudad interconectada. Esta interconexión permite capturar información de diferentes fuentes como sensores, dispositivos personales, smartphones, cámaras e incluso las redes sociales entendidas como redes de sensores humanas. Además, para que la ciudad pueda ser denominada

«inteligente», todos los datos obtenidos deben ser analizados y modelados para obtener información que permita tanto mejorar la toma de decisiones, como conocer el impacto de un suceso [NP11]. Así pues, la obtención de información de ciudadanos, entidades y organismos públicos que pertenecen y desarrollan su labor en dicha ciudad nos puede proporcionar una visión muy interesante acerca de multitud de temas. Alguna de la información que, a priori podría extraerse incluye:

- Previsión del éxito o fracaso de una concentración de carácter reivindicativo, deportivo o cultural de tal forma que tanto los organizadores del evento como las autoridades del entorno pueden establecer los mecanismos adecuados en cuanto a seguridad, suministros, instalaciones, etc.
- Interés en temas sociales, políticos, deportivos y cambios en tendencias de cara a la toma de decisiones estratégicas por parte de organismos públicos.
- Eventos anormales y de índole espontáneo (accidentes, averías, etc.) que se reflejan de manera inmediata en las redes sociales y que puede resultar de interés para los miembros de la ciudad.
- Estado emocional global respecto a servicios públicos (educación, sanidad, etc.) y privados (transporte, energía, etc.) para ayudar a la detección de necesidades y oportunidades de negocio en base a las quejas o deseos de los ciudadanos.
- Reputación de una institución, empresa o personalidad con influencia en la ciudad.
- Estructura social y de interacción de los ciudadanos para entender mejor la ciudad y cómo las actuaciones pueden repercutir en la misma.
- Perfil socio-económico de los usuarios residentes en la ciudad.
- Descripción geo-temporal de la ciudad en lugares de ocio o trabajo de cara a mejorar la eficiencia y orientación de las actuaciones de las instituciones de la ciudad sobre dichas infraestructuras.

Por otro lado, otra parte importante en este proyecto se centra en encontrar la mejor tecnología de almacenamiento para este tipo de información. Generalmente, la mayoría de plataformas y aplicaciones utilizan bases de datos relacionales para almacenar datos relativos a las redes sociales. Sin embargo, el modelado de la estructura de una red social, requiere herramientas más avanzadas que almacenen la información de manera mucho más natural. Por eso centramos nuestra atención en tecnologías de Bases de Datos Orientadas a Grafos, que permiten almacenar información caracterizada por tener un gran número de relaciones entre sus elementos, ofreciendo mayor rendimiento y, por tanto, mayores posibilidades de éxito frente a plataformas que no utilicen esta tecnología.

Así, el presente proyecto pretende responder a preguntas del tipo, ¿podríamos usar las redes sociales como sensor de actividad de una Ciudad Inteligente?, ¿podemos predecir el

éxito de un evento en relación al tráfico de información que se genera en las redes sociales?. Para ello, se construirá un sistema de análisis y detección de eventos en una ciudad inteligente basado en redes sociales.

Capítulo 2

Objetivos

A continuación se enumeran y describen los objetivos y metas que se pretenden conseguir con este proyecto.

2.1 Objetivo general: Crowdsensing en redes sociales

El principal objetivo es el de acceder a la información de redes sociales para ir construyendo y alimentando un sistema que permita mostrar la actividad de una ciudad concreta. Además, esta información permitirá al sistema desarrollar la capacidad de predicción para ofrecer información detallada ante sucesos anormales o extraordinarios relacionados con esa ciudad. Del mismo modo, se pretenden buscar los mejores modelos tanto de almacenamiento como de representación de la información para que tanto el acceso como la visualización de ésta sea lo más intuitivo posible.

2.2 Objetivos específicos

Para conseguir el objetivo general, es necesario definir los objetivos específicos que se detallan a continuación:

2.2.1 Objetivo 1: Recolección de Información

El primero de los objetivos específicos cubre la necesidad de desarrollar un sistema que recolecte la información ofrecida por las redes sociales en torno a una ciudad concreta. Se busca que el sistema sea capaz de seleccionar la información que pertenece al entorno de la ciudad y distinguirla del resto de información. Además, debe preparar la información para su correcto almacenamiento y representación posteriores. Así, para cumplir este objetivo se desarrollará una arquitectura capaz de:

- Obtener datos de valor para ofrecer una información realista, coherente y veraz sobre lo que está ocurriendo en la ciudad.
- Formatear los datos obtenidos para que el sistema sea capaz de almacenarnos y representarlos correctamente. Además, estos datos deben ser fácilmente tratados para obtener la información de una manera eficiente y sencilla.
- Recibir diferentes volúmenes de datos. Como sabemos, la cantidad de actividad en

una red social es diferente según la hora del día y los acontecimientos que rodeen a la sociedad que participa en ella. Es por eso por lo que nuestro sistema debe estar preparado para recibir tanto pequeñas cantidades como grandes volúmenes de datos.

- Monitorizar de manera constante la actividad de la red social. Ya que la capacidad de predicción de un sistema depende en gran medida de la cantidad y contenido de la información que ya conoce, el sistema debe ser capaz de recopilar de manera continua la información de la red social.

2.2.2 Objetivo 2: Análisis de la Información

El siguiente objetivo específico nos ayudará a conseguir las capacidades de aprendizaje y predicción que se buscan para el sistema. Tanto el aprendizaje como la predicción en este tipo de sistemas, son características complejas de conseguir ya que el entorno de una red social es muy cambiante. Para conseguirlo necesitamos que el sistema tenga las siguientes capacidades:

- Capacidad para contabilizar el volumen de información entrante. Esto permitirá conocer cuándo la cantidad de información que está recogiendo el sistema es notablemente superior a lo que el sistema conoce como un volumen normal. Del mismo modo, la noción de normalidad se irá construyendo precisamente con el conteo y análisis de los datos recibidos.
- Capacidad para conocer las relaciones existentes entre los datos que se están recogiendo. Es decir, el sistema debe ser capaz de definir el tema o temas en los que podemos clasificar la información que se obtiene.
- Capacidad para generar estadísticas a partir de actividad previa. Gracias al conteo y análisis de la información, el sistema podrá ofrecer información como la frecuencia con la que ocurre un determinado evento, horas en las que se produce la mayor actividad en la red social, o cuáles son los temas o sucesos que se relacionan con los picos de actividad.

2.2.3 Objetivo 3: Almacenamiento y visualización de la Información

El último de los objetivos específicos tiene que ver con la elección de los modelos de almacenamiento y representación de la información recogida por el sistema. Ambas elecciones pasan por tener en cuenta la forma de los datos que se van a recoger, ya que en ellos, una parte fundamental para obtener información no es tanto el contenido de los mismos si no sus relaciones. Así pues se busca que el sistema de almacenamiento escogido preserve el significado de la relaciones entre elementos y que la plataforma de representación permita mostrar estas relaciones además de mostrar la actividad de la red social, e información estadística y gráfica de las diferentes métricas obtenidas gracias a la capacidad de aprendizaje del sistema.

Capítulo 3

Antecedentes

En este capítulo se exponen todos los elementos necesarios para comprender la finalidad del proyecto y los elementos principales que intervienen en él. Así pues, en primer lugar se definen los conceptos principales en los que se enmarca el ámbito del proyecto:*Redes Sociales*, y los términos *Crowdsensing* y *Ciudad Inteligente*. Las Redes Sociales se definen como elemento de gran importancia en las sociedades actuales y que se ha convertido en un medio para relacionar usuarios y para conocer la aceptación que tienen diferentes sucesos en una comunicad. Esta situación, es aprovechada por diferentes herramientas, entre las que se encuentran las herramientas de Crowdsensing que se encargan de recoger la información que ofrecen los seres humanos al interaccionar con diferentes tecnologías, en nuestro caso, las Redes Sociales. Toda esta información es útil para lo que se conoce como Ciudad Inteligente. A continuación se describen las técnicas de análisis de la información de Redes Sociales, atendiendo tanto a los diferentes mecanismos de Minería de Datos para obtener información seleccionada de la red social, como al posterior análisis de esa información seleccionada.

Por último, se analizan los elementos más importantes que permitirán conseguir todos los objetivos propuestos: Twitter, como Red Social objetivo de análisis; Neo4J, como Base de Datos elegida, y Django, como framework para el desarrollo de la aplicación Web.

3.1 Redes Sociales: Definición, clasificación y estado.

El término red social, en su sentido más clásico, se refiere a una estructura compuesta por personas que están relacionadas entre sí por algún tipo de vínculo - de amistad, familiar, laboral - y que comparten entre sí sus vivencias, opiniones y creencias. Este término, y más concretamente la noción de red, deriva, según Félix Requena Santos, actual Presidente del Centro de Investigaciones Sociológicas (CIS), de la teoría Matemática de Grafos. El Dr. Félix Requena en su artículo «*El concepto de Red Social*» define:

> Una red en Teoría de Grafos es un conjunto de relaciones en el cual las líneas que conectan a los diferentes puntos tienen un valor concreto [San89].

Así pues, podemos representar una red social como un grafo donde, los nodos son las personas que componen la red, y las aristas que conectan los nodos, definen las relaciones

existentes entre entre los individuos de la red. Sin embargo, en los últimos años, el término red social ha evolucionado para poder definir también nuevos conceptos que surgieron con el crecimiento de Internet y su acercamiento a las personas para su uso en el ámbito personal. Actualmente el término red social se utiliza también para referirse a aquellas herramientas o aplicaciones existentes en Internet, y que sirven para relacionar a los usuarios de la misma en torno a intereses u objetivos comunes. Esta nueva acepción engloba un gran número de plataformas que si bien, son muy heterogéneas en cuanto a su fin, tienen en común los elementos básicos que permiten distinguir cualquier otra herramienta de Internet de una red social:

- Usuario: Es un miembro de la red. Para identificarse en ella, posee un «perfil» a través del cual el usuario puede compartir datos personales, tales como su nombre, edad y profesión, aficiones, intereses y gustos. Además, en la mayoría de las redes sociales, un usuario puede categorizarse por su relevancia fuera de la red, lo que permite al resto de usuarios identificar fácilmente a personalidades importantes y reconocidas a nivel público.
- Relación de «amistad»: Establece el vínculo entre usuarios de la red social. Dependiendo de la finalidad de la red social esta relación puede ser bidireccional o unidireccional. Cuando la relación es bidireccional, es necesario que los dos usuarios que se relacionan acepten el vínculo. Así, dos usuarios relacionados de manera bidireccional se denominan amigos. Sin embargo, existen otro tipo de relaciones, unidireccionales, donde el usuario que inicia la relación hacia otro tiene interés en conocer los movimientos de ese usuario en la red, pero no necesariamente tiene que ocurrir lo mismo en sentido contrario. En este caso, decimos que un usuario sigue a otro.
- Publicación: Es el elemento que un usuario comparte con el resto. Según el tipo de red social la publicación puede ser un texto, una foto, música o incluso un enlace a un sitio externo a la red. Una publicación, además tiene múltiples acciones asociadas. La publicación puede ser compartida con los usuarios de la red. Dependiendo del nivel de privacidad de un usuario, la publicación puede ser vista directamente por todos los usuarios de la red o sólo por sus amigos o seguidores. Además una publicación puede ser compartida, es decir, un usuario que tiene acceso a una publicación que pertenece a otro usuario, puede replicar esa publicación para que sus amigos o seguidores también puedan acceder al contenido de la publicación.

La Figura 3.1 representa de modo gráfico los conceptos que se acaban de definir.

Una vez definido el alcance y finalidad de una red social, podemos definir y diferenciar los diferentes tipos de red social que existen hoy en día. Podemos clasificar las redes sociales atendiendo a múltiples criterios: público al que se destinan, temática de la red social... pero la clasificación más extendida es aquella que las clasifica dividiéndolas en dos grupos: redes sociales horizontales o generales y redes sociales verticales o especializadas [Pon12]:

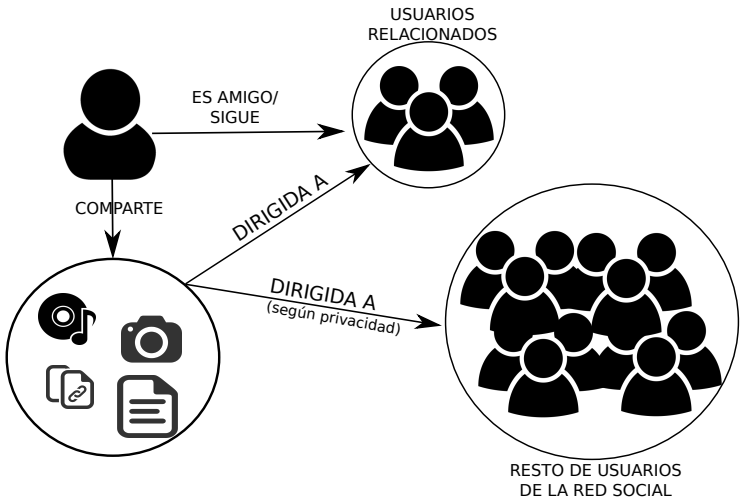

Figura 3.1: Estructura básica de una Red Social

- Redes sociales horizontales: Son aquellas que no tienen una única temática o una temática bien definida. Están destinadas a un amplio público que lo único que busca de la red social, es la interrelación con el resto de usuarios de la red. Ejemplo de este tipo de redes son Facebook o Google+.
- Redes sociales verticales: Al contrario que las redes sociales horizontales, que buscan atraer a una mayoría de usuarios, las redes sociales verticales se centran en un fin concreto. Se crean para atender exclusivamente a una temática o interés por lo que el número de usuarios que acceden a ellas suele ser menor al que accede a una red social general. Puesto que pueden existir diferentes tipos de redes sociales, una para cada temática diferente, clasificar o dividir este tipo de redes sociales puede ser complicado. A continuación se establece una clasificación de de las redes sociales verticales atendiendo a tres dimensiones: temática, contenido compartido y actividad.
    - Clasificación por Temática: A continuación, se listan las categorías más comunes en las que podemos clasificar una red social especializada atendiendo al tema principal de la misma:
        - Profesionales: Son aquellas centradas en el mercado laboral y las actividades comerciales y de negocios. Sus usuarios buscan conocer la actividad de empresas, grupos y otros usuarios en busca de ofertas laborales o un mejor posicionamiento en el mercado laboral. En este caso, el perfil del usuario suele estar compuesto por los datos de su Currículum donde se incluye la experiencia laboral y formación académica. De entre las redes sociales profesionales, la más conocida es LinkedIn.
        - Identidad cultural: Surge con el objetivo de mantener y promover la iden-

tidad de una determinada zona o cultura frente a la globalización, en constante crecimiento en los últimos años. Ejemplos de este tipo de redes son Spaniards, comunidad de españoles en el mundo, o Asianave, red social dedicada a los asiático-americanos.

  - Aficiones: Son redes sociales que buscan relacionar usuarios amantes de actividades de ocio y cultura muy concretas. Ya que están destinadas a compartir vivencias sobre una actividad concreta, existen redes sociales muy particulares y curiosas como Raverly, para los amantes del punto o el ganchillo, o Moteurs, para aquellos aficionados a las motos y el estilo de vida típico de los moteros.
  - Movimientos sociales: Son aquellas cuyo principal objetivo es luchar para conseguir hacer frente a una preocupación social. Las más comunes son las redes en las que participan ecologistas y activistas sociales como Care2.
  - Viajes: Este tipo de red social vertical, es una de las más comunes y con más usuarios. Su finalidad es conectar usuarios que comparten sus experiencias en relación a los viajes que han realizado por todo el mundo. Dentro de las múltiples redes sociales con esta temática podemos destacar Minube o TravBuddy.

- Clasificación por contenido compartido: Esta clasificación diferencia a las redes sociales verticales en relación al tipo de publicación que comparten los usuarios con el resto.
  - Fotos: La finalidad principal de estas redes es la de almacenar, organizar, buscar y compartir fotografías. Las más importantes son Flickr o Pinterest.
  - Música: En este tipo de redes sociales los usuarios pueden escuchar, compartir y organizar música, además de conocer en tiempo real qué están escuchando los usuarios con los que está relacionado o con los que comparte afinidad musical. Actualmente existen un gran número de redes sociales cuyo principal objetivo es compartir música, de entre ellas destacan Spotify o Deezer.
  - Vídeos: En estas redes sociales, cada vez más populares, el objetivo es compartir vídeos sobre las actividades que están realizando los usuarios en cada momento. Permiten comentar y valorar los contenidos que publican los usuarios. Las más importantes actualmente son, la conocida por todos, Youtube, y Vimeo.
  - Documentos: En ellas el objetivo es publicar, compartir y acceder a documentos en diferentes formatos. La más importante de ellas es Scribd.
  - Presentaciones: Muy similares a las anteriores, el objetivo principal de estas redes sociales es compartir presentaciones, típicamente del ámbito profesional o académico. En este caso, la más importante es SlideShare.

    - Noticias: En este tipo de redes sociales, el principal objetivo es mantener informado al usuario en tiempo real, de aquellos sucesos que son de interés para él. Además las redes sociales con contenido informativo ofrecen la posibilidad de que los usuarios puedan compartir sus impresiones a cerca de una misma noticia. Una de las más importantes redes en este campo es Menéame.
    - Lectura: Por último, y en relación al tipo de contenido nos encontramos las redes este tipo de redes sociales, donde los usuarios pueden compartir y clasificar las libros, lecturas y sus opiniones sobre ellas. Ejemplo de estas redes son WeRead o Entrelectores.
- Clasificación por actividad: Con esta última clasificación definimos las redes sociales en cuanto a las posibilidades de interacción que ofrecen.
    - Microblogging: La principal característica de estas redes, es la posibilidad de hacer publicaciones en un corto espacio de texto. La más conocida y popular, que aunque, si bien está encuadrada dentro de las redes verticales, tiene un propósito general, es Twitter.
    - Juegos: Aunque no coincide con los elementos básicos de una red social, este tipo de plataformas relacionan usuarios a través de su participación en un mismo videojuego. Existen un gran número de plataformas que permiten este tipo de interacción, de entre las que destacamos Habbo o World of Worcraft.
    - Geolocalización: El elemento definitorio de este tipo de redes es la posibilidad de localizar una publicación (que puede referirse a una persona, edifico o lugar), en un sitio geolocalizado concreto. Aunque existen redes con este propósito de forma específica, como Panoramio, otros tipos de redes sociales ofrecen cada vez la opción de geolocalización en sus publicaciones. Ejemplo de ello son Twitter o Instagram.

Como podemos ver, existe un gran número de redes sociales que pueden agruparse en torno a un gran número de criterios. De forma adicional, parte de la información que se genera en estas redes sociales puede ser accedida a través de Application Programming Interfaces (APIs) que las redes sociales ponen a disposición pública con objetivos estadísticos, comerciales, etc. Esta información ha sido objeto en los últimos años de investigación de cara a obtener información acerca de la sociedad en general y con respecto a temas específicos. Actualmente existen algunas herramientas que son capaces de explorar una o varias redes sociales para conseguir diferentes objetivos. Un ejemplo es la herramienta Topsy genera estadísticas tomadas a partir de las publicaciones generadas en Twitter en torno a diferentes criterios que establece el usuario de la aplicación. Por ejemplo, un usuario puede ver una gráfica de los últimos treinta con el número de publicaciones que contienen una determi-

nada palabra, o que *mencionan* a un usuario concreto. Esta aplicación puede ser utilizada, por ejemplo, por partidos políticos para conocer que repercusión tiene su representante y los representantes del resto de partidos de cara a unas elecciones. Otra herramienta es ECO, utilizada por el periódico *20 minutos* para conocer la repercusión que tienen las noticias publicadas a través de su página Web tanto en la misma Web como en diferentes Redes Sociales (Facebook, Twitter, LinkedIn o Google+ principalmente). Esta herramienta muestra gráficas que las primeras 40 horas de vida de una noticia, para conocer la repercusión que tuvo la misma, por el número de veces que se *compartió* la noticia por un lector a una Red Social. Además esta herramienta permite ver los comentarios que realizan los usuarios en las redes sociales sobre cada una de las noticias.

Así pues, para cumplir con los objetivos marcados en el presente proyecto, será importante la elección de una red social con un número de usuarios y actividad tal que sea representativa para mostrar la actividad cotidiana de una ciudad.

3.2 Crowdsensing: Características y arquitectura

El crecimiento de Internet y el desarrollo de un gran número de nuevas tecnologías han hecho que surjan nuevas aplicaciones, tecnologías e instrumentos que tienen como núcleo el uso de Internet para ofrecer un servicio. Todas estas nuevas tecnologías se encuentran en objetos de uso cotidiano como teléfonos móviles, relojes o incluso electrodomésticos, que han formado un nuevo ecosistema de telecomunicaciones que hoy en día conocemos como *Intertet of Things (*IoT*)*[Kop11]. Este nuevo entorno ha propiciado la aparición de un conjunto de técnicas que tienen como principal objetivo la recolección de la información potencial que se puede extraer de todos estos nuevos dispositivos. Una de estas técnicas es la denominada *Crowdsensing*, cuyo objetivo es obtener información para contribuir a mejorar un objetivo social o conocer la relevancia de un evento de interés para una sociedad concreta, todo ello gracias a la recolección de la información que ofrecen los individuos de la comunidad mediante el uso de diferentes dispositivos (smartphones, sensores vehiculares...) o herramientas, como las redes sociales, en nuestro caso. En el presente proyecto, nos centramos en la definición de crowdsensing en su sentido más puro, puesto que la información que va a recogerse procede únicamente de la red social con independencia al dispositivo que esté utilizando el individuo.

Existen variantes de este este término que se centran únicamente en la recolección de la información procedente directamente del smartphone del individuo, término que actualmente se conoce como *Mobile Crowdsensing* [Sun13]. En este tipo de plataformas, existen varías características derivadas de la naturaleza del sensor, en este caso, un smartphone, que permiten distinguirlas de las redes de sensores convencionales. En primer lugar, los dispositivos móviles que participan como sensores poseen muchos más recursos de computación, comunicación y almacenamiento que un sensor convencional. En segundo lugar, el despliegue de

la red ya está hecho. Las personas llevan consigo el dispositivo móvil, lo que permite recopilar información de la zona sin necesidad de instalar ningún dispositivo adicional. Por ejemplo, en lugar de instalar cámaras en una carretera para medir el volumen de tráfico, podemos utilizar la información que nos ofrecen los smartphones de los conductores.

Puesto que las características de las plataformas de crowdsensing difieren bastante de las características de las redes de sensores convencionales, sobre todo en lo que se refiere a la naturaleza del sensor, aparecen varios problemas relativos a la información que puede obtenerse de los sensores de los que dispone el smartphone, ya que pueden existir aplicaciones que recopilen información sensible de los usuarios. Por ejemplo, si se toman los datos de movilidad de un usuario, a través de la recolección de datos del GPS, se puede inferir información privada como, por ejemplo sus movimientos habituales o la ubicación de su casa y lugar de trabajo. Sin embargo, si la información que se obtiene es relativa a una comunidad, la información obtenida de los GPS de los usuarios, puede ofrecer información sobre la congestión de determinadas áreas de una ciudad. También es importante asegurarse que los datos de una persona concreta no lleguen a terceros no confiables, o que los datos obtenidos puedan ser manipulados por fuentes externas no autorizadas. Así pues, tanto la seguridad como la integridad de los datos, son los dos desafíos principales de este tipo de aplicaciones.

El principal método para mantener la seguridad de los datos es la *anonimización*, que borra cualquier información que pueda permitir identificar al usuario antes de que la información se comparta con terceras partes. Con respecto a la integridad de los datos, existen grandes retos, puesto que las soluciones propuestas implican diferentes técnicas de cifrado que pueden suponer una pérdida en el rendimiento de la aplicación, e incluso la alteración de los datos obtenidos al intentar precisamente que estos no se vean alterados.

Por eso, para dar soporte a esta nueva técnica, es necesario desarrollar una arquitectura que permita recopilar los datos que ofrecen los usuarios al utilizar sus smartphones, sin que la arquitectura suponga un problema para seguridad de los datos personales que los usuarios intercambian con la red, pero que sirva para obtener información de valor a cerca de las actividades que se realizan o de los temas de relevancia para la comunidad sobre la que se está recopilando la información. En el artículo «*Incentive Mechanisms for Mobile Crowd Sensing:Current State and Challenges of Work*» [Sun13], se propone una arquitectura capaz de cumplir con estos objetivos. Como puede verse en la Figura 3.2, la arquitectura propuesta tiene tres componentes principales: «*Requesters*» o solicitantes, «*Participants*» o participantes y por último la Plataforma de crowdsensing encargada del procesamiento de datos.

- Solicitantes: Bajo este rol se encuentran los usuarios de la aplicación de crowdsensing. El principal reto que aparece en esta parte de la arquitectura es obtener un sistema de representación de los datos obtenidos por la plataforma, que sea capaz de responder a las necesidades de visualización concretas de cada uno de los solicitantes o clien-

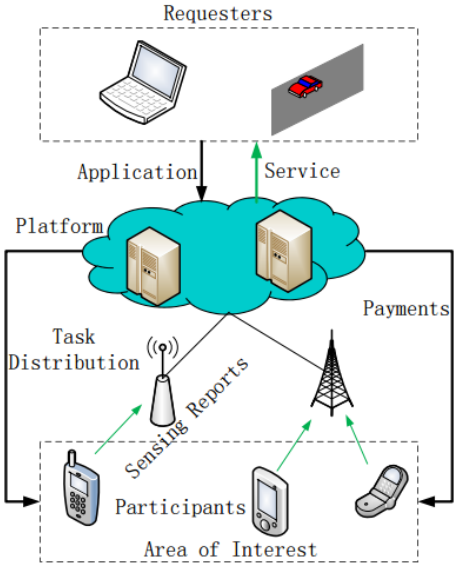

Figura 3.2: Arquitectura de un sistema de Crowdsensing [Sun13].

tes. Así, el objetivo es obtener un visor de contenidos que sea capaz de adaptar los datos de los que dispone a la vista solicitada por cada uno de los clientes, para que la información de la vista sea útil y representativa.

- Participantes: Son todos aquellos dispositivos que disponen de un sensor que recoge y envía los datos a la plataforma. En este caso, podemos encontrarnos con dos situaciones diferentes: aplicaciones con un número fijo de participantes, y aplicaciones con un número variable de participantes. Las primeras, están centradas en escenarios no orientados a conexión, donde los participantes, envían los datos recopilados a la plataforma, y luego es ésta la que se encarga de seleccionar un subconjunto del total. En cambio, en las aplicaciones donde el número de participantes es variable, cada participante ofrece de forma secuencial los datos obtenidos a la plataforma, que los recopila y procesa en el momento en que los datos son recibidos.
- Plataforma: Se encarga de procesar los datos obtenidos de los participantes, para conseguir la información que se le enviará a los solicitantes.

3.3 Ciudades Inteligentes: Arquitectura y servicios

El término Ciudad Inteligente aparece, al igual que las plataformas de *Crowdsensing*, por la revolución tecnológica de las últimas décadas, donde el desarrollo de las Tecnologías de la Información y la Comunicación (TIC) ha revolucionado la vida cotidiana de las personas. Esto ha hecho, que las ciudades deban adaptarse también a estos cambios, ofreciendo a los ciudadanos nuevas formas de comunicarse, trabajar, viajar... todo ello desde una perspectiva centrada en la sostenibilidad y eficiencia energética. Así pues, la definición de Ciudad Inteligente puede abordarse desde dos perspectivas. Si nos referimos a términos gubernamentales

y políticos, una Ciudad Inteligente se refiere todas las infraestructuras desplegadas sobre la ciudad, que contribuyen a mejorar y conseguir el desarrollo sostenible de la misma. Desde una perspectiva tecnológica, nos referimos a Ciudad Inteligente como ciudad interconectada. Esta interconexión permite capturar información de diferentes fuentes como sensores, dispositivos personales, smartphones, cámaras e incluso las redes sociales entendidas como redes de sensores humanas. Además, para que la ciudad pueda ser denominada «inteligente», todos los datos obtenidos deben ser analizados y modelados para obtener información que permita tanto mejorar la toma de decisiones, como conocer el impacto de un suceso [NP11].

El origen de este término está ligado a dos factores: el aumento de la población mundial y su creciente migración de las zonas rurales a los centros urbanos, y la creciente preocupación por la escasez de recursos naturales, que puede poner en peligro a la población mundial, y por el cambio climático. Ante estos nuevos retos para la sociedad, han sido muchas las ciudades que han intentado cambiar su infraestructura para conseguir una gestión sostenible de sus servicios, satisfaciendo además las necesidades de la ciudad y sus ciudadanos. Así, existen una serie de servicios ofrecidos por las ciudades que permiten denotarlas como inteligentes: [PSB+13]

1. Economía Inteligente (Competitividad): Se centra en el espíritu innovador y empresarial, conseguir una buena imagen tanto económica como de las marcas comerciales, la productividad, la flexibilidad del mercado laboral y la capacidad de evolución y transformación del motor económico de la ciudad.
2. Gobierno Inteligente (Participación Ciudadana): En esta categoría de servicios se encuentran aquellos relacionados con la participación de los ciudadanos en temas de gobierno como la toma de decisiones o los servicios públicos y privados, además de conseguir un gobierno transparente donde los ciudadanos conozcan las estrategias y perspectivas políticas.
3. Personas Inteligentes (Capital Social y Humano): Las ciudades inteligentes necesitan personal cualificado para conseguir su evolución y crecimiento. Por lo tanto, se deben considerar aspectos como el nivel de cualificación, pluralidad social y étnica, creatividad y participación en la vida pública.
4. Movilidad Inteligente (Transportes y TIC): Los medios de transporte son esenciales para conseguir el desempeño de las funciones vitales para el progreso de la sociedad. Sin embargo, los medios de transporte actuales no son sostenibles (favorecen al cambio climático, generan emisiones contaminantes y ruido...). Por eso, se están desarrollando nuevas tecnologías de transporte para las Ciudades Inteligentes que permitan la accesibilidad local, internacional y de las TIC, como sistemas de transporte sostenibles e infraestructuras seguras.
5. Vida Inteligente (Calidad de Vida): Una ciudad inteligente debe tener como objetivo

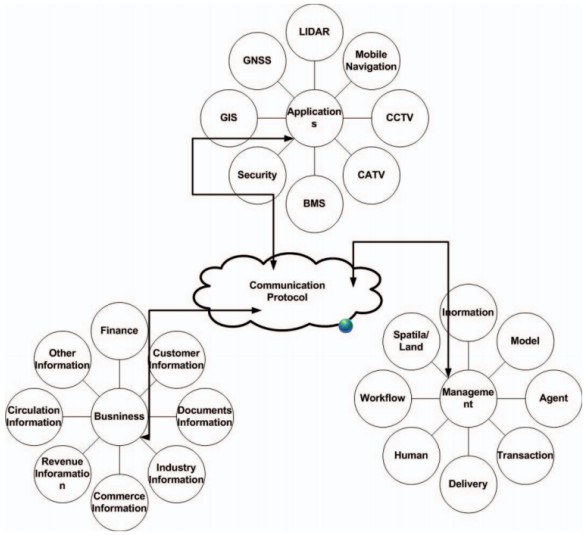

Figura 3.3: Arquitectura de negocio, gestión y aplicaciones de una Ciudad Inteligente [AHRSA09].

> garantizar el bienestar de los ciudadanos ofreciendo servicios que mejoren su experiencia en la ciudad. Estos servicios incluyen incluyen aplicaciones para mejorar y promover la cultura, el ocio, la educación y la salud.

Así pues, el fin de una ciudad inteligente es el de conseguir una gestión eficiente de los recursos y servicios de la ciudad, manteniendo siempre cubiertas las necesidades de los ciudadanos. Pero no es suficiente con desplegar servicios inteligentes de manera independiente, si no que toda la tecnología necesaria para el desarrollo de una ciudad inteligente debe ser capaz de desplegar una infraestructura capaz de recolectar datos de la ciudad de todas la fuentes posibles, favoreciendo a la transmisión, almacenamiento y análisis de datos de forma conjunta para ofrecer servicios a los ciudadanos.

Para conseguir que todos los servicios ofrecidos por una Ciudad Inteligente sean capaces de interrelacionarse, es necesario una arquitectura sea capaz de abarcar la mayoría de las redes, procesos, aplicaciones y actividades asociadas a los servicios de la misma [AHRSA09]. La Figura 3.3 muestra una arquitectura que cumple con este propósito, para el cual está dividida en cuatro unidades principales. La primera de ellas, está formada por las aplicaciones que están relacionadas con la monitorización de los activos de la ciudad, tales como las imágenes obtenidas por satélite, GPS o tecnologías láser, entre otras.

Todas estas aplicaciones son capaces de trabajar de manera independiente, pero es necesario un elemento que permita la interacción entre ellas. Este elemento es la unidad de negocios, cuyo fin es el de utilizar las aplicaciones de la ciudad inteligente. Este elemento de la arquitectura suele utilizarse para diversos fines, de entre los que destacan: obtención de información de los usuarios para la monitorización del comportamiento público, obtención de estadísticas de mayor fiabilidad, información de la industria para conocer la demanda del

mercado o información de los ingresos obtenidos para conocer mejor las actividades comerciales que se dan en la ciudad. El tercero de los elementos de la arquitectura es la unidad de gestión, cuyo objetivo es definir las relaciones, reglas, estrategias y políticas existente entre las aplicaciones y sus correspondientes unidades de negocio. Por último se necesita un protocolo de comunicaciones que se encargue de comunicar las tres unidades entre sí. Dependiendo de la naturaleza de las aplicaciones o de la distribución de todos los componentes a lo largo de la ciudad, será conveniente utilizar la red de cableado convencional, fibra óptica, o protocolos inalámbricos como el Bluetooth, Wi-Fi o GSM.

Existen numerosos ejemplos de ciudades inteligentes en todo el mundo, de entre las que destacan Viena, Toronto, París, Nueva York, Londres, Tokio, Berlín, Copenhague, Hong Kong y Barcelona [Coh12]. Además en España, nos encontramos con la Red Española de Ciudades Inteligentes (RECI), asociación poner en contacto los diferentes proyectos de ciudades inteligentes que se están desarrollando actualmente en España, con el objetivo de mejorar aspectos como la gestión sostenible, el ahorro energético y la calidad de vida de los ciudadanos. Actualmente, el proyecto RECI está formado por 60 ciudades, donde encontramos ciudades como Alicante, Barcelona, Málaga, Gran Canaria, Madrid o Ciudad Real.

Así pues, el propósito de una ciudad inteligente es el de ofrecer a los ciudadanos servicios que se adapten a sus necesidades gracias a la recolección de información de diferentes fuentes, y manteniendo siempre una gestión eficiente de los recursos de los que se dispone.

3.4 Técnicas de recolección de datos: Minería de datos y análisis de resultados

Una vez descrito el concepto de red social, debemos analizar la problemática de extraer información relevante a partir de los datos que ofrece la misma. En una red social, los usuarios comparten todo tipo de contenidos ya sean opiniones propias, comentarios acerca de sus actividades cotidianas, noticias y sucesos de interés... además, permite a organizaciones gubernamentales, empresas y diferentes personalidades de índole público conocer la repercusión que tienen en la sociedad sus movimientos, tomando como métrica el número de publicaciones relacionadas con la propia institución o personaje. Así pues, una red social supone una gran fuente de conocimiento, que puede extraerse con la ayuda de técnicas de minería de datos. El uso de una disciplina como la minería de datos está justificada en este tipo de análisis, ya que, para la extracción de información se necesita analizar previamente una gran cantidad de datos.

3.4.1 Minería de datos: Técnicas, tipos y herramientas.

Tal y como se expone en [Zub14], las técnicas de minería de datos utilizadas en redes sociales pueden dividirse en varios grupos: técnicas que se apoyan en la teoría de grafos, técnicas de análisis de opiniones, técnicas de clasificación de datos y herramientas para la

detección del tema.

Las técnicas que utilizan grafos con herramienta principal para el análisis de información, son técnicas utilizadas históricamente en el análisis de redes sociales. Esta técnica se utiliza para conocer parámetros importantes en la red, tales como las relaciones entre nodos. Un ejemplo típico del uso de este tipo de técnicas es el de conocer la influencia de un determinado usuario por el número de seguidores que tiene. Así, y si representamos de manera visual este tipo de relación, un usuario será más popular cuantos más seguidores tenga. La Figura 3.4 representa claramente esta situación, asociamos la importancia de un nodo al número de vecinos que tenga.

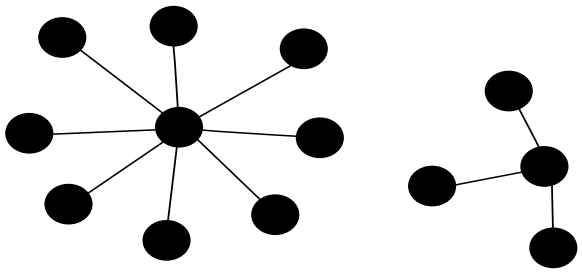

Figura 3.4: Comparación de influencia entre dos nodos

A partir de este concepto, y teniendo en cuenta las relaciones que aparecen entre los usuarios de la red social, disponemos de varias técnicas para conocer la distribución de los usuarios en la red social y mejorar la experiencia de los mismos en cuanto al establecimiento de nuevas relaciones entre usuarios afines o cercanos:

- Detección de la comunidad utilizando agrupamiento jerárquico: Definimos comunidad como un grupo pequeño de usuarios que forman parte de una red de tamaño mayor. Una comunidad está formada por usuarios que comparten intereses y por lo tanto, mantienen un mayor grado de relación entre ellos que entre otros usuarios de la red. Así, para la detección de una comunidad utilizando la teoría de grafos, la técnica más utilizada es el agrupamiento jerárquico. Esta técnica, combinación de otras, tiene como objetivo construir grupos de nodos en relación al número de relaciones existentes entre los nodos que forman en grupo, para así conseguir dividir la red en grupos. La técnica del agrupamiento de vértices es una de las técnicas complementarias al agrupamiento jerárquico y establece que, para resolver los vértices de un grafo incluir al mismo dentro de un espacio vectorial de tal modo que la longitud entre los pares de vértices que componen el espacio se puede medir, es decir, los nodos que componen un mismo grupo pueden unirse para formar este área. El agrupamiento jerárquico se sirve también

de la utilización de medidas de equivalencia estructural para determinar la propensión a una relación entre dos usuarios aún no conectados, en relación al número de amigos que tienen en común.

- Sistema de recomendación entre comunidades de la red social: En este caso, el objetivo es sugerir a un usuario de la red, que establezca una nueva relación con usuarios con los que aún no está conectado directamente pero con el que se relaciona a través de otros elementos. Normalmente, el método de recomendación utilizado emplea medidas de equivalencia estructural, pero puede utilizar otros métodos que sugieren usuarios en relación al contenido de sus publicaciones o incluso, en relación con los intereses que el usuario indica en su perfil.

Por otro lado, nos encontramos las técnicas que se centran en el análisis de opinión. Esta técnica surge con el objetivo de conocer el impacto o influencia que tiene un suceso, producto o servicio en relación con lo que se escribe sobre él en las diferentes redes sociales. Este indicador tiene gran importancia sobre todo en relación con técnicas publicitarias o electores, ya que, por ejemplo, un partido político puede conocer la opinión que se tiene sobre su candidato, o una empresa puede definir su estrategia comercial en relación a los gustos y opiniones de la gente sobre los productos que comercializan. Las técnicas de minería de datos utilizadas en este caso consisten en detectar si la opinión que se tiene sobre el elemento es positiva o negativa.

Otro tipo de técnica utilizada para el análisis de redes sociales, es el que trata de clasificar el contenido de los datos de la red social. Dentro de los métodos de clasificación que se utilizan se distinguen tres tipos en relación a la técnica de aprendizaje que siguen. Así distinguimos técnicas de clasificado no supervisadas, semisupservisadas y supervisadas [Zub14].

- Clasificación no supervisada de la información de las redes sociales: Esta técnica suele utilizarse para contabilizar la cantidad de críticas positivas o negativas que tiene una publicación teniendo en cuenta los adjetivos y adverbios del contenido que valora la publicación. Otra técnica muy utilizada, denominada Part Of Speech (POS) tagging, clasifica la información en cuanto a la valoración positiva o negativa de los adjetivos que van etiquetados en una publicación. Esta técnica supone que las palabras que se etiquetan en una publicación son utilizadas por los usuarios para resumir y clasificar el contenido de su publicación. Existen también otras técnicas de clasificación no supervisadas mas complejas, como la orientación semántica, cuyo objetivo es determinar si la valoración global de una frase es positiva o negativa no sólo por el significado de los adjetivos, si no por el tono y contexto completo de la misma. Así, por ejemplo la expresión «¡Qué película tan buena!», puede estar siendo utilizada con ironía, y querer expresar en realidad que no lo fue.
- Clasificación semisupervisada: En este caso, la diferencia principal con las técnicas

no supervisadas es que, para cada nueva valoración, se almacena en un diccionario si el significado de las palabras y adjetivos etiquetados que se valoran, puede clasificarse como positivo o negativo. La utilización de un diccionario ayuda a determinar con mayor facilidad si un contenido sin etiquetar compone una valoración positiva o negativa.

- Clasificación supervisada: Esta última técnica de supervisión es la más avanzada y compleja, utiliza herramientas de aprendizaje basadas en minería de datos para clasificar la información. El aprendizaje es un elemento clave a la hora de clasificar la información que se obtiene de una red social ya que, al ser una plataforma dinámica en lo que se refiere al número y tipo de usuarios que la utilizan y al tipo de contenido de las publicaciones, es importante que la valoración que se tiene sobre el cometido de la publicaciones cambie según evoluciona la red social.

El último grupo de herramientas para utilizar la minería de datos en redes sociales, tiene como objetivo la detección del tema que tiene mayor importancia en ese momento en la red social. Los algoritmos que se encargan de la detección del tema, son algoritmos que analizan la corriente de datos en tiempo real para distinguir contenidos que se relacionan con un evento de los que no. Esta última técnica es muy utilizada por las redes sociales para sugerir a los usuarios cuáles son los temas del momento. Twitter, por ejemplo, utiliza palabras etiquetadas llamadas «hashtag» para determinar el tema de una publicación. Así, cuando un hashtag está siendo utilizado en muchas publicaciones, se convierte en tema principal o «trending topic». Además, y puesto que en Twitter pueden utilizarse varios hashtag en una misma publicación pueden utilizarse reglas de asociación para detectar los cambios que sufre una tendencia. Una de las técnicas de minería de datos más utilizada en estos casos es Association Rule Mining (ARM) que estudia la evolución del volumen de publicaciones que hablan sobre una tendencia en un periodo de tiempo comprendido entre $t.$ y $t + 1$. Esta técnica básica ha evolucionado en otras más complejas, por ejemplo Rule Matching Threshold (RMT), cuya diferencia con ARM es que RMT es capaz de establecer un umbral para distinguir cuando un tema se ha convertido en tendencia y cuando deja de serlo [Zub14].

Como podemos ver, estas técnicas analizan los datos de las redes sociales atendiendo tanto a la forma de la red social en cuanto a cómo están relacionados sus elementos, como al contenido de las publicaciones para determinar, en ambos casos, la importancia que tiene un usuario o tema concreto.

3.4.2 Análisis de Resultados: Análisis de normalidad como técnica para la detección de eventos

Todas las técnicas mencionadas anteriormente, en especial las que tienen como objetivo clasificar la información mediante aprendizaje continuo y las que pretenden detectar el tema principal en un instante determinado, se basan en la detección de anomalías como elemento

fundamental para conseguir sus resultados. Una anomalía no tiene unas características concretas, no podemos detectarla mediante las búsqueda de patrones concretos, si no que una anomalía se caracteriza, precisamente por no contener características que se entienden como normales. Pero,antes de poder detectar una anomalía debemos averiguar qué es «normal» [DF14a].

Una de la formas básicas para detectar la normalidad es construir un modelo probabilístico de manera progresiva [DF14a]. Supongamos que la mejor estimación que tenemos de una observación i es π_i. La verdadera probabilidad es p_i. Debido a que el modelo elegido es un modelo probabilístico, los valores de π_i se ven limitados de la siguiente manera:

$$\sum_i \pi_i = 1, \pi_i \geq 0$$

$$\sum_i p_i = 1, p_i \geq 0$$

Esto significa que si hacemos π_i grande para algunos periodos de i, debemos hacerlo pequeño para otras i. Además y dado que ningún valor de π_i es menor que cero, tenemos que los valores más pequeños estarán muy cerca del cero, pero nunca menores. Así podemos tomar un modelo de inferencia matemática tal que si tomamos el valor medio de $-\log \pi_i$ más pequeño posible, podemos probar que las probabilidades estimadas de π_i estarán lo más cerca posible de la verdadera probabilidad p_i. De hecho:

$$\max_{\pi} \sum_i p_i \log \pi_i = \sum_i p_i \log p_i$$

donde el valor máximo solo se alcanza si $\pi_i = p_i$ para todo i.

Así cuando sucede un evento raro, su valor de π_i será muy pequeño, pero su valor $-\log \pi_i$ será muy grande. Puede parecer que estos eventos inferirán negativamente en la búsqueda de la normalidad, sin embargo, ya que la probabilidad de que suceda el evento es muy baja, al calcular la media ponderada con las probabilidades, el peso de las anomalías tendrán un aporte muy bajo.

Ahora bien, para descubrir un buen patrón de normalidad, no basta sólo con tener un modelo de aprendizaje automático. El proceso de descubrimiento implica también la visión humana: el individuo que realiza el análisis debe interactuar con el modelo para decidir qué situaciones tienen sentido en relación con el contexto. Por ejemplo, el modelo de probabilidad puede detectar un pico de actividad en la red social y detectarlo como un suceso anómalo. Sin embargo, si ese pico se sucede de manera periódica – por ejemplo, los fines de semana – el pico de actividad deja de ser una anomalía para convertirse en una situación normal. Introducir criterios que no forman parte del modelo probabilístico y que dependen del entorno que conoce el individuo (mayor actividad en la red social unos meses del año que

otros, eventos que, aunque puntuales se sabe que ocurrirán como fiestas y eventos anuales...), pueden ayudar a mejorar la búsqueda de la normalidad y el patrón matemático.

También debe tenerse en cuenta que a la hora de construir una máquina con conocimientos suficientes para detectar una anomalía, primero se debe identificar cual es la mejor forma de presentar los datos que tomará el algoritmo de detección, y a continuación se deben tomar datos suficientes para que se construya un buen modelo de normalidad antes de que la máquina sea capaz de detectar cualquier anomalía. Es decir, en primer lugar, debemos introducir patrones que consideremos normales para que el algoritmo sea capaz de enmarcar y limitar el concepto de normalidad. Así, una vez construido el patrón de normalidad, puede comenzarse la búsqueda de anomalías.

Al igual que para la búsqueda de la normalidad se necesita del conocimiento humano, para que la máquina sea capaz de detectar anomalías, es preciso que demos primero un ejemplo al sistema de qué es anómalo. Por ejemplo, si observando el comportamiento normal de la red social descubrimos, por ejemplo, que por la confirmación de la asistencia a un evento próximo en la ciudad por parte de un personaje famoso, el volumen de publicaciones se dispara durante un periodo corto de tiempo y el tema del que se está hablando tiene que ver con este acontecimiento, entonces podemos indicar a nuestra máquina que esa situación es totalmente eventual.

Por otro lado, otro punto importante en la detección de anomalías, es el hecho de que una situación anómala en principio, puede convertirse en una situación normal si se repite un determinado número de veces. Nuestro modelo de detección de anomalías deberá construir umbrales para detectar esta situación y adaptar el patrón de normalidad a los nuevos eventos, en primer lugar anómalos, que ahora dejan de serlo. Así pues, si queremos desarrollar un modelo capaz de detectar anomalías, sea cual sea la complejidad del mismo, éste debe contener tres ingredientes principalmente: una referencia de aquello que es normal, una métrica para identificar aquellas cosas que pueden ser anómalas y un límite a partir del cual considerar que lo que se está midiendo es una anomalía. La definición de este umbral es quizás el proceso más complejo en el desarrollo del modelo de detección de anomalías, pues debe ser un umbral lo suficientemente preciso para que no se vea alterado por un ruido que en realidad no es un anomalía, y del mismo modo, no deje de identificar eventos que son realmente anómalos.

Así pues, a la hora de definir un modelo de normalidad y su correspondiente modelo de detección de anomalías, debemos tener en cuenta que, sea cual sea la complejidad del sistema desarrollado, es muy importante la intervención en primera instancia del ser humano como elemento motor y principal ayuda en el inicio del sistema, pues sus observaciones, su criterio, y sus ajustes en la decisión de qué es normal, ayudarán al sistema a desarrollarse por la línea correcta.

3.5 Análisis de herramientas a utilizar

En esta sección se analizan los elementos principales escogidos para el desarrollo del presente proyecto. Además se justifica su uso frente a otras tecnologías o herramientas de su misma clase.

3.5.1 Bases de datos orientadas a grafos

Cuando se va a desarrollar una aplicación donde una parte fundamental es la parte de la persistencia, es importante elegir la forma de almacenamiento de datos más adecuada. En nuestro caso, el objetivo es almacenar toda la información relativa al movimiento producido en una red social por los usuarios de una red social. Una parte clave de esta información son las relaciones existentes entre los elementos de la red social (usuarios, publicaciones, etiquetas...), por eso, el tipo de base de datos que vaya a utilizarse debe ofrecer la posibilidad de representar y acceder a las relaciones de forma sencilla y eficiente.

En nuestro caso, se ha optado por la elección de una Base de Datos Orientada a Grafos. Una Base de Datos Orientada a Grafos es un sistema de gestión de bases de datos que permite realizar las operaciones Create, Read, Update, Delete (CRUD) sobre los datos. Normalmente este tipo de bases de datos se crean para sistemas transaccionales.

A la hora de escoger una u otra tecnología de Base de Datos Orientada a Grafos, debemos tener en cuenta dos propiedades:

- El almacenamiento subyacente: Algunas bases de datos utilizan almacenamiento orientado a grafos de forma nativa, que está optimizado y diseñado para administrar grafos. Sin embargo, esto no es así en todas las tecnologías de Bases de Datos Orientadas a Grafos. En algunos casos, los datos se serializan en una base de datos relacional u orientada a objetos.
- El motor de procesamiento: Algunas definiciones consideran que una base de datos está orientada a grafos, cuando usan índices libres de adyacencia [1]. Sin embargo, la definición ofrecida por Ian Robinson, Jim Webber, and Emil Eifrem en su libro «Graph Databases» [RWE13], indican que cualquier base de datos que, desde la perspectiva del usuario se comporte como tal – permitiendo las operaciones CRUD sobre los elementos del grafo – puede considerarse una Base de Datos Orientada a Grafos. Ya que existen diferencias en el rendimiento respecto a las bases de datos que implementan este mecanismo, de las que no, se dice que una base de datos utiliza *procesamiento gráfico nativo* cuando utiliza índices libres de adyacencia.

Una vez definido el marco en el que se encuentran este tipo de bases de datos, podemos plantearnos el por qué del uso de una Base de Datos Orientada a Grafos en lugar de utilizar otras tecnologías más conocidas. Para ello, nos fijamos en la potencia que pueden ofrecer este

[1] Las bases de datos que cumplen esta propiedad son aquellas en las que, cada nodo almacena información sobre sus vecinos, y no existe un índice global para identificar las conexiones entre los nodos [DVMT14].

tipo de bases de datos. Una Base de Datos Orientada a Grafos, ofrece mayor rendimiento, flexibilidad y agilidad que otras bases de datos, cuando el objetivo es tener una base de datos con un gran número de relaciones entre elementos.

- Rendimiento: El rendimiento – entendido como el tiempo de respuesta ante una consulta a la base de datos – de una Base de Datos Orientada a Grafos es superior que el de otras bases de datos convencionales, por ejemplo, en relación al rendimiento de una Base de Datos Relacional. En estas últimas, cuando el volumen de datos crece, el rendimiento se deteriora de manera brusca, sin embargo, en una Base de Datos Orientada a Grafos, el rendimiento permanece prácticamente constante. Esto sucede, porque las consultas en estas bases de datos se lanzan sólo sobre una porción de los datos, por lo que el tiempo de ejecución de la consulta es proporcional a porción sobre la que se realiza la consulta, y no al tamaño total del grafo.
- Flexibilidad: Este tipo de bases de datos permite modificar su estructura conforme se avanza en el desarrollo de la aplicación, en lugar de tener que fijar una estructura previamente, aún cuando todavía no se conocen a la perfección las características ni propiedades de los datos a almacenar. Esto significa, que se pueden añadir nuevos tipos de relaciones entre elementos conforme van apareciendo, sin que esto implique modificar los datos que ya existen en la base de datos. Del mismo modo, se pueden añadir nuevas características a los nodos de un mismo tipo, si que ello implique ninguna migración de los datos.
- Agilidad: Puesto que la base de datos es flexible, ya que es libre de esquema[2], permite que el desarrollo de aplicaciones sea más rápido, siendo muy compatible su uso con el de técnicas ágiles de desarrollo de software.

Para justificar estas propiedades, y por tanto, el uso de este tipo de Base de Datos en el presente proyecto, la comparamos con las Bases de Datos Relacionales, ya que siempre han sido las bases de datos más utilizadas.

Históricamente, las Bases de Datos Relaciones han sido elegidas por los desarrolladores de aplicaciones para almacenar datos conectados y semi-estructurados. Sin embargo, en sus orígenes, las Bases de Datos Relacionales fueron creadas para almacenar formularios y tablas. Es por eso, por lo que, curiosamente, las Bases de Datos relacionales no son las más indicadas para representar las relaciones del mundo real, o en nuestro caso, de una red social. Las relaciones en el mundo real son algo más que una conexión entre elementos, tienen semántica, y diferente peso y fuerza según los elementos que se relacionen. Sin embargo, las Bases de Datos relacionales no lo permiten. Además, si el número de relaciones existente aumenta con el tiempo, el rendimiento de este tipo de base de datos caerá, haciendo incluso que la base de datos no sea escalable para la cantidad de relaciones diferentes existentes. Pa-

[2]Una base de datos libre de esquema es aquella que no necesita definir una estructura preliminar de los elementos que va a contener la base de datos antes de comenzar a insertar datos en ella.

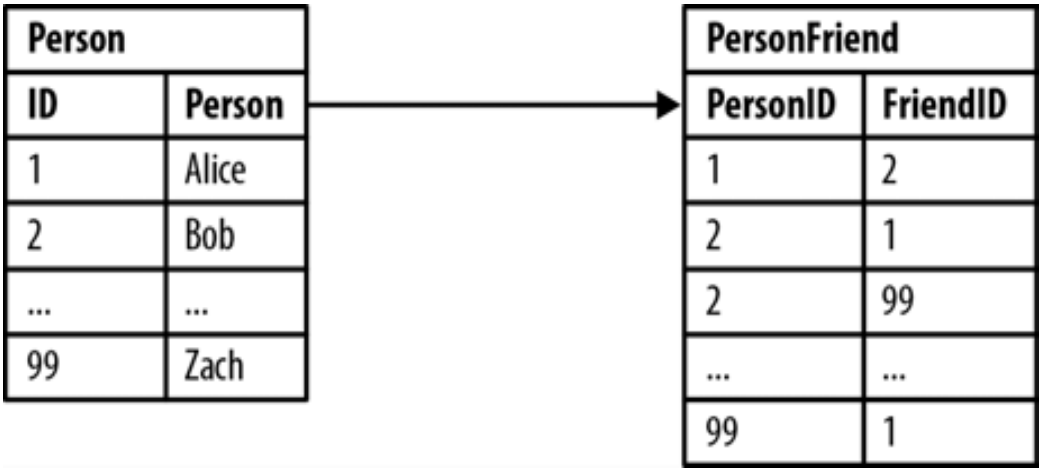

Figura 3.5: Modelado de relación de amistad en una Base de Datos [RWE13].

ra exponer esta situación, analizaremos el ejemplo de la figura 3.5 que muestra una relación simple entre tablas para expresar una relación de amistad.

Si queremos conocer, por ejemplo, cuáles son los amigos de Bob, debemos realizar una consulta con la siguiente forma:

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
 ON PersonFriend.FriendID = p1.ID
JOIN Person p2
 ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Listado 3.1: Consulta: Conocer los amigos de Bob

Esta consulta, no supone mucho trabajo computacional para la base de datos, pero la situación se complica si queremos hacer consultas del tipo *«¿cuáles son los amigos de mis amigos?»*. Estas relaciones de tipo jerárquico se implementan usando relaciones recursivas. Así, por ejemplo, para conocer cuáles son los amigos de los amigos de Alice, la consulta se complica:

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND-OF-FRIEND
FROM PersonFriend pf1 JOIN Person p1
 ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
 ON pf2.PersonID = pf1.FriendID
JOIN Person p2
 ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Listado 3.2: Consulta: Conocer los amigos de Alice

Si en lugar de una Base de Datos Relacional, utilizamos una Base de Datos Orientada a Grafos, encontramos, además, que la complejidad de la consulta se reduce considerablemente. Supongamos que nuestra base de datos está organizada tal y como se muestra en la Figura 3.6. Como podemos ver, tendríamos una serie de nodos, que podrían ser, por ejemplo de tipo «`Persona`», que se relacionan directamente con otros nodos mediante la relación «`KNOWS`», y que muestra la existencia de una relación de amistad entre los individuos. Como podemos ver, en una Base de Datos Relacional, esta relación debe representarse con una tabla intermedia, mientras que en la Base de Datos Orientada a Grafos, la relación se representa precisamente, con una relación explícita.

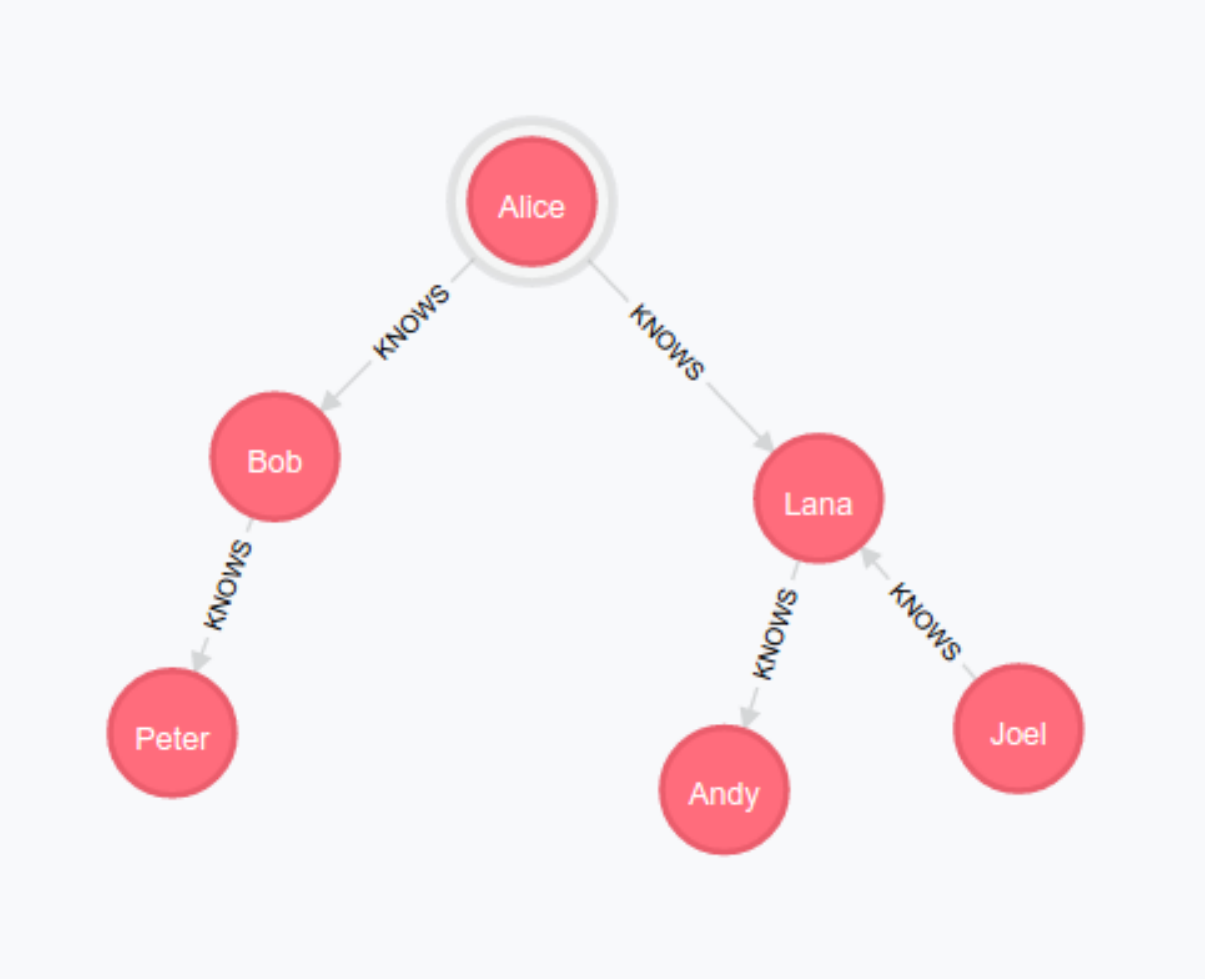

Figura 3.6: Modelado de relación de amistad en una Base de Datos orientada a Grafos

La consulta equivalente[3] que se utilizaría para conseguir los «*amigos de los amigos*» de Alice sería la siguiente:

```
MATCH (p1:Person {name:Alice})-[:KNOWS]->(p2:Person)-[:KNOWS]->(p3:
    Person)
```

[3]La consulta está realizada siguiendo las directivas del lenguaje *Cypher*, diseñado para consultar y actualizar el contenido de una Base de Datos orientada a Grafos [Tea15]. En el Anexo B puede encontrarse una guía de como realizar consultas sobre una Base De Datos Orientada a Grafos con Cypher.

```
RETURN DISTINCT p3
```

Listado 3.3: Consulta: Conocer los amigos de los amigos de Alice

A parte de la simplicidad de la consulta, la consulta realizada en la Base de Datos Relacional es computacionalmente más compleja a pesar de que sólo explora las relaciones a partir de un individuo. Si además queremos aumentar la complejidad de las consultas para conocer «*los amigos de mis amigos de mis amigos*», el coste computacional de la consulta crecerá rápidamente, y con consultas que impliquen cinco ó seis grados de amistad, el rendimiento de la consulta se deteriora considerablemente, debido a la complejidad de relacionar tablas de manera recursiva.

Además de este factor, en una Base de Datos Relacional, es necesario trabajar con un esquema predefinido, lo que hace aún más compleja la necesidad de establecer relaciones excepcionales, ya que darán lugar a valores nulos en el resto de los casos.

Como podemos observar, las relaciones entre datos en las Bases de Datos Relacionales se consiguen mediante conexión de datos implícita, bien añadiendo una clave en una tabla, o bien construyendo una tabla intermedia, que relacione varias tablas. Sin embargo, en las Bases de Datos Orientadas a Grafos, esta conexión es explícita, puesto que los datos se almacenan como datos conectados, es decir, cuando existen conexionen en el dominio, también deben existir en los datos.

Actualmente, existen diferentes tecnologías de bases de datos, que pueden clasificarse atendiendo a qué tipo de almacenamiento utilizan (si utilizan o no almacenamiento orientado a grafos) y a cuál es el motor de procesamiento de los datos. En la figura 3.7 podemos ver un diagrama con las tecnologías de bases de datos más importantes de este tipo. Como podemos ver, la Base de Datos Orientada a Grafos más «pura», en cuanto a que tanto el almacenamiento como el motor de procesamiento son orientados a grafos, es Neo4j, que es base de datos escogida para la realización de este proyecto.

Neo4j es una Base de Datos Orientada a Grafos implementada en Java por *Neo Technology*. Es un motor de persistencia basado en disco totalmente transaccional que almacena los datos estructurados en gráficos y no en tablas [Web12]. Su primera versión fue lanzada en febrero de 2010, y actualmente consta de dos ediciones: una de ellas puede adquirirse bajo la licencia libre de GNU General Public License (GPL) v3. Su otra edición consta de módulos y servicios adicionales tales como la copia de seguridad en línea o servicio de alta disponibilidad, y puede adquirirse bajo la licencia Affero General Public License (AGPL). La principal característica de Neo4j es su forma de almacenamiento de los datos. Cualquier elemento en Neo4j, se almacena en forma de nodo, arista o atributo. Además, tanto los nodos como las aristas pueden tener atributos, algo que permite añadir mayor semántica a las relaciones. Actualmente se encuentra en su versión 2.2.2, lanzada en Mayo de 2015.

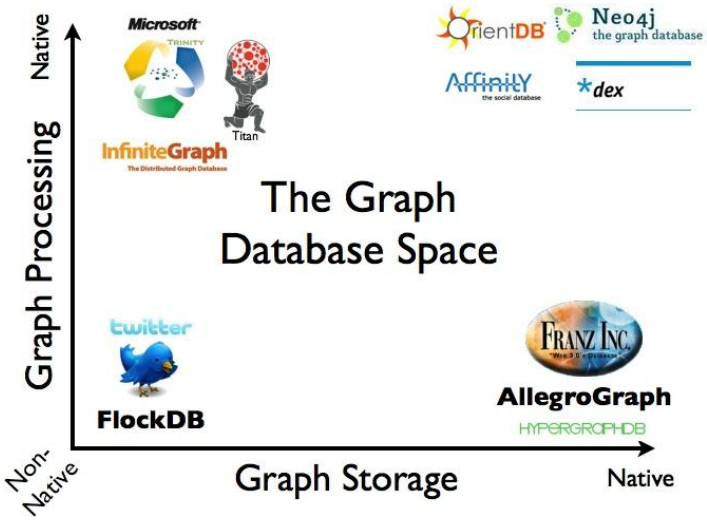

Figura 3.7: Perspectiva de las diferentes tecnologías de Bases de Datos [RWE13].

Así pues, la elección de una Base de Datos Orientada a Grafos está justificada tanto por la naturaleza misma de los datos que van a manejarse como por su rendimiento en el tipo de consultas que realizaremos.

3.5.2 Red Social Twitter

Tal y como hemos visto en la sección 3.1 de este capítulo, actualmente, podemos encontrar un un amplio conjunto de redes sociales, cuya finalidad atiende a fines muy distintos en cada una de ellas. Sin embargo, a la hora de elegir una red social que nos permita conseguir el objetivo del proyecto, necesitamos una red social que sea ampliamente utilizada, no sólo en número de usuarios, si no también en rango de edad para conseguir información representativa sobre una comunidad en concreto. El estudio sobre redes sociales del año 2015, y que realiza cada año la organización Interactive Advertising Bureau (IAB) [S+15] nos ofrece esta información. El estudio que se realizó en diciembre de 2014 con una muestra de 1163 individuos comprendidos entre 18 y 55 años establece que las tres redes sociales más utilizadas son Facebook, YouTube y Twitter, seguidas de Google+, LinkedIn e Instagram. Además de este dato ofrece otra información como la frecuencia de uso de las redes, siendo Facebook y Twitter las redes sociales más utilizadas, seguidas de Instragram, Youtube y Spotify. Gracias a este estudio, establecemos que la elección de la red social que utilizaremos en el proyecto estará entre Facebook, Twitter e Instagram.

Además de este estudio, otro criterio importante a la hora de escoger la red social con la que cumplir el objetivo del proyecto, es que la red social ofrezca algún tipo de API para acceder a las publicaciones de los usuarios que pertenezcan a la comunidad objetivo. Según las características de las APIs para el desarrollo de aplicaciones sobre las redes sociales objetivo, elegimos Twitter por dos motivos fundamentales. El primero es que aunque Facebook es la red social más utilizada y accedida por los usuarios, el concepto de API que ofrece no

permite obtener información sobre las publicaciones de los usuarios, si no que más bien, está enfocada a la creación de aplicaciones que trabajen como complemento a la red social. En segundo lugar, y ante la posibilidad de escoger entre Twitter e Instagram se escoge Twitter ya que, aunque las mismas ofrecen una API muy similar, Twitter es, según el estudio citado anteriormente, más utilizada y accedida que Instagram.

La API de Twitter es una API de tipo Representational State Transfer (REST). Una API, de forma general, se refiere a un conjunto de datos y funciones que permiten la interacción entre programas para intercambiar información. Más concretamente, si hablamos de API en el ámbito de las aplicaciones Web, una API representa la parte principal de un servicio Web, que escucha y responde las peticiones de los clientes. Una API REST no es más que una API que se utiliza en servicios Web modernos [Mas11]. que nos permite tanto acceder a los diferentes recursos de la red social (perfil de los usuarios, tweets ...), como interaccionar con la misma creando publicaciones, contestando mensajes, «siguiendo» a usuarios... a través de nuestro perfil.

Gracias a la API de Twitter, podremos acceder a las publicaciones de los usuarios que estén relacionados de algún modo con el entorno del que se van a capturar los datos.

3.5.3 Desarrollo de una plataforma web: Django

Desde que Tim Berners-Lee inventó la World Wide Web en 1989, y desarrolló la primera versión de HTML en 1990 [C+11], la Web ha sufrido enormes cambios en relación con en volumen y variedad de aplicaciones y servicios que podemos encontrar en la Web y en lo que se refiere a paradigmas de desarrollo Web. En lo que respecta al desarrollo de aplicaciones Web, la evolución va desde la creación de aplicaciones Web de manera tradicional, donde era necesario en manejo de un gran número de tecnologías y lenguajes (HTML, PHP, JavaScript, CSS, Ajax...), hasta la aparición de los *Frameworks* de desarrollo web, que permiten al desarrollador abstraerse de algunas de estas tecnologías, para conseguir aplicaciones más eficientes.

Actualmente existen multitud de frameworks que nos permiten crear una aplicación Web, como por ejemplo Ruby on Rails, CodeIgniter, Pyramid o Django. La principal diferencia entre unos y otros es el lenguage de desarrollo utilizado (PHP, Python, etc.) y es difícil evaluar, para un mismo lenguaje, qué framework es mejor[4].

En el caso de este TFG hemos optado por usar Django por dos motivos principales:

- El *know-how* previo del equipo que va a realizar este trabajo[5] reduce el tiempo de desarrollo ya que se está mas familiarizado con este framework.
- Existe una mejor integración con la base de datos orientada a grafos seleccionada para la capa de persistencia.

[4] por ejemplo Django y Pyramid usan Python
[5] estudiante y directores

- Django es uno de los frameworks de desarrollo Web más completos de entre los frameworks de este tipo en Python. La web Best Web Frameworks, nos ofrece comparativas de diferentes frameworks clasificados por lenguaje de programación. La Figura 3.8 muestra la comparativa.

Framework	Require	MVC	ORM	Auth	Cache	Validator	License	Docs	Website
Django	Python 2.3+	✔	✔	✔	✔	✔	BSD	#	
Pyramid	Python 2.3+	✔	✔	✔	✔	✔	BSD	#	
CherryPy	Python 2.3+	✔	✔	✔	✔	✖	BSD	#	
Bottle	Python 2.5+	✔	✖	✔	✔	✔	MIT	#	
Flask	Python 2.5+	✔	✖	✔	✔	✖	BSD	#	

Figura 3.8: Comparativa Frameworks de desarrollo Web en Python [Bes15].

Capítulo 4

Método y Fases de Trabajo

A La hora de escoger una metodología de trabajo que sea adecuada para la realización del proyecto, es necesario tener en cuenta tanto la naturaleza misma del proyecto como las condiciones en las que se va a realizar. En este capítulo se justifica la necesidad de seguir una metodología de desarrollo ágil como es el Scrum. A continuación, se explican cuáles han sido las fases por las que ha pasado el proyecto, incidiendo en cada una de ellas en cuáles eran los requisitos a cumplir y cuáles han sido las principales dificultades y soluciones encontradas para conseguir el objetivo final de cada fase.

4.1 Método de Trabajo

La elección de la metodología de trabajo a elegir pasa por escoger una que se adapte a la naturaleza del proyecto, es decir, que cumpla con la dinámica de trabajo que es deseable seguir. Aunque el presente proyecto consta de varias partes, una parte importante es la del desarrollo que una aplicación Web que permita al usuario conocer de una manera visual y sencilla qué está pasando en su ciudad gracias a los datos recogidos y procesados de la red social Twitter. Actualmente, el desarrollo de aplicaciones Web se ha convertido en una de las formas más frecuentes de desarrollar software, tanto es así, que en torno a ella ha surgido una disciplina que se encarga de controlar los procesos de trabajo y metodologías de desarrollo que se utilizan. Esta disciplina es la Ingeniería Web, cuyo principal objetivo es proponer un marco de desarrollo ágil que permita obtener una aplicación Web que se adapte a los requisitos del cliente y cambie cuando sea necesario dando una respuesta rápida a la nueva demanda surgida [PL09]. Así pues es necesario que la metodología elegida permita dividir el trabajo en tareas que son potencialmente entregables y sin errores. Por otro lado, y a parte de la naturaleza del proyecto, se debe tener en cuenta el entorno concreto donde se desarrollará el proyecto o producto. En este caso, el proyecto se encuadra en un marco de trabajo donde es muy probable que surjan problemas o nuevas tareas y requisitos, por lo que es imprescindible que la metodología no sea rígida, es decir, permite modificar la planificación o las tareas que se realizan en cada momento, en relación con los requisitos que previsiblemente irán apareciendo con el avance del proyecto, manteniendo siempre como línea de referencia los objetivos que se han definido en el Capítulo 2.

Con esto, se descarta el uso de metodologías de desarrollo clásicas que se basan en establecer una planificación inicial que se mantendrá durante todo el proyecto sin modificar las tareas que lo componen, ni los requisitos iniciales. Así, se optará por una metodología de desarrollo ágil cuyo principal objetivo es desarrollar proyectos de forma iterativa e incremental, permitiendo definir nuevos objetivos en cada iteración. Además la elección de una metodología ágil nos permitirá entregar en intervalos cortos de tiempo[1] un proyecto funcional y que el cliente[2] pueda valorar.

Con el auge de este tipo de metodologías tenemos un amplio abanico de ellas donde elegir: eXtreme Programming (XP), Adaptative Software Development (ASD), Scrum o Crystal Clear, entre otras. De entre todas ellas, *Scrum* es la que más se adapta a las necesidades del proyecto por los motivos que se enumeran a continuación:

- Es una metodología para el desarrollo de proyectos en general, por lo que con ella podremos dar cabida tanto a la planificación del despliegue del entorno como al desarrollo de la aplicación propiamente dicha.
- Una parte fundamental para el desarrollo de este proyecto es el seguimiento del mismo por parte de los tutores. Scrum nos permitirá llevar este seguimiento ya que uno de sus pilares es la celebración de reuniones de forma regular que tienen como objetivo coordinar y planificar el proyecto de forma incremental.

Así, durante el desarrollo del proyecto, se seguirán las siguientes condiciones que derivan del uso de Scrum:

- Todas aquellas nuevas peticiones por parte del coordinador se dividirán en tareas que entrarán en una pila de tareas priorizadas, que posteriormente pasarán a formar parte de un sprint.
- Cada una de las fases de trabajo se corresponderá con un sprint, que tendrá una duración aproximada de dos semanas.
- Al final de cada sprint se realizará una reunión con el coordinador donde se valorará lo que se ha hecho en el sprint recién finalizado y se planificarán los objetivos del nuevo sprint.
- Cada nuevo sprint estará compuesto por las tareas más prioritarias.
- Dentro de cada tarea se seguirá un flujo de trabajo bien definido, expuesto en la Figura 4.1
    - Así, en primer lugar se realizará el análisis y diseño de la tarea, de donde se

[1] Según el manifiesto ágil [B+01]:

> Se entregará software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

[2] En este caso, se asumen que el rol de cliente se asociará a los tutores del presente proyecto.

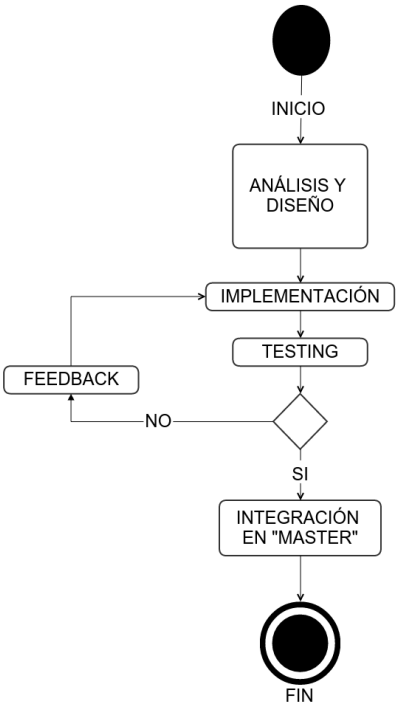

Figura 4.1: Diagrama de estados del flujo de una tarea.

obtendrá una descripción a nivel técnico de la misma. En esta descripción se incluirán, por ejemplo, qué funciones se deben realizar o qué dependencias entre tareas deben tenerse en cuenta para que el proyecto siga siendo funcional.

- A continuación se pasará a la fase de implementación. Ésta se realizará en un entorno paralelo al entorno donde se encuentra el proyecto estable, para evitar que fallos en el desarrollo impidan que el proyecto siga siendo funcional.
- Una vez se ha implementado la tarea, se probará (fase de testing), teniendo en cuenta que cumple con los requisitos marcado en la fase de análisis y diseño.
- Si la tarea no es correcta (no cumple con los requisitos o no es totalmente funcional), pasará al estado de feedback donde se corregirán los fallos reportado en la fase de testing. Cuando la tarea se ha corregido, pasa de nuevo al estado de testing.
- Si como resultado del testing la tarea es correcta, está podrá incluirse en el entorno «master», donde se encuentra el proyecto totalmente funcional y sin fallos.

Con este flujo de estados para cada tarea, conseguiremos que al final de cada sprint obtengamos un proyecto funcional y revisado.

4.2 Fases de Trabajo

En esta sección se definen las fases en las que se ha dividido el trabajo necesario para desarrollar el proyecto. El método de trabajo expuesto en la Sección anterior se sigue a partir de la *Fase III* puesto que las fases previas tienen que ver con el despliegue inicial de la aplicación y la definición de la Estructura de Base de Datos, que no pueden entenderse como fases del desarrollo de la aplicación propiamente dicha. Por otro lado, en esta sección no se pretende explicar cuál ha sido el flujo de trabajo en relación a la metodología escogida, si no que se pretenden exponer las principales dificultades y soluciones adoptadas durante cada una de las fases para conseguir los objetivos marcados.

4.2.1 Fase 0: Evaluación de herramientas

Antes de empezar con el desarrollo del proyecto propiamente dicho en lo que se refiere a implementación de componentes, fue necesario un periodo preliminar en el que evaluar las herramientas que se iban a utilizar, además de un periodo previo de aprendizaje tanto del uso de las mismas como de los lenguajes asociados a ellas. Aunque en el Capítulo 3, ya se analiza y justifica el uso de estas herramientas, es importante resaltar esta fase inicial de investigación en la que se evaluaron las distintas posibilidades disponibles para realizar el proyecto y cumplir con los objetivos marcados. A partir de esta fase, se deciden usar:

- DJANGO como framework de desarrollo de la aplicación Web. Es un entorno de desarrollo en Python que ofrece diferentes utilidades, como la gestión de URLs y la modularización de la plataforma Web en diferentes aplicaciones fácilmente reutilizables, que facilitan las labores de gestión y mantenimiento de la web, permitiendo centrar la atención en el desarrollo de la aplicación Web propiamente dicha.
- NEO4J como base de datos encargada de almacenar toda la información extraída de la red social. Neo4J es una Base de Datos Orientada a Grafos, con la que podemos hacer frente al hecho de tener una información donde una parte fundamental de la misma son las relaciones existentes entre sus elementos. Además, es una base de datos *libre de esquema*, lo que permite que la base de datos vaya creciendo y cambiando sin necesidad de modificar lo que, en términos de base de datos relacional, se conoce como esquema de base datos.
- Twitter como red social objetivo de estudio. Twitter es una de las redes sociales más utilizadas actualmente, por lo que supondrá un buen medio de obtención de información de la actividad de la ciudad. Por otro lado, su elección recae también en el hecho de presentar una API sencilla y que se adapta a los objetivos que quieren conseguirse con este proyecto.

Además de la elección de las herramientas principales, esta fase inicial comprende también la elección y aprendizaje, en su caso, de los lenguajes de programación que se utilizarán durante todo el desarrollo del proyecto. Se escogen en este caso: *Python*, como lenguaje para

el soporte de Django y la interacción con Twitter desde su API; *HTML5*, *JavaScript*,PHP y JSON, para dar soporte a la creación del entorno Web, y por último *Cypher*, como lenguaje para el acceso a la base de datos que nos proporciona Neo4J.

4.2.2 Fase I: Despliegue de la infraestructura

Una vez hemos descargado e instalado todos los elementos necesarios para poder comenzar con la realización de nuestro proyecto, según lo que se describe en el Anexo A, el primer paso es configurar el servidor Web Django, para establecer la estructura preliminar de nuestro proyecto. Antes de nada, es conveniente conocer cómo funciona Django y cómo estructura los diferentes contenidos de una aplicación Web tal y como la conocemos. La forma que tiene Django de estructurar las diferentes capas que componen un proyecto Web, es una variación del MVC (Modelo Vista Controlador) convencional. En este paradigma de desarrollo, el modelo es la aplicación propiamente dicha, la vista es la representación de la aplicación de cara al usuario, es decir, la interfaz, y el controlador se encarga de definir el comportamiento de la aplicación según la entrada producida por el usuario a través de la interfaz [GHJV94]. Sin embargo, el patrón seguido por Django es lo que se conoce como MVT, o Modelo Vista Plantilla. Este patrón tiene muchas similitudes con el Model View Controller (MVC) convencional, [CA12] de hecho:

- El *modelo* en Django sigue siendo el *modelo* en MVC.
- Lo que se denomina *plantilla*, es la *vista* en MVC.
- La **vista** es lo que se denomina *controlador* en MVC.

La Figura 4.2 muestra la interacción de los componentes.

1. El navegador lanza una petición
2. La vista interactúa con el modelo para obtener los datos
3. La vista realiza una llamada a la plantilla.
4. Por último, la plantilla formatea la respuesta para mostrar el resultado de la petición del navegador.

Por tanto, el *modelo* se encarga de definir qué datos se encuentran en Base de Datos en forma de clases de Python. También se encarga de definir los atributos de cada uno de los elementos y las funciones que permiten actualizar los mismos. La *vista* se define mediante funciones en Python cuyo objetivo es determinar qué datos serán visualizados, y otras funciones adicionales como el envío de correos electrónicos, autenticación de servicios externos y validación a través de formularios. Por último, la vista es la propia página HTML con etiquetas que permiten insertar código procedente de Django, además de todos los archivos encargados de crear el contenido de la interfaz: archivos XML, CSS, *JavaScript*....

Después de conocer cómo es la estructura interna de Django, el siguiente paso es definir nuestro proyecto. Para ello, ejecutamos en la terminal:

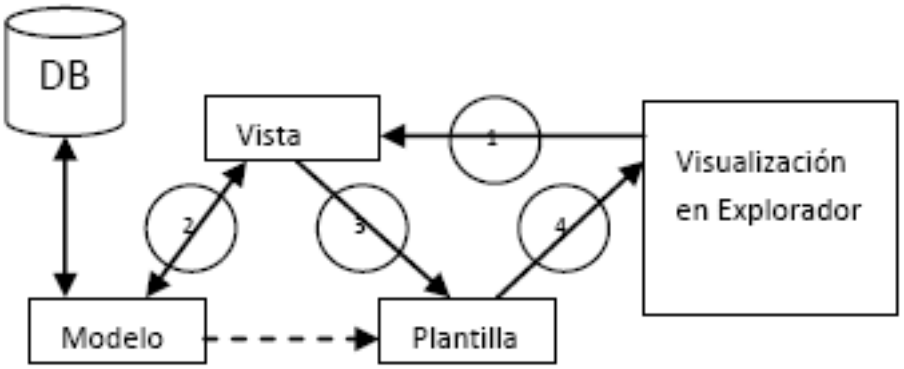

Figura 4.2: Patrón MVT usado en Django. *Fuente: Phython - Django Framework de desarrollo web para perfeccionistas basado en el Modelo MTV*

```
\$ django-admin startproject mysite
```

Listado 4.1: Creación de un proyecto con Django

Donde `mysite` es el nombre de nuestro proyecto. El resultado de ejecutar este comando, es la siguiente estructura de directorio:

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Listado 4.2: Estructura de directorios del proyecto en Django

Estos archivos son:

- El directorio externo `mysite` sólo es el contenedor del proyecto. Su nombre no es representativo.
- `manage.py` es un script de línea de comandos que permite realizar diferentes operaciones sobre el proyectos.
- El directorio `/mysite` interno es el directorio real del proyecto en Python.
- `mysite/settings.py` contiene todas las configuraciones necesarias a realizar sobre el proyecto.
- `mysite/urls.py` se utiliza para declarar todas las URLs accesibles desde el proyecto.
- `mysite/wsgi.py` sirve para ofrecer compatibilidad del proyecto con servidores Web WSGI.

El siguiente paso, es establecer la configuración de la Base de Datos en el archivo `settings.py`. En este archivo, encontramos una parte dedicada a la configuración de las Bases

de Datos a utilizar que podemos ver en el Listado 4.3. Por defecto, Django ofrece una base de datos SQLite3, pero esta configuración puede modificarse para trabajar con *MySQL*, *Oracle* o *PosrgreSQL*. El principal problema que nos encontramos en este punto es establecer una forma de conectar nuestro proyecto de Django con Neo4J, ya que no se ofrece soporte de forma oficial.

```
DATABASES = {

    'default': {
        'ENGINE': 'django.db.backends.sqlite3', # Add '
           postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'NAME':  os.path.join(BASE_DIR, 'db.sqlite3'),
                                 # Or path to database file if
           using sqlite3.
        'USER': '',                      # Not used with sqlite3.
        'PASSWORD': '',                  # Not used with sqlite3.
        'HOST': '',                      # Set to empty string for
           localhost. Not used with sqlite3.
        'PORT': '',                      # Set to empty string for
           default. Not used with sqlite3.
    }

}
```

Listado 4.3: Configuración de las bases de datos en settings.py

Por tanto, es necesario establecer un mecanismo por el cual poder acceder la Base de Datos a través de Python, lenguaje utilizado en Django. Actualmente existen varias opciones con las que poder acceder a Neo4J a través de múltiples lenguajes de programación. En la página oficial de Neo4J se ofrecen multitud de posibilidades para Java, PHP, JavaScript, Ruby o Python, entre otros. Si nos centramos en las posibilidades que se ofrecen para Python, tenemos varios adaptadores tales como *py2neo*, *neo4jRestClient*, *bulbflow*, *neo4django*, *neoModel*, entre otros. La principal restricción a la hora de elegir un adaptador u otro es que sea compatible con Django. Ante esta situación, nos quedan dos opciones para elegir: *Neo4Django* y *NeoModel*. La elección de uno u otro ahora tiene que ver con cuestiones tales como: facilidad de integración, flexibilidad (que no sea sólo para Django si no que pueda usarse también en Python en general), y compatibilidad con las versiones tanto de Neo4J como de Python y Django utilizadas[3]. Con estas nuevas restricciones, nos decantamos por la elección de *NeoModel* como adaptador en Python para el acceso a Neo4J. La elección de este y no de *neo4django* está justificada por varios motivos:

Una vez tenemos definido la forma de acceder a Neo4J a través de Django, y de forma general en Python, podemos continuar con la configuración del servidor. Hasta ahora, está definido lo que en Django se conoce como *proyecto*, pero para desarrollar una aplicación Web tal y como la conocemos, lo siguiente que debemos hacer es crear una (o varias) *apli-*

[3]El Anexo A es una guía de instalación de todos los componentes utilizados en el desarrollo del proyecto. En él, están recogidas las versiones utilizadas de cada elemento.

Cuadro 4.1: Comparativa entre Neo4Django y NeoModel.

	Neo4Django	**NeoModel**
Compatibilidad de Versiones	La versión de Django soportada debe ser mayor o igual a la 1.3 y menor a la 1.6.Esta restricción nos obliga a utilizar un versión muy antigua de Django, que utiliza una sintaxis para sus operaciones muy distinta a la sintaxis de las nuevas versiones.	No impone ninguna restricción acerca de la versión de Django a utilizar,tan sólo que la versión de Neo4J sea la 2.0, 2.1 o 2.2.
Flexibilidad	Esta creado para trabajar únicamente con Django, es decir, no puede usarse con Python por separado.	Permite el trabajo tanto con Django como únicamente con Python.
Facilidad de Integración	Su integración es compleja. Para poder utilizar *Neo4Django* es necesario establecer múltiples parámetros de configuración.	No necesita ningún tipo de configuración para utilizarse de forma conjunta con Django.

caciones dentro de nuestro proyecto. El concepto de aplicación en Django se refiere a una herramienta que se encarga de una función concreta dentro del proyecto. Un proyecto, puede contener varias aplicaciones, y del mismo modo, una aplicación puede estar en varios proyectos. Esta característica ofrece gran modularidad en las aplicaciones Web, que pueden dividirse en varias *aplicaciones de Django* para dividir de forma más clara las partes de la aplicación, además de permitir la reutilización de código de una forma sencilla y eficiente.

Así pues, en nuestro caso, crearemos una aplicación, desde el directorio raíz de nuestro proyecto, de la siguiente manera:

```
\$ python manage.py startapp smart_trends
```

Listado 4.4: Creación de una aplicación en Django

En este caso, hacemos uso del script `manage.py` para crear nuestra aplicación,llamada *smart_trends*. Como resultado, se crea una aplicación dentro de nuestro proyecto, con la siguiente estructura:

```
smart_trends/
    __init__.py
    admin.py
    migrations/
        __init__.py
    models.py
    tests.py
```

```
    views.py
```

Listado 4.5: Estructura de directorios de la aplicación.

En este caso, los archivos más importantes son `models.py` y `views.py`. El primero de ellos, se emplea para definir los elementos que contendrá la Base de Datos, y las funciones para acceder y realizar diferentes modificaciones sobre estos elementos. El archivo `views.py`, define las distintas vistas de la aplicación. Las vistas no son más que funciones Python que se corresponden con una petición a una URL concreta dentro del navegador. Así, cuando desde el navegador se realiza una petición:

1. Se comprueba que la URL solicitada es una URL definida para nuestro proyecto, es decir, debe estar definida en el archivo `mysite/urls.py` Además para que Django identifique este archivo como el archivo donde se encuentran todas las direcciones, es necesario indicar, en el archivo `settings.py`.
2. Si la URL coincide con algún patrón de los definidos, se llama a la vista de la aplicación que se corresponda con la URL, es decir, se llama a una de las funciones definidas en `smart_trends/views.py`
3. Esta función, se encarga de realizar las consultas necesarias a Base de Datos para devolver al navegador la petición solicitada.
4. Cuando la vista ha terminado de recoger los datos, se los envía a la plantilla (código HTML) que representará los datos en el navegador.

Además de definir nuestra aplicación, debemos hacerla «visible» para nuestro proyecto, de lo contrario, no podremos acceder a ella. Para ello, debemos incluir la aplicación dentro de la variable `INSTALLED_APPS`,definida en el archivo `settings.py`, tal y como se indica en el Listado 4.6. Esta variable contiene todas las aplicaciones que se activan cuando se arranca el proyecto. Muchas de las aplicaciones que se arrancan son utilizadas por Django para llegar acabo toda la gestión de la aplicación Web.

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    'django.contrib.admindocs',
    'smart_trends',#app principal
)
```

Listado 4.6: Activación de la aplicación en el proyecto principal

Por último, tenemos que configurar nuestro proyecto para poder, posteriormente, definir las plantillas que permitirán visualizar toda la información. Para ello, en primer lugar, crearemos los directorios `templates`, que contendrán las plantillas de nuestro proyecto, y `static`, que contendrá lo que en Django se define como archivos *estáticos*, que no son más que el resto de archivos –*CSS, JavaScript*, imágenes... – que permiten la visualización de la plantilla. Finalmente, debemos indicar a Django la ruta de estos dos directorios, tal y como podemos ver en el Listado 4.7

```
STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(BASE_DIR,'static'),
)

TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates'),
)
```

Listado 4.7: Configuración de los directorios de las Plantillas en settings.py

4.2.3 Fase II: Definición del modelo de Base de Datos

Como ya se ha indicado, Neo4J es una Base de Datos libre de esquema, lo que quiere decir que pueden añadirse elementos en la base de datos que no sigan ninguna estructura predefinida. Aún así, es necesario establecer una estructura inicial, no a nivel de configuración de la Base de Datos, ya que no requiere ninguna configuración, pero si a nivel conceptual, para conocer qué entidades, atributos y relaciones son necesarias para conseguir los objetivos del proyecto. Se pretende que la estructura de las relaciones entre elementos existentes en Twitter, queden reflejadas lo más fielmente posible en la Base de Datos, para que la representación de los mismos se correspondan con la realidad de la red social. Por otro lado, es importante capturar sólo aquellos atributos y relaciones que sirvan para conocer qué es lo que está sucediendo en la ciudad, descartando aquellos atributos innecesarios, como por ejemplo, datos personales de los usuarios más allá de su localización, o relaciones entre usuarios que nos permitan conocer la relación de amistad existente entre los usuarios.

En nuestro caso, nos interesa conocer:

- Con respecto a las publicaciones (*tweets*): Contenido de la publicación, etiquetas que se emplean en la publicación, usuario que escribe la publicación, usuarios *mencionados* en el tweet y su fecha de creación.
- Con respecto a los usuarios: Nos basta con almacenar su nombre en la red social – nickname – y su localización.
- Con respecto a las etiquetas o *hashtags*: Sólo es necesario almacenar la etiqueta en sí (el texto).

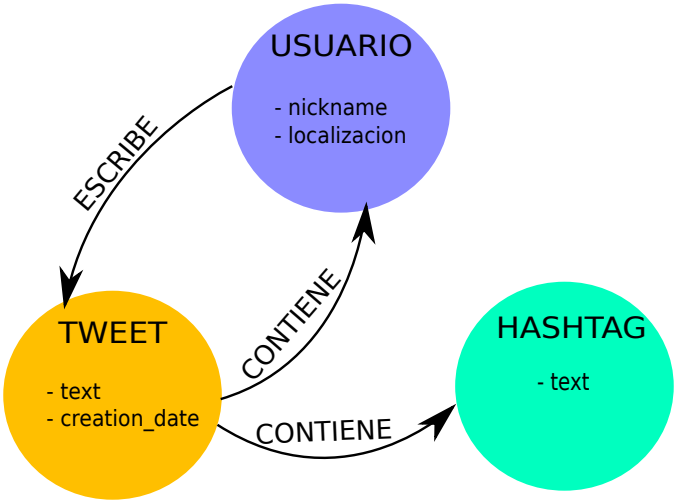

Figura 4.3: Estructura inicial de Base de Datos.

Los elementos anteriores serán representados como nodos en Neo4J. Así, para mantener la relación existente entre cada uno de los nodos, se definen las siguiente relaciones, que serán las aristas en la Base de Datos.

- Relación `ESCRIBE`: Sirve para indicar qué usuario escribe qué tweet. Así tenemos que «*un usuario ESCRIBE un tweet*».
- Relación `CONTIENE`: Se utiliza para conocer, a parte del texto, qué otros contenidos tiene un tweet. En Twitter, un tweet puede contener multitud de elementos: imágenes, vídeos, enlaces a sitios externos, *menciones a usuarios* y *hashtags*. En nuestro caso, sólo nos interesa establecer la relación existente entre la publicación en cuestión y los usuarios y etiquetas que aparecen en él.

Así la estructura inicial de la Base de Datos, sería como se muestra en la Figura 4.2.3

Como podemos observar, en la Base de Datos no se van a almacenar relaciones del tipo «*El usuario A sigue al usuario B*», puesto que el objetivo no es capturar las posibles relaciones de amistad existentes entre los usuarios de la red, si no que lo que se pretende es conseguir conocer el movimiento de la red social en cada momento: quién escribe qué tweet, qué personas aparecen en los tweets o qué etiquetas son las más utilizadas en cada momento.

Una vez hemos establecido el modelo inicial de Base de Datos, tenemos que representarlo en Django. Como sabemos, según la estructura de los proyectos en Django, los elementos de Base de datos se almacenan como objetos en el archivo `models.py`. En nuestro caso, utilizaremos *NeoModel* para adaptar la estructura que emplea Django para conocer los elementos de Base de Datos con la estructura que emplea Neo4J. El Listado de código **??** muestra el contenido del archivo `models.py` del proyecto de Django. En él se representa la estructura

de Base de Datos equivalente a la mostrada en la Figura 4.2.3. Tal y como podemos observar, lo único necesario para poder utilizar la Base de Datos Neo4J con Django, es importar las clases de la librería de *NeoModel* que nos permitan crear los nodos y las relaciones entre los mismos, y establecer el tipo de dato que se corresponde con cata atributo de los nodos. Aunque en Neo4J no es necesario indicar nada acerca de la estructura de los elementos almacenados en Base de Datos, el hecho de trabajar con esta a través de un lenguaje de programación, nos obliga a indicar ciertas restricciones:

- Definición del tipo de dato de los atributos, necesario para poder realizar, en el caso de que fuera necesario operaciones cuya implementación dependa del tipo de dato que se esté utilizando. Por ejemplo, las operaciones de comparación no son iguales si tratamos con enteros o con cadenas, o para añadir un entero a una cadena es necesario hacer un *casting* a *String*.
- Obligatoriedad o no de ciertos elementos a la hora de crear un nodo. Aunque para Neo4J no supone ningún error que se cree un elemento con ciertos campos vacíos, sí existen ciertas restricciones en este sentido de cara a conseguir el objetivo del proyecto, y que además sirven para evitar errores a la hora de almacenar o acceder a datos erróneos en Base de Datos. Así, cuando, a través de NeoModel se cree un nodo de cualquier tipo, deberá contener, como mínimo, los campos que tienen `'required = True'` en su definición.

```
from django.db import models

from neomodel import (StructuredNode, StringProperty, IntegerProperty,
    RelationshipTo, RelationshipFrom)

class User(StructuredNode):

    #PROPERTIES
    nick_name = StringProperty(required = True)
    location = StringProperty()

    #RELATIONS
    writes = RelationshipTo('Tweet', 'WRITES')

class Tweet(StructuredNode):
    text = StringProperty(required = True)
    created_at = IntegerProperty(required = True)

    contains_tag = RelationshipTo('Hashtag','CONTAINS')
    contains_user = RelationshipTo(User, 'CONTAINS')

class Hashtag(StructuredNode):
    text = StringProperty(required = True)
```

Listado 4.8: Estructura de Base de Datos en models.py

Con esta estructura inicial de Base de Datos quedan definidas tanto las relaciones como

los atributos mínimos para conseguir el objetivo del proyecto.

4.2.4 Fase III: Desarrollo de los recolectores de datos

Una vez definida la estructura de Base de Datos, el siguiente paso es recopilar la información de Twitter a través de su API. En la página oficial para desarrolladores de aplicaciones de Twitter, encontramos soporte para trabajar con la API de Twitter a través de muchos lenguajes de programación, como por ejemplo, PHP, JavaScript, Java, .Net, Python, entre muchos otros. En nuestro caso, se opta por elegir *Python* como lenguaje para el desarrollo de los recolectores, principalmente por dos motivos:

- La elección del lenguaje en este caso, no es determinante, puesto que los recolectores no están directamente relacionados con la aplicación Web, por lo que se opta por elegir un lenguaje del que se conoce su sintaxis, semántica y funcionamiento.
- Los recolectores también necesitan interaccionar con Neo4J para almacenar todos los datos extraídos de la red social, y puesto que ya tenemos *NeoModel* como adaptador para conectar la Base de Datos con cualquier programa escrito en Python, se elimina la necesidad de tener que buscar otra alternativa para establecer la conexión con Base de Datos.

Una vez elegido el lenguaje con el que trabajar, y la API de desarrollo, Twitter requiere que todas las aplicaciones que utilicen una API para acceder a los datos de la red social, estén registradas en la *plataforma de desarrolladores de Twitter*. Cuando registramos la aplicación, obtenemos unas claves de acceso y autenticación que son necesarias para conectar con Twitter a través de la API que usemos en nuestra aplicación.

Con todos estos elementos, podemos empezar a construir lo que hemos denominado *recolector*. Un recolector será un script en Python que acceda a las publicaciones que se den en torno a *Ciudad Real* con el fin de conocer la actividad de la ciudad en cada momento. A la hora de definir la estructura del recolector surgen varias cuestiones: *¿Cómo se acotan las publicaciones de Ciudad Real?*, *¿cómo sé que una publicación está hablando algún tema relacionado con la ciudad?*, *¿qué usuarios me pueden aportar información «fiable» a cerca de lo que está ocurriendo?*. Para resolver estas cuestiones, y definir uno o varios recolectores que sean capaces de extraer información correcta y coherente con lo que se está buscando, es necesario observar primero el comportamiento de la red social. En términos generales, el comportamiento de la red social puede explicarse de la siguiente manera: Normalmente, cuando no hay ningún evento de interés para la comunidad, el volumen de publicaciones es mucho menor al que puede llegarse a dar cuando ocurre un evento importante si no para toda la comunidad, para un gran porcentaje de ella, por ejemplo, una final de fútbol, o unas elecciones. Además, cuando no está ocurriendo nada «importante», cada usuario escribe acerca de temas que no están relacionados entre sí, suelen escribir a cerca de lo que están haciendo, o comparten publicaciones de otros usuarios. Sin embargo, cuando se está dando un evento,

a parte del aumento de tweets, la mayoría de ellos tratan sobre el mismo tema, es decir, en la mayoría de las publicaciones hay uno o más *hashtags* que se repiten, y que describen el tema del que se está hablando. Por otro lado, cuando una noticia o evento tiene repercusión sobre la sociedad, muchos de los usuarios tienden a *retuitear*[4] publicaciones de usuarios como periódicos, periodistas o informativos de televisión. Además, se observa una particularidad con respecto a los *tweets geolocalizados*. Según el estudio «*Mapping the global Twitter heartbeat: The geography of Twitter*», publicado por la revista on-line *First Monday*, tan sólo un 2,2 por ciento del total de tweets tiene información geográfica, un 1,8 por ciento indica el lugar donde se encuentra – que puede ser un área muy extensa –, un 1,6 por ciento incluye la ubicación extacta – definida por sus coordenadas –, y un 1,4 por ciento tiene ambas ubicaciones [LWC+13]. A pesar de que el porcentaje de tweets geolocalizados es muy bajo, se ha observado que normalmente, un usuario incluye información de su ubicación cuando se encuentra en una zona donde no es habitual que esté, por ejemplo, está de viaje o asistiendo a un concierto, feria u otro evento.

Con toda esta información obtenida a partir de la observación del comportamiento de los usuarios de Twitter, se establecen los siguientes criterios de búsqueda para extraer información de la red social:

- Es necesario buscar tweets que contengan «*Ciudad Real*» en el texto de la publicación. Con esto, se consigue obtener publicaciones de usuarios que están escribiendo a cerca de la ciudad.
- También es necesario obtener también aquellas publicaciones geolocalizadas en Ciudad Real. Como ya se ha dicho, aunque un porcentaje muy pequeño del total de los tweets se corresponde con publicaciones con información de la ubicación, es importante capturarlos ya que, ante un evento en una zona concreta de la ciudad, este tipo de publicaciones pueden ayudarnos a conocer dónde está ocurriendo dicho evento.
- Por último, es importante conocer qué están escribiendo los usuarios de Ciudad Real. Para encontrar estos usuarios debemos fijarnos en la localización que el usuario indica en su perfil de Twitter. Es cierto que basándonos sólo en este dato puede ocurrir que haya usuarios de Ciudad Real que no tengan definida su localización o que tengan otra que no se corresponde con su situación habitual, o incluso que haya usuarios que actualmente no viven en Ciudad Real pero que tienen esta localización definida. Ante este problema, la solución propuesta es comenzar «enseñando» al sistema qué perfiles se corresponden con personas o entidades de Ciudad Real. Para ello, es necesario comenzar buscando, de forma manual, usuarios o entidades de las que se tenga la certeza de que son de Ciudad Real. Así se buscan entidades como periódicos o revistas de la ciudad, u organismos que representen a la misma, que se almacenarán inicialmente en Base de Datos.

[4]Se refiere a que un usuario comparte con sus seguidores un tweet publicado por otro usuario.

A partir de estos criterios es necesario definir por una parte, *tres recolectores* de publicaciones que tendrán una estructura similar – uno para cada criterio de búsqueda de tweets –, y un *buscador* de usuarios de Ciudad Real. La estructura de los tres recolectores es la misma, la única diferencia existente entre ellos está en los recursos que se solicitan a la red social a través de la API. Así, cada uno de los recolectores tendrá la siguiente estructura: En primer lugar, es necesario autenticar la aplicación con las claves obtenidas al registrar el proyecto en Twitter[5].A continuación, y dentro de un bucle, se realiza una petición a Twitter con el criterio de búsqueda correspondiente para obtener unas u otras publicaciones. En los Listados 4.9 y 4.10 se recogen las peticiones que hay que realizar para obtener las publicaciones que contienen «*Ciudad Real*» y aquellas que están geolocalizadas en la ciudad. En ambos, el atributo `since_id` indica el identificador del tweet a partir del cual se comienza a recoger información. Inicialmente, este valor es vacío y después se va actualizando con la última publicación recogida en la petición anterior. El campo `count` indica el número de peticiones máximo que se recogen en cada petición. Se establece este valor, puesto que la API de Twitter establece unos valores máximos de publicaciones que pueden obtenerse con cada petición. En el Listado 4.9, el campo `term` indica el término de búsqueda por el que filtrar todas las publicaciones, análogamente en el Listado 4.10, el campo `geocode`, filtra por aquellos tweets que estén geolocalizados, y cuya ubicación se encuentre dentro del rango definido.

```
timeline = api.GetSearch(term= "ciudad real",since_id = lastid, count
    = 50)
```

Listado 4.9: Petición para obtener los tweets que contengan «Ciudad Real»

```
timeline = api.GetSearch(geocode="38.59,-3.56,5km",since_id = lastid,
    count = 50)
```

Listado 4.10: Petición para obtener los tweets geolocalizados en Ciudad Real

En el caso del recolector que busca las publicaciones de los usuarios de Ciudad Real, la parte inicial es algo diferente. En primer lugar, se debe realizar una consulta a Neo4J –tal y como se puede ver en el Listado 4.11 – para los usuarios cuya localización sea Ciudad Real. Una vez hemos recuperado los usuarios objetivo, para cada uno de ellos, se hace una petición a Twitter para recuperar los tweets que ha publicado el usuario desde su última publicación. Como podemos ver en el Listado 4.12, la petición filtra los tweets tanto por el identificador del usuario como el de la publicación.

```
MATCH (u:User) WHERE u.location =~ '(?i).*Ciudad Real.*'RETURN u
```

[5]Es necesario indicar que la información recogida está permitida y contemplada en la Política de Privacidad de Twitter [Twi15]. En ella se indica que

> Su información pública de perfil de usuario y Tweets públicos se entregan inmediatamente a través de SMS y nuestras API a nuestros socios y terceros, incluyendo motores de búsqueda, desarrolladores y editores que integran el contenido de Twitter en sus servicios, e instituciones como universidades y agencias públicas de la salud que analizan la información para observar tendencias e ideas.

Por lo tanto, toda la información que se obtiene gracias a las peticiones desde la API se corresponden con información de usuarios con perfil público.

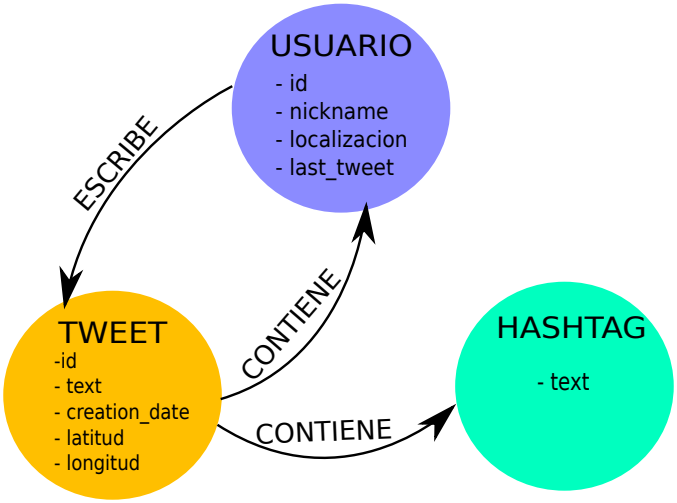

Figura 4.4: Estructura definitiva de Base de Datos.

Listado 4.11: Consulta de Base de Datos para obtener los usuarios de Ciudad Real

```
timeline = api.GetUserTimeline(u.id, since_id = u.last_tweet)
```

Listado 4.12: Petición para obtener las últimas publicaciones de un usuario

Con estos criterios de búsqueda, observamos que la estructura inicial de Base de Datos no está del todo completa para conseguir almacenar toda la información necesaria. En primer lugar, debemos añadir dos campos más en la estructura de los tweets para poder almacenar las coordenadas (latitud y longitud) de los tweets que estén geolocalizados. Además, es necesario modificar la estructura del usuario para poder conocer cuál ha sido el último tweet que publicó y poder así recuperar siempre sus últimas publicaciones. Por último, tanto el caso de los usuarios como en el caso de los tweets es necesario almacenar el identificador global usado por Twitter para poder acceder y actualizar la información de estos registros si fuese necesario. Con estas modificaciones, obtenemos la estructura final de Base de Datos, con los elementos que se muestran en la Figura 4.2.4.

Después de esta parte inicial, en la que se recuperan las publicaciones según el criterio de filtrado, todos los recolectores siguen la misma estructura, para cada una de las publicaciones recogidas:

1. Si la publicación no existe en Base de Datos – puede que otro de los recolectores la haya recogido porque coincida con varios criterios de búsqueda – se guarda el tweet con los atributos `id`, que es identificador global dentro de Twitter, `text`, que el el texto de la publicación, `created_at`, que es la fecha de publicación del tweet y que se alma-

cena en Base de Datos como *timestamp* y por último, para los tweets geolocalizados, se almacenan también los campos `lat` y `lon` que se corresponden con la latitud y la longitud del tweet, respectivamente. Si la publicación existe, no se hace ningún paso más, puesto que otro recolector ya habrá almacenado toda la información necesaria.

2. A continuación, se obtiene el usuario autor de la publicación. Si el usuario no existe, se almacena en Base de datos con los campos `id`, que se corresponde con el identificador en la red social, `nickname` que es el nombre con el que el usuario se da a conocer en la Twitter, `location`, que es la localización que ha indicado el usuario en su perfil y `last_tweet` que es el identificador del último tweet que publicó el usuario. Si por el contrario, el usuario ya existe en Base de Datos, se actualiza el valor del campo `last_tweet` por el identificador del tweet actual. Por último, y en cualquier caso, se crea la relación `ESCRIBE`, para relacionar al usuario con el tweet que ha publicado.
3. El siguiente paso es extraer los *hashtags* de la publicación. Para cada uno de ellos, se comprueba primero si existe la etiqueta en Neo4J, si no existe se crea con el campo `text`, que es el propio texto de la etiqueta. Finalmente, tanto si la etiqueta ya existe como si se acaba de crear, se define la relación `CONTIENE` para indicar que ese hashtag aparece en ese tweet.
4. El último paso consiste en extraer los usuarios que aparecen *mencionados* en el tweet. Para cada uno de ellos, si el usuario no existe, se crea uno nuevo en Base de Datos con los mismos campos con los que se crea al usuario autor de la publicación. Por último, se crea la relación `CONTIENE` para indicar que ese usuario aparece mencionado en la publicación.

Una vez finalizado este proceso para todas las publicaciones que se obtuvieron en la petición a Twitter, se debe esperar un tiempo mínimo para volver a «preguntar» por el mismo recurso. Twitter establece un número máximo de peticiones consecutivas a partir de las cuales, bloquea el recurso solicitado un determinado tiempo. En nuestro caso, el criterio que se sigue, es esperar 60 segundos para volver a solicitar el recurso, siempre y cuando el tiempo que va a permanecer el recurso bloqueado sea menor.

A parte de definir los recolectores, se define también lo que se ha denominado *buscador*, cuyo objetivo es aumentar el número de usuarios de Ciudad Real que están almacenados en Base de Datos y así conseguir abarcar un rango más amplio, y por lo tanto, más fiable, de la actividad en la ciudad. La estructura del buscador es sencilla. En primer lugar, hace una consulta a Neo4J para recuperar los usuarios de Ciudad Real que hay hasta ahora. A continuación, para cada uno de los usuarios, se obtienen sus *seguidores* [6] y para cada seguidor, se almacenan en Base de Datos aquellos que sean de Ciudad Real, y no hayan sido almacenados

[6] Se asume que si un usuario es de Ciudad Real, muchos de sus seguidores lo serán también, más aún si el perfil se corresponde con una entidad pública, donde la mayoría de los seguidores querrán obtener información de Ciudad Real.

previamente.

Con estos dos elementos, recolectores y buscador, se obtiene información representativa sobre la actividad en Twitter producida por usuarios que están en el entorno de Ciudad Real.

4.2.5 Fase IV: Análisis de Normalidad

Tal y como se indica en el Capítulo 2, el sistema que ese está construyendo debe tener «*capacidad para contabilizar el volumen de información entrante. Esto permitirá conocer cuándo la cantidad de información que está recogiendo el sistema es notablemente superior a lo que el sistema conoce como un volumen normal*». Así pues, en esta fase de trabajo, se definen los criterios con los que se ha desarrollado el *análisis de normalidad* de la actividad producida en la red social.

El primer paso, consiste en establecer qué técnica se seguirá para realizar el análisis de normalidad. En este caso, se opta por un análisis sencillo basado en la *media del volumen de publicaciones*. Existen muchas otras técnicas que sirven para cumplir este objetivo, basadas en el uso de la normal en lugar de la media, y que además son capaces de cambiar su percepción de la normalidad con el paso del tiempo [DF14b]. Sin embargo, la complejidad de estas técnicas, y el tiempo de análisis inicial que requieren, hacen que sean rechazas, ya que sobrepasan el alcance del proyecto. Una vez hemos decidido que nuestro análisis de normalidad tomará como valor el volumen de tweets producidos en la red social, es necesario establecer ciertos criterios que permitan que la técnica sea lo más fiable posible. Tal y como se indica en la Sección 3.4.2, el primer paso para comenzar un análisis de normalidad es indicar al sistema qué es normal, y esto sólo puede conseguirse estableciendo criterios de manera manual. En primer lugar, el sistema no sabe a priori que ni en todas la horas del día existe el mismo volumen de tweets – el volumen es superior en las horas centrales del día –, ni que el volumen de publicaciones varía según el día de la semana – suele ser superior los fines de semana –. Así el análisis de normalidad contabilizará, en una primera fase el volumen de publicaciones que se producen en cada hora del día, distinguiendo además en qué día de la semana se está realizando el conteo.

Con estos criterios, el sistema debe empezar a observar la actividad de la red social, durante un tiempo en el que, nuevamente decidido bajo el criterio humano, se espera una actividad «normal». Con esto se pone funcionamiento el *analizador*, que trabaja de la siguiente manera: El analizador lanza una consulta a Neo4J para obtener el número de publicaciones de la última hora, siempre realizando la cuenta a partir de la hora exacta, es decir, en intervalos desde, por ejemplo las 10:00 a las 11:00. Esta cuenta, la almacenará en la tabla `medias` en la Base de Datos *SQLite3*, que tiene los campos `dia`, que indica el día de la semana de la media que se está almacenando; `hora`, que indica la hora del día con la que se corresponde la media, y `media`, que contiene el valor medio normal de publicaciones para ese día de la

semana y esa hora concreta. En las primeras observaciones, donde todavía no tenemos valor medio, se almacena la cuenta de publicaciones, y después se va recalculando ese valor realizando la media de lo que ya hay almacenado para ese día y esa hora, con el nuevo valor obtenido.

Tras un periodo de observación de la red social de dos semanas, se modifica la estructura del analizador para que ahora también sea capaz de detectar si se está produciendo un comportamiento anómalo en la red social. Para ello, es necesario establecer un valor umbral para el cual se establezca que se está superando el volumen de tweets correspondiente a una actividad normal. En nuestro caso, se ha optado por establecer un valor umbral fijo. Al igual que para las técnicas de análisis de normalidad, a la hora de establecer este valor, existen muchas técnicas y de diversa complejidad, que se han descartado porque requieren periodos de observación muy amplios que hacen que su uso se escape del alcance del proyecto.

Así pues, se ha establecido un valor umbral tal que, si la cuenta de tweets para un día y una hora concreta excede en un *veinte por ciento* al valor medio de publicaciones para ese día y esa hora, se genera una alerta. Esta alerta se guarda en la Base de Datos SQLite3 con los campos `hora_inicio` y `hora_fin` como *timestamp*, para poder recuperar posteriormente de Neo4J las publicaciones que se corresponden con la alerta [7], y el campo `cuenta` que indica el total de publicaciones para ese intervalo de tiempo. Si por el contrario, el valor está dentro de lo que se entiende como normal, se ajusta de nuevo el valor de la media correspondiente, teniendo en cuenta, el valor ya almacenado, y el nuevo valor obtenido.

4.2.6 Fase V: Desarrollo de la vista principal

En las secciones anteriores, se define todo lo que podemos llamar la «parte interna» de la aplicación. En esta sección y en la siguiente, se define la parte externa de aplicación, es decir, la aplicación Web que está de cara al usuario y que sirve para mostrar todos los resultados obtenidos tanto de los *recolectores* como del *analizador*. En esta sección, nos centramos concretamente en cómo debe ser la *vista principal* de la aplicación Web, que es la encargada de mostrar toda la información recopilada por los recolectores sobre la actividad en Twitter del área de Ciudad Real.

El objetivo principal es conseguir que el usuario pueda, sin necesidad de mucho esfuerzo, comprender qué está sucediendo en la red social, y sea capaz de detectar cual es el tema principal del que se está hablando, en el caso de que esté ocurriendo un evento en la ciudad. Por otro lado, se busca que quede reflejado el hecho de estar utilizando una Base de Datos Orientada a Grafos, por lo que es importante encontrar una forma de representación de la información que nos permita mostrarla como un grafo.

[7] En la Sección 4.2.7 se explica la utilidad de almacenar esta información en base de datos, en lugar del día y la hora tal y como se guarda para la media.

Representación de la información con grafos : Una parte fundamental de la vista principal de la aplicación Web es la parte que le permitirá al usuario conocer los temas de mayor relevancia en Twitter para los usuarios del entorno de Ciudad Real en el momento en el que se hace la petición a la aplicación. Para ello, se decide representar la actividad de la red social de las últimas *dos horas*[8] en forma de grafo, manteniendo el esquema de relaciones almacenado en Base de Datos. El hecho de elegir las últimas dos horas para mostrar la actividad de Twitter está justificado especialmente por dos motivos: el primero es que se quiere conocer información actual, por lo que mostrar la actividad de momentos anteriores puede hacer que la información obtenida no sea representativa de la actividad real del entorno de Ciudad Real. Otro motivo importante reside en el tiempo de respuesta, ya que, cuanto mayor sea el intervalo de tiempo en el que se recoge la información, mayor será el tiempo de procesado de la petición por parte de Django y el tiempo de representación de dicha información por parte del navegador. Por otro lado, se busca también que este grafo permita conocer qué es lo más importante con solo echar un vistazo a la información que en él se recoge, sin que ello suponga un gran esfuerzo para el usuario. Por eso, es fundamental que el tamaño de los nodos que representan a los hashtags ,usuarios y tweets esté en relación a la importancia que tienen esos elementos para la actividad de la red social. Así:

- El tamaño de un nodo que representa un *hashtag* será mayor cuanto mayor sea el número de apariciones del mismo en las publicaciones del intervalo de tiempo mostrado en el grafo.
- El tamaño de un nodo que representa un *usuario* se establece atendiendo a dos criterios. El número de tweets que ha escrito ese usuario, y el número de veces que ese usuario aparece *mencionado* en un tweet.
- El tamaño de un nodo que representa un *tweet* se define en relación al número de elementos que contiene (*menciones* y *hashtags*). Se define este criterio para las publicaciones ya que, a priori, es un tweet que nos ofrece más información que un tweet que no contiene este tipo de elementos.

Una vez definidos los requisitos para la representación del grafo, el siguiente paso consiste en encontrar una librería o algún tipo de herramienta similar que nos permita representarlo atendiendo a estos criterios:

1. Debe permitir la distinción de los nodos mediante colores, para poder diferenciar hashtags, usuarios y tweets.
2. Debe permitir distinguir el tipo de relación existente entre los nodos mediante colores u otro elemento definitorio. En nuestro caso, un usuario puede relacionarse con un tweet bien porque ese usuario es el autor de éste, o porque ha aparecido mencionado en él.

[8]Obviamente, este intervalo de tiempo es el mismo que se toma para la obtención del resto de información que se muestra en esta vista principal.

3. Debe permitir identificar cada nodo a través de un nombre representativo, por ejemplo, a los usuarios por su *nick* o a las etiquetas por su texto.
4. Debe permitir definir diferentes tamaños para los nodos en relación a los criterios de importancia definidos.

Además de estas restricciones que tienen que ver con la forma de visualización de la información, es necesario también, que la librería tome la información para representar el grafo en un formato que pueda obtenerse fácilmente a partir de la información que se obtiene desde Django y que no requiera, por tanto, muchas transformaciones que influyan en la eficiencia a la hora de obtener y formatear la información.

En Internet podemos encontrar multitud de librerías destinadas a la representación de grafos. Sin embargo, muchas de ellas no cumplen con algunos de los objetivos marcados. Entre nuestras principales alternativas para la representación del grafo, se encontraban las librerías *arbor.js*, *sigma.js* y *Dracula Graph Libray*. De entre todas ellas, la única que cumple con todos los requisitos marcados es *sigma.js* ya que aparte de cumplir todas las restricciones en relación con la forma de representación, obtiene los datos del grafo a partir de un fichero en formato *JSON*, que además de ser un formato estándar de intercambio de datos, puede obtenerse fácilmente desde Django. El hecho de descartar *arbor.js* está en que no permite representar los nodos de diferentes tamaños, algo fundamental en nuestro caso para conocer la relevancia de los elementos en el grafo. Por otro lado, *Dracula Graph Libray* se descarta por su complejidad para representar los grafos, y por la ausencia de un formato estándar a partir del cual obtener los datos que compondrán el grafo.

Además, y para hacer más sencilla la comprensión del grafo y la identificación de los elementos principales que marcan la actividad de la red social, se añaden varias tablas que recopilan la información más significativa del grafo. La primera de ellas recoge las cinco etiquetas más utilizadas, lo que permite conocer fácilmente cuál o cuáles son los temas de mayor importancia. Las dos siguientes tablas se encargan de identificar, por un lado, los usuarios más activos, es decir, los que están generando mayor número de publicaciones, y por otro, aquellos usuarios más mencionados, y que por tanto, se corresponden con una personalidad o institución que está generando interés en la ciudad.

Representación geográfica de tweets : Tal y como se ha indicado, aunque la proporción de tweets geolocalizados con respecto al total es muy pequeña, el hecho de poder visualizar esta información permite conocer dónde se están desarrollando las actividades principales de la ciudad. Para ello, el método más sencillo y que ofrece mayores posibilidades de visualización, es la utilización del servicio de *Mapas de Google*. Este servicio permite, entre otras muchas características:

- La elección de las coordenadas del mapa que se quiere mostrar de forma inicial.

- La interacción del usuario con el mapa. El usuario puede dirigir el mapa a las zonas que desee, o modificar la apariencia del mismo para así adaptarlo a sus necesidades.
- Añadir cualquiera de los elementos que suelen aparecer en estos mapas: marcadores, rutas, lugares de interés...

Gracias a todas estas posibilidades se consigue crear un mapa que coincide con el área que utiliza el recolector de los tweets con información de ubicación, para capturar aquellos localizados en Ciudad Real. En este mapa, se colocará un marcador para cada tweet geolocalizado en las coordenadas almacenadas en Base de Datos para esa publicación. Además, y para aumentar la información que se le ofrece al usuario sobre los tweets con información de ubicación, se puede acceder al autor de la publicación y a su contenido pulsando sobre el marcador.

Gráfica de evolución de actividad : El último de los elementos que se muestran en la vista principal, y que completa el total de la información que el usuario puede conocer, es una gráfica que muestra el volumen de tweets que ha habido en la últimas dos horas. Esta gráfica permite conocer si el número de publicaciones se mantiene estable en todo el intervalo, o existe algún pico de actividad que pueda relacionarse con algún evento. En este caso, y puesto que la gráfica que se debe mostrar es sencilla, nos sirve prácticamente cualquier herramienta que permita dibujar gráficas. Se ha elegido Highcharts ya que es una librería de la que se tenían conocimientos previos a la elaboración del proyecto, y con la que se tenía la certeza de poder conseguir representar la gráfica de la evolución de la actividad de la red social.

Un último elemento añadido a esta vista principal, consiste en un calendario a través del cual el usuario puede elegir una fecha y hora concretas de las que visualizar la actividad y así, poder explorar o buscar eventos pasados.

Con todos estos elementos la apariencia final de la vista principal resulta en lo que podemos ver en la Figura 4.5

4.2.7 Fase VI: Desarrollo de la vista de alertas

La siguiente vista[9], permite al usuario visualizar todos los eventos pasados que han ocurrido en la red social con relación a la actividad de Ciudad Real. Como ya se ha indicado, la tarea de detección de eventos es realizada por el *analizador* que, mediante el establecimiento de un valor umbral es capaz de detectar si el volumen de publicaciones para un día y hora concretos supera, según el umbral, al volumen de publicaciones que se consideran normales para ese día de la semana y esa hora.

Para conseguir que el usuario pueda obtener la mayor información posible de las alertas,

[9]O plantilla en Django.

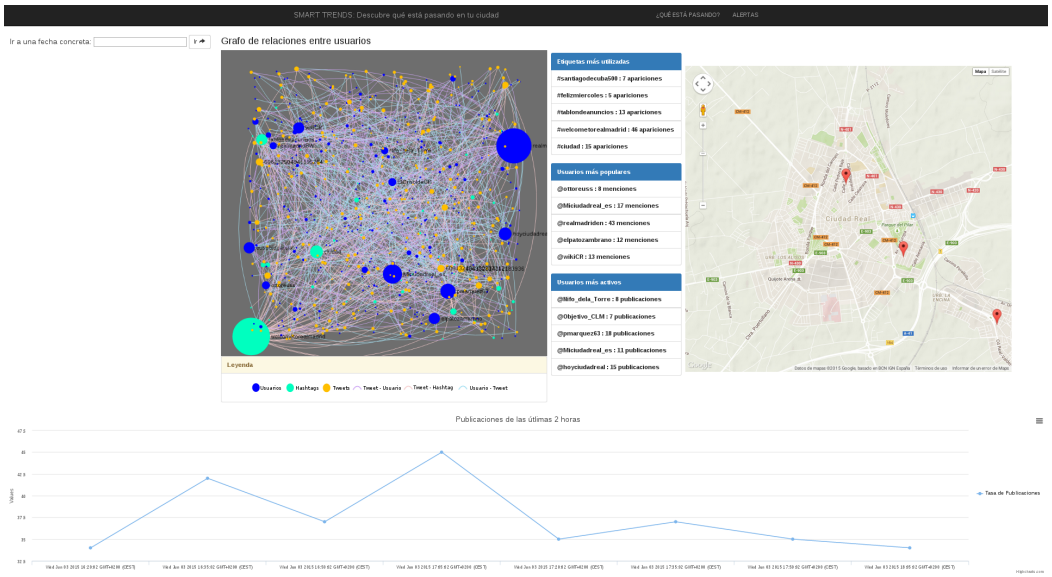

Figura 4.5: Vista principal de la aplicación Web.

esta vista, recupera información tanto a partir de SQLite3, que contiene los datos recogidos en relación al análisis de normalidad, como de Neo4J, que nos servirá para recuperar información de las alertas registradas. Así pues, inicialmente, el usuario podrá visualizar una tabla con todas las alertas registradas hasta el momento. Cada alerta estará identificada por la fecha en la que se produjo, y la hora concreta en la que el analizador detectó la alerta. Si el usuario quiere obtener más información acerca de una alerta concreta, al pulsar sobre ella obtendrá:

- Un gráfico que compara la actividad del día en que se produjo la alerta con respecto a la actividad normal de el día de la semana que se corresponde con el día de la alerta. En forma de diagrama de barras, aparecerá la media para cada hora de ese día, y en forma de línea de puntos aparecerá la actividad del día de la alerta, lo que permitirá al usuario poder comparar la actividad normal con la eventual de una forma más visual.
- Además de esto, el usuario tendrá acceso a información que le revelará cuál era el tema o la situación que desencadenó el evento. Al igual que en la vista principal, el usuario verá tres tablas que le mostrarán los cinco *hashtags* más utilizados, los cinco usuarios más *mencionados* y los cinco usuarios más activos.

Así, la vista de alertas tendrá la apariencia que se muestra en la Figura 4.6.

Además, si el usuario quiere obtener aún más información acerca del evento que desencadenó la alerta, la vista principal le permitirá seleccionar la fecha y hora de la alerta y así recuperar toda la información completa.

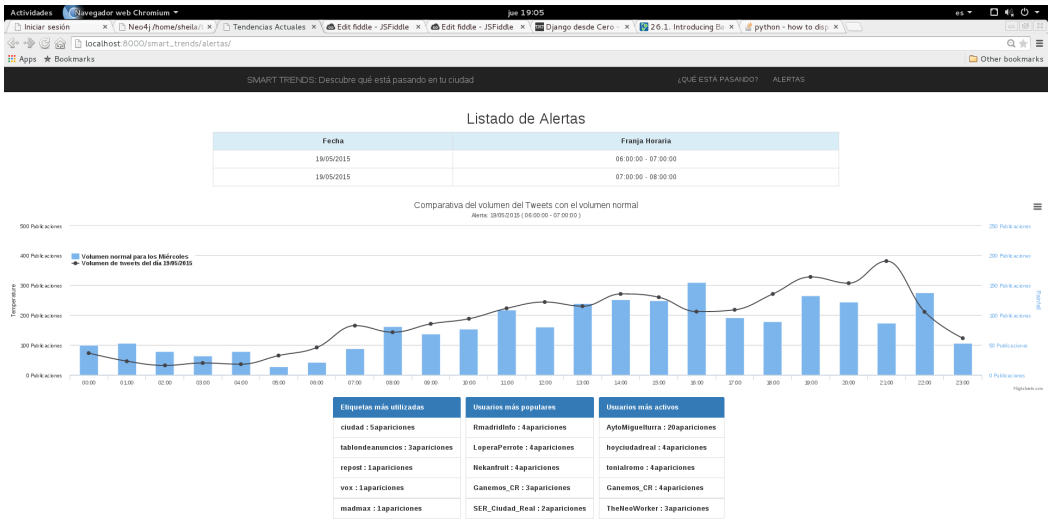

Figura 4.6: Vista de alertas de la aplicación Web.

Capítulo 5

Arquitectura de la aplicación

PARA poder cumplir con todos los objetivos enunciados en el Capítulo 2, y como resultado de todas las fases de desarrollo descritas en la sección 4.2 se ha desarrollado una plataforma que sigue la arquitectura que se muestra en la Figura 5.1. Tal y como podemos ver la arquitectura puede dividirse en cuatro partes principalmente: *Recolectores* , *Analizador* , parte de *Visualización* y por último el núcleo de la plataforma. Las tres primeras partes son fundamentales para conseguir todos los objetivos marcados, y el núcleo, formando por Django y Neo4J, se encarga de unificar e interpretar toda la información procedente tanto de los recolectores como del analizador, además de permitir representar – en la parte de Visualización – toda la información obtenida de estas dos fuentes.

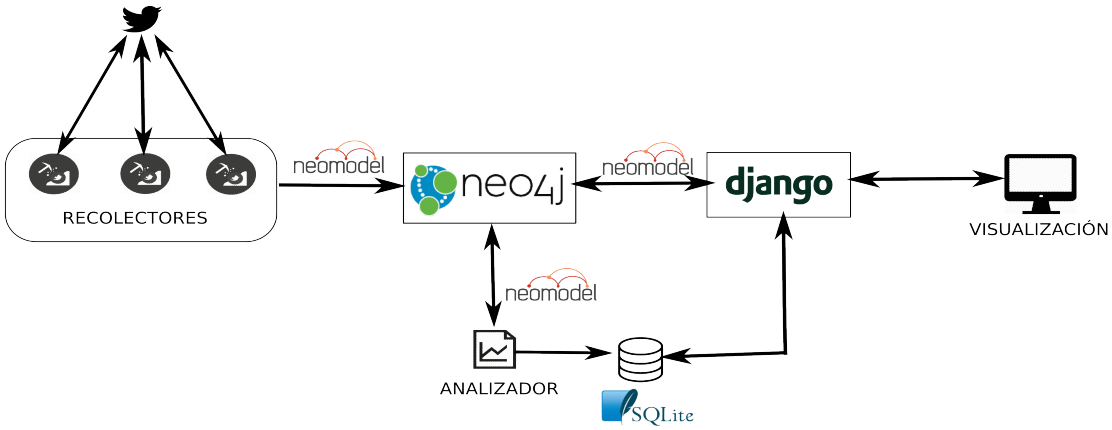

Figura 5.1: Arquitectura de la Plataforma.

Así pues, en primer lugar los *Recolectores* se encargan de recoger datos de Twitter en torno a la actividad que se está generando en torno a Ciudad Real. Para ello, se han desarrollado tres tipos de Recolectores que obtienen publicaciones atendiendo a tres criterios diferentes:

- *Publicaciones de usuarios de Ciudad Real:* Este Recolector, realiza peticiones a Twitter para conocer las últimas publicaciones de los usuarios de Ciudad Real almacenados en Neo4j.
- *Publicaciones geolocalizadas en Ciudad Real:* Este Recolector se encarga de realizar

peticiones a Twitter para conocer qué publicaciones se han hecho en la red social que estén geolocalizadas en Ciudad Real. Gracias a este elemento se puede conocer la ubicación exacta del usuario en el momento en el que hizo de la publicación. Esta información es de gran utilidad ya que nos permite conocer en qué lugar de la ciudad se está desarrollando un evento de importancia, cuando la mayoría de las publicaciones geolocalizadas se encuentren en torno a un mismo lugar.

- *Publicaciones que contengan «Ciudad Real»:* Este Recolector, al igual que el anterior, realiza peticiones a Twitter para obtener un conjunto concreto de publicaciones, pero, en este caso, se buscan publicaciones en las que entre su contenido se encuentre «Ciudad Real». Con este Recolector se pueden conseguir un gran número de publicaciones que se están desarrollando, o que tratan temas en torno al área de Ciudad Real, pero que sin embargo no cumplen con ninguno de los dos criterios de búsqueda de los dos Recolectores anteriores.

A partir de las publicaciones recogidas, cada uno de los recolectores es capaz de extraer de ellas tanto las etiquetas – *hashtags*– utilizadas como el usuario autor de la publicación y los usuarios a los que se *menciona* en cada publicación. Toda esta información, entre la que se incluye por supuesto, las relaciones existentes entre todos los elementos recogidos, se almacena en la Base de Datos Orientada a Grafos, Neo4J.

Otra parte fundamental en la arquitectura de la plataforma es el *Analizador*, cuyo objetivo es realizar el análisis de normalidad. El Analizador realiza una cuenta del número de publicaciones que ha habido para cada hora del día. Para ello, realiza peticiones a Neo4J. Cuando el Analizador recibe la cuenta determina si el volumen de publicaciones es o no «normal». Si el volumen de publicaciones supera el umbral permitido para el número de publicaciones habitual para esa hora del día y ese día de la semana, guardará la cuenta – en una Base de Datos SQLite3 – para esa hora identificándola como una alerta, si no, guardará la cuenta sin más.

Estas dos partes, los Recolectores y el Analizador, se encargan de la parte de procesamiento interno de la herramienta. Su estructura permite que puedan cambiarse por otros, o que realicen funciones atendiendo a otros criterios, sin que para ello haya que modificar el resto de la plataforma. Por ejemplo, se podrían cambiar los criterios de búsqueda de los Recolectores, para que se buscase información de otra ciudad.

La parte de *Visualización*, por su parte, es la parte que está de cara al usuario final, y que le permite interpretar los datos obtenidos en la parte de procesamiento para así conocer cuál es la evolución de la red social, qué está sucediendo en cada momento y qué eventos importantes han ocurrido. Para ello, el navegador del usuario, realizará peticiones a Django, que, a su vez, realizará peticiones tanto a *Neo4J* como a *SQLite3* para obtener la información solicitada.

Además de estos elementos, es necesario destacar la presencia de *Neomodel* como elemento fundamental para la interacción de todos los componentes de la herramienta con Neo4J. La función de Neomodel es la de adaptador, ya que se encarga tanto de realizar peticiones a la Base de Datos que cumplan con la función que se está invocando desde cualquiera de los componentes, como de transformar la respuesta de la Neo4J a un formato de datos orientado a objetos y entendido por Python, lenguaje de programación en el que están implementados los elementos que necesitan obtener información de la Base de Datos.

Capítulo 6

Resultados

COMO resultado del presente proyecto se ha obtenido una aplicación Web capaz de mostrar información sobre la actividad y eventos que ocurren en Ciudad Real. Dicha aplicación consta de dos vistas. La primera, permite visualizar la actividad comprendida en un periodo de dos horas, bien desde la hora actual y dos horas hacia atrás, o bien desde la fecha y hora seleccionadas. La segunda vista, nos ofrece un listado de todas las alertas registradas que se corresponden con un evento o situación anormal en la red social, además de diferente información sobre cada una de las alertas. Ahora bien, ¿cómo podemos interpretar cada una de las informaciones que se ofrecen en ambas vistas? Para comprender el valor de todo lo que se ofrece en cada una de las vistas, debemos mostrar el comportamiento de las mismas, sobre todo el de la vista principal, ante periodos de tiempo de actividad normal y periodos de tiempo en los que se de algún suceso de interés para la sociedad.

6.1 Caso de Estudio I: Comportamiento ante actividad normal

Lo primero que debemos hacer, es estudiar y analizar la información que nos ofrece la vista principal cuando la actividad de red social es *normal*. Si nos fijamos en la Figura 6.1 observaremos varias cosas. En primer lugar vemos un grafo con relativamente pocas relaciones, lo que nos indica que no hay muchos usuarios de la red social en activo. Es cierto que hay usuarios cuya actividad es superior a la de otros usuarios, como por ejemplo *@Miciudadreal_es* o *wikiCr*. Sin embargo, esta situación no nos ofrece ningún indicio de actividad anormal por varias razones:

- En primer lugar, este usuario se corresponde con un perfil que se dedica a escribir tweets acerca del día a día de Ciudad Real, por lo que una actividad superior de este tipo de perfiles, no es significativa a la hora de deducir si se ha producido un evento.
- Por otro lado, se observa que no hay ningún *hashtag* predominante sobre el resto. Ya que los hashtag se utilizan para clasificar las publicaciones por temas, podemos afirmar que no hay ningún evento que esté provocando reacción alguna entre los usuarios de Ciudad Real.

Por otro lado, si nos fijamos en el mapa donde podremos ver qué publicaciones de entre todas las que se recogen en el grafo están *geolocalizadas*, observamos que sólo hay un tweet que

incluya la ubicación del usuario. Como ya hemos dicho, aunque el uso de la ubicación es muy escaso, pequeñas concentraciones de tweets geolocalizados en un mismo área podrían indicarnos el lugar donde se está dando un evento. En este caso, es evidente que ningún usuario está utilizando la geolocalización para informar a sus seguidores del lugar donde está y en el que puede estar ocurriendo un evento relevante. Finalmente, y fijándonos en la gráfica de actividad de la últimas dos horas, observamos un volumen de publicaciones bajo, que en ningún caso supera las treinta publicaciones cada quince minutos. Toda esta información interpretada de manera conjunta, nos indica de manera clara que la ausencia de sucesos extraordinarios en el área donde se está recogiendo la información se refleja también en una actividad de la red social baja.

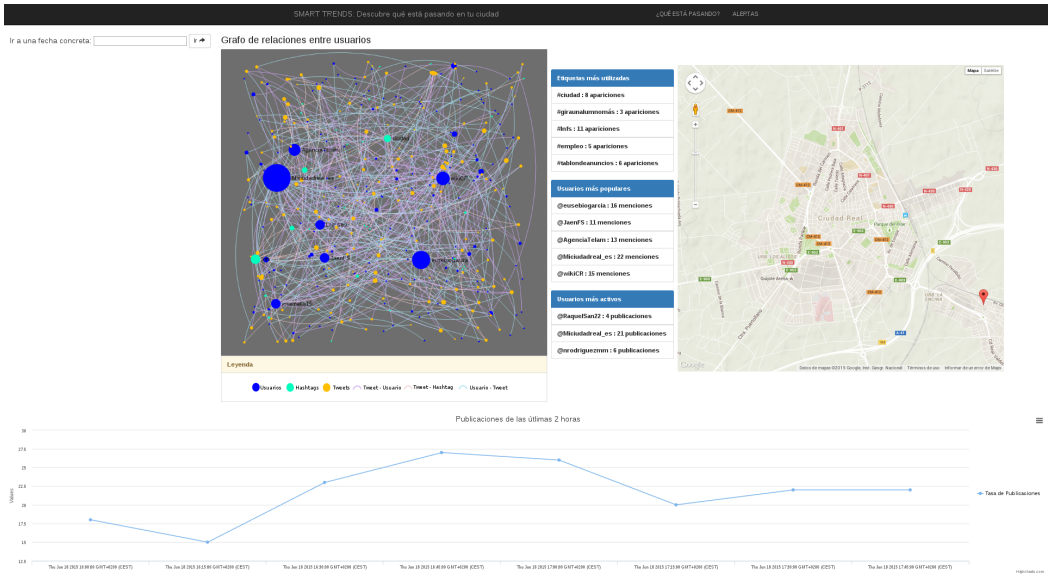

Figura 6.1: Vista Principal: Actividad Normal

6.2 Caso de Estudio II: Elecciones municipales 2015

Una vez analizada la actividad de la red social en ausencia de eventos, podemos comparar y estudiar cómo se comporta la red social ante un acontecimiento de importancia para un subconjunto amplio e importante de la población. Para observar cuál es el comportamiento de la red social ante sucesos anormales, debemos analizar cuál será su comportamiento cuando se produzcan en la sociedad situaciones que previsiblemente causarán un fuere impacto en la sociedad. Para mostrar esta situación se toma el ejemplo de las *Elecciones Municipales* de Mayo de 2015. Así, para analizar esta situación nos serviremos de las dos vistas que nos ofrece la aplicación.

En primer lugar, debemos fijarnos en la vista de alertas, para saber si se han generado alertas correspondientes al veinticuatro de Mayo, día de las elecciones. En este vista, observamos que se han producido alertas para prácticamente todos las horas del día. De hecho, en las Figuras 6.2 y 6.3 observamos que el volumen de publicaciones para los domingos es bas-

tante inferior al volumen registrado el domingo 24 de Mayo. Como podemos ver, el número de publicaciones a las nueve de la mañana ya era de 203 publicaciones frente a la media de once publicaciones que suele registrarse. Del mismo modo, cuando la jornada de votaciones ha finalizado, y los tanto ciudadanos como partidos políticos y ayuntamientos se encuentran expectantes ante el resultado de las elecciones, podemos observar un aumento mucho más acusado de publicaciones – 711 publicaciones registradas a las nueve de la noche, frente a una media de 58 publicaciones –.

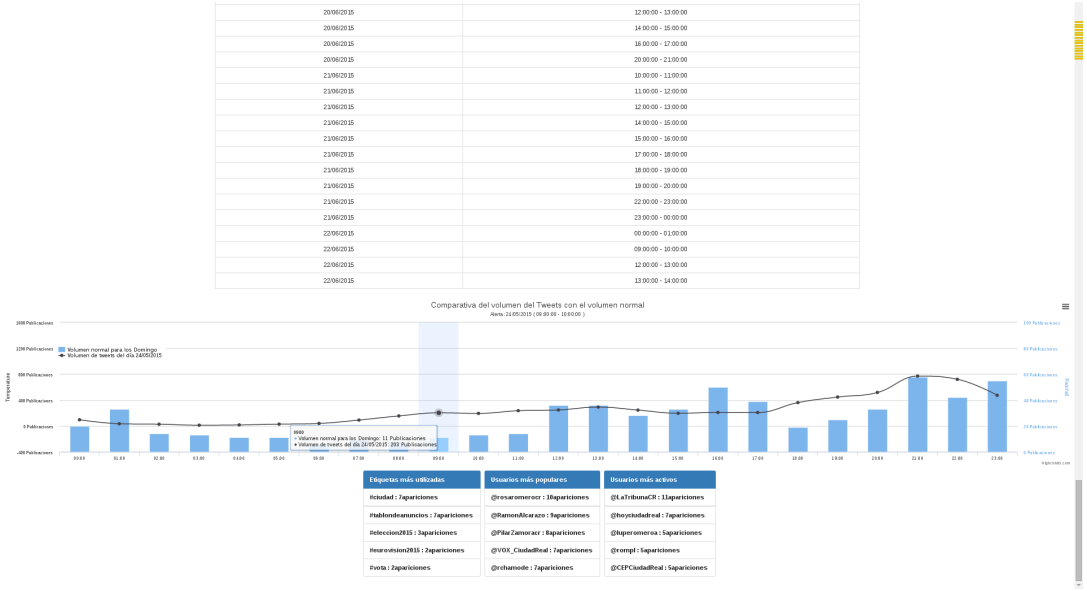

Figura 6.2: Vista de Alertas: Elecciones Municipales 2015. Actividad a las 9 a.m.

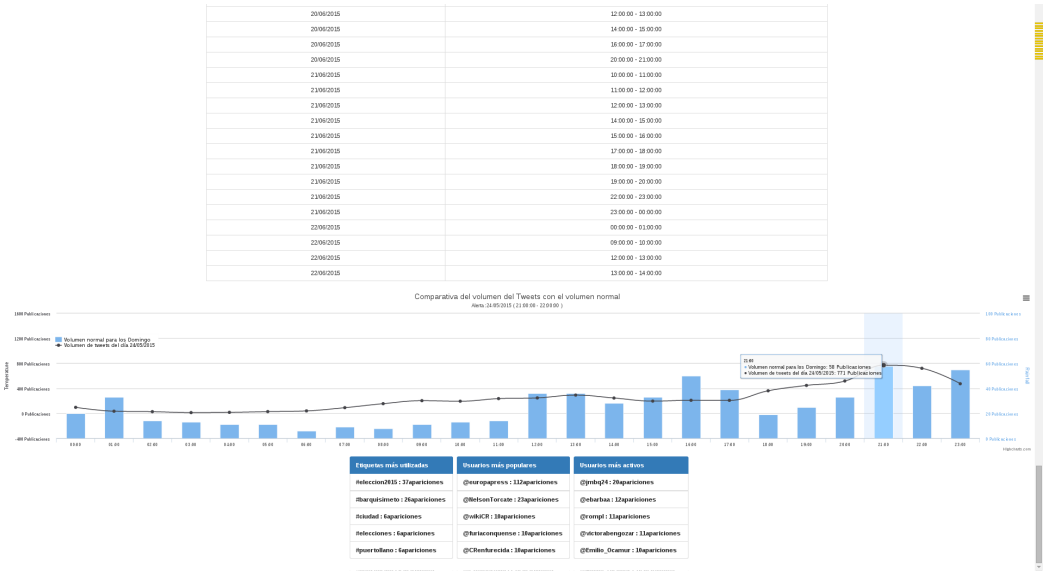

Figura 6.3: Vista de Alertas: Elecciones Municipales 2015. Actividad a las 9 p.m.

Esta situación es también bastante notable en la vista principal de la aplicación, tal y como podemos ver en la Figura 6.4. Resulta bastante interesante observar y comparar las diferen-

cias existentes entre esta Figura y la Figura 6.1, que nos mostraba la actividad normal. En primer lugar, observamos dos grafos de actividad bien distintos, donde el número de relaciones es visiblemente superior en el grafo que recoge la actividad del 24 de Mayo comprendida entre las 19:00 y las 21:00 que en el que representa un periodo de actividad normal. Además del considerable aumento en el número de relaciones, en este caso, resulta bastante representativo el hecho de que algunos nodos del grafo sean considerablemente mayores al resto de nodos. Si nos fijamos en los *usuarios más populares* (aquellos que han sido *mencionados*) un mayor número de veces, encontramos tanto perfiles informativos, como *@wikiCR*, que el resto de usuarios menciona para obtener respuesta de la actividad relativa a las Elecciones, como perfiles relacionados con personalidades y partídos políticos (*@PilarZamoracr, @PPopular*).

Por otro lado, el mapa de tweets geolocalizados, también se hace eco de esta situación, registrando esta vez un mayor número de publicaciones con información de ubicación. Sin embargo, y dado que el evento no es un suceso que esté ocurriendo en un lugar concreto de Ciudad Real, los tweets se encuentran dispersos de manera aleatoria por todo el mapa, por lo que el mapa, en este caso, no nos aporta mucha información más allá de ir en consonancia con el aumento general de publicaciones. Finalmente, y si nos fijamos en la actividad recogida por la gráfica, podemos ver un efecto bastante interesante en la misma: La primera mitad de la gráfica registra un volumen de publicaciones bajo en comparación con la segunda mitad de esta. Este aumento, justo a partir de las 20:00 se debe precisamente al cierre de las urnas electorales, por lo que los usuarios comienzan a evaluar la jornada de votaciones con una mayor intensidad.

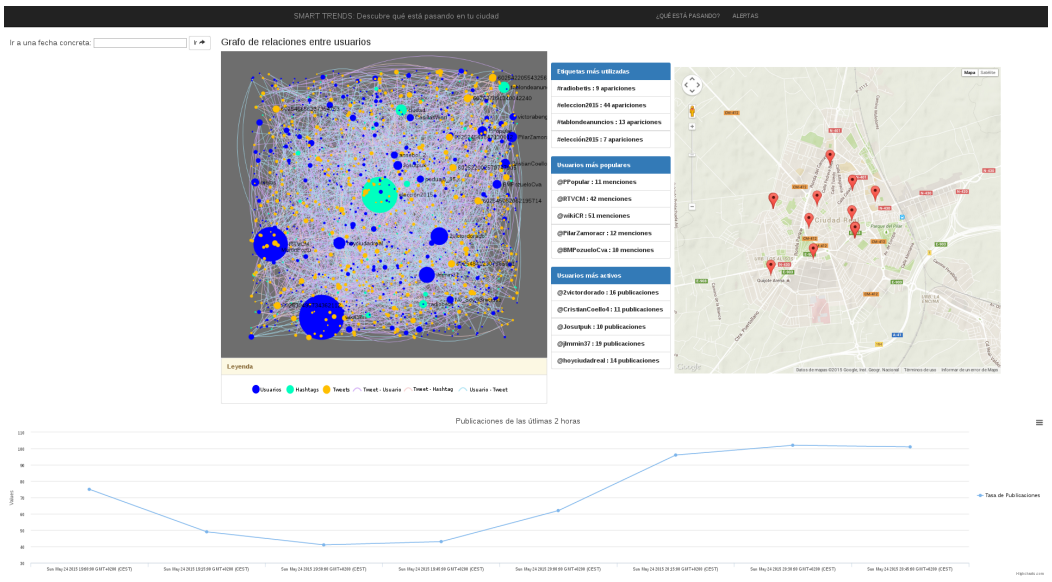

Figura 6.4: Vista Principal: Elecciones Municipales 2015.

Este análisis nos permite tener la certeza de que eventos de interés en la sociedad causan también un gran impacto en las redes sociales y que nos permite conocer, gracias a las eti-

quetas empleadas y los usuarios más activos y populares, cuál es el tema que está causando ese aumento en la actividad de los usuarios de la ciudad.

6.3 Caso de Estudio III: Fenavin 2015 y eventos esporádicos

Un último caso de estudio nos muestra el comportamiento de la red social en presencia de varios eventos. En este caso, se pretendía observar el comportamiento de la red social ante un evento de importancia para Ciudad Real, pero centrado en un colectivo previsiblemente más pequeño que el que se interesaba por el resultado de las Elecciones Municipales. El evento escogido no es otro que *Fenavin*, que se celebra cada dos años en Ciudad Real, y que sirve como impulso para el sector vinícola nacional e internacional. En este caso, el evento se celebró en los días del doce al quince de Mayo de 2015. Observando la vista del listado de alertas, observamos que se produjeron varias alertas para el día doce de Mayo. Tal y como se puede ver en la Figura 6.5, la actividad es considerablemente superior a la media de actividad entre las tres y las 9 de la mañana, y entre las doce del mediodía y las cinco de la tarde. Fijándonos, por ejemplo, en el segundo grupo de alertas observamos una peculiaridad, y es que el pico de actividad, no se debe únicamente a la celebración de Fenavin, si no que ha sido producido también por otro evento.

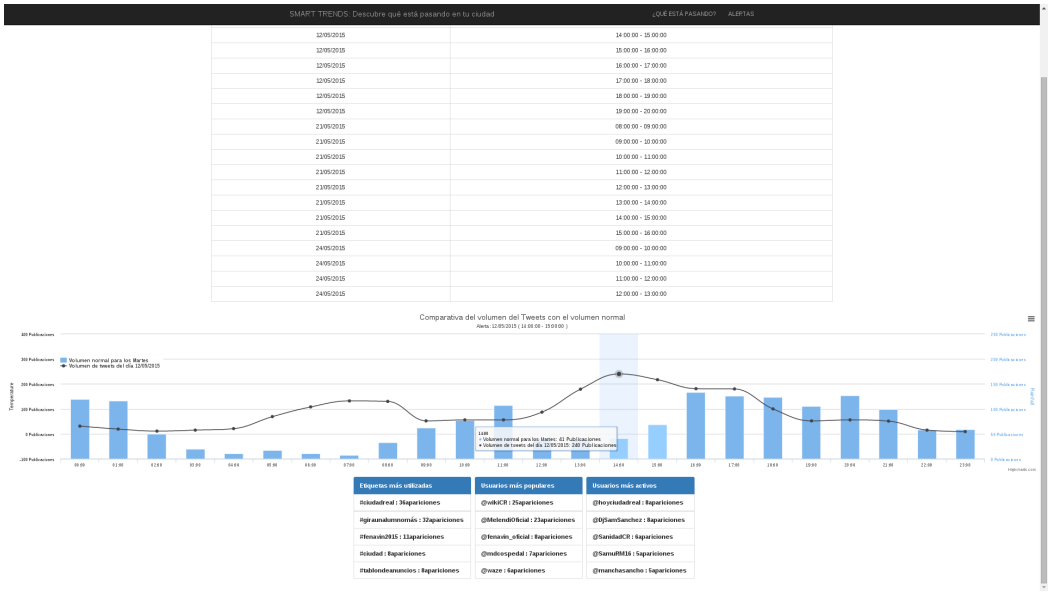

Figura 6.5: Vista de Alertas: Fenavin 2015. Actividad del doce de Mayo

En este caso, las etiquetas más utilizadas son *#ciudadreal*, *#giraunalumnomás* y *fenavin2015*, y los usuarios más populares son *@MelendiOficial* y *@wikiCR*. Con esta información, y utilizando también la vista principal, que nos permite también conocer el contenido de los tweets, nos damos cuenta, de que ha sido la confirmación del concierto del artista Melendi, para el día 27 de Junio, la causa principal del aumento en el número de publicaciones. La Figura 6.6 nos muestra esta situación. En este caso, en el grafo de relaciones, se puede observar claramente la presencia de las tres etiquetas más utilizadas, y que nos sirven para

conocer los temas más importantes en ese momento en Ciudad real. Además, en este caso, el mapa con los tweets geolocalizados, sí nos ofrece información representativa. A diferencia de lo que podíamos ver en la Figura 6.4, donde el mapa no nos aportaba información relevante, en este caso observamos la presencia de varias publicaciones concentradas en un mismo lugar, que coincide con el Recinto Ferial de Ciudad Real, lugar donde se está celebrando Fenavin.

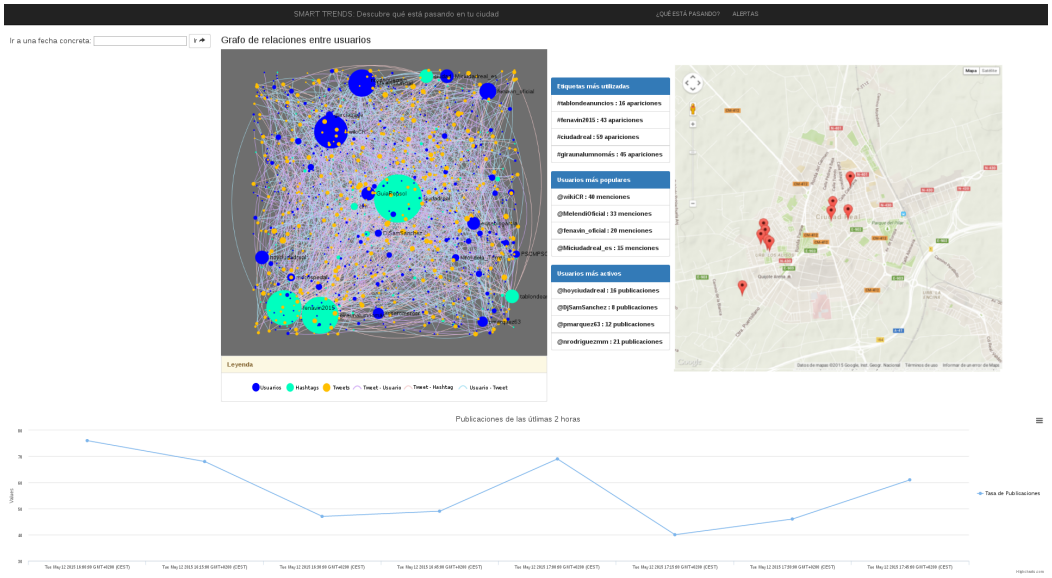

Figura 6.6: Vista Principal: Fenavin 2015 + Concierto de Melendi.

Todos estos casos de estudio ponen de manifiesto la importancia que tienen hoy las redes sociales en la vida cotidiana de las personas, por lo que nos sirven como «termómetro» de la actividad que registran diferentes comunidades. Además, gracias al análisis realizado observamos que las redes sociales pueden servirnos tanto para evaluar la importancia que tiene un evento conocido a priori, como para detectar la presencia de eventos que ocurren de manera totalmente imprevista.

Capítulo 7

Trabajos Futuros

Los resultados obtenidos con este proyecto, pueden considerarse la «semilla» de multitud de aplicaciones y plataformas centradas en ofrecer una visión lo más real posible acerca de las diferentes actividades que pueden surgir en una ciudad o cualquier otro entorno bien delimitado. Así pues, una ampliación de este proyecto podría ofrecer nuevas capacidades tales como:

- La posibilidad de ofrecer información de manera conjunta y homogénea acerca de la actividad de diferentes redes sociales de manera simultánea, permitiendo ampliar así el conjunto de la población que se captura, y por tanto, dando lugar a un sistema que pueda acercarse cada vez más al movimiento real de la comunidad.
- Desarrollar un sistema de predicción de eventos. Para ello, podrían aplicarse multitud de técnicas de análisis que estudien de manera más profunda tanto las relaciones como el volumen de actividad de la red social. El uso de técnicas complejas, unido a un mayor periodo de observación de la red social, ofrecería una visión más amplia y fiable que permitiría predecir eventos gracias al conocimiento almacenado sobre eventos pasados.
- Una mayor interacción por parte del usuario de la aplicación Web, permitiéndole, por ejemplo, indicar el área o comunidad del que se quiere conocer la información.
- Explotar más el valor de los datos almacenados. La ingente cantidad de información almacenada puede servir para muchos otros propósitos más allá de la posibilidad que se ofrece actualmente, que no es otra que la de conocer eventos anormales. Sin embargo, un análisis más exhaustivo de la información que se está almacenado, permitiría separar a los usuarios de la comunidad por rangos de edad, aficiones o intereses comunes para así conocer de una manera más concreta, por ejemplo, cuáles son los temas de actualidad que afectan a cada grupo de usuarios o cuáles son los lugares más frecuentados para cada subgrupo de la comunidad. La plataforma podrá sugerir lugares o actividades de ocio que tienen una valoración general positiva por los usuarios que coincidan con el perfil del usuario solicitante.
- Predecir el éxito o fracaso que tendrá una actividad cultural (concierto, feria, etc.) o estratégica en términos empresariales (campaña publicitaria) en relación con el resultado

de eventos similares que ocurrieron en el pasado.

- Conocer los eventos que ocurren en un entorno concreto.Como si de un periódico u otro elemento informativo se tratase, la plataforma podría ofrecer un resumen de los eventos populares en cada momento, lo que permitiría mantener informado al usuario de la actualidad de su ciudad.
- Informarse sobre temas de interés y conocer la evolución de los mismos en la comunidad. Ofreciendo la posibilidad de configurar alertas, un usuario podría seleccionar los temas que le interesan y recibir notificaciones, por ejemplo, en el correo electrónico, cuando estos temas sean también temas de interés en su ciudad.

Todos estos nuevos objetivos nos sugieren que la plataforma puede ser de enorme utilidad en el entorno de una Ciudad Inteligente, donde todas las infraestructuras están coordinadas para mejorar la calidad de los servicios de la misma, y la experiencia de los usuarios con ella.

Capítulo 8

Conclusiones

COMO resultado principal de este proyecto, hemos conseguido desarrollar un sistema capaz de acceder a la información de las redes sociales – en nuestro caso *Twitter*– para conocer la actividad de una ciudad concreta y poder ofrecer información de importancia para las organizaciones, instituciones y personas que se mueven en el entorno de dicha ciudad. Dicho sistema es capaz de detectar sucesos anormales o extraordinarios relacionados con la ciudad, ofreciendo información detallada a cerca de los mismos.

Para ello se ha desarrollado una plataforma que es capaz de seleccionar la información que pertenece a la ciudad objetivo, distinguiéndola de otra información para así conseguir información realista y coherente acerca de lo que está ocurriendo en la ciudad, o sobre las impresiones de sus ciudadanos sobre temas de importancia en su entorno. Además, mediante el uso de una técnica de *análisis de normalidad* tan sencilla como es el uso de la media para detectar si el número de publicaciones de un momento determinado es notablemente superior al número habitual de publicaciones registradas en la red social, hemos podido aislar y recuperar información de sucesos anormales o eventos que suscitan intereses e inquietudes sobre los usuarios de la red social que se concentran en la ciudad.

Por otro lado, gracias al uso de Neo4J hemos logrado construir una estructura de almacenamiento eficiente e intuitiva en su diseño, que ha permitido representar de manera natural las relaciones que se dan en la red social cuando los usuarios interactúan en ella. Gracias también al uso de esta Base de Datos Orientada a Grafos, ha sido posible dar respuesta al gran volumen de datos que el sistema debía soportar, permitiendo que las operaciones de acceso y recuperación de información no supongan una pérdida de rendimiento que afectaría de forma negativa al servicio ofrecido por la aplicación Web.

Asimismo, y a partir de los elementos encargados del procesamiento y almacenamiento de la información, se ha construido una aplicación Web que permite conocer la actividad de Twitter en torno a los usuarios de Ciudad Real de forma sencilla y representativa. Gracias a ella, se pueden conocer los temas más importantes que surgen en cada momento en torno a la ciudad, así como los usuarios más influyentes o activos de ese área. También, se ha conseguido plasmar en la aplicación Web la actividad normal de la red social, permitiendo compararla con los eventos ocurridos en la ciudad.

Por otro lado, y en relación con la *Tecnología específica* cursada, **Ingeniería de Computadores**, podemos afirmar que se han conseguido superar las competencias específicas de la misma. El Cuadro 8.1 recoge una justificación de todas las competencias que se superan en el presente proyecto.

Cuadro 8.1: Justificación de las competencias específicas abordadas en el TFG

Competencia	**Justificación**
Capacidad de analizar y evaluar arquitecturas de computadores, incluyendo plataformas paralelas y distribuidas, así como desarrollar y optimizar software para las mismas.	Se espera generar y tener que gestionar gran volumen de datos provenientes de la red social monitorizada, solo un entorno distribuido puede lidiar con esta cantidad de información.
Capacidad de diseñar e implementar software de sistema y de comunicaciones.	Tanto el acceso a la información como la puesta a disposición de los usuarios se realizará con dispositivos conectados a la red formando un sistema distribuido multi-dispositvo. El hecho de haber escogido una aplicación Web como medio de representación de los datos, permite su accesibilidad desde una gran variedad de dispositivos.
Capacidad para comprender, aplicar y gestionar la garantía y seguridad de los sistemas informáticos	Dado el carácter personal de los datos a gestionar la seguridad constituirá un aspecto clave del proyecto
Capacidad de analizar, evaluar y seleccionar las plataformas hardware y software más adecuadas para el soporte de aplicaciones empotradas y de tiempo real.	La información de las redes sociales y la obtención de información se realizan en tiempo real no estricto para el cual se deben seleccionar la plataforma hardware/software mas adecuado.

Así pues, podemos afirmar que se han cumplido los objetivos iniciales y de este modo, hemos conseguido desarrollar un sistema que explota la actividad de las redes sociales, con el que damos respuesta a todas las incógnitas iniciales que situaban a las redes sociales como una fuente de información potencialmente útil para conocer intereses y actividades sociales,

y que con el desarrollo de este proyecto hemos logrado poner de manifiesto la utilidad real de las mismas como elemento clave de cara conocer la actividad real de una comunidad. Además, este proyecto abre las puertas a numerosas aplicaciones que pueden resultar de interés tanto para organismos e instituciones públicas como para usuarios corrientes que desean conocer diversa información acerca de los eventos, intereses, opiniones o preocupaciones de carácter social que mueven a una comunidad y que hacen que esto quede reflejado en las redes sociales.

ANEXOS

Anexo A

Instalación

DADO el gran número de tecnologías y herramientas que han sido necesarias para poner en funcionamiento la aplicación, resulta conveniente explicar cuál ha sido el proceso de instalación. A continuación, se listan todos los elementos que han sido necesarios:

- Django, como framework para el soporte y desarrollo de la aplicaión Web.
- Neo4j, como base de datos para el almacenamiento de toda la información procedente de la red social.
- NeoModel, que actúa como adaptador para poder acceder a Neo4J desde Python.
- TwitterApi y Tweepy, como APIs para el acceso a la información de Twitter.

Además resulta conveniente indicar que se ha trabajado con Python en su versión 2.7.8, ya que es el lenguaje principal en el desarrollo de la aplicación y, en ocasiones, resulta determinante el uso de una determinada versión de Python, por ejemplo, para poder trabajar con las APIs de acceso a Twitter o para poder utilizar NeoModel para el acceso a la base de datos. También es importante, que a la hora de seguir esta guía de instalación se utilicen las versiones que se indican, ya que es un punto clave para la compatibilidad e interoperabilidad de los elementos. En primer lugar, instalaremos Django en su versión 1.7.5 . Su instalación no requiere ningún requisito adicional a parte de utilizar Python bien en su última versión (Python 3) o utilizar la versión 2.7. Así, para la instalación, escribiremos en un terminal:

```
pip install Django==1.7.5
```

Cuando el proceso de instalación haya terminado, podemos comprobar que la instalación ha sido correcta escribiendo en el intérprete de Python:

```
>>> import django
>>> print(django.get_version())
```

Con lo que obtendremos la versión instalada, en nuestro caso la 1.7.5.

El siguiente elemento a instalar es Neo4J. En este caso, los requisitos del sistema para la instalación de la base de datos son menos flexibles, ya que se requiere que el sistema pueda ofrecer un rendimiento mínimo para el buen funcionamiento de la base de datos. Así pues, la instalación de Neo4J requiere:

- Que la CPU sea como mínimo Intel Core i3 aunque se recomienda Intel Core i7 para permitir un buen rendimiento de la memoria de E/S a la hora de acceder y recuperar grafos de gran tamaño, así como para poder hacer frente a cómputos sobre los gráficos residentes en memoria. En nuestro caso, se ha trabajado con 2 equipos: un con procesador Intel Core i5, y el otro con procesador Intel Core i7. En ninguno de los casos, ni los cómputos ni el acceso a memoria para acceder al gráfico, han supuesto una pérdida de rendimiento en la aplicación
- Que el disco sea mínimamente un disco SATA, aunque se recomienda un disco SSD. En este caso, la utilización de un tipo de disco u otro está justificada por la rapidez de acceso, siendo los discos SSD más rápidos que los SATA. En nuestro caso, y ya que no resulta crítico un acceso a disco rápido se ha utilizado como almacenamiento el ofrecido por ambos equipos, es decir, discos de tipo SATA.
- Que el sistema de ficheros sea EXT4 o ZFS. En nuestro caso, ya que estamos trabajando con equipos Linux, este requisito está cubierto, ya que el almacenamiento en estos sistemas se monta en discos con sistema de archivos
- Por último, y ya que Neo4J está basado en Java, se requiere OpenJDK7.

Con todos los requisitos cubiertos, podemos pasar a instalar Neo4J. En este caso, la versión instalada es la 2.0.4. La elección de esta versión en particular queda justificada para poder utilizar NeoModel. Así, la instalación se realiza de la siguiente manera:

1. Descargamos la versión 2.0.4 del sitio oficial de Neo4J y descomprimimos el fichero.
2. A continuación, accedemos al directorio raíz de Neo4j y ejecutamos, como superusuario:

```
./bin/neo4j console.
```

El siguiente elemento que debemos instalar es NeoModel, cuya función es poder lanzar consultas a la base de datos, directamente desde Python. Los requisitos necesarios son: trabajar con Python 2.7 o 3.4 y utilizar las versiones 2.0 o 2.1 de Neo4j. Para su instalación, basta con ejecutar desde el terminal:

```
pyp install neomodel
```

Por último, el los últimos elementos a instalar son las APIs para el acceso a la información procedente de Twitter. La primera de ellas, `python-twitter` requiere como única restricción el uso de una versión de Python superior a la 2.6. Para su instalación ejecutamos:

```
pip install python-twitter
```

Finalmente, para instalar la API `tweepy`, ejecutamos:

```
pip install tweepy
```

En este caso, el único requisito es utilizar Python en sus versiones 2.6 , 2.7, 3.3 o 3.4.

Anexo B

Introducción al lenguaje Cypher

CYPHER es un lenguaje declarativo inspirado en SQL que permite realizar consultas sobre grafos. Permite realizar operaciones de selección, inserción, actualización o eliminación de elementos de la Base de Datos orientada a Grafos.

Para ilustrar su funcionamiento, utilizaremos un ejemplo[1]. Supongamos que nuestra Base de Datos sirve para almacenar las relaciones existentes en el mundo del cine, es decir, qué persona actuó en qué película, bajo que rol actuó, quién dirigió qué película... Así, nuestra Base de Datos tendrá una estructura similar a la que podemos ver en la Figura B.1. Como podemos ver, tenemos dos tipos de nodos en nuestra Base de Datos: Personas y Películas. Las Personas pueden relacionarse entre sí mediante la relación «`KNOWS`», que indica que las dos personas se conocen. Esta relación cuenta además con el atributo «`since`» que indica desde qué año se conocen las dos personas. Por otra parte, las personas pueden relacionarse con las películas de dos maneras: Mediante la relación «`ACTED_IN`», que indica que esa persona ha actuado en esa película además de indicar bajo que rol lo hizo. También podemos relacionar una persona con una película mediante la relación «`DIRECTED`» que indica qué persona dirigió la película.

Cypher utiliza el estilo ASCII para representar las consultas a la Base de Datos. Así, para representar un nodo se emplean los paréntesis, para que el nodo parezca un círculo, por ejemplo `nodo`. Del mismo modo, para representar una relación entre dos nodos se emplea una flecha que puede ser de dos formas: La más simple, `->`, indica una relación sin especificar el tipo de relación, y `-[:NOMBRE_RELACION]->`, que indica además el tipo de relación. Por otro lado, si queremos acceder a las propiedades de cualquier elemento de la base de datos se emplea el punto «`.`», por ejemplo `a.name`, para acceder al nombre del actor.

Primeros pasos con Cypher

Conociendo estas propiedades básicas, podemos ejecutar consultas sencillas sobre la Base de Datos. La estructura de una consulta sigue el siguiente esquema:

```
MATCH (node) RETURN node.property
```

[1]Todos los ejemplos y definiciones han sido extraídos de «The Neo4J Manual» [Tea15], y delTutorial Oficial de Neo4J

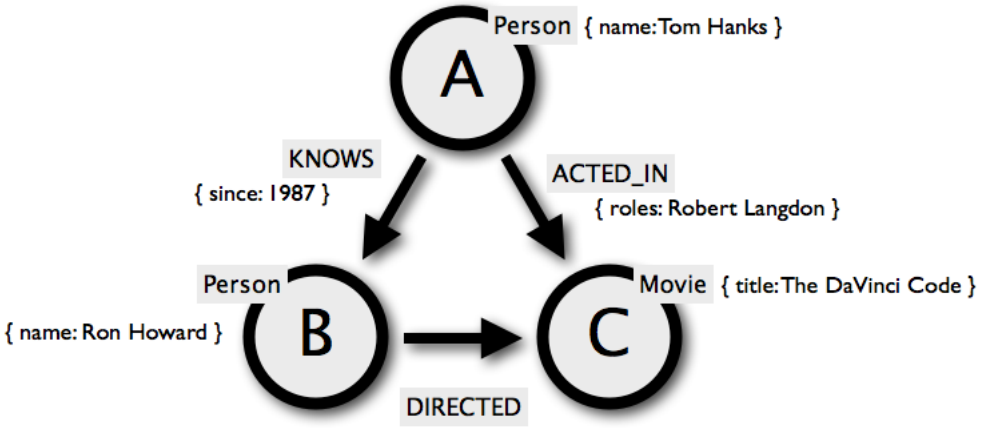

Figura B.1: Estructura de la Base de Datos. *Fuente: Tutorial de Iniciación a Neo4J*

```
MATCH (node1)-->(node2)
RETURN node2.propertyA, node2.propertyB
```

Un elemento muy importante en Cyper es el uso de *etiquetas*. Las etiquetas permiten agrupar nodos que cumplen con las características que se indican en la consulta. Su estructura básica es:

```
MATCH (node:Etiqueta) RETURN node

MATCH (node1:Etiqueta1)-[:REL_TYPE]->(node2:Etiqueta2)
RETURN node1, node2
```

Para entender bien el comportamiento de Cypher, y antes de comenzar a describir características más complejas y potentes, veremos algunos ejemplos de funcionamiento. Por ejemplo, si queremos recuperar todos los nodos de nuestro grafo:

```
MATCH (n)
RETURN n;
```

Esta consulta realiza una búsqueda completa por todo el grafo, buscando nodos que coincidan con el patrón «`(n)`». En este caso, el patrón que se busca es simplemente un nodo, que puede o no tener relaciones, por lo que el patrón coincide con todos los nodos del grafo. Con la clausula «`RETURN`» se devuelve toda la información que coincida con el patrón, en este caso, todos los nodos del grafo, incluyendo sus propiedades.

Si lo que queremos ahora es conocer qué nodos están relacionados con otro nodo:

```
MATCH (n)-->(m)
RETURN n,m;
```

Con esta consulta obtenemos todos los pares de nodos `n, m` que tienen una relación de `n` a `m`.

Haciendo referencia a las relaciones

Existen dos restricciones de integridad de los datos en Neo4J:

1. Una relación siempre tiene un nodo inicial y un nodo final – que puede ser el mismo –. Por lo que no hay relaciones colgantes o enlaces rotos.
2. Toda relación debe tener un tipo.

Cyper permite asignar un identificador a las relaciones, para poder obtener cualquiera de sus propiedades o su tipo:

```
MATCH (node)-[rel]-> ()
RETURN node, rel.property;

MATCH (node)-[rel]-> ()
RETURN node, type(rel);
```

Volviendo a nuestro ejemplo, podemos realizar una consulta que nos devuelva el nombre de las personas que han actuado en alguna película, el nombre de la película y el rol bajo el que actuaban en la misma.

```
MATCH (actor)-[role:ACTED_IN]->(movie)
RETURN actor.name, role.roles, movie.title;
```

Matching a partir de Etiquetas

Una parte importante a la hora de mejorar la eficiencia de las consultas sobre el grafo es reducir el conjunto de búsqueda a través del uso de etiquetas. Así, por ejemplo, para conocer los actores que han actuado en una película usaremos la etiqueta «`Person`», evitando así, que haya que recorrer nodos de otro tipo, por ejemplo, películas, que nunca cumplirán con el patrón que se está buscando.

```
MATCH (a:Person)-[:ACTED_IN]->(m)
RETURN a.name, m.title;
```

.

Si además queremos buscar una persona en concreto, podemos añadir más restricciones a la consulta, añadiendo el valor de la propiedad a encontrar con la clausula `WHERE`.

```
MATCH (Tom:Person)
WHERE Tom.name = "Tom Hanks"
RETURN Tom;
```

.

O de una forma más resumida, añadiendo las propiedades a buscar dentro de la definición del nodo a buscar:

```
MATCH (Tom:Person {name: "Tom Hanks"})
RETURN Tom;
```

.

Listado de clausulas de Cypher

Una vez conocemos las propiedades básicas del lenguaje Cypher, podemos enumerar y describir las clausulas utilizadas por este lenguaje. Ya que Cypher está inspirado en SQL, el funcionamiento de la mayor parte de sus clausulas funciona de manera muy similar a este lenguaje.

Clausulas de consulta sobre el grafo

- `MATCH`: Busca coincidencias en el grafo con el patrón que se indique.
- `WHERE`: Filtra usando predicados o propiedades de los elementos del patrón.
- `RETURN`: Devuelve los datos resultantes de todos los filtros realizados en la consulta. También se encarga de realizar operaciones de agregación (uso de `GROUP BY`,`COUNT` ...).
- `SKIP / LIMIT`: Pagina el resultado de la consulta:

  ```
  MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
  RETURN a.name, m.title
  ORDER BY a.name
  SKIP 10
  LIMIT 5;
  ```

  Devuelve una lista de cinco actores y su correspondiente película, ordenados por el nombre del actor, y saltándose los diez primeros actores.

Clausulas de actualización del grafo

- `CREATE`: Crea los nodos y las relaciones.

  ```
  CREATE (movie:Movie {title: "Mystic River", released:1993})
  RETURN movie;
  ```

  Crea una Película.

```
MATCH (clint:Person),(mystic:Movie)
WHERE clint.name="Clint Eastwood" AND mystic.title="Mystic River"
CREATE (clint)-[:DIRECTED]->(mystic)
RETURN clint, mystic;
```

Crea la relación «DIRECTED» entre una persona y una película concretas.

- MERGE: Crea relaciones de forma única. Usando esta clausula, nos aseguramos de que la relación que cumple con el patrón se crea sólo una vez. Si la relación ya existe, sólo devuelve los datos:

```
MATCH (clint:Person),(mystic:Movie)
WHERE clint.name="Clint Eastwood" AND mystic.title="Mystic River"
MERGE (clint)-[:DIRECTED]->(mystic)
RETURN clint, mystic;
```

Crea la relación «DIRECTED» entre una persona y una película concretas, sólo si la relación no existe.

- CREATE UNIQUE: Crea nodos de forma única. Esta cláusula es una mezcla entre MATCH y SET, ya que realiza siempre el mínimo cambio posible en el grafo, y si todas las propiedades indicadas en la cláusula ya coinciden con algún patrón del grafo, no realiza ningún cambio, simplemente devuelve el patrón correspondiente.
- DELETE: Elimina nodos y relaciones.

```
MATCH (me:Person {name="My Name"})
OPTIONAL MATCH (me)-[r]-()
DELETE me,r;
```

Borra la persona que cumple con la relación y *todas* las posibles relaciones que el nodo pudiera tener. Si no se usa la clausula OPTIONAL MATCH y el nodo tiene alguna relación, el borrado no se ejecutará.

- SET: Actualiza propiedades y etiquetas.

```
MATCH (movie:Movie)
WHERE movie.title="Mystic River"
SET movie.tagline = "We bury our sins here, Dave. We wash them
   clean."
RETURN movie;
```

Actualiza la propiedad «tagline» de la película «*Mystic River*».

- REMOVE: Elimina propiedades y etiquetas.
- FOREACH: Realiza la misma operación de actualización por cada elemento de la lista.
- WITH: Divide una consulta en múltiples partes, pasando los resultados de una parte a la siguiente.

Referencias

[AHRSA09] Mahmoud Al-Hader, Ahmad Rodzi, Abdul Rashid Sharif, y Noordin Ahmad. Smart city components architecture. En *Computational Intelligence, Modelling and Simulation, 2009. CSSim'09. International Conference on*, páginas 93–97. IEEE, 2009.

[B+01] K Beck et al. Principios del Manifiesto Ágil, 2001.

[Bes15] Bestframeworks. Compare Python Frameworks. *Recuperado de: http://www.bestwebframeworks.com/compare-web-frameworks/python/*, 2015.

[C+11] World Wide Web Consortium et al. Facts about W3C, 2011.

[CA12] José Luis Condori Ayala. Phython-DjangoFramework de desarrollo web para perfeccionistasBasado en el Modelo MTV. *Revista de Información, Tecnología y Sociedad*, página 36, 2012.

[Coh12] Boyd Cohen. The top 10 smart cities on the planet. *Co. Exist*, 11, 2012.

[DF14a] Ted Dunning y Ellen Friedman. *Practical Machine Learning: A New Look at Anomaly Detection*, capítulo 1,2, páginas 1–14. .º'Reilly Media, Inc.", 2014.

[DF14b] Ted Dunning y Ellen Friedman. *Practical Machine Learning: A New Look at Anomaly Detection*, capítulo 4, páginas 23–34. .º'Reilly Media, Inc.", 2014.

[DVMT14] Roberto De Virgilio, Antonio Maccioni, y Riccardo Torlone. R2G: a Tool for Migrating Relations to Graphs. En *EDBT*, páginas 640–643, 2014.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, y John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[GYL11] R.K. Ganti, Fan Ye, y Hui Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, November 2011.

[Kop11] Hermann Kopetz. Internet of things. En *Real-time systems*, páginas 307–323. Springer, 2011.

[LWC+13] Kalev Leetaru, Shaowen Wang, Guofeng Cao, Anand Padmanabhan, y Eric Shook. Mapping the global Twitter heartbeat: The geography of Twitter. *First Monday*, 18(5), 2013.

[Mas11] Mark Masse. *REST API design rulebook*. .º'Reilly Media, Inc.", 2011.

[NP11] Taewoo Nam y Theresa A. Pardo. Conceptualizing Smart City with Dimensions of Technology, People, and Institutions. En John Carlo Bertot, Karine Nahon, Soon Ae Chun, Luis F. Luna-Reyes, y Vijay Atluri, editors, *Proceedings of the 12th Annual International Conference on Digital Government Research, DG.O 2011, College Park, MD, USA, June 12 - 15, 2011*, ACM International Conference Proceeding Series, páginas 282–291. Digital Government Research Center, 2011.

[PL09] Roger S Pressman y David Brian Lowe. *Web engineering: a practitioner's approach*. McGraw-Hill Higher Education, 2009.

[Pon12] Isabel Ponce. Monográfico: redes sociales. *Internet web*, 2, 2012.

[PSB+13] Soledad Pellicer, Guadalupe Santa, Andres L Bleda, Rafael Maestre, Antonio J Jara, y A Gomez Skarmeta. A Global Perspective of Smart Cities: A Survey. En *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, páginas 439–444. IEEE, 2013.

[RWE13] Ian Robinson, Jim Webber, y Emil Eifrem. *Graph databases*. .º'Reilly Media, Inc.", 2013.

[S+15] IAB Spain et al. VI Estudio anual redes sociales. *Recuperado de: http://www. iabspain. net/wp-content/uploads/downloads/2015/01/Estudio_-Anual_Redes_Sociales_2015. pdf*, 2015.

[San89] Félix Requena Santos. El concepto de red social. *Reis*, páginas 137–152, 1989.

[Sun13] Jiajun Sun. Incentive mechanisms for mobile crowd sensing: Current states and challenges of work. *arXiv preprint arXiv:1310.8364*, 2013.

[Tea15] The Neo4J Team. *The Neo4J Manual*. Neo Technology, edición 2.2, 2015.

[Twi15] Inc. Twitter. Política de Privacidad de Twitter. *Recuperado de:https://twitter.com/privacy?lang=en*, 2015.

[Web12] Jim Webber. A Programmatic Introduction to Neo4J. En *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '12, páginas 217–218, New York, NY, USA, 2012. ACM. url: `http://doi.acm.org/10.1145/2384716.2384777`.

[Zub14] Mohammad Zuber. A Survey of Data Mining Techniques for Social Network Analysis. *International Journal of Research in Computer Engineering & Electronics*, 3(6), 2014.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **esi-tfg** que se puede encontrar en:
`https://bitbucket.org/arco_group/esi-tfg`