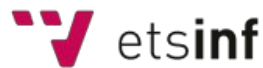




UNIVERSIDAD
POLITECNICA
DE VALENCIA



**DIPLOMA DE ESPECIALIZACIÓN EN BIG DATA
TRABAJO FIN DE DIPLOMA**

INTELIGENCIA SOCIAL DE NEGOCIO CON NEO4J

**Curso 2014-2015
1ª Edición**

**AUTOR
Andrés Ramos Pérez**

**DIRECTOR
Francisco Manuel Rangel Pardo**

Agradecimientos

A Kico Rangel por haber confiado en mí y haber hecho posible este proyecto

A Autoritas Consulting por acogerme en su equipo

A Pilar, mi mujer, por acompañarme en todas las aventuras

A mi familia por hacer posible que escriba estas líneas

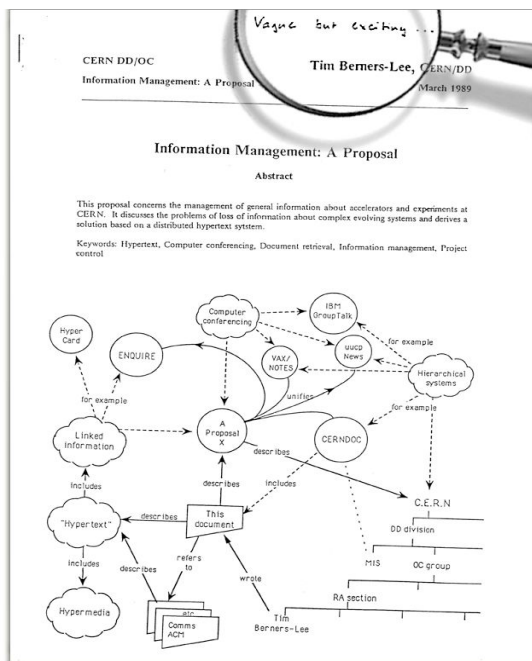
ÍNDICE

Introducción.....	3
Justificación.....	3
Objetivos.....	5
Estructura de la memoria.....	7
Marco metodológico.....	8
Fuentes de datos.....	8
Recuperación de la información.....	10
Almacenamiento y organización de la información.....	12
Dimensionando la información.....	13
Enriqueciendo la información.....	15
Marco tecnológico.....	17
MySQL.....	17
Limitaciones del modelo relacional en Big Data.....	17
Lucene.....	18
Limitaciones del modelo de índices de Lucene en la explotación de relaciones.....	19
Neo4J.....	19
Metodología para la generación de la base de datos de relaciones.....	21
Modelo relacional.....	21
Modelo de relaciones.....	22
Proceso de ETL.....	26
Análisis de prestaciones.....	28
Arquitectura distribuida.....	31
Explotación y visualización.....	34
Configurador del entorno de trabajo. Filtrado de la información.....	34
Preprocesado del grafo.....	41
Visualización de grafos.....	44
Conversación entre usuarios de Twitter.....	44
Coocurrencia de hashtags.....	46
Quién dice qué.....	47
Conclusiones.....	49
Bibliografía.....	50
Anexo A. Código fuente del proceso ETL.....	52

1. Introducción

1.1. Justificación

El concepto de Internet que a día de hoy tenemos dista mucho de sus orígenes. Inicialmente, la información existente en Internet estaba generada por profesionales expertos en la materia (programadores, administradores de sistemas, etc) que desarrollaban sitios web estáticos cuya información era invariable en el tiempo salvo que la



persona encargada de administrar la página la modificara. Este modelo de Internet, conocido como Web 1.0, fue evolucionando poco a poco. Las páginas web comenzaron a permitir la modificación de las mismas mediante gestores de contenido, de manera que el papel de la persona experta ya no era imprescindible cuando se deseaba realizar cambios en la web. Esta evolución bidireccional de la información, conocida como Web 2.0¹ fue ganando usuarios debido a que éstos se sentían partícipes al ver que podían interactuar con otras personas y dejar plasmadas sus inquietudes y comentarios.

Figura 1. Proyecto WWW de Tim Berners Lee.

La proliferación de usuarios en la web 2.0 fue el detonante del nacimiento de las redes sociales. Se supo materializar las necesidades de los usuarios por plasmar en un medio digital sus opiniones, anécdotas, estado de ánimo, etc y sobre todo, compartirlas con los demás. El usuario deja de ser solo un consumidor de información para convertirse en un generador de la misma. Nace el concepto de *prosumer*².

Internet se ha convertido en la mayor fuente de información debido a la gran cantidad de usuarios que lo componen. La diversidad de orígenes distintos convierte esta masa de información en datos desestructurados. Encontrar y compartir información útil en Internet se convierte cada vez en una tarea más complicada. Este reto se convierte en una gran

¹ Término definido por Dale Dougherty de O'Reilly Media en una tormenta de ideas con Craig Cline de MediaLive para la conferencia organizada en Octubre del 2004, bajo el lema *La Web como plataforma*.

² <https://en.wikipedia.org/wiki/Prosumer>

oportunidad para el mundo empresarial, que está aprovechando la proliferación de nuevas tecnologías dedicadas a explotar grandes volúmenes de datos para crear nuevos modelos de negocio. El proceso mediante el cual una empresa es capaz de tomar decisiones en base al análisis de información proveniente de este tipo de fuentes se conoce como inteligencia social de negocio.

Autoritas Consulting³ intenta aprovechar las innumerables ventajas que proporciona este nuevo paradigma de comunicación. Ha centrado su actividad en el sector de la consultoría estratégica generando inteligencia para las organizaciones a partir de datos sociales provenientes de fuentes abiertas. Su objetivo es conocer el entorno (macro y micro) en el cual opera cada cliente (su reputación, su competencia, la identificación de medios influyentes, la gestión de crisis,...) y así ayudar en la toma de decisiones con conocimiento de causa.

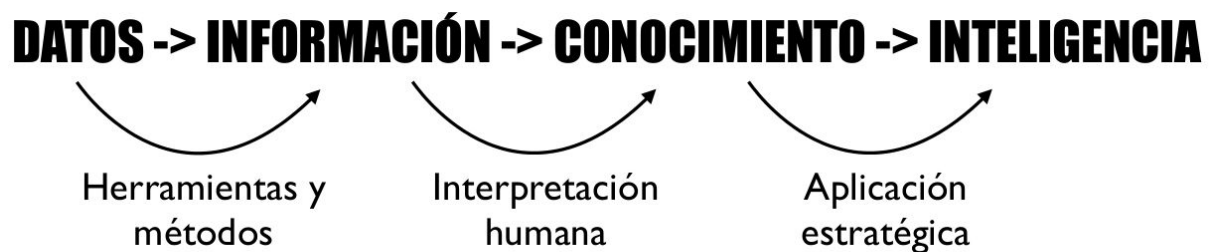


Figura 2. Etapas para la generación de inteligencia a partir de datos brutos.

Tanto en Internet como en redes sociales, los datos por sí solos no siempre aportan valor, además, dichos datos pueden/suelen contener ruido. La aplicación de una metodología y las herramientas de procesamiento adecuadas sobre los datos, convierte a éstos en información susceptible de valor. Podríamos definir información como el resultado de eliminar el ruido al conjunto de datos.

La interpretación de la información por parte de un analista genera conocimiento. Éste por sí solo no sirve de nada si no está alineado con la estrategia de la empresa y ayuda en la toma de decisiones. Cuando se da esta situación, se habla de inteligencia.

³ <http://www.autoritas.net>

En el presente trabajo se propone el desarrollo de una herramienta que explote un modelo de relaciones que facilite el análisis de la información para la generación de inteligencia social de negocio.

Es importante aclarar que el trabajo desarrollado en este proyecto se ha realizado dentro del entorno empresarial, y más concretamente, dentro de una empresa dedicada a la explotación de información proveniente de fuentes abiertas, Autoritas. Por ello, el presente documento se centrará en la solución específica correspondiente a la incorporación de un modelo de relaciones en su herramienta Cosmos Intelligence⁴, pero debe destacarse que dicho modelo podría ser extrapolado para aplicarse a cualquier otra herramienta para mejorar sus procesos de generación de inteligencia a partir del análisis de las relaciones.

1.2. Objetivos

La finalidad de este proyecto es construir un **modelo de relaciones** basado en la red social Twitter y una **herramienta** que permita la explotación del mismo. El modelo de relaciones se construirá en Neo4J⁵, mientras que la herramienta serán portlets de Liferay⁶ desarrollados en Java que se instalarán en la aplicación Cosmos Intelligence.

Para llevar a cabo esta tarea se generará una base de datos de relaciones implementada en Neo4J. Se construirá a partir de los documentos almacenados en el datawarehouse de Autoritas. Para ello se elaborará un proceso ETL que transforme una estructura de datos relacional clásica en una orientada a grafos.

El proceso ETL creará una base de datos en Neo4J para cada uno de los proyectos en los que se trabaja en Autoritas. Cada una de las bases de datos estará formada por nodos de diversa índole, siendo los principales los nodos de tipo **documento** e **influenciador**. De éstos saldrán relaciones hacia otros nodos que complementan su existencia.

El datawarehouse con el que se trabajará está formado por millones de documentos por lo que nos enfrentamos a un problema donde deben considerarse los factores tiempo y uso de máquina. Abordar la tarea con un proceso ETL secuencial no resulta viable debido al tiempo que requeriría, mientras que un ETL paralelo podría colapsar las bases de datos origen.

⁴ <http://www.cosmos-intelligence.com>

⁵ <http://neo4j.com>

⁶ <https://www.liferay.com>

Estas consideraciones se convierten en los puntos clave que garantizan la viabilidad del proyecto.

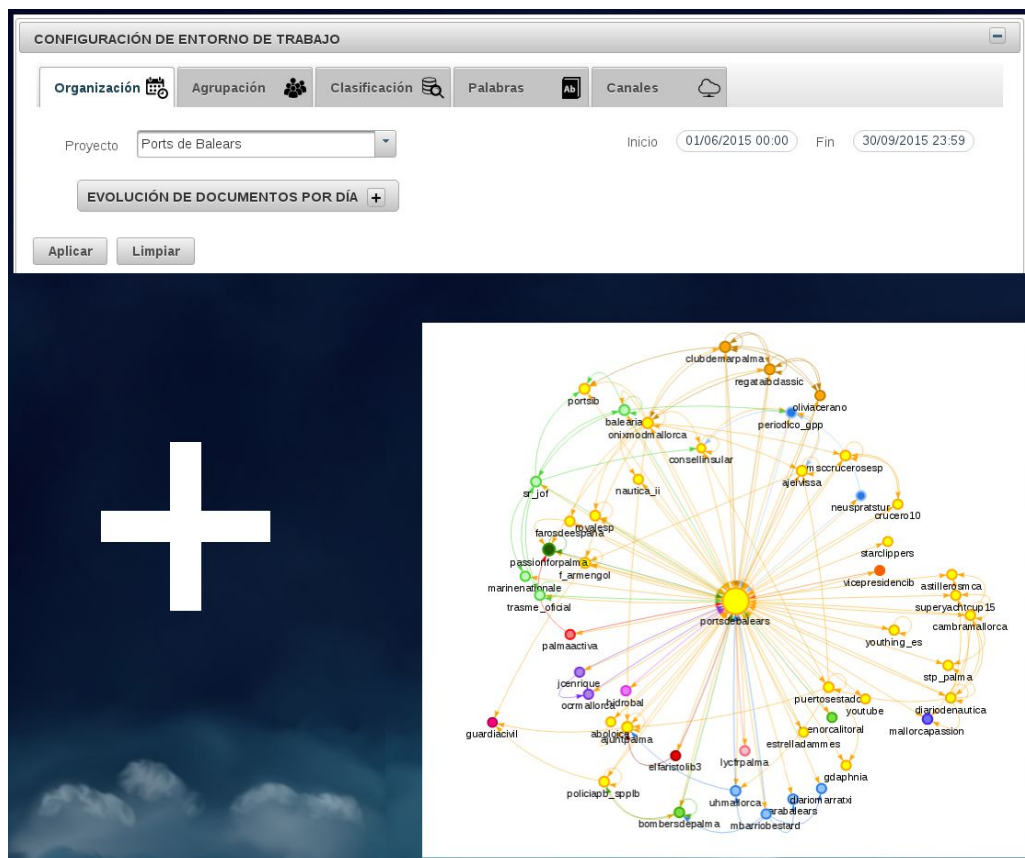


Figura 3. Portlets Configurator del Entorno de Trabajo + Grafo de relaciones.

Una vez construidas las bases de datos en Neo4J, éstas se integrarán en la infraestructura de Cosmos Intelligence. El analista de la herramienta dispondrá de un conjunto de filtros de búsqueda, organizados y agrupados según su finalidad. Éstos se integrarán en unos portlets desarrollados en Java que se instalará en la aplicación Web Cosmos Intelligence.

Antes del desarrollo de este proyecto, las consultas lanzadas por los usuarios de la herramienta iban contra bases de datos MySQL e índices de Lucene. A partir de ahora, las consultas se dirigirán a las bases de datos Neo4J, pero para el usuario será algo transparente. El cambio lo percibirá en la creación de una nueva interfaz de explotación visual de los grafos, lo que le permitirá conocer cómo se relacionan los usuarios y documentos entorno a temáticas.

Una vez generados cada uno de los grafos, éstos podrán ser explotados mediante consultas en lenguaje Cypher. Esta parte cierra el ciclo completo del modelo de inteligencia social de negocio. La explotación de grafos no es objeto de este proyecto y se describe en [13].

1.3. Estructura de la memoria

Este trabajo se ha estructurado en cuatro apartados que engloban todo el proceso necesario para llevar a cabo la creación de un modelo de relaciones en Neo4J y una herramienta que permita su explotación. Estos apartados son:

- **Marco metodológico:** en este apartado se presentan las fuentes de datos de las cuales se nutre la aplicación Cosmos Intelligence, así como el proceso de recuperación, clasificación, enriquecimiento y almacenamiento de dicha información en los diferentes sistemas de almacenamiento.
- **Marco tecnológico:** aquí se describen las fortalezas y debilidades de MySQL, Lucene y Neo4J. Se enumeran las carencias de MySQL y cómo son paliadas por Lucene. A su vez, Lucene posee puntos débiles que con la combinación de Neo4J se complementan para formar un motor de búsqueda capaz de explotar al máximo la información recuperada de Internet.
- **Metodología para la generación de la base de datos de relaciones:** este capítulo describe el modelo de relaciones que se desea alcanzar y el proceso ETL que debe llevarse a cabo para conseguirlo.
- **Explotación y visualización:** por último, se muestran las posibilidades de explotación disponibles al combinar un modelo de relaciones en Neo4J junto con un filtro de búsqueda desarrollado a conciencia para obtener resultados combinados de MySQL, Lucene y Neo4J.

Finalmente se comentan las conclusiones del trabajo así como se discuten posibles vías de trabajo futuro.

2. Marco metodológico

En este apartado se proporciona el marco metodológico sobre el que se desarrolla la herramienta Cosmos y que proporciona la base sobre la que integrar la tecnología propuesta. Concretamente se trata el tema de las fuentes de datos, la recuperación y almacenamiento de la información, su dimensionamiento y enriquecimiento.

2.1. Fuentes de datos

La inteligencia social de negocio se basa en el tratamiento de información recuperada de fuentes de datos abiertas u OSINT (“Open Source Intelligence”). OSINT incluye una amplia variedad de información y fuentes de datos, como por ejemplo:

- **Media:** periódicos, revistas, televisión, radio, etc.
- **Webs basadas en comunidades y contenido generado por usuarios (*redes sociales*):** Facebook, Twitter, Instagram, YouTube, wikis, blogs, etc.
- **Datos públicos:** informes de gobiernos, datos demográficos, debates legislativos, concursos públicos, etc.
- **Datos procedentes de observatorios:** satélites, radio monitores, estaciones climáticas, etc. La disponibilidad de información captada desde satélites como por ejemplo Google Earth, ha permitido el acceso a datos abiertos que unos años atrás únicamente estaban disponibles en grandes organizaciones.
- **Profesionales y académicos:** conferencias, papers académicos, simposios, publicaciones de expertos en la materia, etc.
- **Geoespacial:** atlas, mapas, planos, datos de urbanismo, espaciales, aeronáuticos, etc.

Dada esta taxonomía de fuentes de información, los datos primordiales en la inteligencia social de negocio son los procedentes de redes sociales y otras fuentes de internet como las noticias o los blogs. Las referencias de un usuario hacia otro, las etiquetas en los comentarios y los enlaces a sitios Web que un usuario emite en sus comentarios, permiten elaborar un grafo de relaciones donde se observa la agrupación de los usuarios entorno a temas específicos, personas, ideologías, etc.

El foco de este proyecto se centra en la red social Twitter. En determinados apartados de este documento se podrá hacer referencia a otras fuentes por claridad y de manera comparativa.

2.2. Recuperación de la información

Cuando se desea obtener datos de fuentes abiertas deben considerarse dos aspectos fundamentales: **¿Qué buscamos?** y **¿Cómo lo obtenemos?**.

El **¿Qué buscamos?** hace referencia a la información en sí misma que nos interesa traernos de Internet. Disponemos de tres técnicas de recuperación de datos provenientes de Internet:

- Por búsqueda.
- Por cuenta.
- Por geografía.

Cuando trabajamos en **búsqueda**, lo que hacemos es obtener información que coincida con uno o un grupo de términos. Ejemplo: obtener Tweets donde aparezcan las palabras *fútbol* y *baloncesto*.

En determinadas ocasiones el interés se centra en un sentido más amplio. En lugar de buscar documentos concretos preferimos trabajar con toda la información emitida por una persona u organización. En este caso se realizan búsquedas por **cuenta**, como por ejemplo obtener todo lo que hable *@el_pais* en Twitter.

El último tipo de búsquedas serían aquellas que se realizan por **geografía**. Este criterio de búsqueda va más allá de unas coordenadas de localización que se haría en Twitter. Cuando se obtienen todas las noticias emitidas por periódicos valencianos también se está trabajando a nivel de geografía.

Una vez definidos los criterios de búsqueda nos centramos en **¿Cómo lo obtenemos?**. Este proceso se realiza mediante dos tipos de técnicas:

- API.
- Motor de búsqueda propio [6].

Las redes sociales ponen a disposición de los desarrolladores un conjunto de métodos que les permiten obtener la información almacenada en sus bases de datos. Gracias a estas funciones es posible utilizar datos externos en una aplicación propia sin necesidad de preocuparse por su almacenamiento. Este conjunto de funciones se conocen con el nombre de **API**.

Pero no todos los sitios Web tienen definida una API para poder explotar sus datos. Cuando hablamos de periódicos digitales, blogs, foros, etc, lo único que suele ofrecer la Web en cuestión se limita a un RSS. En estos casos necesitamos desarrollar un **motor de búsqueda propio** que supla las carencias de la inexistencia de una API externa.

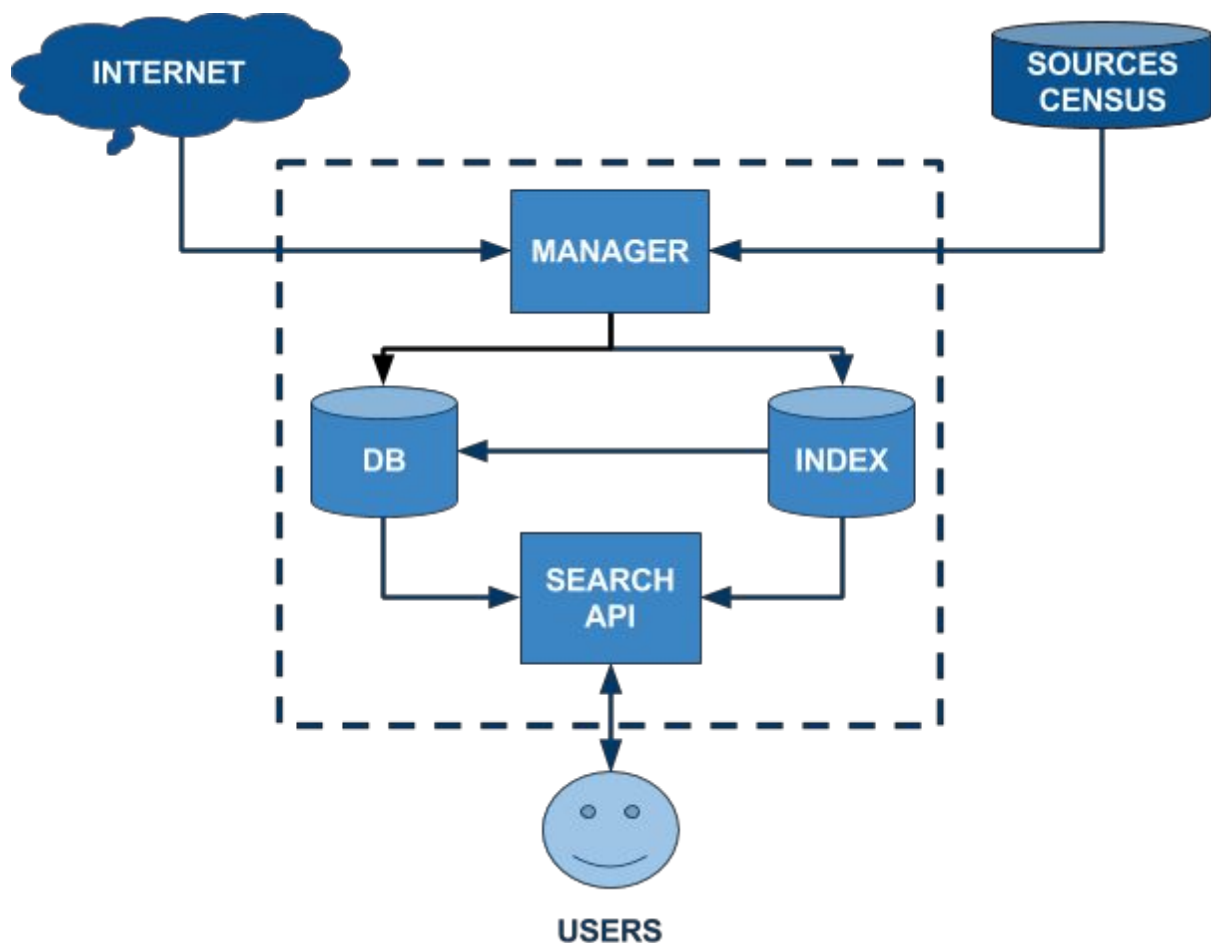


Figura 6. Arquitectura de un motor de búsqueda propio.

Un motor de búsqueda propio está compuesto por los siguientes elementos (Figura 6):

- **Censo de fuentes de información:** lista de sitios Web de los cuales queremos descargar sus publicaciones.

- **Gestor:** parte del motor de búsqueda que se encarga de traer la información de cada uno de los sitios Web del censo y almacenarlo en la base de datos e índice.
- **Base de datos:** lugar donde se almacena la información una vez ha sido procesada.
- **Índice:** apoyo a la base de datos para realizar búsquedas textuales y obtener los identificadores de los documentos que permitirán posteriormente acceder a la base de datos. De esta forma el acceso al dato es más rápido.
- **API:** conjunto de funciones que permiten acceder a la información a los usuarios.

2.3. Almacenamiento y organización de la información

La Figura 7 ejemplifica cómo está estructurada la herramienta Cosmos. En un primer nivel en azul se encuentran los diferentes proyectos. Cada proyecto recupera información de los diferentes canales que hemos comentado al principio de este punto. Dependiendo del tipo de canal podemos encontrarnos a su vez con varios subcanales. Dentro de cada canal/subcanal se recogen las diferentes fuentes de información que generan contenido sobre el proyecto en cuestión. Ésta variará en función del canal donde nos encontremos ya que cada uno ofrece unas posibilidades de explotación diferentes.

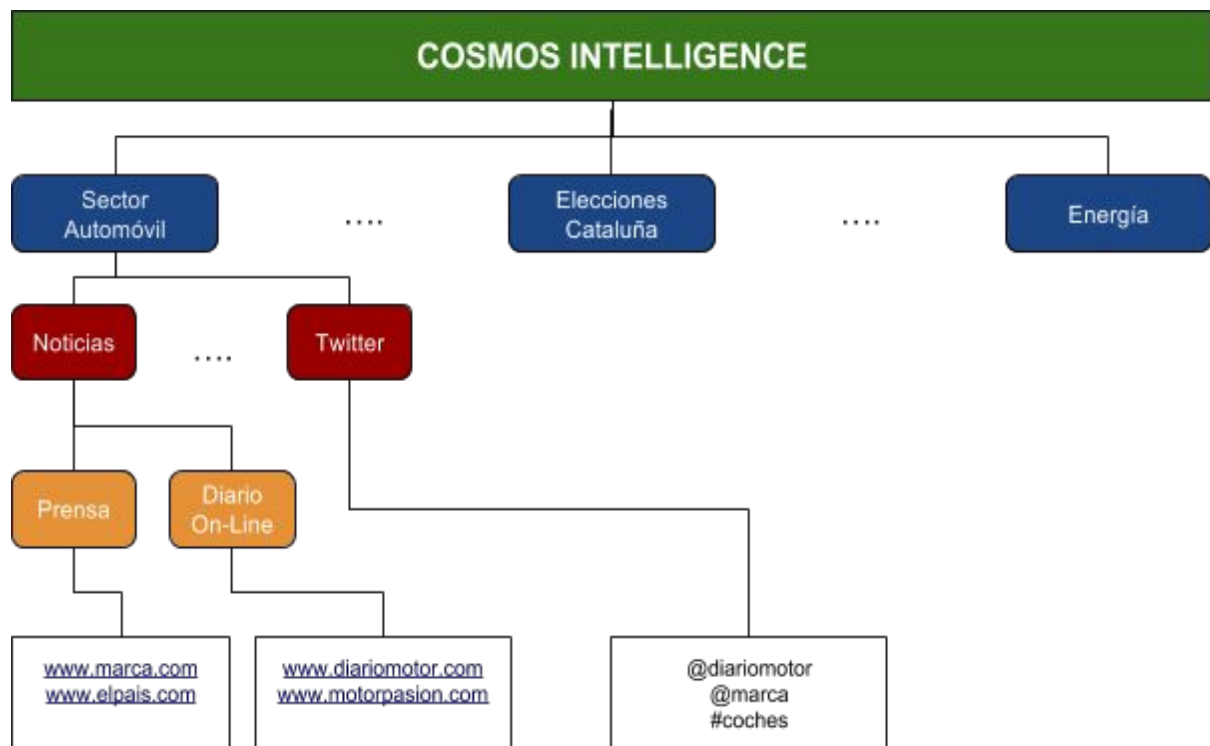


Figura 7. Estructura organizativa de Cosmos Intelligence.

Debido a la diversidad de los tipos de datos (noticia, tweet, comentario en foro, etc), es obligatorio definir una nueva estructura que permita abstraerse del origen y compararlas por igual. Identificamos como **documento** al texto extraído de cualquiera de los canales anteriores, y como **influenciador** [1] al autor del mismo⁷. Toda la información relativa a un documento y su influenciador se almacenan en un datawarehouse plano consistente en tecnologías MySql, Lucene y Amazon S3.

2.4. Dimensionando la información

Una vez recuperada y almacenada la información, se debe procesar, ordenar y etiquetar de modo que se facilite el posterior análisis de la misma. Cuando este procesamiento se alinea con los objetivos estratégicos de la organización en cuestión, se le conoce como **dimensionamiento estratégico**.

El dimensionamiento estratégico consiste en una serie de etiquetas agrupadas en categorías que tematizan el contenido de los documentos. Una etiqueta es asignada a un documento a partir de su tesauro asociado, entendido éste como conjunto de reglas lógicas que permiten la combinación de palabras disparadoras cuya aparición provoca el etiquetado del documento en la etiqueta correspondiente. Las etiquetas a su vez se agrupan bajo categorías, permitiéndose que una misma etiqueta pertenezca a más de una categoría.

En la Figura 8 se muestra el proceso de dimensionar estratégicamente la información. Se parte de los documentos recuperados según el método elegido (descrito en la sección 2.2) y se aplica un procesamiento de los mismos para asignar las etiquetas, y por ende las categorías, correspondientes, dimensionando de este modo los documentos y permitiendo un análisis tematizado de los mismos alineados con la estrategia de la organización.

⁷ Puede ser una organización (Ej: El País) o un usuario (Ej: @andresramper).

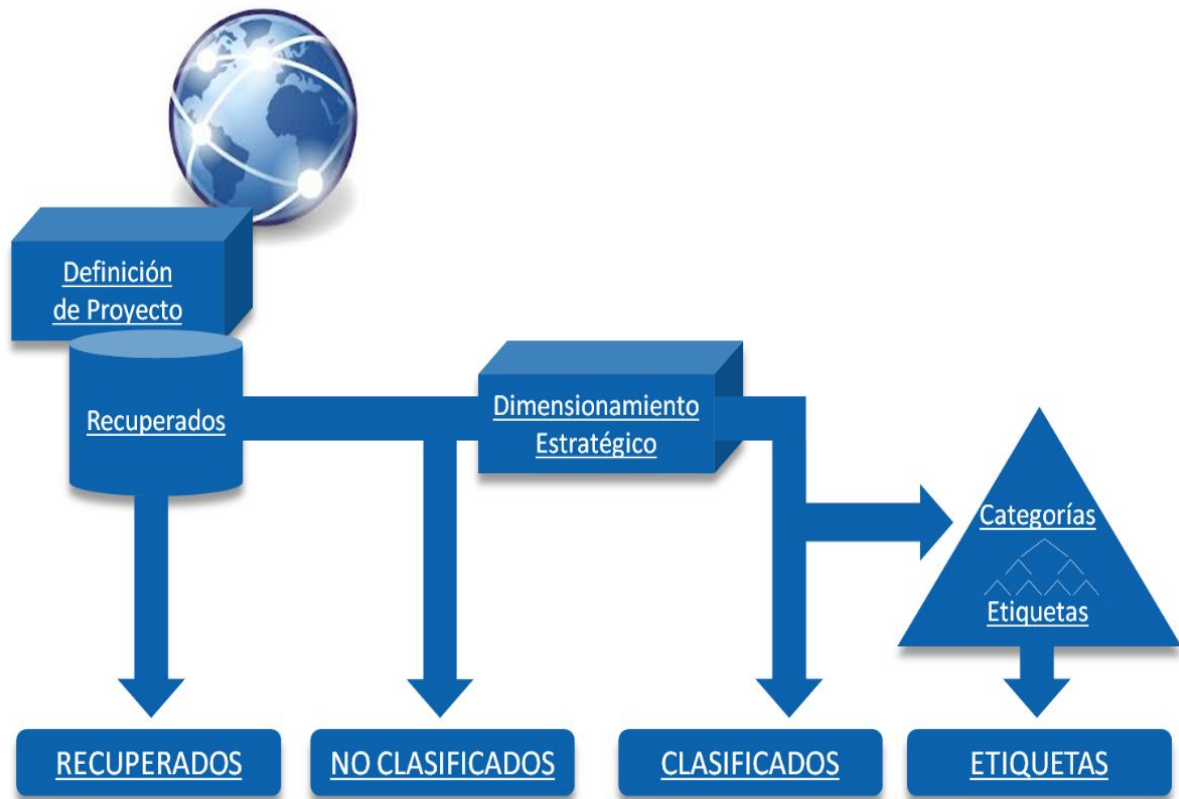


Figura 8. Dimensionamiento estratégico.

El dimensionamiento estratégico se compone de un mix de dimensiones comunes (Ej. valores, personas, atributos de marca, etc.) propuestas por la metodología y dependientes del sector y del tipo de proyecto, y dimensiones específicas que son definidas por el analista propietario del proyecto en cuestión (Ej. tipos de cerveza, partidos políticos, puertos, etc.).

La Figura 9 muestra un ejemplo de dimensionamiento estratégico para un proyecto sobre el *Canal de Panamá*. En él se pueden apreciar categorías como *valores* cuyas etiquetas cuantifican la percepción de conceptos como *transparencia*, *responsabilidad*, *lealtad*, etcétera.

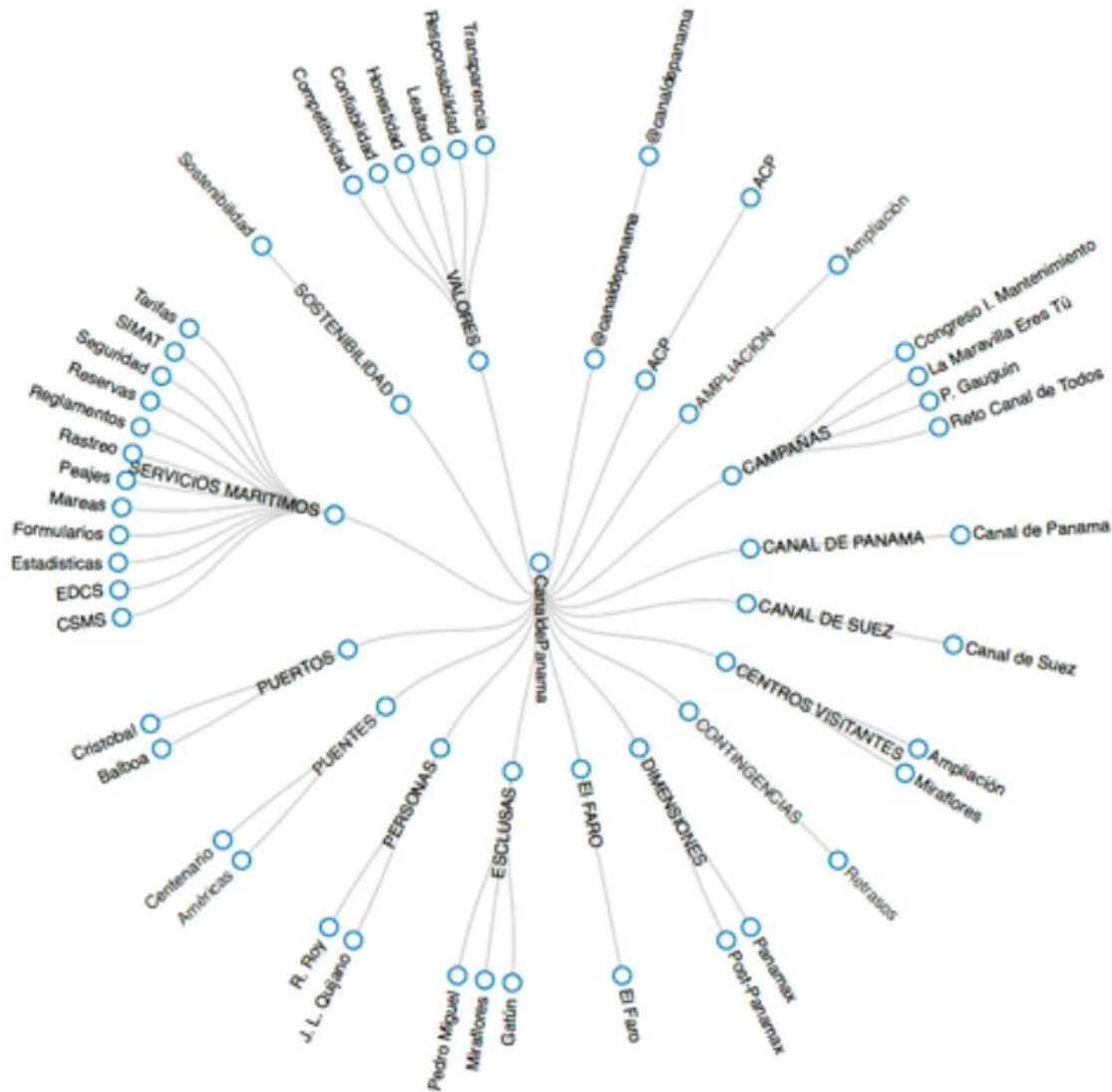


Figura 9. Categorías y etiquetas del proyecto Canal de Panamá.

2.5. Enriqueciendo la información

Tras la recuperación, almacenamiento y dimensionamiento de la información, se procede a enriquecer la misma con una serie de técnicas de procesamiento del lenguaje natural. Algunas de las técnicas utilizadas son las siguientes:

- *Demografía (edad y sexo).*

Una de las líneas de trabajo del author profiling se encarga de conocer rasgos del autor, como su edad y su sexo [17, 18, 19, 20]. Estas técnicas son muy importantes en temas relacionados con la seguridad o fines comerciales, ya que permiten respectivamente

detectar delincuentes en comunidades o enfocar de manera más eficiente campañas de marketing⁸.

- *Afectividad (sentimiento y emociones).*

El objetivo es determinar cómo está expresando un actor el contenido, en qué tonalidad [12] y con qué emotividad [15, 16]. Con la proliferación del número de usuarios en redes sociales el aspecto emocional ha cobrado gran importancia y con ello, han surgido técnicas de procesamiento que buscan conocer lo que los usuarios quieren y necesitan, de modo que se pueda satisfacer mejor su demanda desde el aspecto emocional y no sólo de producto.

- *Geolocalización (idioma y variedad regional del idioma).*

Esta técnica se encarga de la identificación de la región a la cual puede pertenecer un usuario en función del idioma utilizado (identificación de idioma) y cómo lo usa (identificación de variedad o dialecto) [7, 9, 10].

Dentro de esta técnica encontramos tres líneas de trabajo:

- NLI (Native Language Identification): por ejemplo, gente de diferentes nacionalidades hablando inglés.
- LVI (Language Varieties or Dialects Identification): por ejemplo, portugués de Portugal vs. Brasil, ó español de España vs. Argentina vs. Colombia, etc.
- LI (Language Identification) [21]: por ejemplo, distinguir inglés del español.



Figura 10. Técnicas de procesamiento del lenguaje.

Otras líneas de trabajo que enriquecen la información serían la extracción de entidades nombradas como nombres de personas, organizaciones, productos, localizaciones geográficas, etcétera, o la identificación de tendencias [14, 22] entre otras muchas que no se exponen aquí por quedar fuera del ámbito del proyecto y no aportar nada extra al modelo que se presenta.

⁸ <http://www.upv.es/noticias-upv/noticia-7797-analisis-comput-es.html>

3. Marco tecnológico

En este apartado se tratan las tecnologías de almacenamiento utilizadas en cada momento y como aparecen limitaciones en las mismas que obligan a migrar a otro sistema que cubra las carencias de su predecesor.

Las tecnologías con las que se trabaja son: MySQL, Lucene y Neo4J. Explicaremos las limitaciones del modelo relacional y cómo son mejoradas y complementadas por Lucene, y concluimos detallando los problemas de Lucene a la hora de realizar explotación de relaciones y por lo tanto la introducción de Neo4J.

3.1. MySQL

MySQL es un sistema de bases de datos relacional orientado a objetos, multihilo y multiusuario, creado por la empresa MySQL AB⁹, la cual posee el copyright del código fuente del servidor SQL, así como de la marca.

Aunque este software es libre debido a que está distribuido bajo la licencia GPL de la GNU, la compañía MySQL AB distribuye una versión comercial de MySQL, diferenciada únicamente de la versión libre en la cobertura del soporte técnico, y en la posibilidad de integrar esta base de datos en software propietario, ya que si no se estaría vulnerando la licencia GPL.

Se trata de la base de datos de software libre más usada en todo el mundo, debido a su gran rapidez y facilidad de uso. Este apogeo en lo más alto es debido a la existencia de infinidad de librerías y demás herramientas que permiten utilizar MySQL a través de numerosos lenguajes de programación. Además de esto, también ha ayudado su fácil instalación y configuración.

3.1.1. Limitaciones del modelo relacional en Big Data

Hasta la aparición del fenómeno Big Data, las bases de datos relacionales estaban consideradas como la solución más apropiada a la gran parte de los problemas de almacenamiento de datos. Están presentes en la gran mayoría de organizaciones porque entre sus fortalezas destaca el manejo de las operaciones de actualización, sobre todo en los entornos donde la consistencia de la información es un elemento clave.

⁹ <https://www.mysql.com>

Sin embargo, cuando trabajamos con Big Data, este tipo de bases de datos presentan carencias. En primer lugar, no es posible desplegar clusters dinámicamente que permitan escalar fácilmente el sistema. Una segunda debilidad es el tratamiento de datos desestructurados. Cuando se trata información proveniente de redes sociales, blogs, etc no es posible asegurar la estructura que tendrán los datos por lo que dificulta el trabajo con bases de datos relacionales. Además, cuando el usuario tiene libertad de realizar búsquedas textuales en elementos desestructurados donde el volumen de datos es muy elevado, esta tarea se convierte en un proceso crítico.

Por último, con este tipo de sistemas es difícil realizar consultas complejas donde se obtenga, por ejemplo, la distancia más corta entre una pareja de palabras. Además, hay un problema de rendimiento cuando se realizan inserciones múltiples, por ejemplo, insertando a la vez miles de tuits por minuto mientras se consultan las tablas.

En resumen, existe una limitación a la hora de obtener información de relaciones complejas. Por ejemplo, si tenemos una tabla con documentos, otra con etiquetas y una intermedia que las relaciona, sería prohibitivo intentar obtener: dada una etiqueta A, decir qué otras etiquetas tienen los documentos que comparten la etiqueta A.

3.2. Lucene

Con el objetivo de paliar algunas carencias de las bases de datos relacionales referentes a búsquedas complejas y tratamiento de información desestructurada aparece Lucene¹⁰. Es una API de código abierto desarrollada en Java aunque incluye versiones para otros lenguajes: Delphi, Perl, C#, C++, Python, Ruby y PHP.

Lucene es apropiado para utilizar en aplicaciones donde sea necesario indexar y buscar en textos completos. Es una tecnología muy recurrida en la implementación de motores de búsqueda y muy competitiva con respecto a tecnologías propietarias como Oracle [5].

Su forma de trabajo basada en el concepto de documento formado por campos de texto permite abstraerse del formato de fichero y ser igualmente útil para trabajar con textos en

¹⁰ <https://lucene.apache.org/core>

formato HTML, Word, PDF, etc. Siempre que pueda extraerse información de ellos el formato es indiferente.

3.2.1. Limitaciones del modelo de índices de Lucene en la explotación de relaciones

Cuando se desea utilizar Lucene para otros fines diferentes a la indexación de textos, como por ejemplo, relacionar dos personas, ordenar los resultados de salida, trabajar únicamente con una parte de los documentos, etc se vuelve ineficiente y complejo.

Resulta complicado asimilar cómo puede ser más rápido devolviendo una colección de 100K documentos en lugar de los 10 primeros o, por ejemplo, el elevado coste de tiempo que supone realizar un ORDER BY del lenguaje SQL.

3.3. Neo4J

Neo4J es una base de datos orientada a grafos desarrollada en lenguaje Java por la startup americana Neo Technology¹¹. Está licenciada con un modelo dual, tanto por la AGPL v3 como bajo licencia comercial.

Neo4J ha sido implementado en Java. Está clasificado como un motor de persistencia embebido, basado en disco, completamente transaccional que almacena datos estructurados en grafos.

Un grafo es un conjunto de objetos llamados nodos que se conectan entre sí mediante relaciones binarias llamadas aristas o grafos. Una definición matemática de grafo sería:

Un grafo G es un par ordenado $G = (V, E)$, donde:

V es un conjunto de vértices o nodos, y

E es un conjunto de aristas o arcos, que relacionan estos nodos.

Este tipo de base de datos es apropiada, por ejemplo, para modelar redes sociales y sistemas de recomendación. Debe destacarse el uso de una API rest para realizar consultas a la base de datos lo cual permite integrarse con aplicaciones Web.

¹¹ <http://neo4j.com>

Incorporar una tecnología de grafos como Neo4J en una herramienta donde se explotan las relaciones entre usuarios, temas de conversación, comunidades, etc, permite por ejemplo, llegar al foco de una crisis y conocer los influenciadores que se comportan como nexo de unión entre grupos de usuarios. Neo4J es la tecnología más apropiada en el paradigma actual cuando se necesitan crear complejas consultas de acceso a base de datos donde deben realizarse varios saltos para llegar de un punto a otro.

4. Metodología para la generación de la base de datos de relaciones

4.1. Modelo relacional

Siguiendo el marco metodológico descrito en el apartado 2, la información es recuperada de fuentes abiertas (principalmente redes sociales y páginas web), almacenada, dimensionada y enriquecida. Los registros almacenados en el datawarehouse responden a una estructura común que incorpora una serie de campos comunes y otros específicos dependientes del canal del cual se recuperan. De manera específica para Twitter, cada campo podría estar agrupado en uno de los seis grupos conceptuales siguientes:

- Campos generales a todos los canales:
 - Campos de organización.
 - Campos temporales.
 - Campos comunes.
 - Campos de semántica.
- Campos específicos de Twitter:
 - Campos de Twitter procedente de su API.
 - Campos de Twitter procesados.

Campos de organización: a este grupo pertenecen aquellas variables que identifican el origen (canal, subcanal, etc) y destino (proyecto, tipo de subproyecto, subproyecto, agrupación y subagrupación) del documento recuperado.

Campos temporales: almacenan información relacionada con el factor tiempo como son la fecha de generación del documento, fecha de procesado, etc. en diferentes formatos para facilitar su explotación.

Campos comunes: como su nombre indica, la finalidad de estos campos es almacenar información que puede extraerse de los diferentes canales y subcanales. Un ejemplo de algunas variables son influenciador, título, contenido, url, idioma, ubicación geográfica, etc.

Campos de semántica: los campos con características propias del documento pertenecen a esta agrupación. Unos ejemplos de variables serían las categorías y etiquetas a las que pertenece el documento, la lista de palabras que han disparado el etiquetado del documento, etc.

Respecto a los campos específicos de twitter se dispone de los siguientes, algunos de ellos recuperados de la propia API de twitter y otros a partir de procesamiento del lenguaje como el indicado en el apartado 2.5.

Campos de Twitter procedentes de su API: influenciador, bio, localización, antigüedad, número de listas, número de usuarios, número de seguidores, etc.

Campos de Twitter procesados: aplicando técnicas de procesamiento del lenguaje se obtienen atributos como etiquetas, categorías, indicador de existencia de signos de exclamación, pregunta, hashtags, emoticonos, links, menciones, sexo del tuitero, etc.

4.2. Modelo de relaciones

Los componentes principales del modelo de relaciones definido son **documentos** e **influenciadores**. En el modelo de la Figura 11 se muestran algunas de las dimensiones que conceptualmente se pueden identificar en un proyecto de inteligencia social, donde las dos principales de las que parten todas las demás son el influenciador y el documento.

Un **documento** es el texto escrito por un autor, ya sea en forma de comentario en un blog, noticia, Tweet, comentario en Facebook, etc. Dicho documento posee atributos que lo describen y a su vez está relacionado con otros nodos como, por ejemplo, organización (proyecto donde está asociado el documento), etiqueta (tema del cual se habla en el documento), etc.

Influenciador, también conocido como usuario o autor, es el usuario que escribe un documento. Se identifica con el término influenciador (*influencer*) debido a su capacidad potencial de generar influencia por el simple hecho de pertenecer e interactuar en una red social mediante la expresión de sus opiniones en ella [1].

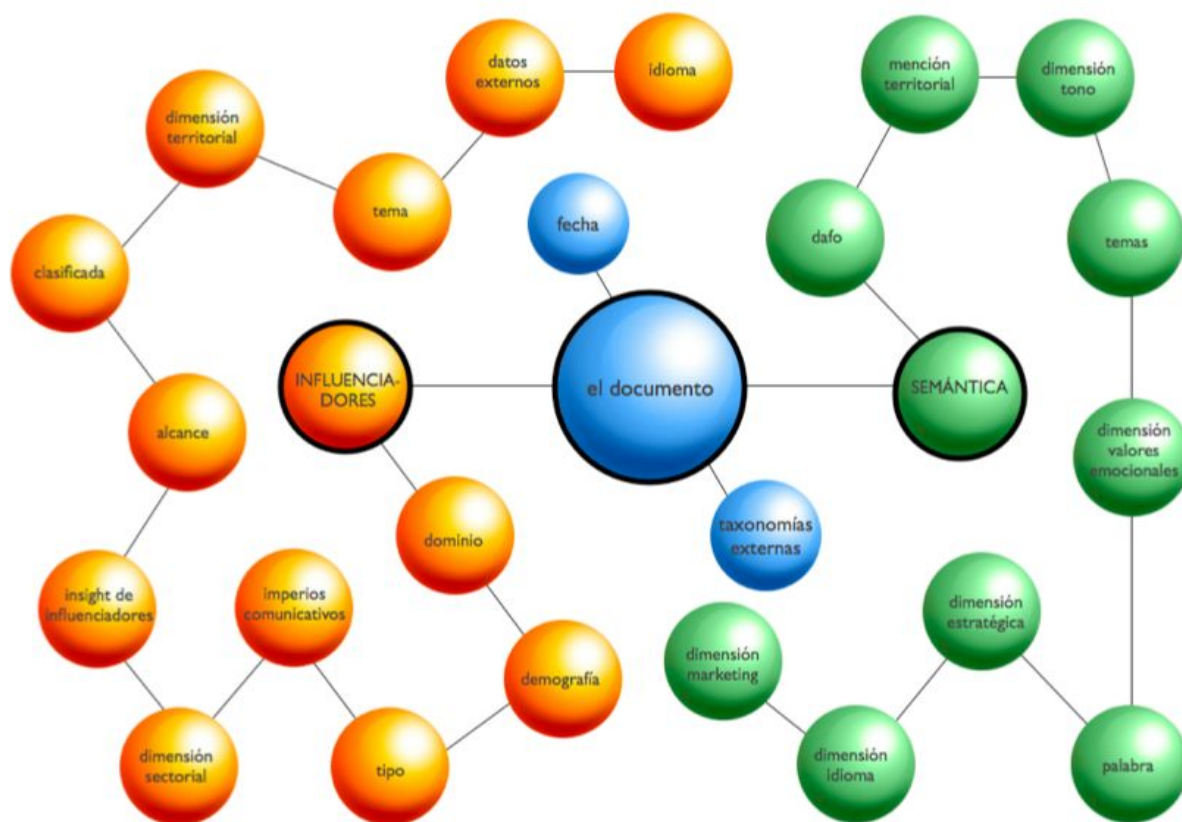


Figura 11. Modelo conceptual de dimensiones informativas.

El presente trabajo ha tomado como referencia el modelo descrito en [8], seleccionando el submodelo correspondiente a la red social Twitter. Su esquema se muestra en la Figura 12.

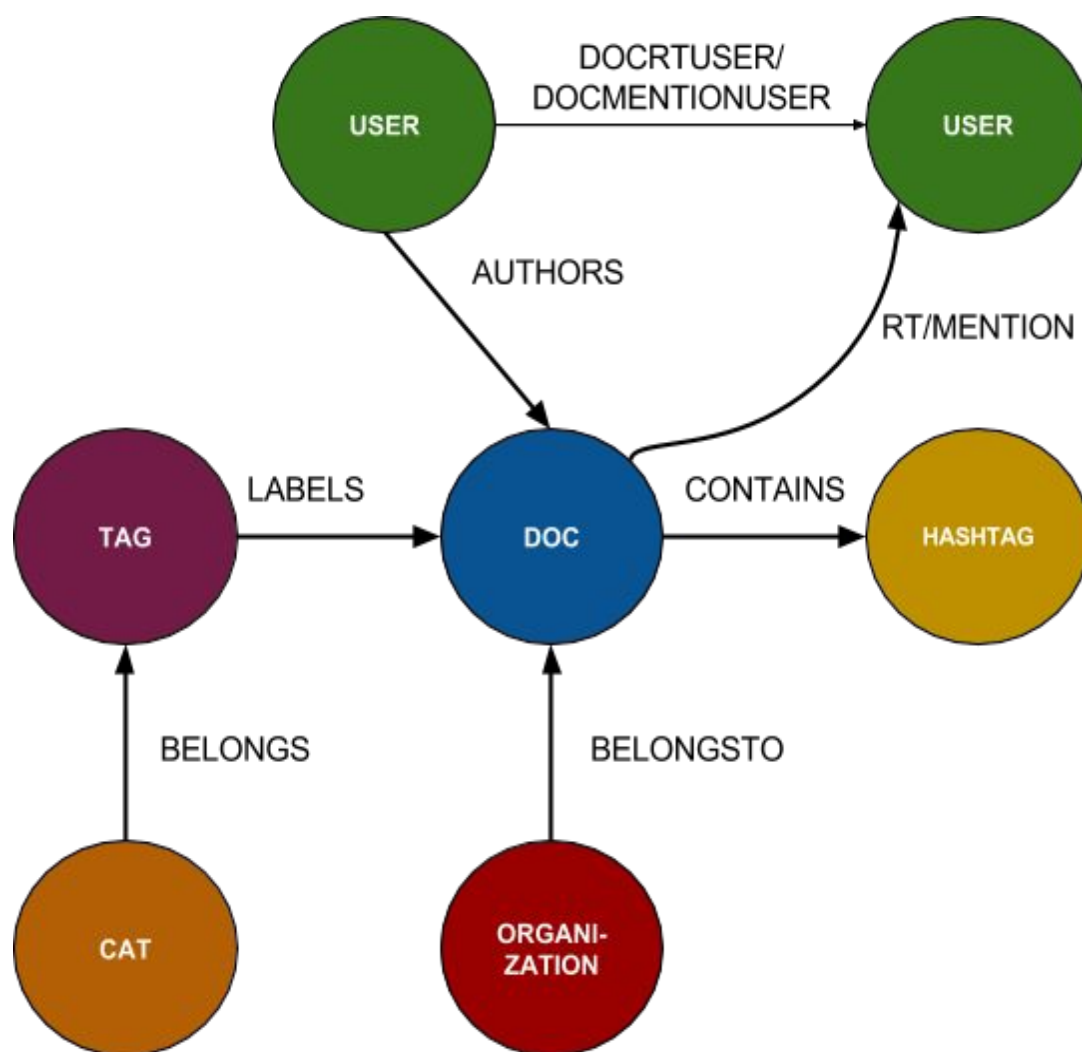


Figura 12. Modelo de relaciones para la red social Twitter.

Los nombres de cada uno de los nodos y su explicación se detalla en la Tabla 1.

Nodo	Explicación
User	Autor de un documento. Puede seguir a otro usuario y le pueden seguir otros usuarios.
Doc	Texto o contenido creado por un usuario.
Organization	Proyecto al cual pertenece un documento.
Hashtag	Etiqueta precedida por una almohadilla que facilita al usuario a identificar de forma rápida el contenido.
Tag	Etiqueta de Cosmos sobre la cual habla un documento.
Cat	Categoría a la cual pertenecen las diferentes etiquetas según su temática.

Tabla 1. Nodos del modelo relacional y explicación de su finalidad.

A continuación en la Tabla 2 se muestra el listado de los atributos asociados a cada uno de los nodos, así como el significado de ellos.

Nodo	Atributo	Significado
User	Antiquity	Antigüedad del usuario.
	Lists	Identificador de las listas del usuario.
	Friends	Usuarios a quien sigue el usuario.
	Followers	Usuarios que siguen al usuario.
	Gender	Sexo del usuario.
Doc	Rt	Indicador de si el documento es un retweet.
	Mention	Menciones del usuario a otros usuarios.
	Hashtag	Hashtag del documento.
	Question	Indicador de si el tweet es una pregunta.
	Exclamation	Indicador de si el tweet es una exclamación.
	Link	Links que contiene el documento.
	Datetime	Fecha en la que emitido el documento.
Organization	Subprjtype	Identificador del tipo de subproyecto (R,C ó I).
	Subprj	Nombre del subproyecto.
	Grouping	Agrupación de temas dentro de un subproyecto.
	Subgrouping	Subagrupación de temas dentro de un subproyecto.
Hashtag	<sin atributos>	
Tag	<sin atributos>	
Cat	<sin atributos>	

Tabla 2. Atributos de cada uno de los dos y su significado.

Por último, en la Tabla 3 se detalla el tipo de relaciones entre los nodos y la finalidad de las mismas.

Nodo origen	Nodo destino	Relación	Significado
user	doc	author	Usuario escribe documento
user	user	rt	Usuario retuitea a un usuario.
user	user	mention	Usuario menciona a un usuario.
organization	doc	belongsto	Documento pertenece a organización.
doc	hashtag	contains	Hashtag de documento.
doc	user	docmentionuser	Documento menciona a usuario.
doc	user	doctuser	Documento retuitea a un usuario.
tag	doc	labels	Etiqueta del documento.
tag	cat	belongs	Categoría del documento.

Tabla 3. Relaciones entre los nodos.

4.3. Proceso de ETL

El proceso ETL genera una base de datos de relaciones en Neo4J a partir de un índice de Lucene. Ha sido desarrollado en Java y realiza las funciones descritas en el Diagrama 1.

En primer lugar se crean los diferentes esquemas del modelo de relaciones. Éstos se corresponden con cada uno de los tipos de nodos: User, Document, Organization, Tag, Cat y Hashtag. A continuación comienza el proceso de creación de nodos y las relaciones entre ellos. En el momento de creación de un nodo se comprueba si ha sido creado anteriormente verificando su existencia a través de su índice.

Las operaciones involucradas en el proceso de creación de una base de datos Neo4J deben estar enmarcadas dentro de una transacción. Existen varias soluciones para llevar a cabo la generación pero no cualquiera es válida debido a los grandes volúmenes de información con los que se trabaja.

Una primera solución consiste en abrir una única transacción, crear el grafo, y cerrarla. Esta aproximación no es válida porque el gran consumo de memoria hace que falle el *garbage*

collector de Java y no termine la creación de la base de datos. Otra posible solución es crear una transacción por cada documento. Esta propuesta es muy lenta y hace que se multiplique el coste de manera prohibitiva.

La opción empírica que mejor ha funcionado, y por lo tanto la que se ha implementado en el proceso, ha consistido en abrir y cerrar la transacción cada cierto número de documentos procesados; concretamente cada 10.000 documentos.

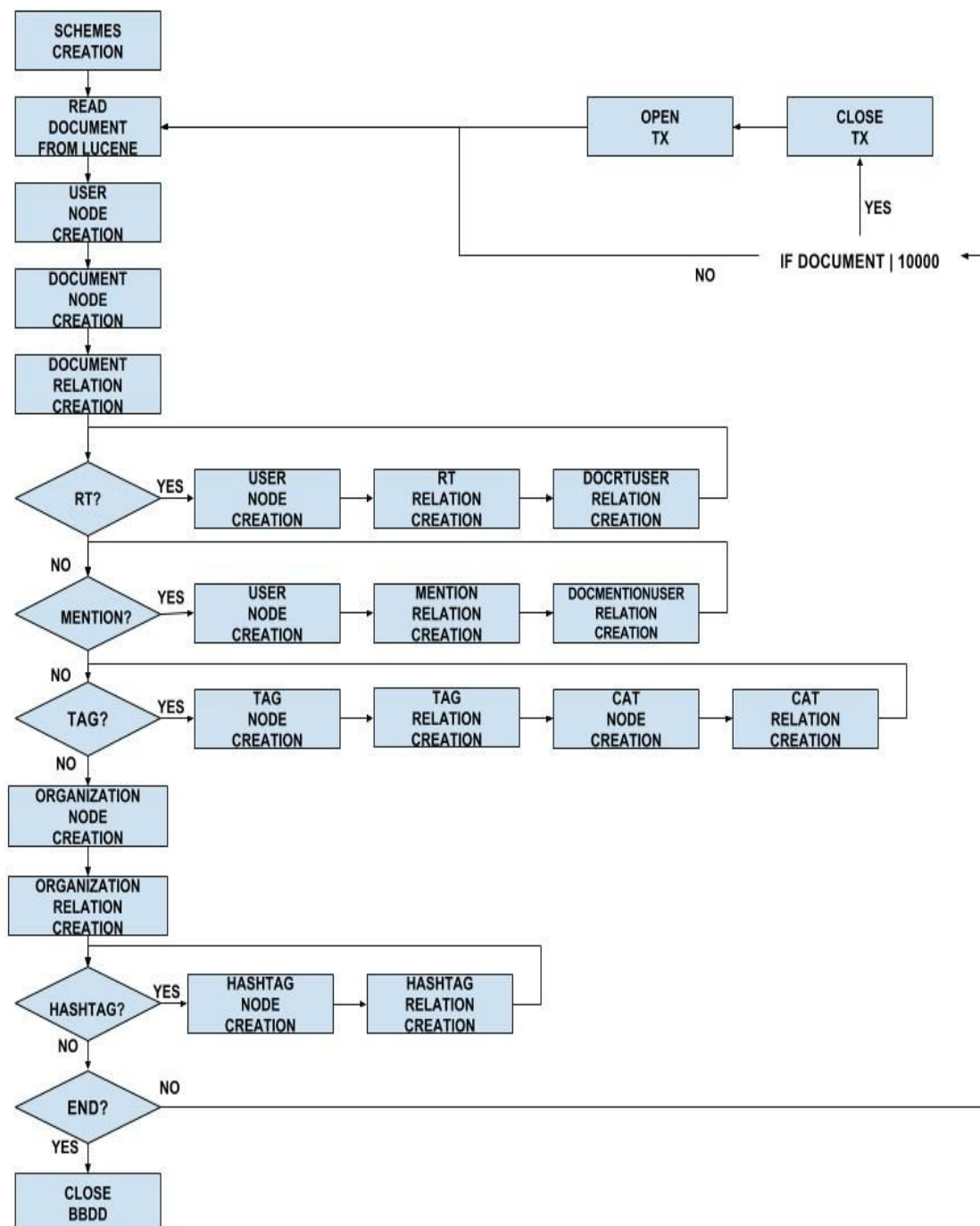


Diagrama 1. Diagrama de flujo del proceso ETL.

En el anexo A se proporciona el código fuente para su reproducibilidad.

4.4. Análisis de prestaciones

Con el objetivo de analizar el rendimiento se ha ejecutado el proceso de manera secuencial con varios proyectos de ejemplo. Las prestaciones de la instancia EC2 de AWS donde se han realizado las pruebas son:

- Modelo m3.xlarge con 4 CPUs, 15GB RAM y 2x40 GB de almacenamiento SSD.
- Procesadores Intel Xeon E5-2670 v2 (Ivy Bridge) de alta frecuencia.
- Almacenamiento de instancias basado en SSD para un rápido rendimiento de E/S.

Los parámetros medidos en la prueba han sido: número de documentos, número de influenciadores, número de hashtags, número de etiquetas, tamaño del índice Lucene, tamaño de la base de datos Neo4J generada, y por último, tiempo de ejecución del proceso.

PROYECTO	Nº DOC.	Nº INF.	Nº #	Nº ETIQ.	Gb Lucene	Gb Neo4J	Tiempo
Podemos	1.660.946	475.124	40.024	202	18	2,6	17:07
Elecciones Cataluña	6.288.008	67.7033	86.451	62	27	9,8	1:23:20
Canal Panamá	602.842	193.298	6.887	203	4,4	1,4	8:38
Turismo Balear	174.420	81.606	11.952	119	2,2	0,3	1:46
Sector Energía	2.842.252	1.165.875	81.104	96	32	5	35:28
Sector Banca	429.168	246.059	429.168	68	11	0,6	3:53
Cumbre Mundial Comunicación Política	375.647	171.953	171.953	352	6,6	0,8	4:31
Cerveza Escudo	1.890.735	1.269.857	95.974	96	13	3,3	22:29
Vinos de España	606.998	363.020	34.497	27	9,9	0,8	4:27
TDAH	351.356	208.716	3.528	83	2,2	0,5	4:10
TV Social	4406864	1407549	108358	186	29	7,6	1:01:09
Dilma Roussef	654.064	128.467	654.064	129	6,2	1,4	8:04

Tabla 4. Estadísticas y prestaciones del proceso ETL para diferentes proyectos.

En la Tabla 5 se ha tomado como unidad de medida el número de documentos almacenados en el índice de Lucene. Se ha escogido este parámetro porque cada uno de los documentos serán convertidos a un nodo en la base de datos de relaciones. Además de ello, un documento es el elemento que contiene la información que permite generar el resto de nodos. Por estos motivos es importante conseguir una alta tasa de procesamiento a nivel de documento, pues es lo que en un entorno de recuperación en tiempo real marcará las necesidades mínimas. Para calcular el número de documentos procesados por segundo se ha convertido la unidad de tiempo a segundos, obteniendo los resultados mostrados en la Tabla 5:

PROYECTO	Nº DOCUMENTOS	SEGUNDOS	DOCUMENTOS / SEGUNDO
Podemos	1.660.946	1.027	1.617,28
Elecciones Cataluña	6.288.008	5.000	1.257,60
Canal Panamá	602.842	518	1.163,79
Turismo Balear	174.420	106	1.645,47
Sector Energía	2.842.252	2.128	1.335,64
Sector Banca	429.168	233	1.841,92
Cumbre Mundial Comunicación Política	375.647	271	1.386,15
Cerveza Escudo	1.890.735	1.349	1.401,58
Vinos de España	606.998	267	2.273,40
TDAH	351.356	250	1.405,42
TV Social	4406864	3.669	1.201,11
Dilma Roussef	654.064	484	1.351,37

Tabla 5. Rendimiento del proceso ETL medido en documentos procesados por segundo.

Para obtener una visión significativa del tiempo de proceso medio esperable, en la Tabla 6 se han calculado los estadísticos básicos de media, mediana, desviación estándar, los cuartiles primero, tercero, el mínimo y el máximo.

Como puede observarse la mayoría de los valores se encuentran en torno a la media sin excesiva desviación, salvo en el caso del proyecto *Vinos de España* que procesa un número de documentos significativamente superior de documentos por segundo. Probablemente este proyecto posea una menor proporción entre documentos e influenciadores, es decir,

existen menos usuarios y además éstos escriben más tuits respecto a los usuarios de otros proyectos¹². Esto se traduce en una menor creación de nodos y relaciones, por lo que se pueden procesar más documentos por segundo.

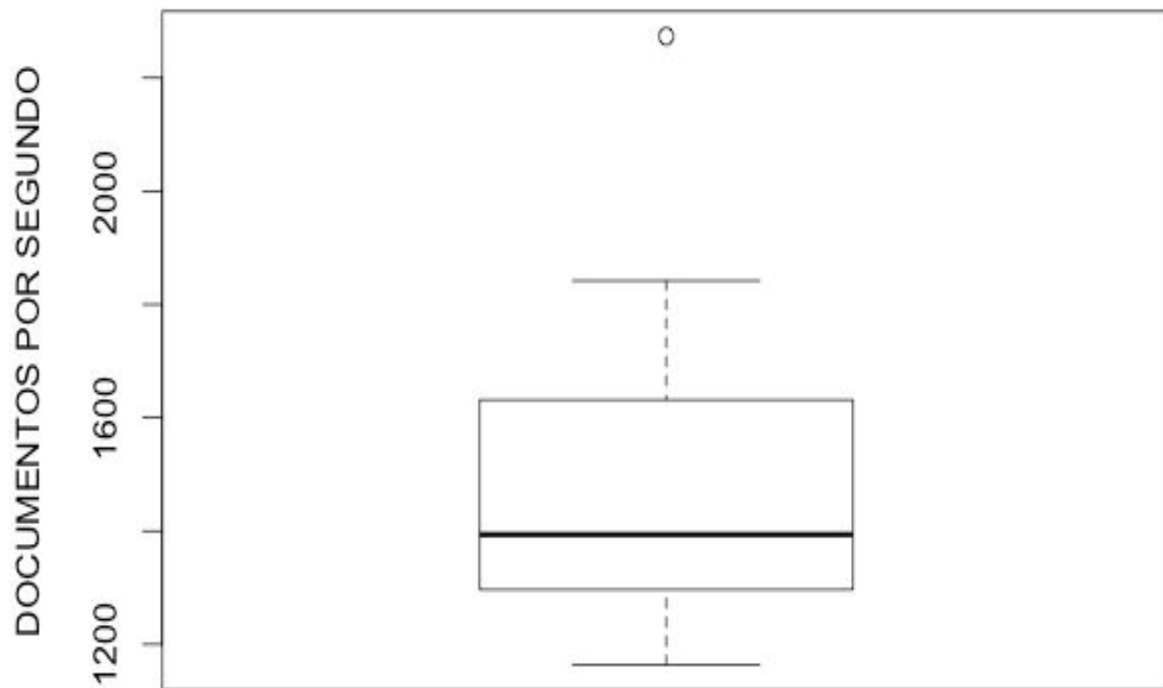


Figura 13. Distribución del rendimiento del proceso ETL medido en documentos procesados por segundo..

MÍNIMO	1er. QUARTIL	MEDIANA	MEDIA	STDEV	3er. QUARTIL	MÁXIMO
1.164	1.316	1.394	1.490	314.88	1.624	2.273

Tabla 6. Estadísticos del rendimiento del proceso ETL medido en documentos procesados por segundo.

Con una media de 1.490 documentos procesables por segundo el **modelo** propuesto y el **proceso** descrito **son óptimos** para ser utilizados en los **proyectos de mayor demanda** donde por ejemplo, una noche de elecciones generales se llegan a recuperar 120 tuits por segundo.

¹² La proporción entre el número de documentos y el número de influenciadores que los generan se conoce como pasión. Una pasión elevada implica la generación de mayor número de documentos por influenciador.

4.5. Arquitectura distribuida

La arquitectura de Cosmos Intelligence está formada por un conjunto de procesos batch y streaming, colas, indexadores y sistemas de almacenamiento. Dentro de este ecosistema debe encajarse la base de datos Neo4J de manera que sume valor y resulte transparente para el resto de elementos, así como que permita la creación de los correspondientes grafos en tiempo real. Centrándonos en la red social Twitter, la arquitectura desarrollada en Cosmos Intelligence para recuperar, clasificar y explotar la información extraída de dicha fuente, responde al esquema mostrado en la Figura 14.

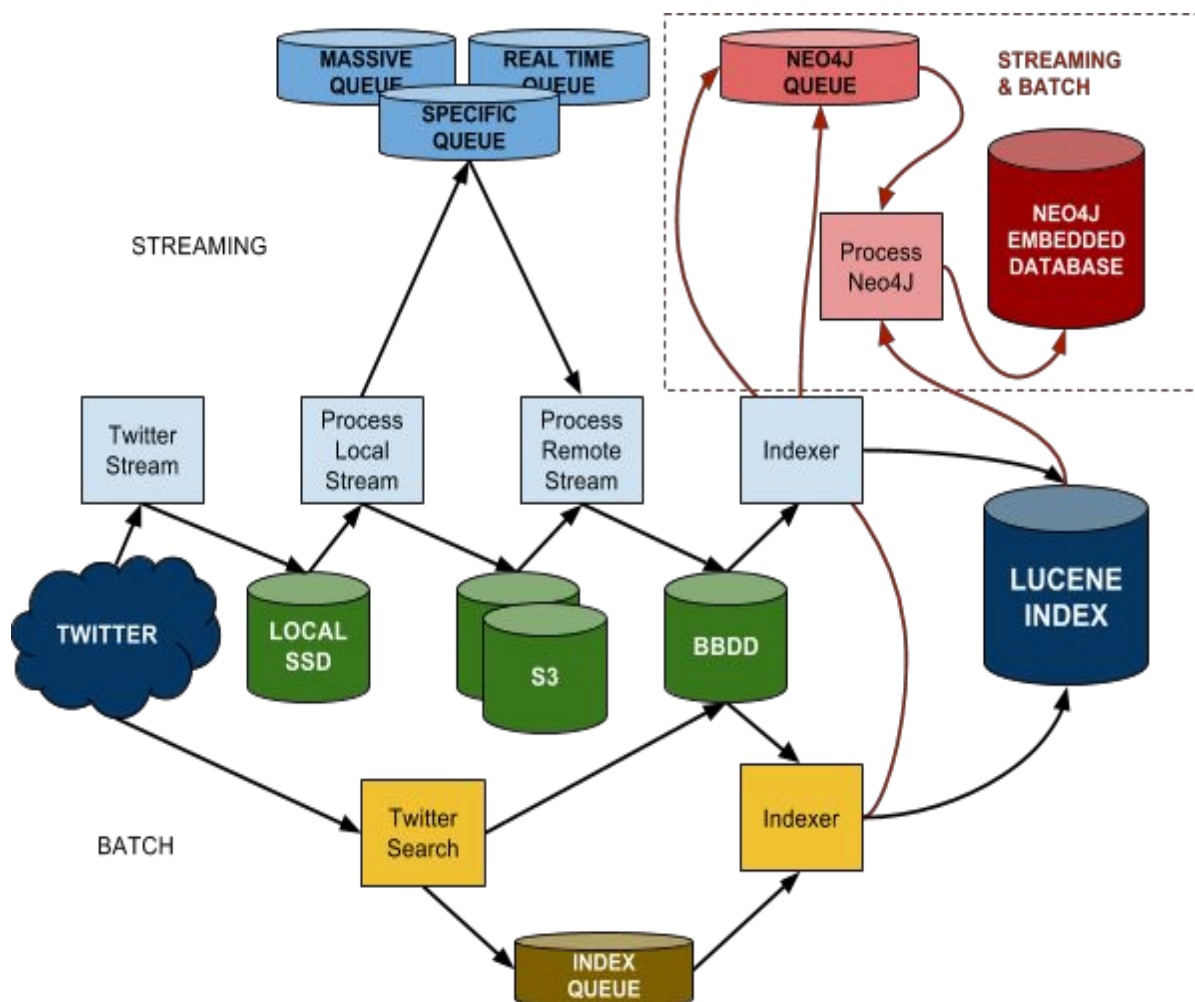


Figura 14. Arquitectura de Cosmos Intelligence.





Símbolo	Significado
	Acceso a la red social Twitter.
	Procesos o agentes que realizan una función concreta.
	Colas de procesos que implementan la filosofía de semáforos.
	Elementos de almacenamiento (discos ssd locales, discos remotos s3, bases de datos SQL, índice de Lucene o bases de datos Neo4J).

Tabla 7. Leyenda de símbolos de la arquitectura de Cosmos Intelligence.

La arquitectura mostrada en la Figura 14 viene definida por los métodos de acceso al dato **según necesidades de volumen y velocidad**. Existen proyectos donde se necesita recuperar la información en **tiempo real** y por lo tanto no se puede perder la conexión con Twitter en ningún momento. Ello requiere una infraestructura que obtenga, clasifique y almacene el dato al instante para que esté disponible en Cosmos en el menor tiempo posible. Este tipo de procesamiento se conoce como **streaming**.

Cuando el dato no necesita ser ofrecido al instante, el procesamiento de información se realiza en lotes lo cual permite minimizar costes. En este caso, el tipo de procesamiento se conoce como **batch**. El número de elementos necesarios respecto a una infraestructura en streaming es menor. Ello motiva la división de la infraestructura en procesado streaming y procesado batch; aún así, los sistemas de almacenamiento son compartidos por ambas tipologías.

Streaming

En primer lugar, el proceso *Twitter Stream* se encarga de descargar los documentos de Twitter y almacenarlos localmente en un *disco SSD*, puesto que el volumen que se puede alcanzar es de cientos de tuits por segundo y por proyecto, y se prima la recuperación para no perder ninguno, delegando a otro proceso su tratamiento.

A continuación, el proceso *Process Local Stream* sube a *Amazon S3* cada uno de los documentos descargados y crea un trabajo en la cola para que el siguiente proceso, *Process Remote Stream*, conozca la existencia de documentos pendiente de procesar. *Process Remote Stream* se encarga de descargar el documento de *Amazon S3*, procesarlo según lo descrito en los apartados 2.4 y 2.5, y almacenarlo en la base de datos correspondiente.

El último proceso de este grupo es la indexación en *Lucene*. *Indexer* consulta el documento almacenado en BBDD y realiza el indexado en *Lucene*. El índice de *Lucene* permite realizar búsquedas textuales de manera eficiente en el contenido de los documentos.

Batch

La parte correspondiente al proceso batch comienza con el proceso *Twitter Search*. Éste recupera mediante búsquedas todos aquellos documentos que cumplen las condiciones indicadas por los analistas. Tras descargar los documentos se almacenan en bases de datos y se crea un trabajo en la cola *INDEX QUEUE* para avisar al proceso *Indexer* que hay trabajos pendientes de atender. Al igual que se realiza en el proceso streaming, *Indexer* procesará el documento almacenado en bases de datos y lo replicará en *Lucene*.

Neo4J

La última parte del proceso tanto streaming como batch corresponde a *Neo4J*, y es la aportación de este trabajo a la arquitectura existente. Siguiendo la misma dinámica anterior, los procesos *Indexer* crean trabajos en la cola específica de *Neo4J* para que, a continuación, *Process Neo4J* recupere el documento del índice de *Lucene*, procese la creación de los nodos y sus relaciones según lo descrito en el apartado 4.3, y lo inserte en la base de datos embedded del proyecto correspondiente.

En resumen, la arquitectura de *Cosmos Intelligence* a nivel de recuperación, clasificación y almacenamiento está montada sobre *Amazon AWS* y se basa en procesado mediante uso de colas de trabajo. De esta manera los procesos son más fáciles de mantener ya que realizan funciones muy concretas y no se ven afectados por cambios en otros puntos de la infraestructura. Además, permiten mayor escalabilidad en función de la demanda.

5. Explotación y visualización

En este apartado se presenta la interfaz de la herramienta de explotación mediante la cual el analista podrá explotar de manera visual la información generada en el modelo relacional descrito e implementado en los apartados previos. El presente apartado se organiza en los siguientes subapartados:

- **Filtrado:** primer punto de configuración del entorno de trabajo donde el analista puede definir con qué organización (proyecto, rango de fechas, subproyecto...) va a trabajar, con qué características de la información (etiquetas) y con qué filtros avanzados (características propias de Twitter).
- **Preprocesamiento:** realizado con Gephi para eliminar el componente gigante, obtener número de nodos y relaciones, influencia por eigenvector y filtrado para visualización de un porcentaje de los más influyentes.
- **Ejemplos de visualización:** conversación, coocurrencia de hashtags y quién dice qué.

5.1. Configurador del entorno de trabajo. Filtrado de la información

Llegados a este punto, la información ha sido recuperada, clasificada y almacenada, y por lo tanto, es momento de interactuar con la herramienta para analizar visualmente la información y generar salidas informativas que permitan extraer conocimiento de la situación.

Una de los pilares de Cosmos Intelligence es la aplicación Web con la cual trabajan los analistas. Es una herramienta propietaria implementada sobre el gestor de contenidos Liferay. El desarrollo de una aplicación en Liferay se basa en pequeñas aplicaciones independientes llamadas portlets. Éstos se desarrollan con el objetivo de cumplir una funcionalidad concreta, y gracias a su naturaleza, pueden ser integrados junto a otros portlets en una misma página y así desplegar una aplicación completa. Lo importante a la hora de desarrollar estas aplicaciones es establecer un canal de comunicación entre los diferentes portlets, como por ejemplo, lanzado eventos broadcast que todos capturarán y almacenando en sesión del navegador aquellos campos que deben ser utilizados como entrada en otros portlets.

La aplicación Cosmos Intelligence está compuesta por numerosas opciones funcionales como puede observarse en el menú lateral de la Figura 15 y definidas en [11].

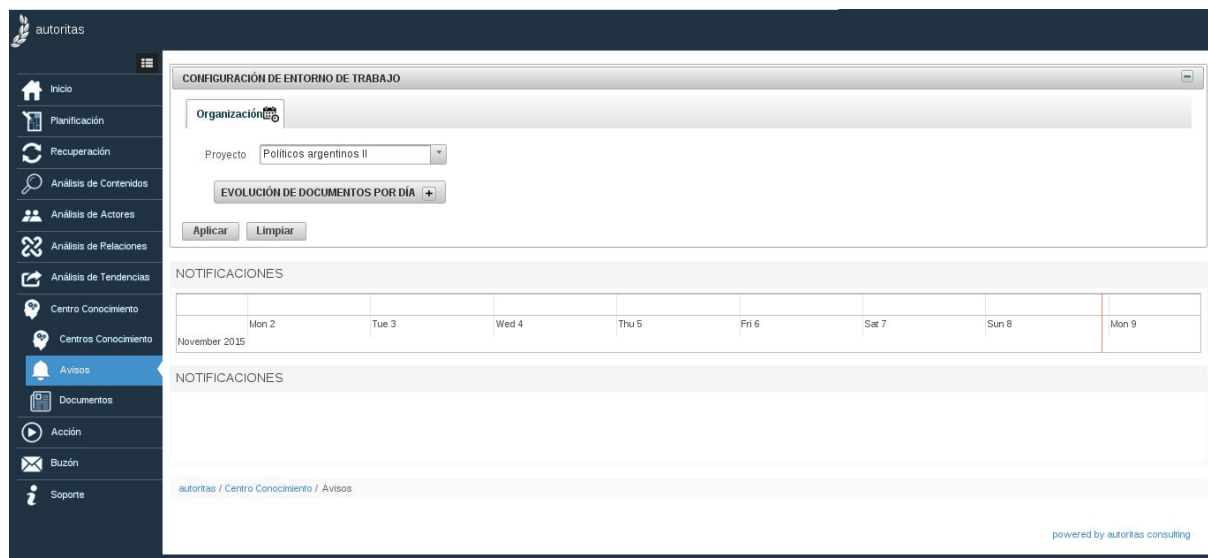


Figura 15. Estructura del frontend de Cosmos Intelligence.

Aquellas pantallas de Cosmos que generan salidas informativas incorporan un filtro de búsqueda llamado *Configurador del Entorno de Trabajo*. Este filtro está dividido en cinco pestañas para poder agrupar, según su finalidad, los campos de búsqueda habilitados al usuario. El analista rellena los campos que le interesan y aplica la búsqueda. Lo que se realiza en ese momento es una llamada a la API de Cosmos Intelligence con todos los campos del filtro de búsqueda. Ésta accederá a los diferentes sistemas de almacenamiento de Cosmos (MySQL, Lucene y Neo4J) y devolverá los resultados al *Configurador del Entorno de Trabajo*. En ese momento, éste lanza un evento que será capturado por los portlets de la pantalla de trabajo y pintarán en pantalla la información requerida.

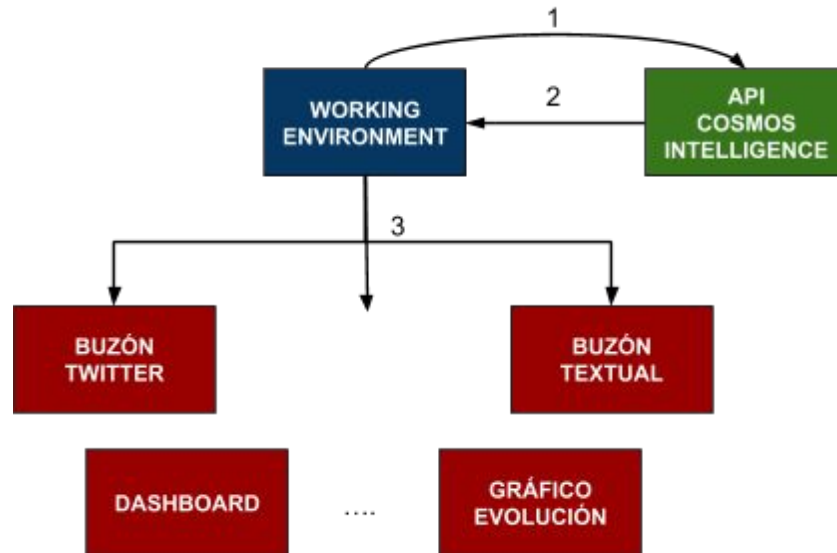


Figura 16. Acciones realizadas al aplicar filtro de búsqueda en el Configurador del Entorno de Trabajo.

La primera pestaña del *Configurador del Entorno de Trabajo* es **Organización**. En ella se indica el *proyecto* y el *periodo de fechas* con el cual se desea trabajar. Para facilitar la elección de dicho periodo se dispone de una gráfica de evolución de documento por día, donde el analista puede observar tres líneas de evolución de documentos por día:

- **Organización:** conjunto de documentos emitidos por la organización del proyecto y/o que hablan sobre ella.
- **Competencia:** documentos donde se habla sobre la competencia de mi organización.
- **Inspiración:** documentos emitidos por aquellas cuentas que se consideran influyentes para el proyecto.

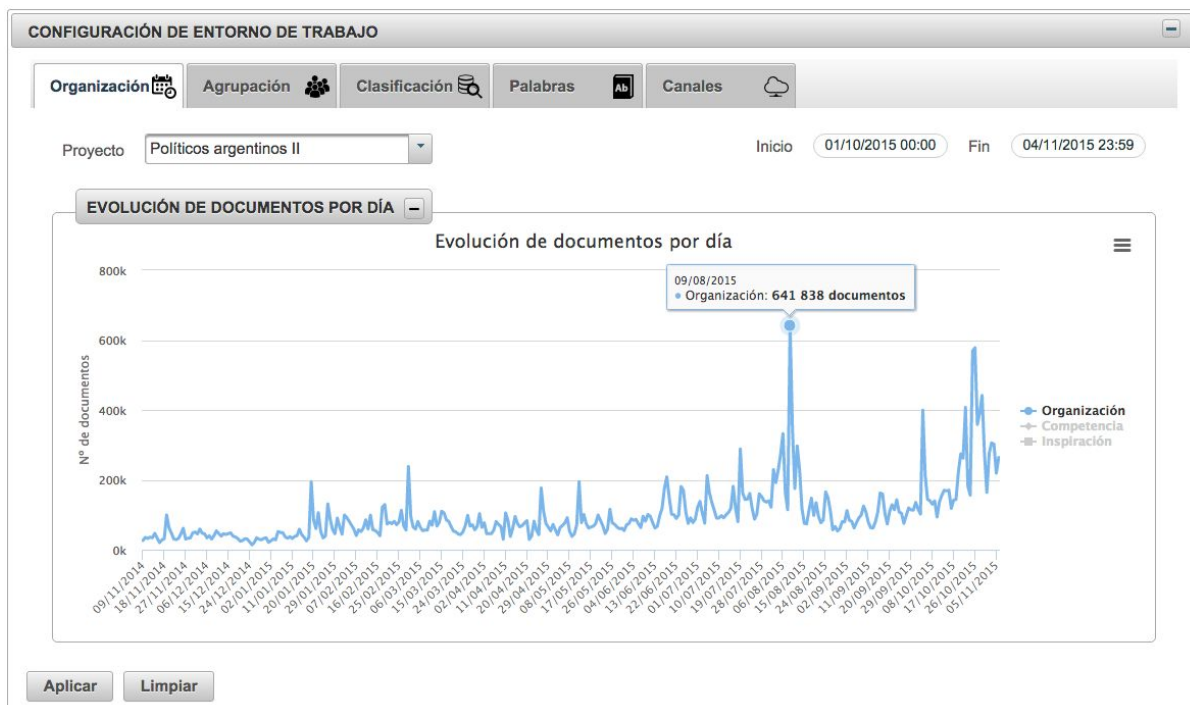


Figura 17. Pestaña Organización del Configurator del Entorno de Trabajo.

En la pestaña **Agrupación** se ofrece la opción de realizar un filtrado más específico a nivel organizativo del proyecto. En primer lugar se permite seleccionar el *tipo de subproyecto* (Organización, Competencia o Inspiración) y a continuación, en función de su valor se seleccionará el subproyecto. Por ejemplo, en el caso del tipo Organización, el *subproyecto* nos permitirá seleccionar aquellos documentos que ha emitido la propia organización, los que han emitido otros influenciadores y hablan de ella, o en su defecto, tanto unos como otros. Como su nombre indica, los selectores de *agrupación* y *subagrupación* complementan a tipo y subproyecto realizado un filtrado de información más minuciosa. Por ejemplo, si se está analizando a la competencia, permite seleccionar un competidor concreto, o llegar a un nivel como comparar lo que se habla de su marca, de sus personas, de sus productos o de sus campañas.

CONFIGURACIÓN DE ENTORNO DE TRABAJO

Organización **Agrupación** Clasificación Palabras Canales

Subproyecto y agrupación

Organización

Subproyectos

Agrupación

Subagrupación

Ha filtrado por:

Tipo Organización

Subproyectos Mi Organizacion, Mi Accion

Agrupación ahorapodemos

Subagrupación Marca, Persona, Producto, Campaña

Aplicar Limpiar

Figura 18. Pestaña Agrupación del Configurador del Entorno de Trabajo.

Puesto que no es posible etiquetar todos los documentos obtenidos de las diferentes fuentes debido a que su contenido no siempre da pie a dicha categorización, el *Configurador del Entorno de Trabajo* se ve obligado a ofrecer la posibilidad de realizar una selección de todos los documentos, aquellos que no han podido ser etiquetados, o en último lugar, los que sí lo han resultado. Estas opciones de filtrado se agrupan bajo la pestaña **Clasificación** mostrada en la Figura 19.

Como se ha comentado anteriormente, un documento se puede clasificar con ninguna, una o varias *etiquetas*. Dichas etiquetas a su vez pueden estar asociadas a una o varias *categorías*. Debido a estas características es posible realizar búsquedas cruzadas de categorías y etiquetas como, por ejemplo, documentos que hablen de un partido político y uno de sus líderes, o que hablen de una persona de la propia organización y además de otra de la competencia.

Además de las opciones anteriores, se ofrece un cuadro informativo donde aparecen las *etiquetas más frecuentes* con las que han sido etiquetados los documentos. Esto sirve de ayuda al analista a la hora de seleccionar las categorías y etiquetas pertinentes.

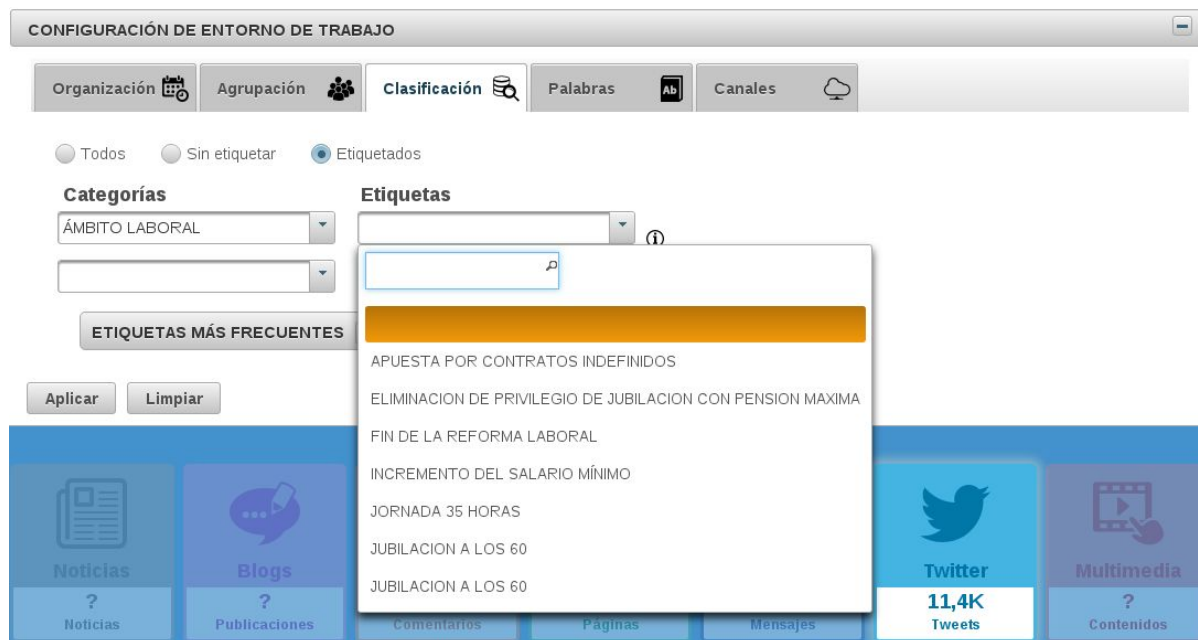


Figura 19. Pestaña Clasificación del Configurator del Entorno de Trabajo.

Gracias a la potencia de Lucene ha sido posible incorporar campos de búsqueda textuales que posibilitan realizar búsquedas dentro del contenido de un documento. Mediante las opciones de la pestaña **Palabras** mostradas en la Figura 20, se permite encontrar documentos que contengan una serie de palabras y/o, documentos donde no aparece ninguna de las palabras indicadas en la opción *ninguna de estas palabras*.

CONFIGURACIÓN DE ENTORNO DE TRABAJO

Organización
Agrupación
Clasificación
Palabras
Canales

Búsqueda libre

Ejemplo: (Energía AND Gas AND NOT Luz) OR Petróleo

☒ Activar búsqueda avanzada

Búsqueda avanzada: encontrar documentos con ...

todas estas palabras
cualquiera de estas palabras
ninguna de estas palabras

Pablo Iglesias
Ejemplo: Gas, Luz
Juan Carlos Monedero

PALABRAS MÁS FRECUENTES

años bien cambio caso **casta** corrupcion dar decir dia el mundo elecciones encuesta errejon **españa** español europa
formacion fuerza general generales gente **gobierno** guerra hecho iglesias iu lider medios monedero monederojc noviembre
nuevo pais parte **partido** politica politico pp presidente programa psOE publico quieren
secretario semana sera sido times vez via

Aplicar

Limpiar

La búsqueda a realizar será:

Pablo Iglesias AND NOT Juan Carlos Monedero

Figura 20. Pestaña Palabras del Configurator del Entorno de Trabajo.

Por último, una vez cumplimentados las opciones de filtrado anteriores, es momento de seleccionar el canal de trabajo en la pestaña **Canales**. Cada uno de los canales posee un filtro de búsqueda avanzado. Dado que el proyecto tratado en este trabajo está basado en Twitter, a continuación se enumeran las opciones de búsqueda avanzada para dicho canal:

- *Influenciador*: permite filtrar por cuentas de usuario.
- *Bio*: encontrar influenciadores cuya bio coincida con la búsqueda.
- *Localización*: buscar en base a la ubicación indicada en la ficha del usuario.

Características de la cuenta.

- *Antigüedad*: filtrar resultados en base a la fecha de creación de la cuenta.
- *En listas*: cribar en función del número de listas que posee el usuario.
- *Amigos*: número de usuarios a los que sigue el influenciador.
- *Seguidores*: número de usuarios que siguen al influenciador.
- *Sexo*: cualquiera, hombre o mujer.

¿Es o contiene ...?

Las opciones siguientes poseen tres estados posibles: sí, no y cualquiera.

- *RT*: indicador de retuit.
- *Mención*: el tweet cita a otro usuario.
- *Hashtag*: indicador de uso de hashtags en el documento.
- *Pregunta*: existencia de signos de interrogación.
- *Exclamación*: existencia de signos de exclamación.
- *Links*: indicador de aparición de enlaces a una URL.

CONFIGURACIÓN DE ENTORNO DE TRABAJO

Organización Agrupación Clasificación Palabras Canales

Canal de trabajo

☐ Noticias ☐ Blogs ☐ Foros ☐ Web ☐ Facebook ☒ Twitter ☐ Multimedia

DISTRIBUCIÓN DE DOCUMENTOS POR CANAL +

☒ Activar búsqueda avanzada

Twitter: búsqueda avanzada

Influenciador

Bio

Localización

Características de la cuenta

Antigüedad Días Amigos Sexo

En listas Seguidores

Miembros

¿Es o contiene...?

☒ RT ☐ Mención ☒ Hashtag ☐ Pregunta ☐ Exclamación ☒ Links

Aplicar Limpiar

Figura 21: Pestaña Canales del Configurador del Entorno de Trabajo.

5.2. Preprocesado del grafo

Una vez filtrada la información y recuperado el grafo desde la base de datos de relaciones implementada con Neo4J siguiendo el modelo descrito en este trabajo, se procede al preprocesado del grafo para reducir su dimensionalidad y permitir una visualización interactiva en el navegador.

Esto es así puesto que debido a la gran dimensionalidad de los proyectos de inteligencia social, se pueden dar grafos con decenas e incluso miles de nodos y cientos de miles de

relaciones, algo irrepresentable gráficamente por motivos de rendimiento y de claridad para el analista. Así pues, interesa de algún modo poder reducir esta dimensionalidad, así como permitir representaciones parciales del grafo donde se muestren los nodos más importantes. Para ello se utiliza la librería Gephi¹³, que permite la importación del grafo obtenido tras la consulta a Neo4J para su procesamiento en, al menos, los siguientes aspectos:

- Selección del componente gigante [4]. El componente gigante de un grafo es el componente conectado del grafo que contiene una fracción fija de los nodos del grafo completo, es decir, un grupo de nodos enlazados entre sí agrupando a la mayoría de los nodos del grafo. Con este filtrado, automáticamente se reduce la dimensionalidad del grafo dejando fuera aquellos nodos menos conectados y que no forman red con el resto de nodos.
- Cálculo de medidas de centralidad, como *eigen vector* [3]. La medida de centralidad *eigen vector* proporciona un indicador de la importancia relativa de un nodo en la red según la posición que ocupa en la propia red. En redes sociales, implica influencia. Disponer de una medida de influencia con respecto a los nodos de la red permite una ordenación y filtrado para mostrar subgrafos con los nodos de mayor influencia, reduciendo así considerablemente la dimensionalidad.
- Cálculo de comunidades [2]. Una comunidad es un conjunto de nodos que están densamente interconectados entre sí y que a su vez están escasamente conectados con el resto del grafo. La detección de comunidades nos permite identificar visualmente grupos de usuarios que interactúan entre sí, hashtags que aparecen conjuntamente, etcétera.

La importancia del preprocesado descrito se puede apreciar en la Figura 22, donde se visualiza un grafo con 1.619 nodos y 3.363 enlaces entre ellos, estadísticas mostradas en la parte superior izquierda de la figura. En el grafo se aprecian grupos de nodos representados en diferentes colores, lo que se corresponde con las comunidades detectadas por el algoritmo y que permiten realizar un análisis de interacción entre los nodos de las mismas. También se pueden apreciar nodos de mayor tamaño, siendo este proporcional a su valor de importancia según la centralidad por *eigen vector*. En la parte derecha de la interfaz se aprecia un control *slider* que indica: “Mostrar 100% nodos”. Este *slider* permite indicar el número de nodos a mostrar teniendo en cuenta su centralidad. Para ello, si se indica un

¹³ <http://gephi.github.io>

porcentaje determinado, se mostrarán todos los nodos que estén en el cuartil superior al porcentaje indicado en valor de *eigen vector*. En la parte superior izquierda se mostrarán estadísticas del número de nodos y ejes visibles en cada momento. El botón *Ver grafo* permitirá mostrar el grafo sólo cuando el usuario haya parametrizado exactamente los nodos a visualizar, evitando así sobrecarga en el lado del cliente.

Otras opciones extra que ofrece la interfaz para un análisis más avanzado son:

- La posibilidad de descargar un gráfico vectorial con todo el grafo (SVG).
- La posibilidad de descargar el fichero GEXF con el grafo completo para su análisis con la herramienta Gephi.

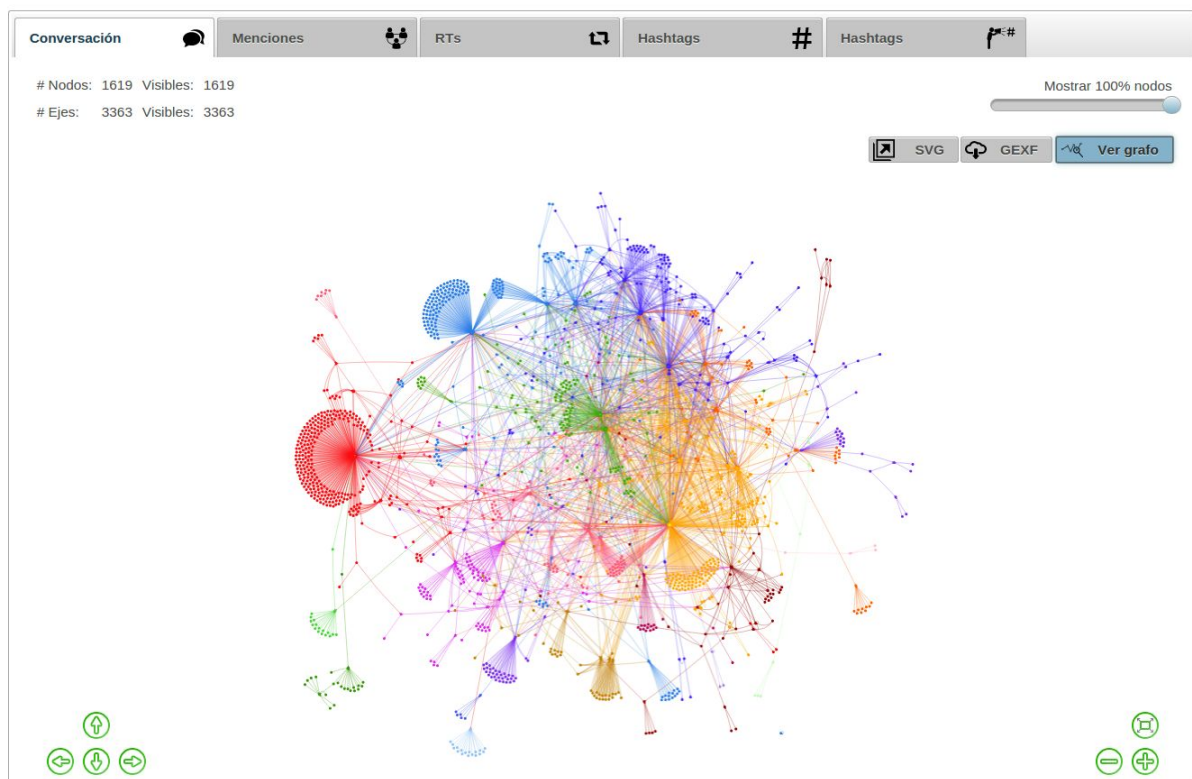


Figura 22. Grafo de conversación en torno al incendio del buque Sorrento.

5.3. Visualización de grafos

La interfaz desarrollada permite la visualización de cinco tipos de grafos:

- **Grafo de conversación**, donde se crea una red a partir de las menciones y RTs que se realizan los usuarios, de manera que se puedan analizar comunidades de usuarios que conversan entre sí.
- **Grafo de menciones**, donde se crea una red a partir de las menciones entre usuarios, de manera que se puedan analizar comunidades de usuarios que se mencionan entre sí.
- **Grafo de RTs**, donde se crea una red a partir de los retuits que se efectúan los usuarios, de manera que se puedan analizar comunidades de usuarios que se retuitean entre sí.
- **Grafo de co-ocurrencia de hashtags**, donde se crea una red a partir de las parejas (o más) de hashtags que aparecen de manera conjunta en cada tuit, de manera que se puedan analizar temas que suelen aparecer de manera conjunta.
- **Grafo de hashtags-usuarios**, donde se crea una red a partir de los hashtags y los usuarios que los emiten, de modo que se pueda analizar quién dice qué.

Puesto que el primer tipo es una combinación de los dos siguientes, a continuación se mostrarán tres ejemplos que incluyen el grafo de conversación, el de co-ocurrencia de hashtags y el de hashtags-usuarios.

5.3.1. Conversación entre usuarios de Twitter

En la Figura 23 se muestra un grafo que representa la conversación entre usuarios de Twitter en torno al incendio del buque Sorrento el pasado 28 de Abril en la costa de Palma de Mallorca.

Tras el procesamiento del grafo y la eliminación del componente gigante, se ha obtenido una representación que contiene 1.619 nodos y 3.363 relaciones, de las cuales el sistema propone la visualización del 3% de los nodos más influyentes. De esta manera el trabajo para el analista es más cómodo puesto que en pantalla se muestran 48 influenciadores con sus 140 relaciones.

Se pueden apreciar diferentes comunidades (diferenciadas por colores) como por ejemplo: comunidades cercanas al entorno oficial de Puerto de Baleares como por ejemplo la cuenta de Balearia o la de José Ramón Bouza, otras comunidades que engloban a las cuentas de salvamento, y por último, las que se agrupan entorno a la comunidad de Greenpeace.

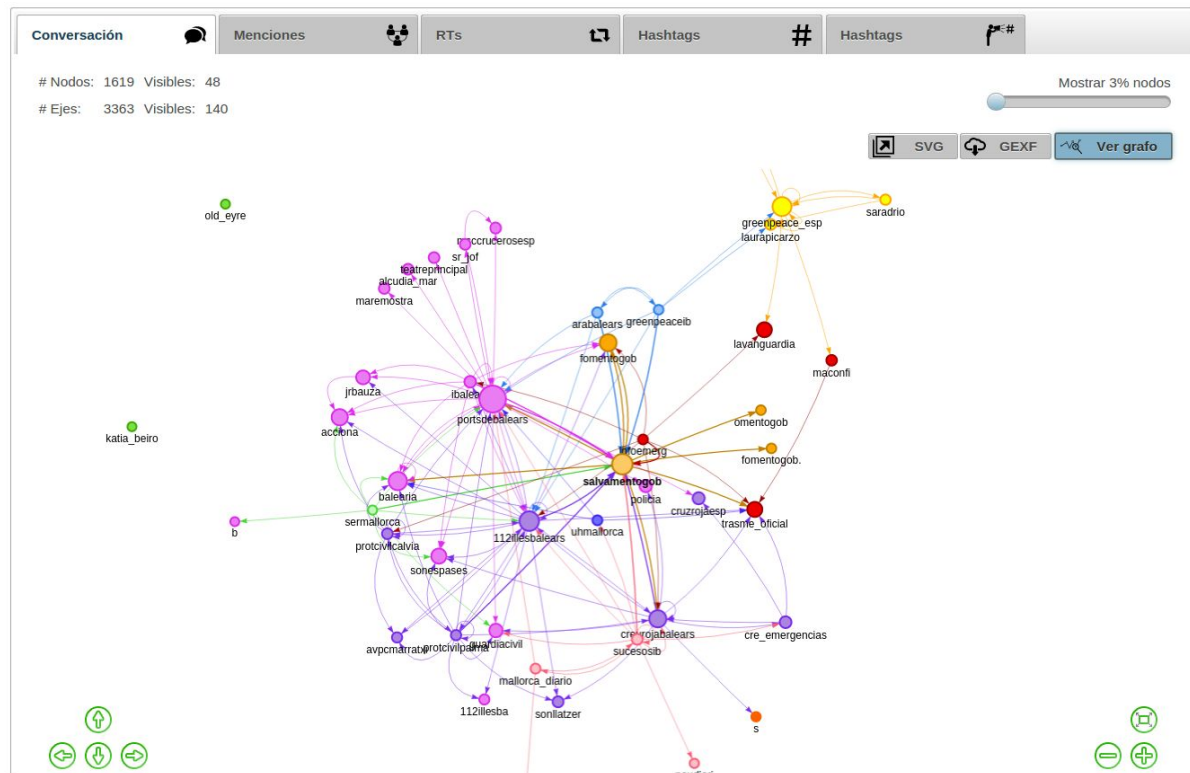


Figura 23: Usuarios activos en la conversación del 28 de abril de 2015 en torno al incendio del buque Sorrento.

La relación entre usuarios puede realizarse de dos maneras posibles; mediante dos consultas que devuelvan los RT y menciones respectivamente (esto no sería eficiente) o bien, obteniendo todos los User y por lo tanto, también los RT y menciones que pudieran unirlos. Esta última **consulta Cypher** sería así:

```
MATCH (o:Organization{organization: "turismobalear"}) -[r:BELONGSTO]->
(n:Document{datetime: "20150428"})<--(h:Hashtag)
OPTIONAL MATCH (t:Tag {tag: "sorrento"})-[l:LABELS]->n-->(u:User)
RETURN n,u
```

En la Figura 24 continua el análisis en torno al incendio del buque Sorrento el pasado 28 de Abril en la costa de Palma de Mallorca, pero en este caso se representa la relación existente entre los hashtags utilizados por los usuarios.

Puede observarse que el hashtag más recurrido por los usuarios ha sido #sorrento, y como la práctica totalidad del resto de nodos están relacionados con éste. Ello significa que los usuarios han hecho uso de diferentes hashtags en un Tweet pero casi siempre han ido acompañados de la etiqueta #sorrento.

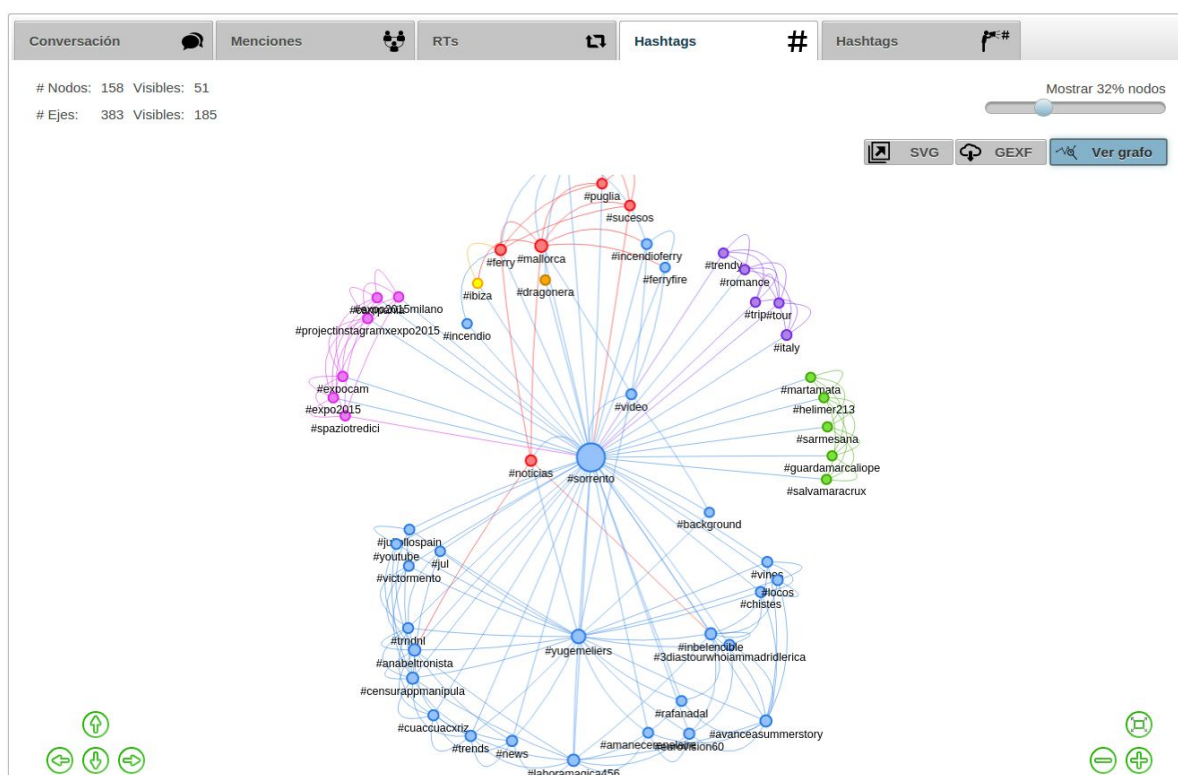


Figura 24: Coocurrencia de hashtags el 28 de abril de 2015 relativos al incendio del buque Sorrento.

46


```

MATCH (o:Organization{organization: "turismobalear"})-
[r:BELONGSTO]-> (n:Document{datetime:"20150428"})<--(h:Hashtag)
OPTIONAL MATCH (t:Tag {tag: "sorrento"})-[l:LABELS]->n-->(u:User)
RETURN n,h

```

5.3.3. Quién dice qué

Por último, la Figura 25 representa una combinación de los dos grafos anteriores. En este caso, se muestra un grafo con los hashtags utilizados entorno al incendio del buque pero relacionado con los usuarios que los han mencionado. Tanto los usuarios como los hashtags están representados por nodos, y las relaciones entre ambos ilustran quien ha mencionado cada hashtag.

El grafo original posee 1.704 nodos y 3.013 relaciones que tras haber sido procesado y eliminado el componente gigante, el sistema propone visualizar el 10% de los mismos, es decir, 1.704 nodos y 3.013 relaciones.

En la parte del grafo mostrada en la Figura 25 se observa como un número elevado de usuarios etiquetan sus Tweets con el hashtag #incendio.

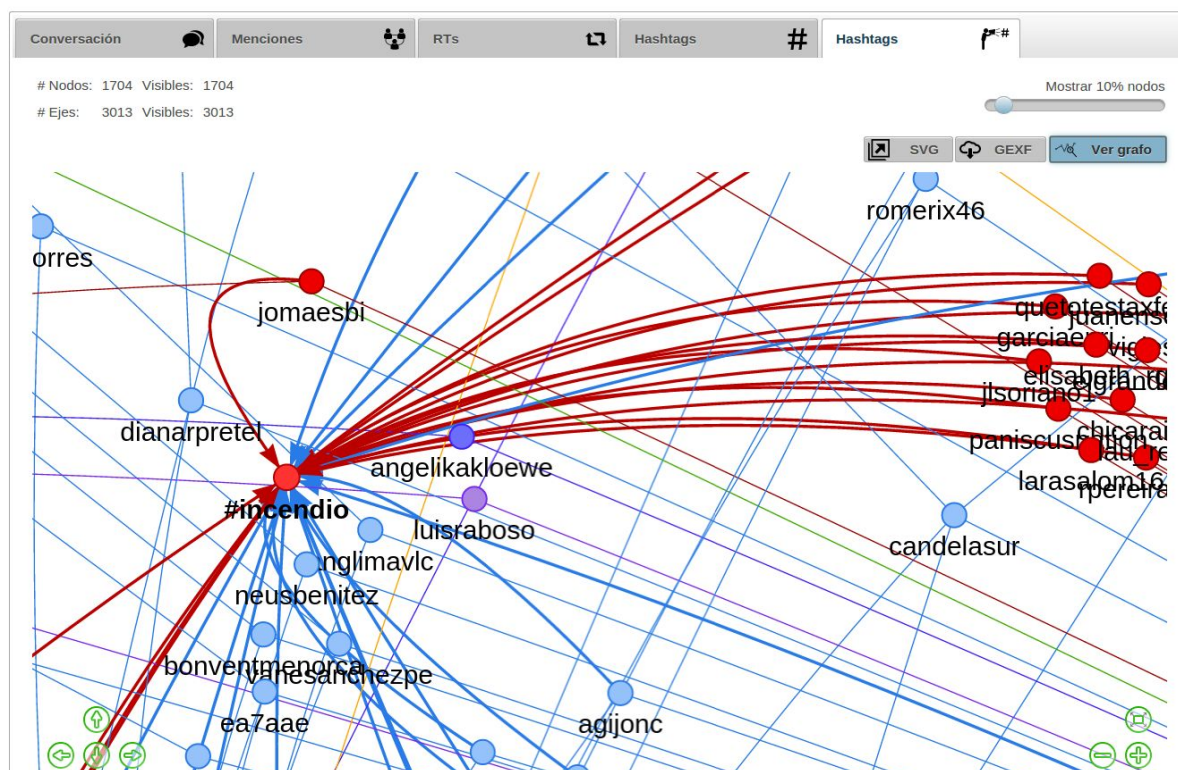


Figura 25: Hashtags nombrados por influenciador el 28 de abril de 2015 relativos al incendio del Sorrento.

En este caso, para conocer los hashtags escritos por cada influenciador necesitamos pivotar sobre el nodo Documento, pues es éste quien conoce al autor del mismo y a su vez los hashtags con los cuales ha sido etiquetado. La **consulta Cypher** que obtiene esta información es la siguiente:

```
MATCH (n:User)-[r:AUTHOR]->({datetime: "20150428"})-[y:HASHTAG]-(x:Hashtag)
RETURN n,y

MATCH (o:Organization{organization: "turismobalear"})-
[r:BELONGSTO]-> (n:Document{datetime:"20150428"})<--(h:Hashtag)
OPTIONAL MATCH (t:Tag)-[l:LABELS]->n-->(u:User)
RETURN n,u,h
```

6. Conclusiones

El desarrollo de este proyecto ha venido motivado por la dificultad de explotar relaciones con datawarehouse planos, debido a que por su naturaleza no están diseñados para ejecutar consultas con elevado número de relaciones, por ejemplo: coocurrencia de hashtags un día determinado entorno a un tema, hashtags nombrados por cada influenciador para un día de crisis en la organización, etc.

La propuesta para paliar estas carencias ha consistido en la definición de un modelo de relaciones y el proceso de conversión de un sistema de almacenamiento plano en una base de datos de relaciones.

Partiendo de un índice de Lucene y mediante un proceso ETL desarrollado en Java se ha conseguido transformar un sistema de almacenamiento plano a una base de datos de relaciones de Neo4J. Gracias a los buenos tiempos de ejecución del proceso, del orden de 1.490 tuits por segundo de media, será posible utilizarlo en los proyectos de mayor demanda donde se llegan a alcanzar picos de recuperación de 120 tuits por segundo.

Un factor primordial del proceso ETL era su integración dentro de la arquitectura existente en para que la generación de las bases de datos se realice en tiempo real. Se ha propuesto un diseño de arquitectura con la infraestructura necesaria que permite realizar dicha incorporación.

Además del proceso ETL, se ha desarrollado una herramienta de explotación que incorpora un configurador de filtros y un visualizador de grafos donde se muestran ejemplos de las sentencias Cypher que lo codifican. De esta manera se consigue realizar búsquedas contra los atributos de los nodos y visualizar los resultados en forma de grafo.

Como trabajo futuro se pretende construir un analizador de consultas Cypher que se incorpore en la herramienta anterior y permita generar consultas dinámicas de inteligencia relacional. Este proyecto está siendo abordado en [13].

7. Bibliografía

1. Aguilar Gómez, A. Deformaciones de la Lengua Española en la Prensa. 1er. Congreso Internacional de la Lengua Española (1997).
2. Blondel, V. Guillaume, J.L. Lambiotte, R. Lefebvre, E. Fast Unfolding of Communities in Large Networks. In: Journal of Statistical Mechanics: Theory and Experiment (2008).
3. Bonacich, P. Factoring and Weighting Approaches to Clique Identification. In: Journal of Mathematical Sociology (1972).
4. Charu G. A. Social Network Data Analytics. Springer (2014).
5. Dart, V., Rangel, F.M., Martínez, J.C., Ballester, J.V. Mejora de la Recuperación de Información en Entorno Oracle: Aplicación Práctica a Recursos Cartográficos. II Congreso Español de Recuperación de Información, CERI (2012).
6. Dart, V. Sistema de Recuperación de Información para la Escucha Activa. Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging. Universitat Politècnica de València (2012). Directores: Paolo Rosso y Francisco Rangel.
7. Fabra, R., Rangel, F., Rosso, P. NLEL UPV Autoritas Participation at Discrimination between Similar Languages (DSL) 2015 Shared Task. Joint Workshop on Language Technology for Closely Related Languages, Varieties and Dialects (LT4CloseLang2 – VarDial 2). Recent Advances in Natural Language Processing-RANLP (2015).
8. Fernández, S. Definición metodológica para la generación de inteligencia relacional. Diploma de especialización en Big Data. Universidad Politécnica de Valencia (2015). (en curso)
9. Franco-Salvador, M., Rangel, F., Rosso, P. Mariona, T., Martí, A. Language Variety Identification using Distributed Representations of Words and Documents. In: 6th International Conference of CLEF on Experimental IR meets Multilinguality, Multimodality, and Interaction, CLEF (2015).
10. Franco-Salvador, M., Rosso, P, Rangel, F. Distributed Representations of Words and Documents for Discriminating Similar Languages. Joint Workshop on Language Technology for Closely Related Languages, Varieties and Dialects (LT4CloseLang2 - VarDial 2). Recent Advances in Natural Language Processing-RANLP (2015).
11. Llinares, M. Metodología para la Generación de Inteligencia desde Fuentes Abiertas (OSINT). Máster en Analista de Inteligencia. Universidades Rey Juan Carlos y Carlos III de Madrid (2015). (en curso).

12. Liu, B., Hu, M. and Cheng, J. Opinion Observer: Analyzing and Comparing Opinions on the Web. In: Proceedings of the 14th International World Wide Web conference (WWW-2005).
13. Núñez, V. Generación de Inteligencia Relacional con Cypher. Diploma de especialización en Big Data. Universidad Politécnica de Valencia (2015). (en curso).
14. Ramos, A., Rangel, F. Autoritas participation at MoReBikeS: Model Reuse with Bike rental Station data. MoReBikeS: 2015 ECML-PKDD Challenge on "Model Reuse with Bike rental Station data" (2015).
15. Rangel F., Hernández I., Rosso P., Reyes A., P. Emotions and Irony per Gender in Facebook. In: Proc. Workshop on Emotion, Social Signals, Sentiment & Linked Open Data (ES³LOD), LREC (2014).
16. Rangel, F., Rosso, P. On the Identification of Emotions in Facebook Comments. Rangel, F., Rosso, P. In Proceedings of the First International Workshop on Emotion and Sentiment in Social and Expressive Media: approaches and perspectives from AI (ESSEM 2013) A workshop of the XIII International Conference of the Italian Association for Artificial Intelligence - AI*IA (2013).
17. Rangel, F., Rosso, P. On the Impact of Emotions on Author Profiling. In: Information Processing & Management (2015).
18. Rangel, F., Rosso, P. On the Multilingual and Genre Robustness of EmoGraphs for Author Profiling in Social Media (2015). In: 6th Int. Conf. of CLEF on Experimental IR meets Multilinguality, Multimodality, and Interaction, CLEF (2015).
19. Rangel F., Rosso P., Chugur I., Potthast M., Trenkmann M., Stein B., Verhoeven B., Daelemans W. Overview of the 2nd Author Profiling Task at PAN 2014. In: Cappellato L., Ferro N., Halvey M., Kraaij W. (Eds.) CLEF 2014 Labs and Workshops, Notebook Papers. CEUR-WS.org (2014).
20. Rangel, F., Rosso, P., Koppel, M., Stamatatos, E., Inches, G. . Overview of the Author Profiling Task at PAN 2013.(Eds.), Note- book Papers of CLEF 2013 LABS and Workshops, CLEF-2013. CEUR-WS.ORG (2013).
21. Shuyo, N. Language Detection Library for Java. <http://code.google.com/p/language-detection> (2010).
22. Volgmann S., Rangel F., Niggemann O., Rosso P. Emotional Trends in Social Media – A State Space Approach. In: Proc. 21st Conf. on Artificial Intelligence, ECAI (2014).

8. Anexo A. Código fuente del proceso ETL

```
/*
 * Link to libraries:
 *
 * http://neo4j.com/docs/stable/tutorials-java-embedded-setup.html
 * http://archive.apache.org/dist/lucene/java/3.0.3/
 */

package ETLneo4jembedded;

// Librerías de Autoritas para encapsular el acceso a los índices por proyecto
import JSupport.Constants.Indexes;
import irutils.Lucene.LuceneMngr;

import com.sun.corba.se.impl.orbutil.graph.Graph;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.concurrent.TimeUnit;

import org.apache.lucene.document.Document;
import org.apache.lucene.index.IndexReader;

import org.neo4j.graphdb.Direction;
import org.neo4j.graphdb.DynamicLabel;
import org.neo4j.graphdb.GraphDatabaseService;
import org.neo4j.graphdb.Label;
import org.neo4j.graphdb.Node;
import org.neo4j.graphdb.Relationship;
import org.neo4j.graphdb.RelationshipType;
import org.neo4j.graphdb.ResourceIterator;
import org.neo4j.graphdb.Result;
import org.neo4j.graphdb.Transaction;
import org.neo4j.graphdb.factory.GraphDatabaseFactory;
import org.neo4j.graphdb.factory.GraphDatabaseSettings;
import org.neo4j.graphdb.schema.IndexDefinition;
import org.neo4j.graphdb.schema.Schema;
```

```

/*
 * @author Andrés Ramos Pérez
 */
public class ETLNeo4J {

    public static String NEO4JPATH = "/mnt/tfm_neo/neo.db/#schema#";
    public static String INDEXPATH =
        "/mnt/tfm_neo/index/projects/#schema#/documents";
    public static String MLPATH = "/mnt/tfm_neo/neo.db/#schema#.gml";

    private static enum RelTypes implements RelationshipType
    {
        RT, MENTION, AUTHOR, LABELS, BELONGS, HASHTAG, BELONGSTO, CONTAINS,
        DOCRTUSER, DOCUMENTIONUSER
    }

    public static void main(String[] args) throws IOException {

        if (args.length<0) {
            System.out.println("USE: java -jar ETLNeo4JEmbedded.jar schema");
            return;
        }

        String sSchema = args[0];
        NEO4JPATH = NEO4JPATH.replaceAll("#schema#", sSchema);
        INDEXPATH = INDEXPATH.replaceAll("#schema#", sSchema);
        MLPATH = MLPATH.replaceAll("#schema#", sSchema);

        CreateAndPopulateUniqDB();          // PROCESO ETL
        PrintStatistics();                   // ESTADÍSTICAS SECCIÓN 4.4

        System.out.println("Finish ETLNeo4JEmbedded");
    }

    private static void CreateAndPopulateUniqDB() throws IOException {
        System.out.println("CreateAndPopulateUniqDB...");

        // DB creation
        GraphDatabaseService graphDb = new GraphDatabaseFactory()
            .newEmbeddedDatabaseBuilder( NEO4JPATH )
            .setConfig( GraphDatabaseSettings.pagecache_memory, "2048M" )
            .setConfig( GraphDatabaseSettings.string_block_size, "60" )
            .setConfig( GraphDatabaseSettings.array_block_size, "300" )
            .newGraphDatabase();

        try ( Transaction tx = graphDb.beginTx() ) {
            graphDb.schema()
                .constraintFor(DynamicLabel.label("User"))

```

```

        .assertPropertyIsUnique("username")
        .create();
graphDb.schema()
        .constraintFor(DynamicLabel.label("Document"))
        .assertPropertyIsUnique("doc")
        .create();
graphDb.schema()
        .constraintFor(DynamicLabel.label("Organization"))
        .assertPropertyIsUnique("organization")
        .create();
graphDb.schema()
        .constraintFor(DynamicLabel.label("Tag"))
        .assertPropertyIsUnique("tag")
        .create();
graphDb.schema()
        .constraintFor(DynamicLabel.label("Cat"))
        .assertPropertyIsUnique("cat")
        .create();
graphDb.schema()
        .constraintFor(DynamicLabel.label("Hashtag"))
        .assertPropertyIsUnique("hashtag")
        .create();
tx.success();
}

Relationship relationship;
IndexReader reader = LuceneMngr_Old.GetReader(INDEXPATH);

int iTx = 1;
Transaction tx = graphDb.beginTx();

for (int i=0; i<reader.maxDoc(); i++) {
    if (reader.isDeleted(i)) continue;
    Document doc = reader.document(i);
    String sChannel = doc.getValues("channel")[0];
    if (!sChannel.equalsIgnoreCase("TWITTER")) {
        continue;
    }

    iTx++;
    if (iTx%10000==0) {
        tx.success();
        tx.close();
        tx = graphDb.beginTx();

        System.out.println("-->" + (i+1) + "/" + reader.maxDoc() + ":" +
            iTx);
    }

    // User/Influencer Node
    String sUser = doc.getValues("influencer")[0];
    sUser = sUser.toLowerCase();
    String sList =

```

```

        (doc.getValues("list").length>0)?doc.getValues("list")[0]:"0";
String sFriends =
        (doc.getValues("friends").length>0)?doc.getValues("friends")[0]:"0";
String sFollowers =
(doc.getValues("followers").length>0)?doc.getValues("followers")[0]:"0";
String sGender =
        (doc.getValues("gender").length>0)?doc.getValues("gender")[0]:"any";

Node influencerNode = CreateOrGetNode(sUser.toLowerCase(), graphDb);
influencerNode.setProperty( "lists", sList );
influencerNode.setProperty( "friends", sFriends );
influencerNode.setProperty( "followers", sFollowers );
influencerNode.setProperty( "gender", sGender );

// Document Node
String sId = doc.getValues("id")[0];
String sRt = (doc.getValues("rt").length>0)?doc.getValues("rt")[0]:"";
String sMention =
        (doc.getValues("mention").length>0)?doc.getValues("mention")[0]:"";
String ssHashtag =
        (doc.getValues("hashtag").length>0)?doc.getValues("hashtag")[0]:"";
String sQuestion =
        (doc.getValues("question").length>0)?doc.getValues("question")[0]:"";
String sExclamation =
(doc.getValues("exclamation").length>0)?doc.getValues("exclamation")[0]:"";
String sLink =
        (doc.getValues("link").length>0)?doc.getValues("link")[0]:"";
String sTimestamp =
(doc.getValues("timestamp").length>0)?doc.getValues("timestamp")[0]:"";

Node docNode = CreateOrGetNodeDoc(sId.toLowerCase(), graphDb);
docNode.setProperty( "rt", sRt );
docNode.setProperty( "mention", sMention );
docNode.setProperty( "hashtag", ssHashtag );
docNode.setProperty( "question", sQuestion );
docNode.setProperty( "exclamation", sExclamation );
docNode.setProperty( "link", sLink );
docNode.setProperty( "datetime", sTimestamp );

RelTypes RelationshipType = RelTypes.AUTHOR;
relationship =
        influencerNode.createRelationshipTo( docNode, RelationshipType );

/*
 * RT and MENTION
 - influencerNode to influencerNode
 - docNode to influencerNode
 */

String sTweet = doc.getValues("content")[0];
int iIni = sTweet.indexOf("@"); if (iIni>=0) iIni++;
int iEnd = IndexOf(sTweet, iIni);
Node userNode;

```



```

while (iIni<iEnd && iIni!=-1)
{
    String sToUser = sTweet.substring(iIni, iEnd);
    if (sToUser.endsWith(":"))
        sToUser = sToUser.substring(0, sToUser.length()-1);

    String sRelationType = "";
    RelTypes RelationTypeDoc;
    String sRelationTypeDoc = "";

    if (sTweet.contains("RT @" + sToUser)) {
        RelationType = RelTypes.RT;
        sRelationType = "RT";
        RelationTypeDoc = RelTypes.DOCRTUSER;
        sRelationTypeDoc = "DOCRTUSER";
    } else {
        RelationType = RelTypes.MENTION;
        sRelationType = "MENTION";
        RelationTypeDoc = RelTypes.DOCMENTIONUSER;
        sRelationTypeDoc = "DOCMENTIONUSER";
    }

    sToUser = sToUser.toLowerCase();

    userNode = CreateOrGetNode(sToUser.toLowerCase(), graphDb);

    relationship =
        influencerNode.createRelationshipTo( userNode, RelationType );
    relationship.setProperty( "type", sRelationType );

    /* Additional relation document to user */
    relationship =
        docNode.createRelationshipTo( userNode, RelationTypeDoc );
    relationship.setProperty( "type", sRelationTypeDoc );

    iIni = sTweet.indexOf("@", iEnd);
    iEnd = IndexOf(sTweet, iIni);
}

// Document Tag and Cat Nodes
String sTagged = doc.getValues("tagged")[0];

if (sTagged.equalsIgnoreCase("T")) {
    String sTag = doc.getValues("tags")[0];
    String[] sTags = sTag.split("\\s+");
    String sCat = doc.getValues("cats")[0];
    String[] sCats = sCat.split("\\s+");
    Node fourthNode;
    Node fifthNode;

    for (iIni=0; iIni<sTags.length; iIni++) {
        RelationType = RelTypes.LABELS;

        /* Tag to document*/

```

```

        fourthNode =
            CreateOrGetNodeTag(sTags[iIni].toLowerCase(), graphDb);
        relationship =
            fourthNode.createRelationshipTo( docNode, RelationType );

        /* Tag to Cat */
        RelationType = RelTypes.BELONGS;

        fifthNode =
            CreateOrGetNodeCat(sCats[iIni].toLowerCase(), graphDb);
        relationship =
            fourthNode.createRelationshipTo( fifthNode, RelationType );
    }
}

// Organization Node
String sOrganization =
    doc.getValues("subprj")[0] + "_" + doc.getValues("search")[0];
String sSubprjtype =
    (doc.getValues("subprjtype").length>0)?doc.getValues("subprjtype")[0]:"";
String sSubprj =
    (doc.getValues("subprj").length>0)?doc.getValues("subprj")[0]:"";
String sSearch =
    (doc.getValues("search").length>0)?doc.getValues("search")[0]:"";

Node sixthNode =
    CreateOrGetNodeOrganization(sOrganization.toLowerCase(), graphDb);
sixthNode.setProperty( "subprjtype", sSubprjtype );
sixthNode.setProperty( "subprj", sSubprj );
sixthNode.setProperty( "search", sSearch );

RelationType = RelTypes.BELONGSTO;
relationship = docNode.createRelationshipTo( sixthNode, RelationType );

// Hashtag Node
String sHashtag =
    (doc.getValues("hashtag").length>0)?doc.getValues("hashtag")[0]:"F";

if (sHashtag.equalsIgnoreCase("T")) {
    String sHashtags = doc.getValues("hashtags")[0];
    String[] sHashtagss = sHashtags.split("\\s+");
    Node hashtagNode;

    for (iIni=0; iIni<sHashtagss.length; iIni++) {
        hashtagNode =
            CreateOrGetNodeHashtag(sHashtagss[iIni].toLowerCase(), graphDb);
        RelationType = RelTypes.HASHTAG;
        relationship =
            hashtagNode.createRelationshipTo( docNode, RelationType );
    }
}

tx.success();
tx.close();

```

```

        graphDb.shutdown();
    }

    private static Node CreateOrGetNode(String user, GraphDatabaseService graphDb) {
        Node result = null;
        Label label = DynamicLabel.label( "User" );
        try ( Transaction tx = graphDb.beginTx() ) {
            try ( ResourceIterator<Node> users = graphDb.findNodes( label,
                "username", user ) ) {
                if (users.hasNext()) {
                    result = users.next();
                }
            }

            if (result==null) {
                result = graphDb.createNode(label);
                result.setProperty( "username", user );
            }

            tx.success();
        }

        return result;
    }

    private static Node CreateOrGetNodeDoc(String doc,
        GraphDatabaseService graphDb) {
        Node result = null;
        ResourceIterator<Node> resultIterator = null;

        try ( Transaction tx = graphDb.beginTx() ) {
            String queryString = "MERGE (n:Document {doc: {doc}}) RETURN n";
            Map<String, Object> parameters = new HashMap();
            parameters.put("doc", doc);
            resultIterator = graphDb.execute(queryString,
                parameters).columnAs("n");
            result = resultIterator.next();
            tx.success();

            return result;
        }
    }

    private static Node CreateOrGetNodeTag(String tag,
        GraphDatabaseService graphDb) {
        Node result = null;
        ResourceIterator<Node> resultIterator = null;

        try ( Transaction tx = graphDb.beginTx() ) {
            String queryString = "MERGE (n:Tag {tag: {tag}}) RETURN n";
            Map<String, Object> parameters = new HashMap();
            parameters.put("tag", tag);
            resultIterator = graphDb.execute(queryString,

```

```

        parameters).columnAs("n");
        result = resultIterator.next();
        tx.success();

        return result;
    }
}

private static Node CreateOrGetNodeCat(String cat,
    GraphDatabaseService graphDb) {
    Node result = null;
    ResourceIterator<Node> resultIterator = null;

    try ( Transaction tx = graphDb.beginTx() ) {
        String queryString = "MERGE (n:Cat {cat: {cat}}) RETURN n";
        Map<String, Object> parameters = new HashMap();
        parameters.put("cat", cat);
        resultIterator = graphDb.execute(queryString,
            parameters).columnAs("n");
        result = resultIterator.next();
        tx.success();

        return result;
    }
}

private static Node CreateOrGetNodeOrganization(String organization,
    GraphDatabaseService graphDb) {
    Node result = null;
    ResourceIterator<Node> resultIterator = null;

    try ( Transaction tx = graphDb.beginTx() ) {
        String queryString = "MERGE (n:Organization {organization:
            {organization}}) RETURN n";
        Map<String, Object> parameters = new HashMap();
        parameters.put("organization", organization);
        resultIterator = graphDb.execute(queryString,
            parameters).columnAs("n");
        result = resultIterator.next();
        tx.success();

        return result;
    }
}

private static Node CreateOrGetNodeHashtag(String hashtag,
    GraphDatabaseService graphDb) {
    Node result = null;
    ResourceIterator<Node> resultIterator = null;

    try ( Transaction tx = graphDb.beginTx() ) {
        String queryString = "MERGE (n:Hashtag {hashtag: {hashtag}}) RETURN n";
        Map<String, Object> parameters = new HashMap();
        parameters.put("hashtag", hashtag);

```

```

        resultIterator = graphDb.execute(queryString,
            parameters).columnAs("n");
        result = resultIterator.next();
        tx.success();

        return result;
    }
}

private static int IndexOf(String content, int ini)
{
    int iPos = ini;
    if (iPos>=0 && iPos<content.length())
    {
        for (iPos=ini;iPos<content.length();iPos++)
        {
            if (!Character.isDigit(content.charAt(iPos)) &&
                !Character.isLetter(content.charAt(iPos)) &&
                content.charAt(iPos) != '_' )
                break;
        }
    }
    return iPos;
}

private static void PrintStatistics() {
    System.out.println("STATISTICS");

    Map config = new HashMap();
    config.put( "read_only", "true" );
    GraphDatabaseService graphDb =
        new GraphDatabaseFactory().newEmbeddedDatabaseBuilder(
            NEO4JPATH )
            .setConfig( config )
            .newGraphDatabase();

    try (Transaction ignored = graphDb.beginTx();
        Result result = graphDb.execute("MATCH (n:User) RETURN count(n)
                                         LIMIT 1")) {

        int i=0;
        while (result.hasNext()) {
            String n = "";
            Map<String, Object> row = result.next();
            for (Entry<String, Object> column : row.entrySet()) {
                n = column.getValue() + " ";
            }
            System.out.println("Usuarios: " + n);
        }

        ignored.close();
    }
}

```

```

try (Transaction ignored = graphDb.beginTx());
    Result result = graphDb.execute("MATCH (n:Hashtag)
        RETURN count(n) LIMIT 1") {

    int i=0;
    while (result.hasNext()) {
        String n = "";
        Map<String, Object> row = result.next();
        for (Entry<String, Object> column : row.entrySet()) {
            n = column.getValue() + "";
        }
        System.out.println("Hashtags: " + n);
    }
    ignored.close();
}

try (Transaction ignored = graphDb.beginTx());
    Result result = graphDb.execute("MATCH (n:Cat)
        RETURN count(n) LIMIT 1") {

    int i=0;
    while (result.hasNext()) {
        String n = "";
        Map<String, Object> row = result.next();
        for (Entry<String, Object> column : row.entrySet()) {
            n = column.getValue() + "";
        }
        System.out.println("Cat: " + n);
    }
    ignored.close();
}

try (Transaction ignored = graphDb.beginTx());
    Result result = graphDb.execute("MATCH (n:Tag)
        RETURN count(n) LIMIT 1") {

    int i=0;
    while (result.hasNext()) {
        String n = "";
        Map<String, Object> row = result.next();
        for (Entry<String, Object> column : row.entrySet()) {
            n = column.getValue() + "";
        }
        System.out.println("Tag: " + n);
    }
    ignored.close();
}

try (Transaction ignored = graphDb.beginTx());
    Result result = graphDb.execute("MATCH (n:Document)
        RETURN count(n) LIMIT 1") {

```

```

        int i=0;
        while (result.hasNext()) {
            String n = "";
            Map<String, Object> row = result.next();
            for (Entry<String, Object> column : row.entrySet()) {
                n = column.getValue() + " ";
            }
            System.out.println("Documentos: " + n);
        }
        ignored.close();
    }
    graphDb.shutdown();
}
}

```