UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Máster en Big Data y Data Science: ciencia e ingeniería de datos

TRABAJO FIN DE MÁSTER

DESCUBRIENDO SIMILITUD ENTRE DOCUMENTOS A PARTIR DE COMPARTICIÓN DE ENTIDADES NOMBRADAS

Emilio José Macías Macías

Tutor: Pablo A. Haya Coll

Julio 2019

DESCUBRIENDO SIMILITUD ENTRE DOCUMENTOS A PARTIR DE COMPARTICIÓN DE ENTIDADES NOMBRADAS

AUTOR: Emilio José Macías Macías

TUTOR: Pablo A. Haya Coll

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Julio de 2019

Resumen

El Procesamiento del Lenguaje Natural (en adelante PLN) consiste en el reconocimiento y manipulación del lenguaje humano por parte de una máquina.

En este Trabajo Fin de Máster (en adelante TFM), se aborda el problema del reconocimiento automático de información relevante en noticias, y cómo usar esta capacidad para descubrir similitud entre documentos y para visualizar la distribución de las entidades nombradas en dichos documentos.

Posibles aplicaciones de este campo de trabajo son la clasificación de noticias, los buscadores de texto, la recomendación de contenidos y la gestión de comentarios en atención al cliente.

En este TFM, se han aplicado diversas técnicas y herramientas de código abierto utilizadas durante este máster. En concreto, cabe destacar a Apache Spark (plataforma de computación distribuida que permite escalar el procesamiento de grandes cantidades de datos), Apache Kafka (plataforma para desacoplar los distintos subsistemas mediante el uso de colas para el procesamiento de datos en tiempo real), Elasticsearch (gestor y buscador de bases de datos NoSQL con indexación de documentos en formato JSON), Kibana (herramienta analítica de visualización de datos en tiempo real y conectada con Elasticsearch) y Spacy (librería Python para PLN con reconocimiento de entidades nombradas, entre otras muchas funciones).

El desarrollo de este trabajo se ha realizado de una manera incremental e iterativa, basada en cuatro prototipos funcionales. El primer prototipo consiste en la anotación manual de un conjunto de noticias de referencia y la posterior evaluación de los modelos pre-entrenados de Spacy para el reconocimiento de entidades nombradas. En el segundo prototipo se pretende mejorar el reconocimiento de esos modelos mediante distintas técnicas de entrenamiento y recursos lingüísticos. El tercer prototipo añade un módulo de persistencia de las noticias y entidades nombradas, así como la visualización de los resultados. Por último, el cuarto prototipo aborda los problemas de la arquitectura escalable y el procesamiento distribuido en tiempo real.

Agradecimientos

Deseo expresar mi agradecimiento a todas las personas que de una forma u otra me han apoyado durante la duración de este máster y en concreto a la realización de este TFM.

En primer lugar, me gustaría agradecer a mi tutor, Pablo Haya, por su encomiable labor a la hora de guiarme durante este trabajo y por su predisposición constante a resolverme las dudas que me iban surgiendo.

A los profesores de la Universidad Autónoma de Madrid y a mis compañeros de clase, por los consejos y apoyos mutuos, y en especial por todo el tiempo que hemos pasado juntos durante estos dos últimos años.

A mis padres, por su sacrificio durante tantos años y por inculcarme unos valores que me han ayudado a ser la persona que soy hoy en día.

Y por último, y no por ello menos importante, a Trisha, por apoyarme en los buenos y en los malos momentos, y por ser la estrella que guía mi camino.

“Un lenguaje nuevo, vasto y poderoso se está desarrollando para el uso futuro del análisis, en el cual se pueden introducir sus principios con el fin de que tengan una aplicación práctica más veloz y precisa al servicio de la humanidad.”

Ada Lovelace, 1843

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

En el último siglo ha habido grandes avances en el campo del procesamiento automático del lenguaje, con hitos como las máquinas Colossus (capaces descifrar mensajes durante la segunda guerra mundial), el test de Turing (por el que se evalúa la habilidad de una máquina para comunicarse de una forma tan inteligente que es indistinguible al ser humano), el supercomputador IBM Watson (capaz de derrotar a los mejores jugadores del juego de preguntas y respuestas Jeopardy!) y el asistente personal Siri (capaz de mantener una conversación hablada con los usuarios). Estos y otros muchos hitos por llegar hacen que el PLN sea uno de los campos con más recorrido de la inteligencia artificial [1].

Dentro del PLN, existen cuatro tipos de análisis claramente diferenciados:

- Análisis morfológico (estructura interna de las palabras)
- Análisis sintáctico (reglas y estructura de las frases)
- Análisis semántico (extracción del significado)
- Análisis pragmático (interpretación del significado según el contexto)

Este TFM se centra en el análisis semántico de noticias en general, y en particular, en la tarea de extracción de información mediante el reconocimiento de entidades nombradas, ya sean de tipo lugar, persona u organización (categorías más comunes de entidades). Esta tarea es muy útil en el análisis de noticias para averiguar qué personas se relacionan con cada organización o en qué lugares ocurren ciertos eventos.

1.2 Objetivos

El principal objetivo de este TFM es poner en práctica los conocimientos adquiridos durante el máster (tanto en la ciencia de datos como en la arquitectura Big Data) para desplegar una solución de software completa, desde la ingesta de datos hasta la visualización de resultados, y manteniendo la interacción entre los distintos módulos.

Aunque es importante conocer los principios teóricos y la función de los distintos *frameworks*, no es hasta poner todas las herramientas en producción interactuando entre sí cuando uno realmente puede profundizar en el funcionamiento interno de *frameworks* como Spark, Kafka y Elasticsearch.

En más detalle, uno de los retos será intentar mejorar la precisión de los modelos de lenguaje ya entrenados por la librería Spacy, una tarea que requiere de altos recursos computacionales, tiempo de entrenamiento y la correcta elección de los datos de *training*.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

1. **Introducción**

   Se mencionan las motivaciones de realizar este trabajo y los objetivos que con él se persiguen. Además, se listan las distintas secciones de esta memoria.

2. **Estado del arte**

   Se describen brevemente los últimos avances en el campo del procesamiento del lenguaje natural en general, del reconocimiento de entidades nombradas en particular y los últimos desarrollos en computación distribuida.

3. **Diseño**

   Se describe la arquitectura del sistema de forma incremental y a través de prototipos funcionales, empezando con la evaluación y mejora de modelos existentes de lenguaje, continuando con la persistencia y visualización, y terminando con la arquitectura del cuarto prototipo (que coincidirá con la del sistema completo).

4. **Desarrollo**

   Se detalla la implementación de los cuatro prototipos funcionales mencionados anteriormente y se describe la estructura de los distintos módulos.

5. **Integración, pruebas y resultados**

   Se explica cómo se integran todos los componentes y se describen las pruebas realizadas para validar el correcto funcionamiento de cada prototipo.

6. **Conclusiones y trabajo futuro**

   Se mencionan las metas alcanzadas durante la realización del trabajo y se listan posibles mejoras futuras para completar el proyecto.

7. **Anexos**

   Se añaden enlaces al código fuente y se describen los pasos necesarios para la instalación del sistema.

2 Estado del arte

2.1 Modelos de lenguaje

Los modelos actuales de lenguaje utilizan una técnica denominada *word embeddings*, por la cual las palabras y frases son tratadas como vectores de números reales de tal forma que se pueden aplicar operaciones matriciales sobre ellos e incluso medir la distancia entre las distintas palabras dentro de un espacio vectorial. Entre los modelos más utilizados para producir *word embeddings* se encuentra Word2vec, que a partir de un corpus de textos es capaz de producir un espacio vectorial multi-dimensional donde a cada palabra del corpus se le asigna un vector dentro de ese espacio.

Hoy en día, los algoritmos de PLN más avanzados están basados en *deep learning*, en concreto en redes recurrentes de tipo LSTM (Long Short-Term Memory) y capas GRU (Gated Recurrent Units). Estas redes procesan secuencias de palabras mientras van almacenando en memoria la información más relevante de lo que han visto hasta ese momento; de esta forma, se puede predecir la siguiente letra o palabra, vital para tareas como traducción de idiomas y generación automática de textos.

Entre los modelos basados en LSTM (de tipo bidireccional) se encuentra *BERT*, un modelo pre-entrenado que tiene en cuenta el contexto y que además tiene la capacidad de extraer *embeddings* [2]. Tal es la precisión, que los modelos entrenados mediante *transfer learning* a partir de BERT, suelen cosechar los mejores resultados en métricas como el *Stanford Question Answering Dataset (SQuAD)*, que consiste en evaluar las respuestas de los modelos a distintas preguntas sobre artículos de Wikipedia [3].

2.2 Reconocimiento de entidades nombradas

Para evaluar los modelos más avanzados en el reconocimiento de entidades nombradas se suele utilizar la anotación en formato BIO, en la que se añaden etiquetas de tipo *B* (inicio de la entidad nombrada), *I* (palabra intermedia o final) y *O* (no entidad). Además, cada una de esas etiquetas va acompañada de una de las tres categorías (*LOC*, *PER* u *ORG*). Existen distintos corpus anotados de referencia que se usan para evaluar la precisión en la predicción de los modelos. En cuanto a las métricas, lo más común es evaluar *precision* (porcentaje de las entidades detectadas que son correctas), *recall* (porcentaje de las entidades anotadas en el corpus que han sido detectadas) y *f1-score* (balance de las dos anteriores).

Si hace unos años los modelos basados en árboles de decisión con *boosting* obtenían los mejores resultados en competiciones como la de CoNLL-2002 (reconocimiento de

entidades nombradas en español) [4], hoy en día, los modelos más precisos en cuanto al reconocimiento de entidades están basados en redes bidireccionales LSTM y redes convolucionales CNN [5].

2.3 Similitud entre textos

Actualmente existen diversas técnicas para representar gráficamente la similitud entre textos. Una de estas técnicas es mediante el uso de grafos o DSN *(Document Similarity Network)*, donde cada vértice representa un documento y la longitud de arista (o ausencia de esta) representa la similitud entre cada par de documentos, tal y como refleja la siguiente imagen:

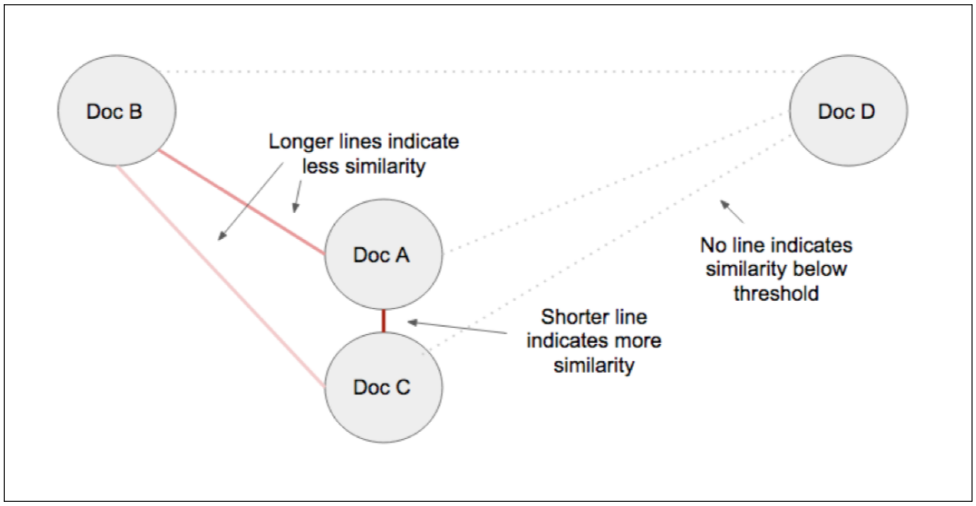

Figura 1: Grafo de similitud entre documentos [6]

Actualmente existen diversas técnicas para calcular la similitud entre textos, basadas en palabras, en corpus o en conocimiento. Una de ellas es mediante *Latent Dirichlet Allocation* aplicado a la extracción de temas (*topics*), donde la longitud de las aristas indica el grado de similitud entre los principales temas de los documentos [6]. Si nos centramos únicamente en las entidades nombradas (objetivo de este TFM), la similitud basada en palabras (o términos) se puede calcular mediante distintas funciones, entre las que se encuentran el índice de Jaccard, la distancia euclídea y la distancia coseno [7].

Un método que se ha estudiado para calcular la similitud entre documentos consiste en extraer tanto las entidades nombradas como los términos más relevantes mediante la medida TF-IDF (*Term Frequency-Inverse Document Frequency*) para, a continuación, crear por cada documento un grafo de n-gramas con dichos términos y la posterior comparación entre las aristas de dichos grafos. El resultado es un valor entre 0 y 1, donde 1 expresa la máxima similitud entre documentos. La siguiente figura representa las distintas fases de este método:

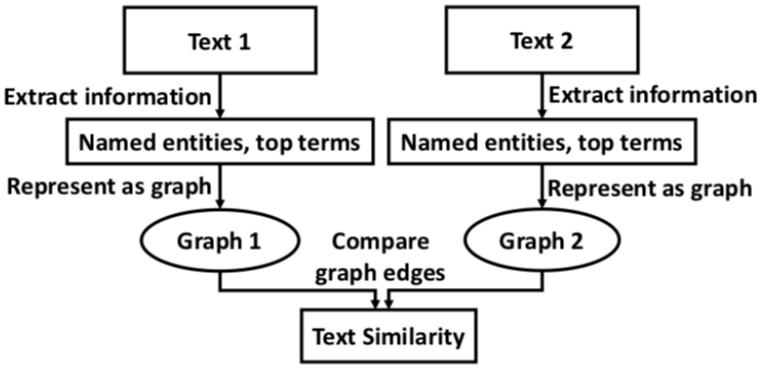

Figura 2: Método de cálculo de similitud entre textos [8]

2.4 Tecnologías de PLN

Entre las librerías específicas de PLN más utilizadas hoy en día se encuentran NLTK y Spacy. NLTK fue una de las primeras librerías para PLN y posee bastantes funciones lingüísticas como análisis léxico, etiquetado gramatical o reconocimiento de entidades nombradas [9]. Sin embargo, NLTK no está pensada para producción a gran escala y su uso se reduce principalmente al ámbito académico y de investigación. Por otro lado, Spacy (utilizada en este TFM) trata de suplir las carencias de NLTK en cuanto a la velocidad de proceso ya que está en gran parte escrita en C, lo que la hace una librería muy recomendable para procesos con alta carga computacional. Además, posee una API sencilla de utilizar para usuarios no familiarizados con el ámbito de la lingüística.

En cuanto a la programación de *deep learning,* aunque existe la posibilidad de desarrollar directamente sobre el núcleo de Tensorflow o Pytorch, existen librerías de alto nivel para la arquitectura y entrenamiento de redes neuronales, como por ejemplo Keras, una API escrita en Python que permite programar redes neuronales de una manera sencilla y abstrayéndose de los elementos más avanzados de bajo nivel. Keras se suele utilizar tanto para el reconocimiento de imágenes como para el procesamiento de lenguaje natural [10].

2.5 Computación distribuida

Aunque existen diversas tecnologías para realizar el procesamiento distribuido de datos, lo más común en los entornos productivos de hoy en día es utilizar el entorno de código libre de Apache sobre un ecosistema Hadoop o Spark. Entre las compañías que utilizan tecnologías Apache se encuentran Netflix, Facebook, Formula 1 o NASA [11].

La siguiente imagen representa las herramientas típicas de un ecosistema Hadoop, de las cuales Zookeeper, Spark y Kafka se han utilizado para el desarrollo de este TFM.

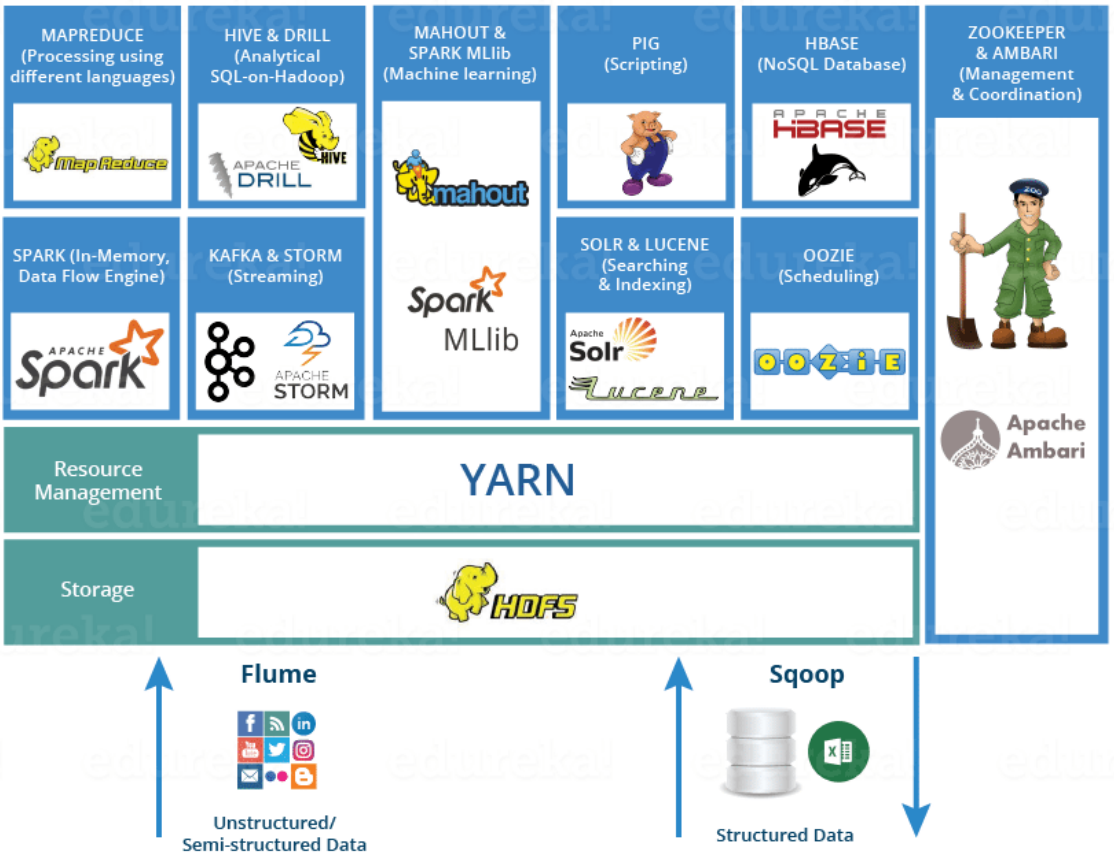

Figura 3: Ecosistema Hadoop [12]

Apache Zookeeper ofrece servicios para la coordinación entre los nodos de un clúster distribuido, manteniendo una alta disponibilidad y simplificando la gestión de los servidores de un entorno Hadoop. A fecha de este TFM, Zookeeper se encuentra en la versión 3.5.5.

Apache Spark es un *framework* tolerante a fallos para computación distribuida, basado en Hadoop. La ventaja que aporta Spark sobre Hadoop MapReduce es la baja latencia debido al uso intensivo de memoria frente a disco. A fecha de este TFM, Spark se encuentra en la versión 2.4.3. Spark dispone de librerías específicas para grafos, machine learning, streaming y SQL, y con API en lenguajes como Scala, Java y Python.

Apache Kafka es una plataforma de alto rendimiento y baja latencia para la manipulación en tiempo real de fuentes de datos, mediante la suscripción de los clientes (*consumers*) a las colas de mensajería. A fecha de este TFM, Kafka se encuentra en la versión 2.2.1.

Por último, en cuanto a la persistencia de datos, existe gran cantidad de bases de datos NoSQL; estos pueden estar orientados a documento, a columnas, a grafo o a clave-valor. Entre los orientados a documento, se encuentra Elasticsearch, que junto con Kibana y el motor Lucene, forman una solución potente para el indexado, búsqueda y visualización de documentos JSON. A fecha de hoy, la plataforma Elastic se encuentra en la versión 7.1.1.

3 Diseño

Para este TFM se ha seguido una metodología iterativa e incremental, produciendo un prototipo funcional por cada una de las cuatro fases que se describen a continuación.

3.1 Primera fase. Evaluación de modelos pre-entrenados

Esta primera fase consiste en la evaluación de los modelos existentes de PLN ya entrenados por la librería Spacy [13]. De las diversas métricas que existen, se ha decidido utilizar *precision, recall* y *f1-score*. Además, se construirá una matriz de confusión entre las tres categorías de entidades (lugar, persona y organización).

Dado que las noticias que vamos a analizar se encuentran en español, utilizaremos uno de los modelos de lenguaje de Spacy para el español, en concreto el *es_core_news_sm*, que ha sido entrenado mediante una red convolucional con artículos de los corpus de AnCora y Wikipedia [14]. Aparte del reconocedor de entidades (*NER*), también dispone de un etiquetador de sintaxis y analizador de dependencias. Enfocándonos únicamente en el *NER*, aunque este modelo es capaz de identificar una cuarta categoría de entidades (MISC), se ha decidido trabajar únicamente con las tres principales (PER, LOC y ORG). Los modelos de lenguaje de Spacy tiene el siguiente *pipeline*:

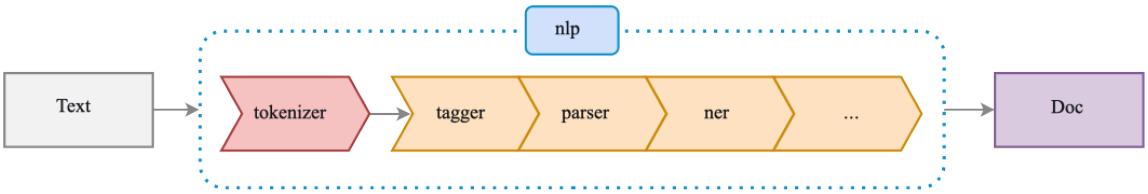

Figura 4: Pipeline de Spacy [15]

El primer paso para poder evaluar la precisión del modelo es disponer de un conjunto de datos de test, que en nuestro caso llamaremos *Gold Standard*, y que estará formado por un grupo de noticias que habremos etiquetado manualmente. Es importante realizar este etiquetado manualmente para otorgar a este conjunto de referencia la máxima fiabilidad.

Antes de etiquetar noticias manualmente, necesitamos obtener un *dataset* de noticias en español que utilizaremos tanto en esta fase de evaluación como en los prototipos posteriores. Vamos a partir de un conjunto de noticias de más de 340 mil documentos en español obtenidos de Webhose [16]. Este conjunto de datos incluye tanto noticias en medios digitales como comentarios en blogs, todos ellos en español y en formato JSON.

Una vez disponemos de un *dataset* amplio de noticias, debemos decidir qué cantidad de noticias vamos a anotar manualmente para nuestro *Gold Standard*. Aunque lo

recomendable sería anotar del orden de varios cientos de noticias para obtener una mayor fiabilidad en la evaluación, por motivos de tiempo, se ha decidido anotar 100 noticias únicamente. Es importante remarcar que la elección de estas 100 noticias debe ser completamente aleatoria para evitar posibles sesgos en los datos de evaluación.

El último paso para obtener el *Gold Standard* consiste en anotar manualmente las 100 noticias de test. Para ello, se ha decidido usar la herramienta web Brat Rapid Annotation Tool [17], que permite fácilmente cargar noticias en formato .txt y seleccionar las entidades y categorías visualmente. Las entidades anotadas se guardan en ficheros con extensión .ann. Las siguiente imagen muestra el panel de anotación de una noticia y su correspondiente fichero de salida:

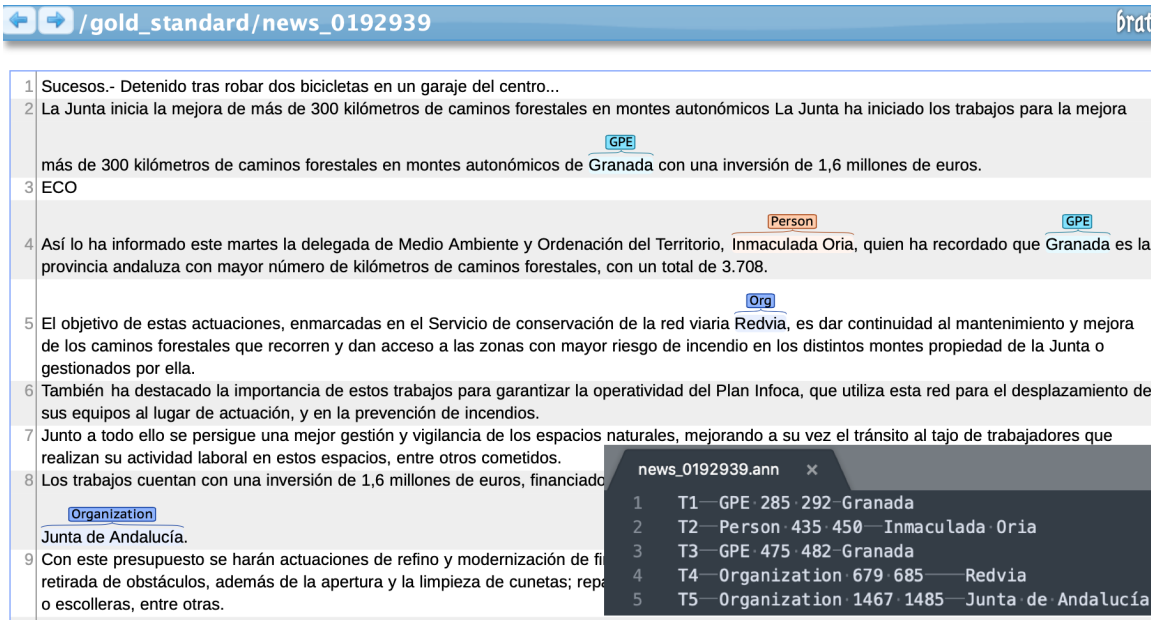

Figura 5: Brat Rapid Annotation Tool y fichero de salida [17]

Con el *Gold Standard* ya anotado, el siguiente paso sería cargar el modelo de Spacy para predecir las entidades de las 100 noticias de test y comparar estas predicciones con las anotaciones del *Gold Standard*. Para ello será necesario instalar las librerías Python de Spacy y de bratreader (que provee de una interfaz para cargar las anotaciones de los ficheros .ann) [18] https://github.com/clips/bratreader [consulta: 17/06/2019].

La siguiente figura representa el *pipeline* del prototipo 1:

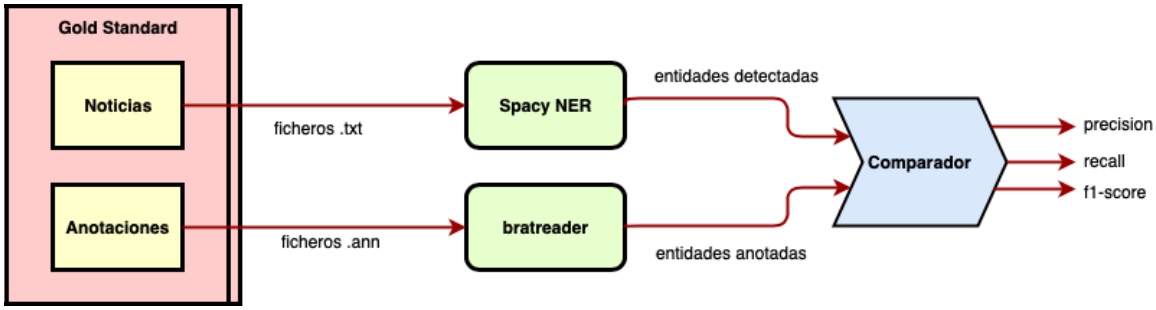

Figura 6: Diseño del prototipo 1

3.2 Segunda fase. Mejora del modelo

Una vez hemos evaluado la precisión del modelo pre-entrenado de Spacy, la siguiente fase consiste en entrenar dicho modelo con el objetivo de mejorar la detección de entidades nombradas de tipo ORG, LOC y PER.

Una de las opciones más fiables para entrenar el modelo de lenguaje consiste en volver a anotar manualmente un conjunto de datos (distintos a los del *Gold Standard*). Esto hace que los ejemplos de entrenamiento sean altamente fiables. Pero se necesitaría un largo periodo de tiempo para anotar manualmente un número de noticias considerablemente grande como para producir mejoras en el modelo. Debido a este inconveniente, se ha decidido utilizar otra estrategia que, aunque pierde en fiabilidad, es más automática y por lo tanto ahorra tiempo de preparación de los datos.

Estos nuevos ejemplos de entrenamiento se generarán sintéticamente mediante el uso de *matching* basado en reglas. Este método consiste en utilizar un repositorio de entidades nombradas junto con su categoría correspondiente, y realizar búsquedas de estas entidades en las noticias de entrenamiento. Por cada fichero de noticia, se utilizarán las coincidencias para generar sintéticamente noticias anotadas de entrenamiento.

Por tanto, es importante usar un repositorio fiable de entidades nombradas, ya que de lo contrario estaremos entrenando el modelo con información errónea, lo que muy probablemente desembocará en una bajada de la precisión.

Otro de los aspectos a tener en cuenta y que se suele pasar por alto, es el conocido como problema del *catastrophic forgetting*, por el cual modelos de lenguaje basados en redes neuronales tienden a olvidar el conocimiento ya aprendido a medida que se va entrenando con nuevos datos. Esto quiere decir que entidades detectadas por el modelo original pueden no ser detectadas por el modelo re-entrenado si los datos de entrenamiento no aportan suficientes ejemplos con dichas entidades ya conocidas. Una de las estrategias más eficientes ante este problema se denomina mecanismo *pseudorehearsal [19]*, que consiste en intercalar ejemplos de entrenamiento con entidades ya detectadas por el

modelo original (datos de revisión) a medida que se van introduciendo nuevos ejemplos y se va entrenando el modelo.

Finalmente, a continuación se listan los parámetros a tener en cuenta para entrenar el modelo de lenguaje, así como los valores propuestos para poder mejorar significativamente la precisión dentro de un tiempo razonable y evitando overfitting. Estos valores se han elegido después de analizar los resultados de varias iteraciones con modificación de parámetros.

Parámetro	**Descripción**	**Valor**
Nº datos de entrenamiento	Hay que tener en cuenta los recursos computacionales disponibles y el tiempo necesario de entrenamiento, pero también que un conjunto demasiado pequeño no aportará mejoras significativas.	5,000
Nº épocas	Similar al anterior. Se necesita reordenar aleatoriamente las noticias de entrenamiento en cada época para evitar favorecer ciertas entidades sobre otras.	4
Tasa de *dropout*	Se descarta cada noticia de entrenamiento con probabilidad 50% para evitar que el modelo memorice ejemplos concretos.	0.5
Tasa de revisión	Balance entre noticias con anotaciones de entidades ya conocidas (revisión) y noticias con anotaciones nuevas generadas sintéticamente mediante *matching* basado en reglas.	0.7

Tabla 1: Parámetros de entrenamiento

3.3 Tercera fase. Persistencia y visualización

La tercera fase del proyecto está dividida en dos partes, por un lado será necesario persistir los resultados del *NER* entrenado, y por otro, analizar visualmente dichos resultados.

Se utilizará Elasticsearch (ES) para indexar las entidades nombradas que han sido detectadas por nuestro modelo [20]. La única restricción que tenemos será que el formato de los documentos de entrada a ES debe ser JSON, por lo que será necesario convertir la salida del *NER*. Sin embargo, Elasticsearch tiene la ventaja de que nos permite utilizar Kibana directamente para realizar analíticas visuales sobre nuestros datos [21].

Para poder indexar en ES, primero necesitamos definir los índices. Para decidir el *mapping*, necesitamos considerar el uso que se va a hacer de estos datos. Los requisitos más importantes deben ser que la estructura de los documentos JSON deberá permitir:

1. REQ-1: Buscar todas las noticias que compartan una determinada entidad

2. REQ-2: Analizar la distribución de las entidades nombradas.

3. REQ-3: Crear un grafo de distancia entre noticias de un intervalo de tiempo.

Dado que Kibana no soporta la agregación de los objetos anidados, no es posible incluir la categoría de cada entidad detectada en cada noticia. Es por ello que es necesario duplicar la información y utilizar dos índices diferentes para poder visualizar, uno con la lista de entidades junto con la noticia, y otro con entidades y categorías. Esta aproximación se inspira en la denormalización que propone ES para evitar hacer *joins* [22]. Además, con el objetivo de obtener las noticias generadas en un intervalo de tiempo concreto, es necesario añadir una marca de tiempo.

Teniendo en cuenta lo anterior, se propone el siguiente *mapping* para los índices:

Índice	**Mapping**	**Requisito**

news-index	"**fullText**": {
	"type": "text"
	},
	"**entities**": {
	"type": "text" (list)
	},
	"**timestamp**": {
	"type": "date"
	}
		REQ-1
		REQ-3

entity-index	"**name": {**
	"type": "text"
	},
	"**label**": {
	"type": "text"
	}
		REQ-2

Tabla 2: Mapping de índices de ES

Además, sería útil poder visualizar las entidades detectadas en su contexto (noticia), por lo que será necesario seguir una anotación en formato *standoff* [23] para poder ser

visualizadas mediante la herramienta de anotación Brat. Para ello, las entidades detectadas por el *NER* deberán almacenarse junto con la posición de sus caracteres inicial y final dentro de la noticia. Esta información se cargará en Brat como ficheros de anotación .ann.

El siguiente diagrama representa la arquitectura del tercer prototipo:

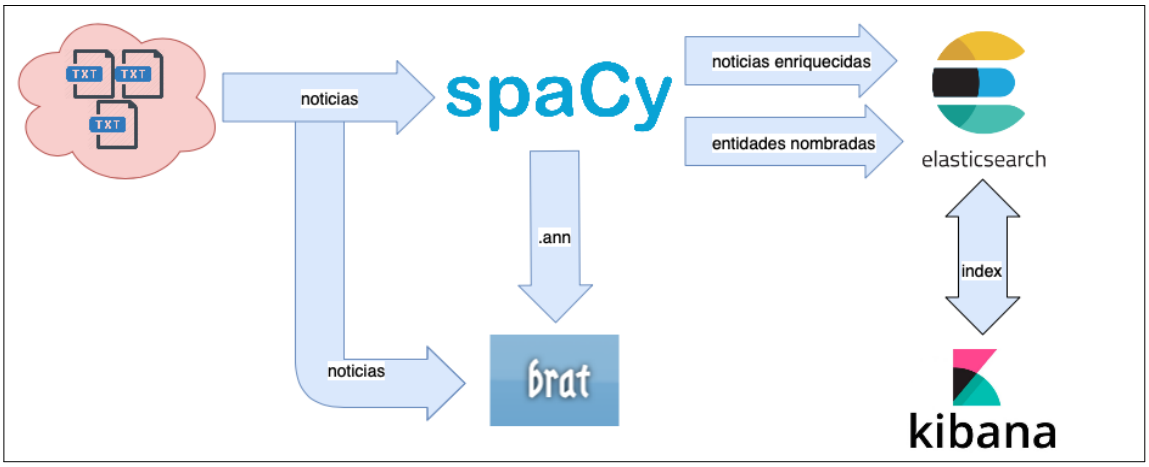

Figura 7: Diseño del prototipo 3

A la hora de llevar a cabo la visualización en Kibana, se propone el uso de un cuadro de mandos donde poder observar la distribución de los distintos tipos de entidad, frecuencias y noticias que nombren un determinado término. Por lo tanto, será necesarios los siguientes elementos de visualización:

1. Gráfico de tarta que represente la proporción de cada una de las tres categorías de entidad, y la proporción de las entidades más frecuentes de cada categoría.

2. Nube de palabras con las entidades más frecuentes, y donde el tamaño de las palabras dependa de la frecuencia de estas.

3. Gráfico de barras con las entidades más frecuentes.

4. Lista de noticias que contengan una determinada entidad.

5. Filtro de tipo de entidad para hacer una analítica más específica.

Por último, y como apoyo a la visualización en Kibana, se mostrará un grafo de puntos que representarán las noticias, y donde la distancia entre ellas determinará cómo de similares son esas noticias en función de las entidades compartidas. Además, será necesario detectar las comunidades dentro de este grafo, es decir, agrupar las noticias que compartan un número elevado de entidades, y poder distinguir estas comunidades visualmente mediante un código de colores. Este grafo se generará bajo demanda y para un intervalo de tiempo concreto, ya que de lo contrario sería demasiado costoso.

Para este caso, se podrían utilizar dos métricas para calcular la similitud entre listas de entidades nombradas: mediante el método de Jaccard o mediante la distancia coseno [24]. Aunque diferentes en el cálculo, ambos métodos comprueban cuántas entidades de una noticia se encuentran (con las mismas palabras) en la otra noticia.

Sin embargo, se propone ir un paso más allá y, en lugar de evaluar únicamente la coincidencia exacta de las entidades, usar la distancia entre las *word embeddings* de ambas listas de entidades. Para ello, Spacy dispone de unas funciones para el cálculo de similitud semántica entre vectores de palabras [25]. Esto hará que dos noticias que, por ejemplo, hablen sobre Donald Trump una, y sobre George Bush la otra, se encuentren semánticamente cerca aunque tengan entidades morfológicamente distintas.

3.4 Cuarta fase. Escalabilidad y procesamiento distribuido

Basándose en el prototipo anterior, la cuarta fase integrará el módulo ya existente de persistencia con los módulos de ingesta (Apache Kafka [26]) y de procesamiento distribuido (Apache Spark [27]). Esto además hará que la arquitectura sea horizontalmente escalable, pudiendo añadir más nodos y así paralelizar el procesamiento de las noticias y que el sistema esté preparado para procesar grandes volúmenes de datos.

Será necesario separar las colas Kafka de entrada (generación de noticias) de las de salida (entidades nombradas y noticias enriquecidas) para desacoplar Spark y Elasticsearch, ya que estos serán subsistemas independientes y las paradas de uno no afectarán al otro.

El siguiente diagrama muestra la arquitectura del cuarto prototipo:

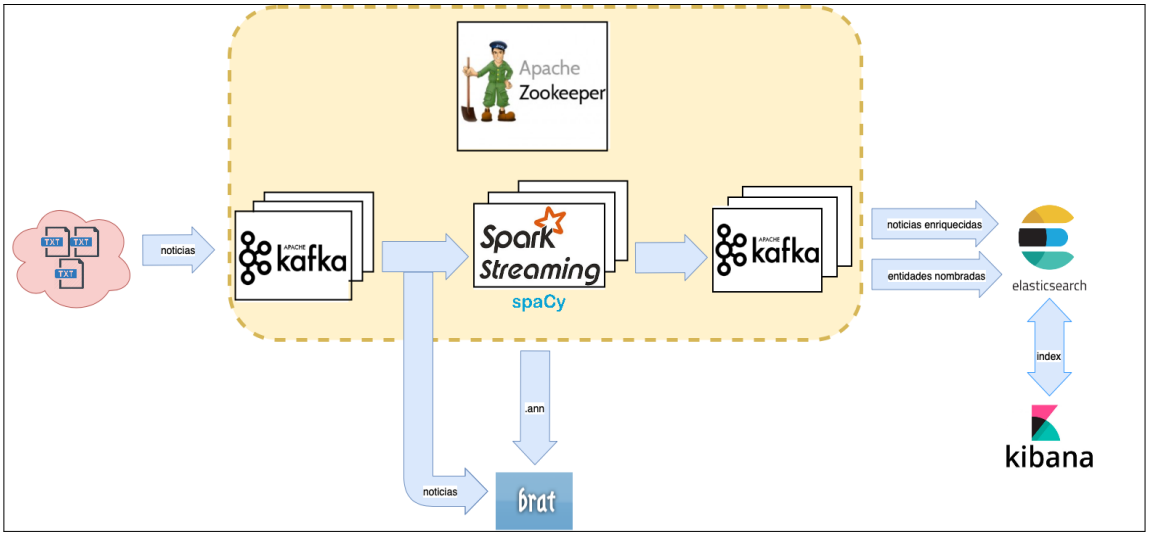

Figura 8: Arquitectura del sistema

Como en este caso no se dispone de una fuente de datos que distribuya noticias en tiempo real, se simulará con una distribución de Poisson, es decir el intervalo de espera entre la recepción de dos noticias será gobernado por un proceso de Poisson donde se ha decidido que la frecuencia media de recepción sea 5 segundos. Esto quiere decir que a medida que aumentamos el número de noticias, el tiempo medio transcurrido entre ellas se irá aproximando cada vez más a 5 segundos. El componente encargado de realizar esta labor será el *producer* de Kafka, que generará mensajes a la cola de noticias en crudo.

A medida que estas noticias se van persistiendo en Kafka, se irán consumiendo por el módulo de Spark Streaming que, mediante la librería de Spacy, irá procesando las noticias por orden de llegada. Una vez procesadas, este mismo módulo se encargará de producir mensajes a las dos colas siguientes de Kafka, una para las noticias enriquecidas, y otra para las entidades nombradas.

Los mensajes de estas dos colas se irán consumiendo por dos *consumers* de Kafka que serán los encargados de persistir la información en los índices correspondientes de Elasticsearch a medida que las noticias van siendo procesadas.

Aparte de Apache Spark y del Kafka *broker*, es importante destacar la presencia de Apache Zookeeper como responsable de la sincronización distribuida entre los distintos nodos del clúster.

4 Desarrollo

4.1 Primera fase. Evaluación de modelos pre-entrenados

Como se mencionó en la etapa de diseño 3.1, las noticias descargadas se encuentran en formato JSON junto con una serie de metadatos asociados. En este caso, solamente nos interesa el texto de la noticia por lo que se ha creado un script de shell para convertir las noticias en JSON a ficheros TXT. Para ello se ha utilizado la herramienta *jq* de línea de comandos específica para procesar ficheros JSON [28].

Una vez tenemos todas las noticias en formato TXT, el siguiente paso ha sido elegir el conjunto para el *Gold Standard*. Para ello se ha creado otro script de shell que, haciendo uso de los comandos *find* y *sort* de Unix, selecciona 100 noticias al azar. Estos 100 ficheros se han separado del resto del repositorio de noticias para evitar modificarlos ya que se usarán como test de referencia.

El paso siguiente ha consistido en cargar estas noticias TXT en la herramienta Brat Rapid Annotation Tool para etiquetar manualmente las entidades junto con su categoría. Este ha sido un trabajo laborioso pero que sin duda consigue aportar la máxima certeza al conjunto de test. Una vez etiquetadas las 100 noticias, Brat crea 100 ficheros de anotación en formato .ANN en el mismo directorio donde se encuentran las noticias sin anotar.

A partir de este momento, comienza el desarrollo en Python. Para este primer prototipo se ha trabajado con Jupyter Notebook sobre un entorno virtual de Conda (utilizado durante todo el proyecto) donde se ha instalado Python 3.7.3 y las siguientes librerías:

- BratReader (v1.0.1): Permite crear la estructura de objetos a partir de un repositorio de ficheros anotados en Brat (.ANN). Disponible en GitHub con licencia GPL [18] https://github.com/clips/bratreader [consulta: 17/06/2019]

- Spacy (v2.1.3): Permite realizar procesamiento de lenguaje natural en Python, incluyendo reconocimiento de entidades nombradas.

- Scikit-Learn (v0.20.3): Librería de machine learning que se ha utilizado para construir la matriz de confusión de las entidades detectadas [29].

- Matplotlib (v3.0.3): Se ha utilizado para representar gráficamente dicha matriz de confusión [30].

Aparte del *notebook* principal, se ha creado un módulo de Python (*evaluation.py*) con funciones auxiliares para la evaluación del *score*. Entre estas funciones, cabe destacar:

- *brat_to_spacy()*, *brat_to_num()*, *spacy_to_num()*: se encargan de la conversión entre categorías de entidad en Spacy y en Brat, ya que utilizan distinta terminología para las mismas categorías.

- *eval_ner()*: función principal que recorre todas las noticias comparando las entidades anotadas que han sido leídas por Bratreader y las que han sido detectadas por Spacy, y actualizando los contadores de los *true positives* (TP), *false positives* (FP) y *false negatives* (FN).

- *calc_score()*: función que a partir del número de TP, FP y FN, calcula la *precision, recall* y *f1-score.*

- *show_conf_matrix()*: función que muestra gráficamente la matriz de confusión generada por scikit-learn a partir de los vectores de entidades del *Gold Standard* y los detectados por el modelo de Spacy.

4.2 Segunda fase. Mejora del modelo

Tal y como se explicó en la fase de diseño 3.2, el primer paso para entrenar el modelo consiste en generar noticias de entrenamiento. Para ello, se han descargado varias listas de entidades de internet [31] y se han limpiado y unificado en un único fichero CSV. En definitiva, esta lista contiene 903 mil entidades, de las cuales 246 mil son lugares (países y ciudades de distintos tamaños), 40 mil organizaciones y 616 mil nombres de personas (en distinta forma y abreviaciones).

Con esta lista masiva de entidades, se construye el *PhraseMatcher* de Spacy [32], que consiste en una herramienta que busca coincidencias a nivel de token para todos los términos sobre los que se ha creado. Con estas coincidencias, se pueden generar automáticamente anotaciones de entidades nombradas, de la misma forma que se haría utilizando la herramienta Brat manualmente.

Aparta del conjunto de noticias anotadas automáticamente, y tal y como se describió en la sección 3.2, es necesario también incluir noticias de revisión en el conjunto de entrenamiento. Para generar este tipo de noticias, se utilizarán las predicciones del modelo original sobre el conjunto de noticias sin anotar.

Una vez tenemos formado el conjunto de entrenamiento (anotaciones sintéticas y de revisión), la siguiente figura muestra el proceso seguido para entrenar un modelo en Spacy. Este entrenamiento se realiza de manera online, es decir, se parte de un modelo pre-entrenado, y se va actualizando de forma iterativa con nuevos ejemplos.

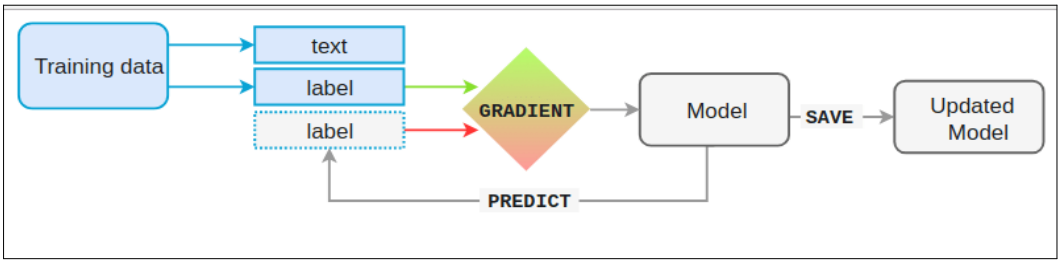

Figura 9: Pipeline de entrenamiento [33]

Durante el entrenamiento, ha sido necesario experimentar con distintos tamaños de *batch*, número de épocas y de tasa de *dropout*, hasta alcanzar una precisión aceptable. En este caso, y teniendo en cuenta los recursos computacionales disponibles, se ha utilizado un tamaño de *batch* bastante bajo para no sobrecargar el largo proceso de entrenamiento. Por otro lado, el método de *dropout* es necesario para que, en cada iteración, el modelo descarte noticias aleatorias de entrenamiento para evitar memorizar los datos. Y por último, también es importante mezclar las noticias en cada iteración (*shuffle*), ya que de lo contrario el modelo podría estar generalizando según el orden de los datos.

Aparte del *notebook* principal y de las librerías ya instaladas en la fase anterior 4.1, se ha creado un nuevo módulo de Python (*training.py*) con funciones auxiliares para el entrenamiento del modelo. Entre estas funciones, cabe destacar:

- *match_entities()*: construye el *matcher* de Spacy con las entidades del repositorio. Se duplica la lista de las entidades para incluir minúsculas y mayúsculas, ya que el *PhraseMatcher* es *case-sensitive*.

- *generate_data()*: genera anotaciones sintéticas a partir del *matching* por reglas y la categoría de las entidades en el fichero CSV. Se descartan las noticias con un numero demasiado alto de coincidencias para evitar saturar el entrenamiento.

- *revise_data()*: crea datos de entrenamiento con las predicciones del modelo original sobre nuevas noticias. Es importante usar las predicciones del modelo pre-entrenado original, y no del que está siendo entrenado.

- *build_training_data()*: separa las noticias de entrenamiento sin anotar en dos conjuntos, uno para revisar y otro para generar anotaciones sintéticas, según la proporción especificada mediante el *revision_rate*.

- *update_model():* actualiza el modelo de Spacy con los dos conjuntos de entrenamiento y con el número de épocas, *dropout* y tamaño de *batch* escogidos.

Además, se ha reutilizado el módulo *evaluation.py* implementado en la primera fase 4.1 para evaluar el modelo actualizado.

4.3 Tercera fase. Persistencia y visualización

Una vez creados los índices de la Tabla 2 en Elasticsearch, ya sería posible empezar a subir documentos en formato JSON. Para ello, se itera el modelo actualizado del prototipo 2 sobre las noticias del repositorio que no han sido utilizadas para el *Gold Standard* o el entrenamiento. A medida que se van procesando noticias, se van indexando documentos en ES.

En cuanto a la parte de visualización, esta se divide en dos tareas: una consiste en visualizar los datos indexados en tiempo real en Kibana (véase sección 5.3), mientras que la otra consiste en la generación de un grafo de distancias entre noticias indexadas.

Es importante destacar que se ha usado un modelo de lenguaje diferente para calcular la similitud entre vectores de entidades. El motivo de esta decisión es que los modelos de Spacy en español no están entrenados con una cantidad de datos tan extensa como sí lo están los modelos en inglés. En concreto, el modelo *en_vectors_web_lg* ha sido entrenado con GloVe [34] y es específico para *word embeddings*. Teniendo en cuenta que los lugares, personas y organizaciones son en su mayoría iguales en ambos idiomas, se ha decidido utilizar este modelo, con resultados aceptables [35].

Para este prototipo, además de los paquetes de Python utilizados para los prototipos anteriores (véase sección 4.1), se han instalado las siguientes librerías:

- Elasticsearch-py (v6.3.1): Aporta una API de Python para Elasticsearch, facilitando la indexación y búsqueda mediante acciones individuales o en lotes [36].

- NetworkX (v2.3): Permite crear grafos y aporta distintas funciones para manipular tanto los nodos como las aristas [37].

- Python-Louvain (v0.13): Permite detectar comunidades dentro de un grafo utilizando el método de Louvain [38].

Aparte del *notebook* principal para este prototipo, se han creado tres nuevos módulos de Python: *persistence.py*, *visualization.py* y *news_graph.py*. El primero es específico para indexar los datos en ES, mientras que los otros dos se encargan de crear el grafo de noticias bajo demanda y detectar las comunidades. Entre las funciones de código de estos módulos, cabe destacar:

- *process_news()*: detecta entidades nombradas en las noticias, genera ficheros JSON y devuelve la lista de acciones para indexar en ES.

- *get_similarity()*: calcula la similitud entre dos vectores de entidades usando *word embeddings*. El resultado es un valor entre 0 y 1 (máxima similitud).

- *find_doc_distances():* calcula la matriz de distancias entre cada par de documentos, usando la similitud entre sus vectores de entidades.

- *make_graph()*: crea el grafo de documentos utilizando la técnica de *multi-dimensional scaling* [39] para encontrar las coordenadas de los puntos según la matriz de distancias. A mayor similitud entre documentos mayor peso se le asigna a la arista entre los dos puntos. Además detecta las comunidades y las representa mediante código de colores.

- *find_entity_frequency()*: mediante una estrategia tipo TF-IDF, filtra las entidades más representativas y diferenciadoras de cada comunidad para incluirlas en la leyenda del grafo.

- *news_graph()*: script que recibe de entrada un número de minutos sobre los que consultar las noticias. Con el objetivo de no volver a calcular la similitud entre dos noticias ya conocidas (proceso costoso), almacena la matriz de distancias ya conocidas en un fichero CSV para usarlo en posteriores versiones del grafo.

4.4 Cuarta fase. Escalabilidad y procesamiento distribuido

Para generar las noticias siguiendo una distribución de Poisson se ha seguido una función exponencial, mediante la cual, a medida que transcurre el tiempo, la probabilidad de no recibir una noticia decae hacia 0, y por lo tanto la probabilidad de recibir al menos una noticia incrementa hacia 1 [40]. La siguiente función calcula el tiempo de espera hasta la siguiente noticia, teniendo en cuenta que *rateparameter* es 1/5 ya que se ha asumido que una noticia suele ocurrir cada 5 segundos:

```
import math
import random

def nextTime(rateParameter):
    return -math.log(1.0 - random.random()) / rateParameter
```

Figura 10: Función para distribución de Poisson

A continuación se muestra el *pipeline* del prototipo 4:

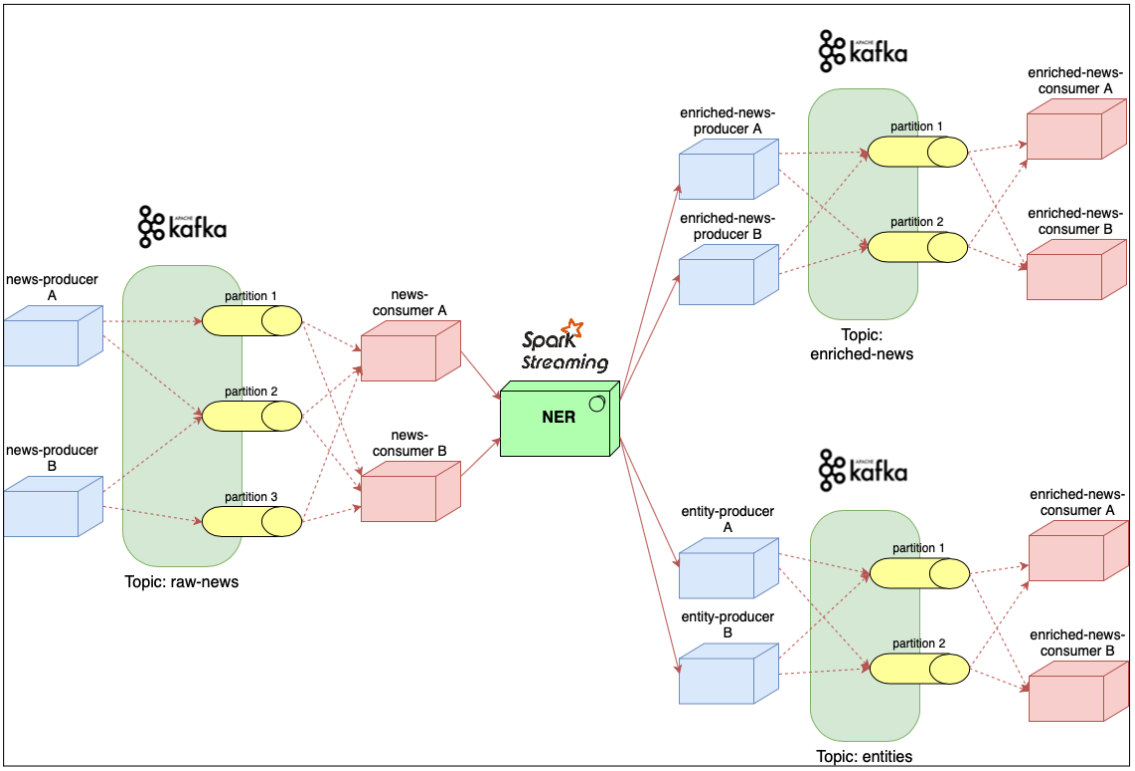

Figura 11: Pipeline del prototipo 4

Antes de crear los *topics*, es necesario arrancar Zookeeper y el *broker* de Kafka, con los siguientes comandos:

- ./zookeeper-server-start.sh config/zookeeper.properties
- ./kafka-server-start.sh config/server.properties

Aunque en modo clúster podemos crear los *topics* con varias particiones y un factor de replicación múltiple, en este caso se han configurado para una única partición y un factor de replicación de 1. Los comandos utilizados para crear y listar los *topics* son:

- ./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic raw-news
- ./kafka-topics.sh --list --zookeeper localhost:2181

Para este prototipo, además de los paquetes de Python utilizados para los prototipos anteriores (véase secciones 4.1 y 4.3), se han instalado las siguientes librerías:

- kafka-python (v1.4.6): Cliente Python para Apache Kafka. Se utiliza para producir y consumir mensajes de las colas Kafka [41].

- pyspark (v2.2.1): API de Python para Apache Spark. Se utiliza para crear el *stream* para el consumo y procesamiento distribuido de las noticias [42].

Este prototipo se divide en cuatro módulos de código claramente diferenciados:

1. *news_producer.py*: encargado de generar las noticias al Kafka *topic* "raw-news".

2. *spark_streaming_ner.py*: consume las noticias del *topic* "raw-news", las procesa con el modelo de Spacy y produce mensajes a las colas "enriched-news" y "entities".

3. *enriched_news_consumer.py*: encargado de consumir los mensajes del *topic* "enriched-news" e indexarlas en ES.

4. *entity_consumer.py*: consume los mensajes del *topic* "entities" y las indexa en ES.

5 Integración, pruebas y resultados

5.1 Primera fase. Evaluación de modelos pre-entrenados

Al ejecutar el *notebook* Python desarrollado en la fase 4.1 sobre el conjunto de noticias del *Gold Standard*, se han obtenido los siguientes resultados:

Métrica	**Score**
Precision	0.565
Recall	0.617
F1-Score	0.590

Tabla 3: Score del modelo pre-entrenado

Por lo tanto, el modelo pre-entrenado de Spacy ha obtenido un 0.481 de *f1-score* para el conjunto *Gold Standard*, lo cual parece bastante mejorable. Se ha alcanzado mayor *recall* (0.595) que *precision* (0.404), lo que indica que se han reconocido la mayoría de entidades efectivamente nombradas, pero también que se han marcado bastantes términos como entidades nombradas cuando en realidad no lo son (falsos positivos).

Con el objetivo de analizar más exhaustivamente los resultados obtenidos por cada una de las tres categorías, a continuación se muestra la matriz de confusión, tanto absoluta como normalizada:

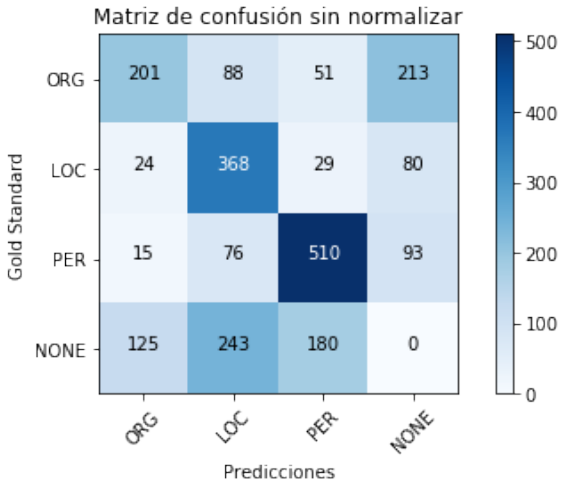

Figura 12: Matriz de confusión absoluta

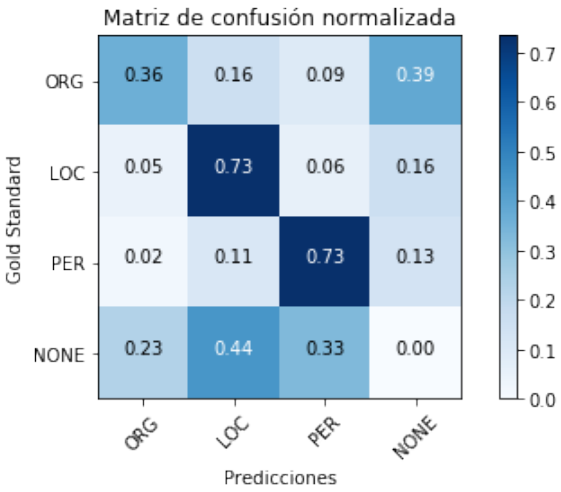

Figura 13: Matriz de confusión normalizada

De las matrices anteriores se pueden extraer las siguientes conclusiones:

1. El 44% de los falsos positivos se han reconocido como lugares.

2. El resultado es aceptable para las entidades nombradas de tipo persona y lugar (73% en ambos casos).

3. La mayoría de organizaciones del *Gold Standard* (64%) no han sido detectadas por el modelo.

4. De las organizaciones no detectadas como tal, la mayoría ni siquiera han sido detectadas como entidades nombradas.

5. El modelo ha confundido los tipos de algunas entidades, principalmente al reconocer organizaciones y personas como lugares (16% y 11%, respectivamente).

En cualquier caso, aunque el rendimiento es bastante mejorable, cabe aclarar que no es tan negativo teniendo en cuenta que este modelo de lenguaje no fue entrenado con un dominio de noticias como este, sino con artículos de Wikipedia.

5.2 Segunda fase. Mejora del modelo

Tras el entrenamiento descrito en la sección 4.2, se han obtenido los siguientes resultados en la detección de entidades:

Métrica	**Modelo pre-entrenado**	**Modelo actualizado**
Precision	0.565	0.588 **(+2%)**
Recall	0.617	0.637 **(+2%)**
F1-Score	0.590	0.611 **(+2%)**

Tabla 4: Comparativa de scores

Como se obtiene de la tabla anterior, el entrenamiento ha mejorado levemente el *score* del modelo en cada una de las tres métricas. Para analizar esta mejora general en más detalle, el siguiente diagrama muestra cómo el *score* ha aumentado en 4 puntos porcentuales para las categorías ORG y PER, mientras que ha bajado en 2 puntos para la categoría LOC.

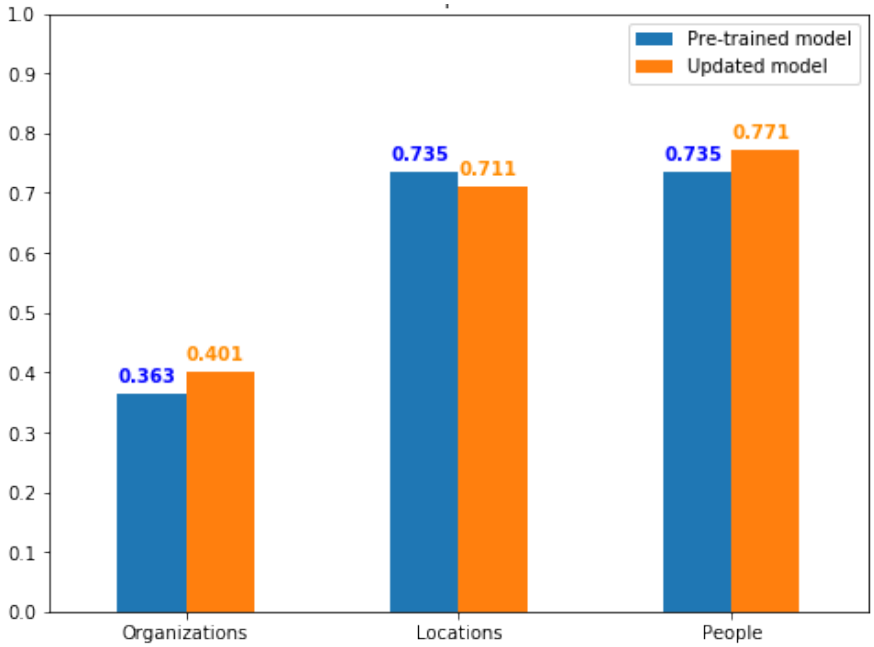

Figura 14: F1-score por categoría

Por último, podemos representar mediante una matriz de confusión normalizada qué categorías son las que más confunde el modelo y en cuáles la detección es más precisa:

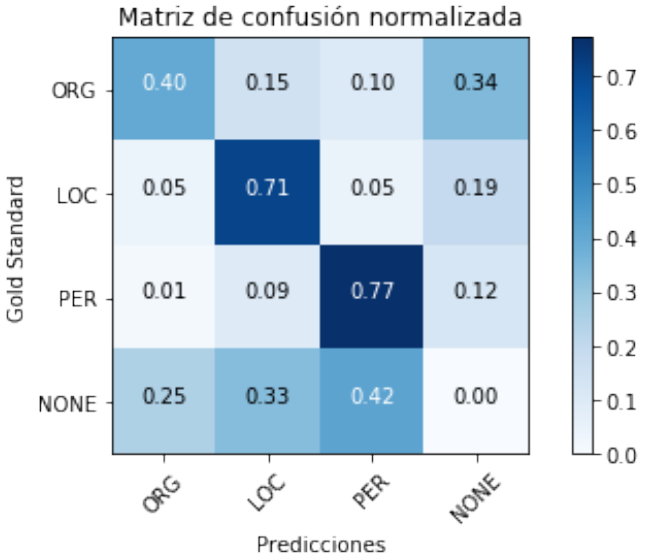

Figura 15: Matriz de confusión (modelo actualizado)

Comparando esta matriz con la obtenida para el modelo original pre-entrenado (véase sección 5.1), se observa que las principales mejoras se dan en las categorías de organizaciones y de personas, a pesar de haber aumentado el número de falsos positivos de personas. Por otro lado, aunque se han reducido el número de falsos positivos de

lugares, se ha reducido ligeramente el *score* de esta categoría. En general, el entrenamiento ha mejorado el modelo, pero sin duda hay margen para una mayor mejora siempre y cuando se emplee un conjunto mayor de entrenamiento y se depure el listado de entidades descargadas de internet.

5.3 Tercera fase. Persistencia y visualización

Una vez los datos están indexados en ES (véase sección 4.3), ya están disponibles para la visualización en el cuadro de mandos de Kibana. También es importante remarcar que existe un filtro en el cuadro de mandos para únicamente visualizar las entidades de una categoría determinada. A continuación se muestran los cuatro elementos del cuadro de mandos con las noticias y entidades que han sido indexadas hasta este momento.

La siguiente imagen muestra el aspecto del cuadro de mandos, con un elemento distinto en cada cuadrante. En este caso, sin aplicar filtros de categoría de entidad, podemos ver que las personas y lugares son bastante más frecuentes que las organizaciones (el doble). En la nube de palabras vemos que España (de tipo LOC) y PSOE (de tipo ORG) son las entidades más repetidas, hecho que se refleja también en el diagrama de barras.

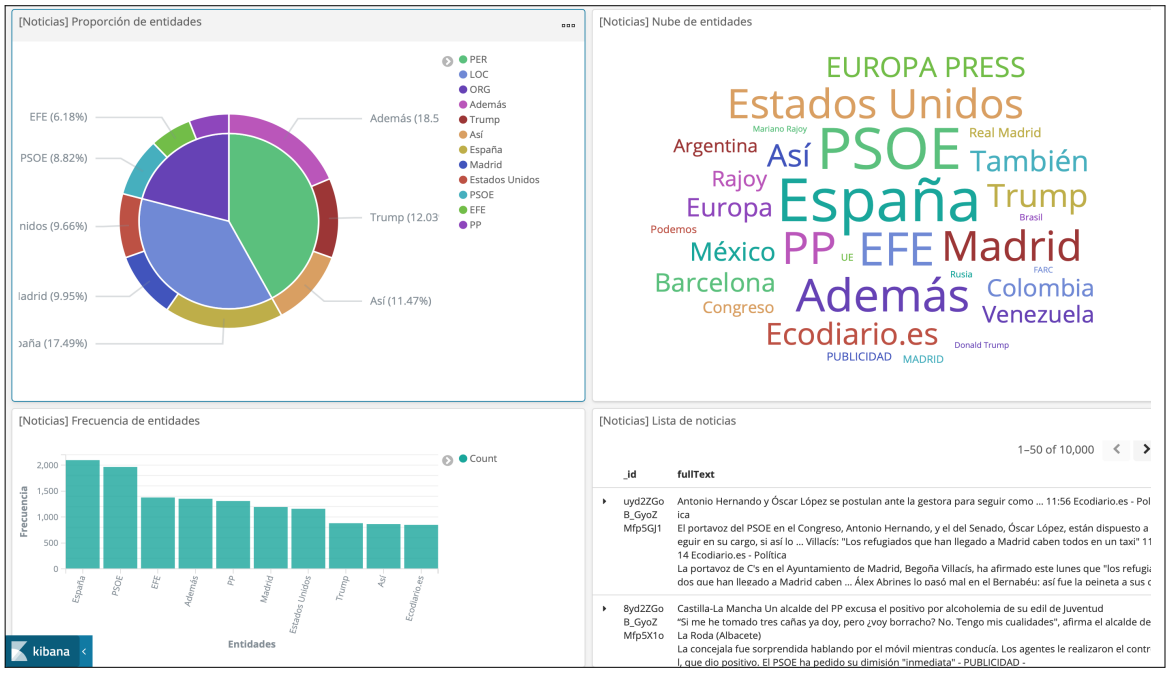

Figura 16: Cuadro de mandos

En el siguiente gráfico de tarta se ha aplicado un filtro para únicamente mostrar las entidades de tipo LOC, y limitando a 10 el número de lugares a mostrar. España, Madrid y Estados Unidos (por este orden) ocupan la mayor proporción de entidades de tipo lugar.

En especial, cabe remarcar que 1 de cada 5 lugares que se mencionan en las noticias se refieren a España, lo cual es comprensible teniendo en cuenta que son noticias en español.

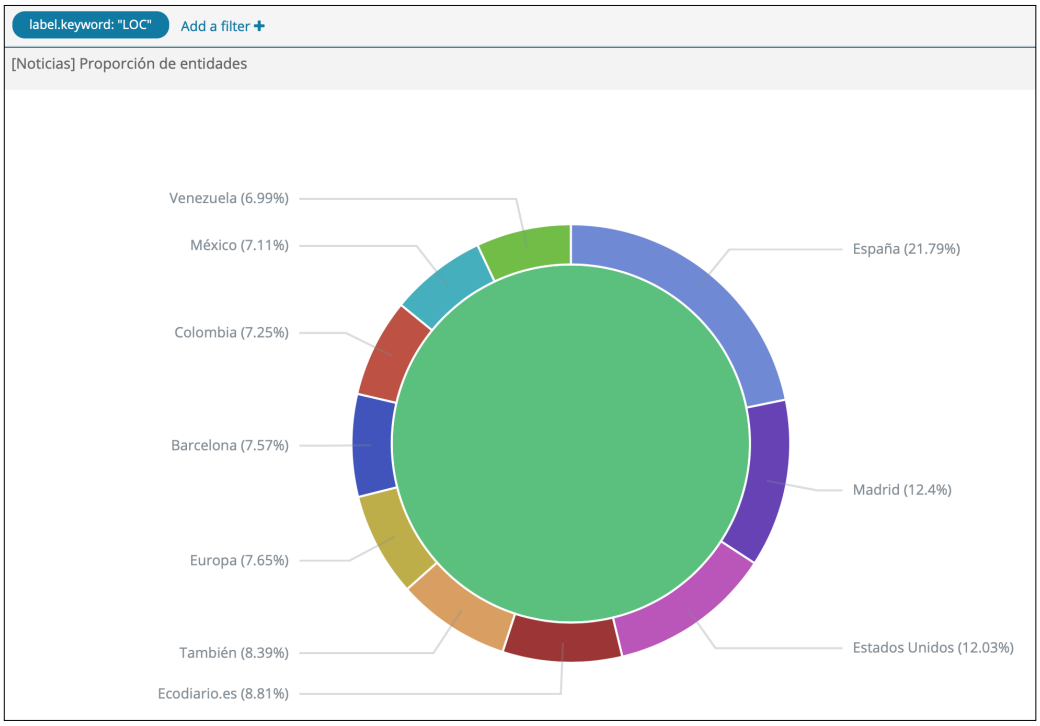

Figura 17: Proporción de entidades y categorías

Para la siguiente nube de entidades se han filtrado las entidades PER, y aunque hay personas esperadas como Donald Trump y Mariano Rajoy, también aparecen palabras como "Así" y "Además" que se han detectado incorrectamente. Aunque Kibana puede ignorar *stopwords* en las representaciones, se ha decidido mantenerlas para así ver que el modelo entrenado sigue siendo mejorable a la hora de reducir los falsos positivos.

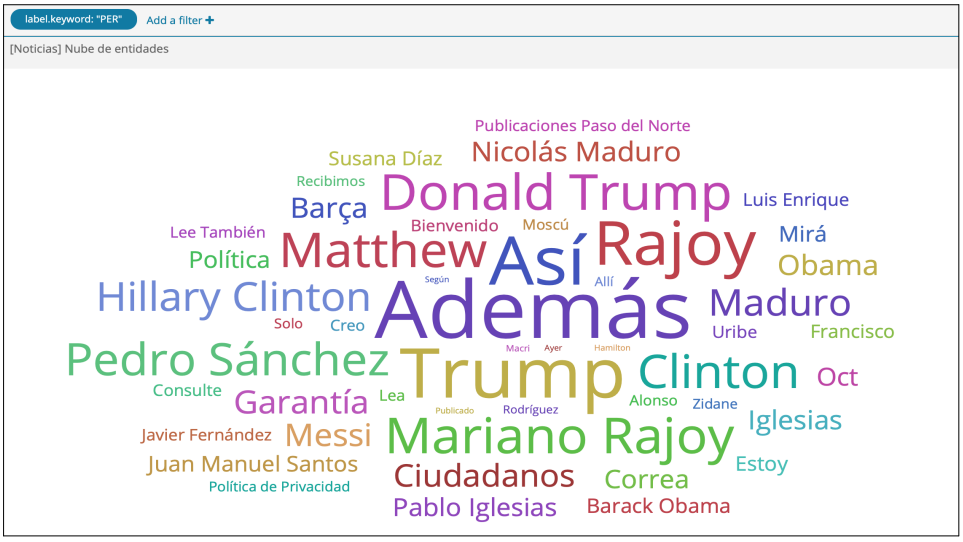

Figura 18: Nube de palabras

El siguiente diagrama de barras muestra las 10 organizaciones más frecuentes del conjunto de noticias, con los partidos políticos en las primeras posiciones. También aparece la agencia EFE ya que muchas de las noticias provienen de esta fuente.

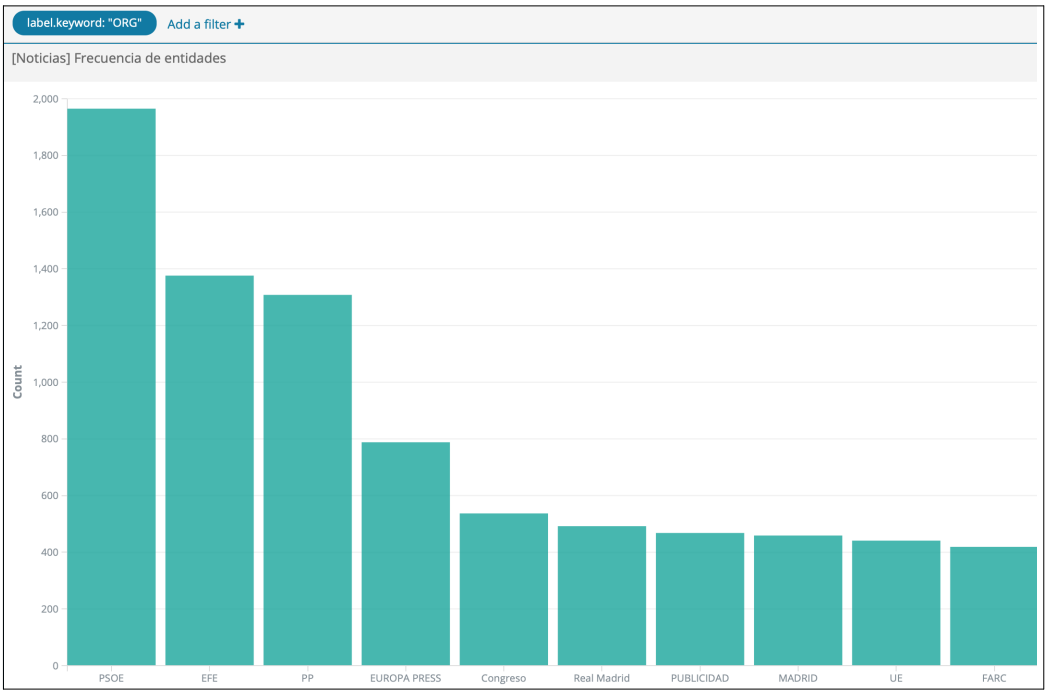

Figura 19: Frecuencia de entidades

Por último, se muestra el resultado de una búsqueda de entidad (Obama), para verificar que el sistema es capaz de encontrar las noticias que incluyen un determinado término.

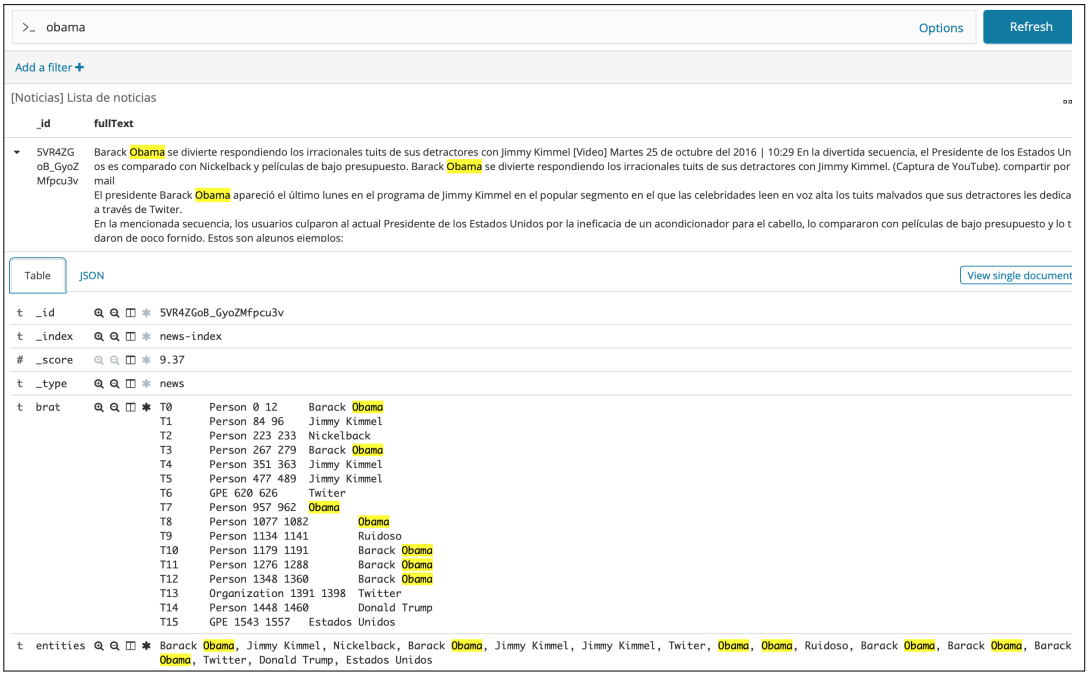

Figura 20: Lista de noticias

Con el objetivo de detectar posibles comunidades de noticias, se ha creado un grafo donde se representan 10,000 noticias posicionadas según la similitud entre ellas. En este caso, se han detectado tres principales comunidades asociadas a las entidades más representativas que aparecen en la leyenda.

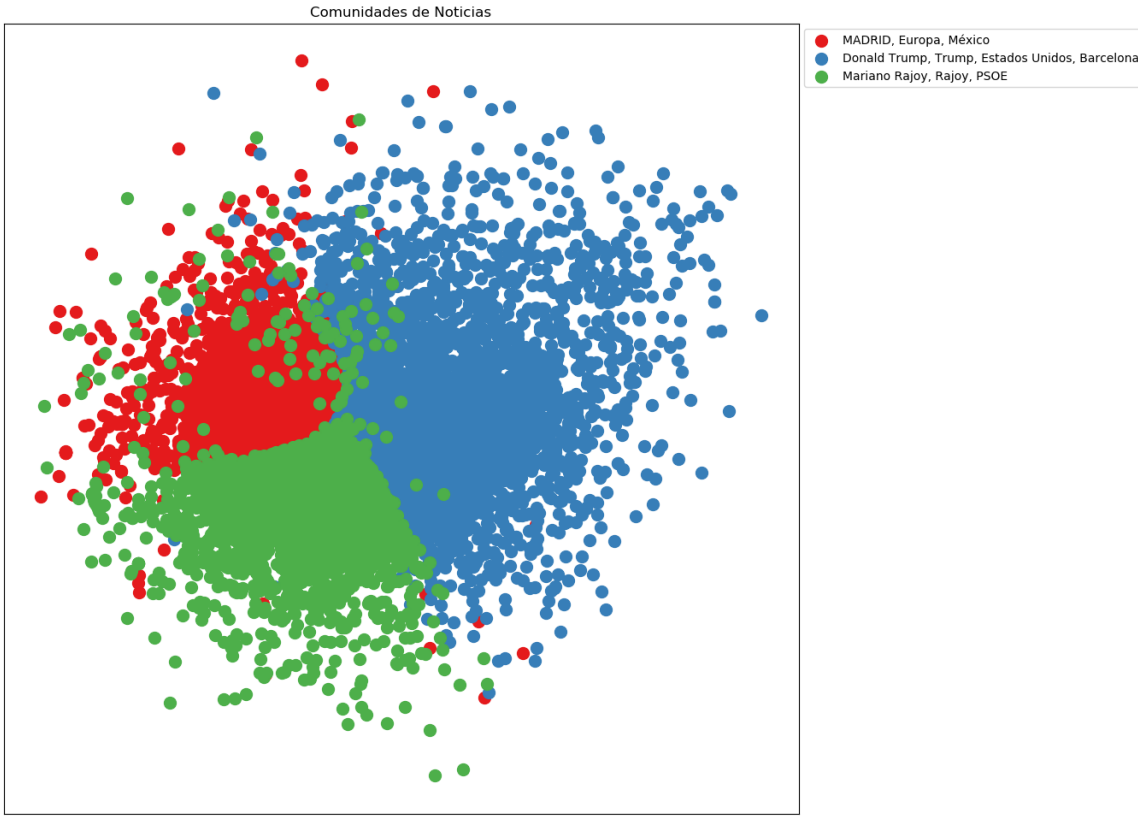

Figura 21: Grafo de noticias

Del diagrama anterior, y a falta de un análisis más detallado, se podría extraer que existen tres grandes comunidades de noticias, una que trata sobre diversos países en general, otra sobre la política americana y otra sobre la política española. También hay que tener en cuenta que las noticias que pertenecen a comunidades cuyo tamaño es inferior al 5% del total, no se muestran en el grafo ya que no serían lo suficientemente significativas. En concreto, hay dos comunidades con aproximadamente 100 noticias cada una que no han alcanzado el umbral mínimo y por tanto no se representan en el grafo.

Es importante tener en cuenta que el algoritmo de generación de comunidades no es determinista y por tanto puede obtener resultados ligeramente distintos en distancias ejecuciones. Por último, se ha obtenido una baja modularidad (0.03), posiblemente debido al hecho de que las noticias son muy dispares en cuanto a las entidades nombradas.

5.4 Cuarta fase. Escalabilidad y procesamiento distribuido

Una vez implementado el prototipo 4 según la sección 4.4, a continuación se describen los resultados.

La siguiente captura muestra los mensajes que se van consumiendo de cada una de las tres colas Kafka. La ventana de la izquierda muestra una noticia en crudo, la ventana superior derecha muestra la misma noticia enriquecida (junto con la lista de entidades detectadas) y la ventana inferior derecha muestra las características de cada entidad (nombre, categoría y posición):

Figura 22: Mensajes de las colas Kafka

A medida que las noticias se van procesando por Spark Streaming, las entidades se van incluyendo en la visualización del cuadro de mandos de Kibana, y también en la herramienta de anotación Brat. Además, tenemos la opción de refrescar el cuadro de mandos en Kibana cada varios segundos para así incluir los nuevos documentos que van siendo indexados.

Con el objetivo de evaluar el sistema durante un largo periodo de tiempo, se ha ejecutado durante 4 horas siguiendo una generación de noticias con distribución de Poisson con tasa de ocurrencia λ=12 noticias/minuto, consiguiendo indexar 2,585 noticias y 54,636 entidades nombradas. La siguiente imagen muestra el contador de documentos indexados para cada uno de los dos índices:

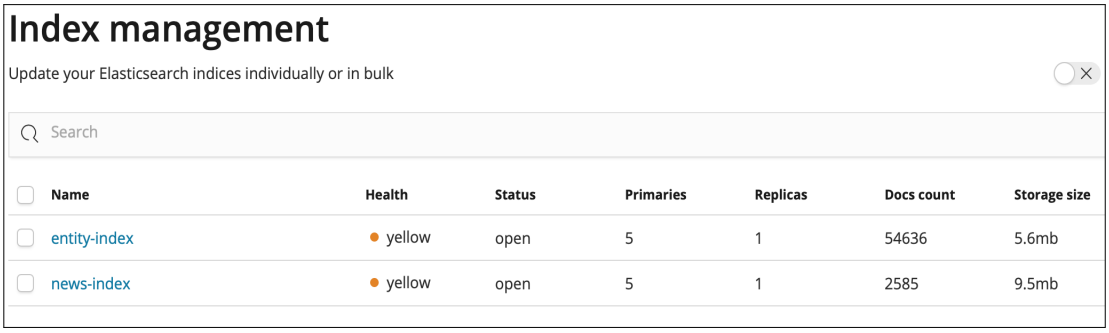

Figura 23: Contador de documentos indexados en ES

5.5 Integración del sistema

Tanto el desarrollo del sistema como las pruebas se han realizado en un entorno virtual de Conda con las siguientes características:

- MacBook Pro 2,2 GHz Intel Core i7, 16GB 1600 MHz DDR3
- Apache Zookeeper 3.4.13
- Apache Kafka 2.2.1
- Apache Spark 2.2.1
- Elasticseach, Kibana 6.5.1
- Brat Rapid Annotation Tool 1.3
- Python 3.7.3

Para facilitar el arranque y parada del sistema, se ha construido un script de shell que se encarga de lanzar todos los procesos necesarios, siguiendo el orden correcto y con las correspondientes esperas entre ellos. También se encarga de parar todos los procesos al leer de teclado la combinación Ctrl+C. A continuación se listan estos procesos:

1. ./zookeeper-server-start.sh config/zookeeper.properties
2. ./kafka-server-start.sh config/server.properties
3. ./elasticsearch
4. ./kibana
5. python brat-v1.3_Crunchy_Frog/standalone.py
6. python enriched_news_consumer.py
7. python entity_consumer.py
8. spark-submit --packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.2.1 --archives scripts/doc-similarity-env.tar.gz#environment spark_streaming_ner.py raw-news
9. python news_producer.py

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este TFM se ha mejorado un modelo de lenguaje, que se ha integrado dentro un sistema reconocedor de entidades nombradas en noticias en español en tiempo real. Además, se han representado visualmente los resultados mediante diferentes diagramas y se ha construido un grafo de similitud entre noticias.

A pesar de haber mejorado la precisión del modelo, se han detectado en la visualización algunos términos incorrectamente detectados. Esta mejora habría sido más significativa de haber solucionado las dos grandes limitaciones del entrenamiento, que son los recursos computacionales y el tiempo necesario para entrenar con un gran número de noticias.

El análisis estadístico confirma que las entidades nombradas más frecuentes son las que se podrían haber esperado de este tipo de noticias. Por último, el grafo de noticias aporta información valiosa al mostrar las distintas comunidades que existen con sus entidades más representativas.

Por lo tanto, se considera que se han alcanzado los principales objetivos de este trabajo:

1. Se ha extraído información relevante de un conjunto de noticias para su posterior visualización.

2. Se ha desarrollado un algoritmo de distancias y detección de comunidades para visualizar las noticias en modo grafo.

3. El sistema construido es escalable horizontalmente y por lo tanto permite paralelizar el procesamiento de grandes volúmenes de noticias.

4. La persistencia escogida permite recuperar todas las noticias que comparten una determinada entidad nombrada.

5. Es posible validar visualmente la precisión del sistema mediante el uso de una herramienta de anotación.

6.2 Trabajo futuro

En cuanto a la ingesta de noticias, una posible mejora consistiría en reemplazar la generación de noticias del disco local por la escucha directa de una fuente de datos. En concreto, se podría utilizar Kafka Connect para escuchar de distintos canales RSS y de este modo poder ir procesando y visualizando las noticias a media que van apareciendo [43].

Una de las partes a mejorar de este sistema es el entrenamiento del *NER* de Spacy. Una forma sencilla pero costosa en tiempo sería anotar manualmente una gran cantidad de noticias utilizando la misma herramienta Brat que se utilizó para crear el *Gold Standard*. Otra estrategia de entrenamiento sería utilizar Word2vec para encontrar los términos más similares a las entidades nombradas ya conocidas [44]. De esta forma podríamos extender la lista de entidades descargadas de internet y volver a utilizar la herramienta de PhraseMatcher de Spacy para generar más ejemplos sintéticos de entrenamiento.

En lo referente al grafo de noticias, un futuro trabajo podría consistir en hacer el grafo interactivo de tal manera que haciendo click o poniendo el ratón sobre uno de los puntos se muestre la noticia en cuestión, y así poder validar visualmente que las noticias vecinas son realmente similares en cuanto a sus entidades nombradas. Para esta tarea se podría utilizar Vega, una librería de código abierto basada en D3.js para generar diagramas interactivos mediante el lenguaje JSON, y que se puede integrar como una nueva visualización dentro de Kibana [45].

Además, se podría hacer más preciso el cálculo de similitud entre noticias si tenemos en cuenta el tema que se trata en ellas, es decir, si son noticias de política, deportes, actualidad, etc. Para detectar el tema se podría incorporar una red LSTM en Keras al *pipeline* del modelo en Spacy [46]. De esta forma, la similitud entre un par de noticias no dependería únicamente de la similitud entre los *word embeddings* de los vectores de entidades, sino que también tendría en cuenta (quizás al 50%) el tema detectado por la red neuronal. Como ejemplo, no parece razonable que una noticia sobre el Real Madrid y otra sobre la empresa ACS se encuentren cercanas en el grafo por el único hecho de que ambas mencionan a Florentino Pérez o la ciudad de Madrid.

Por último, un trabajo futuro debería incluir la arquitectura basada en contenedores. Esto permitiría abstraerse de los diferentes entornos y sistemas operativos, y automatizar el despliegue de nuestro sistema en los distintos nodos del clúster. Para ello, se propone el uso de Docker, un proyecto de código abierto que ya dispone de imágenes para contenedores con Elasticsearch y Kibana [47].

Referencias

[1] https://ethw.org/The_History_of_Natural_Language_Processing [consulta: 15/06/2019]

[2] https://jalammar.github.io/illustrated-bert/ [consulta: 16/06/2019]

[3] https://rajpurkar.github.io/SQuAD-explorer/ [consulta: 16/06/2019]

[4] Erik F. Tjong Kim Sang, "Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition", University of Antwerp, 2002

[5] Jason P.C. Chiu, Eric Nichols, "Named Entity Recognition with Bidirectional LSTM-CNNs", University of British Columia, Honda Research Institute Japan, Julio 2016

[6] Dylan Baker, "The Document Similarity Network: A Novel Technique for Visualizing Relationships in Text Corpora", thesis, Harvey Mudd College, Mayo 2017

[7] Wael H. Gomaa, Aly A. Fahmy, "A Survey of Text Similarity Approaches", Cairo University, Abril 2013

[8] Leonidas Tsekouras, Iraklis Varlamis, George Giannakopoulos, "A Graph-based Text Similarity Measure That Employs Named Entity Information", NCSR Demokritos, Harokopio University of Athens, 2017

[9] Steven Bird, Ewan Klein, Edward Loper, "Natural Language Processing with Python", O'Reilly, Junio 2009

[10] François Chollet, "Deep Learning with Python", Manning Publications Co., 2018

[11] https://www.globenewswire.com/news-release/2019/03/26/1767348/0/en/The-Apache-Software-Foundation-Celebrates-20-Years-of-Community-led-Development-The-Apache-Way.html [consulta: 16/06/2019]

[12] https://www.edureka.co/blog/hadoop-ecosystem [consulta: 16/06/2019]

[13] https://spacy.io [consulta: 15/06/2019]

[14] https://spacy.io/models/es [consulta: 17/06/2019]

[15] https://spacy.io/usage/linguistic-features [consulta: 17/06/2019]

[16] https://webhose.io/free-datasets/spanish-news-articles/ [consulta: 17/06/2019]

[17] http://brat.nlplab.org [consulta: 17/06/2019]

[18] https://github.com/clips/bratreader [consulta: 17/06/2019]

[19] Anthony Robins, "Catastrophic Forgetting, Rehearsal and Pseudorehearsal", University of Otago, 1995

[20] https://www.elastic.co/products/elasticsearch [consulta: 15/06/2019]

[21] https://www.elastic.co/products/kibana [consulta: 15/06/2019]

[22] https://www.elastic.co/guide/en/elasticsearch/guide/current/denormalization.html [consulta: 21/06/2019]

[23] https://brat.nlplab.org/standoff.html [consulta: 18/06/2019]

[24] https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50 [consulta: 18/06/2019]

[25] https://spacy.io/usage/vectors-similarity [consulta: 18/06/2019]

[26] https://kafka.apache.org [consulta: 15/06/2019]

[27] https://spark.apache.org [consulta: 15/06/2019]

[28] https://stedolan.github.io/jq/ [consulta: 19/06/2019]

[29] https://scikit-learn.org [consulta: 19/06/2019]

[30] https://matplotlib.org [consulta: 19/06/2019]

[31] https://ec.europa.eu/jrc/en/language-technologies/jrc-names [consulta: 20/06/2019] y https://datahub.io/core/world-cities#resource-world-cities [consulta: 20/06/2019]

[32] https://spacy.io/usage/rule-based-matching#phrasematcher [consulta: 20/06/2019]

[33] https://spacy.io/usage/training [consulta: 20/06/2019]

[34] Jeffrey Pennington, Richard Socher, Christopher D. Manning, “GloVe: Global Vectors for Word Representation”, Stanford University, 2014

[35] https://spacy.io/models/en#en_vectors_web_lg [consulta: 21/06/2019]

[36] https://elasticsearch-py.readthedocs.io [consulta: 21/06/2019]

[37] https://networkx.github.io [consulta: 21/06/2019]

[38] https://python-louvain.readthedocs.io [consulta: 21/06/2019]

[39] https://scikit-learn.org/stable/modules/manifold.html#multi-dimensional-scaling-mds [consulta: 21/06/2019]

[40] https://preshing.com/20111007/how-to-generate-random-timings-for-a-poisson-process/ [consulta: 23/06/2019]

[41] https://github.com/dpkp/kafka-python [consulta: 23/06/2019]

[42] https://spark.apache.org/docs/2.2.0/api/python/pyspark.html [consulta: 23/06/2019]

[43] https://github.com/kaliy/kafka-connect-rss [consulta: 23/06/2019]

[44] https://www.kaggle.com/jihyeseo/word2vec-gensim-play-look-for-similar-words [consulta: 23/06/2019]

[45] https://www.elastic.co/blog/getting-started-with-vega-visualizations-in-kibana [consulta: 23/06/2019]

[46] https://explosion.ai/blog/spacy-deep-learning-keras [consulta: 23/06/2019]

[47] https://www.docker.elastic.co [consulta: 27/06/2019]

Anexos

Anexo 1: Código fuente

El código fuente de este trabajo se encuentra alojado en el siguiente repositorio de Github:

- https://github.com/ejmacias/tfm.git

Estructura del repositorio:

- **/src**: módulos de Python y notebooks de los cuatro prototipos.
    - /prototype1: fase de evaluación del modelo pre-entrenado.
    - /prototype2: fase de entrenamiento y mejora del modelo.
    - /prototype3: fase de persistencia y visualización.
    - Nota: los ficheros del prototipo 4 son los necesarios para el sistema final y se encuentran en la raiz de /src.
- **/scripts**: scripts de shell de apoyo utilizados durante el desarrollo.
- **environment.yml**: entorno Conda para instalación de librerías necesarias.

Anexo 2: Manual de instalación

1. Se requiere la instalación previa de las siguientes herramientas:

   i. **Brat Rapid Annotation Tool**: https://brat.nlplab.org/installation.html

      - Crear directorio: <brat path>/data/ner_output/

      - Nota: Brat únicamente soporta Python 2.

   ii. **Bratreader**: https://github.com/clips/bratreader

   iii. **Elasticsearch**: https://www.elastic.co/downloads/elasticsearch

   iv. **Kibana**: https://www.elastic.co/downloads/kibana

   v. **Apache Kafka**: https://kafka.apache.org/downloads

   vi. **Apache Spark**: https://spark.apache.org/downloads.html

2. Descargar el código del repositorio de Github.

3. Crear entorno virtual a partir del fichero *environment.yml* con la siguiente instrucción:

   - conda env create -f environment.yml

4. Empaquetar el entorno en un fichero comprimido para poder ser adjuntado posteriormente al comando *spark-submit*, con la siguiente instrucción:

   - conda pack -n my_env -o out_name.tar.gz

5. Actualizar las rutas de los comandos del fichero:

   - */scripts/launch_ner.sh*

6. Crear el directorio **/data_rep**, donde descargar y descomprimir las noticias en formato JSON de la siguiente web:

   - https://webhose.io/free-datasets/spanish-news-articles/

7. Ejecutar el siguiente script para convertir las noticias en formato TXT:
    - ./scripts/json2txt.sh data_rep
8. Lanzar el sistema final con la siguiente instrucción:
    i. ./scripts/launch_ner.sh
9. Abrir Kibana en un navegador web:
    - http://localhost:5601
10. Abrir Brat en un navegador web:
    - http://localhost:8001