

UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR



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

TRABAJO FIN DE MÁSTER

Análisis de sentimiento en redes sociales en tiempo real con Spark

José Manuel Bustos Muñoz
Tutor: Paulo Villegas Núñez

Septiembre 2019

Análisis de sentimiento en redes sociales en tiempo real con Spark

AUTOR: José Manuel Bustos Muñoz
TUTOR: Paulo Villegas Núñez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Septiembre de 2019

Resumen

Este Trabajo Fin de Máster del programa Big Data y Data Science trata sobre el análisis de sentimiento en redes sociales en tiempo real. En el trabajo se ha procurado utilizar y desarrollar los conocimientos adquiridos y tecnologías vistas durante el Máster, tanto de forma teórica como de forma práctica.

Se presentará en la memoria el distinto trabajo realizado tanto en el estudio de la materia, el diseño e implementación del sistema, así como las pruebas realizadas y los resultados obtenidos.

Al final se presentarán las conclusiones a las que se ha llegado, y se propondrán las posibles mejoras y ampliaciones que podrían realizarse al trabajo ya realizado.

Agradecimientos

El autor quiere agradecer en primer lugar a Elena, su pareja, por toda su comprensión, paciencia, y ayuda a lo largo de los dos años de duración del Máster.

Así mismo, también quiere agradecer al tutor de este trabajo por su labor como guía académico, mediante la ayuda prestada a lo largo del desarrollo, así como su dedicación reflejada en la pronta y buena respuesta recibida durante el transcurso del proyecto.

INDICE DE CONTENIDOS

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	1
1.3	Organización de la memoria	1
2	Estado del arte	3
2.1	Procesamiento del lenguaje natural.....	3
2.1.1	Introducción	3
2.1.2	Aplicaciones	4
2.1.3	Librerías de Python para el Procesamiento del Lenguaje Natural	4
2.2	Análisis de sentimientos.....	4
2.2.1	Introducción	4
2.2.2	Aplicaciones	5
2.2.3	Métodos.....	6
2.2.3.1	Enfoques basados en reglas.....	6
2.2.3.2	Sistemas automáticos	6
2.2.3.3	Sistemas híbridos	7
2.3	Arquitectura y componentes.....	7
2.3.1	Arquitectura Lambda.....	7
2.3.2	Componentes de la arquitectura.	8
2.3.2.1	Ingesta de la información	8
2.3.2.2	Bases de Datos	9
2.3.2.3	Análisis y procesamiento de los datos	9
2.3.2.4	Visualización y explotación de los resultados.....	11
3	Diseño	13
3.1	Arquitectura del sistema.....	13
3.1.1	Arquitectura.....	13
3.1.2	Componentes del sistema	14
4	Desarrollo	15
4.1	Despliegue y prueba de módulos	15
4.1.1	Uso de API de Twitter.....	15
4.1.2	Instalación y uso de Apache Nifi	16
4.1.3	Instalación y uso de MongoDB.....	17
4.1.4	Configuración y uso de Apache Kafka	18
4.1.5	Instalación y uso de Spark.....	19
4.1.6	Visualización y explotación	22
4.1.6.1	Tableau	23
4.1.6.2	ElasticSearch y Kibana.....	24
4.2	Obtención de datos	25
4.3	Entrenamiento de modelos	26
4.3.1	Textblob	26
4.3.2	Spacy	27
4.3.3	Scikit-learn	29
4.3.4	Spark ML	31
5	Integración, pruebas y resultados	33
5.1	Integración y arranque del sistema.....	33
5.2	Pruebas y resultados del sistema	33
6	Conclusiones y trabajo futuro	37

6.1 Conclusiones	37
6.2 Trabajo futuro.....	37
Referencias	39
Bibliografía	41
Enlaces consultados	42
Anexos.....	43

INDICE DE FIGURAS

FIGURA 2-1: DIAGRAMA DE ALTO NIVEL DE LA ARQUITECTURA LAMBDA	7
FIGURA 3-1: DIAGRAMA DE LA ARQUITECTURA DEL SISTEMA	13
FIGURA 4-1: CAPTURA CON LOS ATRIBUTOS CON LOS QUE VIENEN LOS DATOS DESDE LA API	15
FIGURA 4-2: CAPTURA DEL FLUJO NIFI DESARROLLADO	17
FIGURA 4-3: CAPTURA CON LAS COLECCIONES DE LA BBDD MONGODB.....	18
FIGURA 4-4: CAPTURA CON LAS COLECCIONES DE MONGODB Y LOS LANZAMIENTOS DE LOS SCRIPTS PYTHON	19
FIGURA 4-5: CONFIGURACIÓN LOCAL DE SPARK.....	20
FIGURA 4-6: EJEMPLO DE DATOS TRAS PASAR POR EL PREPROCESADO Y LIMPIEZA DE LOS TEXTOS	21
FIGURA 4-7: CAPTURA CON LOS DATOS QUE ESTÁN LLEGANDO EN STREAMING TRAS PASAR POR LOS PROCESOS DE LIMPIEZA	21
FIGURA 4-8: CAPTURA DE DATAFRAME CON DATOS RECOGIDOS DE KAFKA EN STRUCTURED STREAMING	22
FIGURA 4-9: CAPTURA CON LOS DATOS YA CON SU SENTIMIENTO CALCULADO TRAS APLICAR UN MODELO ENTRENADO.....	22
FIGURA 4-10: EJEMPLO DE VISUALIZACIÓN CON LOS DATOS PREVIAMENTE CARGADOS EN TABLEAU	23
FIGURA 4-11: CAPTURA CON LOS ÍNDICES CREADOS EN ELASTICSEARCH	24
FIGURA 4-12: PATRONES DE ÍNDICE CREADOS PARA TWEETS EN INGLÉS Y EN ESPAÑOL	24
FIGURA 4-13: DISTRIBUCIÓN DE LA CLASE SENTIMIENTO EN LOS DATASETS FINALES.....	26
FIGURA 4-14: EJEMPLO DE DATOS RESULTANTES AL APLICAR TEXTBLOB PARA ETIQUETAR EL SENTIMIENTO	27
FIGURA 4-15: RESULTADO MODELO DE CLASIFICACIÓN DE TEXTBLOB	27

FIGURA 4-16: EJEMPLO DE DATOS RESULTANTES TRAS PASAR POR LAS FUNCIONES DE SPACY	28
FIGURA 4-17: RESULTADO MODELO DE CLASIFICACIÓN TRAS APLICAR FUNCIONES DE SPACY	28
FIGURA 4-18: RESULTADO MODELO DE CLASIFICACIÓN SIN CLASE NEUTRA	28
FIGURA 4-19: RESULTADO 1ª PRUEBA MODELO DE CLASIFICACIÓN BASADO EN CARACTERES.	29
FIGURA 4-20: RESULTADOS DE DISTINTAS PRUEBAS DE MODELOS DE CLASIFICACIÓN CON DATOS EN INGLÉS.....	30
FIGURA 4-21: RESULTADOS DE DISTINTAS PRUEBAS DE MODELOS DE CLASIFICACIÓN CON DATOS EN ESPAÑOL.....	30
FIGURA 4-22: RESULTADOS DE DISTINTAS PRUEBAS DE MODELOS DE CLASIFICACIÓN CON DATOS EN INGLÉS.....	30
FIGURA 4-23: RESULTADOS DE DISTINTAS PRUEBAS DE MODELOS DE CLASIFICACIÓN CON DATOS EN ESPAÑOL.....	31
FIGURA 4-24: RESULTADOS DISTINTAS PRUEBAS DE MODELOS DE CLASIFICACIÓN EN SPARK ML, CON DATOS EN INGLÉS.....	31
FIGURA 4-25: RESULTADOS DISTINTAS PRUEBAS DE MODELOS DE CLASIFICACIÓN EN SPARK ML, CON DATOS EN ESPAÑOL.	31
FIGURA 5-1: EJEMPLO DE SALIDA POR CONSOLA DE DATOS EN INGLÉS PROCESADOS Y CON EL SENTIMIENTO	34
FIGURA 5-2: EJEMPLO DE SALIDA POR CONSOLA DE DATOS EN ESPAÑOL PROCESADOS Y CON EL SENTIMIENTO	34
FIGURA 5-3: EJEMPLO DE DATOS ALMACENADOS EN STREAMING EN MONGODB CON EL SENTIMIENTO CALCULADO	35
FIGURA 5-4: IMAGEN DE LOS ÍNDICES GENERADOS EN ELASTICSEARCH	35
FIGURA 5-5: IMAGEN DE VISUALIZACIÓN EN KIBANA CON EL RECUENTO POR SENTIMIENTO..	36

1 Introducción

1.1 Motivación

La mayor motivación para desarrollar este trabajo sería la siguiente: poner en práctica y aprender algo más de varias de las tecnologías y herramientas vistas en el Máster. Para ello, se ha optado por realizar un trabajo que simule un sistema completo, desde el inicio hasta el final.

Otra de las motivaciones era el uso de Spark, que es una de las tecnologías más demandadas y en auge en el mercado, y todavía más si nos referimos a un sistema en Streaming.

Por último, otra motivación más personal, es la utilización como fuente de datos de Twitter. Esta aplicación la utilizo de forma personal desde hace ya 10 años, y me atraía el poder recoger sus datos y realizar distintos análisis con este origen de datos.

1.2 Objetivos

En este trabajo se va a desarrollar un sistema de análisis de sentimiento en tiempo real, de los datos de Twitter y con el uso de Spark.

Los objetivos principales serían:

- Trabajar en un sistema completo, desde la ingesta de los datos hasta la explotación de estos, utilizando el framework de Spark para el procesado y análisis de los datos. Así mismo, conseguir realizar una prueba de concepto funcional del sistema, para probar todas sus partes, y obtener unos resultados y conclusiones que puedan ser medidos y evaluados.
- Probar diferentes algoritmos y librerías para el procesamiento del lenguaje natural y el análisis del sentimiento.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Introducción: se presentan la motivación y objetivos del presente trabajo fin de Máster, y la organización de la propia memoria.
- Estado del arte: se describe brevemente la situación actual del procesamiento del lenguaje natural, y del análisis de sentimientos, así como ciertas tecnologías para ello.
- Diseño: se aborda la explicación y concepción del sistema.
- Desarrollo: se presenta cómo se ha montado el sistema para poder realizar las pruebas deseadas al sistema.
- Integración, pruebas y resultados: se explican las distintas pruebas realizadas con el sistema, y los resultados obtenidos.
- Conclusiones y trabajo futuro: se comentan las conclusiones obtenidas durante la realización del trabajo, así como los posibles puntos de mejora y profundización que se ven más factibles.
- Referencias, bibliografía, enlaces consultados, y anexos: se listan las distintas referencias utilizadas, los libros y documentos consultados, y en los anexos se indican los distintos scripts y códigos utilizados en el proyecto y alojados en un repositorio github [1].

2 Estado del arte

2.1 Procesamiento del lenguaje natural

2.1.1 Introducción

El Procesamiento del Lenguaje Natural o NLP [2] es una disciplina que se encuentra entre varias ciencias, como serían la informática, la lingüística y la inteligencia artificial. La idea principal es darles a las máquinas la capacidad de comprender el lenguaje natural, lo cual es una tarea propia de los seres humanos.

La historia del procesamiento del lenguaje natural se inició en la década de 1950, aunque se puede encontrar algo de trabajo anterior. En estos años la mayoría de trabajo realizado sería en torno a sistemas de traducción automática. La mayor evolución ocurrió a finales de la década de 1980, donde hubo una revolución en el procesamiento debido a la introducción del aprendizaje automático, con algoritmos para el procesamiento del lenguaje. También se desarrollaron los primeros sistemas de traducción automática estadística, todo esto debido al aumento constante del poder de cómputo y la disminución gradual del predominio de las teorías lingüísticas.

En los últimos tiempos, la investigación reciente se ha centrado más en algoritmos de aprendizaje semi-supervisado y no supervisados. Además, también se ha generalizado el uso de redes neuronales profundas para el procesamiento del lenguaje natural.

De forma general, en el Procesamiento del Lenguaje Natural se utilizan seis niveles de comprensión, para poder descubrir el significado del texto. Estos niveles son:

- *Nivel fonético*: se revisa la fonética, la forma en la que se pronuncian las palabras.
- *Nivel morfológico*: se estudia la estructura de las palabras para delimitarlas y clasificarlas.
- *Nivel sintáctico*: análisis de la sintaxis, lo que incluye la división de la oración en cada uno de sus componentes.
- *Nivel semántico*: lo que se busca aquí es comprender el significado de la oración.
- *Nivel discursivo*: este nivel examina el significado de la oración en relación con otra oración del propio texto o documento.
- *Nivel pragmático*: se ocupa del análisis de oraciones y como se usan en diferentes situaciones. También, como su significado cambia según la situación.

Estos niveles se complementan entre sí, y el objetivo de los sistemas de NLP es utilizar en una máquina los necesarios según el problema a resolver, y luego crear una oración estructurada y sin ambigüedad para que tenga un significado claro.

En el análisis de textos hay principalmente tres estrategias:

- *Estrategia clásica*: expertos lingüistas desarrollan gramáticas y reglas del lenguaje. Se emplean corpus de texto anotados para validar las gramáticas y reglas, que se explotan para analizar nuevos textos.
- *Estrategia híbrida*: se usan gramáticas y reglas del lenguaje para extraer características complejas. Se emplean métodos estadísticos para entrenar modelos predictivos sobre corpus etiquetados.
- *Estrategia estadística*: se parte de los corpus anotados con los objetivos a predecir. Se extraen características básicas del texto, y con métodos estadísticos como el aprendizaje automático, se entrenan los modelos predictivos.

Algunos de los mayores retos a los que se enfrenta el procesamiento del lenguaje natural serían la ambigüedad, el uso de ironía, detectar la correcta separación entre las palabras, y la posible recepción imperfecta de los datos.

2.1.2 Aplicaciones

El procesamiento del lenguaje natural tiene multitud de aplicaciones, algunas de las principales serían:

- *Análisis de sentimiento*: identificar el sentimiento de un texto, desde muy negativo, a neutral, y hasta muy positivo.
- *Reconocer entidades*: identificar las entidades del texto como podrían ser personas, lugares u organizaciones.
- *Clasificación de texto*: analizar el contenido de un documento para asignar palabras clave o etiquetas a párrafos del texto.
- *Crear chatbots*: generación de bots que puedan interactuar con las personas.
- *Resumir texto*: extraer las ideas más importantes de un documento.
- *Traducción automática*: conseguir traducir de forma automática textos entre distintos idiomas.

Con todas estas posibles aplicaciones, se puede usar en muchos ámbitos, además con el crecimiento del volumen de datos que se genera cada año y el gran aumento de datos en formato texto, cada vez va a ser más importante el procesamiento del lenguaje natural. Se estima que, por norma general, alrededor del 80% de los datos que se generan son no estructurados [3][4][5], y gran parte de estos son en forma de texto libre o lenguaje natural.

2.1.3 Librerías de Python para el Procesamiento del Lenguaje Natural

Python es uno de los lenguajes más populares actualmente, con gran importancia en campos como el de la inteligencia artificial. Por lo tanto, hay gran cantidad de librerías en Python para el procesamiento del lenguaje natural.

Enumeramos algunas de las librerías más conocidas:

- *NLTK*: la librería líder para el procesamiento del lenguaje natural. Proporciona gran cantidad de corpus y recursos léxicos, y un gran conjunto de bibliotecas.
- *Textblob*: simplifica el procesamiento de texto, y posee una suave curva de aprendizaje. Además de proporcionar algún algoritmo de clasificación para usar con un modelo de aprendizaje supervisado, da la opción de calcular el sentimiento del texto directamente utilizando métodos propios que obtienen la polaridad y subjetividad del texto.
- *Stanford Core NLP*: escrita en Java, pero posee una interfaz en Python. Para muchos constituye el estado del arte sobre las técnicas del Procesamiento del Lenguaje Natural.
- *Spacy*: librería que sobresale por su facilidad de uso y su velocidad.
- *Scikit-Learn*: aunque no sea una librería diseñada explícitamente para el procesamiento del lenguaje natural, es una de las más utilizadas en Python para todo lo relacionado con machine learning e inteligencia artificial, como sería el caso del NLP.

También con la popularidad del aprendizaje profundo, muchos de los frameworks que se utilizan en Deep Learning pueden ser aplicados para realizar modelos de NLP.

2.2 Análisis de sentimientos

2.2.1 Introducción

El análisis de sentimientos [6] es un área de investigación enmarcada dentro del campo del procesamiento del lenguaje natural, y cuyo objetivo fundamental es el tratamiento computacional de opiniones, sentimientos y subjetividad en textos. En este contexto, una opinión es una valoración positiva o negativa acerca de un producto, servicio, organización, persona o cualquier otro tipo de ente sobre la que se expresa un texto determinado.

La llegada de la web 2.0 y la popularización de las redes sociales como twitter, han catapultado este campo de investigación de la inteligencia artificial. Esta ingente cantidad de información junto al aumento de la potencia de computación de los ordenadores, han hecho posible la aplicación de técnicas de aprendizaje automático para la clasificación de los textos en base a su polaridad sentimental.

El análisis de sentimientos puede ser modelado como un problema de clasificación donde se deben resolver dos subproblemas: clasificar una oración como subjetiva u objetiva (clasificación de la subjetividad), y clasificar una oración como expresión de una opinión positiva, negativa o neutra (clasificación de polaridad). La clasificación de la polaridad de un texto es la tarea en la que se centran la mayoría de los estudios y sistemas.

Ejemplo del proceso en el análisis de sentimientos:

- *Filtración de datos*: primero se usan las palabras claves para descartar el contenido no deseado, y luego establecer palabras para obtener categorías según su polaridad.
- *Extracción del contenido*: una vez pasado el filtro, se elimina el contenido no deseado y se comienza a trabajar con el contenido de calidad.
- *Análisis de contenido*: el contenido útil y de calidad se categorizará donde corresponda. Esto puede hacerlo un algoritmo o una persona física.
- *Revisión*: proceso de revisar todo lo realizado hasta ahora, ver posibles aspectos a mejorar, o corregir errores que hayan podido ocurrir.

En cuanto a los niveles de alcance en los que se puede aplicar serían los siguientes:

- Nivel de *documento*: se obtiene el sentimiento de un documento completo o un párrafo.
- Nivel de *sentencia*: se obtiene el sentimiento de una sola frase.
- Nivel *sub-frase*: se obtiene el sentimiento de las sub-expresiones dentro de una oración.

Además del alcance, se puede clasificar el análisis de sentimiento entre varios tipos:

- *Análisis de sentimiento de grano fino*: se busca en las opiniones un mayor detalle pudiendo clasificar el sentimiento en más clases como podrían ser “muy positivo” o “muy negativo”.
- *Detección de la emoción*: el objetivo sería detectar las distintas emociones como podrían ser la felicidad, ira, tristeza, y similares.
- *Análisis de sentimiento basado en aspectos*: además de tener en cuenta la polaridad, se quiere ver sobre que características o aspectos de un producto está tratando el texto.
- *Análisis de intención*: intenta sacar de un texto la intención implícita con la que se ha emitido o escrito, como podría ser diferenciar si un texto está emitiendo una queja.

2.2.2 Aplicaciones

En la actualidad, con el gran y constante incremento de información pública y privada, y en gran medida que corresponde a información no estructurada en formato texto, el número de aplicaciones prácticas en las que se puede utilizar el análisis de opiniones sigue creciendo. Por ejemplo, aunque los textos de twitter sean cortos, el análisis de sentimiento aplicado al microblogging ha mostrado ser un indicador válido para analizar el sentimiento político [7].

Veamos varios puntos para los que puede servir el análisis de sentimiento:

- Obtener datos de calidad, evitando tener multitud de datos que no tienen valor real para tomar decisiones.
- Poder tomar decisiones en tiempo real.
- Conseguir desarrollar mejores estrategias empresariales.
- Gestionar la reputación online, y ayudar en las acciones que deben tomarse dentro de un plan estratégico de marketing online.

En base a esto, algunas de las aplicaciones prácticas más importantes podrían ser el monitoreo de medios sociales y de marca, donde se podrían analizar tweets, mensajes de Facebook, o noticias de la web para estudiar el sentimiento de un público en particular sobre una marca o servicio. También puede utilizarse en sistemas de atención al cliente, para mejorar el servicio y poder analizar todo lo que ha ocurrido durante el mismo.

2.2.3 Métodos

2.2.3.1 Enfoques basados en reglas

Los enfoques basados en reglas por lo general definen un conjunto de reglas en algún tipo de lenguaje de programación que identifican la subjetividad, la polaridad o el sujeto de una opinión. Las reglas pueden usar variedad de entradas como serían técnicas de procesamiento del lenguaje natural como la tokenización, y recursos como los léxicos, que son listas de palabras y expresiones.

Un ejemplo básico de una aplicación basada en reglas sería:

1. Definir dos listas de palabras polarizadas, por sentimiento negativo y positivo.
2. Dado un texto, contar el número de palabras positivas y de negativas que aparecen.
3. Si el número de apariciones de palabras positivas es mayor, devolver un sentimiento positivo, o al contrario si el caso es al revés.

Este sistema sería bastante ingenuo, y si se quiere un procesamiento más avanzado el sistema se vuelve bastante complejo, y con un mantenimiento difícil y costoso.

2.2.3.2 Sistemas automáticos

Los sistemas automáticos se basan en el aprendizaje de las máquinas. Generalmente, la tarea de análisis de sentimientos se define como un problema de clasificación donde un clasificador se alimenta de entrada con un texto o documento, y devuelve la categoría correspondiente en cuanto a la polaridad de este, ya sea positiva, negativa, o neutra.

Un clasificador de este tipo de sistemas tiene como flujo de proceso el siguiente:

- *Procesos de formación y predicción:* En el proceso de entrenamiento, el modelo aprende a asociar una entrada (texto) con una salida (etiqueta de sentimiento). En el proceso de predicción, se introduce texto nuevo sin anotar para predecir las etiquetas correspondientes.

De forma general, suele englobar las siguientes etapas:

- *Extracción de características del texto:* el primer paso es transformar el texto en una representación numérica, normalmente un vector. Por lo general, cada componente del vector representa la frecuencia de una palabra o expresión en un diccionario predefinido.
- *Algoritmos de clasificación:* la etapa de clasificación suele implicar un modelo estadístico como podrían ser Naive Bayes, Regresión logística o Máquinas de vectores de soporte.
- *Métricas y evaluación:* hay muchas maneras para obtener métricas de rendimiento y evaluar un clasificador. Algunas de las medidas más habituales para evaluar el modelo son:
 - Accuracy: es la medida más simple e intuitiva, y representa la razón entre las predicciones correctas sobre el total de predicciones realizadas.
 - Precisión: es la razón entre el número de documentos clasificados correctamente como pertenecientes a la clase A, y el número total de documentos que han sido clasificados por el modelo como de clase A.
 - Recall: es la relación entre los documentos clasificados correctamente como pertenecientes a la clase A, y la suma de todos los documentos de la clase A.
 - F-score: se presenta como la media armónica entre las medidas de Precisión y Recall.

2.2.3.3 Sistemas híbridos

El concepto de los llamados sistemas híbridos es bastante intuitivo: intentar combinar lo mejor de los otros dos sistemas, tanto los basados en reglas como los sistemas automáticos. Por lo general, con esta combinación, los métodos pueden mejorar tanto su exactitud como su precisión.

2.3 Arquitectura y componentes

2.3.1 Arquitectura Lambda

Debido a que se dispone de un volumen cada vez mayor de datos, y a la necesidad de analizarlos y obtener valor de ellos lo antes posible, surge la necesidad de definir nuevas arquitecturas para cubrir casos de uso distintos de los que había hasta ahora. Una de las arquitecturas más comunes para estos proyectos es la Arquitectura Lambda, que en este caso es la elegida para este trabajo.

Dos conceptos que definir serían el procesamiento batch y el procesamiento en streaming:

- Batch: hace referencia a un procesamiento de datos capturados y almacenados previamente.
- Streaming: un procesamiento es de tipo streaming cuando está continuamente recibiendo y tratando nueva información según va llegando, sin tener un fin en lo referente al tiempo.

La arquitectura Lambda apareció en el año 2012 y se atribuye a Nathan Marz [8]. La definió en base a su experiencia en sistemas de tratamiento de datos distribuidos, y su objetivo era tener un sistema robusto tolerante a fallos, tanto humanos como de hardware, que fuera linealmente escalable y que permitiese realizar escrituras y lecturas con baja latencia.

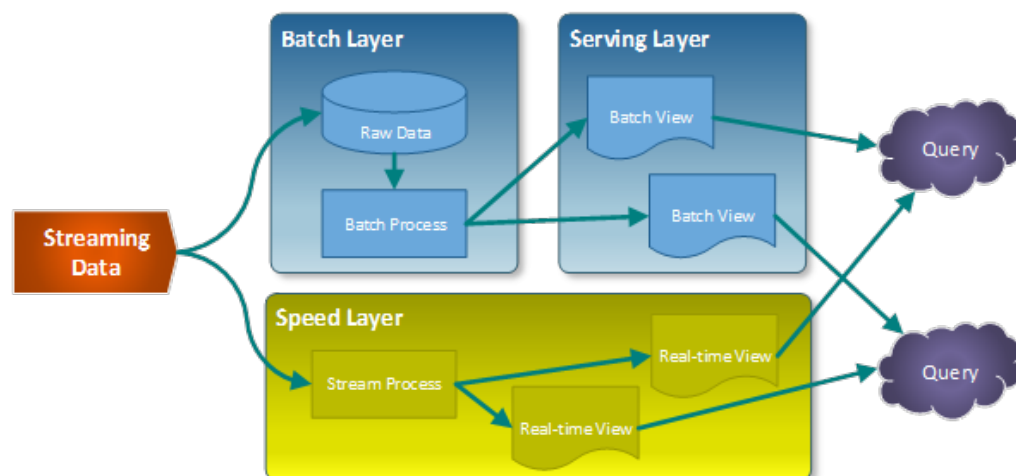


Figura 2-1: Diagrama de alto nivel de la Arquitectura Lambda

Las características de la Arquitectura Lambda son:

- La nueva información recogida por el sistema se envía tanto a la capa de batch como a la capa de streaming (denominada como Speed Layer en la imagen anterior).
- En la capa batch (Batch Layer) se gestiona la información en crudo, es decir, sin modificar. Los datos nuevos se añaden a los ya existentes. Seguidamente se hace un tratamiento mediante un proceso batch cuyo resultado serán las denominadas Batch Views, que se usarán en la capa que sirve los datos para ofrecer la información ya transformada al exterior.
- La capa que sirve los datos o Serving Layer, indexa las Batch Views generadas en el paso anterior de forma que puedan ser consultadas con baja latencia.
- La capa de streaming o Speed Layer, compensa la alta latencia de las escrituras que ocurre en la serving layer y solo tiene en cuenta los datos nuevos.

- Finalmente, la respuesta a las consultas realizadas se construye combinando los resultados de las Batch Views y de las vistas en tiempo real (Real-time Views), las cuales se han generado en el paso anterior.

En resumen, este tipo de arquitectura se caracteriza por utilizar distintas capas para el procesamiento batch y el streaming.

2.3.2 Componentes de la arquitectura.

Se van a presentar ciertas tecnologías que pueden ser utilizadas en las diferentes partes de la arquitectura de un sistema de este tipo.

2.3.2.1 Ingesta de la información

2.3.2.1.1 Apache Nifi

Apache Nifi [9] es una plataforma integrada de procesamiento y logística de datos en tiempo real, para automatizar el movimiento de datos entre diferentes sistemas de forma rápida, fácil y segura. Se encarga de cargar datos que pueden ser de diferentes fuentes, los pasa por un flujo de procesos para su tratamiento, y los vuelca en otra fuente.

Se caracteriza por tener una interfaz web, muy potente e intuitiva que permite diseñar y configurar de forma visual el flujo de datos, así como operar sobre el proceso (arranque y parada), y monitorizar el estado en búsqueda de posibles errores que se vayan produciendo.

Las principales características son las siguientes:

- La ingesta de información no afecta al rendimiento del servicio del origen ni del destino.
- Independencia de las fuentes de datos, admite fuentes dispersas y distribuidas de diferentes formatos, esquemas, protocolos o velocidades y tamaños.
- Permite cargar y grabar datos de multitud de tecnologías: HDFS, Elasticsearch, BBDD SQL, MongoDB, FTP, etc.
- Transformación entre numerosos formatos de datos: JSON, XML, Avro, CSV, etc.
- Permite conexión con otros sistemas de procesamiento como Kafka o Flume.
- La ejecución en paralelo mediante Zookeeper permite correr múltiples instancias de Nifi.
- Permite el rastreo de datos en tiempo real.

2.3.2.1.2 Apache Kafka

Apache Kafka [10] es un sistema de intermediación de mensajes basado en el modelo publicador/suscriptor. Se considera un sistema persistente, escalable, replicado y tolerante a fallos. A estas características, se añade la velocidad de lecturas y escrituras que lo convierten en una herramienta excelente para comunicaciones en tiempo real o streaming.

Funciona basándose en los siguientes conceptos:

- *Topic* (tema): Categorías en las que clasificar los mensajes enviados a Kafka.
- *Producer* (productor): Clientes conectados responsables de publicar los mensajes.
- *Consumer* (consumidor): Clientes conectados encargados de consumir los mensajes.
- *Broker* (nodos): Nodos que forman el cluster.
- *Offset*: es el indicador que indica a cada consumidor el último elemento que ha leído. Esto hace que si se cae el sistema no se pierdan los datos.

Algunas de sus principales características son:

- Proporciona conectores para conectarse a casi cualquier fuente de datos, y también para almacenar los datos en sitios como podría ser HDFS, Amazon S3 o Elasticsearch.
- Permite la implementación en diferentes lenguajes: Java, Scala, Python, Ruby, etc.

- Utiliza Apache Zookeeper para almacenar el estado de los nodos.
- Permite ingestar grandes volúmenes de datos.
- Buen rendimiento entre latencias bajas.
- Permite escalamiento horizontal.
- Diferentes grupos de consumidores pueden consumir mensajes a diferente ritmo.
- Actúa de amortiguador entre productores y consumidores, ideal para absorber picos de carga.

2.3.2.2 Bases de Datos

2.3.2.2.1 MongoDB

Las bases de datos NoSQL aportan esquemas más flexibles y entornos altamente distribuidos, y esto es de gran valor para proyectos que tratan con gran cantidad de datos, a diferente velocidad, y en muchos casos refiriéndonos a datos semiestructurados o no estructurados. En el caso de MongoDB [11], se trata de una BBDD NoSQL orientada a documentos, que almacenan datos de tipo documento, y tienen gran flexibilidad con esquemas de datos dinámicos.

Algunas de las características principales de MongoDB:

- Los documentos se almacenan en BSON, que es una estructura JSON en formato binario, que otorga mayor velocidad en las consultas.
- Gran velocidad y sencillo sistema de consulta de los contenidos de la BBDD.
- Proporciona APIs y drivers para gran cantidad de lenguajes de programación.
- Guarda estructuras de datos con esquema dinámico.
- Cualquier campo de un documento puede ser indexado. También soporta índices secundarios.
- Soporta replicación, y escala de forma horizontal usando el concepto de shard. De esta manera se balancea la carga.

2.3.2.3 Análisis y procesamiento de los datos

2.3.2.3.1 Spark

Apache Spark [12] es un framework de computación en clúster open source. Sus creadores lo definen como una tecnología rápida y genérica para el procesamiento de datos a gran escala. Es rápido en comparación con otras soluciones anteriores para la gestión de grandes volúmenes de datos, como sería Hadoop. El secreto de Spark radica en la memoria (RAM) que hace que el procesamiento sea más rápido que el realizado en el disco, además de su computación perezosa (lazy processing) basada en grafos de proceso (DAG).

Algunas de sus principales características:

- Proporciona una interfaz para la programación de clústeres completos con paralelismo de datos y tolerancia a fallos.
- Proporciona APIs en Java, Scala, Python y R.
- Proporciona un motor optimizado que soporta la ejecución de grafos de proceso en general.
- Soporta un conjunto extenso y rico de herramientas de alto nivel entre las que se incluyen Spark SQL (para el procesamiento de datos estructurados basada en SQL), MLlib para implementar machine learning, GraphX para el procesamiento de grafos y Spark Streaming.

El punto de partida de Apache Spark es el Resilient Distributed Dataset (RDD), un set de datos dividido en particiones. Un RDD es un conjunto de elementos, con tolerancia a fallos (fault-tolerant), que se pueden ejecutar en paralelo. Luego, en la versión Spark 2.0, llegó el DataFrame,

donde el dataset está organizado en columnas con nombre. Conceptualmente es equivalente a una base de datos relacional o un dataframe en R o Python, pero con sistemas de optimización más sofisticados. Los dataframes se pueden obtener de una variedad de fuentes tales como: datos estructurados, tabla en Hive, bases de datos externas o RDD existentes.

En resumen, el dataframe API es el medio utilizado por los desarrolladores de Spark para facilitar el trabajo con datos en el framework. Es muy similar a los de Pandas y R, pero con muchas ventajas para el procesamiento masivo y distribuido.

2.3.2.3.2 Spark Streaming

Spark Streaming [13] es una extensión de la API core de Spark, que da respuesta al procesamiento de datos en tiempo real de forma escalable, con alto rendimiento y tolerancia a fallos. Como idea general se podría decir que Spark Streaming toma un flujo de datos continuo y lo convierte en un flujo discreto denominado DStream, para que el core de Spark lo pueda procesar.

Un Dstream no es más que una abstracción proporcionada por Spark Streaming que representa a una secuencia de RDDs ordenados en el tiempo, donde cada uno de ellos guarda datos de un intervalo concreto. Con esta abstracción se consigue que el core lo analice sin enterarse de que está procesando un flujo de datos, ya que el trabajo de crear y coordinar los RDDs es realizado por Spark Streaming.

Con todo esto, convierte a Apache Spark en el verdadero sucesor de Hadoop ya que proporciona el primer marco que soporta completamente la arquitectura lambda, la cual nos permite trabajar simultáneamente el procesamiento de datos en batch y streaming.

Con las nuevas versiones de Spark, ha surgido Structured Streaming, que es un mecanismo que se construye encima de SparkSQL y nos permite la ingesta de datos en tiempo real en el motor SQL y su procesamiento. Esto nos da muchos beneficios, porque es muy interesante ingestar datos en tiempo real y transformarlos para obtener resultados, y además poder hacerlos apoyándonos en SparkSQL.

Las principales características de Structured Streaming son las siguientes:

- Podemos realizar en general las mismas operaciones que realizamos sobre SparkSQL, con los DataFrames y los DataSets.
- La herramienta es la que se encarga de gestionar el streaming, es decir, la herramienta va creando pequeños procesos microbatch con cada uno de los datos que va recibiendo.
- Se asegura el end-to-end.

2.3.2.3.3 Spark ML

MLlib o Spark MLlib [14] es la librería de Machine Learning de Apache Spark. El framework de computación distribuida que incorpora esta librería permite hacer uso de una serie de algoritmos de Machine Learning. Lo interesante de todo esto, es que los algoritmos de Machine Learning que están implementados dentro de MLlib pueden ser escalados y paralelizados, aprovechando toda la base y potencia de Spark.

Dentro de MLlib podemos encontrar dos APIs:

- La API principal o Spark ML, basada en DataFrames y que es esta dentro del paquete ml.
- La API original o Spark MLlib, que hace uso de RDDs y esta dentro del paquete mllib. Actualmente se encuentra en modo mantenimiento y muy probable que deje de existir en Spark 3.0.

El cambio de pasar de API basada en RDDs a otra basada en DataFrames, se debe a que estos últimos son unas estructuras de datos más apropiadas para optimización y fáciles de manejar, y como vimos anteriormente en la herramienta de streaming, es la tónica habitual ahora mismo, el pasar a utilizar siempre que sea posible Dataframes en lugar de RDDs.

2.3.2.4 Visualización y explotación de los resultados

2.3.2.4.1 Elasticsearch y Kibana

Elasticsearch [15] es un servidor de búsqueda en tiempo real que proporciona almacenamiento indexado y distribuido. Está basado en Lucene como motor de indexación, por lo tanto, proporciona gran potencia de búsquedas de texto, y simplifica las consultas utilizando la interfaz web RestFul.

Además de toda la potencia de búsqueda sobre texto, tiene otras características importantes como son:

- Soporta configuración en clúster con servicios de alta disponibilidad.
- Permite explotación de datos a gran escala.
- Al ser distribuido facilita la escalabilidad.
- Permite analítica en tiempo real.

Con el uso de Elasticsearch, vienen asociadas otras herramientas, y una es la herramienta de visualización Kibana. Es una herramienta que nos permite la visualización de información almacenada e indexada en Elasticsearch, de forma fácil e intuitiva, mediante una interfaz web.

Kibana ofrece análisis en tiempo real, un algoritmo de búsqueda muy flexible y diferentes tipos de vistas para los datos, como podrían ser histogramas o diagramas circulares. Al ser una aplicación web escrita en Javascript, Kibana puede usarse en cualquier plataforma.

3 Diseño

3.1 Arquitectura del sistema.

3.1.1 Arquitectura

La arquitectura elegida para el sistema sería la que se ha visto en el punto anterior, o sea, la arquitectura Lambda. Pensamos que es la arquitectura ideal para un sistema con estas características, donde se debe prestar atención tanto al procesamiento en streaming, como al procesamiento batch.

En nuestro caso, la capa batch se nutrirá de los datasets o corpus con tweets de distinta índole con el sentimiento ya anotado. Estos datasets se tratarán para formar los conjuntos de datos finales, que serán usados en el procesamiento en Spark ML donde se generarán y entrenarán los modelos de clasificación, que posteriormente se podrán usar en Streaming para predecir el sentimiento de los tweets que lleguen al sistema por el flujo streaming.

La capa streaming se alimentará desde la API streaming de twitter, la cual nos proporcionará el flujo streaming de datos. La ingesta al sistema se hará con el uso de Apache Nifi, y mediante Apache Kafka se transmitirán los datos hacia Spark Streaming, donde se realizará el procesamiento que acabará dando como resultado la predicción del sentimiento de estos datos del flujo streaming. Para la explotación y visualización del sentimiento calculado, se utilizará el conjunto de Elasticsearch y Kibana.

Para la persistencia del sistema, se apostará por utilizar la BBDD MongoDB, que irá almacenando tanto los datos que lleguen al sistema en diferentes tablas o colecciones, como para ir guardando los datos a los que se vaya calculando su sentimiento. Podría utilizarse posteriormente para hacer análisis de históricos, o alimentar algún panel de control o administración del sistema.

En el siguiente diagrama se expone la arquitectura de nuestro sistema con las distintas tecnologías y herramientas elegidas, que comentaremos brevemente después el porqué de dichas elecciones.

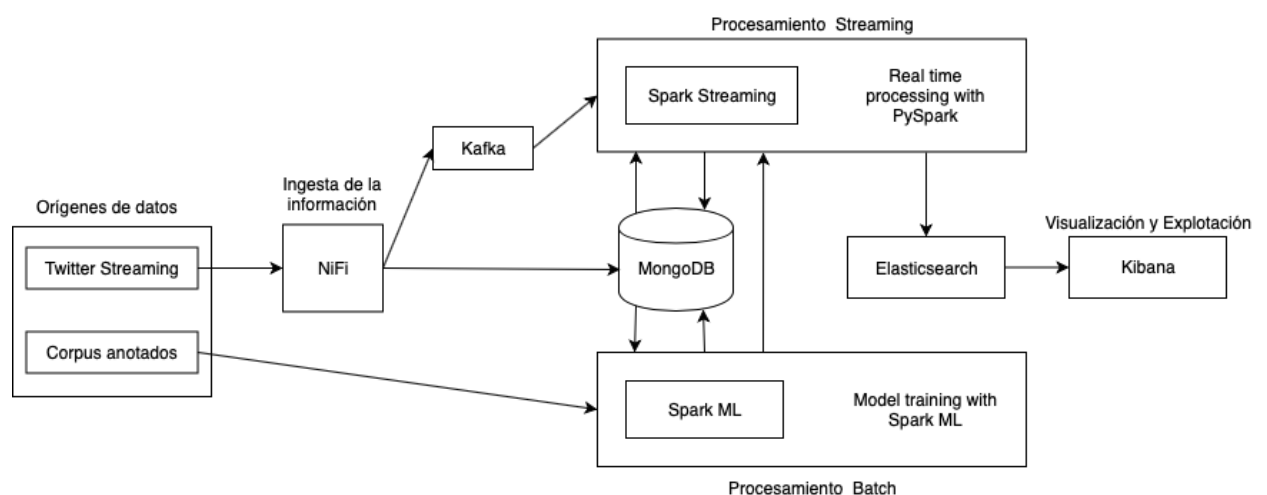


Figura 3-1: Diagrama de la arquitectura del sistema

3.1.2 Componentes del sistema

Todos los componentes elegidos para formar el sistema son tecnologías o herramientas vistas durante el máster, algo que hemos considerado crucial para su elección, por tener ya conocimientos de estas lo cual ayudará en su implementación, y por poner en práctica estos conocimientos adquiridos. Además, también hay otros motivos puramente más técnicos que comentaremos a continuación.

Estos componentes utilizados serían los siguientes para cada una de las partes del sistema:

- Ingesta de la información
 - *Apache Nifi*: se elige Nifi como primera pieza del sistema, para la ingesta de los datos desde twitter en el flujo streaming. Hay que destacar su facilidad de uso e instalación, que mediante su interfaz web amigable facilita mucho el trabajo con la herramienta. Además, tiene un procesador propio para conectar con twitter, y usando las credenciales pertinentes conectar con la API streaming y comenzar a inyectar datos en tiempo real. También tiene procesadores para realizar conexiones con muchas otras herramientas, como podrían ser MongoDB o Kafka, lo cual lo convierte en una herramienta muy potente e ideal para sistemas streaming.
 - *Apache Kafka*: Kafka nos parece una gran opción por su escalabilidad y tolerancia a fallos, para conectar con Apache Nifi y transmitir el flujo de datos hacia las siguientes patas del sistema como serían la persistencia y el procesamiento de los datos, que se harán con MongoDB y Spark. Encaja de manera ideal, igual que pasaba con Nifi, al realizar las conexiones con las otras herramientas de manera sencilla. Además, proporciona muchas opciones, como es el uso de distintos topics por idioma, separando así desde el inicio el flujo de datos según el idioma, y siendo más sencillo así para las siguientes partes al tener ya dividido el flujo de datos.
- Bases de datos o persistencia
 - *MongoDB*: al ser una BBDD orientada a documentos, y recogerse los datos de la fuente de origen en formato JSON, nos parece una solución ideal para la persistencia. Además, proporciona gran velocidad e integración con otras herramientas. Así como la facilidad de uso, por ejemplo, utilizando una herramienta como MongoDB Compass, que evita el uso de la consola y lo hace más sencillo.
- Análisis y procesamiento de los datos
 - *Spark*: A la hora del procesamiento en el sistema, había muchas opciones, como podría ser por ejemplo utilizar Apache Storm, pero nos decidimos por Apache Spark ya que nos proporciona todo lo necesario tanto en la parte en streaming como en la parte de machine learning, además de la mayor facilidad que otorga la API de dataframes para realizar todo el trabajo, tanto en Spark ML como en streaming, con el uso de Structured Streaming. Por lo tanto, parecía la opción más lógica al reunir en su framework todas las partes que necesitábamos para el procesamiento de los datos.
- Visualización y explotación de los resultados
 - *Elasticsearch y Kibana*: se eligen estas herramientas para la visualización por el conjunto que forman y la potencia que otorgan, la facilidad de conexión entre Spark y Elasticsearch, y la interfaz tan amigable que proporciona Kibana para realizar los análisis y visualizaciones. Además de que son herramientas pensadas para el tratamiento de textos y uso en tiempo real.

4 Desarrollo

4.1 Despliegue y prueba de módulos

4.1.1 Uso de API de Twitter

Lo primero que se necesitó fue crear una cuenta de developer en twitter, para usar las APIs que proporciona la aplicación, y conseguir las credenciales necesarias. Se va a usar la API de streaming, para poder capturar tweets en streaming.

Se debe ir a la url de desarrolladores de twitter [16], crear una cuenta como desarrollador, y posteriormente crear una aplicación para obtener las credenciales. Cuando está realizado el proceso se puede acceder a la aplicación y ver los datos de ésta, y en otra pestaña coger las credenciales que se deben usar luego en las conexiones, que en nuestro caso sería desde Apache Nifi.

Para capturar los tweets usando las credenciales conseguidas, se va a usar la librería “*tweepy*” [17], lanzando un fichero python capturaremos los tweets en streaming, y posteriormente los cargamos y visualizamos en un notebook. La idea de este primer acercamiento a los datos que llegan de twitter, es hacer un primer análisis de estos para ver los formatos y los atributos que tienen, además de seleccionar la información de los datos que nos va a interesar para el trabajo posterior.

El primer paso sería instalar en la máquina la librería, que nos va a permitir desde un script Python conectar a la API para obtener los datos. Lo hacemos desde la consola: “*pip install tweepy*”.

Luego usamos el fichero Python comentado anteriormente: “*conseguir_tweets.py*” [18], donde utilizando la librería “*tweepy*” y las credenciales de la API, se obtiene un fichero ‘*json*’ con los datos.

Al lanzar por consola el script, se le pasan argumentos que serán palabras clave por las que capturar los tweets. Ejemplo de lanzamiento del script: “*python conseguir_tweets.py /Barcelona Messi...*”.

Después de finalizar la conexión se puede ver cómo se ha generado en el directorio donde hemos lanzado el fichero anterior, el archivo “*stream_test.json*” con los datos. Con este archivo ‘*json*’ con los datos ya creamos el notebook “*pruebas_tweets*” [19], donde cargamos los datos recogidos de twitter, y visualizamos sus atributos y el esquema que tienen.

Los tweets vienen con 38 atributos según se puede ver en la siguiente imagen:

```
print("num_rows: %d\tColumns: %d\n" % (df.shape[0], df.shape[1]))
print("Columns:\n", list(df.columns))

num_rows: 7633 Columns: 38

Columns:
['contributors', 'coordinates', 'created_at', 'display_text_range', 'entities', 'extended_entities', 'extended_tweet',
 'favorite_count', 'favorited', 'filter_level', 'geo', 'id', 'id_str', 'in_reply_to_screen_name', 'in_reply_to_status_id',
 'in_reply_to_status_id_str', 'in_reply_to_user_id', 'in_reply_to_user_id_str', 'is_quote_status', 'lang', 'limit', 'place',
 'possibly_sensitive', 'quote_count', 'quoted_status', 'quoted_status_id', 'quoted_status_id_str', 'quoted_status_permalink',
 'reply_count', 'retweet_count', 'retweeted', 'retweeted_status', 'source', 'text', 'timestamp_ms', 'truncated', 'user', 'withheld_in_countries']
```

Figura 4-1: captura con los atributos con los que vienen los datos desde la API

Revisando la documentación oficial [20], donde se puede ver el significado de cada atributo, se piensa que la información que podría ser relevante sería la siguiente:

- ‘*extended_entities*’: Cuando el tweet contiene entre una y cuatro fotos nativas, o un video, o un GIF animado, contiene un array de metadatos ‘*multimedia*’.
- ‘*extended_tweet*’: texto del tweet extendido.
- ‘*truncated*’: Indica si el valor del parámetro de texto se truncó por exceso de caracteres.
- ‘*created_at*’: Es la fecha en la cual se ha escrito el tweet.
- ‘*coordinates*’: Localización geográfica, especificada por el usuario u otra aplicación.
- ‘*entities*’: Es un objeto que contiene información sobre las distintas “entidades” que puede contener un tweet, como hashtags o urls.

- *'favorite_count'*: Número de “me gusta” que ha recibido el tweet.
- *'id'*: Es la id única por tweet que lo representa, en formato numérico.
- *'lang'*: Idioma en el cual está escrito el tweet.
- *'quote_count'*: Número de citas que ha recibido el tweet.
- *'quoted_status'*: contiene dentro el JSON del tweet al que citan.
- *'reply_count'*: Número de respuestas que ha recibido el tweet.
- *'retweet_count'*: Número de retweet que ha recibido el tweet.
- *'retweeted_status'*: contiene dentro el JSON del tweet al cual se le ha hecho retweet.
- *'text'*: Aquí se encuentra el texto del tweet siempre y cuando no sea superior a 140 caracteres.
- *'user'*: Es un objeto con varios atributos sobre el usuario que haya escrito el tweet.

Creemos que de los datos que arroja twitter, estos atributos serían los que tienen mayor valor para poder hacer distintos análisis y trabajo con ellos.

Para este trabajo, donde lo que vamos a analizar es el análisis de sentimiento en streaming, realmente solo necesitamos el texto de cada tweet, aunque también puede ser buena idea recoger el contador de interacciones de cada tweet, o el idioma del tweet si va a separarse el flujo por idioma.

4.1.2 Instalación y uso de Apache Nifi

Apache Nifi será la primera herramienta del sistema, y se encargará de la ingesta de los datos al sistema, desde el origen. Se conectará desde Nifi con twitter usando las credenciales obtenidas en el punto anterior, y a través de una interfaz web amigable y sencilla, se llevará el flujo de datos de twitter hacia la BBDD MongoDB, y hacia Kafka, que seguirá la transmisión de datos por el sistema.

Una vez descargado apache Nifi desde la url correspondiente [21], se accede a la carpeta descargada y se puede arrancar Nifi desde la terminal, lanzando el script “*nifi.sh*” junto al comando “*start*”. Posteriormente, una vez arrancado Nifi, se puede acceder a la interfaz web de la herramienta, indicando el puerto. En nuestro caso se accede con la dirección ‘*http://localhost:8081/nifi/*’.

El flujo que se realiza para obtener los datos desde twitter e ingestarlos en el sistema sería el siguiente:

- Procesador *GetTwitter*: es el procesador inicial y es donde se configuran las credenciales y tokens de twitter para hacer la conexión a la API y recoger los datos.
- Procesador *EvaluateJsonPath*: en este procesador se evalúan los datos json y se crean los dos atributos por los que se filtrará después los tweets. Se usan los atributos “lang” y “text”, porque el filtro que se va a hacer a todo lo que llegue será coger los tweets que no estén vacíos, y que sean en español o inglés.
- Procesador *RouteOnAttribute*: aquí se cogerán los tweets que sean en español o en inglés, y se mandarán a los procesadores de PutKafka y PutMongo según corresponda.
- Procesadores *PutKafka*: son los procesadores que van a llevar los datos a Kafka. Se configura en cada procesador el topic correspondiente, ya que tenemos un topic para cada idioma.
- Procesadores *PutMongo*: por un lado, tendremos un procesador que envía los datos a MongoDB, tanto en inglés como en español, para alimentar una tabla máster con todos los tweets que lleguen de ambos idiomas; y por otro lado un procesador que envía a una tabla de Mongo todos los tweets que llegan al sistema y no están catalogados ni en español ni en inglés. Esto se realiza en paralelo mientras los datos llegan a los procesadores Kafka.
- Procesador *PutFile*: a este procesador se envían los posibles errores que ocurran durante el flujo en otros procesadores, y en la ruta configurada se irían almacenando ficheros tipo ‘*.json*’ con los datos que hayan producido el error.

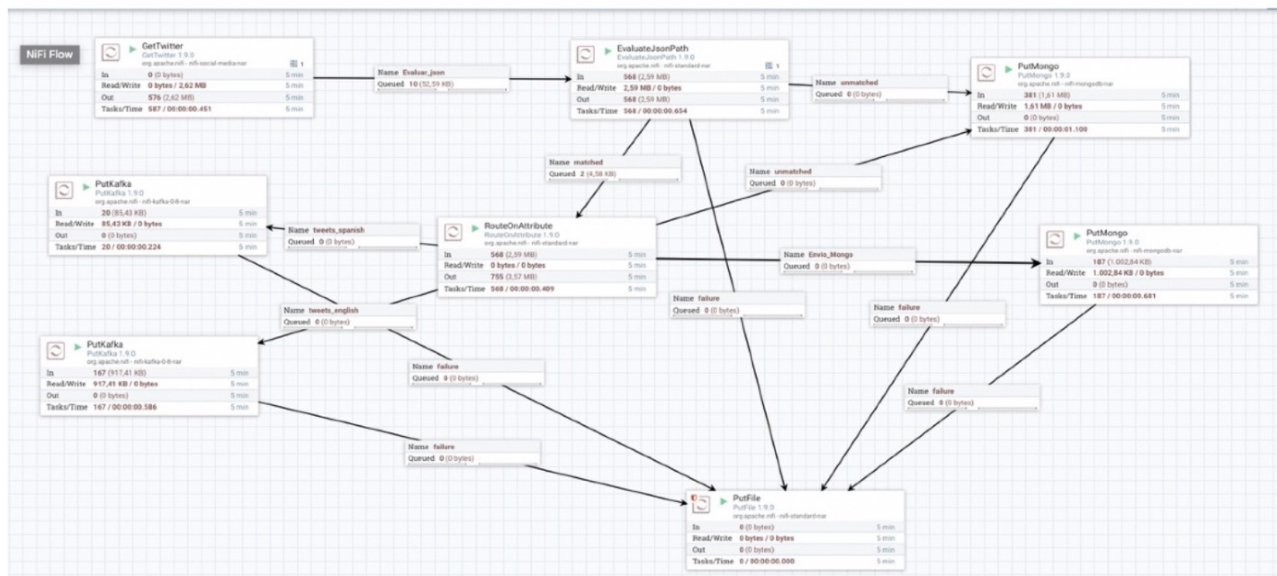


Figura 4-2: captura del flujo Nifi desarrollado

Algunas de las configuraciones de los procesadores del flujo serían:

- Configuración procesador *evaluateJsonPath*: se crean las dos variables “*twitter.lang*” y “*twitter.text*” a las que se les asignan los atributos “*lang*” y “*text*” para el idioma y el texto.
- Configuración procesador *putFile*: se añade el directorio donde guardar los registros que provoquen errores en el flujo. Se deben dar permisos de escritura en dicho directorio.
- Configuración procesador *RouteOnAttribute*: se generan los flujos de salida, para cada idioma, y se les asigna una expresión regular donde usando las dos variables creadas anteriormente, se llevarán los tweets con texto español por un lado, y los tweets con texto inglés por otro. Ejemplo de expresión regular para el idioma inglés, donde también controlamos que el texto no esté vacío: “*\${twitter.text:isEmpty():not():and(\${twitter.lang:equals("en")})}*”.
- Configuración procesador *PutKafka*: en estos procesadores que llevan la información a Kafka se indica el broker, en este caso “*localhost*”, el puerto del nodo Kafka, y el topic.
- Configuración procesador *PutMongo*: se debe configurar la url de Mongo en local, el nombre de la base de datos y la colección en la que se insertarán los datos. En este caso los datos serían los siguientes:
 - Mongo URI: ‘*mongodb://localhost:27017*’
 - Mongo Database Name: ‘*tfm_twitter*’
 - Mongo Collection Name: ‘*tweets_master*’

Cuando todo está configurado, el flujo se activa y se ve cómo empieza a llegar la información y va pasando por los procesadores hasta llegar a los procesadores finales que envían la información a Kafka y a MongoDB. Se va actualizando cada poco tiempo el flujo y se va viendo el aumento de información que pasa por el mismo, como el número de datos que entran y salen, o su tamaño en MB.

4.1.3 Instalación y uso de MongoDB

Como BBDD se decide utilizar ‘MongoDB’, la cual proporciona mucha flexibilidad en el almacenamiento, y trabaja con documentos ‘.json’, lo cual viene muy bien por ser el formato en el que se recogen los datos desde twitter.

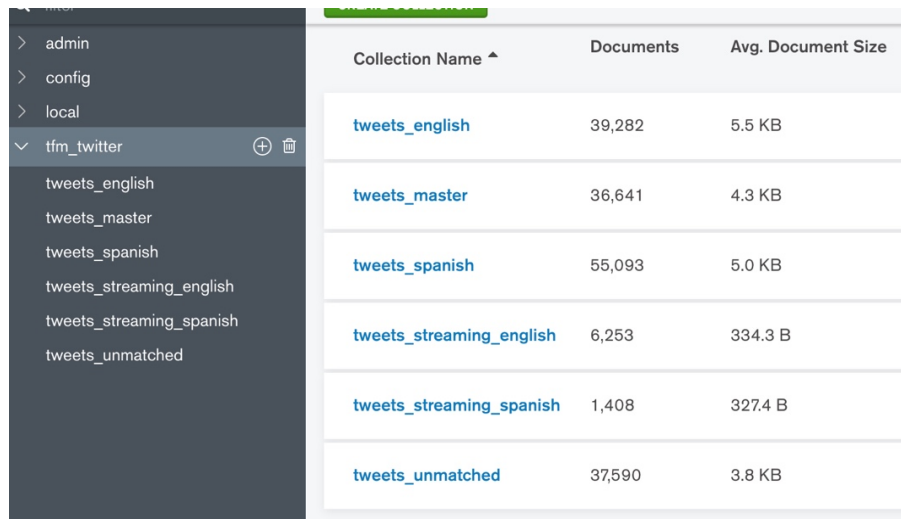
Nos descargamos a local MongoDB desde la dirección habilitada para ello [22]. Una vez descargado, iremos a la carpeta de Mongo y podremos ejecutar desde la terminal el comando necesario para ejecutar MongoDB: ‘*mongod*’. Cuando está arrancado Mongo, desde la consola se podría acceder y

gestionar una BBDD, pero vamos a usar la aplicación ‘MongoDB Compass’ que es más visual. Al acceder a Compass se indica el hostname ‘localhost’ y el puerto que usará Mongo que es el ‘27017’.

Para la generación de la BBDD y las colecciones, se crea un script ‘mongo_creacion.js’ [23]. El script se lanza desde consola: ‘./mongo mongo_creacion.js’, y una vez ejecutado podemos ver las tablas.

Se crea la BBDD ‘tfm_twitter’ y dentro las siguientes colecciones o tablas:

- ‘tweets_master’: se almacenan desde Nifi todos los registros con todos sus atributos, para tener un backup de todos los datos que lleguen al sistema sin filtros.
- ‘tweets_english’: se almacenarán los tweets catalogados con idioma inglés.
- ‘tweets_spanish’: se almacenarán los tweets catalogados con idioma español.
- ‘tweets_streaming_english’: en esta colección se almacenarán los textos en inglés ya con su sentimiento etiquetado en streaming usando un modelo de clasificación.
- ‘tweets_streaming_spanish’: en esta colección se almacenarán los textos en español ya con su sentimiento etiquetado en streaming usando un modelo de clasificación.
- ‘tweets_unmatched’: se almacenarán los registros desde Nifi que no están catalogados en idioma inglés ni español, por una posible mejora como tener en cuenta más idiomas.



The screenshot shows the MongoDB Compass interface. On the left, a sidebar lists the database structure: 'admin', 'config', 'local', and 'tfm_twitter'. The 'tfm_twitter' database is expanded, showing collections: 'tweets_english', 'tweets_master', 'tweets_spanish', 'tweets_streaming_english', 'tweets_streaming_spanish', and 'tweets_unmatched'. On the right, a table displays the details of these collections.

Collection Name	Documents	Avg. Document Size
tweets_english	39,282	5.5 KB
tweets_master	36,641	4.3 KB
tweets_spanish	55,093	5.0 KB
tweets_streaming_english	6,253	334.3 B
tweets_streaming_spanish	1,408	327.4 B
tweets_unmatched	37,590	3.8 KB

Figura 4-3: captura con las colecciones de la BBDD MongoDB

4.1.4 Configuración y uso de Apache Kafka

Se va a utilizar Apache Kafka para continuar con el flujo de datos en streaming desde la ingesta con Nifi hasta Spark donde se procesarán los datos.

Una vez descargado Apache Kafka desde la url correspondiente [24], en el directorio de configuración ‘./config’ vamos a realizar unas modificaciones en la configuración: vamos a crear 3 ficheros ‘server.properties’, y así simularemos tener un clúster con distintos nodos. Para ello, en cada uno de estos ficheros properties, debemos indicar el ‘broker.id’ correspondiente, indicar el puerto que usará cada nodo, e indicar la ruta para los logs, que será distinta en cada server.

Aunque realmente al probarse el sistema en local, no haría falta usar brokers distintos, si parecía interesante comprobar que se puede simular el tener un clúster. Al trabajar con un clúster distribuido, con varios nodos, es como se aprovecha la potencia de Kafka exprimiendo el paralelismo que brinda, y otorgaría escalado horizontal y tolerancia a fallos, además de la replicación ya que la información sería replicada entre los nodos del clúster.

Una vez realizadas estas configuraciones, se arranca Kafka en el sistema. Primero debe levantarse ‘zookeeper’, y luego levantar los 3 nodos de Kafka también desde la consola:

Arrancar zookeeper desde la consola: ‘./bin/zookeeper-server-start.sh ./config/zookeeper.properties’.

Spark Master at spark://MBP-de-Jose.home:7077

URL: spark://MBP-de-Jose.home:7077

Alive Workers: 2

Cores in use: 16 Total, 0 Used

Memory in use: 30.0 GB Total, 0.0 B Used

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (2)

Worker Id	Address	State
worker-20190924065145-192.168.1.150-49519	192.168.1.150:49519	ALIVE
worker-20190924065147-192.168.1.150-49521	192.168.1.150:49521	ALIVE

Figura 4-5: configuración local de Spark

Con Spark se puede trabajar con distintas APIs y lenguajes, y en este caso usamos Python, a través de Pyspark, ya que es el lenguaje con el que se ha trabajado en la mayor parte de asignaturas del Máster. Desde local vamos a trabajar en el desarrollo utilizando la herramienta de Jupyter Notebook, por tanto, vamos a configurar que al levantar PySpark desde cualquier ubicación, se arranque directamente en Jupyter Notebook, por mayor comodidad de arranque y del trabajo en sí mismo.

Para ello se modifica el fichero “*~/.bash_profile*” y se añade la ruta donde está Spark-2.4 en nuestro local. Una vez configurado esto, ya ejecutamos ‘*pyspark*’ desde una terminal, y en el navegador web aparece el Jupyter Notebook, donde podremos crear y ejecutar notebooks con Spark.

Para poder utilizar Spark correctamente con otras herramientas como podrían ser MongoDB o Kafka, hace falta descargar distintos jars y paquetes, que una vez descargados [28] a la carpeta correspondiente del directorio de Spark, deben añadirse al fichero de configuración por defecto de Spark “*spark-defaults.conf*”, y así que al lanzar la herramienta puedan utilizarse. Se han tenido que añadir al fichero de configuración los jars relacionados con streaming, kafka, mongodb o elasticsearch.

Se crea el notebook “*Mongo_Spark*” [29], donde se realizan pruebas con Spark conectando con MongoDB, recogiendo los datos almacenados, y trabajando con ellos usando tanto RDDs como Dataframes. Para la conexión con MongoDB, probamos la librería “*pymongo*”, y el conector oficial entre Mongo y Spark. “*Pymongo*” lo instalamos vía consola: ‘*pip install pymongo*’, y el conector de Spark y Mongo está configurado en Spark y lo podemos utilizar en el notebook.

Aunque se probó a cargar los datos como RDD (Pymongo los carga como RDD), y trabajar con este tipo de datos de Spark, finalmente se seguirá adelante trabajando con Dataframes a lo largo de las distintas etapas del sistema. Por lo tanto, se usará siempre el conector oficial Spark-Mongo, que directamente al cargar datos lo hace nativamente como dataframe.

Una vez se tienen los dataframes con los datos cargados desde MongoDB, se aplican unas funciones de preprocesado y limpieza de los textos. Estas funciones de limpieza han sido implementadas como parte del trabajo de este TFM, en base al uso de expresiones regulares, que se aplicarán a cada uno de los textos. Este proceso de limpieza de los textos se centrará en:

- Tokenización de los textos para tenerlos divididos en palabras/tokens.
- Eliminar de los textos urls, imágenes, emoticonos u otros elementos similares.
- Eliminar de los textos caracteres que no aporten valor real al análisis de sentimiento a realizar.

Usaremos estas funciones aplicadas como base de la limpieza y preprocesado de los tweets, y posteriormente se aplicarán a los tweets en streaming antes de calcularles el sentimiento.

	text	tokens_clean
0	@EJFC26 Y Messi	[messi]
1	RT @btsmoonchild64: These group photos deserve...	[group, photos, deserve, attention]
2	@rehankkhanNDS Overacting *	[overacting]
3	Pay day play day 🙄🙄🙄🙄🙄	[pay, day, play, day]
4	@pandaeyed1 Thank you!	[thank]
5	RT @liamyoung: Strange that Tony Blair has sud...	[strange, tony, blair, suddenly, become, ...]
6	Hard work puts you where good luck can find you.	[hard, work, puts, good, luck, find]
7	RT @xCiphxr: When creative kids try playing co...	[creative, kids, try, playing, comp]
8	RT @bonang_m: I'm working on one as we speak. ...	[im, working, one, speak, thank, incredib...
9	RT @akashbanerjee: After #PulwamaAttack, terro...	[pulwamaattack, terrorists, scored, bigger ...]
10	RT @nodp0d: I am people https://t.co/zpJyQOW10	[people]
11	RT @VINTAGE4MOMS: Vintage 2 piece hammered cop...	[vintage, piece, hammered, copper, pot, l...
12	RT @godisgodisback: Beyoncé: WORLD STOP!\n\nWo...	[beyonc, world, stop, world]
13	No. It just needs money and connections. Your ...	[needs, money, connections, average, norma...
14	RT @PlanetRockRadio: The upcoming album from @...	[upcoming, album, boasts, new, songs, inc...
15	@MoEgger1530 I see ya! https://t.co/xAJd7aL2LC	[see, ya]

Figura 4-6: ejemplo de datos tras pasar por el preprocesado y limpieza de los textos

Las primeras pruebas que realizamos de la capa streaming las hacemos en el notebook “*Spark_Streaming*” [30]. Aquí probamos con Spark Streaming, a generar un proceso en streaming que recoja los datos de Kafka, y realizar el preprocesado y limpieza con ellos, en lugar de hacerlo como antes con los datos previamente almacenados en la BBDD. En este notebook, se crea el contexto Spark Streaming necesario, y se conecta con el topic correspondiente de Kafka para obtener los datos que están llegando a ese topic. Se usará para ello el método “*createDirectStream*”, que conecta al topic de Kafka que le pasemos y genera un objeto con el que podremos interactuar.

Hacemos una pequeña prueba, donde en cada micro-batch streaming (intervalo de tiempo) sacamos por pantalla la siguiente información: número de registros del batch, los textos de estos tweets, los textos tras pasar por la limpieza, y los textos ya limpios divididos en tokens.

```

-----
Time: 2019-08-15 15:04:00
-----
Tweets del batch: 97

-----
Time: 2019-08-15 15:04:00
-----
Tweets total (medio minuto rolling): 97

-----
Time: 2019-08-15 15:04:00
-----
Castrá y asesina a su yerno por no pagar la manutención de sus hijos https://t.co/a2jkxzyJf
Para que tú unción sea real algo debe morir en ty @PastorVladimir @EsmeraldadeRiva @00Zavala @AMCompaz
RT @MissLadrillos: Para que el próximo informe alconadamon, el cels y amnesty sobre fake news, ah no cierto: https://t.co/WfD5Krf8PW
RT @Ozuna_Pr: Yo X Ti @rosaliavt https://t.co/19h2v6RMcJ
Ya me he encargado yo de tener a las mejores personas a mi lado
...

-----
Time: 2019-08-15 15:04:00
-----
castra asesina yerno pagar manutencion hijos
uncion real debe morir ty
proximo informe alconadamon cels amnesty fake news ah cierto
x
encargado tener mejores personas lado
...

-----
Time: 2019-08-15 15:04:00
-----
['castra', 'asesina', 'yerno', 'pagar', 'manutencion', 'hijos']
['uncion', 'real', 'debe', 'morir', 'ty']
['proximo', 'informe', 'alconadamon', 'cels', 'amnesty', 'fake', 'news', 'ah', 'cierto']
['x']

```

Figura 4-7: captura con los datos que están llegando en streaming tras pasar por los procesos de limpieza

Se realizan operaciones satisfactorias, pero se decide tirar por el uso de ‘Spark Structured Streaming’, en lugar de Spark Streaming. ‘Structured Streaming’ es una opción más moderna, y tiene un mejor comportamiento nativo con dataframes, lo cual viene bien ya que en el sistema siempre vamos a trabajar con dataframes.

En el notebook de “*Spark_structured_streaming*” [31], hemos conectado en streaming con el flujo de datos del sistema, ya utilizando esta nueva opción de streaming que proporciona Spark. Se conecta con Kafka satisfactoriamente, y recibimos los datos de los dos topics que tenemos, pudiendo sacar como dataframe los datos recibidos:

	text	sentiment
0	Jajajajajaja https://t.co/zt4NncmOgg	None
1	RT @00INVICTUS: Pleno al quince, todo el gobie...	None
2	RT @angelriver01: Hola mi nombre es Ángel Rive...	None
3	RT @SofaVallejos3: Abro hilo de las cosas favo...	None
4	RT @conchetujade: podemos hablar de el poder q...	None
5	RT @anluma99: La orquesta de Filadelfia y la Ó...	None

Figura 4-8: captura de dataframe con datos recogidos de Kafka en Structured Streaming

Una vez se tiene un dataframe con los datos recogidos de los topics de Kafka, ya se pueden hacer queries sobre estos datos, procesarlos y visualizarlos. En este notebook se ha realizado además una primera prueba de concepto del sistema, ya que aparte de capturar en streaming los tweets que están llegando, se ha entrenado un modelo de clasificación, y se ha aplicado posteriormente a los datos recibidos desde Kafka.

	text	sentiment
0	Jajajajajaja https://t.co/zt4NncmOgg	2
1	RT @00INVICTUS: Pleno al quince, todo el gobie...	0
2	RT @angelriver01: Hola mi nombre es Ángel Rive...	2
3	RT @SofaVallejos3: Abro hilo de las cosas favo...	2
4	RT @conchetujade: podemos hablar de el poder q...	0
5	RT @anluma99: La orquesta de Filadelfia y la Ó...	2

Figura 4-9: captura con los datos ya con su sentimiento calculado tras aplicar un modelo entrenado

Por último, también se realiza la primera prueba para una vez están los datos con el sentimiento calculado, almacenarlos en distintos formatos o persistirlos, para su posterior explotación y visualización. Se prueba su guardado como fichero ‘.csv’, su persistencia desde Spark a la colección correspondiente de MongoDB, y la persistencia en Elasticsearch.

En este punto, las distintas partes del sistema estarían probadas, y se calcula el sentimiento de los datos que llegan en streaming, pero se va a seguir profundizando en la parte del streaming para ver posibles mejoras y ampliaciones, además de en otras partes del sistema.

4.1.6 Visualización y explotación

En la parte final del sistema, de visualización y explotación, se ha probado con dos herramientas: Tableau, y Elasticsearch con Kibana. La idea para añadir la visualización al sistema sería poder alimentar dicha visualización en streaming con los datos tras calcularles el sentimiento.

4.1.6.1 Tableau

La primera prueba para la visualización fue usar Tableau [32]. Es una de las herramientas de visualización más potente y fácil de usar. Además, tiene conector con MongoDB y podría usarse para atacar los datos que queramos ir visualizando.

Se probó la conexión desde Tableau con MongoDB siguiendo los siguientes pasos:

- Instalar en el pc “*openssl*” que es necesario si se requiere conexión ssl entre Mongo y Tableau.
- Bajar e instalar el conector de Mongo y Tableau, “*Mongo-Bi*”. Por consola, se lanza “*mongodrdl*” y se indica el servidor donde corre Mongo, la BBDD y la colección, generándose así un archivo con el esquema para la conexión.
- Dejar listo el conector Mongo-Bi lanzando “*mongosqld*” e indicándole el esquema creado, la ruta y puerto donde corre Mongo en el sistema.

Una vez realizados los pasos anteriores, se arranca Tableau, y se selecciona la conexión con MongoDB. Para probar Tableau, se descarga desde la página de descargas [33].

Al abrir Tableau y seleccionar la conexión Mongo, se abre una pantalla para colocar los datos de conexión, y al ser la primera vez nos indica un mensaje de que necesitamos descargar e instalar otros controladores. Esto es debido a que por detrás el conector funciona con MySQL, y hay que bajar e instalar los controladores ‘*iODBC*’ y ‘*Odbc*’.

Una vez descargados e instalados los controladores anteriores, ya podemos ver las colecciones de la BBDD donde están almacenados los datos del sistema, y cargar la colección requerida cargándose los datos y pudiendo ver los distintos atributos de los datos. Lo siguiente sería crear una hoja de trabajo para hacer un primer ejemplo:

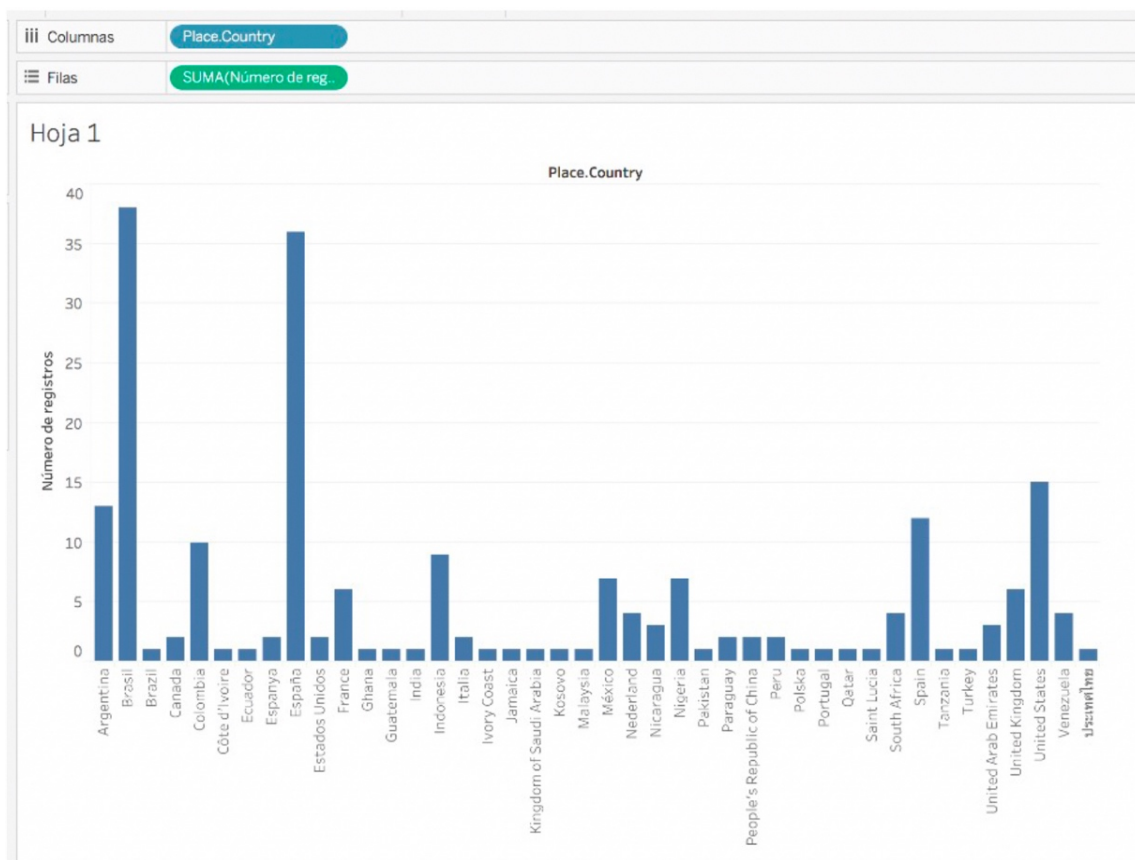


Figura 4-10: ejemplo de visualización con los datos previamente cargados en Tableau

4.1.6.2 ElasticSearch y Kibana

Después de realizar la prueba con Tableau, se vieron otras opciones para introducir la visualización en el sistema en streaming, y se decidió apostar por el uso conjunto de Elasticsearch y Kibana. Tableau podría ser una buena opción para realizar visualizaciones posteriormente y analizar los datos almacenados en MongoDB como un histórico, pero con el uso de Elasticsearch y Kibana conseguimos una mejor solución para ir tratando en streaming los datos que llegan al sistema y se procesan en Spark. Se pueden almacenar en streaming en Elasticsearch, y consumir desde Kibana, con una gran tasa de refresco, además que son herramientas que proporcionan una gran eficiencia en el tratamiento de textos.

Nos descargamos tanto Elasticsearch como Kibana desde sus respectivas páginas de descarga [34][35]. Una vez descargadas a local, debemos realizar una configuración en Kibana para enlazarla con Elasticsearch, y que cuando se carguen datos en Elastic, se puedan atacar desde Kibana. Vamos al directorio de configuración de Kibana que sería `/conf`, y se modifica el fichero `'kibana.yml'`, indicando la url donde corre Elasticsearch: `'elasticsearch.hosts: [http://localhost:9200]'`. Cuando está configurado este enlace, vamos a cada uno de los directorios, y desde la carpeta `/bin` de cada una, se pueden arrancar las aplicaciones desde consola, simplemente lanzando tanto `'elasticsearch'` como `'kibana'`.

Desde el navegador, se puede acceder a los distintos índices creados en Elastic, y ver otra información como sería el número de documentos de cada uno de los índices. Para ver esto se entra a la siguiente url: `'http://localhost:9200/_cat/indices?v'`.

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
green	open	.kibana	R2DnU202TBkBpOkALNs6w	1	0	24	0	57.1kb	57.1kb
green	open	kibana_sample_data_flights	N0rM73ucSlugAgo2z9qnsA	1	0	13059	0	6.2mb	6.2mb
yellow	open	tweets_sentiment_spa	U-tPgdi_TAC3UkA4_ieqQ	5	1	13609	0	5.2mb	5.2mb
yellow	open	tweets_sentiment_eng	OFd2EBj-QnGsVG0DZgFk6g	5	1	93685	0	31.6mb	31.6mb

Figura 4-11: captura con los índices creados en Elasticsearch

Cuando ya tenemos algún índice creado en Elastic, y documentos almacenados, accedemos a Kibana desde el navegador con la siguiente url: `'http://localhost:5601/app/kibana#home?_g=()'`.

En Kibana, hay que configurar el índice del que queremos ver los datos de Elastic y hacer visualizaciones con ellos, para ello creamos un patrón de índice para cada idioma y así podemos cargar sus datos y realizar búsquedas sobre ellos.

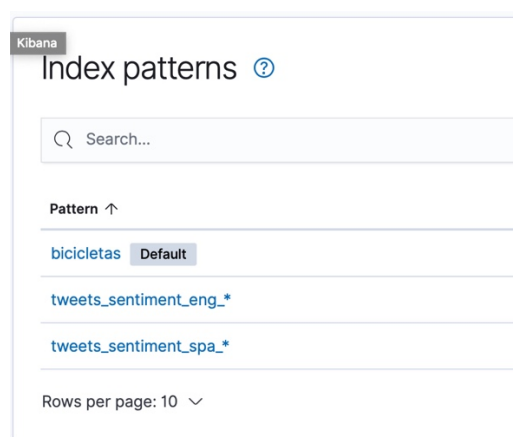


Figura 4-12: patrones de índice creados para tweets en inglés y en español

Se va a realizar una visualización sencilla, mostrando un contador de los datos que llegan y corresponden a cada patrón definido, y un contador para cada clase de sentimiento: negativo, neutro y positivo. En Kibana hay una actualización casi instantánea, con un refresco automático incluso cada 5 segundos.

4.2 Obtención de datos

Se va a utilizar aprendizaje supervisado para el entrenamiento de modelos de clasificación, que permitan predecir el sentimiento de los tweets que lleguen en streaming al sistema. Para poder entrenar los modelos, se debe disponer de corpus o datasets de tweets que tengan ya el sentimiento etiquetado. Se consiguen descargar varios datasets de datos etiquetados, tanto en inglés como en español, de distintas webs especializadas en conjuntos de datos, y también datasets utilizados en ejemplos [36][37][38][39][40][41][42][43].

Los ficheros descargados serían los siguientes:

- *'weather_agg_dfe.csv'*: tweets relacionados con el clima, en diferentes localizaciones y fechas, con el sentimiento etiquetado. Idioma inglés.
- *'text_emotion.csv'*: recopilación de tweets donde se expresan emociones. Vienen catalogados no sólo como sentimiento negativo, neutro o positivo, si no en un rango más amplio de emociones. Idioma inglés.
- *'train.csv'*: recopilación aleatoria de tweets con el sentimiento etiquetado. Idioma inglés.
- *'general-test-tagged-3l.xml'*: recopilación de tweets con el sentimiento etiquetado. Idioma español.
- *'general-train-tagged-3l.xml'*: recopilación de tweets con el sentimiento etiquetado. Idioma español.
- *'intertass-development-tagged.xml'*: recopilación de tweets con el sentimiento etiquetado. Idioma español.
- *'intertass-train-tagged.xml'*: recopilación de tweets con el sentimiento etiquetado. Idioma español.
- *'politics-test-tagged.xml'*: recopilación de tweets relacionados con la política con el sentimiento etiquetado. Idioma español.
- *'tweets_airlines.csv'*: tweets relacionados con aerolíneas, con el sentimiento etiquetado. Idioma inglés.
- *'full-corpus.csv'*: recopilación de tweets con opiniones sobre diferentes empresas de servicios como Apple o Microsoft. Idioma inglés.
- *'training.1600000.processed.noemoticon.csv'*: recopilación de tweets con opiniones sobre algún servicio o alguna persona, con el sentimiento etiquetado. Idioma inglés.
- *'testdata.manual.2009.06.14.csv'*: recopilación de tweets con opiniones sobre algún servicio o alguna persona, con el sentimiento etiquetado. Idioma inglés.

Utilizamos un notebook con Python llamado *'generacion_dataset'* [44] donde se cargan todos los ficheros descargados, y se unifican en los ficheros .csv finales, tanto en inglés como en español.

Estos datasets que formamos tendrán el atributo texto de los tweets, y el atributo sentimiento. Como el atributo sentimiento puede tener distintos valores, vamos a unificarlos todos de la siguiente forma: sentimiento negativo = 0, sentimiento neutro = 1, y sentimiento positivo = 2.

Para luego entrenar los modelos, el valor del sentimiento debe ser numérico, por eso lo etiquetamos de la anterior forma.

Los ficheros .csv que acabamos generando para posteriormente utilizar con los modelos serían:

- *df_result_english*: dataset con todos los registros en inglés.
- *df_result_spanish* [45]: dataset con todos los registros en español.
- *df_result_english_neutral* [46]: dataset con registros en inglés con el mismo número de registros con sentimiento positivo, negativo y neutro.
- *df_result_spanish_neutral* [47]: dataset con registros en español con el mismo número de registros con sentimiento positivo, negativo y neutro.
- *df_result_english_noNeutral* [48]: dataset con registros en inglés con el mismo número de registros con sentimiento positivo y negativo. No hay registros con sentimiento neutro.
- *df_result_spanish_noNeutral* [49]: dataset con registros en español con el mismo número de registros con sentimiento positivo y negativo. No hay registros con sentimiento neutro.

Además del dataset final con todos los registros en inglés, y el otro con todos los registros en español, se generan otros 4 conjuntos, 2 por idioma, donde en unos dejamos solo tweets con sentimiento positivo y negativo, y en otros tenemos las 3 clases, pero con el mismo número de datos por cada clase. A la hora de entrenar y evaluar un modelo de clasificación, es importante tener un gran número de registros, y también es mejor que el dataset esté balanceado en el número de registros por clase. Por estos motivos se generan los distintos conjuntos descritos anteriormente, y así podremos hacer algunas pruebas con cada uno.

Dataset	Negativos	Neutros	Positivos	Total
<i>df result english</i>	871.980 (49,56%)	14.470 (0,82%)	872.864 (49,61%)	1.759.314
<i>df result spanish</i>	19.344 (39,75%)	3.110 (6,39%)	26.204 (53,85%)	48.658
<i>df result english neutral</i>	14.000 (33,33%)	14.000 (33,33%)	14.000 (33,33%)	42.000
<i>df result spanish neutral</i>	3.000 (33,33%)	3.000 (33,33%)	3.000 (33,33%)	9.000
<i>df result english noNeutral</i>	40.000 (50%)	-	40.000 (50%)	80.000
<i>df result spanish noNeutral</i>	19.000 (50%)	-	19.000 (50%)	38.000

Figura 4-13: distribución de la clase sentimiento en los datasets finales

4.3 Entrenamiento de modelos

Una vez hechas las primeras pruebas, consiguiendo conectar Spark y Mongo por si es necesario usar los datos almacenados, hacer la limpieza y preprocesado de los textos, y probada la conexión en Spark Streaming y Structured Streaming, nos centramos en el análisis de sentimiento de los textos.

Hay muchas opciones para hacer análisis de sentimiento, con multitud de librerías. Vamos a realizar pruebas con Textblob, Spacy, Scikit-learn, y Spark ML. Para ello tenemos los siguientes notebooks donde se ha trabajado en ello:

- ‘*Textblob_analisis_sentimiento*’ [50]: se prueba la librería Textblob.
- ‘*Spacy_analisis_sentimiento*’ [51]: se prueba Spacy para entrenar un modelo.
- ‘*ScikitLearn_modelos_caracteres_palabras*’ [52]: se prueba con Scikit learn y distintos algoritmos de clasificación y parámetros.
- ‘*ScikitLearn_seleccion_modelo*’ [53]: con las pruebas del notebook anterior, se prueban con los mismos parámetros 4 algoritmos de clasificación para acabar eligiendo el que mejores resultados obtenga, para aplicarlo como modelo para los datos en streaming.
- ‘*Spark_MLlib_analisis_sentimiento*’ [54]: se realizan las pruebas de distintos modelos, pero haciendo todo el trabajo con Spark ML.

4.3.1 Textblob

Lo primero fue instalar la librería Textblob [55], lo hacemos desde la consola: “*pip install textblob*”. Después trabajamos en el notebook “*Textblob_analisis_sentimiento*”, donde se han seguido los siguientes puntos:

- Carga de los datos en inglés desde MongoDB, utilizando el conector entre Spark y Mongo. Estos datos almacenados en la BBDD del sistema servirán para predecir su sentimiento con la herramienta que textblob proporciona.
- Limpieza de los textos. Aplicamos las funciones de limpieza de textos ya probadas, para eliminar palabras que no aporten valor, urls, imágenes, y caracteres extraños.
- Con los textos limpios, aplicamos textblob y sacamos el valor del sentimiento por texto. Textblob permite calcular directamente el sentimiento, sin necesidad de usar modelos de clasificación, o sea, aprendizaje supervisado.

```

-----
                                text  sentiment
0                                @EJFC26 Y Messi          1
1  RT @btsmoonchild64: These group photos deserve...      2
2                                @rehankkhanNDS Overacting * 1
3                                Pay day play day🤔🤔🤔🤔🤔🤔 1
4                                @pandaeyed1 Thank you!     1
5  RT @liamyoung: Strange that Tony Blair has sud...      0
6  Hard work puts you where good luck can find you.        2
7  RT @xCiphxr: When creative kids try playing co...       2
8  RT @bonang_m: I'm working on one as we speak. ...      2
9  RT @akashbanerjee: After #PulwamaAttack, terro...       2

```

Figura 4-14: Ejemplo de datos resultantes al aplicar Textblob para etiquetar el sentimiento

- Cargamos los datasets generados en formato ‘.csv’ con el sentimiento ya etiquetado, para eliminarlo y ver como lo clasifica Textblob. Después, comparamos con los datos originales. Al comparar vemos que se han etiquetado erróneamente un 55% de los datos.
- Textblob también viene con algún clasificador para hacer aprendizaje supervisado. Usamos el clasificador Naive Bayes, y se obtienen estos resultados midiendo el modelo con accuracy:

Modelo/Clasificador	Datos	Accuracy
Naive Bayes	Datos en inglés	72,83%
Naive Bayes	Datos en español	81,3%

Figura 4-15: resultado modelo de clasificación de Textblob

Como conclusiones podemos decir que el uso de Textblob es rápido y sencillo, y puede etiquetar directamente los datos (aprendizaje no supervisado) sin necesidad de entrenar un modelo con datos ya etiquetados. Probándolo con nuestros datos etiquetados nos da unos resultados bastante mejorables. Además, se ha probado con datos en inglés ya que para español habría que traducir previamente los textos y tiene un coste computacional elevado.

Uno de los problemas de textblob es que no consigue del todo acertar con el contexto y subjetividad de los textos, y se ve bastante afectado en sus resultados por ello. Aunque esto es obviamente el gran problema de cualquier técnica de clasificación de textos.

Al entrenar con un modelo y algoritmo de clasificación parecen obtenerse mejores resultados, y podemos probarlo en ambos idiomas. De todos modos, la medida usada para evaluar los modelos, accuracy, no es la más adecuada, y cuando probemos con otros modelos y algoritmos de clasificación usaremos otras métricas.

Seguiremos por esta vía del aprendizaje supervisado, probando posteriormente con Spacy, Scikit-learn, y finalmente con Spark ML.

4.3.2 Spacy

La siguiente librería utilizada para el análisis de sentimiento sería Spacy [56], que permite buenos análisis morfológicos y sintácticos. Trabajando en el notebook con Spacy, al que llamaremos ‘Spacy_analisis_sentimiento’, primero descargamos la librería y los modelos para los idiomas inglés y español, que serían los idiomas de los datos de los que disponemos. Descarga e instalación de spacy desde consola: ‘*pip install spacy*’.

Enumeramos el trabajo realizado en el notebook:

- Carga de los datasets almacenados en ‘.csv’, para utilizarlos para entrenar los modelos.
- Limpieza de los textos. Aplicamos las funciones de limpieza de textos ya probadas.
- Descargamos los modelos de spacy para inglés y español, y posteriormente los cargamos, para poder aplicar a los datos las distintas funcionalidades de la librería.
Descarga de modelo en inglés: “*!python -m spacy download en*”.
Carga del modelo anterior: “*nlp_eng = spacy.load('en')*”.

- Usamos los datasets etiquetados y preprocesados, y con las funciones de Spacy para cada conjunto de datos vamos a obtener dos columnas: una con los textos separados por tokens, y la última con los textos limpios de tokens que no aporten valor para el análisis de sentimiento como pueden ser adverbios o artículos. Además, se pasan los verbos y nombres a su raíz para simplificar el conjunto de datos.

	text	sentiment	analyzed	posfilter
0	no te libraras de ayudar me nos besos y gra...	1	(, no, te, libraras, de, ayudar, me, nos, ,...	librar ayudar beso gracia
1	gracias mar	2	(, gracias, mar)	gracia mar
2	off pensando en el regalito sinde la que se v...	0	(off, pensando, en, el, regalito, sinde, , la...	pensar regalito sinde sgae corrupto sacar conc...
3	conozco a alguien q es adicto al drama ja ja ...	2	(conozco, a, alguien, q, es, adicto, al, drama...	conocer adicto drama sonar d
4	toca grabacion dl especial navideno mari ...	2	(toca, , grabacion, dl, especial, navideno...	tocar grabacion especial navideno mari crisma

Figura 4-16: Ejemplo de datos resultantes tras pasar por las funciones de spacy

- Con los textos procesados y finales, se generan los conjuntos de train y test, se crea el Pipeline para el modelo y se realizan pruebas con distintos parámetros, entrenando el modelo y obteniendo el score como anteriormente en el clasificador de Textblob.

Modelo/Clasificador	Datos	Accuracy
LinearSVC	Datos en inglés	77,05%
LinearSVC	Datos en español	80,68%

Figura 4-17: resultado modelo de clasificación tras aplicar funciones de Spacy

Se obtiene una mejora importante con los datos en inglés. Con los datos de español se obtiene el mismo resultado que tras aplicar el algoritmo de clasificación que venía con Textblob.

- Este problema es desbalanceado, ya que existen más datos de un sentimiento en concreto. Por ello, usar accuracy como métrica de la calidad del modelo no es aconsejable, ya que un modelo que clasifique todos los textos de un sentimiento en concreto podrá tener una precisión muy alta, a pesar de su falta de utilidad en práctica. En su lugar podría emplearse otra métrica como podría ser la métrica ‘*roc_auc_score*’, que tiene en cuenta la importancia de ambas clases. La curva ROC es una representación gráfica de la razón o ratio de verdaderos positivos frente al ratio de falsos positivos. AUC es como se llama comúnmente al área bajo la curva ROC, es el indicador más usado, y este índice se puede interpretar como la probabilidad de que un clasificador ordenará o puntuará una instancia positiva elegida aleatoriamente más alta que una negativa.

Además, para esta prueba, vamos a tener en cuenta sólo los datos negativos y positivos, para ver su impacto. Eliminamos de los datos los registros con sentimiento neutro, y volvemos a entrenar y evaluar el modelo.

Modelo/Clasificador	Datos	Accuracy	Roc_auc_score
LinearSVC	Datos en inglés (sin clase neutra)	77,51%	77,49%
LinearSVC	Datos en español (sin clase neutra)	86,11%	85,24%

Figura 4-18: resultado modelo de clasificación sin clase neutra

Se nota una mejora con los datos en español, al eliminar la clase neutra. Aunque eso puede depender de los datos, su tamaño, y otros factores.

De forma general, incluso en varios estudios sobre el análisis de sentimientos, se ha visto que la neutralidad tiene mayor importancia de la que podría parecer, incluso llegando a ver que la precisión global de un clasificador mejora si se incluye la clase neutra. Evidentemente en la realidad existe la neutralidad, y no todas las oraciones tienen un sentimiento, por lo tanto lo mejor sería contar con la

neutralidad. Además si no se cuenta con clase neutro es más sencillo caer en el sobreajuste al entrenar los modelos.

Por todo esto, a partir de ahora seguiremos probando con los datasets completos tanto en español como inglés, con las tres clases de sentimiento (negativo, neutro y positivo), por tanto con un problema de clasificación multiclase.

Para concluir, se aprecia como con los datos en español se obtienen mejores resultados que en inglés, algo que también ocurrió con el clasificador de textblob. Vamos a probar con la librería Scikit-learn y con Spark ML, que tienen más opciones y clasificadores.

4.3.3 Scikit-learn

En los dos notebooks usando esta librería [57], vamos a probar distintos algoritmos de clasificación y parámetros, para ver el mejor modelo posible. Primero instalamos la librería desde consola: *'pip install scikit-learn'*.

Se ha trabajado en dos notebooks con esta librería:

- Notebook *'ScikitLearn_modelos_caracteres_palabras'*:
 - Carga de los datasets almacenados en *'csv'*, para utilizar estos datos para entrenar los modelos de clasificación.
 - Limpieza de los textos. Aplicamos las funciones de limpieza de textos ya probadas, para eliminar palabras que no aporten valor, urls, imágenes, y caracteres extraños.
 - Generación de los conjuntos de train y test con los datos de ambos idiomas.
 - Generación y entrenamiento de los modelos.

Hacemos distintas pruebas, con algo más de profundidad que en la anterior prueba con spacy. Probamos con modelos basados en caracteres y en palabras, esto se define en el vectorizador que se usa en el pipeline del modelo. Para combinar fácilmente este proceso de vectorización con un modelo de clasificación vamos a usar un Pipeline de scikit-learn. "Pipeline" nos permite encadenar varios procesos de modelado, de forma que se ejecuten de forma coordinada. Aquí vamos a definir un Pipeline que primero aplica la vectorización por frecuencias de caracteres, y después la pasa a un modelo de clasificación.

Para ambos modelos se monta un proceso donde se convierte el texto dado en un vector de frecuencia de caracteres y de palabras respectivamente, y luego se aplica el sistema de aprendizaje automático sobre los vectores que obtengamos.

Por último, usamos validación cruzada y búsqueda de hiperparámetros con el método *"GridSearchCV"*, y así podemos estudiar los mejores parámetros para obtener mejores resultados. Vamos a probar con distintos valores en distintas opciones del clasificador y del vectorizador, y así intentar mejorar los resultados.

Clasificador	Vectorizador	Datos	Pipeline	Accuracy
LinearSVC	CountVectorizer (char)	Datos en inglés	Vectorizer - Classifier	75,5%
LinearSVC	CountVectorizer (char)	Datos en español	Vectorizer - Classifier	80,10%

Figura 4-19: resultado 1ª prueba modelo de clasificación basado en caracteres

En nuestro sistema donde hay más de dos clases, y además los conjuntos de datos están desequilibrados, debemos tener en cuenta otras métricas que no sean accuracy. Usando el método de *'classification_report'* obtenemos las distintas medidas y combinaciones entre ellas, teniendo en cuenta para cada medida cada una de las clases, y sacando las medidas globales. La medida más fiable, al ser una medida armónica entre las otras dos, sería F-score.

A continuación, mostramos dos tablas con los resultados obtenidos en las pruebas con distintos modelos de clasificación, tanto con datos en inglés como en español. Los mostramos en dos tablas, una por idioma, y los mejores resultados se resaltan en color verde.

Clasificador	Vectorizador	Datos	Pipeline	Precisión	Recall	F-score
LinearSVC	CountVectorizer (char)	Inglés	Vectorizer - Classifier	76%	76%	75%
LinearSVC	CountVectorizer (word)	Inglés	Vectorizer - Classifier	78%	78%	78%
SGDClassifier	CountVectorizer (word)	Inglés	Vectorizer - Classifier	77%	77%	77%
SGDClassifier	CountVectorizer (word)	Inglés	Vectorizer – TfIdf - Classifier	73%	73%	73%
SGDClassifier	HashingVectorizer (word)	Inglés	Vectorizer – TfIdf - Classifier	73%	73%	73%

Figura 4-20: resultados de distintas pruebas de modelos de clasificación con datos en inglés.

Clasificador	Vectorizador	Datos	Pipeline	Precisión	Recall	F-score
LinearSVC	CountVectorizer (char)	Español	Vectorizer - Classifier	78%	80%	78%
LinearSVC	CountVectorizer (word)	Español	Vectorizer - Classifier	81%	83%	82%
SGDClassifier	CountVectorizer (word)	Español	Vectorizer - Classifier	80%	82%	80%
SGDClassifier	CountVectorizer (word)	Español	Vectorizer – TfIdf - Classifier	71%	75%	72%
SGDClassifier	HashingVectorizer (word)	Español	Vectorizer – TfIdf - Classifier	70%	74%	71%

Figura 4-21: resultados de distintas pruebas de modelos de clasificación con datos en español.

Se obtienen mejores resultados con modelos basados en palabras, que en caracteres. También se obtienen ligeramente mejores resultados en ambos idiomas con el clasificador LinearSVC, pero consume bastante más, y se ejecuta con bastante mayor velocidad el clasificador SGDClassifier, por lo que utilizamos también este último con asiduidad.

Otros elementos que probamos:

- Añadir al pipeline el uso de ‘*TF-IDF*’: Term Frequency - Inverse Document Frequency (tf-idf) es una técnica similar a bag-of-words, pero en la que se pesan cada una de las palabras o tokens según la relevancia que parezcan tener en el texto analizado. Se introduce como prueba con uno de los algoritmos para ver su impacto.
- Uso de vectorizador ‘*HashingVectorizer*’: sigue una estrategia basada en ‘hashing trick’, una técnica alternativa a las estrategias basadas en diccionarios como la de CountVectorizer.

Con las pruebas realizadas parece que los mejores resultados se obtienen con el uso del algoritmo de clasificación LinearSVC. Por norma general se obtienen ligeramente mejores resultados con los datos en español que con los datos en inglés, puede ser debido a los datos en sí. Obviamente para obtener mejores resultados se pueden usar muchos parámetros con distintos valores, y se podría seguir profundizando en los modelos y llegar a obtener mejores resultados. Pero por razones de tiempo y de coste computacional que supondría no se entra en mayor profundidad.

- Notebook ‘*ScikitLearn_seleccion_modelo*’:
 - Carga de los datasets almacenados en ‘.csv’, para utilizar estos datos para entrenar los modelos de clasificación.
 - Limpieza de los textos. Aplicamos las funciones de limpieza de textos ya probadas.
 - Generación de los conjuntos de train y test con los datos de ambos idiomas.
 - Generación y entrenamiento de los modelos. Se prueban 4 algoritmos de clasificación, con el pipeline que mejores resultados dieron anteriormente.

Clasificador	Vectorizador	Datos	Pipeline	Precisión	Recall	F-score
Naive Bayes	TfidfVectorizer (word)	Inglés	Vectorizer - Tokenizer - Classifier	76%	76%	76%
KNeighbors	CountVectorizer (word)	Inglés	Vectorizer - Tokenizer - Classifier	67%	67%	67%
LinearSVC	TfidfVectorizer (word)	Inglés	Vectorizer - Tokenizer - Classifier	77%	77%	77%
Arboles decisión	CountVectorizer (word)	Inglés	Vectorizer - Tokenizer - Classifier	69%	69%	69%

Figura 4-22: resultados de distintas pruebas de modelos de clasificación con datos en inglés.

Clasificador	Vectorizador	Datos	Pipeline	Precisión	Recall	F-score
Naive Bayes	CountVectorizer (word)	Español	Vectorizer - Tokenizer - Classifier	80%	81%	79%
KNeighbors	CountVectorizer (word)	Español	Vectorizer - Tokenizer - Classifier	63%	65%	61%
LinearSVC	TfidfVectorizer (word)	Español	Vectorizer - Tokenizer - Classifier	82%	84%	82%
Arboles decisión	CountVectorizer (word)	Español	Vectorizer - Tokenizer - Classifier	72%	74%	73%

Figura 4-23: resultados de distintas pruebas de modelos de clasificación con datos en español.

En este segundo notebook se elimina código repetitivo modularizándolo todo en funciones, y se prueban 4 algoritmos distintos de clasificación: máquinas vector soporte, árboles de decisión, Naive Bayes y K Nearest Neighbours (KNN). El mejor resultado con las opciones configuradas se obtiene con el algoritmo de SVMs, así que en el caso de probar en streaming con Scikit-learn, usaríamos este algoritmo. Lo ideal para evitar repetir código y tener las pruebas lo más limpias posible, es salvar el modelo final a disco, y posteriormente cargarlo en Spark Streaming para utilizarlo con los datos en streaming.

4.3.4 Spark ML

Como Spark nos proporciona una librería de machine learning, y aunque ya con las pruebas realizadas con Scikit-learn podríamos aplicarlo en Spark Streaming, vamos a realizar el trabajo de entrenamiento de modelos de clasificación con Spark ML.

Trabajamos en el siguiente notebook: *'Spark_MLlib_analisis_sentimiento'*, con los siguientes pasos:

- Carga de los datasets almacenados en *'csv'*, para usar estos datos para entrenar los modelos.
- Limpieza de los textos. Aplicamos las funciones de limpieza de textos necesarias para eliminar de los textos todos los caracteres y palabras sin valor para el análisis de sentimiento.
- Generación de los conjuntos de train y test con los datos de ambos idiomas.
- Generación y entrenamiento de los modelos.

Para evaluar los modelos tendremos en cuenta la métrica F-score, que sería la adecuada dada la distribución en clases de los datos que tenemos, un problema de clasificación multiclase.

Clasificador	Vectorizador	Datos	Pipeline	F-score
LogisticRegression	HashingVectorizer	Inglés	Vectorizer - Classifier	71,62%
LogisticRegression	CountVectorizer	Inglés	Vectorizer - Classifier	72,75%
LogisticRegression	CountVectorizer	Inglés	Vectorizer - Tfidf - Classifier	73,85%
Naive Bayes	CountVectorizer	Inglés	Vectorizer - Tfidf - Classifier	73,40%
Random Forest	CountVectorizer	Inglés	Vectorizer - Tfidf - Classifier	55,39%
Naive Bayes	CountVectorizer	Inglés	Vectorizer - Tfidf - Classifier	73,40%
LogisticRegression	CountVectorizer	Inglés	Vectorizer - Tfidf - Classifier	76,05%

Figura 4-24: resultados distintas pruebas de modelos de clasificación en Spark ML, con datos en inglés.

Clasificador	Vectorizador	Datos	Pipeline	F-score
LogisticRegression	HashingVectorizer	Español	Vectorizer - Classifier	75,88%
LogisticRegression	CountVectorizer	Español	Vectorizer - Classifier	76,51%
LogisticRegression	CountVectorizer	Español	Vectorizer - Tfidf - Classifier	77%
Naive Bayes	CountVectorizer	Español	Vectorizer - Tfidf - Classifier	76,20%
Random Forest	CountVectorizer	Español	Vectorizer - Tfidf - Classifier	38,46%
Naive Bayes	CountVectorizer	Español	Vectorizer - Tfidf - Classifier	76,20%
LogisticRegression	CountVectorizer	Español	Vectorizer - Tfidf - Classifier	80,66%

Figura 4-25: resultados distintas pruebas de modelos de clasificación en Spark ML, con datos en español.

Conseguimos realizar con Spark ML todo el proceso que anteriormente se había hecho con Scikit-learn. Aunque como ya se ha comentado anteriormente, el modelo de clasificación para usarlo en

streaming se podría aplicar con el desarrollo realizado con la librería de scikit-learn, probamos y finalmente nos quedaremos con Spark ML, ya que es el framework que se ha elegido para el procesamiento de los datos, y también porque se han obtenido unos resultados muy similares, por lo que cualquiera de las opciones la vemos igual de válida para el objetivo del proyecto.

El clasificador que mejores resultados nos da sería ‘logistic regression’. Se prueban algunos algoritmos de clasificación distintos de los vistos con Scikit-learn, ya que en Spark ML no todos los algoritmos ni métricas están operativos y soportados.

Para conseguir estos resultados, utilizamos técnicas como cross validation, y búsqueda de hiperparámetros en rejilla, para buscar los mejores hiperparámetros para el modelo, usando conjuntos de validación. Durante el proceso de validación, el conjunto de datos se divide en dos (entrenamiento y validación). El conjunto de entrenamiento se utiliza para entrenar modelos con cada uno de los posibles conjuntos de valores de los hiperparámetros. Cada uno de estos modelos se evalúa sobre el conjunto de validación, y el que da los mejores resultados (de acuerdo con el Evaluator) es seleccionado.

5 Integración, pruebas y resultados

5.1 Integración y arranque del sistema

Para integrar todas las partes del sistema, y arrancarlas de forma más sencilla, rápida y limpia en local, se han generado los siguientes scripts para arrancar y apagar el sistema:

- *arrancar_sistema.sh* [58]: el primer script arranca las herramientas necesarias del sistema, que serían Nifi, MongoDB, Zookeeper, Kafka, Spark, ElasticSearch y Kibana. También abre la aplicación de MongoDB Compass, por si se quiere gestionar la BBDD, y lanza en el navegador la instancia de Nifi para poder comenzar el flujo del sistema.
- *arrancar_streaming.sh* [59]: una vez se ha arrancado el flujo desde Nifi, se ejecuta este script para lanzar los ficheros Python que arrancan los consumidores Kafka de los topics, y almacenan los datos en las tablas de MongoDB. También abre en el navegador tanto pyspark con jupyter notebook para poder codificar y probar Spark, como la interfaz de Kibana para ver la visualización de los datos.
- *apagar_sistema.sh* [60]: script que apaga las distintas herramientas del sistema.

Por tanto, lo primero que haríamos sería lanzar desde la terminal el primer script, y una vez esté todo arrancado y se abra en el navegador la interfaz de Nifi, lanzaremos el flujo de ingesta de los datos pulsando el botón correspondiente en las opciones de Nifi. Ejemplo de lanzamiento de uno de los scripts desde consola: `./arrancar_sistema.sh`.

En este punto, ya estarían ingestándose al sistema los datos desde twitter, por lo que el siguiente paso sería arrancar el segundo script, que lanzará los scripts que comenzaran a consumir los datos del sistema en streaming. También se lanzarán en el navegador las siguientes herramientas que harán falta para probar el sistema como sería el notebook para ejecutar el código de Spark, y la interfaz de Kibana para la visualización de los datos.

5.2 Pruebas y resultados del sistema

En el último notebook con código desarrollado, se ha implementado la arquitectura lambda elegida para el sistema, y se puede realizar la prueba de concepto del sistema completo. El modelo final, generado y entrenado en la capa batch, se utiliza en la capa speed que sería donde en streaming se van recogiendo los datos que llegan al sistema, se procesan, y se les calcula el sentimiento.

En el notebook llamado `'Spark_MLlib_structured_streaming_consola'` [61], se prueba la capa speed del sistema, y se siguen los siguientes pasos:

- Carga del modelo con mejores resultados en los entrenamientos previos.
- Conexión desde Spark Structured Streaming con Kafka para obtener los datos que llegan en streaming a los dos topics.
- Preprocesado y aplicación del modelo a los datos recogidos en streaming.
- Visualización por consola de los datos después de la aplicación del modelo, ya con el sentimiento calculado.
- Inserción de los datos en MongoDB y ElasticSearch, para poder explotarlos en Kibana.

Cuando ya se han recogido los datos de Kafka para ambos topics, les vamos aplicando las funciones de preprocesado y limpieza, y posteriormente el modelo cargado para predecir el sentimiento de cada tweet, se lanza por consola cada uno de los procesos streaming y se podrían ver los datos procesados como los ejemplos de las siguientes imágenes:

Batch: 1

id	timestamp	texto	interacciones	probabilidad	predict
[1169158607767818240]	[2019-09-04 18:02:06]	[RT @royzkingin: Here is my attention that you want so bad.			
[1169158607767818240]	[2019-09-04 18:02:06]	[1. It is my mood and i have the right to decide it	[0.7339178919792175]	[0.0]	[2.0]
[1169158607767818240]	[2019-09-04 18:02:06]	[2. This is Twitter and i can...]	[0.0]	[0.575866889630127]	[2.0]
[1169158607755867393]	[2019-09-04 18:02:06]	[RT @JadineNATION: Make sure to log on to your FB accounts later at 538pm for Pond's FB Live!			
[1169158607734284289]	[2019-09-04 18:02:06]	[RT @BesshaHuggensNadine: https://t.co/2wkkarUba9 10	[0.5739529788192749]	[2.0]	[2.0]
[1169158607734284289]	[2019-09-04 18:02:06]	[RT @BesshaHuggensNadine: A view from the Starwars #MilleniumFalcon? Can @astro_luca make the Kessel Run in less than 12 parsecs? We don't know about that...]	[0.0]	[0.4971981644638432]	[0.0]
[1169158607746883584]	[2019-09-04 18:02:06]	[RT @Annahmugure: Eating with your siblings waiting for them to face the other side you steal meat🍖. https://t.co/C851r4U7bY	[0.0]	[0.6518546287427979]	[2.0]
[1169158607738462208]	[2019-09-04 18:02:06]	[RT @AlexandJose: Alex and Jose !	[0.0]	[0.7459386120872498]	[2.0]
[1169158607738462208]	[2019-09-04 18:02:06]	[RT @eoplesvote_uk @AlistairBurtUK Thank you 🇺🇸	[0.0]	[0.868897988278656]	[2.0]
[1169158607755186177]	[2019-09-04 18:02:06]	[RT @stunner_thay12 I love you mom	[0.0]	[0.755328893661499]	[2.0]
[1169158607746797569]	[2019-09-04 18:02:06]	[RT @Few Signals For The day. #HappyTrading #TradeSensibly https://t.co/XXQuHjigs]	[0.0]	[0.5819871425628662]	[2.0]

Figura 5-1: Ejemplo de salida por consola de datos en inglés procesados y con el sentimiento

Batch: 2

id	timestamp	texto	interacciones	probabilidad	predict
[1169159824998763265]	[2019-09-04 18:06:49]	[RT @cacoteomundial: #NowPlaying on #CacoteoRadio #Reggaeton #NDappNeeded Listen Now Ele A El Dominio Ft. Osquel - Gracias A Dios - https://...	[0.0]	[0.9751598238945887]	[2.0]
[1169159824998763265]	[2019-09-04 18:06:49]	[RT @Cristinagallach: Preparando la sesión inaugural del curso 2019-20 de la @UQUniversidat con @px_xala y todo el equipo técnico. La #Uni...	[0.0]	[0.7962883946384321]	[2.0]
[1169159824998763265]	[2019-09-04 18:06:49]	[RT @noticiascyl: #Valladolid #Sociedad ENTREVISTA 🗣️ Charlamos con @LauraSerranoB , una artista vallisoletana que ha conseguido arrancar, con...	[0.0]	[0.8646965622981917]	[2.0]
[1169159824998763265]	[2019-09-04 18:06:49]	[RT @thayspenalver: Tengo años preguntando lo mismo: cómo duerme en paz un país con presos políticos? Y cuántos que lo han leído están en es...	[0.0]	[0.9572432841168213]	[0.0]

Figura 5-2: Ejemplo de salida por consola de datos en español procesados y con el sentimiento

La información que se ha decidido sacar de los datos tras aplicar el modelo sería la siguiente:

- *ID*: identificador único del tweet.
- *Texto*: el texto del tweet. Se ha usado una función para ver que, si el registro viene con el atributo “truncated” a “true”, hay que ir al atributo “full_text” y coger ese texto en lugar del texto que viene en el atributo “text”, ya que el tweet tendría una longitud mayor a la normal y el valor del campo “text” viene cortado.
- *Interacciones*: sería la suma de replies, citas, retweets y favoritos de cada tweet.
- *Probabilidad*: % con la que el modelo ha decidido que el sentimiento del tweet tenga el valor asignado.
- *Prediction*: el valor del sentimiento calculado por el modelo para cada tweet.
- *Timestamp*: fecha y hora del procesado de los datos.

Se ha decidido sacar esta información porque sería la de mayor valor para el análisis y visualización que se va a realizar. En un futuro si se decidiera ampliar los análisis que se fueran a realizar a los datos, se estudiaría la inclusión de más información de la que traen los tweets.

Una vez ya se están recogiendo los tweets y se les calcula el sentimiento en streaming pudiendo mostrarse por consola como se ha visto anteriormente, también se envían tanto a MongoDB como a Elasticsearch. Cuando ya están los datos en Elasticsearch, tanto en inglés como español y con el sentimiento calculado, se puede ir a Kibana y ver la visualización realizada prácticamente en tiempo real, actualizando a medida que lleguen datos nuevos.

Para enviar los datos a MongoDB en streaming, se deben usar funciones “foreach_batch” ya que todavía no hay soporte directo entre PySpark y MongoDB en Structured Streaming para poder hacer las inserciones directamente.

En las colecciones de la BBDD para guardar datos en streaming con el sentimiento calculado, pueden verse los datos almacenados con el formato o atributos descritos anteriormente:

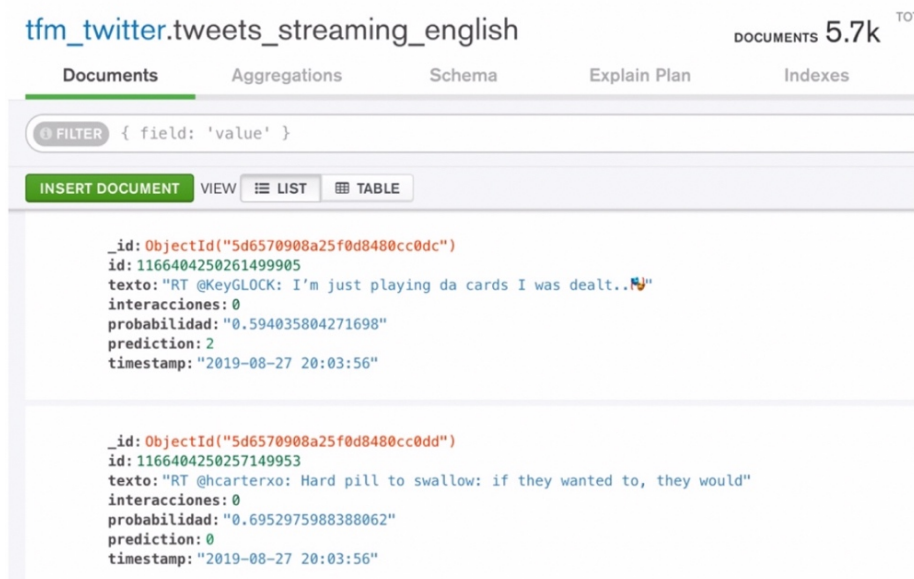


Figura 5-3: Ejemplo de datos almacenados en streaming en MongoDB con el sentimiento calculado

Aunque con el uso de MongoDB y Tableau podrían analizarse los datos con cierto refresco, es mejor solución para un sistema en streaming el uso de Elasticsearch y Kibana, para refrescar casi al momento las visualizaciones generadas con los datos que vayan llegando al sistema y procesándose.

Para insertar los datos en streaming en Elasticsearch si se puede realizar de forma directa entre Structured Streaming con Python, y Elasticsearch. Al hacer la inserción en Elasticsearch, se debe generar el índice para esos documentos, al cual se añade la fecha y hora de la inserción. Se van a crear los índices con el siguiente patrón:

- Datos en inglés: `"tweets_sentiment_eng_*"`
- Datos en español: `"tweets_sentiment_spa_*"`

Una vez creados los índices e insertados los datos en Elasticsearch, en Kibana el primer paso para poder hacer visualizaciones sería generar estos dos patrones de índices.

Si abrimos en el navegador la url para ver los índices creados en Elasticsearch, se pueden apreciar los índices creados en las pruebas en streaming del sistema, y en cada índice se puede ver cierta información como el número de documentos almacenados en el mismo.

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	tweets_sentiment_eng_20190830-081704	4Pa7tFojRYSV-5GpxU3a0A	1	1	460	0	319.7kb	319.7kb
green	open	.kibana_task_manager	l4sPxdowTPi6YUf1sd8CA	1	0	2	0	53.7kb	53.7kb
yellow	open	tweets_sentiment_spa_20190830-172246	d_b1FraKRTeJgQtu0HJAZA	1	1	189	0	139.8kb	139.8kb
yellow	open	tweets_sentiment_eng_20190904-101744	nd2aay1bSJ2f4xe0h68j6w	1	1	445	0	325.8kb	325.8kb
green	open	.kibana_1	nX32Ixj9SiCf-N2Ewskvpg	1	0	6	1	171.6kb	171.6kb

Figura 5-4: imagen de los índices generados en Elasticsearch

Una vez ya están los datos en Elasticsearch nos iríamos a la url de Kibana, donde se ha generado la siguiente visualización: un tablero canvas donde se puede ver una gráfica con las distintas clases de sentimiento en las que se clasifican los tweets (positivos, negativos y neutros), y para cada clase un contador con el número de tweets que pertenecen a esa clase dentro del índice en cuestión, sean datos en inglés o español. Además, un contador donde aparece el número total de datos procesados que han llegado a la herramienta, para cada idioma.

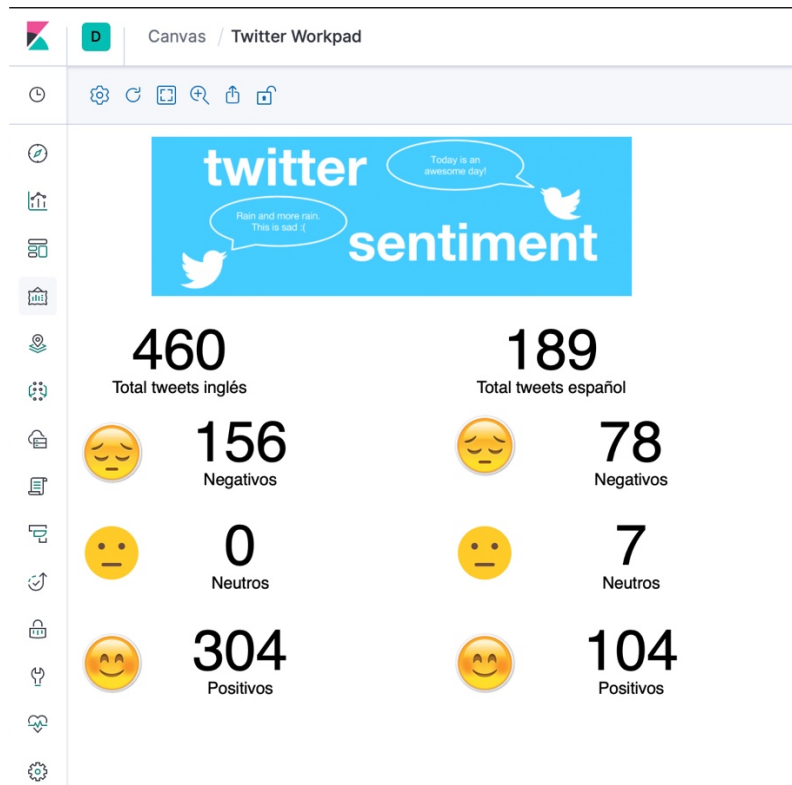


Figura 5-5: imagen de visualización en Kibana con el recuento por sentimiento

Esta visualización se puede ir refrescando automáticamente y con un tiempo cercano al streaming, con lo cual se iría viendo como van cambiando los registros.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Con este desarrollo se quería por un lado trabajar con los datos de twitter, hacerlo en Streaming -que es algo que cada vez está más en auge-, y además poner en práctica gran parte de las tecnologías y herramientas vistas en el Máster como podrían ser: Apache Nifi, Spark o Kafka, así como algoritmos de Machine Learning, en este caso aplicados al procesado de lenguaje natural. Como todo esto se ha podido implementar y probar, el resultado del trabajo es satisfactorio, al menos en cuanto a los objetivos planteados inicialmente.

El trabajo realizado se ha llevado a cabo en una máquina local, por lo cual no se ha implementado el sistema en un clúster, ni se ha trabajado con unas prestaciones muy altas, o lo suficientemente altas como para trabajar con una gran cantidad de datos. Aun así, aunque se haya hecho en una máquina local, la arquitectura usada es nativamente escalable, por las tecnologías utilizadas, por lo que pensamos que se ha demostrado satisfactoriamente la implementación y uso del sistema propuesto, tanto de forma práctica como teórica.

Se ha podido comprobar con la práctica realizada, como la recogida, análisis y preprocesado de los datos es una de las tareas más tediosas y que más tiempo llevan en trabajos relacionados con Big Data y Data Science. Para poder conseguir mejores resultados en el mismo tiempo o incluso menos tiempo de trabajo, sería buena idea contar con mejores máquinas, ya que es algo que dictamina en gran parte hasta donde se puede llegar o que resultados se pueden obtener.

También ha sido satisfactoria la decisión de herramientas y tecnologías utilizadas en el sistema, debido en mayor parte a la facilidad y potencia al mismo tiempo que otorgan por ejemplo Apache Nifi para la ingesta de datos en un sistema, o Kafka para la transmisión de estos a un sistema Streaming. Aunque se destaque su facilidad de uso, por razones de tiempo no se ha podido profundizar en estas tecnologías, que, aunque sean sencillas de implementar, tienen muchas más capacidades y opciones que no se han podido probar y que harían el sistema más potente y complejo.

El campo de estudio del trabajo, el procesamiento del lenguaje natural y análisis del sentimiento del texto es un campo muy interesante, pero al mismo tiempo muy complejo. Se está conforme con la elección del tema de estudio, pero lógicamente por su complejidad y amplitud de su alcance, haría falta bastante más tiempo del disponible y estimulado para el trabajo, para poder adquirir los conocimientos necesarios y para ponerlos en práctica profundizando todo lo que la materia permite.

6.2 Trabajo futuro

Sobre el trabajo futuro que podría hacerse para mejorar y ampliar el trabajo, se centraría en los siguientes puntos:

- Infraestructura del sistema: implantación del sistema en un clúster, con máquinas más potentes, para poder trabajar con más datos y hacer análisis más complejos que requieran más poder de computación.
 - Se podría implementar el sistema con la utilización de máquinas virtuales de tipo Cloudera, simulando un clúster entre ellas.
 - Otra opción sería utilizar algún tipo de solución cloud para el montaje del sistema.
 - Por último, la última opción sería el uso de un clúster ya montado como sería alguno de los disponibles en la universidad.
- Herramientas utilizadas: en algunas de las herramientas utilizadas como podrían ser “Nifi”, “Kafka” o “MongoDB”, podría profundizarse más en ellas, y por ejemplo introducir temas de

paralelismo, que en el caso de montar el sistema en un clúster sería algo para tener en cuenta de cara a la eficiencia del sistema.

- Modelos de Machine Learning:
 - Al realizar las pruebas de clasificación del sentimiento con modelos, se podría probar tanto con más tipos de algoritmos de clasificación, como el uso de más parámetros y mayor cantidad de valores posibles para estos.
 - Probar con redes neuronales para predecir el sentimiento de los tweets.
- Información de los tweets y análisis a realizar: por lo que se quería probar en este trabajo, lo que nos interesaba realmente es el texto del tweet, y algún atributo más, por lo tanto, se ha desechado la mayor parte de información que traen estos datos. Si se recogiera y tratara una mayor parte de los atributos de estos datos, se podrían llevar a cabo una mayor cantidad de análisis y de mayor complejidad.
Algunas opciones:
 - Podría introducirse en algún punto del sistema un filtro para filtrar los tweets según una temática, y poder hacer los análisis y ver los resultados de un tema en concreto. Sería útil para analizar el sentimiento que provoca cierto tema entre los usuarios de la red social, y en base a ello poder tomar medidas.
 - Si se tuviera en cuenta la información relativa a usuarios de los tweets, se podría estudiar la relación entre los usuarios y sus tweets, y como puede influir con el sentimiento. Para este tipo de análisis, podrían usarse tecnologías orientadas a grafos, como podría ser la BBDD de tipo NoSQL “Neo4j”.
- Visualización: en la parte final del sistema, la visualización y explotación, se ha realizado una prueba simple de concepto, viendo la conexión entre Spark en Streaming y ElasticSearch con Kibana, para visualizar los datos de forma online. Con Tableau igualmente, se probó la conexión entre Tableau y la BBDD de MongoDB para poder hacer visualizaciones offline de los datos históricos recogidos de Twitter, o los datos que se hayan ido almacenando en la base de datos con el sentimiento calculado. En ambas herramientas podría ampliarse el trabajo realizado, y hacer más visualizaciones y más complejas, ya que ambas herramientas otorgan un sinfín de posibilidades.

Referencias

- [1] **Github, TFM_Master_BigData_DataScience**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/
- [2] **Wikipedia, Procesamiento de lenguajes naturales**. [En línea]. Disponible: https://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales
- [3] **Analítica de negocios, diferencias entre datos estructurados y no estructurados**. [En línea]. Disponible: <https://www.analiticanegocios.com/guias/diferencias-datos-estructurados-no-estructurados/>
- [4] **itop tecnología y negocio, Big Data: ¿Cuáles son los datos no estructurados generados por máquinas? ¿Y por las personas?**. [En línea]. Disponible: <https://www.itop.es/blog/item/big-data-cuales-son-los-datos-no-estructurados-generados-por-maquinas-y-por-las-personas.html>
- [5] **El blog de Kyocera, Diferencia entre datos estructurados y no estructurados**. [En línea]. Disponible: <https://smarterworkspaces.kyocera.es/blog/diferencia-datos-estructurados-no-estructurados/>
- [6] **Wikipedia, Análisis de sentimiento**. [En línea]. Disponible: https://es.wikipedia.org/wiki/Análisis_de_sentimiento
- [7] **Medium, Gradient Talks: Análisis de las Elecciones Generales del 26J en Twitter, desde la perspectiva del Big Data**. [En línea]. Disponible: <https://medium.com/gradient-talks/análisis-elecciones-generales-26j-españa-desde-la-perspectiva-de-los-medios-sociales-e01e55e51cc2>
- [8] **Lambda Architecture**. [En línea]. Disponible: <http://lambda-architecture.net>
- [9] **Apache NiFi**. [En línea]. Disponible: <https://nifi.apache.org/>
- [10] **Apache Kafka, A distributed streaming platform**. [En línea]. Disponible: <https://kafka.apache.org/>
- [11] **MongoDB, The database for modern applications**. [En línea]. Disponible: <https://www.mongodb.com>
- [12] **Apache Spark**. [En línea]. Disponible: <https://spark.apache.org>
- [13] **Apache Spark Streaming**. [En línea]. Disponible: <https://spark.apache.org/streaming/>
- [14] **Apache Spark MLlib**. [En línea]. Disponible: <https://spark.apache.org/mllib/>
- [15] **Elasticsearch**. [En línea]. Disponible: <https://www.elastic.co/es/>
- [16] **Twitter developers**. [En línea]. Disponible: <https://developer.twitter.com/>
- [17] **Tweepy, Documentation**. [En línea]. Disponible: <https://tweepy.readthedocs.io/en/v3.5.0/>
- [18] **Github, script 'conseguir tweets'**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/conseguir_tweets.py
- [19] **Github, Notebook 'pruebas tweets'**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/pruebas_tweets.ipynb
- [20] **Twitter developers, Documentation**. [En línea]. Disponible: <https://developer.twitter.com/en/docs.html>
- [21] **Apache NiFi, Downloads**. [En línea]. Disponible: <https://nifi.apache.org/download.html>
- [22] **MongoDB, Download Center**. [En línea]. Disponible: <https://www.mongodb.com/download-center/community>
- [23] **Github, script 'mongo creación'**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/mongo_creacion.js
- [24] **Apache Kafka, Downloads**. [En línea]. Disponible: <https://kafka.apache.org/downloads>
- [25] **Github, script 'kafka mongo spanish'**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/kafka_mongo_spanish.py
- [26] **Github, script 'kafka mongo english'**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/kafka_mongo_english.py
- [27] **Apache Spark, Downloads**. [En línea]. Disponible: <https://spark.apache.org/downloads.html>
- [28] **Maven Repository**. [En línea]. Disponible: <https://mvnrepository.com>
- [29] **Github, Notebook 'mongo spark'**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/Mongo_Spark.ipynb
- [30] **Github, Notebook 'spark streaming'**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/Spark_streaming.ipynb
- [31] **Github, Notebook 'spark structured streaming'**. [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/Spark_structured_streaming.ipynb
- [32] **Tableau**. [En línea]. Disponible: <https://www.tableau.com>
- [33] **Tableau, Downloads**. [En línea]. Disponible: <https://www.tableau.com/es-es/products/desktop/download>
- [34] **Elasticsearch, Downloads**. [En línea]. Disponible: <https://www.elastic.co/es/downloads/elasticsearch>
- [35] **Kibana, Downloads**. [En línea]. Disponible: <https://www.elastic.co/es/downloads/kibana>
- [36] **Kaggle, Twitter sentiment analysis**. [En línea]. Disponible: <https://www.kaggle.com/c/twitter-sentiment-analysis2/data>

- [37] **Github, awesome twitter data.** [En línea]. Disponible: <https://github.com/shaypal5/awesome-twitter-data#tweet-datasets-labelled>
- [38] **Sentiment140, for academics.** [En línea]. Disponible: <http://help.sentiment140.com/for-students/>
- [39] **Data world, weather sentiment.** [En línea]. Disponible: <https://data.world/crowdflower/weather-sentiment>
- [40] **Manually annotated corpora.** [En línea]. Disponible: <https://www.clarin.eu/resource-families/manually-annotated-corpora>
- [41] **Tass, Workshop on Semantic Analysis at SEPLN.** [En línea]. Disponible: <http://www.sepln.org/workshops/tass/>
- [42] **W3, Datasets.** [En línea]. Disponible: <https://www.w3.org/community/sentiment/wiki/Datasets#Spanish>
- [43] **TIMM, Córpora, Bases de datos y Recursos lingüísticos.** [En línea]. Disponible: <http://timm.ujaen.es/corpora/>
- [44] **Github, Notebook ‘generacion dataset’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/generacion_dataset.ipynb
- [45] **Github, csv ‘result spanish’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/df_result_spanish.csv
- [46] **Github, csv ‘result english neutral’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/df_result_english_neutral.csv
- [47] **Github, csv ‘result spanish neutral’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/df_result_spanish_neutral.csv
- [48] **Github, csv ‘result english no neutral’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/df_result_english_noNeutral.csv
- [49] **Github, csv ‘result spanish no neutral’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/df_result_spanish_noNeutral.csv
- [50] **Github, Notebook ‘textblob análisis sentimiento’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/Textblob_analisis_sentimiento.ipynb
- [51] **Github, Notebook ‘spacy análisis sentimiento’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/Spacy_analisis_sentimiento.ipynb
- [52] **Github, Notebook ‘scikit learn modelos caracteres palabras’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/ScikitLearn_modelos_caracteres_palabras.ipynb
- [53] **Github, Notebook ‘scikit learn selección modelo’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/ScikitLearn_seleccion_modelo.ipynb
- [54] **Github, Notebook ‘spark MLlib análisis sentimiento’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/Spark_MLlib_analisis_sentimiento.ipynb
- [55] **Textblob, Simplified Text Processing.** [En línea]. Disponible: <https://textblob.readthedocs.io/en/dev/>
- [56] **spaCy, Industrial-Strength Natural Language Processing in Python.** [En línea]. Disponible: <https://spacy.io/>
- [57] **Scikit Learn, Machine Learning in Python.** [En línea]. Disponible: <https://scikit-learn.org/stable/index.html>
- [58] **Github, script ‘arrancar sistema’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/arrancar_sistema.sh
- [59] **Github, script ‘arrancar streaming’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/arrancar_streaming.sh
- [60] **Github, script ‘apagar sistema’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/apagar_sistema.sh
- [61] **Github, Notebook ‘spark MLlib structured streaming consola’.** [En línea]. Disponible: https://github.com/bustos85/TFM_Master_BigData_DataScience/blob/master/Spark_MLlib_structured_streaming_consola.ipynb

Bibliografía

Chodorow, K. (2015). *MongoDB: The Definitive Guide, 2nd Edition*. O'Reilly Media.

Shapira, G., Narkhede, N., & Palino, T. (2017). *Kafka: The Definitive Guide. Real-Time Data and Stream Processing at Scale*. O'Reilly Media.

Gormley, C., & Tong, Z. (2015). *Elasticsearch: The Definitive Guide. A Distributed Real-Time Search and Analytics Engine*. O'Reilly Media.

Zaharia, M., & Chambers, B. (2018). *Spark: The Definitive Guide. Big Data Processing Made Simple*. O'Reilly Media.

Garillot, F., & Maas, G. (2019). *Stream Processing with Apache Spark. Mastering Structured Streaming and Spark Streaming*. O'Reilly Media.

Marz, N., & Warren, J. (2015). *Big Data. Principles and best practices of scalable real-time data systems*. Manning.

Enlaces consultados

Se indican los enlaces más relevantes que han sido consultados durante el desarrollo del trabajo.

MongoDB – Documentation

<https://docs.mongodb.com/manual/>

MongoDB – Conexión con Tableau

<https://www.mongodb.com/tableau>

MongoDB – Connector for Spark

<https://docs.mongodb.com/spark-connector/current/>

Apache NiFi – User Guide

<https://nifi.apache.org/docs/nifi-docs/html/user-guide.html>

Apache NiFi – Documentation

<https://nifi.apache.org/docs.html>

Apache Kafka – Documentation

<https://kafka.apache.org/documentation/>

Apache Spark Streaming – Documentation

<https://spark.apache.org/docs/latest/streaming-programming-guide.html>

Apache Spark Structured Streaming – Documentation

<https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>

Apache Spark MLlib – Documentation

<https://spark.apache.org/docs/latest/ml-guide.html>

Elasticsearch – Documentation

<https://www.elastic.co/guide/index.html>

Kibana – Documentation

<https://www.elastic.co/guide/en/kibana/7.2/index.html>

MonkeyLearn Sentiment Analysis

<https://monkeylearn.com/sentiment-analysis/>

DatumBox – The importance of Neutral Class in Sentiment Analysis

<http://blog.datumbox.com/the-importance-of-neutral-class-in-sentiment-analysis/>

Wikipedia – Validación cruzada

https://es.wikipedia.org/wiki/Validación_cruzada

Wikipedia – Tf-idf

<https://es.wikipedia.org/wiki/Tf-idf>

Wikipedia – Multiclass classification

https://en.m.wikipedia.org/wiki/Multiclass_classification

Wikipedia – Precision and recall

https://en.m.wikipedia.org/wiki/Precision_and_recall

Scikit-Learn – Documentation

<https://scikit-learn.org/stable/documentation.html>

Anexos

Se recogen aquí todos los scripts y el código generado durante el desarrollo del trabajo.

Toda esta información está disponible en el siguiente enlace:

https://github.com/bustos85/TFM_Master_BigData_DataScience

La información estaría dividida de la siguiente forma:

- Datasets:
 - *df_result_english*: dataset con todos los registros en inglés.
 - *df_result_spanish*: dataset con todos los registros en español.
 - *df_result_english_neutral*: dataset con registros en inglés con el mismo número de registros con sentimiento positivo, negativo y neutro.
 - *df_result_spanish_neutral*: dataset con registros en español con el mismo número de registros con sentimiento positivo, negativo y neutro.
 - *df_result_english_noNeutral*: dataset con registros en inglés con el mismo número de registros con sentimiento positivo y negativo. No hay registros con sentimiento neutro.
 - *df_result_spanish_noNeutral*: dataset con registros en español con el mismo número de registros con sentimiento positivo y negativo. No hay registros con sentimiento neutro.
- Scripts:
 - *arrancar_sistema.sh*: script para arrancar en local las distintas herramientas del sistema.
 - *arrancar_streaming.sh*: script para arrancar en local las herramientas involucradas en la parte streaming del sistema.
 - *apagar_sistema.sh*: script para apagar las herramientas del sistema.
 - *mongo_creacion.js*: script para crear las tablas en la BBDD Mongo.
 - *conseguir_tweets.py*: script Python que conecta con la API de twitter y consigue traer tweets.
 - *kafka_mongo_english.py*: script Python que consume del topic con datos en inglés, y almacena los datos en la tabla correspondiente de Mongo.
 - *kafka_mongo_spanish.py*: script Python que consume del topic con datos en español, y almacena los datos en la tabla correspondiente de Mongo.
- Notebooks:
 - *prueba_tweets.ipynb*: carga de fichero json obtenido a partir de fichero “*conseguir_tweets.py*” con las credenciales de la API, para ver el formato y atributos de los datos de Twitter.
 - *generación_dataset.ipynb*: generación de los ficheros csv finales con datos en inglés y español con etiqueta de sentimiento calculada, para entrenar los modelos.
 - *Mongo_spark.ipynb*: primeras pruebas con Mongo y Spark, conexión entre ambos, carga de datos tanto en RDDs como en Dataframes.
 - *Spark_streaming.ipynb*: pruebas con Spark Streaming, obteniendo los datos de Kafka y haciendo algunas operaciones con esos datos en streaming.
 - *Textblob_analisis_sentimiento.ipynb*: pruebas con Textblob para hacer el análisis de sentimiento.
 - *Spacy_analisis_sentimiento.ipynb*: pruebas con Spacy para generar modelos que se entrenen y sean capaces de predecir el sentimiento de un texto.
 - *ScikitLearn_modelos_caracteres_palabras.ipynb*: pruebas con Scikit Learn para generar modelos que se entrenen y sean capaces de predecir el sentimiento de un texto.

- *ScikitLearn_seleccion_modelo.ipynb*: prueba del modelo con 4 algoritmos de clasificación, y selección del que de mejores resultados para utilizarlo en streaming.
- *Spark_structured_streaming.ipynb*: pruebas con Structured streaming. Se obtienen los datos de Kafka como dataframes, y se usa el modelo de clasificación con mejor resultado en las pruebas. El dataframe resultante con el sentimiento se almacena en Elasticsearch para visualizarlo con Kibana, y en MongoDB.
- *Spark_MLlib_analisis_sentimiento.ipynb*: pruebas a realizar el preprocesado y calcular el sentimiento de los tweets con Spark ML.
- *Spark_MLlib_structured_streaming.ipynb*: pruebas con Structured streaming y Spark ML.
- *Spark_MLlib_structured_streaming_consola.ipynb*: pruebas con Structured streaming y Spark ML, para calcular el sentimiento de los tweets en streaming.