
*Facultad de Ciencias*

# Optimización de tiempo de ejecución con PySpark de Hadoop de un análisis de sentimientos de tweets

## (Optimization of the runtime of tweet sentiment analysis with Hadoop's PySpark)

**Trabajo de Fin de Máster**
**para acceder al**

**MÁSTER EN Ciencia de Datos**

**Autor: Daria Angélica Escobar Galaburda**

**Director\es: Aida Palacio Hoz**

**Septiembre - 2022**

**Resumen**

Cuando se trata del análisis y procesamiento de volúmenes grandes de datos del orden de los GB o superiores, las herramientas utilizadas usualmente como Python nativo no suelen ser suficientes si se requiere obtener resultados en un tiempo reducido. Por ello, el presente proyecto pretende demostrar la eficiencia del uso de herramientas que hacen uso de los sistemas distribuidos, en este caso PySpark.

Se realiza un análisis de sentimientos junto a su respectivo preprocesado de datos de un conjunto de *tweets* cuyo tamaño supera los 20 GB utilizando la librería TextBlob utilizando tanto Python nativo como PySpark, midiendo en ambos casos el tiempo de ejecución de operaciones como descompresión, lectura y escritura de datos a un archivo csv, modificaciones al conjunto de datos, limpieza de texto y la clasificación de sentimientos reflejados en los *tweets* en negativo, positivo o neutro.

Se realiza una comparación de los tiempos de ejecución obtenidos logrando demostrar que al adaptar el código utilizado en Python a PySpark se reduce la operación más costosa, la clasificación de sentimientos con TextBlob, de horas a menos de un segundo. Como conclusión en los resultados se obtiene una reducción final del 96% del tiempo empleado con Python, pasando de invertir casi 16 horas a tan solo 38 minutos con la ayuda de PySpark.

***Palabras clave:*** Computación distribuida, Hadoop, Apache Spark, PySpark, Python, análisis de sentimientos, NLP, TextBlob.

**Abstract**

When it comes to the analysis and processing of large volumes of data in the order of GB or more, the tools normally used such as native Python are usually not enough if you need to obtain results in a short time. Therefore, this project aims to demonstrate the efficiency using tools that make use of distributed systems, in this case PySpark.

A sentiment analysis is carried out along with its respective data preprocessing of a set of tweets whose size exceeds 20 GB using the TextBlob library with both native Python and PySpark, measuring in both cases the execution time of operations such as decompression, reading and writing data to a csv file, modifications to the data set, text cleaning and the classification of sentiments reflected in the tweets in negative, positive or neutral.

A comparison of the obtained execution times is made, demonstrating that adapting the code used in Python to PySpark reduces the most expensive operation, the classification of sentiments with TextBlob, from hours to less than a second. As a conclusion in the results, a final reduction of 96% of the time spent with Python is obtained, going from investing almost 16 hours to only 38 minutes with the help of PySpark.

***Keywords:*** Distributed Computing, Hadoop, Apache Spark, PySpark, Python, Sentiment Analysis, NLP, TextBlob.

# 1. INTRODUCCIÓN

A la hora de analizar y procesar datos, nos encontramos comúnmente con el inconveniente de que el tiempo de procesamiento depende del tamaño del conjunto de los datos utilizado y de la capacidad de nuestra máquina. Esto supone un inconveniente cuando se quieren obtener resultados en un tiempo reducido, por ejemplo, para aplicaciones en tiempo real, que necesiten el procesamiento de una gran cantidad de datos en intervalos de corta duración.

Los sistemas distribuidos permiten procesar enormes cantidades de datos en varias máquinas a la vez, reduciendo significativamente el tiempo en el que se logra obtener los resultados buscados. A su vez, Hadoop es un *framework* para aplicaciones distribuidas, trabajando ya sea en un cluster de máquinas o en la nube, que maneja el procesamiento en paralelo, dejando que el usuario se concentre en la lógica de la aplicación.

Además de reducir el tiempo a través de los sistemas distribuidos, también es posible disminuirlo aún más cargando los datos en la memoria, consiguiendo mayor velocidad de procesamiento que trabajando en disco. Hadoop posee una extensión, Apache Spark, que permite realizar esto, con una API sencilla en lenguajes conocidos como Python, R, Scala y Java.

Para poder demostrar la eficiencia del uso de estas herramientas se usa un dataset del orden de los gigabytes, el cual contiene texto que puede ser usado para un análisis. Sobre este dataset se realiza un preprocesamiento y un análisis de los datos para finalmente, a través de un entrenamiento con un método de *Natural Language Processing*, obtener un análisis de sentimientos. Este proceso se realiza de dos formas, utilizando librerías tradicionales de Python y con Apache Spark para obtener una comparación de la velocidad con el que se obtienen los resultados de los procesos.

## 1.1 Motivación

A continuación, se realizará un análisis del problema mediante un árbol de problemas y se concluirá con la definición del problema.

### *1.1.1 Situación problemática*

En la Figura 1 se muestra el árbol de problemas que ha permitido realizar un análisis de la causa y efectos que provocan un tiempo largo de procesado de una gran cantidad de datos.

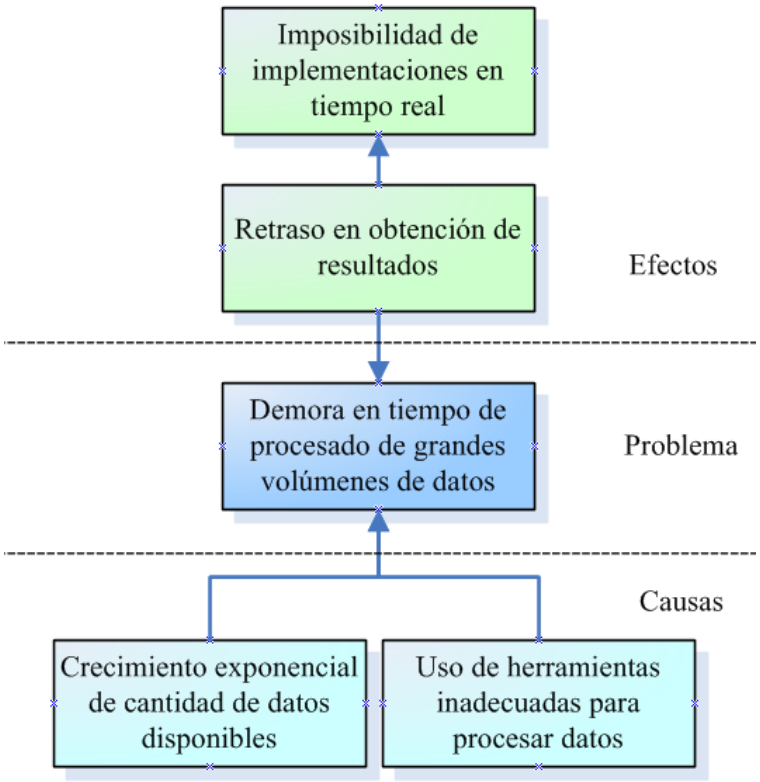


Figura 1. Árbol de problemas

Hoy en día muchas empresas, negocios e investigaciones basan muchas de sus decisiones en la enorme cantidad de datos que estos recopilan. La cantidad de datos almacenados ha ido aumentando exponencialmente en los últimos años. Esto ha dado lugar a que cada vez sean mayores los volúmenes de datos que están disponibles y son necesarios para obtener información valiosa a través del análisis y procesado de estos.

Al mismo tiempo, al aumentar la cantidad de datos, también se incrementa la cantidad de tiempo necesario para poder trabajar sobre estos, ya que el análisis requiere una mayor cantidad de procesos. En muchas aplicaciones, el tiempo reducido de obtención de resultados es primordial, ya que la información va aumentando continuamente y es necesario ir actualizando los resultados de los análisis. Por ejemplo, en aplicaciones de tiempo real, si el tiempo de análisis es mayor al tiempo en el que se generan nuevos datos, nos encontramos con un problema ya que el resultado de los análisis quedaría obsoleto para el momento de finalización del procesado de los datos.

Por este motivo, han ido surgiendo herramientas que permiten que el tiempo que se emplee para el procesado de datos tenga una duración que posibilite obtener resultados provechosos. Sin embargo, debido a la demanda de analistas de datos, se tiene trabajando en el área muchos

principiantes e inexpertos que utilizan métodos y herramientas que no se adecuan a la cantidad de datos a analizar, causando demoras innecesarias en la obtención de resultados.

### *1.1.2 Definición del problema*

Resumiendo la situación problemática planteada, obtenemos que el uso de herramientas inadecuadas para el procesado de grandes volúmenes de datos provoca retrasos en la obtención de resultados e imposibilita la implementación de sistemas en tiempo real.

## 1.2 Objetivos

A continuación, se definen el objetivo general y los objetivos específicos del proyecto.

### *1.2.1 Objetivo general*

Demostrar la eficiencia del uso de las herramientas de computación distribuida en el procesado de grandes volúmenes de datos.

### *1.2.2 Objetivos específicos*

1. Explorar un set de datos de acceso abierto publicado en la plataforma Kaggle, del orden de los GB.
2. Familiarizarse con los datos utilizando técnicas de análisis tradicionales que se puedan adaptar a Spark posteriormente.
3. Preprocesar los datos adecuándolos para la siguiente fase.
4. Aplicar un método de clasificación de sentimientos.
5. Medir el tiempo de creación de datos, preprocesado y clasificación, tanto con Python como con PySpark y comparar los resultados obtenidos con ambas herramientas.

## 1.3 Alcances

- Se trabajará con datos reales de la red social de twitter proporcionados por un usuario de la plataforma Kaggle.
- Se utilizarán modelos de análisis de sentimientos creados previamente por terceros.
- Se medirán los tiempos de descompresión y unión de los datos, limpieza del texto y clasificación de sentimientos.

### 1.4 Límites

- No se buscará hallar el mejor método de *machine learning* para el análisis de sentimientos.
- No se enfocará en obtener la mejor clasificación para el análisis de sentimientos.
- Se demostrará que el tiempo de procesamiento se reduce significativamente con la herramienta correcta sin optimizar este tiempo al máximo.
- Los datos no se extraerán diariamente; se utilizarán los *tweets* del 28 de Febrero al 27 de Julio para un primer análisis y del 28 de Febrero al 8 de Septiembre para el segundo.

### 1.5 Estructura del proyecto

En el capítulo 2 se pone en contexto el trabajo explicando las diferentes herramientas sobre las que se va a apoyar el proyecto, así como los métodos de análisis y las definiciones necesarias para entender el desarrollo de los capítulos posteriores. En el capítulo 3 se describen los procedimientos, métodos, acciones y la información empleada para el desarrollo del problema de investigación. En el capítulo 4 se realiza la descripción detallada de las distintas etapas necesarias para realizar un procedimiento de análisis de sentimientos tanto con Python como con PySpark y se hace una comparación de los tiempos de ejecución con cada una de estas herramientas. Finalmente, en el capítulo 5 se plantean las conclusiones obtenidas respecto tanto a los resultados como el procedimiento y recomendaciones para trabajos futuros basados en el presente proyecto.

# 2. MARCO TEÓRICO

En el presente capítulo se desarrollan los fundamentos teóricos en los que se basa el presente trabajo. Se empieza con una descripción de qué es la computación distribuída y cuáles son sus ventajas. Dentro de esto se explica qué es Hadoop, HDFS y YARN y la utilidad de estos. Así mismo se explica que es Apache Spark y PySpark y los beneficios que conlleva su uso. A continuación, se aclara la definición de NLP y algunos de los conceptos importantes de este como la tokenización, NLTK y un método de NLP como el análisis de sentimientos. Para este último se exponen 2 métodos, el TextBlob y RoBERTa. También se da una breve explicación de los n-gramas. Finalmente, el capítulo cierra con la definición de regex.

## 2.1 Computación distribuida

Un sistema distribuido es una colección de dispositivos computacionales independientes que cooperan entre sí para solucionar un problema que no se puede resolver individualmente y aparece para el usuario como una sola computadora proporcionando una vista única del sistema. La agregación coordinada de estas computadoras distribuidas permite el acceso a una gran capacidad de computación [2] [16].

Los sistemas distribuidos son sistemas informáticos que contienen múltiples procesadores conectados por una red de comunicación. En estos sistemas, los procesadores se comunican entre sí mediante mensajes que se envían a través de la red. Tienen las siguientes características: sin reloj físico común, sin memoria compartida, separación geográfica y autonomía y heterogeneidad, lo que significa que cada procesador puede ejecutar un sistema operativo diferente [11] [21].

Existen diferentes herramientas para el procesamiento de datos en sistemas distribuidos. La mayoría tienen Hadoop como base. Entre los más populares encontramos Amazon EMR, Google Cloud Dataflow, Confluent, Apache Storm, Apache Spark, entre otros. Los tres primeros mencionados son herramientas de pago, mientras los últimos dos, pertenecientes a Apache, son de uso libre. Sin embargo, Spark posee una API para el desarrollo con Python, denominado PySpark, lo que lo hace sencillo de entender y de emplear. Por este motivo, a continuación, se detallan los conceptos de Hadoop y de Spark.

### *2.1.1 Hadoop*

Hadoop es el proyecto top-level de *Apache Software Foundation* que contiene los diversos subproyectos de Hadoop que surgieron de *Apache Incubator*. El proyecto Hadoop es un *framework* de código abierto para escribir y ejecutar aplicaciones distribuidas que permiten el almacenamiento, análisis y procesamiento de conjuntos de datos muy grandes. Tiene capacidades computacionales muy altas debido a la distribución de todos los procesos en clústers de *hardware* básico de bajo coste. Hadoop maneja los detalles de procesamiento, lo que deja a los desarrolladores libres para concentrarse en la lógica de la aplicación. [7] [22] [36] [19] [40] [43]

Las ventajas de Hadoop más importantes son: accesibilidad, ejecutándose en clústeres de máquinas básicas o en servicios de computación en la nube; robusto, porque está diseñado como un servicio tolerante a fallos de hardware; escalable, incrementando su computación de manera lineal para manejar datos más grandes al agregar más nodos al clúster; y simple, permitiendo a los usuarios escribir rápidamente código paralelo eficiente. [7] [19] [40]

#### *2.1.1.1 Ecosistema de Hadoop*

Hadoop utiliza dos componentes principales: modelo de programación *MapReduce* para el procesamiento de datos y el Sistema de archivos distribuidos de Hadoop (HDFS) el cual almacena los datos y proporciona una capa de abstracción entre todos los dispositivos disponibles. Además, posee la herramienta YARN, framework que permite a Hadoop soportar diversos motores de ejecución como *MapReduce*. [38]

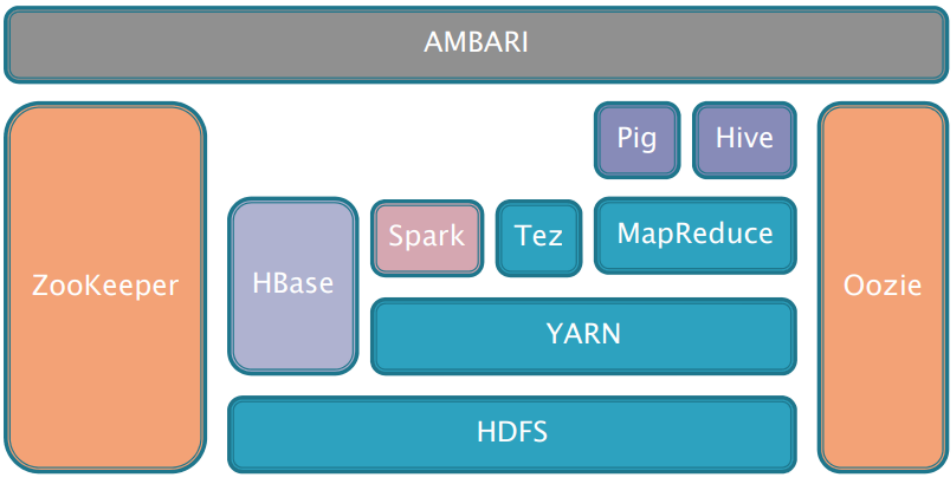


Figura 2. Ecosistema de Hadoop [4]

A la vez, Hadoop tiene un ecosistema amplio de herramientas de código abierto y patentadas mostrado en la Figura 2. Estos proyectos no necesitan usarse todos en conjunto, por lo que deben conocerse bien y determinar cuáles son los adecuados para el proyecto en Hadoop a realizarse. Algunos de estos son:

- Pig: proporciona una plataforma en Hadoop para personalizar, analizar y manipular grandes conjuntos de datos. El idioma de Pig se conoce como Pig Latin. [33]
- Hive: paquete de almacenamiento de datos creado sobre Hadoop y acceso similar a SQL para datos en HDFS y otras fuentes de entrada de Hadoop. [38]
- Tez: Permite la creación de un gráfico acíclico dirigido complejo de tareas para el procesamiento de datos.
- Spark: Mejora el paradigma *MapReduce*. [4]
- HBase: Una base de datos no relacional de código abierto construida sobre HDFS. Está desarrollado para acceso aleatorio de lectura y escritura en tiempo real a conjuntos de datos muy grandes. [38]
- Zookeeper: Servicio para la coordinación de procesos distribuidos y para la configuración compartida, es decir la redundancia en caso de fallos.
- Oozie: Planificador de *workflows*. Permite soportar varios tipos de *jobs*.
- Ambari: Proporciona una interfaz de gestión completa. [4]

#### *2.1.1.2 Clúster de Hadoop*

Un clúster de Hadoop es un conjunto de máquinas conectadas en red (conocidas como nodos) que pueden trabajar juntas en el mismo problema. El almacenamiento y el procesamiento de datos ocurren dentro de este clúster de máquinas. Diferentes usuarios pueden enviar trabajos informáticos a Hadoop desde clientes individuales, que pueden ser sus propias máquinas de escritorio en ubicaciones remotas desde el clúster de Hadoop. En un clúster de cómputo, muchos nodos pueden compartir cargas de trabajo y aprovechar un ancho de banda agregado muy grande en todo el clúster. Las capacidades computacionales y de almacenamiento de Hadoop escalan con la adición de hosts a un clúster de Hadoop; los clústers con cientos de *hosts* pueden computar fácilmente sobre volúmenes de datos de petabytes. [7] [19] [40] [43]

La arquitectura de un clúster de Hadoop es la ya muy conocida *master-slave*. Para el nodo que actúa como master se puede tener uno o más nodos que controlan los sistemas de procesamiento y almacenamiento en Hadoop, y uno o más nodos esclavos que se encargan principalmente de

almacenar datos y procesarlos. Esta arquitectura consta de los siguientes componentes principales:

- Un modelo de comunicación donde un proceso, llamado master, tiene control sobre uno o más procesos, llamados esclavos.
- El master YARN (Yet Another Resource Negotiator) realiza la programación real del trabajo para las aplicaciones YARN y funciona como administrador de recursos.
- El master *MapReduce* es un motor computacional basado en batches, responsable de organizar dónde se debe programar el trabajo computacional en los nodos esclavos. *MapReduce* se implementa como una aplicación YARN.
- El master HDFS (*Hadoop Distributed File System*) es responsable del almacenamiento de datos y la partición del almacenamiento entre los nodos esclavos y el seguimiento de dónde se encuentran los datos. [19]

#### *2.1.1.3 HDFS*

HDFS es un sistema de archivos diseñado para el procesamiento de datos distribuidos a gran escala. Cuando un conjunto de datos supera la capacidad de almacenamiento de una sola máquina física, es necesario particionarlo en varios nodos. HDFS (Sistema de archivos distribuido de Hadoop) proporciona una arquitectura distribuida para almacenamiento a una escala extremadamente grande, que se puede ampliar fácilmente escalando horizontalmente. Un conjunto muy grande de datos puede ser almacenado como un único archivo en HDFS, algo que no es posible en otros sistemas de archivos. HDFS está optimizado para un alto rendimiento y funciona mejor al leer y escribir archivos grandes (gigabytes, petabytes, etc.). [7] [19] [22] [36] [43]

Los servicios HDFS son proporcionados por dos procesos: *NameNode* se encarga de la gestión de los metadatos del sistema de archivos y proporciona servicios de gestión y control, y *DataNode* proporciona servicios de almacenamiento y recuperación de bloques. [40]

Las características clave de HDFS son:

- HDFS utiliza tamaños de bloque inusualmente grandes y optimizaciones de ubicación de datos para reducir la entrada/salida de la red. [19]
- HDFS no maneja particularmente bien el acceso aleatorio [40]. Está optimizado para el rendimiento sobre la latencia; es muy eficiente en la transmisión de solicitudes de

lectura para archivos grandes, pero pobre en la búsqueda de solicitudes para muchos archivos pequeños. [36]

- Dado que HDFS no es un sistema de archivos nativo de Unix, las operaciones estándar de lectura/escritura de archivos no funcionan en él [22]. Está optimizado para cargas de trabajo que generalmente son del tipo de escritura única y lectura múltiple. Es decir que el contenido de los archivos individuales no se puede modificar, aparte de agregar nuevos datos al final del archivo. [7]
- En lugar de manejar los fallos del disco al tener redundancias físicas en discos, HDFS usa la replicación. Cada uno de los bloques que componen un archivo se almacena en varios nodos dentro del clúster, y el *NameNode* de HDFS monitorea constantemente los informes enviados por cada *DataNode* para garantizar que las fallas no hayan dejado ningún bloque por debajo del factor de replicación deseado. [36]

#### *2.1.1.4 YARN*

Los componentes de programación de MapReduce se externalizaron y se desarrollaron como un nuevo componente llamado YARN, abreviatura de *Yet Another Resource Negotiator*. YARN es independiente del tipo de trabajo que se realiza en Hadoop; todo lo que requiere es que las aplicaciones que deseen operar en Hadoop se implementen como aplicaciones YARN. Como resultado, *MapReduce* es una aplicación YARN. [19]

Es una herramienta que permite que otros frameworks de procesamiento de datos se ejecuten en Hadoop. YARN está destinado a proporcionar una programación de carga de trabajo más eficiente y flexible, así como una función de administración de recursos, que en última instancia permitirán que Hadoop ejecute más que solo trabajos de *MapReduce*. [7]

La arquitectura de YARN se compone de los siguientes componentes:

- Administrador de recursos: El componente central de YARN es el Administrador de recursos, que gobierna todos los recursos de procesamiento de datos en el clúster de Hadoop. Es un programador dedicado a asignar recursos a las aplicaciones solicitantes. Sus únicas tareas son mantener una visión global de todos los recursos en el clúster, manejar solicitudes de recursos, programar la solicitud y luego asignar recursos a la aplicación solicitante.
- Administrador de nodos: Cada nodo esclavo tiene un Administrador de nodos, que actúa como esclavo para el Administrador de recursos. Cada administrador de nodos

realiza un seguimiento de los recursos de procesamiento de datos disponibles en su nodo esclavo y envía informes periódicos al administrador de recursos.

- Maestro de aplicaciones: Cada aplicación que se ejecuta en el clúster de Hadoop tiene su propia instancia maestra de aplicaciones dedicada. A lo largo de su vida, esta envía mensajes al Administrador de recursos con su estado y el estado de las necesidades de recursos de la aplicación. [7]

### *2.1.2 Apache Spark*

Apache Spark comenzó como un proyecto de investigación en UC Berkeley AMPLab en 2009 y fue de código abierto a principios de 2010. Muchas de las ideas detrás del sistema se presentaron en varios trabajos de investigación a lo largo de los años. Este creció hasta convertirse en una comunidad de desarrolladores y se trasladó a *Apache Software Foundation* en 2013. Hoy es desarrollado en colaboración por una comunidad de cientos de desarrolladores. [35]

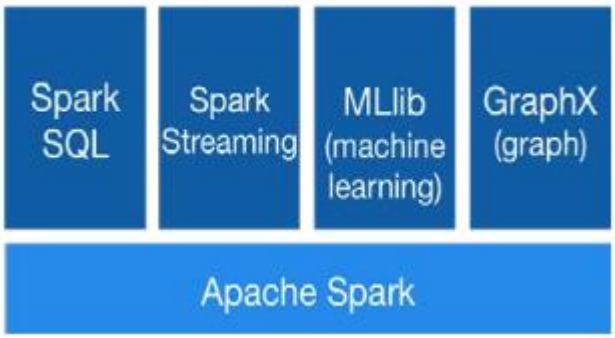


Figura 3. Stack de Apache Spark [32]

Spark es un sistema informático distribuido de propósito general y una plataforma de procesamiento de datos de código abierto que utiliza abstracción de memoria distribuida para procesar grandes volúmenes de datos de manera eficiente para cumplir con los requisitos computacionales del análisis masivo de datos [31] [42]. Spark es compatible con la computación en memoria, lo que le permite consultar datos mucho más rápido en comparación con motores basados en disco como Hadoop. Spark es una actualización avanzada de Hadoop destinada a mejorar la capacidad de análisis de este, siendo sus funciones del motor bastante más avanzadas y diferentes [32]. Spark proporciona una API de desarrollo en Python, Java, Scala y R [34]. Este sistema también proporciona una gran cantidad de herramientas de alto nivel, como la herramienta de aprendizaje automático MLlib, el procesamiento de datos

estructurados, Spark SQL, el procesamiento de gráficos, Graph X y un motor de procesamiento de flujo llamado Spark Streaming como se puede ver en la Figura 3. [32]

Spark es una de las populares plataformas en la nube de código abierto que presenta el concepto de conjuntos de datos distribuidos resistentes (RDD) para permitir el procesamiento rápido de grandes volúmenes de datos aprovechando la memoria distribuida. RDD es una colección de objetos de solo lectura particionados en diferentes nodos en el clúster para que los datos en RDD se puedan procesar en paralelo. Las operaciones en RDD colocan automáticamente las tareas en particiones, manteniendo la localidad de los datos persistentes. Sus operaciones de datos en memoria lo hacen ideal para aplicaciones iterativas. Además, Spark también proporciona dos extensiones de RDD: DataFrame y Dataset. Los usuarios de Spark pueden cambiar sin problemas entre estos a través de simples llamadas a la API. [9] [15] [42]

#### *2.1.2.1 PySpark*

PySpark es una API basada en Python que brinda acceso a Spark utilizando el lenguaje de programación Python para complementar el uso de Apache Spark. Utiliza la librería Py4J para manejar las comunicaciones con Spark. Tiene como beneficio la inclusión de una fácil integración con otros lenguajes, como Java, Scala y R. [30] [37]

Una de las principales ventajas de utilizar PySpark es la capacidad de ejecutarlo en un entorno interactivo. Proporciona una interfaz sencilla, sintaxis y lenguaje fáciles de aprender, mejor legibilidad, mantenimiento y familiaridad, una interfaz simple, fácil y completa, y muy preferida para implementar algoritmos de aprendizaje automático. [34]

## 2.2 Procesamiento del lenguaje natural

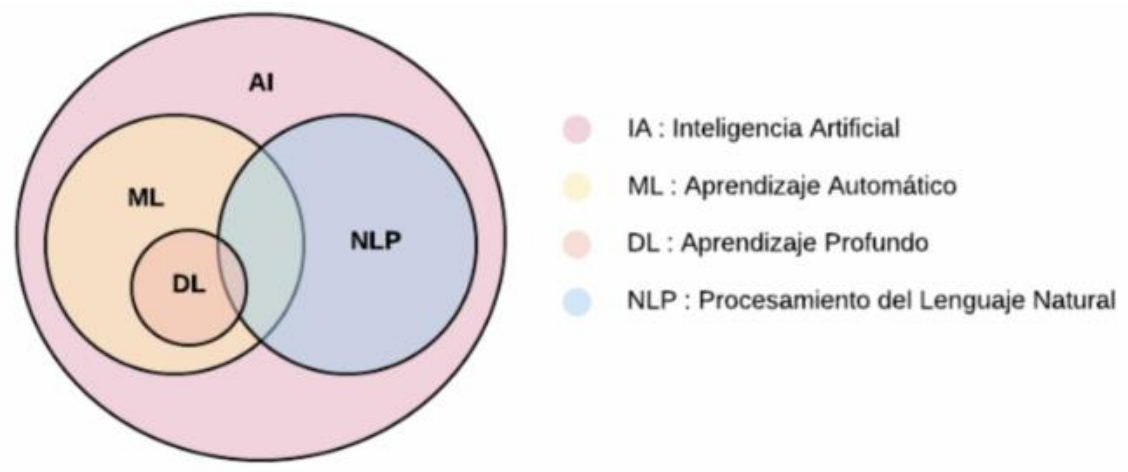


Figura 4. El NLP se solapa con otros campos dentro de la inteligencia artificial [3]

La lingüística computacional, también conocida como procesamiento del lenguaje natural (NLP), como se observa en la Figura 4, es el subcampo de la ciencia computacional que se ocupa del uso de una colección de técnicas computacionales para aprender, comprender, analizar y producir contenido del lenguaje humano, haciendo que el lenguaje humano sea accesible para las computadoras. [6] [18]

El NLP comenzó en la década de 1950 como la intersección de la inteligencia artificial y la lingüística [26]. En la última década, el procesamiento del lenguaje natural se incorporó a nuestra vida diaria: la traducción automática es omnipresente en la web y en las redes sociales; la clasificación de texto evita, por ejemplo, que nuestras bandejas de entrada de correo electrónico queden inoperativas cuando se produce una avalancha de spam; los motores de búsqueda han ido más allá de la coincidencia de cadenas y el análisis de redes hacia un alto grado de sofisticación lingüística; Los sistemas de diálogo proporcionan una forma cada vez más común y eficaz de obtener y compartir información. Estas diversas aplicaciones se basan en un conjunto común de ideas, basándose en algoritmos, lingüística, lógica, estadísticas y más. [8]

Una de las principales limitaciones de NLP actual es el hecho de que la mayoría de los recursos y sistemas de NLP están disponibles sólo para lenguajes de recursos elevados (HRL), como inglés, francés, español, alemán y chino, los sistemas lingüísticos computacionales aún tienen múltiples propósitos: el objetivo puede ser ayudar a la comunicación humano-humano, como en la traducción automática (MT); ayudar a la comunicación hombre-máquina, como con agentes conversacionales; o beneficiar tanto a los humanos como a las máquinas mediante el análisis y el aprendizaje de la enorme cantidad de contenido del lenguaje humano que ahora está disponible en línea. [18] [26]

### *2.2.1 Tokenización*

Después de importar textos, el siguiente paso habitual es convertir el texto legible por humanos en algo que una máquina pueda leer pre procesando el texto. El procesamiento previo reduce significativamente el tamaño de los documentos de texto de entrada y las acciones involucradas en este paso son la determinación del límite de la oración, la eliminación de palabras vacías específicas del lenguaje natural, la tokenización y la lematización. Entre estos, la acción más esencial e importante es la tokenización. Las decisiones tomadas durante la tokenización tienen un efecto significativo en el análisis posterior. [1] [41]

En NLP, la tokenización es el proceso de dividir el texto de entrada, que para una computadora es solo una larga cadena de caracteres, en subunidades, llamadas tokens. Un token es una parte de un todo, por lo que una palabra es un token en una oración y una oración es un token en un párrafo. Estos tokens luego se introducen en los pasos posteriores de procesamiento del lenguaje natural, como por ejemplo en el análisis morfológico y en el etiquetado de clases de palabras. [13] [29]

### *2.2.2 NLTK*

*Natural Language Toolkit* (NLTK) es una completa biblioteca de Python para procesamiento de lenguaje natural y análisis de texto que tiene como objetivo crear programas de Python para trabajar con datos de lenguaje humano. NLTK se usa a menudo para la creación rápida de prototipos de programas de procesamiento de texto e incluso se puede usar en aplicaciones de producción. [28] [29]

Proporciona interfaces fáciles de usar para más de 50 corpus y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, lematización, etiquetado, análisis y razonamiento semántico, contenedores para bibliotecas NLP de potencia industrial, y un foro de discusión activo. Los tipos de datos incluyen tokens, etiquetas, fragmentos, árboles y estructuras de características. [24] [28]

Diseñado originalmente para la enseñanza, se ha adoptado en la industria para la investigación y el desarrollo debido a su utilidad y amplitud de cobertura y es ideal para estudiantes que están aprendiendo NLP o realizando investigaciones en NLP o áreas estrechamente relacionadas. [24] [29]

### *2.2.3 Análisis de sentimientos*

El análisis de sentimientos (SA), una de las tecnologías emergentes clave, brinda a las personas la libertad de analizar una gran cantidad de contenido generado por usuarios disponible en la web. El análisis de sentimientos se puede definir como un estudio detallado de NLP que incluye el estudio computacional de opiniones y sentimientos expresados en formato textual, rastreando el estado de ánimo del público sobre un tema, producto o servicio específico. [5] [17]

El análisis sentimental consiste en determinar la tendencia o actitud de un comunicador a través de la polaridad contextual de su escritura o habla. Incluye numerosas tareas, por ejemplo,

extracción de conclusiones, caracterización de evaluación, arreglo de subjetividad, resumen de suposiciones o detección de spam. El SA planea examinar las suposiciones, actitudes, mentalidades, conclusiones, sentimientos, etc. de los individuos hacia artículos, personas, sujetos, asociaciones y administraciones, etc. [14] [17]

#### *2.2.3.1 TextBlob*

TextBlob es una biblioteca de Python utilizada para analizar los datos presentes en formato de texto. Proporciona una API simple para sumergirse en tareas comunes de procesamiento del lenguaje natural (NLP), como el etiquetado de partes del discurso, la extracción de frases nominales, el análisis de sentimientos, la clasificación, la traducción, el reconocimiento de voz y más. [17]

Funciona en el marco de NLTK. Nos devuelve los niveles de polaridad y subjetividad. La polaridad se encuentra entre (-1,+1) donde -1 significa sentimientos negativos y +1 significa sentimientos positivos. La polaridad se invierte con palabras de negación. La subjetividad se encuentra entre (0,1). La subjetividad se ocupa de la información en la detección si está basada en hechos o es personal. A mayor subjetividad, mayor cantidad de datos personales y fácticos. [5]

#### *2.2.3.1 RoBERTa*

En el artículo de investigación de Liu Yinhan et al, los autores explican que BERT actualmente no está bien entrenado y muestran que se pueden obtener resultados sustancialmente mejores con algunas modificaciones. Los autores cambiaron el nombre de su modelo BERT a RoBERTa, que significa Un enfoque previo al entrenamiento BERT robustamente optimizado (robustly optimized BERT pre-training approach). Además de modificar algunos de los hiper parámetros de BERT, las diferencias clave se relacionan con la forma diferente en que se entrena el modelo. [23] [27]

Las siglas BERT significan representación de codificador bidireccional de transformadores (*Bidirectional Encoder Representations from Transformers*). Se trata de una técnica basada en redes neuronales para el pre-entrenamiento del procesamiento del lenguaje natural (NLP). Los modelos BERT pueden interpretar el contexto completo de una palabra analizando las palabras que vienen antes y después, lo que resulta muy útil para comprender realmente lo que quiere transmitir el texto. [25]

### *2.2.4 N-gramas*

Los N-gramas de textos se utilizan ampliamente en tareas de minería de texto y procesamiento de lenguaje natural. Son secuencias continuas de palabras, símbolos o tokens en un documento. Entran en juego cuando tratamos con datos de texto en tareas de NLP. Esta secuencia de elementos, suele consolidar un texto que tiene sentido al leerse o podría formar parte de otro texto, es decir que no son un conjunto ordenado sin sentido. La "N" hace referencia a la cantidad de elementos que tendrán los n-gramas. [10] [39]

## 2.3 Regex

Una expresión regular (regex o regexp para abreviar) es una cadena de texto especial que permite crear patrones que ayudan a hacer coincidir, ubicar y administrar combinaciones en cadenas. Cuando se intenta comprender las expresiones regulares por primera vez, parece que se trata de un idioma diferente. Sin embargo, dominar las expresiones regulares puede ahorrarle horas si trabajase con texto o si se necesita analizar grandes cantidades de datos. En la Tabla 1 se muestran algunos ejemplos de regex. [12] [20]

| Caracter | ¿Qué hace? | Ejemplo | Matches |
|---|---|---|---|
| ^ | Coincide con el inicio de línea | ^abc | abc, abcdef.., abc123 |
| $ | Coincide con el final de línea | abc$ | my:abc, 123abc, theabc |
| [a-z] | Coincide con cualquier caracter entre la "a" y la "z" | [b-z] | bc, mind, xyz |

Tabla 1. Ejemplos de regex

# 3. MARCO METODOLÓGICO

En este capítulo se describen los procedimientos, métodos, acciones y la información empleada para el desarrollo del problema de investigación.

## 3.1 Tipo de investigación

El presente trabajo de máster sigue un proceso cuantitativo. Se realizará una medición y cuantificación del tiempo que tarda en realizar un análisis de sentimientos sobre un set de datos de *tweets* sobre el conflicto actual entre Rusia y Ucrania con PySpark y se lo comparará con el tiempo en que tarda en realizar este mismo procedimiento con Python sobre la plataforma *Jupyter Notebook*.

El alcance del proyecto buscará generar conocimientos que sean aplicables a cualquier otro proyecto que implique el análisis y procesamiento de una gran cantidad de datos.

## 3.2 Objeto de investigación

El objeto de investigación es el tiempo demorado en realizar un análisis y posterior procesamiento de un set de datos de tweets sobre el conflicto actual entre Rusia y Ucrania con diferentes métodos.

## 3.3 Fuentes de información

Las fuentes de información empleadas en el proyecto son de tipo secundaria, tales como: libros, artículos e información de Internet relacionados con el área de investigación. Además, se manejan datos reales de *tweets* realizados por personas que hacen uso de la plataforma de *Twitter*, los cuales fueron proporcionados por un usuario de la plataforma *Kaggle*.

## 4. MARCO PRÁCTICO

En este capítulo se realiza la descripción de las distintas etapas necesarias para alcanzar el objetivo propuesto del presente proyecto.

### 4.1 Selección de lenguaje

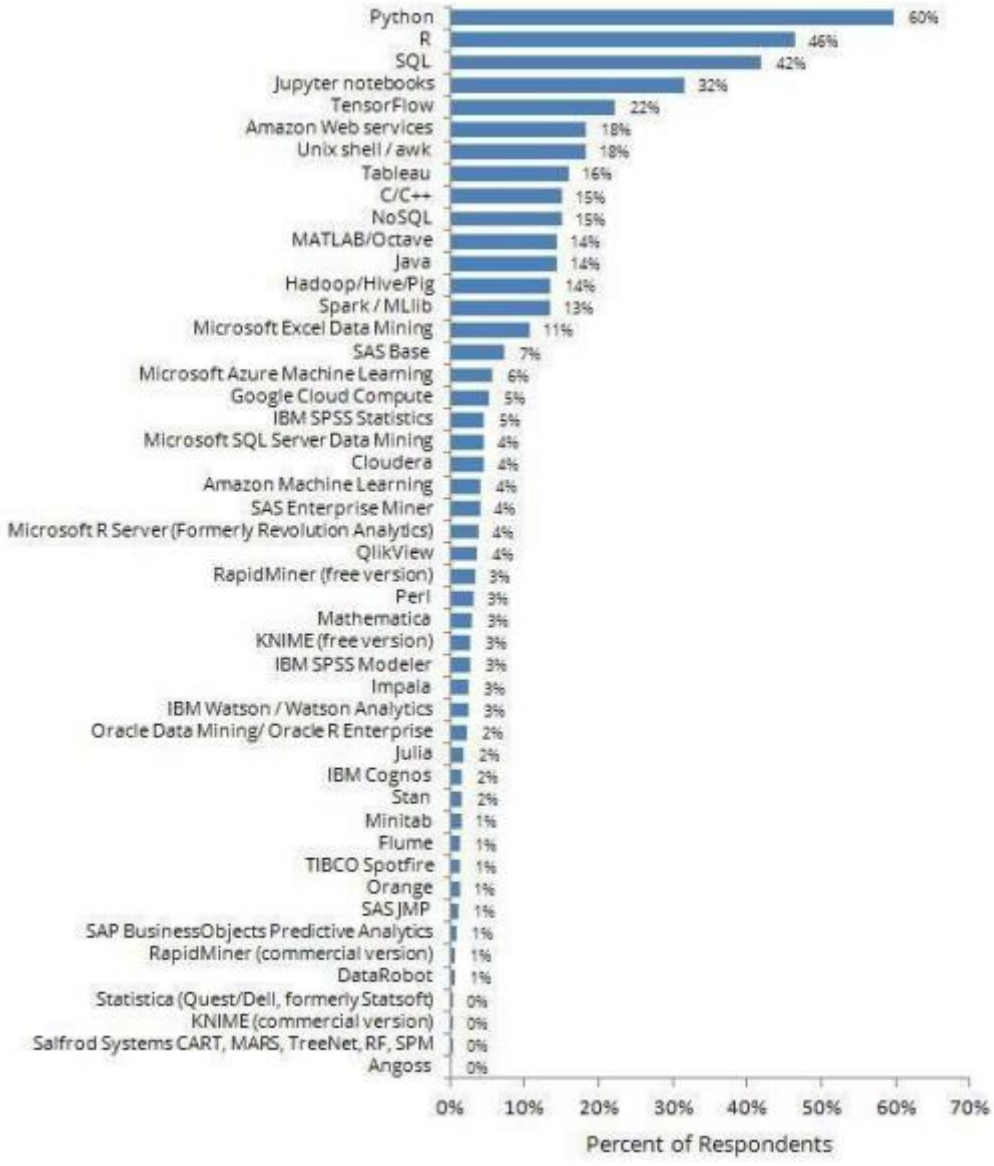


Figura 5. Herramientas, tecnologías y lenguajes utilizados en el 2017 para *data science*

La Figura 5 muestra una comparativa realizada en base a datos de Kaggle 2017 "*the State of Data Science and Machine Learning study*", donde se encuestó a un total de 10153 personas respecto a las herramientas de *data science*, análisis, tecnologías o lenguajes que estuvieron usando en el 2017. Los resultados se muestran en términos porcentuales.

Se puede destacar la gran cantidad de personas que prefieren el uso de Python y R para aplicaciones relacionadas con data science y *Machine Learning*. Los mismos mencionados son utilizados por el 60% y el 46% correspondientemente. Por este motivo, se utiliza Python como lenguaje para la realización de este proyecto, dado que al estar en el primer puesto existe una mayor comunidad y por lo tanto mayor facilidad de encontrar fuentes e información.

### 4.2 Conjunto de datos

El conjunto de datos empleado en el presente proyecto comprende información sobre los usuarios de *Twitter* y sus *tweets* referentes al conflicto actual entre Rusia y Ucrania ya que es un tema actual que afecta a muchos sectores. Esta información contiene fechas, recuentos de algunos datos, entre otros correspondientes a los *tweets* publicados y los usuarios que los publicaron. En la Tabla 2 se muestra la información que se proporciona en el set de datos, junto con una breve descripción sobre el significado de sus valores.

El set de datos se obtuvo de la plataforma Kaggle bajo el título de "*Ukraine Conflict Twitter Dataset*". Consta de datos a partir del 28 de Febrero de 2022 y se actualiza diariamente. Al momento de la descarga de datos (27 de Julio de 2022) el set incluía 11 GB de información comprimidos en archivos gzip y divididos en varios *batches* más pequeños de datos de entre 30 a 140 MB.

| **Columna** | **Descripción** |
|---|---|
| **userid** | Número de identificación del usuario |
| **username** | Nombre del usuario |
| **acctdesc** | Descripción de la cuenta de usuario |
| **location** | Ubicación de la cuenta |
| **following** | Número de usuarios a los que sigue esta cuenta |
| **followers** | Número de usuarios que siguen esta cuenta |
| **totaltweets** | Número de *tweets* publicados por esta cuenta |
| **usercreatedts** | Fecha y hora en el que fue creado este *tweet* |
| **tweetid** | Número de identificación del *tweet* |
| **tweetcreatedts** | Fecha y hora en el que fue creado este *tweet* |

| | |
|---|---|
| **retweetcount** | Número de veces que fue retweeteado este *tweet* |
| **text** | Texto contenido en el *tweet* |
| **hashtags** | Hashtags añadidos al *tweet* |
| **language** | Idioma utilizado en el texto del *tweet* |
| **cordinates** | Coordenadas de la ubicación |
| **favorite_count** | Número de veces que este *tweet* fue marcado como favorito |
| **extractedts** | Fecha y hora en la que esta información fue extraída. |

Tabla 2. Descripción de los campos del conjunto de datos

### *4.2.1 Unión de datos*

El conjunto de datos viene distribuido en archivos comprimidos de entre 30 a 140 MB, cada uno conteniendo la información de *tweets* de un día. Para hacer la labor más sencilla posteriormente y evitar el uso de muchos bucles, es necesario descomprimir estos archivos y unirlos en un único conjunto de datos en formato csv.

El total de este nuevo archivo es de 26,6 GB debido a que, al descomprimirlo, su tamaño aumenta considerablemente. Contiene 18 columnas, 17 las ya mencionadas y un índice, y 44.993.322 filas.

El tiempo empleado para descomprimir y unir los archivos en un único dataframe es de 8042,75 segundos, junto con otros 1553,55 segundos para guardarlo en un archivo csv. Medimos también cuánto tiempo demora en importar el archivo nuevamente y obtenemos un total de 3022,1 segundos.

### *4.2.2 Exploración y preprocesamiento de datos*

Cada columna presenta datos de tipo *object*, exceptuando *favorite_count* el cual tiene datos de tipo *float64*. Para poder explorar los datos con mayor facilidad, es necesaria una modificación en el tipo de datos de las columnas de tiempo *usercreatedts*, *tweetcreatedts* y *extractedts*, convirtiéndolas al tipo *datetime*.

Con el cambio de tipo realizado podemos obtener información sobre la frecuencia de los *tweets* mensualmente. En la Figura 6 podemos observar que a medida que pasa el tiempo, la cantidad de *tweets* sobre el tema disminuye mostrando que existe una pérdida de interés por parte de los

usuarios de *Twitter*. En cuanto al mes de febrero, se tiene un conteo bajo debido a que el set de datos con el que se trabaja comienza el día 28 de febrero.

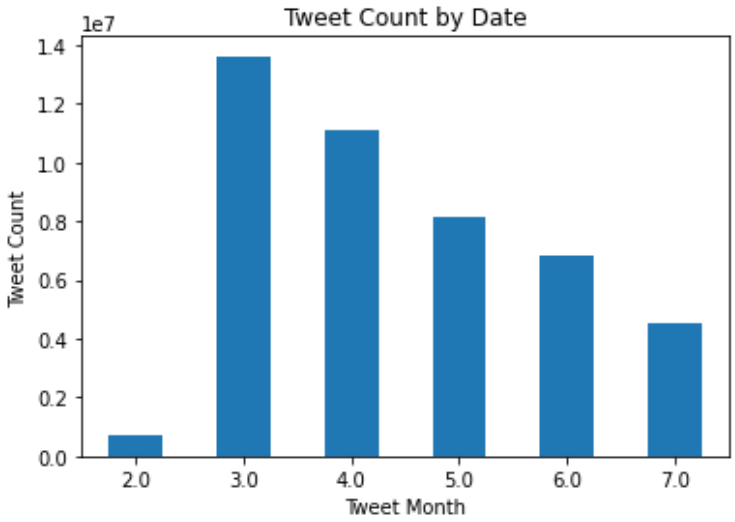


Figura 6. Conteo de *tweets* mensuales

Para posteriormente realizar una exploración del texto contenido en los *tweets* y un análisis de sentimientos, lo mejor es quedarse con un solo idioma. Para ello, realizamos el conteo de *tweets* por idioma para observar cual es el predominante. La Figura 7 muestra como el inglés es el lenguaje más utilizado en este conjunto de datos, seguido por el alemán, francés e italiano. Por este motivo, filtraremos las entradas que estén en inglés. Esta operación toma 98,38 segundos, dando como resultado un nuevo conjunto de datos que contiene 30246122 filas, lo cual representa el 67.22% de datos del set inicial.

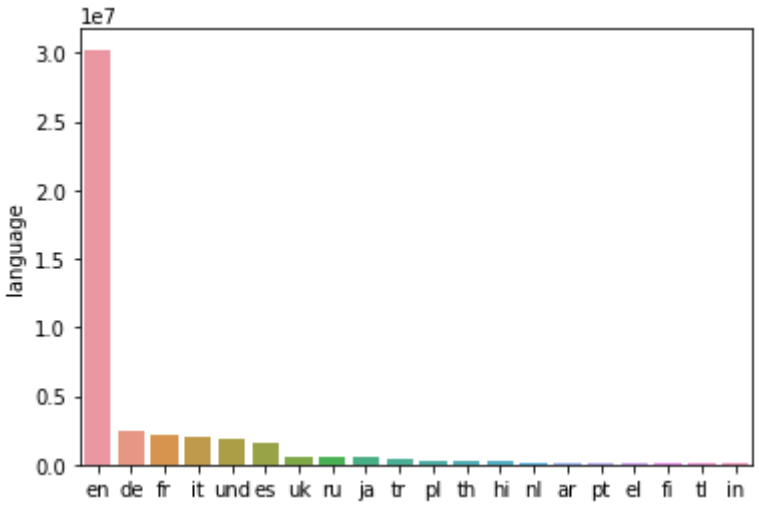


Figura 7. Conteo de *tweets* por idioma

El siguiente paso es revisar qué columnas contienen valor NaN (Not a Number). Encontramos que las columnas *coordinates*, *location* y *acctdesc* tienen un 99,97%, 41,72% y 20,8% de sus

datos como NaN respectivamente. Por lo tanto, prescindiremos de la columna coordinates ya que está prácticamente no contiene información alguna. En cuanto a las otras dos columnas no se realiza ninguna modificación hasta que se realice alguna acción sobre estas.

### *4.2.3 Limpieza de datos*

Es necesario quitar, modificar u ordenar algunos de los datos, sobre todo de la columna *text* que es sobre la que se realiza el análisis de sentimientos.

Lo primero es no tener filas duplicadas, lo cual teóricamente no debería suceder, sin embargo, se realiza una verificación a partir de la columna *tweetid*, el cual debería ser único y se conoce que si existen filas duplicadas, por lo cual se procede a eliminar los duplicados.

Para poder pasar como entradas los textos contenidos en los *tweets* a un algoritmo de análisis de sentimientos, este debe tener la mayor claridad posible en el texto. Para ello es necesario quitar texto que no aporte información a la hora de ejecutar el análisis de sentimientos como las URLs y menciones a otros usuarios.

En *Twitter* existen los *hashtags*, palabras clave que las personas utilizan para marcar el tema del contenido que están compartiendo en las redes sociales. Estos llevan por delante el # y a continuación el texto. Eliminaremos únicamente el símbolo # ya que el texto contenido en *hashtag* podría incluir información relevante. Por otro lado, existen *tweets* que únicamente contienen *hashtags* y que sin ellos quedarían vacíos.

Algunos usuarios utilizan letras repetidas para expresar gritos o resaltar alguna palabra, como por ejemplo “hoooola”. Los algoritmos de análisis de texto pueden tomar esto como otra palabra, por lo cual lo cortaremos a dos letras ya que existen varias palabras en inglés que contienen dos letras repetidas seguidas.

El texto de los *tweets* contiene “\n” denotando saltos de línea y “&amp” para el símbolo “&”, el cual normalmente se utiliza para reemplazar la palabra “and”. Cambiaremos los saltos de línea por un espacio y pondremos la palabra “and” textualmente.

Para quitar las URLs, menciones y letras repetidas definimos patrones regex (*regular expression*):

- ➔ Patrón para URL: r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
- ➔ Patrón para usuarios: '@[^\s]+'

- ➔ Patrón de repetición: r"(.)\1\1+"
- ➔ Patrón de reemplazo de repetición: r"\1\1"

Los emoticonos son reconocidos por algunos de los algoritmos de análisis de texto, por lo cual no será necesario convertirlos a texto como es el caso cuando se utilizan métodos de *machine learning* tradicionales.

Finalmente reseteamos el índice ya que al quitar duplicados y *tweets* en otros idiomas que no sean inglés este queda desconfigurado y con esto, la columna de *text* queda prácticamente lista para pasar por un análisis de sentimientos con las modificaciones hechas como se muestra en el ejemplo en la Figura 8.

```
df_clean["text"][2]
```

```
'Russia to invest $40 billion in Iran’s oil sector URL'
```

Figura 8. Ejemplo de tweet con URL remplazada

### 4.3 Análisis de sentimientos

En la actualidad, existen varios algoritmos creados para el análisis de sentimientos. En su mayoría estos algoritmos son para textos cortos, como son en este caso los *tweets*. En este caso haremos uso de dos librerías para procesamiento textual de datos descritas en el anterior capítulo para el análisis de sentimientos de *sets* de *Twitter*: TextBlob y RoBERTa.

#### *4.3.1 TextBlob*

Utilizando la biblioteca de Python TextBlob, determinamos si el sentimiento que emite cada *tweet* es negativo, positivo o neutro. TextBlob nos devuelve un valor denominado polaridad, el cual se encuentra entre -1 y 1. Por este motivo, primero es necesario definir una función que, a partir de umbrales especificados, determine qué etiqueta de sentimiento tenga el *tweet*. Intentamos que la mayoría de los *tweets* sean clasificados como negativos o positivos, pues al ser clasificados como neutros no obtenemos tanta información como en los otros dos casos. Para ellos definimos como umbrales los siguientes valores:

- ➔ Si la polaridad es mayor a 0,01 es positivo.
- ➔ Si la polaridad es menor a -0,01 es negativo.
- ➔ Si la polaridad es igual o se encuentra entre -0,01 y 0,01, es neutro.

Una vez aplicada la función TextBlob y clasificados los sentimientos de acuerdo a los umbrales establecidos, realizando una estadística de todas las etiquetas obtenemos 10.221.500, neutros, 12.972.766 positivos y 7.031.220 negativos. Como se aprecia en la Figura 9, la tercera parte de los *tweets* fueron clasificados como neutros y de los restantes dos tercios, tenemos más positivos que negativos, lo cual viene a ser una sorpresa tomando en cuenta que el tema es una guerra. Cerca de la mitad de los *tweets* son clasificados como positivos, mientras que sólo un 23,3% son clasificados como negativos, dejando así un 33,8% de los tweets en neutro.

El tiempo de clasificación fue de 707 minutos. El empleo de casi 12 horas para obtener un etiquetado donde un tercio de los datos son clasificados como neutros no parece ser una buena inversión, ya que da la impresión de que no obtenemos suficiente información respecto a los sentimientos de los usuarios sobre el conflicto entre Rusia y Ucrania.

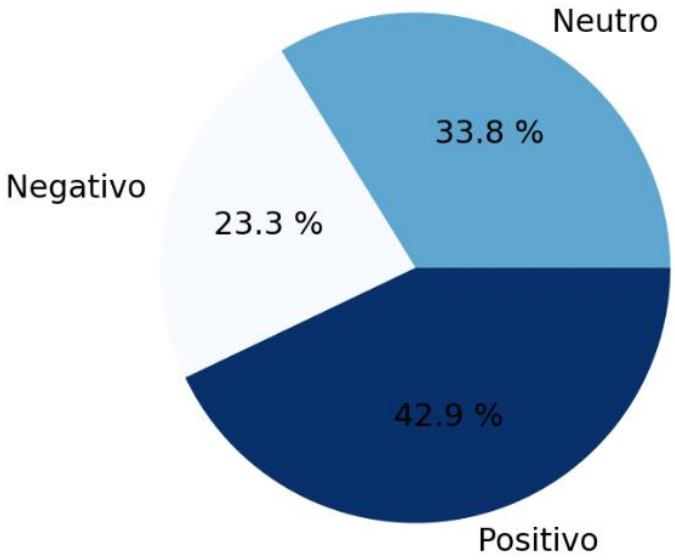


Figura 9. Gráfico de tortas de la clasificación de sentimientos con TextBlob en Python

### *4.3.2 RoBERTa*

Utilizamos la técnica de RoBERTa para volver a catalogar los sentimientos de nuestro set de datos, sin embargo, en esta ocasión utilizaremos solo uno de los archivos y no todo nuestro conjunto, ya que esta técnica suele durar un tiempo mucho más amplio que el TextBlob usado previamente. Por esto, aplicaremos la técnica a uno de los archivos al azar, en este caso el que corresponde a los *tweets* del 26 de Julio.

Creamos una instancia del modelo RoBERTa y a su vez un tokenizador, el cual se encarga de preparar las entradas para el modelo en el formato apropiado. Definimos una función que toma

todo el *dataframe* y retorna todos los *tweets* codificados. Truncamos a un *lenght* máximo de 512 *tokens* porque es el máximo que permite este modelo y aplicamos este encoder a los *tweets*.

Al codificar los *tweets*, lo que obtenemos como resultado en realidad es un diccionario con dos campos: los *input ids* que son los *tokens*, llamados tensores en este caso, obtenidos de convertir el *tweet* en números; y *attention mask*, campo que le indica al modelo qué *tokens* deberían recibir más atención y cuáles no.

Una vez codificados, pasamos como entrada los *tweets* al modelo obteniendo así como resultado 3 números entre el 0 y el 1 correspondientes a las etiquetas "negativo", "neutro" y " positivo". Estos números representan qué calificación tiene el *tweet* bajo cada sentimiento. En nuestro caso, elegiremos como sentimiento aquella etiqueta que tenga el mayor valor para cada *tweet*. Para entender mejor, ponemos como ejemplo el siguiente *tweet*:

> *"Chicago. Meeting of wounded Ukrainian fighters who flew in the us for prosthetics.\n\nI am very grateful to everyone who helps our guys to recover!\n\nStandWithUkraine URL"*

Después de codificarlo y pasarlo al modelo obtenemos los siguientes valores para cada etiqueta:

- ➔ negativo 0,0136930505
- ➔ neutral 0,15507655
- ➔ positivo 0,83123046

Podemos ver que el porcentaje positivo domina con una gran diferencia sobre el resto. Por ello, este *tweet* recibe la etiqueta de "positivo". Podemos comprobar que esta etiqueta es correcta tan solo leyendo el texto del *tweet* en el cual se puede apreciar de manera clara el positivismo del usuario.

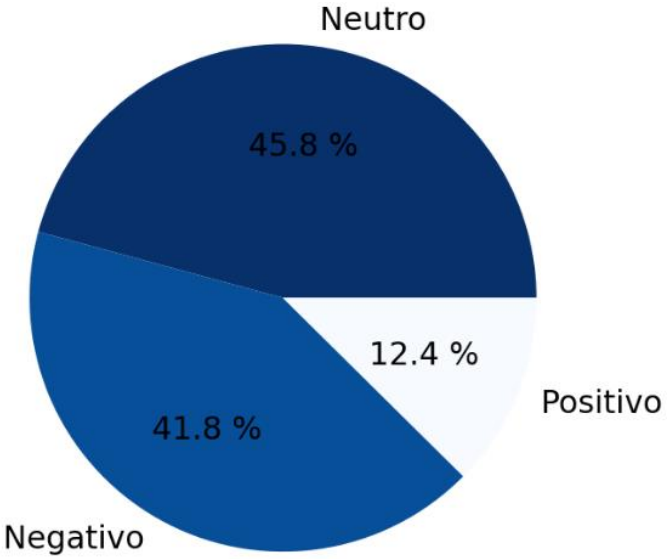


Figura 10. Gráfico de tortas de la clasificación de sentimientos con RoBERTa en Python

Una vez asignadas las etiquetas de acuerdo al mayor valor entre las tres posibilidades, realizamos el conteo de cuantos *tweets* hay por etiqueta. Este archivo contiene 76.187 *tweets*, de los cuales RoBERTa clasifica 34.893 como neutros, 31.844 como negativos y 9.450 como positivos. En la Figura 10 podemos ver que, en esta ocasión, casi la mitad fueron clasificados como neutros y, en la segunda mitad, un muy alto porcentaje de *tweets* negativos dejando tan solo un 12,4% cómo *tweets* clasificados como positivos.

El tiempo empleado para obtener estas etiquetas es de 205 minutos, es decir más de 3 horas. Tomando en cuenta que tenemos los datos de casi 5 meses cada uno de unos 30 días, el etiquetado de todo nuestro set de datos demoraría más de 450 horas, lo que equivale a prácticamente 19 días.

### *4.3.3 Comparación de TextBlob con RoBERTa*

El tiempo empleado con RoBERTa es claramente mucho mayor que el tiempo empleado con el método de TextBlob y, a pesar de que para este archivo aparenta presentar resultados más acordes a lo que uno espera, es decir una mayor cantidad de opiniones negativas respecto al conflicto existente entre los dos países, no podemos asegurar que el resultado final sobre todo el conjunto tenga esta misma distribución.


Figura 11. *Wordcloud* de *tweets* clasificados como negativos con RoBERTa

Por otra parte, como se ve en el ejemplo del *tweet* en la sección 4.3.2, que un comentario sea clasificado como positivo no implica que esté a favor de la guerra, sino puede simplemente estar mostrando apoyo a la gente que está sufriendo por la situación. Para tener mayor claridad sobre qué tipo de comentarios tenemos clasificados bajo cada etiqueta, podemos valernos de los n-gramas y las nubes de palabras. Por ejemplo, podemos crear nubes para ver qué palabras aparecen con más frecuencia en los *tweets* negativos usando ambos tipos de métodos. La Figura 11 enseña el *wordcloud* de *tweets* negativos con RoBERTa, mientras la Figura 12 nos muestra las palabras frecuentes halladas en los *tweets* clasificados como negativos con TextBlob. En ambos casos se encierra con cuadros rojos aquellas palabras que está claro tienen una connotación negativa.

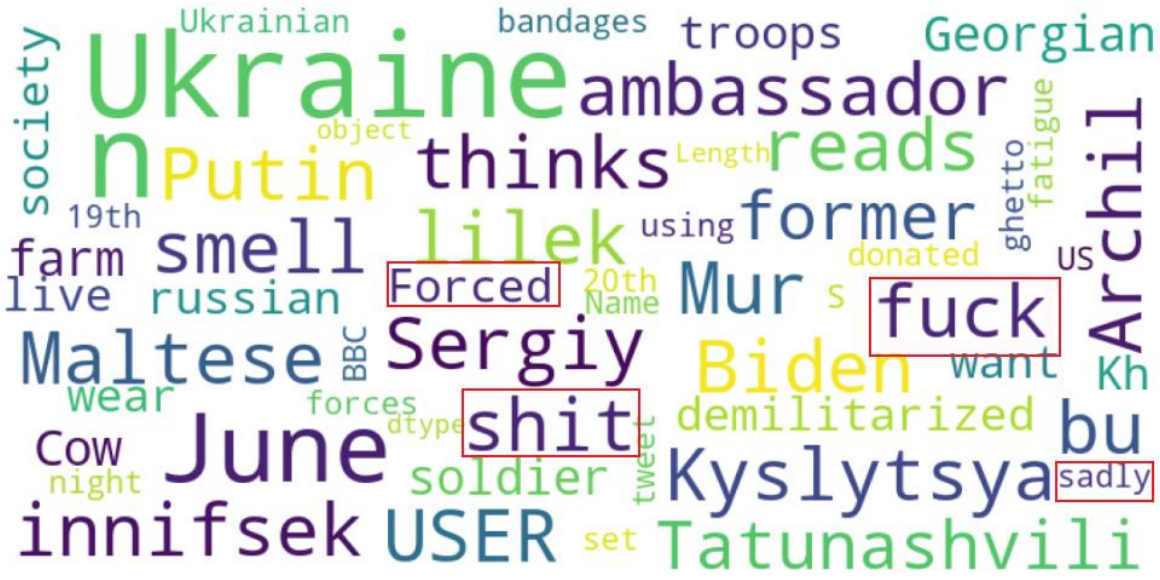


Figura 12. *Wordcloud* de *tweets* clasificados como negativos con TextBlob

Revisemos también que wordcloud se genera a partir de los *tweets* clasificados como positivos por *TextBlob*, ya que estos son los que predominan según este método. La Figura 13 enseña las palabras más frecuentes en los *tweets* positivos. Prestando atención a algunas palabras como "*hope*", "*welcomed*", "*strong*" "*warmly*", "*worth*", "*great*", podemos suponer que la mayoría de estos *tweets* en realidad son de apoyo y aliento, por lo cual podemos entender que una gran parte de los usuarios en este dataset, publican *tweets* de respaldo a los afectados.

Utilizamos los n-gramas para ver qué tipo de texto es clasificado como neutro. Revisamos que n-gramas son los más importantes o, en otras palabras, más frecuentes. La Figura 14 muestra el top 10 n-gramas de 6 palabras generados a partir de los *tweets* que RoBERTa clasificó como neutros. Observamos que el de mayor frecuencia es una repetición de la palabra *user*, que viene a ser el reemplazo de las menciones a otros usuarios. Que este n-grama se cree nos da a entender que existen varios *tweets* que probablemente tienen un conjunto grande de menciones que provocan que sean clasificados como neutros. También podemos ver en los siguientes 4 n-gramas que varios de los *tweets* neutros son tan sólo informativos, por lo cual no pueden definirse como positivos ni negativos.


Figura 13. *Wordcloud* de *tweets* clasificados como positivos con TextBlob

Finalmente, es difícil definir cuál de los dos métodos obtiene mejores resultados, ya que uno enseña que existe mayor porcentaje de positivos mientras que el otro enseña un mayor porcentaje de negativos. Sin embargo, dado que *TextBlob* retorna resultados en un tiempo mucho menor, tiene un menor porcentaje de *tweets* clasificados como neutros y su implementación es más sencilla, nos quedaremos con este método para posteriormente reproducirlo con la herramienta de computación distribuida, PySpark.

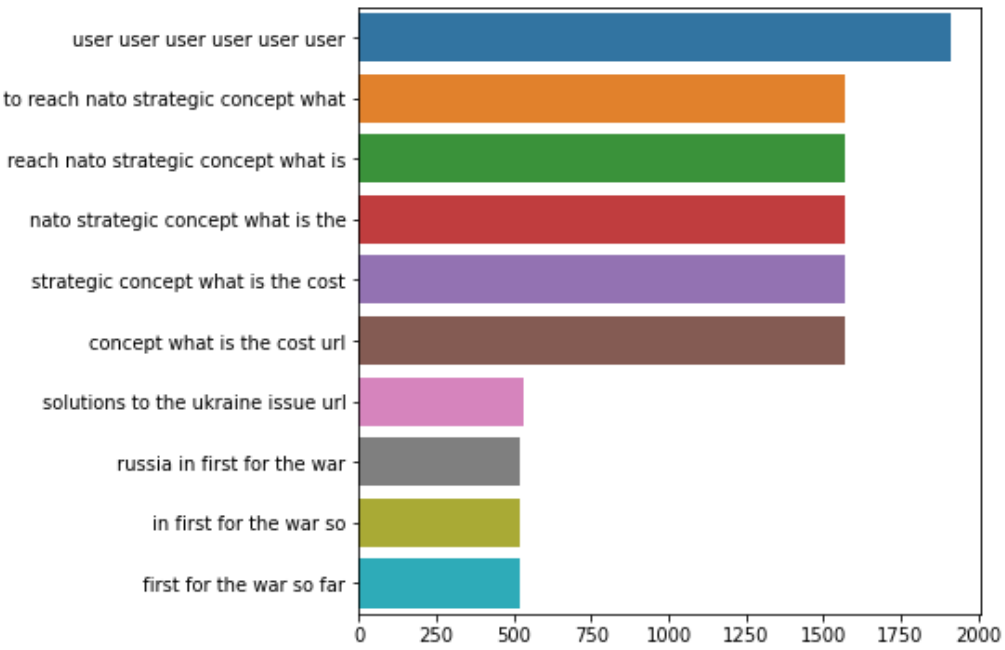


Figura 14. Top 10 n-gramas de 6 palabras de *tweets* neutros con RoBERTa

## 4.4 Análisis de Dataset en un clúster PySpark

Como se pudo observar previamente, el tiempo que demora en realizarse todo el proceso de análisis de datos de un dataset del orden de los GB es excesivo. Si tomamos en cuenta que este dataset se actualiza diariamente y se quisiera realizar una aplicación de análisis de sentimientos en tiempo real, no sería factible. Por este motivo, en esta sección realizamos el mismo procedimiento que en la sección 4.3, pero en esta ocasión se lo adapta utilizando la API de PySpark para intentar conseguir tiempos más bajos de la ejecución de los diferentes pasos para el análisis de sentimientos del set de datos elegido.

### *4.4.1 Características de nuestro clúster PySpark sobre Apache Hadoop*

El grupo de computación avanzada del Instituto de Física de Cantabria (IFCA, CSIC - UC) participa en el proyecto FACE (*FAir Computational Epidemiology*) junto a otros institutos como el IFISC, CEAB y IEGD pertenecientes al Consejo Superior de Investigaciones Científicas (CSIC). Este proyecto provee tanto un modelo como una plataforma de simulación donde los científicos pueden explotar una gran variedad de datasets para responder de manera rápida a los brotes epidémicos a través de servicios de infraestructura de investigación que apoyen la investigación sobre epidemias.

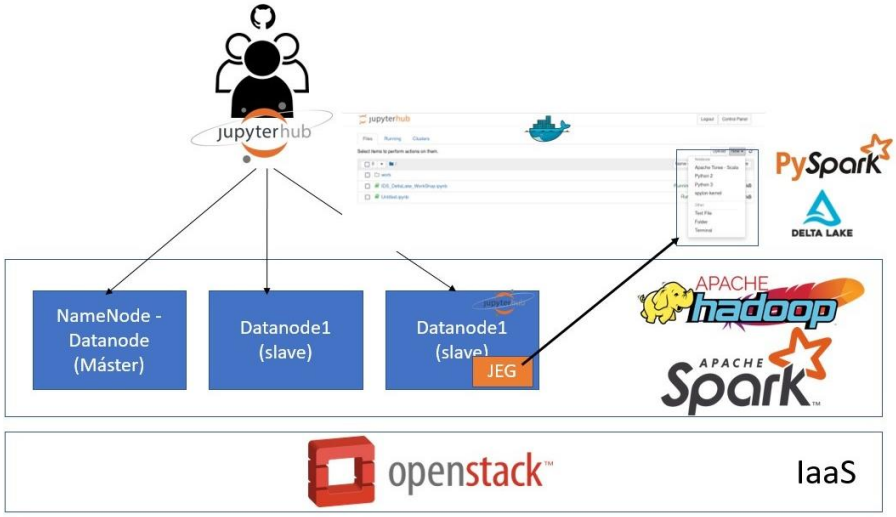


Figura 15. Plataforma de procesamiento de datos en el proyecto FACE

La plataforma desarrollada en dicho proyecto se utilizará en este trabajo para la realización del análisis de los datos sobre un clúster de PySpark. En la Figura 15 se muestran los componentes actuales de la plataforma, los cuales se han ido describiendo a lo largo del marco teórico. El clúster de PySpark consta de 3 nodos de Hadoop + Spark desplegados sobre máquinas virtuales

dentro del clúster de *Cloud Computing* basado en la plataforma *OpenStack*. En la Tabla 3 de abajo se detallan las características de estos nodos:

| Instance Name | Image | IP Address | Flavor | VCPU | RAM | Disk |
|---|---|---|---|---|---|---|
| **Namenode** | IFCA Ubuntu 20.04 | 172.16.64.18 | cm4.8xlarge | 32 | 58.6 GB | 30 GB |
| **slave1** | IFCA Ubuntu 20.04 | 172.16.64.7 | cm4.8xlarge | 32 | 58.6 GB | 30 GB |
| **slave2** | IFCA Ubuntu 20.04 | 172.16.64.1 | cm4.4xlarge | 16 | 29.3 GB | 30 GB |

Tabla 3. Descripción de los nodos sobre los que se ha desplegado el clúster de Hadoop + Spark

Para finalizar, se utiliza la plataforma *Jupyterhub* para el acceso interactivo de múltiples usuarios a la plataforma. El dataset de *tweets* del conflicto Ucrania-Rusia se ha copiado dentro del clúster de HDFS para hacerlo accesible a través de esta plataforma. Así mismo se asigna como recursos 2 cores.

A continuación, se explican en resumen los pasos realizados a la hora de adaptar el análisis descrito en la anterior sección al entorno Pyspark junto a HDFS.

### *4.4.2 Unión de datos con PySpark*

Previo a realizar cualquier operación, es necesario crear una sesión en Spark. La forma básica de crear una sesión es importando la librería necesaria, y luego definiendo donde se crea la sesión y el nombre de esta. Al crear una sesión es cuando se puden asignar recursos, por ejemplo, los 2 cores que usaremos. A continuación, se muestra un ejemplo:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local").
config("spark.driver.cores","2").appName("Nom
bre de la sesion").getOrCreate()
```

A continuación, el proceso de unión de todos los archivos pequeños en uno solo es el mismo que en Python con algunas diferencias en la forma de manejar los datos como ser que en vez de comenzar un un dataframe de pandas vacío, utilizamos el tipo de datos principal de Spark, los RDDs, los cuales son una colección de elementos particionados a través de los nodos del clúster y que pueden ser operados en paralelo.

Al momento de descargar el dataset de *tweets* para copiarlo dentro del cluster de HDFS, a diferencia de cuando se trabajó con Python, la cantidad de datos es mayor ya que la fecha de descarga es del 8 de Septiembre y por lo tanto tenemos los datos de 42 días más, por lo cual, la

cantidad de datos finales que tenemos es de 55.926.014 filas, es decir 10.932.692 filas más que en el dataset trabajado en Python.

A pesar de esta gran diferencia de datos que existe, PySpark emplea 436 segundos en unir los datos y 1812 segundos en transcribirlos a un archivo csv. Así mismo, el tiempo que emplea en importar ese archivo csv creado es de 0.32 segundos.

### *4.4.3 Pre procesamiento de datos con PySpark*

Se omite la exploración de los datos, ya que este paso ya se realizó con Python y las ejecuciones de las órdenes para conocer los datos no demoran más de unos pocos segundos, por lo cual no es necesario realizar una comparativa de tiempo de ejecución con PySpark. Sin embargo, si medimos el tiempo de ejecución de alguna de las modificaciones que realizamos sobre nuestro set de datos.

Ya sabemos que el idioma predominante es el inglés y también sabemos que la columna *coordinates* no nos aporta información alguna ya que en su mayoría está contenida por valores NaN. Por lo tanto, filtramos aquellos *tweets* que contengan el valor "en" en la columna *language* y procedemos a eliminar las columnas de *language* y *coordinates*. Esta operación toma 0,142 segundos.

### *4.4.4 Limpieza de texto con PySpark*

Procedemos a quedarnos con las filas únicas, ya que sabemos que existen filas duplicadas, y por lo tanto necesitamos deshacernos de estas. Seguidamente limpiaremos el texto como lo habíamos hecho antes, reemplazando las URLs por "URL", las menciones a otros usuarios por "USER", los saltos de línea (\n) por un espacio, el uso de "&" por "and", la repetición de una misma letra seguida tres veces o más por dos de esta misma y por último eliminamos el símbolo "#". El procedimiento viene a ser el mismo, sin embargo, se lo adecua a PySpark, ya que previamente utilizábamos funciones de pandas para hacer estos cambios. La sumatoria de tiempo de aplicar todos estos cambios es de 0.115 segundos.

### *4.4.5 Análisis de sentimientos con PySpark*

Para esta etapa utilizamos una de las funciones de PySpark, llamada udf (*user defined function*), la cual nos permite adaptar el método de TextBlob a la manera en que trabaja PySpark, definiéndolo dentro de una función, la cual luego le pasaremos como argumento a la función udf , creando así una nueva función que finalmente, aplicaremos sobre toda la columna de *text*.

A continuación, se enseña un trozo de código de cómo se aplica udf a la función apply_blob, la cual se encarga de recibir un texto, aplicarle TextBlob y mediante los umbrales definidos en la sección 4.3.1, retorna negativo, positivo o neutro. Seguidamente se aplica esta nueva función a la columna que contiene los *tweets*, creando un nuevo set de datos con una nueva columna llamada *sentiment*:

```
txtBlob = udf(apply_blob)

df_blob   =   df_clean.withColumn("sentiment",
txtBlob(df_clean['text']))
```

Para todos los pasos mencionados, la ejecución fue instantánea, durando tan solo 0,148 segundos. Dado que aplicamos el mismo método, se espera que los resultados sean similares a los que obtuvimos previamente, con la diferencia de que esta vez contamos con una mayor cantidad de datos. De modo que, obtenemos un total de 7.137.286 tweets negativos, 13.305.352 positivos y 10.900.472 neutros. Como se observa en la Figura 16, los porcentajes de cada uno de estos respecto al total, son similares a lo que obtuvimos anteriormente con Python.

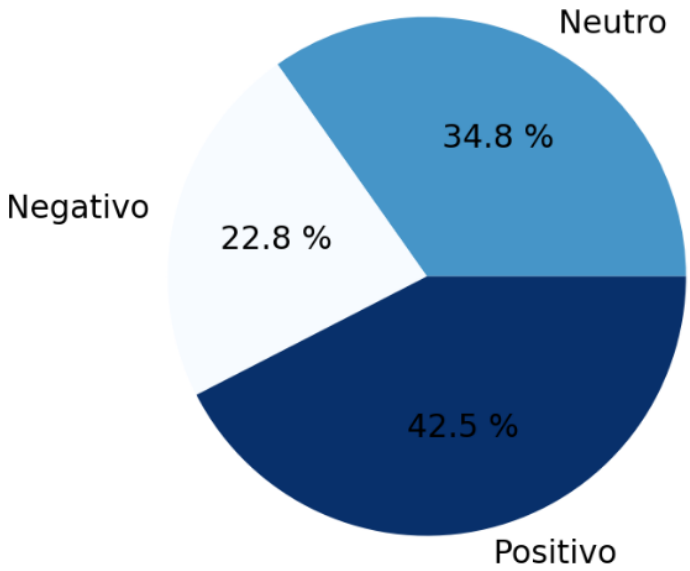


Figura 16. Gráfico de tortas de la clasificación de sentimientos con TextBlob en PySpark

### 4.5 Resultados

A lo largo de la sección 4.3 y 4.4, encontramos la medición de tiempo de distintos pasos para alcanzar la deseada clasificación de sentimientos, viendo que en definitiva PySpark reduce el tiempo empleado significativamente en casi todas las operaciones. En la Tabla 4 se realiza una comparativa de los tiempos medidos de la ejecución de las diferentes operaciones con Python y PySpark.

| Operación | Python [segundos] | PySpark [segundos] |
|---|---|---|
| Concatenación de archivos | 8042,75 | 436 |
| Escritura csv | 1553,55 | 1812 |
| Lectura de csv | 3022,1 | 0,32 |
| Eliminación de columna | 98,38 | 0,142 |
| Limpieza de texto | 1754,72 | 0,115 |
| Análisis de sentimientos | 42405,8 | 0,148 |
| Total tiempo invertido | 56876,55 | 2248,765 |

Tabla 4. Tiempos de ejecución de operaciones con Python y PySpark

Como se observa en la Tabla 4, destacan dos de las operaciones, primero la escritura de los datos en un archivo csv, donde no existe ningún tipo de disminución de tiempo, sino que se obtiene prácticamente lo mismo, tomando en cuenta que en PySpark analizamos un set de datos más grande. Esto puede deberse a varios factores, como ser que no se esté particionando la operación de escritura, o que, al contrario, se esté partiendo en grupos tan pequeños que finalmente sea casi como hacer la escritura de manera secuencial como en Python. En general, es común que esta operación demore ya que debe gestionar los recursos que posee (cores y memoria) y las conexiones entre estos mientras envía los datos, lo cual afecta negativamente en el tiempo de ejecución.

La otra operación que resalta es el análisis de sentimientos. El tiempo que demora PySpark en realizar algo que a Python le tomó casi medio día, es tan solo de una décima de segundo, siendo esto apenas el 0,00026% del tiempo que Python emplea para obtener una clasificación de sentimientos del texto de los *tweets*. Esto tiene gran importancia, ya que esta es la parte clave de cualquier análisis de datos ya que normalmente el uso de técnicas de *machine learning* suele tomar un tiempo largo relativo al tamaño del conjunto de datos que se tiene.

La Tabla 4 muestra al final la sumatoria total de tiempo de las operaciones que se midieron, las cuales eran las más significativas. A pesar de la demora que tenemos por la escritura del archivo, logramos pasar de casi 16 horas con Python a 38 minutos con PySpark, lo cual equivale a tan solo un 4%. Esto hace que, por ejemplo, una implementación en tiempo real de tweets agregados diariamente al conjunto de datos sea posible.

# 5. CONCLUSIONES Y TRABAJO FUTURO

## 5.1 Conclusiones

En base a los resultados obtenidos y a los objetivos definidos en el trabajo, las conclusiones que obtenemos son las siguientes:

La herramienta de computación distribuida PySpark reduce significativamente en un 96% el tiempo de ejecución de operaciones de procesamiento y análisis de volúmenes de datos del orden los GB, convirtiendo 16 horas de trabajo en 38 minutos, de los cuales 30 son invertidos en la escritura de datos. El tiempo de escritura con PySpark depende de que la partición de los datos sea la adecuada y se tenga los recursos necesarios, por lo cual, este tiempo puede ser reducido aún más si así se lo requiere.

Cuando se trabaja con NLP es necesario preprocesar el texto para que se asemeje lo mejor posible al lenguaje natural de los humanos, ya que los métodos de procesamiento natural de lenguaje son creados de modo que entienden mejor cuando el texto está correctamente escrito, y por lo tanto se obtienen mejores resultados. Así mismo, dependiendo del método de análisis de sentimientos que se utiliza, la clasificación final de sentimientos puede ser completamente distinta por lo cual es necesario analizar los resultados para ver cuál es el que mejor se adecua a las necesidades.

Este trabajo demuestra que, utilizando la herramienta adecuada, en este caso Spark, se puede disminuir significativamente el tiempo que se emplea para el análisis y procesado de grandes volúmenes de datos ya sea para NLP, machine learning o cualquier otro procedimiento que implique trabajar con otra volumetría de datos.

## 5.2 Trabajo futuro

El presente proyecto solo enseña los resultados obtenidos sobre un conjunto de datos fijo. Es posible crear una aplicación que analice los datos en tiempo real y emita resultados diariamente, de modo que se pueda dar a conocer información actualizada de manera continua. Por otro lado, para obtener un mejor resultado en el análisis de sentimientos se puede hacer una investigación más exhaustiva de los métodos disponibles, ya que existen muchos de estos, cada uno con diferentes características en cuanto al procedimiento que llevan y los resultados que retornan, de modo que se sepa cuáles son los pasos que se pueden agregar al preprocesado que adecuen mejor los datos al método seleccionado.

El tiempo de escritura de datos con PySpark puede ser reducido particionando los datos de una manera más óptima ya que en este trabajo se usaron solo 2 cores, omitiendo el aumento de estos ya que se había ya logrado el objetivo, que era reducir significativamente el tiempo inicial que obteníamos con Python a modo de demostrar la eficiencia del uso de la herramienta correcta. Así mismo, si el tiempo de procesado aumenta debido a una mayor cantidad de datos que los utilizados en el presente proyecto, se puede hacer pruebas modificando los recursos disponibles.

**BIBLIOGRAFÍA**


1. A. MULLEN, Lincoln, et al. Fast, consistent tokenization of natural language text. *Journal of Open Source Software*, 2018, vol. 3, no 23, p. 655.

2. ATTIYA, Hagit; WELCH, Jennifer. *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004, p. 1-2.

3. BRICEÑO, Bertha; FERNANDEZ, Eugenia. Aplicando el procesamiento del lenguaje natural para clasificar artículos del coronavirus. En: *BID: Banco Interamericano del Desarrollo* [en línea]. Disponible en: https://blogs.iadb.org/conocimiento-abierto/es/aplicando-el-procesamiento-del-lenguaje-natural-para-clasificar-articulos-del-coronavirus/ [consulta: 2 de Septiembre 2022].

4. CABRILLO, Ibán. *Herramientas en la nube para la ciencia de datos: Hadoop*. Instituto de Física de Cantabria, 2022.

5. CHAUDHRI, Abhishek Akshay; SARANYA, S. S.; DUBEY, Sparsh. Implementation paper on analyzing COVID-19 vaccines on twitter dataset using tweepy and text blob. *Annals of the Romanian Society for Cell Biology*, 2021, p. 8393-8396.

6. CHOWDHARY, KR1442. Natural language processing. *Fundamentals of artificial intelligence*, 2020, p. 603-649.

7. DEROOS, Dirk. *Hadoop for dummies*. John Wiley & Sons, 2014, p. 13, 53-54, 108-110.

8. EISENSTEIN, Jacob. *Introduction to natural language processing*. MIT press, 2019.

9. GARCÍA-GIL, Diego, et al. A comparison on scalability for batch big data processing on Apache Spark and Apache Flink. *Big Data Analytics*, 2017, vol. 2, no 1, p. 1-11.

10. GANESAN, Kavita. What are N-Grams?. En: *Kavita Ganesan* [en línea]. Disponible en: https://kavita-ganesan.com/what-are-n-grams/ [consulta: 6 de Septiembre 2022].

11. GARG, Vijay K. *Elements of distributed computing*. John Wiley & Sons, 2002, p.1.

12. GOYVAERTS, Jan. The Premier website about Regular Expressions. En: *Regular Expressions.info* [en línea]. Disponible en: https://www.regular-expressions.info/ [consulta: 2 de Septimbre 2022].

13. GREFENSTETTE, Gregory. Tokenization. En *Syntactic Wordclass Tagging*. Springer, Dordrecht, 1999. p. 117-133.

14. GUJJAR, J. Praveen; KUMAR, H. P. Sentiment analysis: Textblob for decision making. *Int. J. Sci. Res. Eng. Trends*, 2021, vol. 7, no 2, p. 1097-1099.

15. GUO, Runxin, et al. Bioinformatics applications on apache spark. *GigaScience*, 2018, vol. 7, no 8, p. giy098.

16. HAJIBABA, Majid; GORGIN, Saeid. A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing. *Journal of computing and information technology*, 2014, vol. 22, no 2, p. 69-84.

17. HAZARIKA, Ditiman, et al. Sentiment Analysis on Twitter by Using TextBlob for Natural Language Processing. *ICRMAT*, 2020, vol. 24, p. 63-67.

18. HIRSCHBERG, Julia; MANNING, Christopher D. Advances in natural language processing. *Science*, 2015, vol. 349, no 6245, p. 261-266.

19. HOLMES, Alex. *Hadoop in practice*. Simon and Schuster, 2014, p. 4-8, 22-23.

20. HOPE, Computer. Regex. En: *Computer Hope* [en línea]. Disponible en: https://www.computerhope.com/jargon/r/regex.html [consulta: 2 de Septimbre 2022].

21. KSHEMKALYANI, Ajay D.; SINGHAL, Mukesh. *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011, p 1-2.

22. LAM, Chuck. *Hadoop in action*. Simon and Schuster, 2010, p. 4-5, 38.

23. LIU, Yinhan, et al. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

24. LOPER, Edward; BIRD, Steven. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

25. MARTÍNEZ, Merche. Google BERT: actualización para entender el lenguaje natural. En: *Human Level* [en línea]. Disponible en: https://www.humanlevel.com/articulos/posicionamiento-natural-buscadores/google-bert-actualizacion-para-entender-el-lenguaje-natural.html [consulta: 2 de Septimbre 2022].

26. NADKARNI, Prakash M.; OHNO-MACHADO, Lucila; CHAPMAN, Wendy W. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 2011, vol. 18, no 5, p. 544-551.

27. NARAYANASWAMY, Gagan Reddy. *Exploiting BERT and RoBERTa to improve performance for aspect based sentiment analysis*. 2021. Tesis Doctoral. dissertation. Technological University Dublin, Dublin. https://doi. org/10. 21427/3w9n-we77.

28. NLTK. Documentation. En: *Natural Language Toolkit* [en línea]. Disponible en: https://www.nltk.org/ [consulta: 1 Septiembre 2022].

29. PERKINS, Jacob. *Python 3 text processing with NLTK 3 cookbook*. Packt Publishing Ltd, 2014.

30. RANGANATHAN, G. Real time anomaly detection techniques using pyspark frame work. *Journal of Artificial Intelligence*, 2020, vol. 2, no 01, p. 20-30.

31. SALLOUM, Salman, et al. Big data analytics on Apache Spark. *International Journal of Data Science and Analytics*, 2016, vol. 1, no 3, p. 145-164.

32. SHORO, Abdul Ghaffar; SOOMRO, Tariq Rahim. Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*, 2015.

33. SINGH, Vikash Kumar, et al. A literature review on Hadoop ecosystem and various techniques of big data optimization. *Advances in Data and Information Sciences*, 2018, p. 231-240.

34. SPARK, Apache. Apache spark. *Retrieved January*, 2018, vol. 17, no 1, p. 2018.

35. SPARK, Apache. Apache Spark history. En: *Apache Spark* [en línea]. Disponible en: https://spark.apache.org/history.html [consulta: 6 Julio 2022].

36. TURKINGTON, Garry. *Hadoop Beginner's Guide*. Packt Publishing Ltd, 2013, p. 15-17.

37. ULLAH, Rahmat; ARSLAN, Tughrul. PySpark-based optimization of microwave image reconstruction algorithm for head imaging big data on high-performance computing and Google cloud platform. *Applied Sciences*, 2020, vol. 10, no 10, p. 3382.

38. UZUNKAYA, Can; ENSARI, Tolga; KAVURUCU, Yusuf. Hadoop ecosystem and its analysis on tweets. *Procedia-Social and Behavioral Sciences*, 2015, vol. 195, p. 1890-1897.

39. V, Nithyashree. What Are n-grams and How to Implement Them in Python?. En: Analytics Vidhya [en línea]. Disponible en: https://www.analyticsvidhya.com/blog/2021/09/what-are-n-grams-and-how-to-implement-them-in-python/ [consulta: 6 de Septiembre 2022]

40. VENNER, Jason. *Pro hadoop*. Apress, 2009, p. 4-6.

41. VIJAYARANI, S., et al. Text mining: open source tokenization tools-an analysis. *Advanced Computational Intelligence: An International Journal (ACII)*, 2016, vol. 3, no 1, p. 37-47.

42. WANG, Kewen; KHAN, Mohammad Maifi Hasan. Performance prediction for apache spark platform. En *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, 2015. p. 166-173.

43. WHITE, Tom. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012, p. 11-12, 41.