FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

IMPLEMENTACIÓN DE UN FRAMEWORK DE BIG DATA PARA EL ANÁLISIS DE SENTIMIENTOS EN REDES SOCIALES POR MEDIO DE APACHE SPARK

AUTOR

JOSÉ LUIS GUAMBO HEREDIA

AÑO

2020

FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

IMPLEMENTACIÓN DE UN FRAMEWORK DE BIG DATA PARA EL ANÁLISIS DE SENTIMIENTOS EN REDES SOCIALES POR MEDIO DE APACHE SPARK.

Trabajo de Titulación presentado en conformidad a los requisitos establecidos para optar por el título de Ingeniero en Electrónica y Redes de Información.

Profesor Guía

Dr. William Eduardo Villegas Chiliquinga

Autor

José Luis Guambo Heredia

Año

2020

DECLARACIÓN PROFESOR GUÍA

"Declaro haber dirigido el trabajo, Implementación de un framework de big data para el análisis de sentimientos en redes sociales por medio de Apache Spark, a través de reuniones periódicas con el estudiante José Luis Guambo Heredia, en el semestre 202020, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación"

William Eduardo Villegas Chiliquinga
Doctor en Informática
C.I: 1715338263

DECLARACIÓN PROFESOR CORRECTOR

“Declaro haber revisado este trabajo, Implementación de un framework de big data para el análisis de sentimientos en redes sociales por medio de Apache Spark, de José Luis Guambo Heredia en el semestre 202020, dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación”

Iván Patricio Ortiz Garcés
Magíster en Redes de Comunicaciones
C.I: 0602356776

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE.

“Declaro que este trabajo es original, de mi autoría, que se ha citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes.”

José Luis Guambo Heredia

CI: 0603720848

AGRADECIMIENTOS

A la Universidad de las Américas por darme una segunda oportunidad y permitirme estudiar lo que amo. Llegue con pocas expectativas y me voy realmente sorprendido de la alta calidad de los docentes.

DEDICATORIA

A mi padre José por su entrega de confianza total, apoyo incondicional y amor infinito. A mi hermana Paola que es mi ejemplo para seguir. A mi familia por llenarme de felicidad a cada segundo.

RESUMEN

Las redes sociales son una herramienta para conocer las opiniones y preferencias del público, y la información contenida en las publicaciones de los usuarios es invaluable. Un framework de análisis de sentimiento de redes sociales es implementado a través de Apache Spark, para conocer la apreciación de los usuarios de estas redes respecto a determinada marca, entidad o temática.

Se diseña la arquitectura general del framework, analizando los requerimientos a través de historias de usuarios y generando tres diseños: alto nivel, bajo nivel e interfaz. En el diseño de alto nivel se produce un diagrama de actividades que define el flujo de trabajo del framework. El diseño de bajo nivel genera un diagrama de actividades con mayor nivel de detalle que el primero. El diseño de interfaz es una propuesta inicial de la interfaz gráfica para el usuario final.

El proceso de implementación se documenta de manera detallada. Se analizan los problemas de librerías existentes y las alternativas tomadas para sortear los inconvenientes. Se incluye la programación realizada y la modificación a librerías de terceros. Se desarrollan pruebas unitarias en paralelo a la implementación.

El comportamiento del framework es evaluado a través de una prueba de integración, se analizan los resultados de dichas evaluaciones y se plantean modificaciones para la implementación de futuras versiones.

ABSTRACT

Social networks are tools to know opinions and preferences of the public, information contained in posts from users is invaluable. A framework for social networks sentiment analysis is implemented through Apache Spark, to know user's appreciation according to brand, entity, or topic.

The overall architecture of the framework is designed, analyzing requirements through user stories, and generating three designs: high level, low level, and interface. In high level design an activities diagram is generated, this design defines the framework's workflow. Low level design generates an activities diagram with a greater level of detail than the previous one. Interface design is an initial proposal of the graphic interface for the final user.

Process of implementation is documented in detail. Problems of libraries used are analyzed and alternatives taken to get around those drawbacks are implemented. Programming done and modifications made to third party libraries are included. Unit tests are developed in parallel to the implementation.

Behavior of the framework is evaluated through an integration test, results from those evaluations are analyzed and modifications for future releases are proposed.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En la actualidad, para las redes sociales los usuarios activos en Ecuador representan el 71% de la población (Kemp, 2019). Los usuarios por medio de la interacción con la plataforma y con otros usuarios, son capaces de compartir de manera explícita e implícita ideas, sentimientos y opiniones.

Los usuarios de las redes sociales interactúan con la aplicación y con otros usuarios, generando grandes cantidades de información y de naturaleza variada, por ejemplo, con categorización variada como mensajes de texto, estados y Tweets (Schneider & Karmiol, 2019).

Varios estudios han encontrado que la naturaleza de la información disponible en internet corresponde a un 95% de información no estructurada (fotos, videos y audio) o semiestructurada (archivos XML y JSON). Solo el 5% de esta información es de tipo estructurada y está contenida en bases de datos tradicionales, como es la información transaccional y relacional (Gandomi & Haider, 2015).

Una de las limitantes del análisis de datos es el volumen y la velocidad a la que se produce y procesa la información. Los métodos tradicionales de procesamientos no pueden ser usados para tratar con este tipo de información y se deben buscar alternativas.

Entre las técnicas que sobresalen se observan algoritmos de procesamiento y uso de cómputo distribuido. Esta metodología de procesamiento de la información usa el principio denominado divide y conquista, en la que tareas sumamente grandes y complejas que los sistemas no distribuidos ejecutarían de forma secuencial, se dividen en pequeñas tareas y son ejecutadas en forma paralela en varios nodos del arreglo (Schneider & Karmiol, 2019).

El análisis de datos tiene varias aplicaciones, y algunos casos de uso con enfoque al área de TI, aplicaciones médicas y enfoques de marketing. Esta última área atrae a gran cantidad de inversionistas debido al alto rédito

económico que genera, mediante la segmentación de clientes, priorización de ventas y análisis del sentimiento (Mendelevitch et al., 2017).

El acceso a la información contenida en redes sociales se facilita gracias al uso de varias API, sin embargo, entre los desafíos actuales se encuentra la restricción del acceso al contenido mencionado con el fin de monetizar su contenido (Batrinca & Treleaven, 2014).

Este accionar de las empresas encargadas de administrar dichas plataformas, ha buscado que los usuarios busquen alternativas al acceso a la información por medio de las API, dado que al ser parte de la red social y generar contenido, asumen una falsa sensación de propiedad sobre el mismo.

La plataforma Buffer dedicada al negocio de análisis de datos pone a disposición del público 27 plataformas como Buffer Analyze, Sprout social, Hootsuite, Zoho, entre otras, además de paquetes de análisis propios de las redes sociales como Facebook e Instagram Insights y Twitter, Pinterest, Linkedin y YouTube Analytics.

Sin embargo, dichas herramientas tienen una limitante que son herramientas comerciales que requieren de una subscripción mensual o versiones gratuitas con funcionalidades limitadas (Lee, 2019). De igual forma herramientas sin costo como biBIGDATA presentan la limitante de analizar una sola red social como es el caso de Twitter (*Big Data - Demo - Stratebi*, s/f).

1.1. Objetivo General

Implementar un framework de big data para análisis de sentimiento en redes sociales con el uso de Apache Spark.

1.2. Objetivos específicos

1. Investigar los conceptos relacionados con análisis de datos y la documentación del framework Apache Spark.

2. Implementar el framework Apache Spark para el análisis de sentimientos de redes sociales.
3. Analizar los resultados del análisis de sentimientos de acuerdo con las búsquedas realizadas en las diferentes redes sociales

1.3. Alcance

En el presente anteproyecto se implementará una plataforma para análisis de sentimientos de redes sociales por medio del framework de software libre Apache Spark.

Se investigarán los principales conceptos relacionados con análisis de datos, Apache Spark y análisis de sentimiento. Se diseñará la plataforma siguiendo recomendaciones de los desarrolladores, para usar de manera efectiva los recursos de hardware disponibles y el acceso a los datos de las redes sociales.

Se implementará la plataforma y se analizarán los resultados de la implementación en base a parámetros de comportamiento funcional.

1.4. Justificación

El proyecto representa una opción viable para pequeñas y medianas empresas relacionadas a marketing y ventas. Mediante el análisis de datos de redes sociales, las empresas son capaces de obtener información invaluable que les permita segmentar a distintos grupos de clientes, para utilizar técnicas de marketing personalizadas.

La implementación de esta plataforma implica reducción en los costos adquisitivos, al comparar la capacidad de cómputo de clústers de equipos de bajas prestaciones con servidores. El abaratamiento de costos también viene de la mano del licenciamiento por medio del uso software gratuito y software libre, frente al licenciamiento por uso de software propietario en negocios tradicionales.

El uso de software libre para la implementación garantiza la libertad total de uso del framework de análisis, es decir que las empresas que decidan implementar este tipo de solución podrán utilizar el sistema distribuido con cualquier finalidad y para efectuar cualquier tipo de actividad, sin restricción.

2. MARCO TEÓRICO

Este capítulo esta designado para investigar los conceptos de big data, redes sociales y Apache Spark. Se describen las características de la información considerada como big data. En la sección de redes sociales se resumen las principales plataformas y los métodos de acceso a la información.

Se investiga sobre la plataforma Apache Spark, sus elementos, operaciones y comandos básicos. Los principales algoritmos de clasificación son investigados en la sección de aprendizaje de máquina, además de una herramienta de visualización de datos compatible con Spark.

2.1. Big data

La definición de big data no ha llegado a un consenso general, para ciertos autores es una colección de información demasiado grande y compleja para ser almacenada y procesada con tecnologías tradicionales (Schneider & Karmiol, 2019). La mayoría de los trabajos concuerda en tres características que la información debe poseer para ser considerada big data, conocidas también como las tres uves: volumen, variedad y velocidad.

El volumen de la data se refiere a la cantidad de información, que debido a la explosión de dispositivos conectados e información de sensores alcanzará un estimado de 40 trillones de gigabytes para 2020 (Mendelevitch et al., 2017). La variedad de la información trata el formato de la información, se tiene tres tipos: estructurada, semiestructurada y no estructurada.

La información estructurada corresponde a datos almacenada en bases de datos tradicionales, con entidades previamente definidas y que se relacionan entre sí,

por ejemplo, entidades que definen una persona, un libro o un producto en bases de datos SQL.

Los archivos JSON y XML son ejemplos de información semiestructurada, en la que de forma general se tienen una o varias etiquetas a las que les corresponden uno o varios valores, y adquieren significado en base al tratamiento del usuario. Finalmente se tiene información no estructurada que corresponde a los datos de audio, imágenes y video (Schneider & Karmiol, 2019).

El tratamiento y control de Big Data permite que las empresas exploren varios casos de uso en industrias como: entidades financieras, telecomunicaciones, ventas, salud. En las áreas asociadas a ventas y bancos, el manejo de Big Data permite el análisis del comportamiento del usuario en base a las transacciones que este realiza, con la finalidad de predecir el futuro comportamiento en la adquisición de productos y servicios.

En las telecomunicaciones se analizan los archivos de log generados por equipos y sensores que generan alarmas y permiten determinar el uso medio de recursos, luego en base a dicho utilización se busca optimizar las configuraciones de los dispositivos para mejorar la eficiencia energética.

El área de la salud se ve beneficiada del análisis de las historias clínicas de miles de pacientes con sintomatología similar para determinar las posibles causas de las enfermedades y el tratamiento exitoso de estas.

2.2. Redes sociales

Las redes sociales son plataformas creadas para que los usuarios compartan información: imágenes, audio, videos y texto con otros usuarios, permitiéndoles expresar sus sentimientos y pensamientos. Esta información permite que los usuarios interactúen entre sí, simulando interacción social física, pero a través de medios virtuales.

Existen un sin número de redes sociales: Facebook, Twitter, YouTube, Instagram, LinkedIn, WhatsApp, entre otras. Se analizan cuatro redes sociales

en particular: Facebook debido a que es una de las plataformas con mayor número de usuarios a nivel mundial, Twitter por la facilidad que brinda a desarrolladores de software para la extracción de datos, YouTube por ser la red social líder en compartición de videos e Instagram por ser una de las redes más representativas para la compartición de fotografías.

2.2.1. Twitter

Una forma de describir a Twitter es como un servicio de microblogging que permite a las personas comunicarse con mensajes cortos que corresponden a pensamientos o ideas (Russell & Klassen, 2019).

Un acercamiento más a profundidad de esta plataforma permite definir ciertos elementos clave: usuarios, tweets y líneas de tiempo. Los usuarios se refieren a personas, empresas e incluso entidades ficticias que hacen uso de la plataforma. Un tweet es una cadena de caracteres que se asocia al estado de un usuario, contiene dos datos que lo definen denominados entidades y lugares.

Las entidades de un Tweet son menciones a otros usuarios, hashtags, direcciones URL y cualesquiera otros tipos de media asociada al Tweet, mientras los lugares representan localizaciones físicas del planeta. El modelo de relaciones de esta plataforma es de carácter asimétrico, es decir una decisión unilateral entre un usuario que desea seguir a otro (Russell & Klassen, 2019).

2.2.1.1. API de Twitter

La API de Twitter permite acceder a la información contenida en la plataforma. Existen 6 entregas de la API que varían en función a la necesidad del usuario final: Standard, Premium, Enterprise, Ads, para sitios web y Developer Labs.

Standard: Permite publicar, recibir y participar con los tweets y las líneas de tiempo, mensajes directos, administrar y obtener información pública de una cuenta, crear y manejar listas, seguir, buscar y obtener usuarios.

Premium: Una funcionalidad de filtrado avanzado por medio de la API Search Tweets, información en tiempo real de cuentas públicas y aceptación.

Enterprise: Acceso a Tweets e información de cuentas públicas en tiempo real, Tweets históricos y percepción de los Tweets.

Ads: Integrada con la plataforma de anuncios de Twitter.

Twitter para sitios web: Agrega contenido de Twitter a un sitio web.

Developer Labs: Terminales experimentales usados para desarrollo.

Las API listadas requieren distinto nivel de acceso y de tipo de autenticación. La versión estándar es la API sin costo y tiene dos requerimientos: una cuenta de desarrollador y utilizar protocolo de autenticación OAuth 1.0 (*Twitter Inc.*, s/f).

Este protocolo provee un método para que clientes accedan a los recursos de un servidor en nombre de otro usuario o cliente (dueño del recurso), adicionalmente el dueño del recurso otorga el acceso sin compartir sus credenciales (RFC 5849 - The OAuth 1.0 Protocol, s/f).

La API Standard tiene un límite respecto a las consultas o recursos, cada límite de tasa se divide en intervalos de 15 minutos y todos los terminales de usuario que se conectan deben ser autenticados (no existen llamadas anónimas). Twitter tiene dos paquetes disponibles para peticiones GET, el primero es de 15 llamadas cada 15 minutos, y el segundo de 180 llamadas cada 15 minutos.

En el caso que una aplicación exceda el límite de tasa establecido para realizar las consultas, será añadida a la lista negra y no obtendrá respuestas de la API de Twitter.

2.2.2. Facebook

Facebook es una de las plataformas más populares en el mundo. Se puede pensar en esta red como un todo en uno que incluye: actualización de estados,

publicación de fotos y videos, mensajería instantánea, localización física, juegos, compra y venta, entre otras (Russell & Klassen, 2019).

El método de acceso a la aplicación es la API de Facebook, que en versiones recientes para garantizar la privacidad de los usuarios no permite extraer la información de actualizaciones de estado o intereses.

Una función que se mantiene y resulta interesante para la plataforma a desarrollar es medir la aceptación del usuario en páginas públicas, especialmente creado por compañías y celebridades. Esta plataforma lleva un modelo de relación simétrico, es decir se genera una relación entre dos usuarios cuando ambas partes han aceptado.

2.2.2.1. API de Facebook

El acceso a la información de la plataforma Facebook se realiza por medio de Graph API. Esta API es el medio principal para ingresar y extraer información de Facebook, se basa en HTTP y permite realizar tareas como: consultar información, publicar nuevas historias, gestionar anuncios, subir fotos, entre otras. (Facebook for Developers, s/f)

Graph API representa la información de Facebook por medio de un grafo social, tiene tres elementos fundamentales: nodos, aristas y campos.

Nodos: Son objetos individuales que representan un usuario, foto, página o comentario.

Aristas: La conexión entre un objeto y varios objetos, por ejemplo, un usuario y las fotos relacionadas al usuario.

Campos: Información que describe el objeto, como ejemplo se tiene el nombre de un usuario específico o su lugar de trabajo.

Los límites de acceso a los recursos para esta aplicación se calculan por las llamadas realizadas durante una hora. Se pueden hacer dos distinciones,

dependiendo si la consulta se realiza como un usuario o una aplicación. Para las aplicaciones el límite de tasa es de 200 por el número de usuarios, mientras que si se accede a Graph API como un usuario, Facebook mantiene confidencial el límite de llamadas que puede realizar el usuario (Facebook for Developers, s/f).

2.2.3. YouTube

YouTube es una plataforma de entretenimiento que no se limita a la compartición de multimedia (audio y video), pero que también permite la interacción entre usuarios a través de comentarios. También se conoce como una mezcla entre tres industrias: televisiva, musical y fílmica, en la cual los usuarios pueden subir y visualizar videos, empoderando a los usuarios para la creación y desarrollo de contenido (Khan, 2017).

2.2.3.1. API de YouTube

La API de YouTube (versión 3 al momento de la escritura) es una interfaz que permite a los desarrolladores acceder a los recursos de la plataforma y realizar operaciones sobre dichos recursos.

Existen cuatro operaciones: listar, insertar, actualizar y eliminar recursos, y en cuanto a los recursos disponibles existen dieciocho recursos en total, pero los relevantes incluyen: actividad, canal, comentario, video, lista de reproducción y resultado de búsqueda (Google Developers, s/f).

Similar a como se manejan las otras redes sociales, existe una cuota máxima permitida para consultas a la API, que varía de forma dinámica respecto a la aplicación desarrollada. Esta limitación permite que la red social garantice la calidad del servicio y acceso adecuado para todos los usuarios.

Las operaciones consumen unidades de la API, por defecto una aplicación puede usar hasta diez mil unidades por día, y en general una operación de lectura consume 1 unidad, una operación de escritura 50 unidades y subir un video 1600 unidades (valores aproximados dados por el fabricante).

2.2.4. Instagram

Instagram es una red social para que los usuarios compartan fotos y videos dentro de la plataforma, y así otras plataformas. La idea principal de dicha red es el uso de una etiqueta (hashtag) por medio de la cual los usuarios pueden encontrar videos y fotografías (Sheldon & Bryant, 2016).

En un inicio esta red social estaba disponible únicamente para dispositivos móviles, pero se ha expandido y hoy en día permite el acceso desde cualquier dispositivo o navegador.

2.2.4.1. API de Instagram

La API de la red social Instagram se denomina Graph API, similar a la interfaz de Facebook debido a que esta red social es propiedad de Facebook. La API se puede utilizar para compartir contenido multimedia (fotos, videos), responder dicho contenido a través de comentarios, buscar contenido en base a etiquetas, entre otros (Instagram, s/f).

Para poder utilizar la API oficial de Instagram, es necesario registrar la aplicación y completar un proceso de revisión, en la que Instagram se asegura que el software siga los lineamientos de la red social. Las principales tareas incluyen: administración, creación de contenido, moderación, anuncios y análisis.

2.3. Scraping

Las API son herramientas para la recolección de información, proveen un mecanismo para compartir información de manera conveniente, y con un formato en particular. Pero existen dos escenarios en particular que muestran las limitantes de una API, cuando el recurso al que se desea acceder no cuenta con una API e incluso cuando autor si provee una API.

En el primer caso, los autores de la información no creyeron pertinente la construcción de una API debido a que la información que proveen es mínima o de poca importancia, los autores no cuentan con la infraestructura para

implementar y mantener una API o la información es valiosa (o está protegida) y no debe ser compartida.

En el segundo caso, en el que si existe una API para la recolección de información, existen tres inconvenientes: límites del volumen y frecuencia de transmisión de la información impuestos por los fabricantes, tipo de información (audio, video, texto, entre otros) y el formato disponible de dicha información (Mitchell, 2018).

En estos casos particulares, es conveniente la utilización de scrapers web. Un scraper web es una herramienta para la recolección y procesamiento de grandes cantidades de información de manera rápida, y en general estas herramientas pueden acceder a lugares que los buscadores tradicionales (Firefox, Chrome, entre otros) no pueden.

Ciertos autores definen al scraping, en particular el scraping web como una técnica que ayudan a transformar información HTML no estructurada en información estructurada, que puede ser almacenada en bases de datos u hojas de cálculo (Wu, 2019).

2.3.1. Scraper de Twitter

El scraper utilizado para la red social Twitter, es twitter-scraper v0.4.2, programado en Python, para una versión mínimo de Python v3.6, extremadamente rápido y sin restricciones de la API de Twitter (Isguzar, 2020) .

Se distribuye bajo licencia MIT, un tipo de licencia poco restrictiva que permite el uso, copia, modificación, publicación del software (entre otras características). Esta herramienta permite obtener los tweets de un perfil determinado o los tweets relacionados a un hashtag.

2.3.2. Scraper de Facebook

El scraper utilizado para la red social Facebook, es facebook-scraper v0.1.12, programado en Python, para una versión mínima de Python v3.6, permite

scraping de páginas públicas de Facebook sin necesidad de una llave de API (Zuñiga, 2020).

Se distribuye bajo licencia MIT, y recibe como primer parámetro el nombre de la página. Entre los parámetros opcionales de la herramienta se encuentran: ID de grupo, el número de páginas que se puede solicitar, un temporizador para especificar el tiempo entre cada consulta, entre otros.

2.3.3. Scraper de Instagram

El scraper utilizado para la red social Instagram es InstaTouch v2.2.7, programado en lenguaje JavaScript, útil para extraer publicaciones e información de usuarios de Instagram, sin la necesidad de una llave de API (Nord, s/f). La herramienta se distribuye bajo licencia MIT, permite scraping de publicaciones en base al nombre de usuario, hashtag o ubicación.

De manera adicional permite scraping de comentarios de publicaciones específicas de Instagram y devuelve un archivo de tipo JSON o CSV, según lo especifique el usuario. Se caracteriza porque no tiene limitaciones en cuanto a la tasa de consultas, e incluso permite conocer los usuarios que le dieron me gusta a determinada publicación.

2.3.4. Scraper de YouTube

Para el scraping de la red social YouTube, se utilizan dos herramientas. La descarga de comentarios es posible por medio de la herramienta youtube-comment-downloader, un script para descarga de comentarios de YouTube sin utilizar la API de YouTube, escrito en lenguaje Python y que requiere una versión mínima de Python v.2.7 (Bouman, s/f). El script original se encuentra disponible en el Anexo 1.

El script se distribuye bajo licencia MIT, recibe como parámetro el ID del video de YouTube y devuelve un archivo tipo JSON con los comentarios descargados.

La segunda herramienta es Selenium, una herramienta de automatización que permite emular la interacción de un usuarios con un navegador, y su componente principal es WebDriver, una interfaz que permite definir conjuntos de instrucciones en los principales navegadores del mercado (Software Freedom Conservancy, s/f). Se encuentra disponible en varios lenguajes de programación (Java, Python, C#, Kotlin, entre otros).

2.4. Apache Spark

Apache Spark es una plataforma unificada para escribir aplicaciones de big data, está diseñado para soportar una amplia variedad de tareas de análisis de datos desde carga de información y consultas SQL hasta aprendizaje de máquina. (Chambers & Zaharia, 2018).

Esta plataforma se caracteriza por proveer una interfaz amigable y simplificada para los usuarios finales a través de un conjunto de API, soporta lenguajes de programación ampliamente usados como Python, Java, Scala y R, y librerías que simplifican las tareas relacionadas a big data. En la Figura 1 se puede observar el diseño de alto nivel del framework Apache Spark.

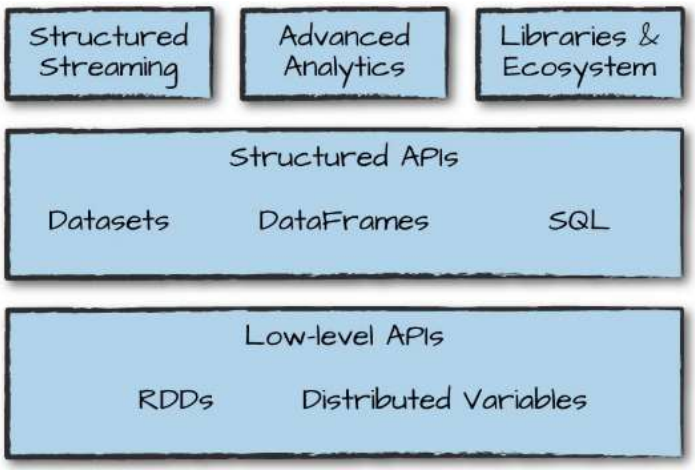


Figura 1. Diseño de alto nivel de los componentes y librerías de Spark.

Tomado de Chambers & Zaharia, 2018.

2.4.1. Librerías

Las librerías de la plataforma Spark son uno de los componentes más destacables y que han permitido que el proyecto sea desarrollado con tanta celeridad. Spark soporta librerías estándar que se distribuyen juntamente con el motor de computación Spark, y un arreglo de librerías externas desarrolladas por terceros.

Entre las principales librerías se tiene Spark SQL para SQL e información estructurada, MLlib para aprendizaje de máquina, GraphX para análisis de grafos (Chambers & Zaharia, 2018).

2.4.2. Términos de Apache Spark

SparkSession: Es el proceso encargado de controlar la aplicación Spark, una instancia de SparkSession permite que Spark ejecute las acciones definidas por el usuario.

DataFrames: Es una API estructurada que representa una tabla de datos con filas y columnas.

Schema: Una lista que define las columnas y los tipos de datos de las columnas.

Partition: Colección de filas almacenadas en una máquina física del clúster. Spark reparte la información en pedazos, y cada pedazo se denomina partición.

Transformación: Las estructuras de datos (DataFrames, Dataset, RDD) en Spark son inmutables, esto significa que una vez creadas no se pueden cambiar. Una transformación es un conjunto de instrucciones que pueden modificar la información, y esta modificación se lleva a cabo solo cuando el usuario llama a una acción.

Existen dos tipos de transformaciones: estrechas y amplias, donde para una transformación estrecha, una partición de entrada contribuye a una partición de salida, mientras para una transformación amplia una partición de entrada

contribuye a varias particiones de salida. Las transformaciones estrechas se realizan en memoria, y las amplias se escriben en disco.

Evaluación perezosa: Spark no ejecuta las operaciones definidas por el usuario de manera inmediata, en contraste espera hasta el último momento mientras va construyendo un plan de transformaciones para modificar la información. Este plan de transformaciones es optimizado por Spark para utilizar de la manera más eficiente los recursos computacionales.

Acciones: Una acción instruye a Spark para que inicie el cómputo de las transformaciones definidas en el plan de transformaciones. Existen tres tipos de acciones: acciones para ver información en la consola, acciones para recolectar información (el resultado es un objeto de un lenguaje de programación específico) y acciones para escribir en medios de salidas de datos.

Spark UI: Interfaz web para monitorear el progreso de un trabajo de Spark, disponible en el puerto 4040 del nodo.

Dataset: API estructurada de tipeo seguro disponible para dos lenguajes de programación: Java y Scala. Para ambos lenguajes de programación existe una clase Dataset que se parametriza con el objecto correspondiente. La característica de tipeo seguro evita que el usuario vea los objetos de un Dataset como una clase diferente a la ingresada inicialmente.

Streaming estructurado: API de alto nivel que permite extraer valores de sistemas de streaming de manera rápida. Permite escribir un trabajo en lotes y con cambios mínimos en el código convertirlo a un trabajo de streaming.

API de bajo nivel: API que permiten manipulación de objetos Java y Python, revelan las características de ejecución física (por ejemplo, particiones) a los usuarios. Son útiles para trabajar con datos crudos de información sin procesar y datos no estructurados, además cuando se busca ejecución distribuida con eficiencia extrema.

Resilent Distributed Dataset (RDD): Colección inmutable y particionada de registros que se puede trabajar en forma paralela. En un RDD cada registro es un objeto JAVA, Scala o Python, mientras que en un DataFrame cada registro es una fila estructurada que contiene campos de un esquema conocido.

2.4.3. API estructuradas

Las API estructuradas de Spark son herramientas para manipular todo tipo de información: desde archivos de log no estructurados, archivos CSV semiestructurados y archivos altamente estructurados como por ejemplo Parquet.

En Spark existen tres tipos de API fundamentales: Datasets, Dataframes y tablas o vistas SQL. Una API estructurada en Spark es la abstracción principal para escribir un flujo de trabajo. Un flujo de trabajo consiste en cuatro pasos: primero se escribe el código (Dataset, Dataframe o SQL), en segundo lugar, Spark lo convierte en un plan lógico, en tercero Spark transforma el plan lógico a un plan físico y el último paso es ejecutar el plan físico.

2.4.4. Fuentes de datos

Spark permite el trabajo con fuentes de información variadas, existen seis fuentes de datos fundamentales y cientos de fuentes creadas por terceros. Las fuentes fundamentales son: CSV, JSON, Parquet, ORC, conexiones JDBC/ODBC y archivos de texto plano.

CSV: Comma-Separated Value, es un archivo cuyos valores se separan por comas, cada línea representa un registro y cada coma separa un campo del registro.

JSON: JavaScript Object Notation, para el trabajo con Spark se recomienda el uso de JSON delimitados por línea. Este tipo de archivos tiene la particularidad de ser de fácil lectura para las personas.

Parquet: Archivo que almacena la información en columnas, y cuenta con optimizaciones de almacenamiento respecto a CSV y JSON. Tiene compresión por columna, que permite leer columnas individuales en lugar de archivos enteros. Permite almacenar tipos complejos, por ejemplo, un arreglo dentro de una columna.

ORC: Similar al archivo Parquet, optimizado para cargas de trabajo Hadoop.

Bases de datos SQL: Sistemas de almacenamiento de información que entienden y hablan SQL.

Archivos de texto: Archivo de texto plano en el cual cada línea del archivo representa un registro en el DataFrame.

2.4.4.1. Lectura de datos

Se realiza por medio de un lector de DataFrame, accesible desde el atributo de lectura de una sesión SparkSession. Al lector de DataFrame se le especifican tres valores que son el formato, esquema y modo de lectura. Adicionalmente tiene valores opcionales como la ruta de archivo, la opción de inferir el esquema automáticamente, entre otras.

2.4.4.2. Modos de Lectura

El modo de lectura especifica el accionar de Spark cuando se encuentre frente a registros o información mal estructurada, especialmente con fuentes de datos semiestructuradas. Existen tres modos de lectura: permissive, dropMalformed y failFast.

Permissive: Es el modo por defecto. Cuando Spark encuentra un registro corrupto, establece todos los campos en nulo y aumenta una columna de tipo string denominada called_corrupt_record.

DropMalformed: Elimina la fila con el registro corrupto.

failFast: Spark falla al encontrar registros corruptos.

2.4.4.3. Escritura de datos

Se realiza por medio de un escritor de DataFrame, accesible desde el atributo de escritura de una sesión SparkSession. Se deben especificar dos valores principales que son el formato y el modo de guardado. Adicionalmente se pueden especificar valores opcionales como ruta del archivo, formato de fecha, entre otros.

2.4.4.4. Modos de guardado

El modo de guardado especifica la acción a tomar por Spark en el caso de encontrar información en la ubicación que se quiere escribir.

Append: Agrega los archivos de salida a la lista de archivos que existen en la ubicación.

Overwrite: Sobreescribe la información existente.

ErrorIfExists: Es el valor por defecto. Arroja un error en caso de que exista información en la ubicación a escribir y Spark falla.

Ignore: No se modifica el DataFrame actual en caso de existir información en la ubicación especificada.

2.5. Aprendizaje de máquina

La inteligencia artificial es la habilidad de una máquina para imitar el comportamiento humano inteligente y el aprendizaje de máquina es una ramificación de la inteligencia artificial que permite que un sistema aprenda o sea entrenado en base a datos, antes que por medio de programación explícita. (Schneider & Karmiol, 2019)

Un modelo de aprendizaje de máquina es entrenado en base a grandes cantidades de datos con el comportamiento deseado o para la tarea especificada. Mientras mayor sea el volumen de información, mayor será la precisión del modelo.

2.5.1. Aprendizaje supervisado

El aprendizaje supervisado es un tipo de aprendizaje de máquina que permite entrenar un modelo en base a datos históricos previamente etiquetados con la finalidad de predecir los valores de dichas etiquetas para nuevos datos. (Schneider & Karmiol, 2019)

2.5.1.1. Clasificación

Este tipo de aprendizaje de máquina supervisado entrena el modelo para predecir un valor que representa una categoría. Se caracteriza porque la salida pertenece a un conjunto de categorías finitas (Schneider & Karmiol, 2019). Existen principalmente tres tipos de clasificación: binaria, multiclase y multietiqueta.

La clasificación binaria predice si el dato pertenece a uno de dos grupos, la clasificación multiclase indica si el dato pertenece a uno de más de dos grupos y la clasificación multietiqueta tiene la característica de que un solo dato de entrada puede producir varias etiquetas de salida.

2.5.1.2. Modelos de clasificación en MLlib

La librería de aprendizaje máquina MLlib para Spark provee cuatro modelos para clasificación: regresión logística, árbol de decisiones, bosques aleatorios y árboles de gradiente potenciado.

Regresión logística: Es un modelo linear que combina entradas individuales con pesos específicos generados durante el entrenamiento. Provee un resumen de modelo con la información sobre el modelo entrenado.

Árbol de decisiones: Este modelo clasifica en base a decisiones simples, donde un árbol se entrena con todos los datos, es amigable y de fácil interpretación. Soporta clasificación multiclase y genera resultados en dos columnas distintas, una para la predicción y otra para la probabilidad.

Bosques aleatorios: Es una extensión del árbol de decisiones, en el cual en lugar de entrenar un solo árbol con todos los datos se entrenan varios árboles con la información repartida en varios grupos. La predicción se realiza al promediar los resultados de todos los árboles.

Árboles de gradiente potenciado: Es una extensión del árbol de decisiones, en el que todos los datos se reparten en varios grupos y se entrenan varios árboles. Se diferencia de los bosques aleatorios, cada árbol hace una predicción con peso, la predicción de ciertos árboles será más importante que la de otros.

2.5.2. Conjunto de datos

2.5.2.1. Idioma

El conjunto de datos de idioma es autoría de Tatoeba, una base de datos en la web que se almacenan oraciones y traducciones, y corresponde a una lista de oraciones de distintos idiomas, estructurados en tres campos: ID de la oración, código de idioma (tres caracteres) y el texto de la oración, disponible en un archivo de tipo CSV (Tatoeba, s/f).

2.5.2.2. Sentimiento

El conjunto de datos para análisis de sentimientos, corresponde a 250 000 tweets en español, clasificados en tres grupos: tweets con sentimientos positivos, tweets con sentimientos negativos y tweets sin clasificar, con la consideración de que ningún tweet se repite (Garnacho, s/f).

2.6. Apache Zeppelin

Apache Zeppelin es una libreta web que permite análisis de datos y visualización de forma interactiva, además permite crear documentos colaborativos en SQL, Scala, Python y otros lenguajes de programación. Esta herramienta de visualización es de naturaleza software libre y posee integración directa con el framework Apache Spark (no se necesitan módulos, extensiones o librerías separadas).

Zeppelin permite construir gráficos estadísticos como pueden ser: gráficos de barras, gráficos de pastel, gráficos de líneas de tendencia, entre otros (Apache Zeppelin, s/f).

Además, provee métodos de ingreso de datos de usuario a través de formas dinámicas que permiten definir cuadros de texto, listas desplegables y opciones de selección múltiple.

Apache Zeppelin utiliza el concepto de intérprete, donde un intérprete es la extensión que permite que Zeppelin utilice un lenguaje de programación (Python, SQL, Scala) o unidad de procesamiento de datos específica (Spark, Elasticsearch).

2.6.1. Interfaz gráfica

El componente fundamental de la interfaz gráfica de Zeppelin es un párrafo. Una libreta puede estar compuesta por uno o varios párrafos. A cada párrafo se le puede especificar un intérprete, permitiendo combinar varios lenguajes de programación en una sola libreta. Un párrafo está compuesto por tres secciones: sección de código, sección de comandos del párrafo y sección de resultados, como se muestra en la Figura 2.

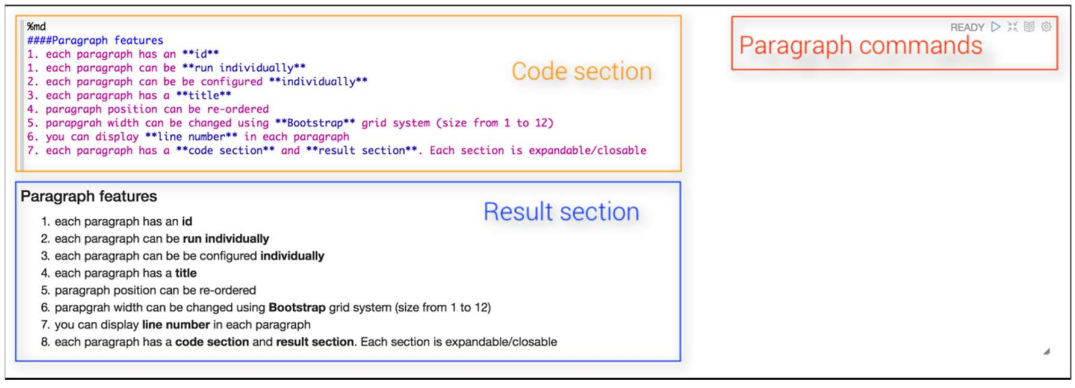


Figura 2. Estructura de un párrafo de Apache Zeppelin.

Tomado de Apache Project, s/f.

La sección de código permite introducir los comandos específicos del lenguaje de programación o unidad de procesamiento, la sección de resultados despliega la salida correspondiente a la ejecución de comandos y la sección de comandos de párrafo permite ejecutar acciones sobre el párrafo. Estas acciones pueden ser: ejecutar, eliminar, pausar y ocultar el párrafo, entre otras.

3. DISEÑO

En este capítulo se diseña el framework de análisis de sentimiento por medio de Apache Spark. El levantamiento de requerimientos se realiza por medio de historias usuario. Se realizan tres diseños: diseño de alto nivel, diseño de bajo nivel y diseño de interfaz.

En el diseño de alto nivel se definen tres fases y nueve actividades, en el diseño de bajo nivel se subdividen las actividades con un mayor nivel de detalle y en el diseño de interfaz se propone la interfaz gráfica con la cual interactúa el usuario final.

3.1. Requerimientos

Las historias de usuario permiten descubrir de manera simple las necesidades del usuario final al manipular un programa. Esta técnica de recolección de requerimientos se enfoca en el problema que se desea resolver con la implementación del software, y deja a discreción del desarrollador el camino que se debe tomar para conseguirlo.

Las historias de usuario se caracterizan por tener un menor nivel de formalidad que otras técnicas de análisis de requerimientos, requieren menor cantidad de documentación, proveen buenos resultados y son especialmente útiles en tres escenarios: el cliente del desarrollador no está familiarizado con programación, no existe una aplicación de referencia o se busca empoderar la creatividad del desarrollador.

3.1.1. Historia de usuario Nro. 1

El usuario final ingresa una palabra clave. Cuando el usuario presiona un botón, la aplicación despliega los comentarios de redes sociales (únicamente idioma español) y con la predicción del sentimiento (bueno o malo).

3.1.2. Historia de usuario Nro. 2

El usuario final presiona un botón y se despliega un gráfico con el resumen estadístico de los resultados encontrados, esto incluye el número de comentarios por cada red social y el número de comentarios por cada tipo de sentimiento.

3.2. Arquitectura general

Se presenta la arquitectura general de la aplicación, con cuatro redes sociales para obtención de datos, en base a la relevancia de las redes sociales y la disponibilidad de librerías de software libre para la extracción de datos.

Twitter, Instagram y Facebook cuentan con herramientas de scraping que buscan y extraen comentarios. Para la red social YouTube existe una herramienta de extracción de comentarios, pero esta herramienta no realiza búsqueda, razón por la que se complementa con la herramienta Selenium.

Para cada una de las redes sociales se crea un archivo intermediario (JSON, CSV o TXT de acuerdo con la herramienta de scraping) y en dicho archivo se descargan los comentarios.

En el framework se realiza la integración de los cuatro archivos, uno para cada red social, generando un Dataframe con todos los comentarios extraídos. Se aplican algoritmos de aprendizaje de máquina para detectar idioma y predecir el sentimiento de los comentarios. Se genera una tabla con los comentarios y gráficos para los resultados estadísticos. El diseño de arquitectura general de la solución de análisis de sentimiento se muestra en la Figura 3.

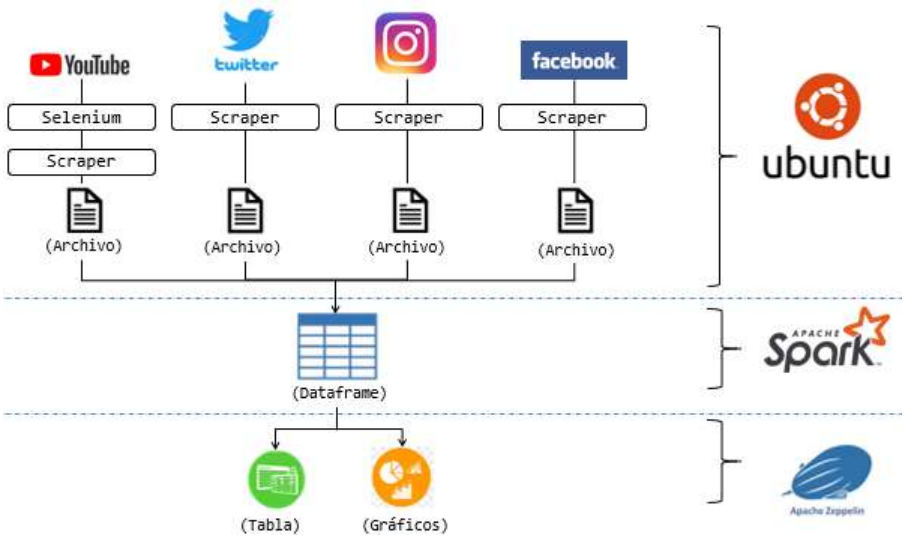


Figura 3. Diseño de la arquitectura general del framework.

3.2.1. Diseño de alto nivel

Para la fase de diseño de alto nivel se define un diagrama de actividades. En este tipo de diseño los rectángulos representan una actividad, los círculos el inicio y fin de las actividades, los conectores representan el flujo de trabajo.

El framework se divide en tres fases: extracción, procesamiento y visualización. En la fase de extracción se establece la interacción directa con las redes sociales, la fase de procesamiento incluye la limpieza de texto y la predicción de idioma y sentimiento, la fase de visualización muestra los comentarios encontrados y un resumen estadístico de los comentarios respecto al sentimiento y red social de origen. En la Figura 4 se observa el diseño de alto nivel del framework.

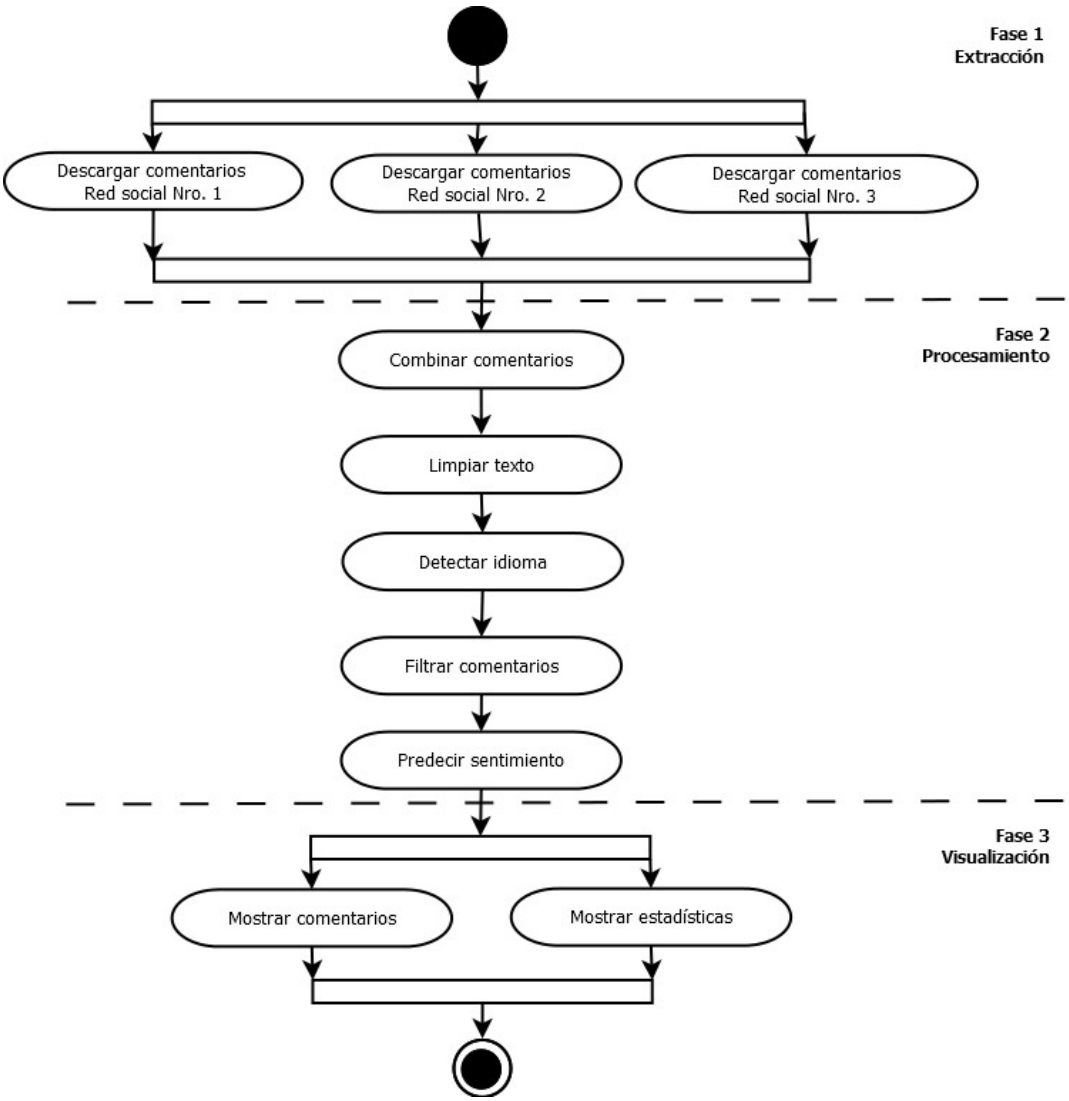


Figura 4. Diseño de alto nivel del framework de análisis de sentimiento.

3.2.2. Diseño de bajo nivel

El diseño de bajo nivel especifica de manera detallada las actividades necesarias para analizar el sentimiento de las redes sociales a través del framework Apache Spark.

En la fase de extracción se realiza una actividad, que es la descarga de datos, insertando una palabra clave para búsqueda de comentarios relacionados en las redes sociales a través de herramientas de scraping.

La fase de procesamiento incluye cinco actividades: combinación de comentarios, limpieza de texto, detección de idioma, filtrado de comentarios y predicción de sentimiento.

La actividad de combinación de comentarios involucra extraer el campo de texto de los comentarios, agregar la red social de donde proviene el comentario y unir todos los registros en un Dataframe.

La actividad de limpieza de texto involucra el uso de expresiones regulares para eliminar nombres de usuarios, hashtags, signos de puntuación, entre otros, que aportan poco significado al sentimiento del comentario.

En la actividad de detección de idioma se realizan dos actividades, entrenamiento del modelo de aprendizaje de máquina y la aplicación del modelo. La actividad de entrenamiento del modelo de aprendizaje de máquina se subdivide en ocho actividades que producen un modelo entrenado, con un porcentaje de precisión de las predicciones realizadas.

La actividad de aplicación del modelo de aprendizaje de máquina toma cinco actividades de las utilizadas durante el entrenamiento del modelo, se toman aquellas necesarias para realizar la predicción, pero se eliminan actividades como la evaluación del modelo. Una vez realizada la detección de idioma, se eliminan aquellos comentarios que no pertenecen a idioma español, denominada actividad de filtrado de comentarios.

La actividad de detección de sentimiento implica las dos actividades descritas anteriormente (entrenamiento y aplicación del modelo de aprendizaje de máquina), respecto a un conjunto de datos de entrenamiento diferente, y con un propósito distinto (predicción de sentimiento).

La última fase del flujo de trabajo es la fase de visualización, en la que se despliegan los comentarios, con la detección de idioma y predicción de sentimiento, además de gráficos con el resumen estadístico de los resultados. Para generar estos resultados se agrupan los comentarios en base a la categoría de sentimiento y la red social de la fueron extraídos. El diagrama de diseño se especifica en la Figura 5.

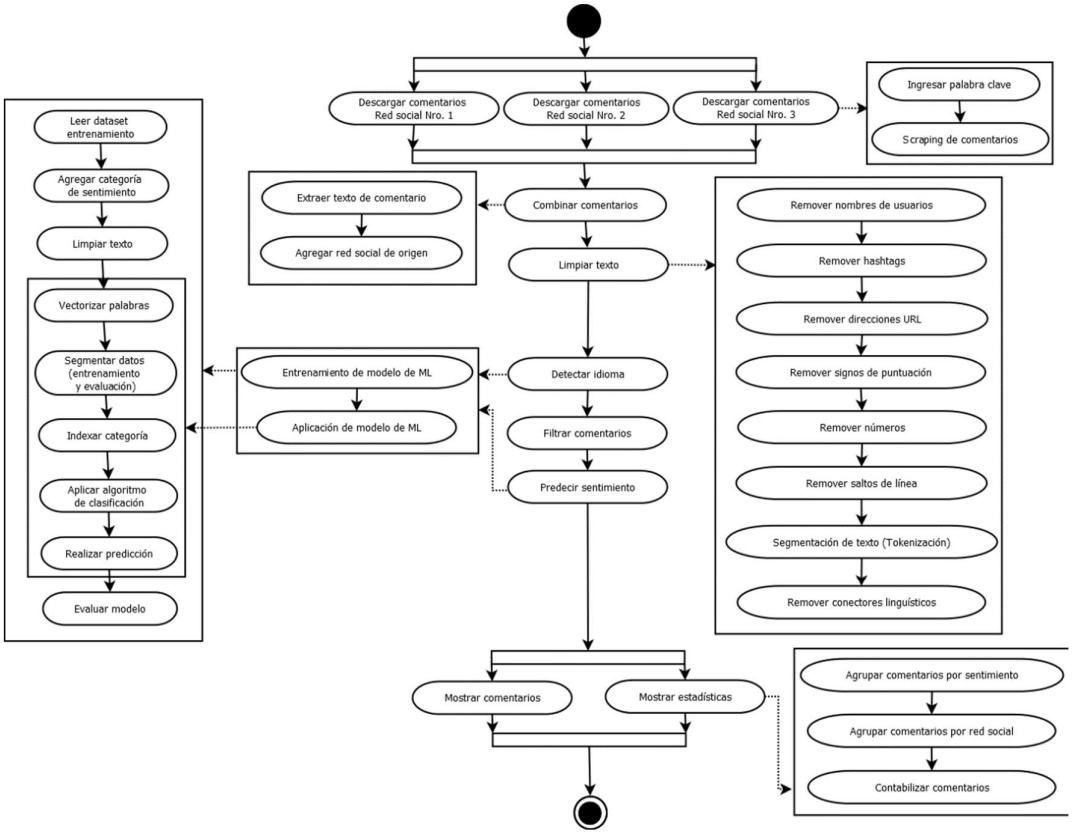


Figura 5. Diseño de bajo nivel del framework de análisis de sentimiento.

3.2.3. Diseño de interfaz

La interfaz gráfica cuenta con un cuadro de texto, en el que el usuario ingresa la palabra clave, un listado de los comentarios de redes sociales con las

predicciones de idioma y sentimiento realizadas, y un gráfico con el resumen estadístico de los resultados.

Los resultados estadísticos incluyen: totalidad de comentarios, el número total de comentarios respecto a cada tipo de sentimiento encontrado y el número total de comentarios respecto a la red social de origen. La propuesta de interfaz gráfica se observa en la Figura 6.

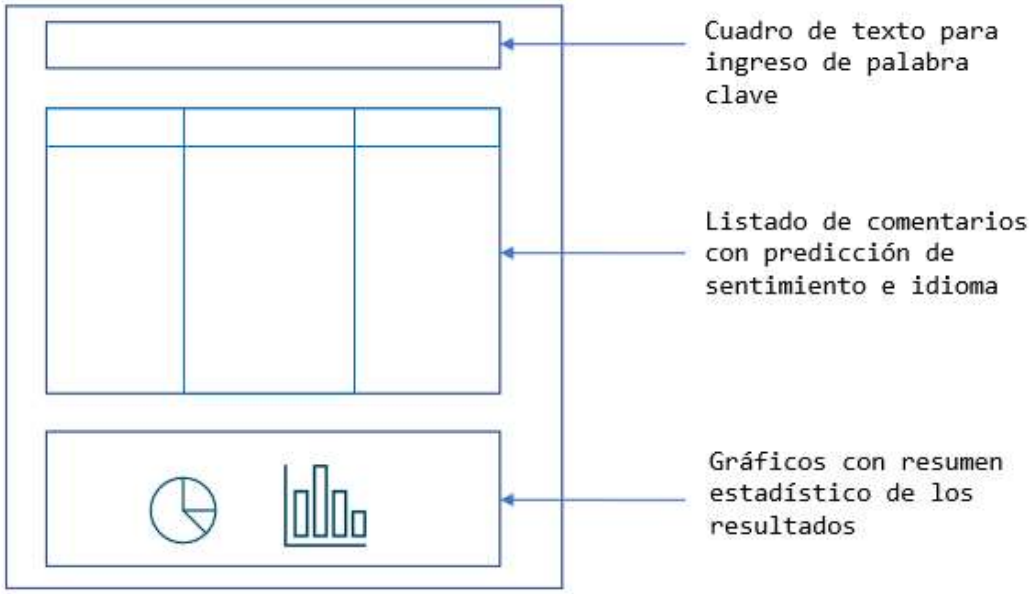


Figura 6. Propuesta de interfaz gráfica del framework.

4. IMPLEMENTACIÓN

En este capítulo se detalla el proceso de implementación del framework. Por medio de herramientas de scraping se buscan comentarios en redes sociales, para luego descargarlos en archivos de texto. Un modelo de aprendizaje de máquina es entrenado y aplicado para detección de idiomas, y se eliminan los comentarios que no corresponden al idioma español. Un segundo modelo es entrenado para predecir el sentimiento de los comentarios. Se implementa una interfaz gráfica para visualizar los resultados de análisis de sentimiento. Se llevan a cabo pruebas unitarias en paralelo a cada paso de la implementación.

4.1. Extracción de datos

El proceso de extracción de datos corresponde a la interacción directa con las redes sociales. Desde cada red social se buscan comentarios relacionados a una palabra clave a través de herramientas de scraping. Los comentarios encontrados se descargan en archivos intermediarios y son almacenados en el sistema operativo Ubuntu. El tipo de archivo es dependiente de la herramienta de scraping utilizada, siendo los principales formatos JSON, CSV, TXT, entre otros.

4.1.1. Datos desde Twitter

Para extraer los comentarios generados en Twitter se utiliza la herramienta de Python twitter-scraper v0.4.2 (última versión estable a la fecha de escritura). Esta herramienta realiza consultas a la API de JavaScript que la plataforma Twitter tiene a disposición del público. Al momento de la escritura se requiere que Python tenga una versión mínima de 3.6.

En la Figura 7 se observa el ejemplo de una consulta por uso de la herramienta, se ha seleccionado el hashtag coronavirus (#coronavirus), y se muestran los resultados de los comentarios asociados a dicha etiqueta. Se puede verificar que la consulta extrae todos los comentarios, sin importar el idioma en el que fueron escritos inicialmente.

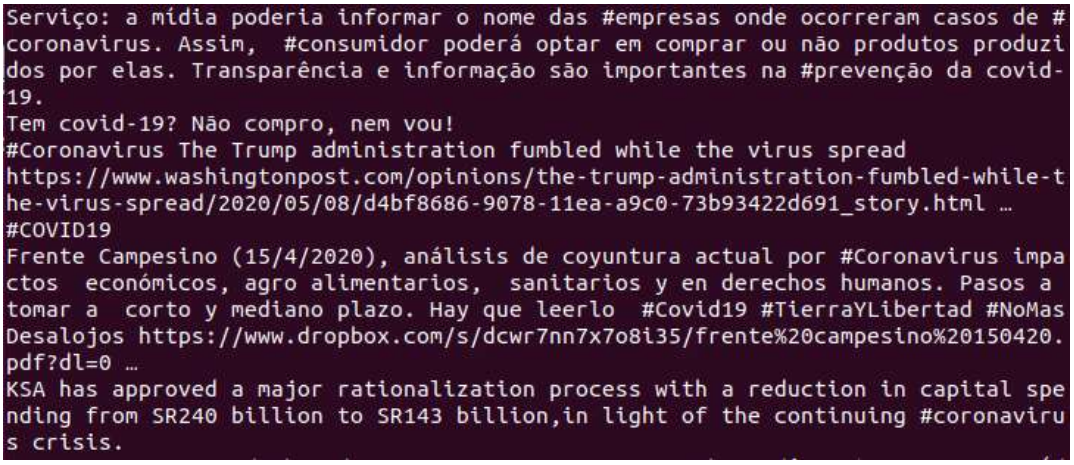


Figura 7. Consulta de comentarios con el scraper de Twitter.

4.1.2. Datos desde Facebook

Para consultar los datos de la plataforma Facebook se utiliza la herramienta para Python facebook-scraper v0.1.12 (última versión a la fecha de escritura). En dicha herramienta se puede realizar una consulta de manera generalizada, utilizando el buscador de la plataforma, o mediante la especificación de un grupo o página particular. Permite adicionalmente consultar las reacciones a las publicaciones.

Similar a la herramienta utilizada para consultar datos en la plataforma Twitter, esta herramienta no distingue del idioma de las publicaciones, como se puede observar en la Figura 8, y es necesario realizar un proceso de limpieza para excluir los resultados que no se encuentren en español.

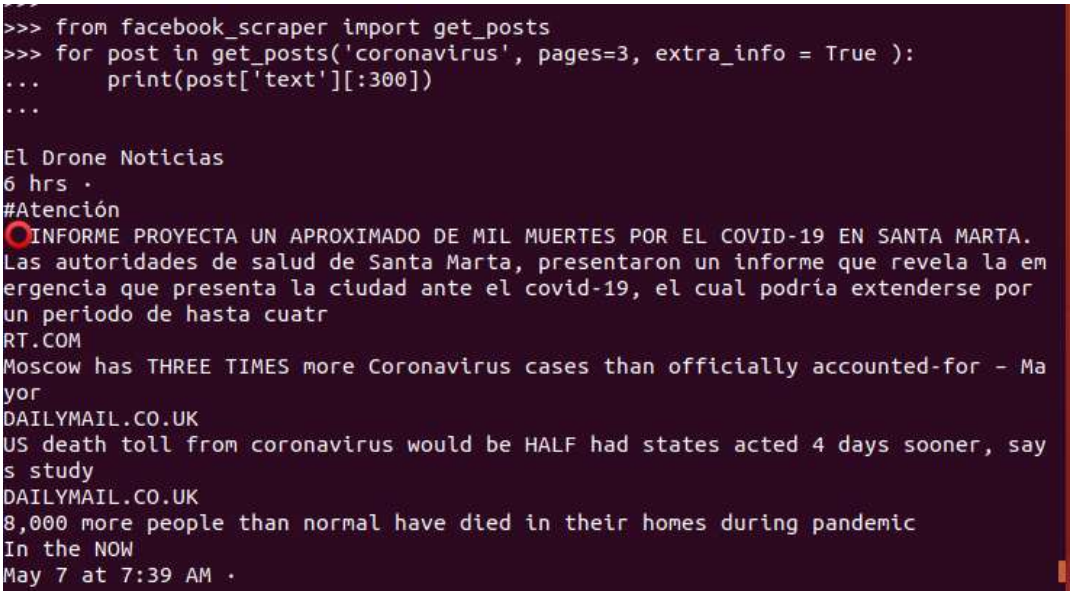


Figura 8. Consulta de comentarios con el scraper de Facebook

4.1.3. Datos desde YouTube

La extracción de datos desde la red social YouTube representa un caso particular, pues no existe una herramienta que realice búsqueda de comentarios en base a una palabra clave. La herramienta Yotube-Comment-downloader encontrada permite la descarga de los comentarios de un video especificado, y su código original se encuentra disponible en el Anexo 1 (Bouman, s/f).

La limitante del script original para descarga de comentarios recibe un identificador de video, y no una palabra clave como se propuso en el diseño inicial del framework. Para solventar esta limitante, se utiliza una herramienta de automatización del proceso de navegación web.

La herramienta Selenium permite ejecutar tareas de pruebas en navegadores web, y bajo este contexto se creó la función scrape_youtube en el script downloader.py de descarga de comentarios. La función scrape_youtube utiliza la librería Selenium y el navegador web Mozilla Firefox para acceder a la página principal de YouTube para Ecuador, introduce una palabra en el cuadro de

búsqueda de YouTube y envía la tecla *Enter* que despliega los videos encontrados.

Los elementos web con las direcciones URL de los videos fueron identificados por medio de la herramienta de inspección de elemento del navegador y se determinó que el ID del elemento web con la dirección URL era video-title, como se puede observar en la Figura 9.

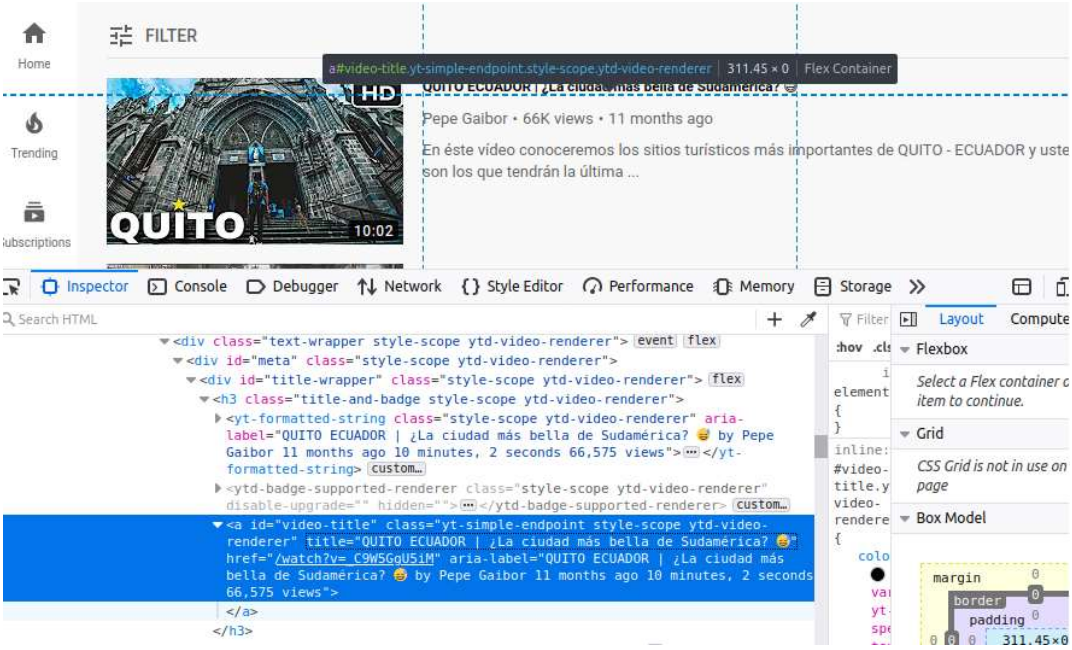


Figura 9. Identificación del ID del video de Youtube.

Una vez obtenido el ID del elemento web, se utilizó la función de Selenium find_elements_by_id(), escrita en lenguaje de programación Python, que permite obtener todos los elementos de la página web dado el ID del elemento.

Esta función devuelve una lista de elementos web, almacenada como listadescripcion, y por medio de una estructura *for*, se recorrió de manera iterativa dicha lista, donde de cada elemento se obtuvo el atributo *href* que contiene la dirección URL, se segmento la dirección URL por medio de la función urlparse() y con expresiones regulares se eliminaron las caracteres que no correspondían al ID del video.

Los identificadores se almacenaron en una nueva lista denominada listaidvideo, y dicha lista se retornó como resultado de la ejecución de la función scrape_youtube(), como se muestra en la Figura 10.

```
def scrape_youtube(palabra):
    paginayoutube = "https://www.youtube.com/?hl=es&gl=EC"
    driver = webdriver.Firefox()
    driver.get(paginayoutube)
    driver.find_element(By.NAME, "search_query").click()
    driver.find_element(By.NAME, "search_query").send_keys(palabra)
    driver.find_element(By.NAME, "search_query").send_keys(Keys.ENTER)
    time.sleep(5)
    listadescripcion = driver.find_elements_by_id("video-title")
    listaidvideo = []
    for elemento in listadescripcion:
        elemento.get_attribute("href")
        print(elemento.get_attribute("href"))
        value = elemento.get_attribute("href")
        url_data = urlparse(value)
        print(url_data.query)
        idvideo= url_data.query.replace("v=","")
        listaidvideo.append(idvideo)
    #print(listaidvideo)
    return listaidvideo
```

Figura 10. Función scrape_youtube() con un parámetro de entrada.

El script original downloader.py recibe originalmente los siguientes parámetros: ID del video, nombre del archivo de salida JSON donde se almacenan los comentarios y el número máximo de comentarios a descargar.

Se tomo en cuenta dos consideraciones para la modificación del script, en primer lugar, Selenium no permite ejecutar Firefox como un superusuario, y segundo, pasar parámetros a un script de Python aumenta la complejidad cuando se utiliza el intérprete de Python o el intérprete de Shell de Zeppelin.

Para solventar el problema de la ejecución de Firefox, se optó por usar el comando *su*, que permite ejecutar un comando como otro usuario, y se creó un usuario que no pertenece el grupo de superusuarios. Para solventar el conflicto del paso de parámetros al interprete, se modificó el archivo para descarga de comentarios de YouTube, eliminando los parámetros.

En el script downloader.py se define la función leer_palabra_archivo(), para proveer de un método de ingreso de la palabra buscada en las redes sociales. Esta función, mostrada en la Figura 11, lee un archivo de texto ubicado en el mismo directorio del archivo downloader.py y retorna la palabra encontrada.

```
def leer_palabra_archivo():
    f = open("archivoconpalabra.txt","r")
    palabra_busqueda = f.read()
    print(palabra_busqueda)
    return palabra_busqueda
```

Figura 11. Función leer_palabra_archivo().

Para la eliminación de los argumentos restantes (límite de comentarios descargados y nombre del archivo de salida), se modifica la función principal del script para descarga de comentarios downloader.py. Se limita el número de comentarios a los diez comentarios por video, se establece el nombre del archivo de salida como comentarios.json y se modifica la función que tomaba un solo ID (en el script original) para que recorra la lista de ID de videos retornada desde la función scrape_youtube(). La función main() modificada se muestra en la Figura 12.

```
def main(palabraclave):

    try:
        output = 'comentarios.json'
        limit = 10
        palabra = leer_palabra_archivo()
        print(palabra)
        lista_youtube_id = scrape_youtube(palabra)
        file = open("comentarios.json","r+")
        file.truncate(0)
        file.close()

        for youtube_id in lista_youtube_id:

            print('Downloading Youtube comments for video:', youtube_id)
            count = 0
            with io.open(output, 'a', encoding='utf8') as fp:
                sys.stdout.write('Downloaded %d comment(s)\r' % count)
                sys.stdout.flush()
                for comment in download_comments(youtube_id):
                    comment_json = json.dumps(comment, ensure_ascii=False)
                    print(comment_json.decode('utf-8') if isinstance(comment_json, bytes) else comment_json, file=fp)
                    count += 1
                    sys.stdout.write('Downloaded %d comment(s)\r' % count)
                    sys.stdout.flush()
                    if limit and count >= limit:
                        break
                print('\nDone!')

    except Exception as e:
        print('Error:', str(e))
        sys.exit(1)
```

Figura 12. Modificación del script de descarga de comentarios.

Una vez realizada las modificaciones, se ingresa una palabra clave y se obtiene un archivo JSON con los comentarios de los videos, como se puede observar en la Figura 13.


```
{"cid": "UgzYoZZAqdHNu7czOud4AaABAg", "text": "Plata,plata plata,señores", "tim
e": "Hace 1 año", "author": "Anthony Dagua", "votes": "1", "photo": "https://yt
3.ggpht.com/a/AATXAJygcH4RVsxmk1rwzJkyXLbHJV1s72lglBI1aA=s48-c-k-c0xffffffff-no
-rj-mo"}
{"cid": "Ugyxek8ter_ZG1Cl3vF4AaABAg", "text": "Hechos de ricos, sueños de pobre
s 😑", "time": "Hace 1 año", "author": "Gissell G.", "votes": "7", "photo": "ht
tps://yt3.ggpht.com/a/AATXAJy04rGGzdScO8iJCXoEpkbvsmtOPJtEeoftrQ=s48-c-k-c0xfff
fffff-no-rj-mo"}
{"cid": "UgzCorEmmlEp4abDUzp4AaABAg", "text": "Intro del sketch de enchufe tv \
"chichico va a comprar condones\"", "time": "Hace 1 año", "author": "The Taco's
 Project", "votes": "1", "photo": "https://yt3.ggpht.com/a/AATXAJy8Lvxz0F12YNkw
96wCh7IhuLth5jeWPn0N7w=s48-c-k-c0xffffffff-no-rj-mo"}
{"cid": "Ugw4DaSbL1dVfGp4pG54AaABAg", "text": "Esa U es una máquina, el videito
 no más quedo feito, teniendo una escuela de cine y tv derian hacre un producto
 genial, saludos a todos", "time": "Hace 2 años", "author": "BATRACIO STUDIO FÍ
LMICO", "votes": 0, "photo": "https://yt3.ggpht.com/a/AATXAJzMu7OQVTPtjjIAZR3Al
L16vsbPm9CwnGuq_A=s48-c-k-c0xffffffff-no-rj-mo"}
{"cid": "UgyjmAg40X_YT7Eda0t4AaABAg", "text": "Achof la sanpanchof", "time": "H
ace 2 años", "author": "Alan", "votes": "1", "photo": "https://yt3.ggpht.com/a/
AATXAJwjiJfNEYBd5y0AbZF6hXpl8N24Tfx66nABqg=s48-c-k-c0xffffffff-no-rj-mo"}
```


Figura 13. Archivo JSON con comentarios de YouTube.

4.1.4. Datos desde Instagram

La extracción de datos de la red social Instagram se realiza por medio del scraper InstaTouch, que utiliza la API para recuperar publicaciones basadas en nombre de usuario, hashtag o ubicación, además de los comentarios de dichas publicaciones.

La instalación de esta librería se realizó a través del administrador de paquetes npm. Instatouch recibe como parámetros el hashtag, el nombre de archivo, el tipo de archivo (permite seleccionar CSV o JSON) y genera un archivo con los comentarios asociados al hashtag, como se puede observar en la Figura 14.

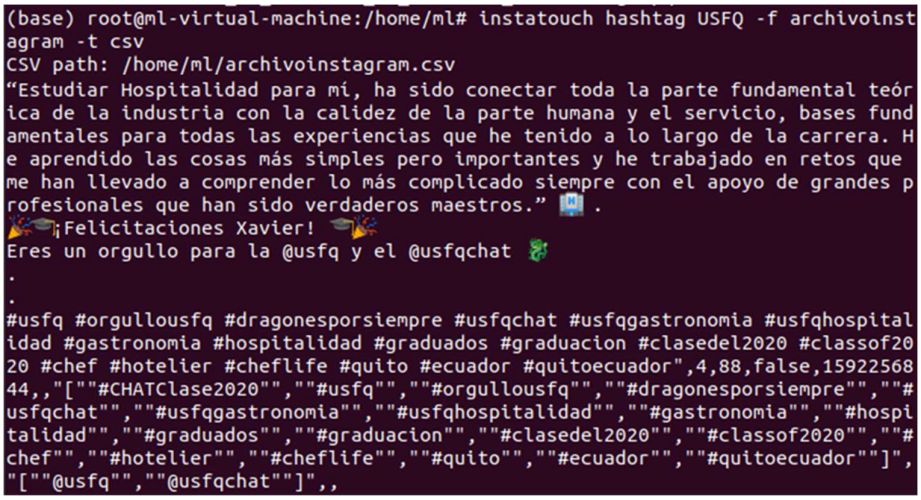


Figura 14. Archivo JSON con comentarios de Instagram.

4.1.5. Integración al framework Spark

Se crea un dataframe con el nombre redesSociales_df, y se realiza la lectura con la sesión Spark denominada spark. La característica de Spark es la inferencia del esquema de forma automática (opción por defecto) al leer un archivo JSON como se puede observar en la Figura 15.

```
>>> facebook_df4.printSchema()
root
 |-- _corrupt_record: string (nullable = true)
 |-- comments: long (nullable = true)
 |-- fetched_time: string (nullable = true)
 |-- image: string (nullable = true)
 |-- likes: long (nullable = true)
 |-- link: string (nullable = true)
 |-- post_id: string (nullable = true)
 |-- post_text: string (nullable = true)
 |-- post_url: string (nullable = true)
 |-- reactions: struct (nullable = true)
 |    |-- like: long (nullable = true)
 |    |-- sorry: long (nullable = true)
 |-- shared_text: string (nullable = true)
 |-- shares: long (nullable = true)
 |-- text: string (nullable = true)
 |-- time: string (nullable = true)
 |-- w3_fb_url: string (nullable = true)
```

Figura 15. Inferencia automática de esquema de un archivo JSON.

Se puede observar la inferencia automática de columnas, tales como tiempo y fecha cuando se publicó, el número de me gusta que recibió, la dirección URL donde se puede encontrar, entre otros.

Merece la pena realizar énfasis a la columna text, columna que contiene el comentario asociado a la búsqueda, y como se ve en la Figura 16 contiene el texto compartido por la publicación de Facebook.

```
>>> facebook_df.select("text").show(20, False)
+-------------------------------------------------------------------------------
|text
+-------------------------------------------------------------------------------
|DAILYMAIL.CO.UK
Three children have died from a rare inflammatory syndrome linked to coronavirus|
+-------------------------------------------------------------------------------
```

Figura 16. Consulta al dataframe con el comentario de Facebook.

La inferencia se realiza para cada uno de los archivos con los comentarios. Para el caso de los dataframe de Twitter, YouTube y Facebook se utiliza la inferencia de archivos JSON, mientras que para el dataframe de Instagram se usa la inferencia de archivos CSV.

Mediante la operación unión se combinan los dataframes de Twitter, YouTube e Instagram, gracias a las dos columnas texto y red_social disponibles en cada uno, y el resultado de esta operación es un nuevo dataframe denominado redesSociales_df.

El dataframe de Facebook presenta una particularidad en las pruebas unitarias realizadas, debido a que para ciertas palabras clave no se encuentran resultados, devolviendo un valor nulo, y produciendo una excepción de apuntamiento a nulo.

Mediante la estructura try, se genera una excepción controlada en caso de que el scraper no pueda comentarios desde Facebook. En caso de que la obtención de comentarios sea posible para esta red social, se realiza la operación de unión del dataframe facebook_df con el dataframe redesSociales_df. La programación que comprende la combinación y el control de excepciones se puede observar en la Figura 17.

```
twitter_df = spark.read.format('json').load("/home/ml/Downloads/zeppelin-0.9.0-
preview1-bin-all/archivotwitter.json")
twitter_df = twitter_df.withColumn("redsocial",lit("twitter"))
twitter_df = twitter_df.select("text","redsocial")

youtube_df = spark.read.format('json').load("/home/user1/comentarios.json")
youtube_df = youtube_df.withColumn("redsocial",lit("youtube"))
youtube_df = youtube_df.select("text","redsocial")
instagram_df = spark.read.format('csv').option("header", "true").load("/home/ml/
Downloads/zeppelin-0.9.0-preview1-bin-all/archivoinstagram.csv")

instagram_df = instagram_df.withColumn("redsocial",lit("instagram"))
instagram_df = instagram_df.select("description","redsocial")
instagram_df = instagram_df.withColumnRenamed("description","text")

redesSociales_df = twitter_df.union(youtube_df).union(instagram_df)
try:
  facebook_df = spark.read.format('json').load("/home/ml/Downloads/
zeppelin-0.9.0-preview1-bin-all/archivofacebook.json")
  facebook_df = facebook_df.withColumn("redsocial",lit("facebook"))
  facebook_df = facebook_df.select("text","redsocial")
  redesSociales_df = redesSociales_df.union(facebook_df)
except:
  print("No se pudo obtener comentarios de facebook")
```

Figura 17. Integración de los archivos generados por scraping.

4.2. Detección de idioma

Las librerías para extracción de datos incluyen comentarios de todos los idiomas, razón por la cual se debe discriminar comentarios que no se encuentren en español. La automatización de este proceso se realiza a través del aprendizaje de máquina, donde se entrena un modelo en base a un conjunto de datos con comentarios en varios idiomas, previamente etiquetados.

4.2.1. Conjunto de datos

El conjunto de datos para el entrenamiento del modelo corresponde a un archivo CSV, con tres campos: ID, código del idioma (formato ISO 639-3), y texto de la oración. Se realiza la lectura por medio de una sesión Spark, por medio de la opción para lectura de archivos CSV, como se observa en el *Figura 18*.

```
idioma_df = spark.read.csv('/home/ml/Downloads/DatasetIdiomas/sentences.csv',
                           sep='\t')
```

Figura 18. Lectura de archivo CSV.

En la Figura 19 se observa la inferencia del esquema del dataframe para detección de idioma, con las tres columnas descritas con anterioridad (ID, código y texto). Se visualiza los tres primeros resultados de la consulta para las oraciones en idioma español, se agrupan las oraciones en base a la columna de código de idioma y se contabiliza para cada agrupación. La última consulta es devuelve el número de registros totales, para este caso 8351252.

```
>>> idioma_df.printSchema()
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)

>>> idioma_df.where(idioma_df._c1 == "spa").show(3)
+----+---+--------------------+
| _c0|_c1|                 _c2|
+----+---+--------------------+
|2481|spa|   ¡Intentemos algo!|
|2482|spa|Tengo que irme a ...|
|2483|spa|¿Qué estás haciendo?|
+----+---+--------------------+
only showing top 3 rows

>>> idioma_df.groupBy(idioma_df._c1).count().show(3)
+---+-----+
|_c1|count|
+---+-----+
|ces|43755|
|ckt|  531|
|pus|   44|
+---+-----+
only showing top 3 rows

>>> idioma_df.count()
8351252
```

Figura 19. Esquema del dataframe de idioma y consultas SQL.

Debido a que el conjunto de datos almacena oraciones de todos los idiomas del globo, existe un problema al momento de entrenar el modelo por medio de aprendizaje de máquina debido a las diferentes codificaciones usadas. Para solucionar el problema se utiliza la característica de Apache Spark que permite definir funciones de usuario, denominada User Defined Functions.

En la *Figura 20* se define y registra la funciónCodificar que recibe un dato de tipo String, y lo trata de decodificar en UTF-8. En caso afirmativo devuelve la oración decodificada, caso contrario devuelve un valor nulo. Se procede a eliminar todas las filas del dataframe que tengan al menos un valor nulo en alguna de sus columnas.

```
def funcionCodificar(oracion):
    try:
        return oracion.decode('utf-8')
    except:
        return None


spark.udf.register("funcionCodificar", funcionCodificar)
from pyspark.sql.functions import udf
funcionCodificar_udf = udf(funcionCodificar)
idioma_df = idioma_df.select("_c0","_c1","_c2", funcionCodificar_udf("_c2").alias("nuevaColumna"))
idioma_df=idioma_df.na.drop()
```

Figura 20. Función definida por el usuario funcionCodificar().

Luego de eliminar las filas que no corresponden a codificación UTF-8 se ha eliminado el problema de entrenamiento del modelo para codificación que no sea UTF-8 y se ha eliminado el número total de registros a 3432901, como se observa en la Figura 21.

```
>>> idioma_df.count()
3432901
```

Figura 21. Consulta del número de registros codificados en UTF-8.

4.2.2. Entrenamiento del modelo

Para el entrenamiento del modelo se utiliza la librería fastText, una librería open source ligera para clasificación y representación de texto. En él se segmenta un 80% de los datos para entrenamiento y 20% para evaluación del modelo. Se definen las ubicaciones de los archivos de entrenamiento y evaluación del modelo.

El modelo es entrenado por medio del entrenamiento supervisado, que recibe como parámetro el archivo de entrenamiento, y se almacena el modelo en un archivo de extensión BIN. La programación se observa en la *Figura 22*.

```
## segmentación datos entrenamiento evaluación
idioma_entrenamiento_df, idioma_evaluacion_df = idioma_df.randomSplit([0.8, 0.2], 42)

## Formato de almacenamiento __label__<class> <text>
import fasttext
TRAIN_FILE = '/home/ml/Downloads/DatasetIdiomas/fasttext_entrenamiento.txt'
TEST_FILE = '/home/ml/Downloads/DatasetIdiomas/fasttext_evaluacion.txt'
# Creación de archivo de entrenamiento
with open(TRAIN_FILE, 'w') as fp:
        for i, row in enumerate(idioma_entrenamiento_df.toLocalIterator()):
            fp.write('__label__%s %s\n' % (row['_c1'],
                                           row['_c2']))
            if i % 1000 == 0:
                print(i)


# Creación de archivo de evaluación
with open(TEST_FILE, 'w') as fp:
        for i, row in enumerate(idioma_evaluacion_df.toLocalIterator()):
            fp.write('__label__%s %s\n' % (row['_c1'],
                                           row['_c2']))
            if i % 1000 == 0:
                print(i)


# Entrenamiento del modelo
model = fasttext.train_supervised(TRAIN_FILE)
model = save_model("deteccion_idioma.bin")
```

Figura 22. Código de entrenamiento del modelo para detección de idioma.

El entrenamiento del modelo de aprendizaje de máquina se realiza sobre el dataframe con aproximadamente tres millones de registros. En la Figura 23 se puede observar un estimado de 2 horas 19 minutos y 8 segundos para la culminación del entrenamiento.

```
Progress:    0.3% words/sec/thread:     4659 lr:  0.099669 avg.loss:  1.328991 ETA
Progress:    0.3% words/sec/thread:     4652 lr:  0.099668 avg.loss:  1.327174 ETA
Progress:    0.3% words/sec/thread:     4649 lr:  0.099667 avg.loss:  1.325772 ETA
Progress:    0.3% words/sec/thread:     4644 lr:  0.099666 avg.loss:  1.323881 ETA
Progress:    0.3% words/sec/thread:     4643 lr:  0.099665 avg.loss:  1.321942 ETA
Progress:    0.3% words/sec/thread:     4638 lr:  0.099664 avg.loss:  1.319279 ETA
Progress:    0.3% words/sec/thread:     4628 lr:  0.099663 avg.loss:  1.319450 ETA
:   2h19m 8s
```

Figura 23. Tiempo de entrenamiento del modelo para detección de idioma.

4.2.3. Evaluación del modelo

La evaluación del modelo se logra aplicando el modelo entrenado detección_idioma.bin sobre el conjunto de datos de evaluación que se separado con anterioridad. Se analizan el número de registros totales de los datos para evaluación y se calcula la precisión del modelo, es decir el número de veces que el modelo realizó una predicción correcta.

La precisión del modelo se calcula al comparar la predicción realizada, con la etiqueta del idioma en el conjunto de evaluación. Se obtiene una precisión del 97,1% del modelo de detección de idioma, para un total de 686278 registros destinados a evaluación, como se puede observar en la Figura 24.

```
>>> def print_results(N, p, r):
...     print("N\t" + str(N))
...     print("P@{}\t{:.3f}".format(1, p))
...     print("R@{}\t{:.3f}".format(1, r))
...
>>> print_results(*modelo.test(TEST_FILE))
N       686278
P@1     0.971
R@1     0.971
```

Figura 24. Precisión del modelo de detección de idioma

La detección de idioma se realiza siguiendo el proceso de limpieza definido en la creación y entrenamiento del modelo de aprendizaje de máquina para análisis de sentimiento. Se remueven nombres de usuario, etiquetas, direcciones, puntuación, números y se añade la eliminación de símbolos de salto de línea, con el código disponible en la Figura 25.

```
def funcionEliminarSlash(column):
    return trim(lower(regexp_replace(column,'\\n', ''))).alias('stopped')
```

Figura 25. Función para eliminar los saltos de línea en el texto.

Para realizar el análisis de modelo, se utiliza la característica de Apache Spark que permite incluir funciones definidas por el usuario (UDF). Esto se debe a que el modelo realiza la predicción de idioma por cada línea, y en el caso del dataframe se va a ejecutar la función para cada una de sus filas.

La función es definida en un archivo de texto con extensión .py, en este caso el archivo fasttext_lang_classifier.py, en este archivo se importa la librería fasttext (para procesamiento de lenguaje), se carga el modelo entrenado anteriormente para la detección de idioma (detección_idioma.bin) y se crea la función para predecir el idioma predict_language que tendrá un parámetro del tipo columna y devolverá una lista con la etiqueta predicha por el modelo de ML. La función se observa en la Figura 26.

```
def predict_language(msg):
    pred = model.predict(msg)[0][0]
    return pred
```

Figura 26. Función de predicción de idioma.

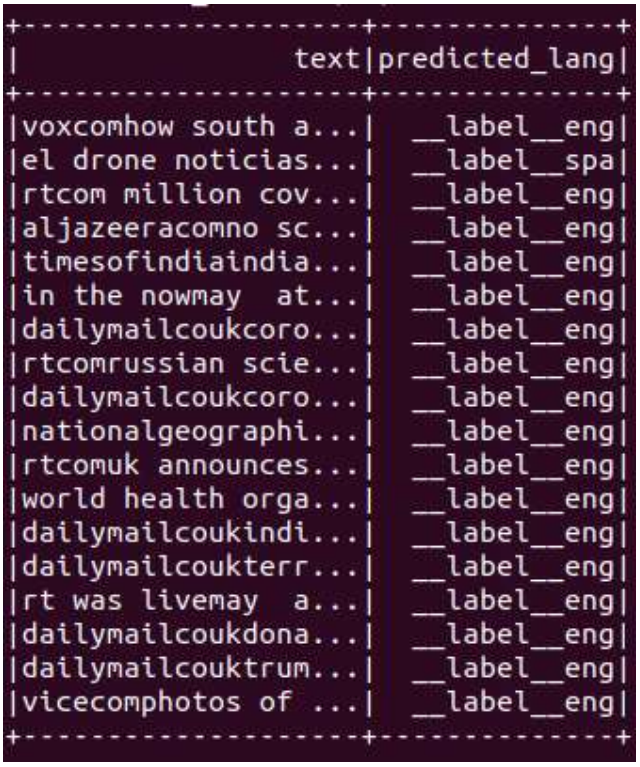
El resultado obtenido es una nueva columna denominada idioma_predicho que almacena la predicción de idioma realizada por el modelo de ML, como un código de tres caracteres. En la Figura 27 se observa la llamada al archivo externo de Python, al incluir el modelo y el archivo en el contexto sc de Spark.

```
from pyspark.sql.functions import col, udf
import fasttext_lang_classifier
udf_predict_language = udf(fasttext_lang_classifier.predict_language)
sc.addFile('deteccion_idioma.bin')
sc.addPyFile('fasttext_lang_classifier.py')

twitter_df = twitter_df.withColumn('predicted_lang',
                                   udf_predict_language(col('text')))
```

Figura 27. Llamada desde el contexto de Spark al archivo de Python.

Se realiza el llamado a la UDF udf_predict_language que recibe la columna texto y almacena la predicción para cada fila en la nueva columna idioma_predicho. El resultado se observa en la Figura 28.



text	predicted_lang
voxcomhow south a...	__label__eng
el drone noticias...	__label__spa
rtcom million cov...	__label__eng
aljazeeracomno sc...	__label__eng
timesofindiaindia...	__label__eng
in the nowmay at...	__label__eng
dailymailcoukcoro...	__label__eng
rtcomrussian scie...	__label__eng
dailymailcoukcoro...	__label__eng
nationalgeographi...	__label__eng
rtcomuk announces...	__label__eng
world health orga...	__label__eng
dailymailcoukindi...	__label__eng
dailymailcoukterr...	__label__eng
rt was livemay a...	__label__eng
dailymailcoukdona...	__label__eng
dailymailcouktrum...	__label__eng
vicecomphotos of ...	__label__eng

Figura 28. Consulta al dataframe que almacena comentarios

Se procede a remover los caracteres innecesarios de la columna de predicción y se filtra el dataframe para que únicamente almacenen comentarios en español. La Figura 29 muestra la función para remover los caracteres, además del filtrado de comentarios para el idioma español y el resultado se observa en la Figura 30.

```
def removerCodigo(column):
    return trim(lower(regexp_replace(column,'__label__', ''))).alias('stopped')


twitter_df=twitter_df.withColumn('predicted_lang',removerCodigo('predicted_lang'))
twitter_df=twitter_df.where(twitter_df.predicted_lang == "spa")
```

Figura 29. Función para eliminar caracteres.

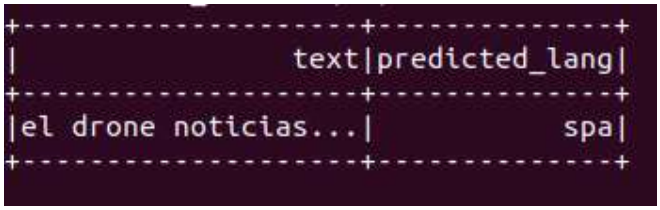


Figura 30. Resultado del filtrado de comentarios.

4.3. Análisis de sentimiento

4.3.1. Entrenamiento del modelo

Para entrenar el modelo por medio del framework Apache Spark, se comienza preparando el conjunto de datos. El conjunto de datos corresponde a 250 000 tweets clasificados en 3 grupos: sentimientos positivos, sentimientos negativos y una categoría de tweets crudos sin clasificar.

Se crea una sesión Spark y se utiliza la función de lectura de texto, creando un dataframe para sentimientos positivos, y se le agrega la columna categoría con valor bueno. El procedimiento se repite para la creación del dataframe con sentimientos negativos, con la columna categoría y para cada fila se agrega en la columna el valor malo.

Una vez creados ambos dataframes, se procede a realiza la operación de unión de dataframes. El único requisito es que la columna de cada dataframe tenga la misma denominación, de tal manera se tienen dos columnas comunes para ambos dataframe (value y categoria). El resultado obtenido se puede observar en la Figura 31, con 177576 registros o filas totales.

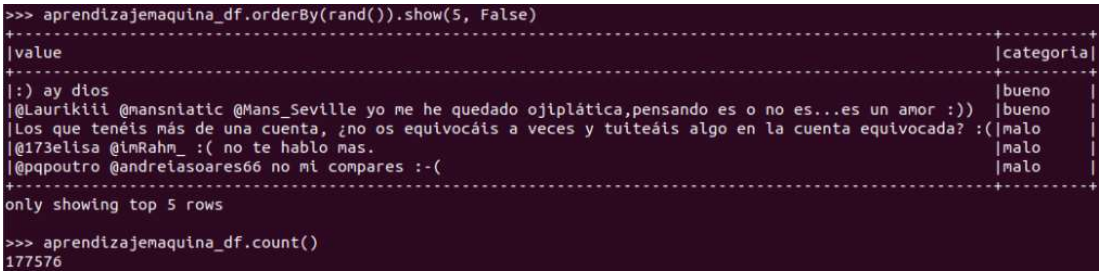


Figura 31. Dataframe del conjunto de datos de tweets clasificados.

Finalmente se realiza una consulta para conocer el número de registros para cada categoría. Para la categoría bueno se tienen 50066 registros, y para la categoría malo se tienen 50036 registros, como se puede apreciar en la Figura 32.

```
>>> aprendizajemaquina_df.groupBy("categoria").count().show()
+---------+-----+
|categoria|count|
+---------+-----+
|    bueno|50066|
|     malo|50036|
+---------+-----+
```

Figura 32. Número de registros para cada categoría de sentimiento.

4.3.2. Proceso de limpieza de texto

La limpieza de texto permite extraer las palabras relevantes de cada comentario. Las principales acciones que involucran el proceso de limpieza es la conversión a minúsculas de cada palabra, remover: signos de puntuación, números, espacios, etiquetas (como <html> o <p>), conectores de lenguaje.

Adicionalmente para el conjunto de datos es necesario eliminar los nombres de usuarios y las direcciones HTTP que se encuentran de manera frecuente en los comentarios. Para la limpieza de datos se utiliza la librería de aprendizaje máquina de PySpark denomanida ml.

En la Figura 33 se muestra una consulta de los primeros 5 valores del dataframe, con texto crudo que no ha sido procesado previamente.

```
>>> aprendizajemaquina_df.select("value").show(5,False)
+--------------------------------------------------------------------------------------------------------------------+
|value                                                                                                               |
+--------------------------------------------------------------------------------------------------------------------+
|Se imaginan a los chicos agradeciendo por el premio con cara de orgullo?.Que bonito :).#MTVHottest One Direction|
|Eclesiastes4:9-12 ♡Siempre, promesa :)  https://t.co/XbrYsqa43T                                                     |
|@pedroj_ramirez Qué saborío, PJ. ya no compartes ni un gintonic con nosotros. :)                                    |
|Buenos dias para todos. Feliz inicio de semana. :-) http://t.co/svMgEcaxLr                                          |
|@pepedom @bquintero Gracias! No es así, deja claro que es el 100% de aquí http://t.co/cD3VFu7hnH }:)                |
+--------------------------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

Figura 33. Texto del conjunto de datos para creación del modelo de ML

4.3.2.1. Remover nombres de usuarios

Se define la función removerUsuarios, que aceptará un parámetro del tipo columna, y cada línea de texto será comparado con un conjunto de expresiones regulares. En caso de que la comparación sea exitosa, se removerá la coincidencia, la función se muestra en la Figura 34.

```
def removerUsuarios(column):
    return trim(lower(regexp_replace(column,'@([^ ]+)', ''))).alias('stopped')
```

Figura 34. Función para remover nombres de usuario.

El resultado de aplicar la función al dataframe se puede visualizar en la Figura 35.

```
>>> aprendizajemaquina_df.select("value").show(5,False)
+-----------------------------------------------------------------------------------------------------------------+
|value                                                                                                            |
+-----------------------------------------------------------------------------------------------------------------+
|se imaginan a los chicos agradeciendo por el premio con cara de orgullo?.que bonito :).#mtvhottest one direction|
|eclesiastes4:9-12 ♡siempre, promesa :)  https://t.co/xbrysqa43t                                                  |
|qué sabor!o, pj. ya no compartes ni un gintonic con nosotros. :)                                                 |
|buenos dias para todos. feliz inicio de semana. :-) http://t.co/svmgecaxlr                                       |
|gracias! no es así, deja claro que es el 100% de aquí http://t.co/cd3vfu7hnh }:)                                 |
+-----------------------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

Figura 35. Resultado de aplicar la función para remover usuarios.

4.3.2.2. Remover direcciones HTTP

Se define la función removerDirecciones para que tome un parámetro del tipo columna, lo compare con la expresión regular que caracteriza las direcciones HTTP y se lo elimine. La función usada se puede revisar en la Figura 36.

```
def removerDirecciones(column):
    return trim(lower(regexp_replace(column,'https?://([^ ]+)', ''))).alias('stopped')
```

Figura 36. Función para remover direcciones de correo.

Una vez aplicada la función removerDirecciones sobre la columna value del dataframe, se eliminan direcciones HTTP o HTTPS adjuntas en los comentarios, como se puede observar en la Figura 37.

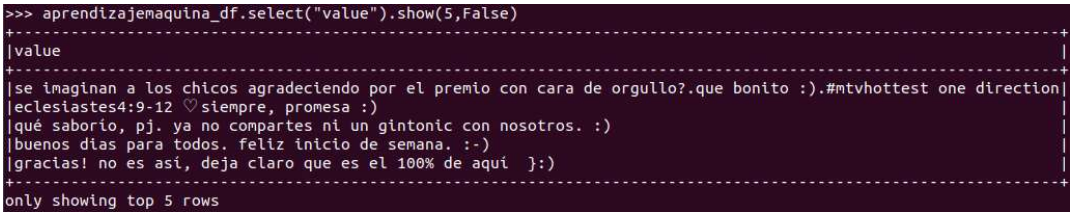


Figura 37. Resultado de la función para remover direcciones de correo.

4.3.2.3. Remover signos de puntuación y números

Se define la función removerPuntuacionNumeros, que tomará un parámetro del tipo columna, los comparará con la expresión regular que caracteriza a letras mayúsculas y minúsculas, incluyendo aquellas vocales puntuadas (á,é,í,ó,ú). La función se describe en la Figura 38.

```
def removerPuntuacionNumeros(column):
    return trim(lower(regexp_replace(column,'[^\s\p{L}]', ''))).alias('stopped')
```

Figura 38. Función para remover signos de puntuación y números.

En caso afirmativo mantendrá el texto, caso contrario removerá el carácter. Una vez aplicada la función para la columna que contiene el texto en el dataframe, se obtiene el resultado presentado en la Figura 39.

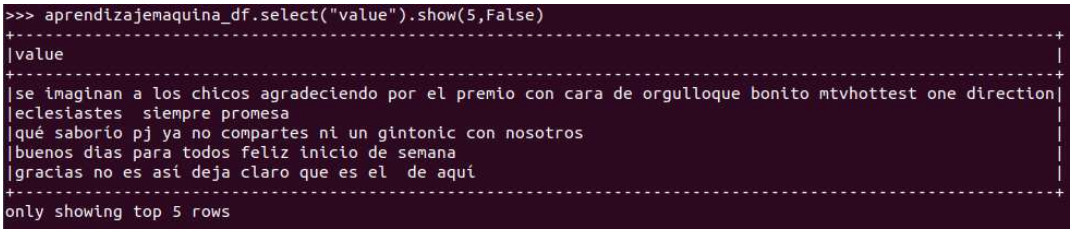


Figura 39. Resultado de aplicar la función para remover signos y números.

4.3.2.4. Tokenización

La tokenización del dataframe consiste en separar una oración en palabras. El resultado de aplicar tokenización al dataframe se muestra en la Figura 40, en la

que los valores del proceso de tokenización se han almacenado en la columna palabras, como una lista de elementos del tipo *String*.

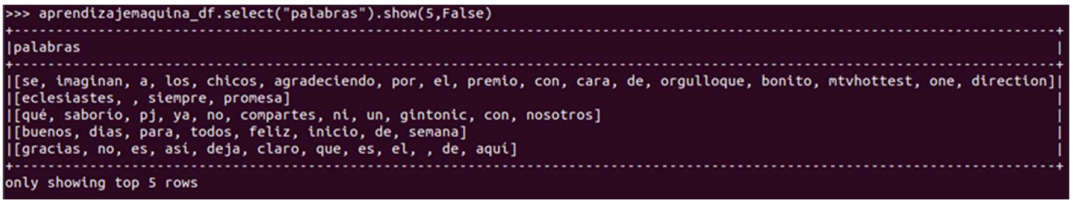


Figura 40. Resultado de tokenización del dataframe de comentarios.

4.3.2.5. Filtrado de conectores

Los conectores son palabras en el idioma español que permiten estructurar una oración, dándole mayor coherencia, pero se repiten con frecuencia y aportan poco significado. Por lo tanto, se eliminan para ahorrar recursos de procesamiento al analizar el sentido de un comentario.

El filtrado de conectores se logra creando una lista con las palabras consideradas como conectores para el idioma español, luego se especifica una columna de entrada (palabras) y una columna de para los resultados del filtrado del texto (filtro_conectores). Esta operación se realiza por medio de la función Tokenizer, incluida en la librería de machine learning de PySpark, disponible en la Figura 41.

```
from pyspark.ml.feature import Tokenizer
tokenizer = Tokenizer(inputCol="value", outputCol="palabras")
aprendizajemaquina_df = tokenizer.transform(aprendizajemaquina_df)
```

Figura 41. Función para separar las palabras de cada comentario.

En la Figura 42 se evidencia el filtrado de conectores del dataframe de tokens, luego de comparar con una lista de los conectores para el idioma español.

```
>>> aprendizajemaquina_df.select("filtro_conectores").show(5,False)
+----------------------------------------------------------------------------------------------+
|filtro_conectores                                                                             |
+----------------------------------------------------------------------------------------------+
|[imaginan, chicos, agradeciendo, premio, cara, orgulloque, bonito, mtvhottest, one, direction]|
|[eclesiastes, , siempre, promesa]                                                             |
|[saborio, pj, compartes, gintonic]                                                            |
|[buenos, dias, feliz, inicio, semana]                                                         |
|[gracias, así, deja, claro, , aquí]                                                           |
+----------------------------------------------------------------------------------------------+
only showing top 5 rows
```

Figura 42. Filtrado de conectores en texto.

4.3.3. Creación y entrenamiento del modelo de ML

Una vez que se ha completado el proceso de limpieza de los comentarios de redes sociales, se procede a crear y entrenar el modelo de ML, un procedimiento que involucra cuatro pasos: vectorización, segmentación de datos, entrenamiento y evaluación.

El proceso comienza con la transformación de palabras a vectores espaciales, donde a cada conjunto de palabras que representan un comentario, le corresponde un vector espacial.

El segundo paso consiste en segmentar los datos del dataframe de comentarios, una parte para entrenamiento del modelo de aprendizaje y el resto para evaluar dicho modelo.

En el tercer paso se aplica el algoritmo de aprendizaje de máquina (regresión logística) para el entrenamiento del modelo.

El cuarto paso es evaluar el modelo de aprendizaje de máquina, respecto a la sección del dataframe separada en el paso dos.

4.3.3.1. Transformación de palabras a vectores espaciales

La transformación se realiza a través de Word2vec, un conjunto de modelos utilizados en aprendizaje de máquina para reconstruir el contexto lingüístico de un conjunto de palabras en un documento. De manera específica este modelo realiza un mapeo de un dato tipo String a un dato tipo Vector.

Los parámetros usados para este modelo son cuatro El parámetro vectorSize define el número de dimensiones que tendrá el vector, en este caso 100 dimensiones. El parámetro minCount especifica el número mínimo de veces que la palabra debe repetirse en el comentario para ser incluida en el vector, al asignarle el valor 0 se le indica al modelo incluir cualquier palabra que forme parte del comentario, incluso si esta no se repite.

El parámetro inputCol indica el nombre de la columna de entrada, para este caso filtro_conectores que posee los comentarios limpios y en forma de tokens (separados por palabras). El parámetro outputCol indica el nombre de la columna en la que se almacenarán los vectores de salida.

Los parámetros que recibe Word2Vec para la vectorización se pueden apreciar en la Figura 43.

```
from pyspark.ml.feature import Word2Vec
from pyspark.ml import Pipeline

w2v = Word2Vec(vectorSize=100, minCount=0, inputCol="filtro_conectores", outputCol="vectores")
aprendizajemaquina_word2vec_modelo = w2v.fit(aprendizajemaquina_df)
aprendizajemaquina_df = aprendizajemaquina_word2vec_modelo.transform(aprendizajemaquina_df)
```

Figura 43. Mapeo de palabras a vectores a través de Word2Vec.

El resultado del mapeo de datos de texto a vectores se aprecia en la Figura 44. Se ha realizado una consulta del dataframe de trabajo, para los campos que especifican la categoría, el campo que mantiene los tokens limpiados y la columna que almacena los vectores.

```
>>> aprendizajemaquina_df.select("categoria","filtro_conectores","vectores").or
derBy(rand()).show(5)
+---------+--------------------+--------------------+
|categoria|   filtro_conectores|            vectores|
+---------+--------------------+--------------------+
|     malo|         [odio, ser]|[-0.0430519245564...|
|    bueno|[gracias, seguirm...|[0.51103863386171...|
|    bueno|    [buena, noticia]|[0.00282658869400...|
|     malo| [puf, pues, mierda]|[-0.0102022693026...|
|     malo|         [asi, mona]|[-1.4075729995965...|
+---------+--------------------+--------------------+
only showing top 5 rows
```

Figura 44. Resultado del mapeo de palabras a vectores.

4.3.3.2. Segmentación de datos

La segmentación de datos consiste en asignar una parte de los datos para entrenamiento del modelo de ML, y el restante para evaluación de la precisión del modelo. En la Figura 45 se especifica un 80% de la data para entrenamiento y 20% para evaluación.

```
aprendizajemaquina_entrenamiento_df, aprendizajemaquina_evaluacion_df = aprendizajemaquina_df.randomSplit([0.8, 0.2])
```

Figura 45. Segmentación de datos para entrenamiento y evaluación.

4.3.3.3. Aplicación de algoritmo de clasificación

Para el entrenamiento del modelo se opta por el algoritmo de regresión logística. La función LogisticRegression es la encargada de crear el clasificador del modelo de regresión logística. Se definen cinco parámetros para la función, uno que define el tipo de clasificación, dos columnas de entrada que toman datos de entrada existentes, y dos columnas encargadas de almacenar datos de salida.

El parámetro family define el tipo de clasificación, ya sea binomial o multinomial. Las columnas de entrada son dos: labelCol, que contiene los datos de etiqueta y featuresCol que corresponde la columna de vectores. Las columnas de salida son: predictionCol, que almacena el resultado de la predicción del algoritmo y probabilityCol la cual indica la probabilidad de que el resultado predicho sea correcto.

La función StringIndexer se encarga de mapear una columna de etiquetas (columna de entrada) a una columna de índices (columna de salida), en este caso mapea la columna existente (categoría) a una nueva columna (etiqueta_ml). La correspondencia entre etiqueta e índice es de uno a uno, con valores enteros que empiezan en cero, hasta el número de etiquetas distintas de la columna.

Debido a que se han definido dos categorías: bueno y malo, existirán dos valores para la columna etiqueta_ml, donde a la categoría bueno le corresponde el índice 1 y a la categoría malo el índice 0, como se observa en la Figura *46*.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import StringIndexer

si = StringIndexer(inputCol="categoria", outputCol="etiquetas_ml")
clasificador_rl = LogisticRegression(family="multinomial", featuresCol="vectores"\
, labelCol="etiquetas_ml",predictionCol="prediccion",probabilityCol="probabilidad")
clasificador_rl_pipeline = Pipeline(stages=[si,clasificador_rl])
modelo_entrenado_rl = clasificador_rl_pipeline.fit(aprendizajemaquina_entrenamiento_df)\
.transform(aprendizajemaquina_evaluacion_df)
```

Figura 46. Código de entrenamiento del modelo de sentimiento.

Posteriormente se crea el pipeline de clasificación clasificador_rl_pipeline. Un pipeline es una secuencia de fases, donde una fase puede ser un estimador o un transformador. El estimador se asocia a la función fit, y se encarga de adaptar los datos de entrada al modelo de ML. El transformador puede invocar la función transform que produce el conjunto de datos para las siguientes fases. Un PipelineModel es una combinación de modelos adaptados y transformadores.

Recapitulando el proceso de entramiento del modelo, la primera fase es la indexación (a cada categoría asignarle un entero), adaptación del modelo de acuerdo con los índices (que el modelo clasifique como comentarios buenos o malos) y finalmente la transformación, en la que se produce un nuevo dataframe, con el esquema del dataframe aprendizajemaquina_df, más las columnas de predicción y probabilidad del modelo, como se puede observar en la Figura 47.

```
>>> modelo_entrenado_rl.printSchema()
root
 |-- value: string (nullable = true)
 |-- categoria: string (nullable = false)
 |-- palabras: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- filtro_conectores: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- vectores: vector (nullable = true)
 |-- etiquetas_ml: double (nullable = false)
 |-- rawPrediction: vector (nullable = true)
 |-- probabilidad: vector (nullable = true)
 |-- prediccion: double (nullable = false)
```

Figura 47. Esquema del dataframe de modelo entrenado.

4.3.3.4. Evaluación del modelo

La evaluación del modelo se hace en base a las predicciones realizadas. El parámetro de evaluación del modelo es la precisión, un porcentaje que representa el número de predicciones correctas respecto a la categorización original del conjunto de dato.

Se crea el evaluador de clasificación evaluador_modelo_rl, en el que se especifica la columna de etiquetas, la columna de predicción y el nombre de la métrica, para este caso accuracy, como se puede ver en el Figura 48.

```
evaluador_modelo_rl= MulticlassClassificationEvaluator(
    labelCol="etiquetas_ml", predictionCol="prediccion", metricName="accuracy")

accuracy = evaluador_modelo_rl.evaluate(modelo_entrenado_rl)
print("Precision = %g" % (accuracy))
```

Figura 48. Código para la evaluación del modelo de ML.

Una vez evaluado el modelo se obtiene el porcentaje de precisión del modelo, para este caso particular del modelo de regresión logística se determina un 66.67% de precisión respecto a las predicciones, como se observa en la Figura 49.

```
>>> print("Precision = %g" % (accuracy))
Precision = 0.666667
```

Figura 49. Precisión del modelo de ML para análisis de sentimiento.

4.4. Proceso de predicción de sentimiento

Para realizar la predicción del sentimiento, se crea un nuevo dataframe denominado redesSociales_df, por medio de la operación de unión entre el dataframe con los comentarios de Facebook y el dataframe con los comentarios de Twitter. En la Figura 50 se observa la creación de los dataframe, la operación de unión, además de que se eliminan los elementos duplicados.

```
facebook_df = spark.read.format('json').load("/home/ml/tesis/archivo8.json")
facebook_df = facebook_df.select("text")
twitter_df = spark.read.format('json').load("/home/ml/tesis/archivo7.json")
twitter_df = twitter_df.select("text")
redesSociales_df = facebook_df.union(twitter_df)
redesSociales_df.count()
redesSociales_df = redesSociales_df.dropDuplicates()
redesSociales_df.select("text").show(5,False)
```

Figura 50. Creación del dataframe de redes sociales.

Se realiza limpieza de texto, con las funciones definidas anteriormente: remover usuarios, remover hashtag, remover direcciones, remover puntuación y números, eliminar backslash (saltos de línea), además se eliminan los registros con valor Null, por medio de la función na.drop() aplicada al dataframe de redes sociales.

La detección de idioma se realiza a través del modelo de aprendizaje de máquina detección_idioma.bin entrenado con anterioridad. Se crea un archivo Python clasificador_idioma.py en el directorio raíz de PySpark, y en el archivo se define una UDF predecir_idioma para predecir el idioma del texto.

La función definida por el usuario carga la librería fasttext, además el modelo de detección de idioma detección_idioma.bin entrenado, recibe un parámetro del tipo columna, realiza la predicción de idioma a través de la función predict() y

devuelve un String del tipo _label_xxx, donde las x representan el código de idioma. En la Figura 51 se observa el contenido del archivo clasificador_idioma.py.

```
import fasttext
model = fasttext.load_model("deteccion_idioma.bin")
def predecir_idioma(msg):
    pred = model.predict(msg)[0][0]
    return pred
```

Figura 51. Archivo externo de Python para la carga del modelo.

Para realizar la llamada al archivo externo de Python es necesario incluir el archivo en el contexto actual de Spark sc, de igual manera se incluye el archivo del modelo de detección de idioma en el contexto. Una vez incluidos ambos archivos se puede realizar la llamada a la UDF udf_predecir_idioma, enviándole la columna texto localmente, como se observa en la Figura 52.

```
from pyspark.sql.functions import col, udf
import clasificador_idioma
udf_predecir_idioma = udf(clasificador_idioma.predecir_idioma)
sc.addFile('deteccion_idioma.bin')
sc.addPyFile('clasificador_idioma.py')

redesSociales_df = redesSociales_df.withColumn('predicted_lang',
                                               udf_predecir_idioma(col('texto')))
```

Figura 52. Registro y llamado de la UDF para predicción de idioma.

4.4.1. Adaptación del modelo de análisis de sentimiento

Para utilizar el modelo de predicción del sentimiento del comentario entrenado con anterioridad, se debe procesar el texto del dataframe de redes sociales. Se reutilizan tres pasos definidos con anterioridad: tokenización, eliminación de conectores lingüísticos y de forma posterior la vectorización

En la Figura 53 se definen los pasos para vectorización de los comentarios del dataframe de redes sociales, recibe como entrada la columna filtro_conectores

(lista de palabras procesadas) y devuelve una columna de vectores correspondiente a las palabras.

```
w2v = Word2Vec(vectorSize=100, minCount=0, inputCol="filtro_conectores", outputCol="vectores")
redesSociales_word2vec_modelo = w2v.fit(redesSociales_df)
redesSociales_df = redesSociales_word2vec_modelo.transform(redesSociales_df)
```

Figura 53. Vectorización del dataframe de redes sociales.

Se crea el modelo de análisis de sentimiento por medio de la función fit(), adaptándolo a los datos de entrenamiento recopilados con anterioridad. La predicción de sentimiento se realiza aplicando el modelo de análisis de sentimiento sobre el dataframe redesSociales_df.

La Figura 54 muestra el proceso de adaptación del modelo al dataframe de entrenamiento (análisis de sentimiento) y la creación del dataframe que contiene la predicción del modelo.

```
modelo_sentimiento = clasificador_rl_pipeline.fit(aprendizajemaquina_entrenamiento_df)
prediccion_sentimiento_redesSociales_df = modelo_sentimiento.transform(redesSociales_df)
```

Figura 54. Adaptación del modelo de análisis de sentimiento al dataframe.

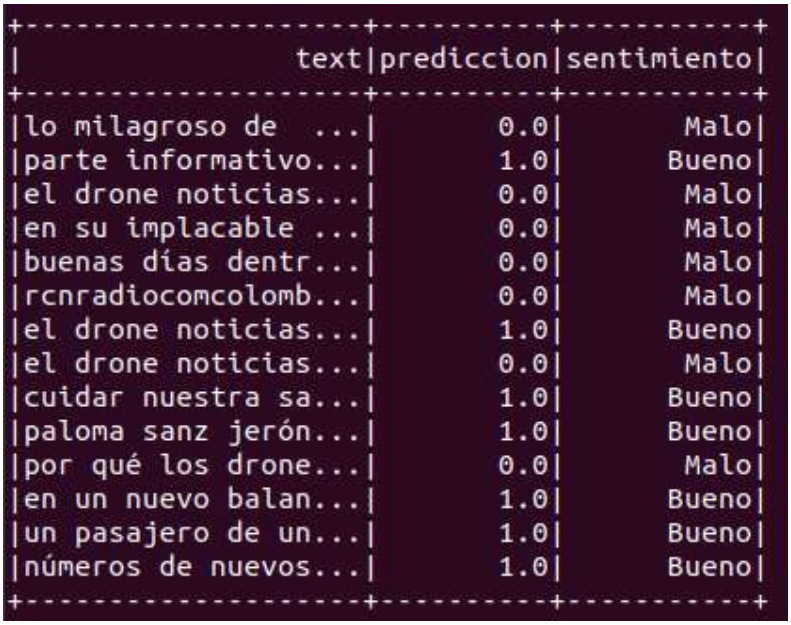
El dataframe prediccion_sentimiento_redesSociales_df es generado para almacenar la predicción de sentimiento aplicada al dataframe con los comentarios de Twitter y Facebook. Sin embargo, cabe destacar que la predicción realizada es un valor del tipo Double.

Se define la función prediccionLiteral, para transformar estos valores numéricos a palabras que representen el sentimiento del comentario. Se registra dicha función como UDF y se aplica sobre la columna prediccion que almacena la predicción numérica, devolviendo una nueva columna denominada sentimiento que almacena la predicción literal, la programación se define en la Figura 55 y el resultado en la Figura 56.

```
def prediccionLiteral(column):
    if column == 1.0:
        return "Bueno"
    else:
        return "Malo"


prediccionLiteral_udf = udf(prediccionLiteral)
prediccion_sentimiento_redesSociales_df=prediccion_sentimiento_redesSociales_df\
    .withColumn('sentimiento',prediccionLiteral_udf(prediccion_sentimiento_redesSociales_df.prediccion))
```

Figura 55. Transformación de predicciones numéricas a literales.



text	prediccion	sentimiento
lo milagroso de ...	0.0	Malo
parte informativo...	1.0	Bueno
el drone noticias...	0.0	Malo
en su implacable ...	0.0	Malo
buenas días dentr...	0.0	Malo
rcnradiocomcolomb...	0.0	Malo
el drone noticias...	1.0	Bueno
el drone noticias...	0.0	Malo
cuidar nuestra sa...	1.0	Bueno
paloma sanz jerón...	1.0	Bueno
por qué los drone...	0.0	Malo
en un nuevo balan...	1.0	Bueno
un pasajero de un...	1.0	Bueno
números de nuevos...	1.0	Bueno

Figura 56. Resultado de la trasformación a predicciones literales.

El dataframe prediccion_sentimiento_redesSociales_df almacena los comentarios de redes sociales, con sus respectivas predicciones de sentimiento. En la Figura 57 ,se definen las operaciones necesarias para mostrar los resultados porcentuales en el terminal, y los resultados se visualizan en la Figura 58.

```
buenos = prediccion_sentimiento_redesSociales_df\
    .where(prediccion_sentimiento_redesSociales_df.sentimiento == "Bueno").count()
malos = prediccion_sentimiento_redesSociales_df\
    .where(prediccion_sentimiento_redesSociales_df.sentimiento == "Malo").count()
total =prediccion_sentimiento_redesSociales_df.count()
porcentajebuenos = int(buenos * 100.00 /total)
porcentajemalos = 100 -porcentajebuenos
print("Buenos = " + str(porcentajebuenos) + "%"\
    , "Malos = "+str(porcentajemalos)+"%", "Total de comentarios ="+str(total))
```

Figura 57. Programación para cálculo porcentual de predicciones.

```
('Buenos = 50%', 'Malos = 50%', 'Total de comentarios =14')
```

Figura 58. Total de comentarios y resultados porcentuales de la predicción.

4.4.2. Integración de componentes

Se integran los tres componentes existentes durante la implementación del framework: código BASH para extracción de datos, comando en terminal para separación de resultados en archivo JSON de redes sociales y código PYTHON para lectura de comentarios, predicción de sentimiento e idioma y contabilización de resultados.

Para la integración se utiliza Apache Zeppelin, una herramienta para análisis de datos y colaboración en línea, por medio de libretas de trabajo, con integración directa para Apache Spark.

Existe un problema de compatibilidad debido que las herramientas de scraping (facebook-scraper y twitter-scraper) utilizan Python v3.6.10, mientras la última versión estable de PySpark trabaja sobre Python v2.7.17. Por tal razón se instala la herramienta conda, que permite administrar y manejar entornos virtuales de Python.

Conda permite agregar el entorno datosredes en el que se seleccionará Python v3.6.10, y se utilizará el entorno por defecto base con la versión v2.7.17. También cabe destacar que facebook-scraper y twitter-scraper no están disponibles para

el gestor de paquetes conda, por lo que se instalarán por medio del gestor pip, y estarán disponibles al activar el ambiente datosredes.

Para el ingreso de la palabra clave de búsqueda, se utiliza la variable global de Zeppelin z, que permite crear la entrada (cuadro de texto) y que la información almacenada esté disponible entre párrafos de la libreta de trabajo.

En la Figura 59 se puede observar cómo distintas versiones de Python (v3.6.10 para scraping y v2.7.17 para Spark) pueden convivir en la misma libreta de trabajo, y se puede mezclar con comandos directos al Shell de Linux.

Figura 59. Párrafos de la libreta de Apache Zeppelin.

En el script de Python se definen dos resultados que se muestran directamente en la interfaz gráfica. La variable z utiliza el contexto Zeppelin para la gráfica y presentación de resultados. Por medio del uso del contexto de Zeppelin una tabla adquiere de forma automática un formato legible y ordenado, además permite insertar de forma automática gráficos en base a los datos disponibles.

En la Figura 60 se presenta el dataframe con 4 columnas: el comentario original, la predicción de idioma, el comentario luego del proceso de limpieza y la predicción de sentimiento. Luego, se agrupa el dataframe en base a la columna de sentimientos, y la contabilización del número de registros.

```
z.show(prediccion_sentimiento_redesSociales_df.select(prediccion_sentimiento_redesSociales_df.text\
    , prediccion_sentimiento_redesSociales_df.idioma_predicho\
    , prediccion_sentimiento_redesSociales_df.filtro_conectores\
    , prediccion_sentimiento_redesSociales_df.sentimiento))

pastel_df = prediccion_sentimiento_redesSociales_df.groupby(prediccion_sentimiento_redesSociales_df
    .sentimiento).count()
z.show(pastel_df)
```

Figura 60. Consulta de datos para el dataframe de redes sociales.

Las libretas de Zeppelin permiten separar el ambiente de colaboración, en el que se muestra el código y los resultados de ejecución del código, de un ambiente de reporte que oculta la programación y muestra únicamente los resultados. El ambiente de reporte es el usado para mostrar los resultados del análisis de sentimiento en comentarios de redes sociales, como se muestra en la Figura 61.

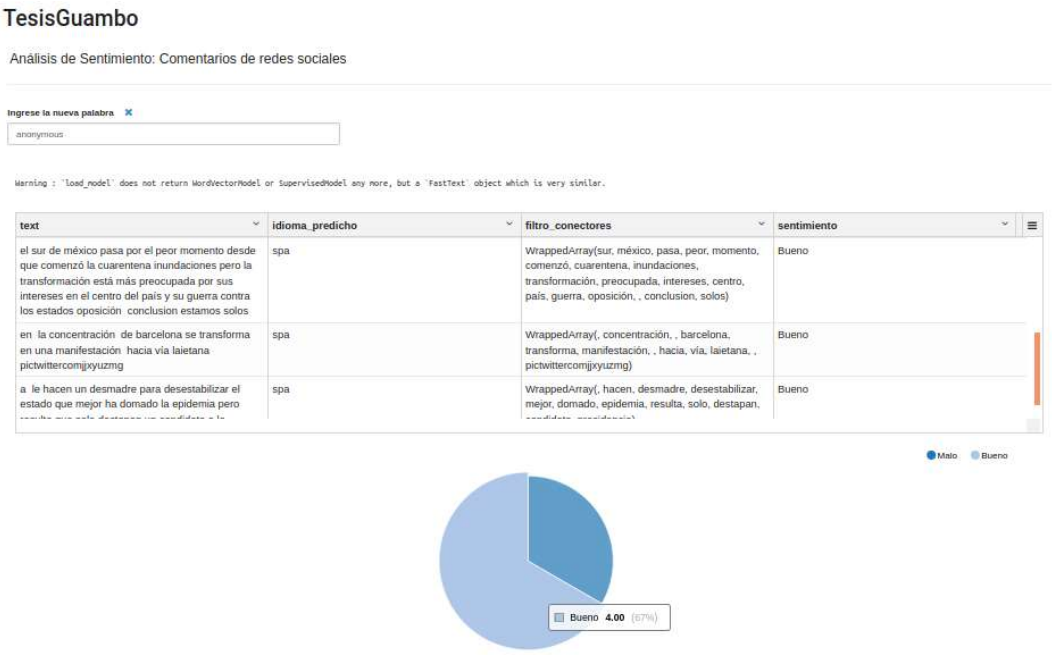


Figura 61. Interfaz gráfica del framework de análisis de sentimientos.

La presentación al usuario se mejora por medio del uso de párrafos embebidos de la Apache Zeppelin, por medio de llamados a la REST API de Zeppelin. Se ocultan los comandos de control de la libreta web, mejorando la transparencia del usuario final. El llamado se realiza a los párrafos de la libreta Web que

contienen: el cuadro de texto para entrada de datos y los resultados del análisis de sentimiento.

De manera adicional se logra personalizar el contenido de la aplicación web, por medio de HTML y hojas de estilo. La interfaz se puede observar en la Figura 62, y una imagen de mayor resolución se en el Anexo 2.

Figura 62. Interfaz gráfica embebida en una página web.

5. PRUEBAS Y ANÁLISIS DE RESULTADOS

En el capítulo anterior sobre la implementación, se realizaron pruebas unitarias para cada componente (scraping, procesamiento de texto, evaluación de modelo y generación de gráficos) de forma paralela a la implementación.

El propósito de este capítulo es realizar la prueba de integración de los componentes individuales del framework de análisis de sentimiento, se analizan fallos y posibles causas, y se plantean soluciones para implementación en futuras versiones del framework.

5.1. Prueba de integración

La prueba de integración permite validar el correcto funcionamiento entre componentes del framework del análisis de sentimiento, y tiene dos objetivos principales: verificar la funcionalidad del framework e identificar posibles fallos (bugs).

La verificación de la funcionalidad permite determinar si los requerimientos encontrados a través de las historias de usuario fueron satisfechos. La identificación de bugs permite detectar fallos en la implementación del sistema, que pueden ser producto de la incompatibilidad entre secciones de código.

Basada en la arquitectura general del framework, descrita en el capítulo de diseño, se busca probar la integración de los siguientes componentes:

- Módulos de scraping integrados con Apache Spark.
- Módulos de scraping integrado con Apache Zeppelin.
- Apache Spark integrado con Apache Zeppelin.

Para verificar las integraciones mencionadas, se ingresa una palabra clave en el framework y se anotan los resultados obtenidos y esperados, como se menciona en la Tabla 1.

Tabla 1

Prueba de integración del framework de análisis de sentimiento.

Número de prueba	1
Objetivo	Validar la implementación de las funcionalidades descritas en las historias de usuario, en el framework de análisis de sentimiento.
Palabra clave	UDLA
Resultado esperado	1. Comentarios mostrados. 2. Idioma detectado. 3. Sentimiento detectado. 4. Gráfico estadístico del sentimiento. 5. Tabla de resumen de resultados.
Resultado obtenido	1. Se muestran los comentarios. 2. Se detecta el idioma. 3. Se detecta el sentimiento. 4. Se muestra el gráfico estadístico del sentimiento. 5. Se tabula el resumen de resultados.
Bugs	Ninguno
Captura de pantalla	La evidencia se muestra en la Figura 63

La implementación finalizada se observa plasmada en la Figura 63, donde se ha realizada el análisis de sentimiento para la palabra clave definida en la prueba de integración.

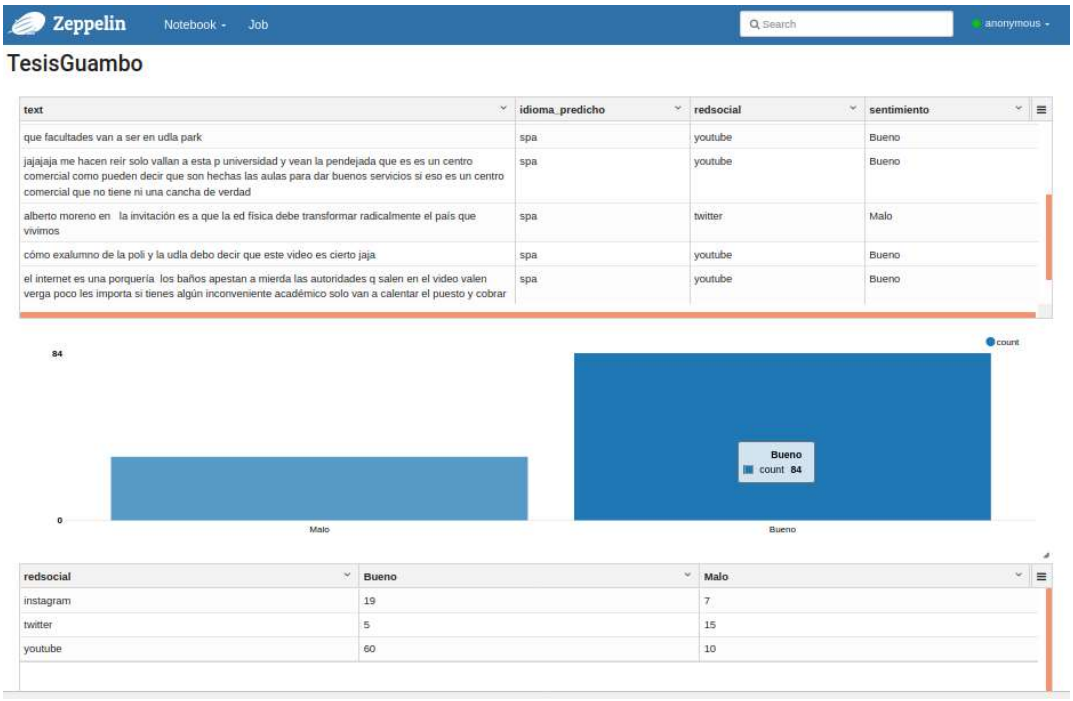


Figura 63. Prueba de integración del framework de análisis de sentimiento.

5.2. Análisis de resultados

Para realizar el análisis de resultados, se evalúan y discuten los resultados obtenidos en la prueba de integración desde dos perspectivas: comportamiento funcional y comportamiento de la interfaz.

La evaluación de comportamiento funcional se enfoca en el punto de vista del desarrollador y la evaluación desde el punto de vista del usuario final. De manera posterior se valida el éxito de la implementación, al contrastar el funcionamiento del framework con las historias de usuario.

5.2.1. Evaluación de comportamiento funcional

Para analizar el comportamiento funcional del framework, se asume el rol de desarrollador y se evalúa la implementación de las funcionalidades del framework. Los resultados se muestran en la Tabla 2.

Tabla 2

Resultados de evaluación del comportamiento funcional

Parámetro de evaluación	¿Se implementó?
Extracción de comentarios	✔
Detección de idioma	✔
Predicción de sentimiento	✔
Agrupación de comentarios por tipo de sentimiento	✔
Agrupación de comentarios por red social de origen	✔

Una vez realizada la prueba de integración se obtienen resultados exitosos. El framework implementa las funcionalidades descritas en las historias de usuario del capítulo de Diseño.

El framework realiza la extracción de comentarios de tres de las cuatro redes sociales planteadas. Los módulos de scraping para Twitter, YouTube e Instagram funcionan correctamente, sin embargo, el módulo de scraping de Facebook no extrae comentarios.

Esto se debe a que es la red social más privativa de las seleccionadas, y para la palabra clave ingresada UDLA, no existen páginas públicas disponibles. En este caso se puede observar que no existe problemas de interacción de los scrapers con el framework de análisis de sentimiento, sino una limitación en cuanto a la funcionalidad de la librería facebook-scraper utilizada.

La detección del idioma se implementa correctamente, categorizando de manera adecuada todos los comentarios, este resultado es atribuido principalmente a la buena calidad del conjunto de datos de entrenamiento.

La predicción de sentimiento muestra fallas. Este resultado es esperable, debido a que el algoritmo de clasificación utilizado mostro una precisión del 66,66%, sin embargo, es una característica notoria desde el punto de vista del usuario final.

Existen varias causas por la que se produce esta clasificación incorrecta, por ejemplo, el overfitting, en el que el modelo de predicción trabaja correctamente con los datos de entrenamiento, y con información semejante, pero tiene problemas para realizar predicción sobre nuevos datos. Este se produce porque el dataset de entrenamiento utilizado corresponde a información de una sola red social (tweets), y aun cuando comparte el idioma designado en las otras redes sociales, los usuarios entre distintas redes sociales utilizan diferente forma de expresión.

De los comentarios extraídos se observó que el nivel de formalidad variaba entre distintas redes sociales, siendo Twitter la red social con el nivel más alto de formalismo, caracterizado porque los usuarios utilizaban palabras completas, buena sintaxis y evitaban las obscenidades.

La longitud del texto entre redes sociales fue otra distinción. Para Twitter que representa el conjunto de datos de entrenamiento, el máximo correspondía a 280 caracteres, mientras Instagram permitía 300 caracteres, Facebook 8000 y YouTube no tenía un límite establecido.

Esta diferencia de longitud y formalismo del texto usado en el entrenamiento con el texto evaluado pueden influir en la capacidad de realizar predicciones acertadas sobre información nueva.

En cuanto a la agrupación de comentarios respecto a tipo de sentimiento y red social de origen no se presentaron inconvenientes, y se pudo observar que el

número de comentarios extraídos por los módulos de scraping variaba de manera considerable.

De manera particular, para la red social YouTube, se pudo notar en las pruebas realizadas, que las palabras ingresadas que tocaban temas antiguos o se referían a marcas con mayor tiempo en el mercado que otras, generaban mayor número de comentarios.

Se puede asumir que esto se debe a que es más demorado elaborar y publicar un video sobre determinado tema, que realizar una publicación de manera instantánea en las otras redes sociales.

5.2.2. Evaluación de la interfaz gráfica

En la evaluación de la interfaz gráfica, se analiza la implementación de las secciones descritas en la interfaz propuesta en la fase de diseño, y se enfoca en la aceptación del usuario final. Los parámetros evaluados se observan en la Tabla 3.

Tabla 3

Resultados de la evaluación de la interfaz gráfica

Parámetro de evaluación	¿Se implementó?
Cuadro de texto para entrada de datos	✔
Tabla de registros de comentarios con cuatro columnas: comentario, idioma, sentimiento y red social	✔
Gráfico de barras con resultados por tipo de sentimiento	✔
Tabla de resumen de resultados por red social de origen	✔

Se puede observar que todos los parámetros evaluados cumplen con el objetivo de transmitir la información al usuario final. En el caso del cuadro de texto, se presenta un solo cuadro en la interfaz, para simplificar el proceso de búsqueda y el número de componentes con los que interactúa el usuario final. Sin embargo,

el diseño es plano, y se puede incorporar ejemplos dentro del cuadro de texto que faciliten la búsqueda del usuario.

La tabla de registros cumple con el objetivo de mostrar los comentarios, predicciones y origen, pero tiene el inconveniente al momento de trabajar con comentarios largos. Esto provoca casos en los que se muestre un solo registro en la pantalla, y el usuario deba deslizarse por medio de la barra de navegación para acceder a los otros resultados.

El tamaño fijo de la lista desplegada es otra limitante de esta sección. Incrementar el tamaño ayudaría a que el usuario final pueda observar un mayor número de comentarios. Debido a que Zeppelin tiene un límite de despliegue de información de 102 400 bytes, se optó por definir el número máximo de registros como 10.

En caso de que la empresa contratante requiera mayor número de comentarios mostrados, se puede optar por eliminar la restricción (no recomendado por el fabricante), o buscar otras alternativas de visualización en lugar de Zeppelin, como puede ser Kibanas.

El gráfico de barras cumple con el objetivo de entregar información visual al usuario final, pero es un requisito que el usuario acerque el cursor sobre el gráfico para visualizar el detalle del número de comentarios. Cambiar esta limitación, y permitir que el número de comentarios por sentimiento se muestre de forma automática sin necesidad de mover el cursor puede mejorar la experiencia del usuario.

Zeppelin presenta otra limitante, pues no permite cambiar el tipo de grafico (barras, pastel, tendencia, tabla) cuando la libreta web se encuentra en modo reporte. Esto reduce la experiencia del usuario final, que debe acceder al modo por defecto para realizar el cambio.

La tabla de resumen entrega los datos de los comentarios por la red social de origen, y sufre del mismo problema de la sección anterior, y es que Zeppelin no permite el cambio desde una tabla hacia un gráfico en el modo de reporte.

5.2.3. Evaluación de la implementación

Este tipo de evaluación busca validar el éxito de la implementación en general. Para realizarlo se contrastan las historias de usuario de la fase de diseño, con los resultados de la prueba de integración, y los resultados se muestran en la Tabla 4.

Tabla 4

Resultados de la evaluación de la implementación.

Requerimiento	¿Se implementó?
Historia de usuario Nro.1	✔
Historia de usuario Nro. 2	✔

6. CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

El framework Apache Spark simplifica las tareas relacionadas con el análisis de datos, como es la lectura de distintitos tipos de archivo a través de la inferencia automática de esquema y el entrenamiento de modelos de aprendizajes de máquina, además se encuentra disponible en varios lenguajes de programación, haciendo accesibles los conceptos de big data, aprendizaje de máquina y análisis de datos al alcance a usuarios sin un conocimiento técnico profundo.

Mediante la investigación se determinó la necesidad de usar herramientas de scraping para retirar las limitantes de número de consultas permitidas, registro y costo que poseen las API oficiales de las redes sociales. También se exploraron las ideas fundamentales de Apache Spark y el aprendizaje de máquina.

La implementación de un framework de análisis de sentimiento mediante el uso de herramientas de software libre y gratuitas fue posible. Las empresas que implementen esta solución reducirán costos en la parte de licenciamiento y libertad de uso para destinar el análisis de sentimiento a cualquier tipo de actividad.

El proceso de limpieza de texto removió las palabras que aportaban poco significado al sentimiento del comentario, se realizó a través de expresiones regulares, e involucró las siguientes actividades: remover nombres de usuario, hashtags, direcciones URL, signos de puntuación, números, saltos de líneas y conectores lingüísticos.

El algoritmo de clasificación seleccionado fue el de regresión logística, por ser el más simple de implementar de los cuatro analizados. Este algoritmo permitió clasificación multietiqueta para la predicción de idioma en 128 categorías con una precisión del 97,1%, y la predicción de sentimiento en 2 categorías con una precisión del 66,7%.

La integración de los distintos lenguajes de programación y unidades de procesamiento del framework de análisis de sentimiento se realizó a través de una libreta web de Apache Zeppelin, mediante llamadas a distintos intérpretes, permitiendo un desarrollo y testing del framework de forma modular.

La presentación de datos se realizó por medio de párrafos embebidos en una página web, y consultas a la REST API de la libreta web. Se logró mejorar la presentación para el usuario final, ocultando los comandos de control de la libreta web.

La extracción de comentarios fue satisfactoria para tres de las cuatro redes sociales seleccionadas: YouTube, Twitter e Instagram. La extracción desde Facebook presento errores y se controló por medio de una excepción.

6.2. Recomendaciones

La búsqueda de comentarios en el framework desarrollado se limita a una palabra clave, o conjunto de palabras sin espacios. Futuros trabajos pueden enfocarse en la extracción de comentarios en base a varias palabras claves u oraciones, en específico el procesamiento para segmentar el campo de entrada en varias etiquetas.

En escenarios en los que las herramientas de scraping existentes no se ajusten a la función requerida para extraer datos, se puede optar por combinar herramientas de automatización web con las librerías existentes, como se realizó en el desarrollo del framework al combinar Selenium con la librería de descarga de comentarios de YouTube.

Para casos en los cuales no exista un conjunto de datos para análisis de sentimiento, el desarrollador puede considerar la opción de generar el dataset por medio de herramientas de automatización. Esto debido a que los conjuntos de datos disponibles en idioma español son limitados, de calidad variable, e inciden de forma directa en la precisión del modelo, como fue el caso del modelo de predicción de sentimiento.

El uso de Apache Zeppelin y otras libretas puede ser considerado en proyectos que requieran desarrollo y testing modular, además de aplicaciones en las que no se puede definir un lenguaje de programación homogéneo para todos los módulos. También simplifica la visualización de datos por medio del uso del modo reporte, o la exportación de datos a través de enlaces y consultas a la REST API de la libreta web.

En entornos de producción, se debe evaluar la necesidad de contratar paquetes de acceso a las API oficiales, en especial si el análisis de sentimiento se realiza sobre contenido de la red social Facebook, que resultó ser la más restrictiva de las cuatro redes sociales.

REFERENCIAS

Apache Project. (s/f). Apache Zeppelin 0.8.0 Documentation: Explore Apache Zeppelin UI. Recuperado el 25 de junio de 2020, de https://zeppelin.apache.org/docs/0.8.0/quickstart/explore_ui.html

Apache Zeppelin. (s/f). Apache Zeppelin 0.8.2 Documentation: Interpreter in Apache Zeppelin. Recuperado el 24 de junio de 2020, de http://zeppelin.apache.org/docs/latest/usage/interpreter/overview.html

Batrinca, B., & Treleaven, P. C. (2014). Social media analytics: a survey of techniques, tools and platforms. AI and Society, 30(1), 89–116. https://doi.org/10.1007/s00146-014-0549-4

Big Data - Demo - Stratebi. (s/f). Recuperado el 17 de diciembre de 2019, de http://bigdata.stratebi.com/real-time/index.htm;jsessionid=E4E0887186CFC92ED7E3A9C5E124B089.bd1

Bouman, E. (s/f). YouTube-comment-downloader. Recuperado el 16 de junio de 2020, de https://github.com/egbertbouman/youtube-comment-downloader

Chambers, B., & Zaharia, M. (2018). Spark: The Definitive Guide Big Data Processing Made Simple (1ra ed.). O'Reilly Media, Inc.

Facebook for Developers, (s/f). Información general - API Graph - Documentación - Facebook para desarrolladores. Recuperado el 30 de abril de 2020, de https://developers.facebook.com/docs/graph-api/overview/

Facebook for Developers (s/f). Limitaciones de frecuencia - API Graph. .Recuperado el 30 de abril de 2020, de https://developers.facebook.com/docs/graph-api/overview/rate-

limiting

Gandomi, A., & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management, 35(2), 137–144. https://doi.org/10.1016/j.ijinfomgt.2014.10.007

Garnacho, D. (s/f). garnachod/TwitterSentimentDataset: Dataset de sentimientos en español. Recuperado el 25 de junio de 2020, de https://github.com/garnachod/TwitterSentimentDataset

Google Developers, (s/f). YouTube Data API Overview | Google Developers. Recuperado el 25 de junio de 2020, de https://developers.google.com/youtube/v3/getting-started

Instagram. (s/f). Instagram - Search & Explore. Recuperado el 25 de junio de 2020, de https://about.instagram.com/features/search-and-explore

Isguzar, B. (2020). Twitter-scraper · PyPI. (s/f). Recuperado el 24 de junio de 2020, de https://pypi.org/project/twitter-scraper/

Kemp, S. (2019). Digital 2019: Ecuador — DataReportal – Global Digital Insights. https://datareportal.com/reports/digital-2019-ecuador

Khan, M. L. (2017). Social media engagement: What motivates user participation and consumption on YouTube? Computers in Human Behavior, 66, 236–247. https://doi.org/10.1016/j.chb.2016.09.024

Lee, K. (2019). 27 Free and Paid Social Media Analytics for Marketers. https://buffer.com/library/social-media-analytics-tools/

Mendelevitch, O., Stella, C., & Eadline, D. (2017). Practical Data Science with Hadoop and Spark: Designing and Building Effective Analytics at Scale. Pearson Education, Inc.

Mitchell, R. (2018). Web Scraping with Python. En Web Scraping with Python, (2da ed.). https://doi.org/10.1017/CBO9781107415324.004

Nord, A. (s/f). InstaTouch. Recuperado el 24 de junio de 2020, de https://github.com/drawrowfly/instagram-scraper

RFC 5849 - The OAuth 1.0 Protocol. (s/f). Recuperado el 30 de abril de 2020, de https://tools.ietf.org/html/rfc5849

Russell, M., & Klassen, M. (2019). Mining the Social Web (3ra ed.). O'Reilly Media, Inc.

Schneider, R. D., & Karmiol, J. (2019). Spark 2nd IBM Limited Edition. http://www.wiley.com/go/permissions.

Sheldon, P., & Bryant, K. (2016). Instagram: Motives for its use and relationship to narcissism and contextual age. Computers in Human Behavior, 58, 89–97. https://doi.org/10.1016/j.chb.2015.12.059

Software Freedom Conservancy, (s/f). The Selenium Browser Automation Project :: Documentation for Selenium. Recuperado el 24 de junio de 2020, de https://www.selenium.dev/documentation/en/

Tatoeba. (s/f). Download sentences - Tatoeba. Recuperado el 21 de mayo de 2020, de https://tatoeba.org/eng/downloads

Twitter Inc., (s/f). API reference index — Twitter Developers. Recuperado el 30 de abril de 2020, de https://developer.twitter.com/en/docs/api-reference-index

Wu, J. (2019). Web Scraping Using Python: A Step By Step Guide. Recuperado el 26 de junio de 2020, de https://www.octoparse.com/blog/web-scraping-using-python

Zuñiga, K. (2020). Facebook-scraper - PyPI.Recuperado el 24 de junio de 2020, de https://pypi.org/project/facebook-scraper/

ANEXOS

Anexo 1. Script original de la herramienta de descarga de comentarios de YouTube.

```
#!/usr/bin/env
Python

                from __future__ import print_function

                import io
                import json
                import os
                import sys
                import time

                import argparse
                import lxml.html
                import requests
                from lxml.cssselect import CSSSelector

                YOUTUBE_VIDEO_URL = 'https://www.youtube.com/watch?v={youtube_id}'
                YOUTUBE_COMMENTS_AJAX_URL_OLD =
                'https://www.youtube.com/comment_ajax'
                YOUTUBE_COMMENTS_AJAX_URL_NEW =
                'https://www.youtube.com/comment_service_ajax'

                USER_AGENT = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
                (KHTML, like Gecko) Chrome/79.0.3945.130 Safari/537.36'


                def find_value(html, key, num_chars=2, separator=''):
                    pos_begin = html.find(key) + len(key) + num_chars
                    pos_end = html.find(separator, pos_begin)
                    return html[pos_begin: pos_end]


                def ajax_request(session, url, params=None, data=None, headers=None, retries=5,
                sleep=20):
                    for _ in range(retries):
                        response = session.post(url, params=params, data=data, headers=headers)
                        if response.status_code == 200:
                            return response.json()
                        if response.status_code in [403, 413]:
                            return {}
                        else:
                            time.sleep(sleep)
```

```
def download_comments(youtube_id, sleep=.1):
    if r'\isLiveContent\:true' in
requests.get(YOUTUBE_VIDEO_URL.format(youtube_id=youtube_id)).text:
        print('Live stream detected! Not all comments may be downloaded.')
        return download_comments_new_api(youtube_id, sleep)
    return download_comments_old_api(youtube_id, sleep)


def download_comments_new_api(youtube_id, sleep=1):
    # Use the new youtube API to download some comments
    session = requests.Session()
    session.headers['User-Agent'] = USER_AGENT

    response = session.get(YOUTUBE_VIDEO_URL.format(youtube_id=youtube_id))
    html = response.text
    session_token = find_value(html, 'XSRF_TOKEN', 3)

    data = json.loads(find_value(html, 'window[ytInitialData] = ', 0, '\n').rstrip(';'))
    for renderer in search_dict(data, 'itemSectionRenderer'):
        ncd = next(search_dict(renderer, 'nextContinuationData'), None)
        if ncd:
            Break
    continuations = [(ncd['continuation'], ncd['clickTrackingParams'])]

    while continuations:
        continuation, itct = continuations.pop()
        response = ajax_request(session, YOUTUBE_COMMENTS_AJAX_URL_NEW,
                                params={'action_get_comments': 1,
                                        'pbj': 1,
                                        'ctoken': continuation,
                                        'continuation': continuation,
                                        'itct': itct},
                                data={'session_token': session_token},
                                headers={'X-YouTube-Client-Name': '1',
                                         'X-YouTube-Client-Version': '2.20200207.03.01'})

        if not response:
            Break
        if list(search_dict(response, 'externalErrorMessage')):
            raise RuntimeError('Error returned from server: ' + next(search_dict(response,
'externalErrorMessage')))
```

```
        # Ordering matters. The newest continuations should go first.
        continuations = [(ncd['continuation'], ncd['clickTrackingParams'])
                         for ncd in search_dict(response, 'nextContinuationData')] +
continuations

        for comment in search_dict(response, 'commentRenderer'):
            yield {'cid': comment['commentId'],
                   'text': ''.join([c['text'] for c in comment['contentText']['runs']]),
                   'time': comment['publishedTimeText']['runs'][0]['text'],
                   'author': comment.get('authorText', {}).get('simpleText', ''),
                   'channel': comment['authorEndpoint']['browseEndpoint']['browseId'],
                   'votes': comment.get('voteCount', {}).get('simpleText', '0'),
                   'photo': comment['authorThumbnail']['thumbnails'][-1]['url']}

        time.sleep(sleep)


def search_dict(partial, key):
    if isinstance(partial, dict):
        for k, v in partial.items():
            if k == key:
                yield v
            else:
                for o in search_dict(v, key):
                    yield o
    elif isinstance(partial, list):
        for i in partial:
            for o in search_dict(i, key):
                yield o


def download_comments_old_api(youtube_id, sleep=1):
    # Use the old youtube API to download all comments (does not work for live
streams)
    session = requests.Session()
    session.headers['User-Agent'] = USER_AGENT

    # Get YouTube page with initial comments
    response = session.get(YOUTUBE_VIDEO_URL.format(youtube_id=youtube_id))
    html = response.text

    reply_cids = extract_reply_cids(html)

    ret_cids = []
```

```
    for comment in extract_comments(html):
        ret_cids.append(comment['cid'])
        yield comment

    page_token = find_value(html, 'data-token')
    session_token = find_value(html, 'XSRF_TOKEN', 3)

    first_iteration = True

    # Get remaining comments (the same as pressing the 'Show more' button)
    while page_token:
        data = {'video_id': youtube_id,
                'session_token': session_token}

        params = {'action_load_comments': 1,
                  'order_by_time': True,
                  'filter': youtube_id}

        if first_iteration:
            params['order_menu'] = True
        else:
            data['page_token'] = page_token

        response = ajax_request(session, YOUTUBE_COMMENTS_AJAX_URL_OLD,
params, data)
        if not response:
            break

        page_token, html = response.get('page_token', None), response['html_content']

        reply_cids += extract_reply_cids(html)
        for comment in extract_comments(html):
            if comment['cid'] not in ret_cids:
                ret_cids.append(comment['cid'])
                yield comment

        first_iteration = False
        time.sleep(sleep)

    # Get replies (the same as pressing the 'View all X replies' link)
    for cid in reply_cids:
        data = {'comment_id': cid,
                'video_id': youtube_id,
                'can_reply': 1,
```

```
                'session_token': session_token}

        params = {'action_load_replies': 1,
                  'order_by_time': True,
                  'filter': youtube_id,
                  'tab': 'inbox'}

        response = ajax_request(session, YOUTUBE_COMMENTS_AJAX_URL_OLD,
params, data)
        if not response:
            Break

        html = response['html_content']

        for comment in extract_comments(html):
            if comment['cid'] not in ret_cids:
                ret_cids.append(comment['cid'])
                yield comment
        time.sleep(sleep)


def extract_comments(html):
    tree = lxml.html.fromstring(html)
    item_sel = CSSSelector('.comment-item')
    text_sel = CSSSelector('.comment-text-content')
    time_sel = CSSSelector('.time')
    author_sel = CSSSelector('.user-name')
    vote_sel = CSSSelector('.like-count.off')
    photo_sel = CSSSelector('.user-photo')

    for item in item_sel(tree):
        yield {'cid': item.get('data-cid'),
               'text': text_sel(item)[0].text_content(),
               'time': time_sel(item)[0].text_content().strip(),
               'author': author_sel(item)[0].text_content(),
               'channel': item[0].get('href').replace('/channel/',").strip(),
               'votes': vote_sel(item)[0].text_content() if len(vote_sel(item)) > 0 else 0,
               'photo': photo_sel(item)[0].get('src')}


def extract_reply_cids(html):
    tree = lxml.html.fromstring(html)
    sel = CSSSelector('.comment-replies-header > .load-comments')
    return [i.get('data-cid') for i in sel(tree)]
```

```
def main(argv):
    parser = argparse.ArgumentParser(add_help=False, description=('Download
YouTube comments without using the YouTube API'))
    parser.add_argument('--help', '-h', action='help', default=argparse.SUPPRESS,
help='Show this help message and exit')
    parser.add_argument('--youtubeid', '-y', help='ID of YouTube video for which to
download the comments')
    parser.add_argument('--output', '-o', help='Output filename (output format is line
delimited JSON)')
    parser.add_argument('--limit', '-l', type=int, help='Limit the number of comments')

    try:
        args = parser.parse_args(argv)

        youtube_id = args.youtubeid
        output = args.output
        limit = args.limit

        if not youtube_id or not output:
            parser.print_usage()
            raise ValueError('you need to specify a YouTube ID and an output filename')

        if os.sep in output:
            outdir = os.path.dirname(output)
            if not os.path.exists(outdir):
                os.makedirs(outdir)

        print('Downloading YouTube comments for video:', youtube_id)
        count = 0
        with io.open(output, 'w', encoding='utf8') as fp:
            sys.stdout.write('Downloaded %d comment(s)\r' % count)
            sys.stdout.flush()
            start_time = time.time()
            for comment in download_comments(youtube_id):
                comment_json = json.dumps(comment, ensure_ascii=False)
                print(comment_json.decode('utf-8') if isinstance(comment_json, bytes) else
comment_json, file=fp)
                count += 1
                sys.stdout.write('Downloaded %d comment(s)\r' % count)
                sys.stdout.flush()
                if limit and count >= limit:
                    break
```

```
        print('\n[{:.2f} seconds] Done!'.format(time.time() - start_time))

    except Exception as e:
        print('Error:', str(e))
        sys.exit(1)


if __name__ == __main__:
    main(sys.argv[1:])
```

Anexo 2. Interfaz gráfica del framework del análisis de sentimiento, por medio del uso de párrafos embebidos, HTML y hojas de estilo.