Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2023.

Analizador de sentimientos - Twitter

202200007 – Natalia Mariel Calderón Echeverría

Resumen

El presente proyecto se concentró en al análisis, procesamiento y reconocimiento de sentimiento de mensajes emitidos en la red social twitter, con este objetivo se desarrollo una herramienta capaz de analizar dicho contenido, tanto el sentimiento como los hashtags involucrados y los users mencionados.

La empresa Tecnologías Chapinas, SA requeria de una herramienta capaz de analizar los sentimientos de los usuarios, los hashtags utilizados y los users mencionados, esto con el objetivo de tener un mejor entendimiento de la opinión sobre su marca dentro de dicha red social. Para poder brindar un mejor análisis de los sentimientos se crea un reporte en formato xml que le permite a la empresa conocer la cantidad de mensajes positivos, negativos y neutros que recibió durante una fecha en específico. La solución planteada toma en cuenta estos detalles y hace uso de la programación orientada a objetos (POO) y de los frameworks: Django y Flask para construir el frontend y backend necesario para levantar el proyecto.

Palabras clave

Twitter, users, hashtags, endpoints, frameworks

Abstract

The present project focused on the analysis, processing, and sentiment recognition of messages posted on the social network Twitter. With this goal in mind, a tool was developed capable of analyzing such content, including both the sentiment, the involved hashtags, and the mentioned users.

The company Tecnologías Chapinas, SA required a tool capable of analyzing user sentiments, used hashtags, and mentioned users, with the aim of gaining a better understanding of the opinion about their brand within the mentioned social network, Twitter. To provide a more comprehensive sentiment analysis, a report is generated in XML format, allowing the company to understand the quantity of positive, negative, and neutral messages received during a specific date. The proposed solution takes these details into account and makes use of object-oriented programming (OOP) and frameworks such as Django and Flask to build the necessary frontend and backend required for the project.

Keywords

Twitter, users, hashtags, endpoints, frameworks

Introducción

La comprensión de los sentimientos expresados en los mensajes ha sido abordada mediante un enfoque que implica el reconocimiento de palabras clave dentro del texto. Por consiguiente, se ha llevado a cabo una cuidadosa consideración sobre la adquisición de la base de datos para su análisis. En este sentido, se ha propuesto la implementación de un diccionario dinámico de términos que pueda ser cargado en el programa, permitiendo así una flexibilidad que posibilite la adición y eliminación de palabras clave según las necesidades específicas de la empresa y los criterios de análisis deseados.

Este diccionario nació debido a la necesidad del mismo a ser flexible y poder agregar y eliminar palabras claves según las necesidades de la empresa y los criterios que desea aplicar. Basados en esta base de datos y en la estrategia mencionada, se ha planteado la estructuración logística del análisis, incluyendo la identificación de usuarios y hashtags relevantes en el proceso.

El criterio para identificar usuarios y hashtags se ha definido rigurosamente, minimizando cualquier posibilidad de ambigüedad durante el análisis de los mensajes. Este enfoque ha requerido un grado de precisión técnica, garantizando la inclusión exclusiva de aquellos usuarios y hashtags que cumplen con los requisitos específicos. En el caso de los hashtags, se ha establecido que estos se encuentren delimitados por el símbolo numeral, mientras que en el caso de los usuarios, se ha especificado que deben comenzar con un símbolo de arroba, seguido de caracteres alfanuméricos. Cabe destacar que aquellos usuarios que no cumplan con estos requisitos, como por ejemplo, la presencia de espacios en su identificación, no serán considerados como tales en el análisis.

Los sentimientos de los mensajes se llevo a cabo a través del conteo de las palabras positivas y negativas dentro de un mismo mensaje, es decir, si un mensaje cuenta con mas palabras positivas que negativas este será catalogado como un mensaje positivo. Esto también se aplica al caso en el que un mensaje contenga mas palabras negativas que positivas, dicho mensaje será catalogado como negativo.

Una vez planteada la lógica y la interacción de las solicitudes con las estructuras de datos y los requisitos empresariales, se procedió al desarrollo del frontend a través de Django. Este paso ha permitido establecer una conexión entre ambos componentes, con el objetivo de facilitar el acceso y la usabilidad del programa para la empresa interesada.

Desarrollo del tema

El enunciado establecido para este proyecto establece la necesidad de poder trabajar a partir de la creación de una herramienta que analice datos, procede dichos datos y devuelva de manera ordenada y sistematizada ciertos datos en base al análisis realizado. Este proyecto también hace necesario la creacion de una herramienta que funcione a través de la creacion de un backend en el framework Flask y un Frontend hecho con ayuda del framework Django, esto nos permite establecer los límites y alcances del proyecto así también la manera en la cual será planteado, es por eso que se dividió en varias sección con objetivo de facilitar el análisis y mejorar la calidad de la herramienta final.

a. Entrada de datos

Esta etapa inicia con la creación de lo que será el esqueleto de nuestro proyecto, es decir la manera en la que los archivos serian leídos y la manera en la que estos serian almacenados.

El archivo se lee mediante el uso de la biblioteca ElementTree de Python, que permite la manipulación de datos XML de una manera amigable y para su

correcta manipulación se crearon las siguientes variables:

- xml_file: Representa el archivo XML cargado.
- xml_string: Contiene la cadena decodificada del archivo XML.
- xml_root: Representa el objeto XML raíz generado a partir de la cadena decodificada.

Luego de esto se presenta la necesidad de almacenar varios mensajes y de establecer los usuarios y hashtags dentro de los mismos, para estos se definieron los siguientes diccionarios y sets.

- **stored_data(diccionario)**: Un diccionario que almacena los mensajes, usuarios y hashtags asociados con fechas específicas.

La clave es la fecha y el valor es otro diccionario que contiene las listas de mensajes, conjuntos de usuarios y conjuntos de hashtags:

- **users**: Un conjunto que almacena los usuarios extraídos de los mensajes.
- **hashtags**: Un conjunto que almacena los hashtags extraídos de los mensajes.

Una vez definidas los elementos a usar, se propone y se escribe en código el siguiente proceso de lectura y almacenamiento. Primero se verificar si el archivo esta presente en la solicitud y si este es valido, es decir tiene extensión XML.

Una vez verificado, el archivo se lee y se codifica como una cadena, y posteriormente se convierte en un objeto XML manejable mediante la función **ET.fromstring** de la biblioteca ElementTree. A partir de este objeto, se extraen elementos clave, como la fecha y el texto del mensaje, así como cualquier usuario o hashtag mencionado en el mensaje. Por ultimo, Los datos se almacenan en una estructura de diccionario denominada **stored_data**. Esta estructura mantiene registros de mensajes, usuarios y hashtags asociados con fechas específicas. Si hay mensajes con la misma fecha, se agrupan bajo la misma clave de fecha, y los usuarios y hashtags se agregan a los conjuntos de usuarios y hashtags respectivamente.

Figura 1. Interfaz Grafica.

Fuente: elaboración propia, a través del proyecto [tkinter]

b. Archivos de salida

Para esta etapa era necesario la correcta creación de dos funciones que nos permitieran manipular la información obtenida de los archivos de entrada para luego analizarla y generar un reporte sobre los datos de interés.

- resumenMensajes
  Para generar este archivo se creo la función: generate_xml(stored_data).

La función realiza lo siguiente. Primero, crea un elemento XML principal llamado 'MENSAJES_RECIBIDOS'. Luego, itera a través de los elementos en el diccionario stored_data y agrega subelementos para cada fecha, que incluyen información sobre la fecha, el número de mensajes recibidos, el número de usuarios mencionados y el número de hashtags incluidos. Por último, utiliza el módulo

xml.dom.minidom para formatear el XML de manera legible y lo guarda en un archivo llamado "resumenMensajes.xml".

resumenMensajes.xml

```xml
<?xml version="1.0" ?>
<MENSAJES_RECIBIDOS>
    <TIEMPO>
        <FECHA>15/01/2023</FECHA>
        <MSJ_RECIBIDOS>4</MSJ_RECIBIDOS>
        <USR_MENCIONADOS>5</USR_MENCIONADOS>
        <HASH_INCLUIDOS>5</HASH_INCLUIDOS>
    </TIEMPO>
    <TIEMPO>
        <FECHA>08/08/2023</FECHA>
        <MSJ_RECIBIDOS>1</MSJ_RECIBIDOS>
        <USR_MENCIONADOS>2</USR_MENCIONADOS>
        <HASH_INCLUIDOS>1</HASH_INCLUIDOS>
    </TIEMPO>
    <TIEMPO>
        <FECHA>02/10/2023</FECHA>
        <MSJ_RECIBIDOS>3</MSJ_RECIBIDOS>
        <USR_MENCIONADOS>5</USR_MENCIONADOS>
        <HASH_INCLUIDOS>4</HASH_INCLUIDOS>
    </TIEMPO>
</MENSAJES_RECIBIDOS>
```

Figura 1. **resumenMensajes**.xml
Fuente: elaboración propia, a través del proyecto

- **resumenConfig**

Para generar este archivo se creo la función:" generate_configxml(pos, neg)".

La función realiza lo siguiente: Primero, crea conjuntos a partir de las palabras positivas y negativas. Luego, calcula la cantidad de palabras positivas, palabras negativas y palabras neutras.

Posteriormente, crea un elemento raíz llamado 'CONFIG_RECIBIDA' y agrega subelementos que muestran las estadísticas de las palabras positivas, las palabras positivas rechazadas, las palabras negativas y las palabras negativas rechazadas.

Por ultimo, utiliza el módulo xml.dom.minidom para formatear el XML de manera legible y lo guarda en un archivo llamado "resumenConfig.xml

resumenConfig.xml

```xml
<?xml version="1.0" ?>
<CONFIG_RECIBIDA>
    <PALABRAS_POSITIVAS>5</PALABRAS_POSITIVAS>
    <PALABRAS_POSITIVAS_RECHAZADA>1</PALABRAS_POSITIVAS_RECHAZADA>
    <PALABRAS_NEGATIVAS>7</PALABRAS_NEGATIVAS>
    <PALABRAS_NEGATIVAS_RECHAZADA>1</PALABRAS_NEGATIVAS_RECHAZADA>
</CONFIG_RECIBIDA>
```

Figura 2. **resumenConfig**.xml
Fuente: elaboración propia, a través del proyecto

c. Endpoints – post / get

En esta etapa fue necesario establecer las peticiones que necesitábamos hacer y la naturaleza de las mismas, es decir definir si se trataba de un Post o un Get.

Para esto es necesario tener claro que el método POST, se utiliza para enviar datos al servidor para crear o actualizar un recurso. Cuando un cliente envía una solicitud POST, generalmente se incluyen datos en el cuerpo de la solicitud.

Por otro lado, el método GET se utiliza para solicitar datos de un recurso específico del servidor. Cuando un cliente envía una solicitud GET, los parámetros se envían en la URL. Tomando esto en cuenta se crearon los siguientes endpoint.

- **/devolverUsuarios**

Este endpoint recupera los usuarios mencionados en una fecha específica y sus recuentos correspondientes, para ello utiliza las siguientes variables y estructuras de datos:

- **date_to_check:** Almacena la fecha proporcionada por el usuario.
- **users:** Lista de usuarios mencionados.
- **user_counts:** Contador de usuarios.

- **/devolverHashtags**

ste endpoint recupera los hashtags mencionados en una fecha específica y sus recuentos correspondientes, para ello utiliza las siguientes variables y estructuras de datos:

- **date_to_check:** Almacena la fecha proporcionada por el usuario.
- **hashtags:** Conjunto de hashtags mencionados.
- **hashtags_counts:** Contador de hashtags.

- **/clasificarMensajes**

  Este endpoint clasifica los mensajes como positivos, negativos o neutrales para una fecha específica, en función de las palabras positivas y negativas predefinidas, para ello utiliza las siguientes variables y estructuras de datos:

  - **date_to_check:** Almacena la fecha proporcionada por el usuario.
  - **cuenta_positivos, cuenta_neg, cuenta_neutral:** Contadores para mensajes positivos, negativos y neutros respectivamente.
  - **pos, neg:** Conjuntos de palabras positivas y negativas respectivamente.
- **/clearData**

  Este endpoint borra todos los datos almacenados y restablece las listas de palabras positivas y negativas, para ello utiliza las siguientes variables y estructuras de datos:

  - **stored_data:** Diccionario que almacena los datos procesados.
  - **pos, neg:** Listas de palabras positivas y negativas respectivamente.

d. Frontend – Django

Esta etapa final del proyecto se basa en la correcta relación entre el Backend y nuestro Frontend. Para el Frontend se siguieron las restricciones y se uso el frameworks Django.

Django, una estructura web de código abierto y de gran nivel, sirve para simplificar el proceso de desarrollo de aplicaciones web complejas y adaptables en Python. Gracias a su diseño organizado y su enfoque en la eficacia y el reciclaje de elementos, Django permite a los programadores crear aplicaciones web de manera ágil y efectiva. Al incluir un conjunto completo de herramientas y funciones integradas, Django facilita la incorporación de funciones avanzadas como la autenticación de usuarios, el manejo de bases de datos y la generación de contenido dinámico.

El uso de Django en el proyecto nos permitió conectar los endpoint con los elementos necesario del proyecto, a la vez que se agregaban dichas conexiones fue necesario usar: HTML, CSS y JavaScript para generar la pagina de inicio amigable al usuario, que es en donde se llevaran a cabo todas las funciones descritas y es la pantalla con la que trabajara el usuario de la empresa. Para esto se le agregaron funcionalidades a cada uno de los botones, y dependiendo de la funcionalidad de cada botón se agrego un textArea en donde se reflejan los resultados obtenidos.

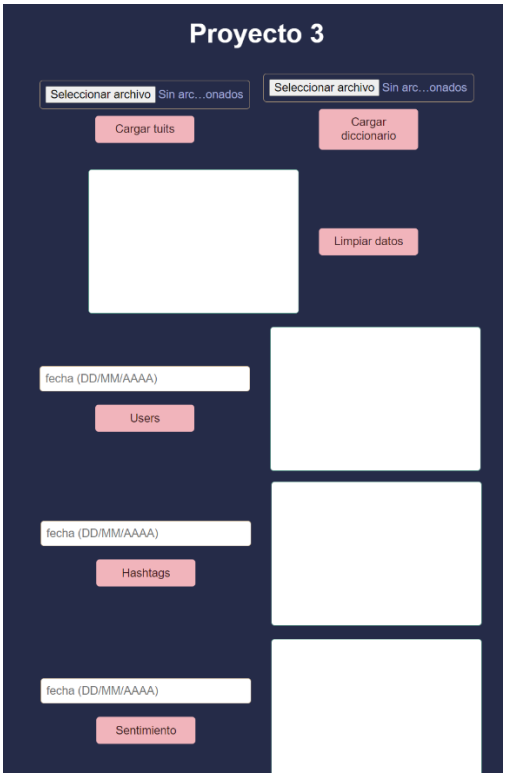


Figura 3. Frontend al iniciar

Fuente: elaboración propia, a través del proyecto

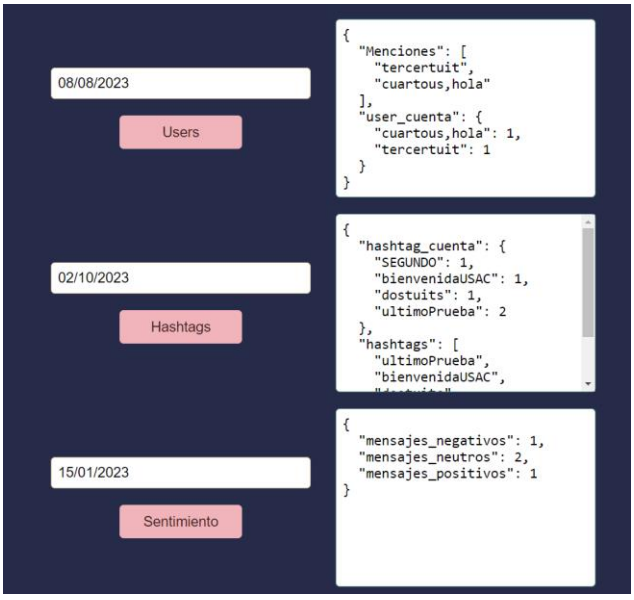


Figura 4. Resultado peticiones

Fuente: elaboración propia, a través del proyecto

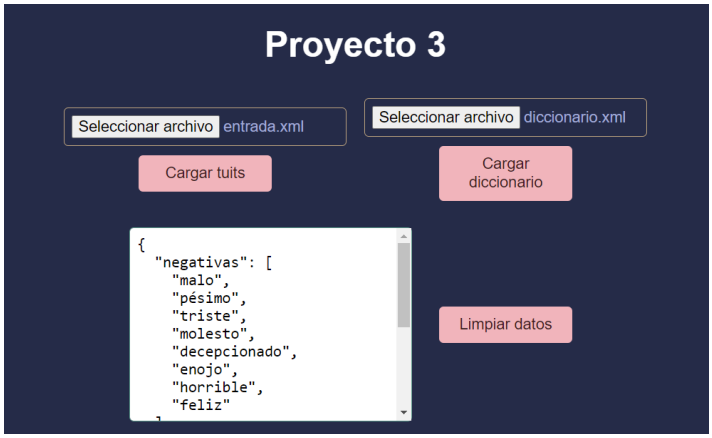


Figura 5. Resultado cargar archivos

Fuente: elaboración propia, a través del proyecto

Conclusiones

A pesar de la complejidad inherente del proyecto, la división de esta herramienta en componentes Frontend y Backend proporciona una ventaja considerable al abordar y comprender el problema en profundidad.

Esta estrategia no solo contribuye al desarrollo lógico y programación del proyecto, sino que también simplifica en gran medida el análisis de sentimientos en mensajes, una tarea que a menudo está llena de ambigüedades y desafíos.

El proceso de análisis de sentimientos en mensajes es reconocido por su naturaleza repleta de matices y ambigüedades. Sin embargo, gracias a la orientación que se ha aplicado a esta herramienta, se ha logrado reducir significativamente la incertidumbre y, al mismo tiempo, se ha permitido identificar con un alto grado de precisión el sentimiento presente en estos mensajes, distinguiendo entre positivo, negativo y neutro.

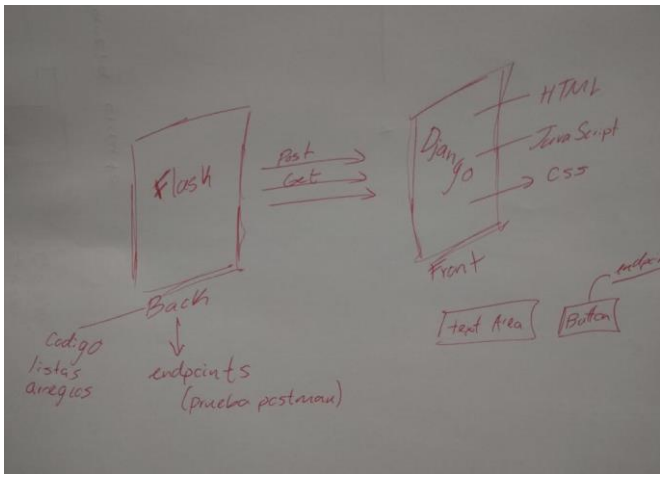
Referencias bibliográficas

Quickstart — Flask Documentation (3.0.X). (s. f.). https://flask.palletsprojects.com/en/3.0.x/quickstart/#a-minimal-application

Django. (s. f.). Django Project. https://docs.djangoproject.com/en/4.2/

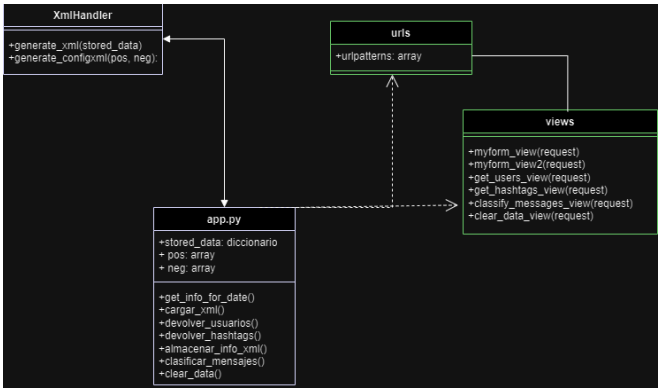
xml.etree.ElementTree — The ElementTree XML API. (s. f.). Python documentation. https://docs.python.org/3/library/xml.etree.elementtree.html

Anexos



Anexo 1. Primer esquema de idea sobre como funciona un backend y un frontend junto

Fuente: elaboración propia.



Anexo 2. Diagrama de Clases

Fuente: elaboración propia.