Analisis de Sentimientos por Texto

Juan David Cotacio Sánchez / Valery Dayana Triana Garcia / Juliana Carolina Avila Sánchez

I. Introducción

En el prototipo ”Creación de un Servicio de Análisis de Sentimientos para Textos Largos” aprendimos a desarrollar un servicio que evalúe sentimientos en textos extensos, con una interfaz gráfica e integración con redes sociales. Usando Python, FastAPI/Django, HTML, CSS, JavaScript y transformers de Hugging Face, creamos un modelo de análisis de sentimientos, una interfaz de usuario y un sistema de filtros de publicación. Al finalizar, seremos capaz de implementar estas herramientas y documentar el proyecto.

A. Como empezar el prototipo

Para crear nuestro prototipo, empezamos por buscar un modelo existente que realizara análisis de emociones en texto. Utilizamos la plataforma Hugging Face, que alberga muchos modelos de procesamiento del lenguaje natural (PLN) para distintas tareas. Una vez que encontramos un modelo adecuado, lo importamos a nuestro entorno de desarrollo en Visual Studio.

En este caso, utilizamos el modelo ”nlptown/bert-base-multilingual-uncased-sentiment”, capaz de realizar análisis de significado multilingüe.

Ademas de añadir una configuración del servidor FastAPI para gestionar las solicitudes de análisis de emociones asi creando nuestro microservicio.

```python
from fastapi import FastAPI, Query, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from transformers import pipeline
from faker import Faker
import random

clasificador = pipeline('sentiment-analysis', model='nlptown/bert-base-multilingual-uncased-sentiment')


app = FastAPI()

```

Figure 1. Implementación del Modelo y FastAPI

B. Como se Implemento el Modelo y FastApi

Importación e inicialización: se importan las bibliotecas necesarias, incluidas FastAPI, transformadores, Faker y random. El clasificador de sentimientos se inicializa utilizando el modelo ”nlptown/bert-base-multilingual-uncased-sentiment”.

Configuración de CORS: CORS está configurado para permitir peticiones de cualquier origen.

Se utiliza Faker para generar nombres de usuario ficticios en español. Se definen comentarios de ejemplo para simular opiniones sobre la universidad.

Mapeo de sentimientos: se crea una función para mapear etiquetas del modelo (1 estrella, 2 estrellas, etc.) a sentimientos en español (triste, decepcionado, etc.).

Generación de comentarios ficticios: se define una función para generar una lista de comentarios ficticios seleccionados aleatoriamente a partir de ejemplos predefinidos.

```python
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


faker = Faker('es_ES')


def mapear_sentimiento(label: str):
    sentimiento_mapping = {
        '1 star': 'triste',
        '2 stars': 'decepcionado',
        '3 stars': 'neutral',
        '4 stars': 'satisfecho',
        '5 stars': 'feliz'
    }
    return sentimiento_mapping.get(label, 'desconocido')


comentarios_ejemplos = [
    "El ambiente de estudio en la Universidad Konrad Lorenz es excelente.",
    "Los profesores son muy atentos y siempre dispuestos a ayudar.",
    "Me encanta la biblioteca, tiene una gran cantidad de recursos.",
    "Las instalaciones deportivas son de primera calidad.",
    "Estoy muy satisfecho con el plan de estudios de mi carrera.",
    "El proceso de matrícula fue muy sencillo y rápido.",
    "Hay muchas oportunidades para participar en proyectos de investigación.",
    "El campus es muy bonito y bien cuidado.",
    "El servicio de cafetería ofrece una buena variedad de alimentos.",
    "Las actividades extracurriculares son muy variadas e interesantes."
]
```

Figure 2. Implemtacion del CORS, Faker, Mapeo de los sentimientos y Inicio del Fake/ Mock

C. Implementación Java Script

Este código JavaScript implementamos en una interfaz de usuario web para interactuar con la API creada en Python utilizando FastAPI. A continuación cómo se relaciona este código JavaScript con el código Python proporcionado. Para nuestra GUI quisimos implementar JS en los botones para que estos al momento de realizar un clic arrojara la información, esto lo logramos creando su respectivo evento al botón, además para conectar nuestro endpoint implementamos una solicitud fetch a la URL de la API de recomendación y comentarios analizados: (recomendacion) y (comentarios analizados), pasando el texto como parámetro de consulta y en la otra función en formato JSON.Para al final mostrar todo en nuestra GUI hecha en HTML Y CSS.

D. Pruebas del Prototipo

Al probar un prototipo desarrollado con FastAPI, JavaScript, Python, CSS, HTML y Hugging Face, es importante com-

```python
def generar_comentarios_mock():
    comentarios = []
    for _ in range(15):
        comentario = random.choice(comentarios_ejemplos)
        usuario = faker.user_name()
        comentarios.append({"usuario": usuario, "comentario": comentario})
    return comentarios


@app.get("/comentarios_analizados")
def obtener_comentarios_analizados():
    comentarios = generar_comentarios_mock()
    comentarios_analizados = []
    for comentario in comentarios:
        resultado = clasificador(comentario['comentario'])
        sentimiento = mapear_sentimiento(resultado[0]['label'])
        comentarios_analizados.append({
            "usuario": comentario['usuario'],
            "comentario": comentario['comentario'],
            "sentimiento": sentimiento,
            "puntaje": resultado[0]['score']
        })
    return comentarios_analizados

@app.get("/recomendacion")
def recomendar_publicacion(texto: str = Query(..., description="Texto para analizar y recomendar")):
    if not isinstance(texto, str):
        raise HTTPException(status_code=400, detail="El input debe ser una cadena de texto.")


    max_tokens = 1000
    if len(texto.split()) > max_tokens:
        raise HTTPException(status_code=400, detail=f"El texto no debe exceder los {max_tokens} tokens.")

    resultado = clasificador(texto)
    puntaje = resultado[0]['score']
    sentimiento = mapear_sentimiento(resultado[0]['label'])


    publicar = sentimiento in ['satisfecho', 'feliz', 'neutral']

    return {
        "texto": texto,
        "sentimiento": sentimiento,
        "puntaje": puntaje,
        "publicar": publicar
    }
```

Figure 3. Finalización del Fake Mock e End Points de la API

```javascript
document.getElementById('analyze-button').addEventListener('click', function() {
    const text = document.getElementById('input-text').value.trim();

    if (!text) {
        alert('Por favor, escribe algún texto para analizar.');
        return;
    }

    const maxTokens = 1000;
    if (text.split(/\s+/).length > maxTokens) {
        alert(`El texto no debe exceder los ${maxTokens} tokens.`);
        return;
    }

    fetch(`http://127.0.0.1:3001/recomendacion?texto=${encodeURIComponent(text)}`)
        .then(response => {
            if (!response.ok) {
                throw new Error('Error en la solicitud');
            }
            return response.json();
        })
        .then(data => {
            if (data.sentimiento && data.puntaje !== undefined && data.publicar !== undefined) {
                document.getElementById('sentiment').innerText = `Sentimi (parameter) data: any }`;
                document.getElementById('score').innerText = `Puntaje: ${data.puntaje}`;
                document.getElementById('publish').innerText = `Recomendación de Publicación: ${data.publicar ? 'Sí' : 'No'}`;
            } else {
                throw new Error('Estructura de datos inesperada');
            }
        })
        .catch(error => {
            console.error('Error:', error);
            alert('Hubo un problema con la solicitud. Por favor, intenta de nuevo.');
        });
});
```

Figure 4. Conexion endpoint recomendacion con JS

probar características clave como la precisión del análisis de sentimientos y la correcta actualización de los comentarios. También deben evaluarse aspectos de rendimiento como la estabilidad del servidor y el tiempo de respuesta bajo carga. Las pruebas de usabilidad son cruciales para garantizar una interfaz de usuario intuitiva y accesible en distintos dispositivos. Mediante pruebas exhaustivas en estos ámbitos, se puede detectar y resolver cualquier problema para garantizar que el prototipo funciona de forma eficaz y satisfactoria para los usuarios.

```javascript
function actualizarComentarios() {
    fetch('http://127.0.0.1:3001/comentarios_analizados')
        .then(response => response.json())
        .then(data => {
            const comentariosDiv = document.getElementById('comentarios');
            comentariosDiv.innerHTML = ''; // Limpiar comentarios anteriores
            data.forEach(comentario => {
                const comentarioElement = document.createElement('div');
                comentarioElement.classList.add('comentario');
                comentarioElement.innerHTML = `
                    <p><strong>Usuario:</strong> ${comentario.usuario}</p>
                    <p><strong>Comentario:</strong> ${comentario.comentario}</p>
                    <p><strong>Sentimiento:</strong> ${comentario.sentimiento}</p>
                    <p><strong>Puntaje:</strong> ${comentario.puntaje.toFixed(2)}</p>
                `;
                comentariosDiv.appendChild(comentarioElement);
            });
        })
        .catch(error => {
            console.error('Error al obtener los comentarios:', error);
        });
}


setInterval(actualizarComentarios, 20000);
document.addEventListener('DOMContentLoaded', actualizarComentarios);

```

Figure 5. Conexion endpoint y facke/mock comentario analizados con JS

E. Vista General del Prototipo

Al centrarnos en una interfaz de usuario intuitiva, los prototipos son accesibles a una amplia gama de usuarios y pueden interactuar cómodamente con ellos usuarios de todos los niveles de destreza. Esta facilidad de uso no sólo aumenta la versatilidad de aplicación en distintos ámbitos, sino que también promueve una experiencia de usuario positiva que facilita la adopción y la utilidad en diversos contextos. Además, el proceso de desarrollo del proyecto no sólo permitió crear aplicaciones funcionales, sino que también brindó la oportunidad de aprender y aplicar nuevos conocimientos, reforzando así sus competencias y preparándoles para futuros retos y oportunidades en el ámbito técnico.

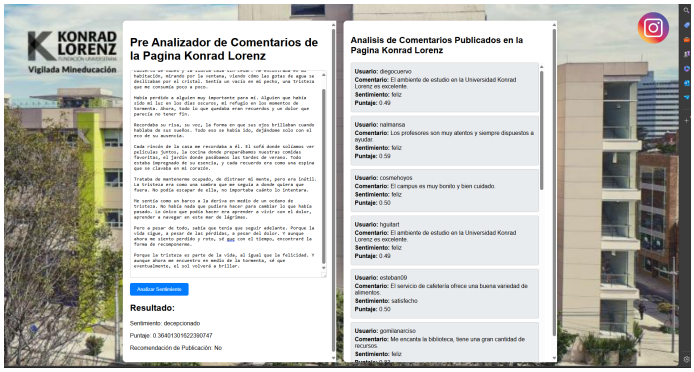

Figure 6. Vista Final Del Prototipo

F. Dificultades

Durante el desarrollo de este proyecto, se encontraron algunas dificultades a la hora de conectar el código JavaScript a los puntos finales de API desarrollados con FastAPI: aunque FastAPI simplifica la creación de API RESTful, la integración con el front-end puede ser difícil, especialmente cuando las solicitudes son asíncronas y el procesamiento de datos

JSON puede resultar complicado. Además, la instalación y configuración de un modelo de análisis de sentimientos en una máquina local puede resultar complicada debido a las dependencias adicionales y a la configuración específica del entorno. A pesar de estos retos, la resolución de problemas ayudó a superar estas dificultades y permitió avanzar en el desarrollo del prototipo.