Inteligencia de Negocios

Juan Nicolás García Esquibel

201717860

Entrega 2 del proyecto

Introducción

La etapa actual se enfoca en la implementación de una aplicación web que permite entrenar y actualizar de manera incremental un modelo de clasificación de textos en español, orientado a identificar el ODS correspondiente a distintas opiniones. Para esto, se desarrolló una API con FastAPI que gestiona operaciones como predicción, reentrenamiento, reinicio y monitoreo del modelo, junto con una interfaz creada en React y Vite que facilita la interacción con el usuario. Esta solución posibilita realizar pruebas, observar métricas e incorporar nuevos datos de forma continua, asegurando un aprendizaje adaptable, eficiente y comprensible para todo tipo de usuario.

Diseño de la aplicación

Al considerar los posibles escenarios de uso, una aplicación web resulta la opción más adecuada, ya que la interacción con el modelo se da principalmente desde un entorno de trabajo en computador, especialmente cuando se requiere analizar o gestionar textos extensos. Aun así, la interfaz se diseñó de forma adaptable, permitiendo su uso en dispositivos móviles si fuera necesario.

Backend

Para el desarrollo del backend se utilizó FastAPI, un framework de Python que ofrece un excelente equilibrio entre simplicidad, rendimiento y compatibilidad con librerías de machine learning. Su enfoque asíncrono permite gestionar múltiples solicitudes de manera eficiente, garantizando tiempos de respuesta bajos y escalabilidad. Además, su integración con Swagger proporciona una documentación automática que facilita tanto las pruebas como la exploración de los diferentes endpoints implementados, como /predict, /retrain, /reset, /health e /info.

Frontend

En la parte del frontend se optó por React, una librería ampliamente utilizada para construir interfaces de usuario modernas, dinámicas y modulares. Su estructura basada en componentes facilita el mantenimiento, la reutilización de código y la integración directa con APIs REST. La aplicación se desarrolló con Vite para aprovechar su rapidez en el entorno de desarrollo y su eficiente proceso de compilación. La paleta visual se adaptó a tonos verdes, buscando transmitir claridad y coherencia con el propósito analítico del proyecto. Finalmente, el diseño

fue pensado para ofrecer una experiencia fluida y adaptable, garantizando que los usuarios puedan interactuar con el modelo de manera intuitiva desde cualquier dispositivo.

Detalles del back

En lo referente al pipeline, se mantiene la estructura general desarrollada en la Etapa 1, preservando los pasos de normalización del texto como la conversión a minúsculas, la eliminación de caracteres no ASCII, la sustitución de números por marcadores, la remoción de puntuación y stopwords en español, y la lematización con spaCy, que permite reducir la variabilidad morfológica del lenguaje. No obstante, en esta versión se introduce una mejora significativa mediante el uso de HashingVectorizer en lugar de TF-IDF, dado que su naturaleza stateless posibilita transformar los textos en vectores de longitud fija sin necesidad de almacenar vocabularios, lo que lo hace ideal para procesos de aprendizaje incremental. Finalmente, el modelo se entrena con un SGDClassifier, que admite actualizaciones progresivas utilizando el método partial_fit, permitiendo que el modelo se adapte a nuevos datos sin requerir un reentrenamiento completo. Esta elección responde al objetivo de lograr un flujo de entrenamiento continuo y eficiente, a diferencia de los algoritmos usados previamente, que necesitaban reiniciar el proceso de aprendizaje en cada actualización.

Detalles del Front

La estructura presenta distintas secciones que permiten una interacción completa con la API y con el modelo de clasificación. La interfaz principal presenta un menú claro y funcional con botones que dirigen al usuario a cada uno de los endpoints disponibles.

La primera sección corresponde a Predicción, donde el usuario puede ingresar una o varias opiniones y obtener como respuesta la categoría ODS asignada junto con la probabilidad asociada. Además, se ofrece un resumen interpretativo que describe de manera amigable el resultado de cada texto analizado, pensado para facilitar la comprensión a usuarios no técnicos.

La segunda sección es Reentrenamiento, que permite subir un archivo CSV con nuevos datos para actualizar el modelo de forma incremental. Tras completar el proceso, la aplicación muestra las métricas globales y por clase, incluyendo precision, recall y F1-score, además de representar los resultados en un formato fácilmente legible. Dentro de esta misma vista se añadió un botón Reiniciar, el cual ejecuta el endpoint /reset y elimina el modelo actual para comenzar nuevamente desde cero, lo que resulta útil durante las pruebas o en caso de errores en el entrenamiento.

La tercera sección corresponde a Salud del servicio, donde se consulta el endpoint /health para verificar el estado general de la API y confirmar si existe un modelo cargado. En caso de que no haya uno disponible, se muestra un mensaje de advertencia en color naranja que sugiere iniciar un nuevo entrenamiento. Finalmente, la sección Información del modelo resume detalles técnicos como la versión del modelo, el estado de spaCy y la disponibilidad de las stopwords, consolidando en un solo espacio toda la información relevante sobre el sistema y su funcionamiento.

Usuario final e importancia de la aplicación

La aplicación desarrollada tiene un valor significativo para organizaciones e instituciones interesadas en analizar y clasificar grandes volúmenes de información textual relacionada con los Objetivos de Desarrollo Sostenible (ODS). Por ejemplo, puede ser empleada por analistas de sostenibilidad o equipos de comunicación que buscan identificar a qué objetivo se asocian las opiniones, comentarios o publicaciones de sus comunidades. Gracias a la clasificación automática y a las probabilidades generadas por el modelo, los usuarios pueden priorizar el análisis de ciertos textos, detectar sesgos o tendencias en la percepción pública y tomar decisiones informadas sobre sus estrategias sociales o ambientales.

De esta forma, la aplicación no solo facilita el procesamiento eficiente de información en entornos donde el volumen de texto es alto, sino que también contribuye a mejorar la trazabilidad y coherencia del impacto social reportado, permitiendo a las organizaciones mantener una visión más clara y cuantificable sobre las temáticas que se relacionan con los distintos ODS.

Riesgos del uso

Aunque la aplicación ofrece un apoyo valioso en la clasificación de textos, es importante reconocer sus limitaciones. El modelo, aun después de múltiples reentrenamientos, puede cometer errores de clasificación, generando falsos positivos o negativos. Esto significa que una opinión podría asociarse de manera incorrecta a un ODS distinto al que realmente corresponde. Por esa razón, los resultados deben interpretarse como una herramienta de apoyo y no como una decisión definitiva, siempre complementados con una revisión humana que valide los casos más ambiguos.

También es fundamental atender aspectos relacionados con la privacidad y seguridad de los datos, ya que los textos procesados pueden contener información sensible o provenir de fuentes privadas. Para mitigar riesgos, se recomienda mantener comunicaciones cifradas, evitar almacenar información innecesaria y gestionar los datos de entrenamiento de forma ética y responsable. Finalmente, a medida que el modelo crezca en tamaño y complejidad, aumentarán los requerimientos de cómputo y almacenamiento, por lo que es necesario planificar recursos, sistemas de respaldo y estrategias de mantenimiento que garanticen la continuidad y estabilidad del servicio.

Despliegue de la aplicación y recursos sugeridos

Para la ejecución y mantenimiento del sistema se requiere un entorno que soporte Python, FastAPI y las librerías de machine learning utilizadas en el modelo. En la mayoría de los

casos, un servidor con entre 2 y 4 núcleos de CPU y 4 a 8 GB de RAM será suficiente para manejar las operaciones de predicción y reentrenamiento de manera eficiente. Sin embargo, si se prevé un flujo constante de texto o reentrenamientos frecuentes, puede ser recomendable implementar una infraestructura escalable en la nube, aprovechando servicios como AWS, Azure o Google Cloud Platform, que permiten aumentar los recursos de forma dinámica según la demanda.

El modelo entrenado se almacena como un archivo binario mediante Joblib, ubicado en el mismo servidor donde corre la API, por lo que es fundamental contar con copias de seguridad periódicas para evitar la pérdida de información o de versiones actualizadas del modelo. En cuanto al frontend, la aplicación desarrollada con React y Vite puede desplegarse en el mismo servidor o en un servicio de hosting independiente, garantizando la correcta configuración de CORS para permitir la comunicación entre ambos componentes.

Para optimizar la portabilidad y facilitar el mantenimiento, una alternativa viable es el uso de contenedores Docker que integren tanto la API como el frontend, asegurando consistencia entre entornos y simplificando futuras implementaciones. Finalmente, la herramienta puede integrarse fácilmente dentro del flujo operativo de la organización, por ejemplo, como parte de un proceso de análisis o validación de texto antes de su publicación, permitiendo automatizar tareas y mejorar la eficiencia en la toma de decisiones relacionadas con la clasificación temática de la información.

Comparación con el modelo de la entrega 1

	precision	recall	f1-score
1	0.93	0.96	0.95
3	0.97	0.97	0.97
4	0.99	0.98	0.99
accuracy			0.97
macro avg	0.96	0.97	0.97
weighted avg	0.97	0.97	0.97

En comparación con el modelo de la primera etapa, los resultados actuales muestran una leve disminución en las métricas globales, lo cual puede explicarse por el uso de un archivo distinto para el entrenamiento, con una composición de datos y una distribución de clases diferentes. Esto influye en la capacidad del modelo para mantener un equilibrio uniforme en sus predicciones. Además, en esta versión se implementó un enfoque incremental basado en HashingVectorizer y el clasificador SGDClassifier, sustituyendo el esquema anterior con TF IDF y modelos entrenados desde cero. Esta nueva estrategia permite reentrenar el modelo sin perder lo aprendido previamente y facilita la adaptación a datos nuevos, aunque también puede generar mayor sensibilidad a la proporción entre clases y a los parámetros de

optimización. En conjunto, el resultado refleja un sistema más flexible y escalable, capaz de mantener un rendimiento competitivo bajo condiciones de actualización continua.

Definiciones de reentrenamiento

Alternativa 1. Reentrenamiento incremental (implementada)

Consiste en actualizar el modelo existente sin reiniciar todo el proceso de aprendizaje. En este enfoque, el modelo conserva los parámetros previamente aprendidos y los ajusta a medida que recibe nuevos datos. En el proyecto se aplicó esta estrategia utilizando la función partial_fit del SGDClassifier, que permite incorporar nueva información sin perder lo ya aprendido. Este método mejora la eficiencia y facilita la adaptación continua del modelo.

Alternativa 2. Reentrenamiento completo tradicional

En esta alternativa se mantienen los mismos algoritmos y técnicas, pero cada actualización implica volver a entrenar el modelo desde cero con todos los datos disponibles, antiguos y nuevos. Aunque asegura una base más uniforme y evita el riesgo de acumulación de sesgos, requiere un alto costo computacional y no permite actualizaciones rápidas o incrementales.

Alternativa 3. Reentrenamiento con transferencia de conocimiento

Esta opción plantea partir de un modelo previamente entrenado, como BERT u otro modelo de lenguaje, y ajustarlo con los datos específicos del proyecto mediante un proceso de fine tuning. La ventaja es que se aprovecha conocimiento previo y se obtiene un modelo más preciso, pero requiere recursos computacionales significativos y experiencia avanzada en modelos de gran escala, por lo que no se implementó en esta etapa.

Estrategia para el uso de IA generativa

Con el objetivo de aumentar y diversificar el conjunto de datos empleado en el entrenamiento del modelo, se exploraron dos estrategias basadas en el uso de modelos generativos de lenguaje, que permiten crear nuevas opiniones relacionadas con los ODS a partir de textos originales.

Alternativa 1. Generación automática mediante script local

En esta primera opción se implementó un script sencillo en Python que recibe un texto como entrada y genera una nueva opinión aplicando estrategias de reformulación y negación. Por ejemplo, si una oración hacía referencia a un impacto positivo en educación, el script producía una versión opuesta, destacando una carencia o problema en el mismo contexto. Este método busca ampliar la variabilidad del dataset manteniendo coherencia semántica y

equilibrio entre clases, aprovechando patrones lingüísticos simples sin depender de herramientas externas.

Alternativa 2. Generación conversacional con ChatGPT

La segunda estrategia consistió en mantener una sesión interactiva con ChatGPT, compartiendo directamente el CSV base y entrenando el modelo conversacional para seguir una de las tres estrategias de aumentación previamente definidas: negación, generalización o detalle contextual. De esta forma, se podían generar múltiples nuevas opiniones para cada clase, ajustando el tono, la extensión y el enfoque según el ODS correspondiente. Esta aproximación ofrece un control más fino sobre la calidad y variedad de las muestras generadas, permitiendo obtener datos más realistas y balanceados para el posterior reentrenamiento del modelo.

Trabajo individual

El desarrollo comenzó con la definición de los frameworks a utilizar, seleccionando FastAPI para el backend y React para el frontend por su compatibilidad, rendimiento y facilidad de integración. Luego, se inició la construcción de la API, diseñando la estructura del pipeline y definiendo la lógica para implementar el reentrenamiento incremental. Durante esta etapa se fueron desarrollando y probando los endpoints de manera individual, aprovechando la documentación automática de Swagger para validar el comportamiento y depurar errores rápidamente. Una vez establecida la base del backend, se desarrolló el frontend y se probaron las interacciones con la API para garantizar una integración fluida y una experiencia de usuario clara. Finalmente, todo el proceso se fue documentando en el presente informe, dejando evidencia de las decisiones técnicas y de diseño adoptadas en cada fase del trabajo.