Joan Sebastian Amaya Bohorquez-202011318

Nicolas Lara Gomez-202012455

David Mateo Barbosa Monsalve-202110756

Link del video con la demostración de la aplicación:

https://youtu.be/zVAIilPn24k

Indicaciones para ejecutar la app:

Para ejecutar la aplicación, primero hay que iniciar el backend (FastAPI) y luego el frontend (React, Vue, etc.). Para ello, abre una terminal y entra a la carpeta apirest, donde está el backend. Instala las dependencias con pip install -r requirements.txt y luego inicia el servidor con uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload. Esto pondrá el backend en funcionamiento en el puerto 8000. Luego, abre otra terminal, ve a la carpeta frontend, instala las dependencias con npm install si es necesario, y ejecuta npm start. Esto hará que el frontend se ejecute en http://localhost:3000 y se conecte al backend en http://localhost:8000.

Sección 1 – Automatización del proceso y exposición vía API REST

1.1. Descripción general del proceso de automatización

En esta etapa del proyecto se implementó un sistema automatizado que permite procesar, entrenar, guardar y consumir un modelo de analítica de texto mediante un servicio web. El enfoque se centró en encapsular todas las tareas de limpieza, transformación y entrenamiento dentro de un pipeline de machine learning reutilizable, persistente y fácil de consumir desde una API REST.

El sistema fue desarrollado desde el enfoque del ingeniero de datos, asegurando que tanto la preparación como el acceso al modelo sean consistentes, reproducibles y escalables. Adicionalmente, se diseñaron herramientas que permiten no solo predecir, sino también reentrenar el modelo de forma controlada, garantizando que el archivo binario (modelo_fake_news.pkl) se actualice para futuras predicciones.

1.2 Implementación del pipeline

Se utilizó la librería scikit-learn para construir un Pipeline que unifica los siguientes pasos:

- Limpieza de texto (eliminación de caracteres no ASCII, puntuación, stopwords, números, etc.)

- Tokenización y lematización utilizando NLTK y spaCy
- Vectorización del texto mediante CountVectorizer, limitado a un vocabulario controlado
- Entrenamiento de un modelo GradientBoostingClassifier, optimizado mediante GridSearchCV
- Empaquetado de todo el flujo en un pipeline serializable con joblib

El pipeline fue entrenado inicialmente sobre un conjunto de noticias clasificadas y posteriormente guardado como un archivo .pkl para su reutilización.

1.3 Persistencia del modelo y almacenamiento incremental del dataset

Como GradientBoostingClassifier no admite reentrenamiento incremental con partial_fit, se diseñó un mecanismo de almacenamiento acumulativo. Cada vez que se recibe un nuevo conjunto de datos para reentrenar el modelo, dichos datos se almacenan de manera permanente mediante el módulo pickle, en un archivo binario dentro del directorio datasets.

Este dataset acumulado se vuelve a cargar completamente al momento del reentrenamiento, lo que permite reconstruir el modelo desde cero con todos los ejemplos históricos más los nuevos. Esta solución permite conservar el conocimiento adquirido sin depender de técnicas como warm_start, y evita las limitaciones técnicas de los clasificadores ensemble.

1.4 Estructura de la API REST

Se diseñó una API REST modular compuesta por dos endpoints principales que interactúan con el pipeline persistido. El formato de entrada y salida se mantiene en JSON, respetando los nombres y estructuras del archivo CSV original proporcionado.

Endpoint 1: /predict

- Recibe una o más instancias de datos en el cuerpo del request, incluyendo los campos titulo y descripcion.
- Cada texto es preprocesado utilizando las mismas funciones del pipeline original.
- El modelo devuelve una lista de predicciones (0 o 1), junto con la probabilidad asociada a cada predicción.
- Las predicciones se devuelven en el mismo orden en el que fueron recibidas.

Endpoint 2: /retrain

- Recibe un conjunto de instancias etiquetadas, que incluyen titulo, descripcion y label.
- Los datos son preprocesados igual que en el entrenamiento inicial.
- Los nuevos ejemplos se agregan al dataset acumulado.
- Se recarga el dataset completo (antiguo + nuevo), se transforma con el vectorizador, y se entrena nuevamente el modelo desde cero.
- El modelo actualizado sobrescribe el archivo modelo_fake_news.pkl.
- Se devuelven métricas de desempeño del nuevo modelo: Precision, Recall y F1-score.

1.5 Estrategias de reentrenamiento consideradas

A continuación se describen tres enfoques posibles para el reentrenamiento del modelo, evaluando sus ventajas y desventajas:

a) Reentrenamiento completo desde cero (estrategia implementada) Descripción: Se acumulan todos los datos utilizados en reentrenamientos previos, y se entrena un nuevo modelo desde cero usando el conjunto total. Ventaja: Se mantiene toda la información histórica y se garantiza la integridad del conocimiento aprendido. Desventaja: El entrenamiento puede tardar más a medida que el conjunto de datos crece.

b) Reentrenamiento incremental usando partial_fit Descripción: Se entrena el modelo solo con los nuevos ejemplos, conservando internamente los parámetros del modelo anterior. Ventaja: Es rápido y escalable, ideal para flujos continuos de datos. Desventaja: No es compatible con modelos ensemble como GradientBoostingClassifier, y requiere clasificadores específicos.

c) Reentrenamiento con warm_start Descripción: Se incrementa el número de iteraciones del modelo original, agregando más árboles al ensemble sin eliminar los anteriores. Ventaja: Permite extender el modelo sin reconstruirlo completamente. Desventaja: Puede introducir sobreajuste si se agregan demasiados árboles, y requiere manejar manualmente el número de iteraciones.

1.6 Estrategia implementada

La estrategia seleccionada fue el reentrenamiento completo desde cero con acumulación de datos. Esta decisión se tomó debido a que el modelo GradientBoostingClassifier no admite partial_fit, y el uso de warm_start requiere un control preciso del número de iteraciones, lo cual puede ser complejo en flujos de datos variables.

Para resolver esta limitación, se diseñó un sistema auxiliar que guarda todos los ejemplos históricos en un archivo binario independiente. Cada vez que se solicita

un reentrenamiento, el sistema combina los nuevos datos con el dataset anterior, y entrena el pipeline desde cero, asegurando así que el modelo conserve todo el conocimiento aprendido en iteraciones previas. El modelo actualizado es sobrescrito para ser utilizado en futuras predicciones. Esta solución equilibra precisión, consistencia y estabilidad sin perder escalabilidad a largo plazo.

Sección 2:

- **Descripción del rol que hará uso de la aplicación**

Analistas de Inteligencia en medios de comunicación, organismos gubernamentales y equipos de investigación académica. Estos profesionales desempeñan un papel clave en la verificación de noticias y la lucha contra la desinformación, especialmente en temas políticos. Dado que la organización que desarrolla esta aplicación está conformada por un grupo de académicos preocupados por el impacto de las noticias falsas en la política, el objetivo principal es proporcionar una herramienta confiable para la evaluación de noticias y la detección de posibles intentos de manipulación de la opinión pública. La existencia de esta aplicación es fundamental para estos analistas, ya que les permite tomar decisiones basadas en datos verificados, contribuyendo a la transparencia informativa y al fortalecimiento de la democracia.

- **Recursos informáticos necesarios para ejecutar la aplicación**

Para su despliegue, la aplicación requiere una computadora con más de 16 GB de RAM y un procesador igual o superior a un Intel Core i3 de 7ma generación. Además, es fundamental contar con el dataset original para garantizar la precisión del modelo (adicional a los datos que van en el segundo endpoint), así como todas las dependencias necesarias para su correcto funcionamiento. También es clave disponer de almacenamiento suficiente para manejar tanto los datos de entrada como los modelos entrenados, asegurando que estos archivos se almacenen de manera organizada y sean fácilmente accesibles para la persistencia del modelo. Es importante definir claramente dónde se guardarán estos archivos y qué mecanismos de respaldo o versionamiento se implementarán para evitar pérdidas de información o problemas de consistencia en el modelo.

Hipotéticamente, si se optara por un despliegue en la nube, la aplicación podría alojarse en plataformas como AWS, Google Cloud o Azure, lo que permitiría mayor escalabilidad y disponibilidad. En este escenario, se podrían aprovechar servicios de almacenamiento en la nube para la persistencia del modelo y bases de datos gestionadas para almacenar el historial de predicciones y re-entrenamientos. Sin embargo, dado que el enfoque principal es un despliegue en hardware local con las especificaciones mencionadas, esta opción se plantea solo como una alternativa teórica en caso de que se requiera una mayor accesibilidad y colaboración remota. En términos de integración con la organización, la aplicación puede ser utilizada de manera independiente o conectarse con sistemas internos

de verificación de noticias de medios de comunicación y agencias gubernamentales. Además, su disponibilidad de descarga en línea facilita su acceso a investigadores académicos y fact-checkers en todo el mundo.

Entre los riesgos asociados al uso de la aplicación, se debe considerar la posible dependencia excesiva del modelo sin validación humana, lo que podría generar sesgos en la clasificación de noticias. Además, existe el riesgo de que actores malintencionados intenten manipular el re-entrenamiento con datos sesgados para alterar los resultados del modelo. También es fundamental garantizar la privacidad de los datos ingresados por los usuarios, especialmente si la aplicación es utilizada por periodistas o analistas en contextos políticos sensibles. Para mitigar estos riesgos, se deben implementar mecanismos de auditoría, transparencia en los datos utilizados para el entrenamiento y una interfaz que fomente la validación humana antes de tomar decisiones basadas en la clasificación automática de noticias.

Roles descripción y repartir puntos:

Roles y Tareas Realizadas por Cada Integrante

El equipo de trabajo estuvo conformado por tres integrantes: David Mateo Barbosa Monsalve, Joan Sebastian Amaya Bohorquez y Nicolas Lara. A continuación, se describen los roles asumidos por cada integrante, las tareas realizadas, el tiempo dedicado a cada actividad, los algoritmos utilizados, los retos enfrentados y las soluciones implementadas.

David Mateo Barbosa Monsalve

David Mateo desempeñó el rol de Ingeniero de Datos, encargándose del desarrollo de pipelines para la automatización del proceso de entrenamiento y reentrenamiento del modelo. También implementó funciones para la persistencia del modelo en almacenamiento local y configuró el proceso de carga de datos y preprocesamiento automatizado. Dedicó aproximadamente 15 horas a estas tareas, trabajando con procesos de preprocesamiento de texto, selección de características y almacenamiento en formato binario. Entre los principales retos enfrentados estuvieron problemas de eficiencia en la carga y transformación de datos, que fueron solucionados mediante la optimización de manipulación de datos con librerías como Pandas y NumPy, además del uso de técnicas de serialización para mejorar la persistencia del modelo.

Joan Sebastian Amaya Bohorquez

Joan asumió el rol de Ingeniero de Software responsable de la API y las pruebas, enfocándose en el desarrollo de los endpoints en FastAPI para la detección de noticias falsas. También implementó pruebas en Postman para garantizar la

funcionalidad de la API y diseñó la interfaz del frontend. Dedicó aproximadamente 15 horas a estas tareas. Trabajó con la implementación de endpoints para predicción y reentrenamiento, además de validaciones de datos mediante esquemas JSON. Los principales retos que enfrentó fueron la compatibilidad entre los modelos entrenados y la API, así como la validación eficiente de entradas. Como solución, utilizó Pydantic para la validación de datos y realizó pruebas continuas con Postman para garantizar la estabilidad de los endpoints.

Nicolas Lara

Nicolas se encargó del rol de Ingeniero de Software responsable del desarrollo del frontend y reporte de resultados. Desarrolló la interfaz gráfica de la aplicación, implementó componentes interactivos para la carga y visualización de resultados y elaboró el informe final del proyecto. Dedicó alrededor de 15 horas a estas tareas. Trabajó con el consumo de la API desde la interfaz y en la visualización de datos mediante gráficos interactivos. Uno de los principales desafíos fue la integración de la API con la interfaz gráfica, así como la presentación clara de los resultados. Para solucionarlo, utilizó frameworks frontend y mejoró la experiencia de usuario mediante gráficos interactivos.

Distribución de Puntos y Evaluación del Trabajo en Equipo

Para evaluar la contribución de cada integrante, se distribuyeron 100 puntos de la siguiente manera:

- David Mateo Barbosa Monsalve: 33 puntos (automatización del modelo y persistencia de datos)
- Joan Sebastian Amaya Bohorquez: 34 puntos (desarrollo de la API y pruebas)
- Nicolas Lara: 33 puntos (desarrollo del frontend y reporte de resultados)

Puntos Para Mejorar en la Siguiente Entrega

1. Mejor coordinación en la integración de todo el entregable para sea acorde en las entregas de todos.
2. Optimización del proceso de reentrenamiento para mejorar la eficiencia en la actualización del modelo.
3. Mayor comunicación para garantizar la alineación de tareas y evitar retrasos en la entrega final.

Cada integrante cumplió con su respectivo rol y se lograron los objetivos propuestos para la etapa del proyecto.