Proyecto 1 etapa 2: Automatización y uso de modelos de analítica de textos

Grupo 24

Juan David Duarte - 202215070

Luisa Hernández - 202114093

Juan Sebastián Sánchez - 202121498

Contexto del problema

La difusión de noticias falsas sobre política a nivel nacional e internacional representa una amenaza significativa para la gobernabilidad y la seguridad global. A nivel interno, estas noticias pueden generar polarización, manipular la opinión pública y debilitar la confianza en las instituciones democráticas, afectando procesos como elecciones y la aceptación de medidas gubernamentales. En el ámbito internacional, pueden provocar tensiones diplomáticas, influir en decisiones de política exterior e incluso ser utilizadas en campañas de desestabilización por actores específicos. Motivados por esta problemática, un grupo de académicos ha recolectado y generado un conjunto de datos de noticias con el propósito de desarrollar modelos de aprendizaje automático que permitan la detección automática de noticias falsas.

Entendimiento y preparación de los datos

Se realizó un análisis exploratorio de texto para evaluar la distribución de las palabras y la cantidad de registros en cada categoría (noticia real o falsa). Adicionalmente, se analiza la longitud promedio de los textos y la presencia de caracteres especiales o signos de puntuación que puedan afectar el procesamiento posterior.

En la preparación y transformación de los datos, se aplican técnicas de limpieza como conversión a minúsculas, eliminación de números, signos de puntuación y caracteres especiales. También se eliminan palabras irrelevantes (stopwords) y se normalizan los términos mediante lemmatización y stemming.

Implementación realizada por el ingeniero de datos

Proceso de automatización de la preparación de datos

El proceso de preparación de datos se dividió en 3 fases incluidas por el pipeline. En primer lugar, utilizamos la clase TextProcessor que incluye diferentes funciones. Process corrige contracciones en el texto, tokeniza el texto en palabras, elimina stopwords en español, plica stemming y lematización para normalizar palabras y usa la función normalizar_palabra para eliminar diacríticos y caracteres no alfabéticos. Finalmente, retorna el texto procesado.

TextVectorizer hace uso de CountVectorizer con nm-gramas de 1 a 3 palabras y establece un filtro para considerar solo las palabras que aparezcan al menos 2 veces en el documento. El resultado es un DataFrame con las columnas del vocabulario que obtuvimos.

La clase FeatureScaler utiliza el método StandardScaler para escalar los datos sin centrar en la media. Retorna un DataFrame con las columnas del vocabulario obtenido y los valores estandarizados.

Construcción del modelo

Se utilizó el algoritmo de clasificación Gradient Boosting para entrenar el modelo ya que en la entrega anterior nos arrojó las mejores métricas para el conjunto test. Para encontrar los mejores hiperparámetros utilizamos GridSearch. Por ello, se utilizó GradientBoostingClassifier con 100 estimadores, una máxima profundidad de 5 y un estado aleatorio fijo. El modelo ya entrenado lo exportamos en un pipeline que contiene el procesamiento del texto, la transformación del conjunto de test y la clasificación. El pipeline preentrenado se utiliza en la aplicación para predecir si una noticia es falsa o verdadera.

```
# Crear un pipeline para el procesamiento del texto, la transformación del conjunto de test y la clasificación
pipeline = Pipeline([
    # Paso 1: Preprocesar el texto
    ('text_preprocessing', FunctionTransformer(text_preprocessing_function)),

    # Paso 2: Vectorizar el texto (usando el vectorizador preentrenado)
    ('vectorization', FunctionTransformer(vectorization_function)),

    # Paso 3: Escalar las características (usando el escalador preentrenado)
    ('scaling', FunctionTransformer(scaling_function)),

    # Paso 4: Clasificar el texto
    ('classification', clasificador_gradiente_final2)
])
```

Persistencia del modelo

El pipeline preentrenado se guardó en un archivo pipeline.joblib. Este pipeline se puede cargar y sobreescribir. Esto será importante a la hora de clasificar noticias y reentrenar el modelo.

```
dump(pipeline, 'pipeline.joblib')
```

```
# Cargar el pipeline preentrenado
pipeline = load('pipeline.joblib')
```

Acceso por medio de la API REST

El backend de la aplicación fue implementado con Python. FastAPI fue el framework utilizado para construir la API en el entorno web. La comunicación se realiza a través de objetos JSON. A continuación, podemos ver la creación de la API y dos clases que

utilizaremos para la gestión de peticiones a la hora de clasificar textos y reentrenar el modelo.

```
app = FastAPI(title="API de Clasificación y Reentrenamiento de Modelo")
```

```
# Modelos de datos para la API
class PredictionInstance(BaseModel):
    message: str
```

```
class RetrainInstance(BaseModel):
    message: str
    Label: int
```

En el siguiente código podemos ver como correr la aplicación.

```
# Para ejecutar la API, en la terminal ejecutar:
# uvicorn main:app --reload
```

Endpoint POST/predict

Este código define un endpoint en FastAPI (`/predict`) que recibe una lista de instancias (`PredictionInstance`), las convierte en un DataFrame de pandas y las procesa con un modelo de *machine learning* (`pipeline`). Primero, verifica que se hayan proporcionado datos y, luego, utiliza el modelo para predecir las etiquetas (`predict`) y sus probabilidades (`predict_proba`). Si ocurre algún error en la predicción, lanza una excepción HTTP 500. Posteriormente, construye la respuesta iterando sobre las predicciones, extrayendo la probabilidad correspondiente y manejando posibles errores. Finalmente, devuelve una lista de objetos `PredictionResponse` con la predicción y su probabilidad asociada.

```
@app.post("/predict", response_model=List[PredictionResponse])
def predict(instances: List[PredictionInstance]):
    if not instances:
        raise HTTPException(status_code=400, detail="No se proporcionaron datos.")

    # Convertir instancias a DataFrame
    df = pd.DataFrame([instance.dict() for instance in instances])
    try:
        preds = pipeline.predict(df["message"])
        probs = pipeline.predict_proba(df["message"])
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error durante la predicción: {e}")

    response = []
    for i, pred in enumerate(preds):
        try:
            class_index = list(pipeline.classes_).index(pred)
            prob = probs[i][class_index]
        except Exception:
            prob = None
        response.append(PredictionResponse(prediction=int(pred), probability=float(prob)))
    return response
```

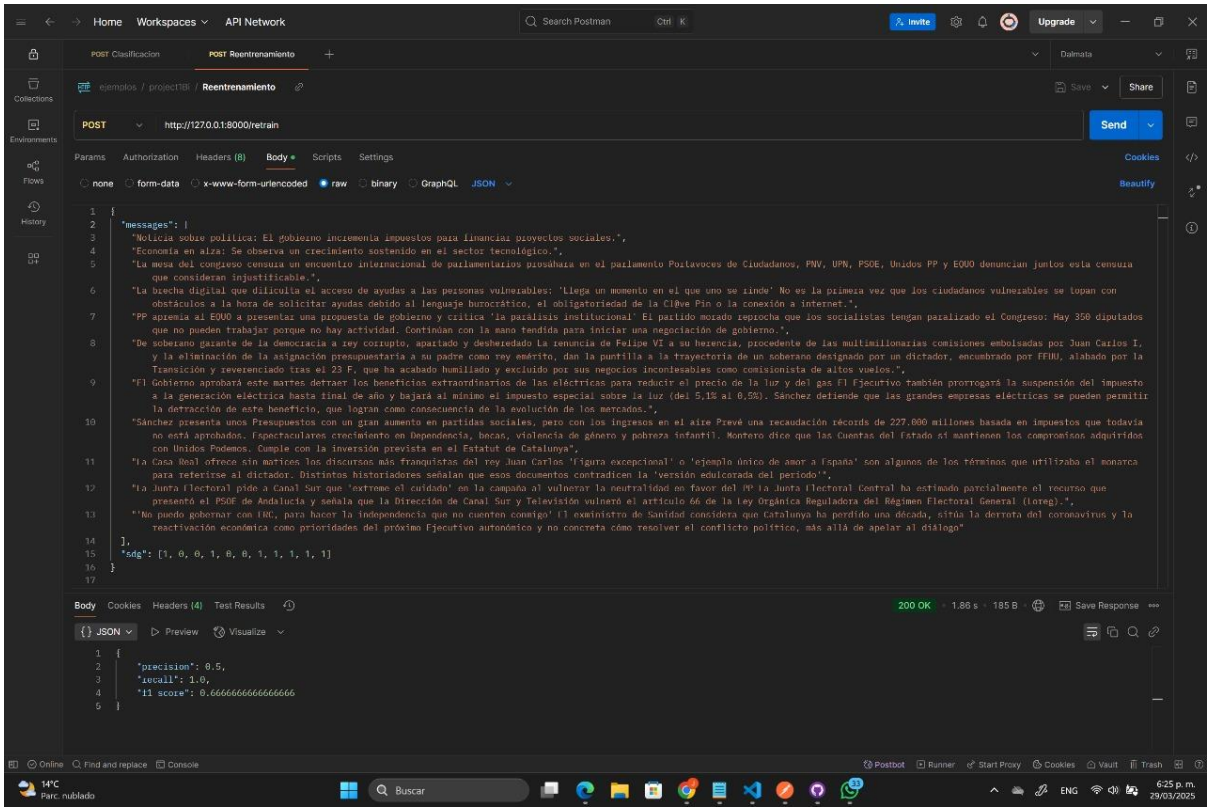
Endpoint POST/retrain

Este código define un endpoint en FastAPI (`/retrain`) que permite reentrenar un modelo de *machine learning* con nuevos datos. Primero, verifica que se haya proporcionado información y la convierte en un DataFrame. Luego, separa los datos en entrenamiento y validación usando `train_test_split`. Intenta ajustar el modelo

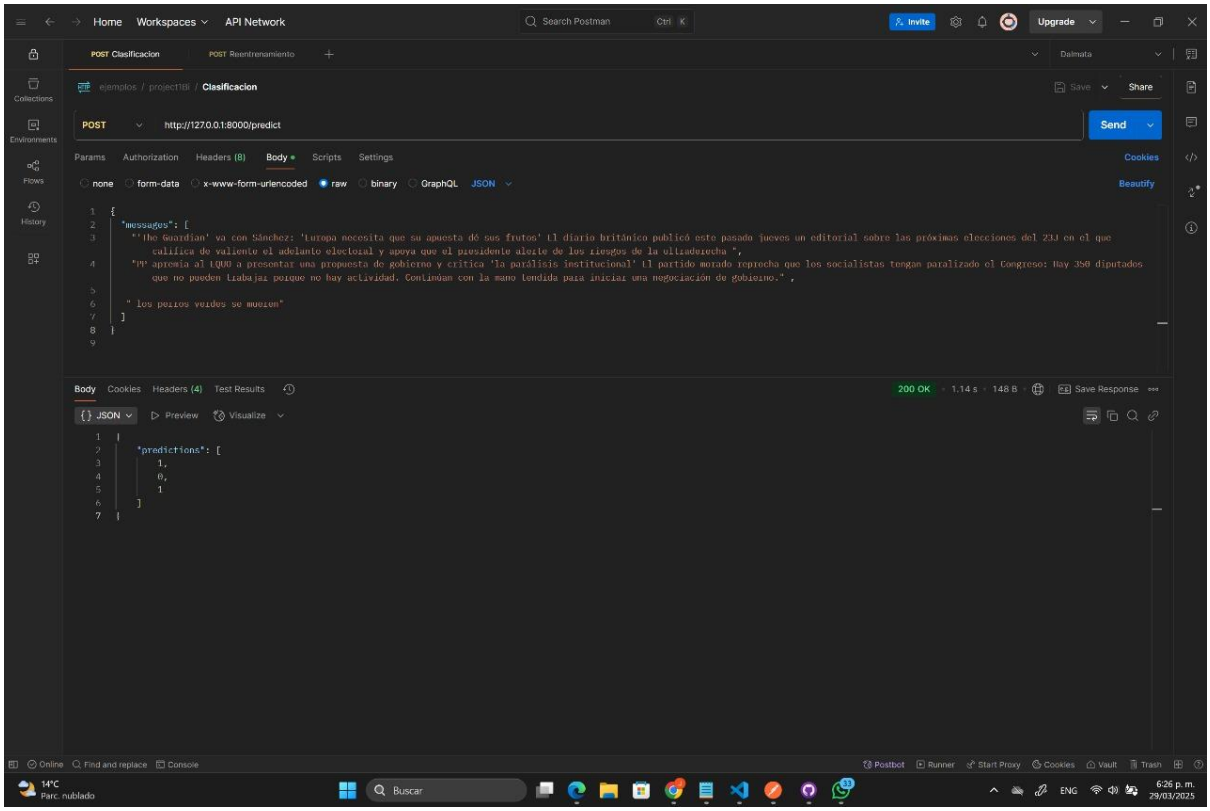
(`pipeline.fit`) y, si ocurre un error, devuelve una excepción HTTP 500. Después, evalúa el modelo calculando precisión, recall y F1-score sobre el conjunto de validación. Finalmente, guarda el modelo actualizado en un archivo y devuelve un mensaje de éxito junto con las métricas obtenidas.

```python
@app.post("/retrain")
def retrain(request: RetrainRequest):
    if not request.data:
        raise HTTPException(status_code=400, detail="No se proporcionaron datos para reentrenar.")

    # Convertir datos a DataFrame
    df = pd.DataFrame([instance.dict() for instance in request.data])
    X = df["message"]
    y = df["Label"]

    # Dividir para evaluar
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=0)
    try:
        pipeline.fit(X_train, y_train)
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error durante el reentrenamiento: {e}")

    try:
        y_pred = pipeline.predict(X_val)
        precision = precision_score(y_val, y_pred, average='weighted')
        recall = recall_score(y_val, y_pred, average='weighted')
        f1 = f1_score(y_val, y_pred, average='weighted')
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error al calcular las métricas: {e}")

    try:
        joblib.dump(pipeline, MODEL_PATH)
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error al guardar el modelo: {e}")

    return {
        "precision": precision,
        "recall": recall,
        "f1_score": f1,
        "mensaje": "Modelo reentrenado exitosamente utilizando reentrenamiento total."
    }
```

Prueba de entrenamiento



Pruebas de clasificación



Reentrenamiento

El reentrenamiento de modelos de aprendizaje automático es esencial para mantener su precisión y relevancia frente a datos cambiantes. A continuación, se presentan tres enfoques distintos para el reentrenamiento:

Aprendizaje Incremental: Este método actualiza el modelo existente incorporando nuevos datos sin necesidad de reutilizar el conjunto de datos original. El modelo ajusta sus parámetros progresivamente a medida que recibe información adicional.

Ventaja: Es eficiente en términos de tiempo y recursos computacionales, ya que evita el reentrenamiento completo desde cero.

Desventaja: Existe el riesgo de que el modelo sufra *desvanecimiento catastrófico*, olvidando patrones previamente aprendidos al enfocarse demasiado en los datos nuevos.

https://es.statisticseasily.com/glossario/what-is-incremental-learning/

Aprendizaje por Lotes (Batch Learning): Implica acumular nuevos datos durante un período y luego reentrenar el modelo desde cero utilizando tanto los datos antiguos como los nuevos.

Ventaja: Mantiene la coherencia y precisión del modelo al considerar el conjunto completo de datos durante el reentrenamiento.

Desventaja: Es computacionalmente costoso y requiere más tiempo, ya que implica procesar todo el conjunto de datos nuevamente.

https://interactivechaos.com/es/manual/tutorial-de-machine-learning/aprendizaje-batch-y-online

Ajuste Fino (Fine-Tuning): Consiste en tomar un modelo previamente entrenado y continuar su entrenamiento con un nuevo conjunto de datos, generalmente con una tasa de aprendizaje más baja.

Ventaja: Permite adaptar modelos generales a contextos específicos, mejorando el rendimiento en tareas particulares.

Desventaja: Si no se maneja adecuadamente, puede llevar al sobreajuste, donde el modelo se adapta demasiado a los nuevos datos y pierde capacidad de generalización.

https://www.ibm.com/es-es/think/topics/fine-tuning

Desarrollo de aplicación y justificación

Usuarios y roles

La aplicación está diseñada para ser utilizada por analistas de datos y profesionales en verificación de información en medios digitales. Su principal propósito es permitir la evaluación automatizada de noticias para identificar si son reales o falsas, apoyándose en un modelo de aprendizaje automático previamente entrenado. Este desarrollo es crucial para los usuarios, ya que proporciona una herramienta de validación rápida y precisa, reduciendo el tiempo necesario para la verificación manual y mejorando la toma de decisiones.

Conexión de la aplicación y el proceso de negocio apoyado

La aplicación se relaciona directamente con la automatización del proceso de negocio de verificación de información en medios digitales. Actualmente, este proceso consume muchos recursos humanos y monetarios. La solución desarrollada permite utilizar un modelo analítico subyacente que puede ser aplicado y reentrenado por los usuarios expertos, facilitando la clasificación de grandes volúmenes de datos de manera eficiente y escalable.

El modelo analítico subyacente se basa en el procesamiento de lenguaje natural y aprendizaje profundo para detectar patrones en el contenido de los textos ingresados. De esta forma, la aplicación optimiza el tiempo de análisis y mejora la confiabilidad de los resultados.

Importancia de la existencia de la aplicación

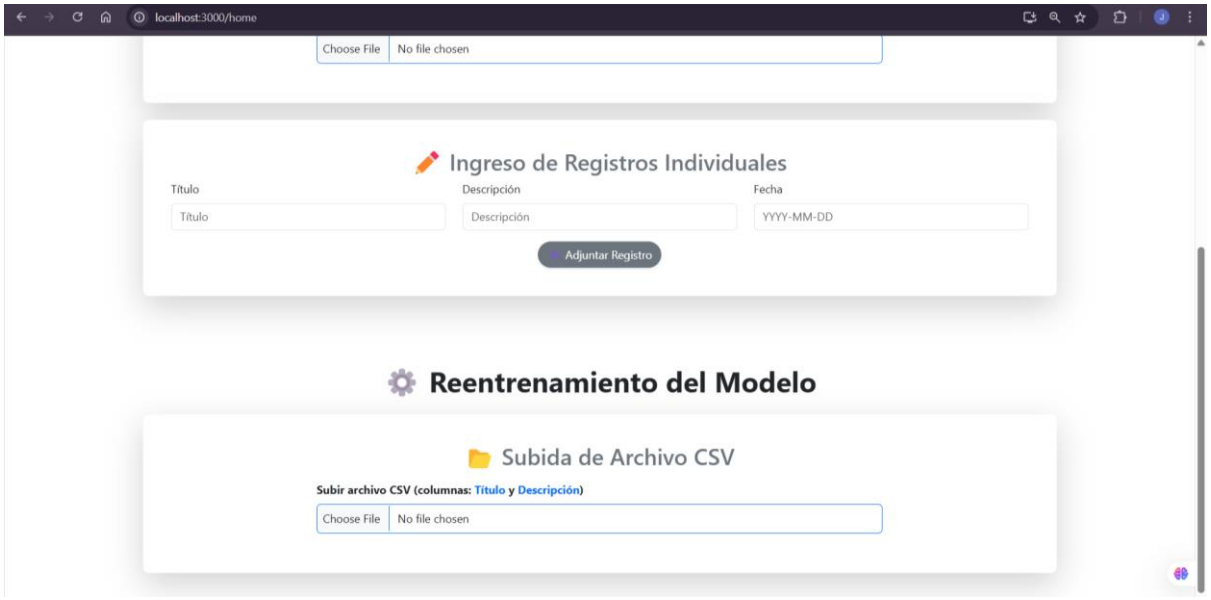
La aplicación es crucial para automatizar la verificación de información, disminuir la carga de trabajo sobre los analistas y mejorar la toma de decisiones. Al ofrecer una herramienta confiable y eficiente, se optimiza la detección de noticias falsas, lo que contribuye a combatir la desinformación.

Para los usuarios expertos, la posibilidad de reentrenar el modelo con nuevos datos permite mantener la aplicación actualizada frente a nuevas técnicas de generación de contenido engañoso. Además, la aplicación ofrece una visualización clara de los factores que influyeron en la clasificación, aumentando la transparencia del proceso y generando confianza en los resultados.

Funcionamiento de la aplicación

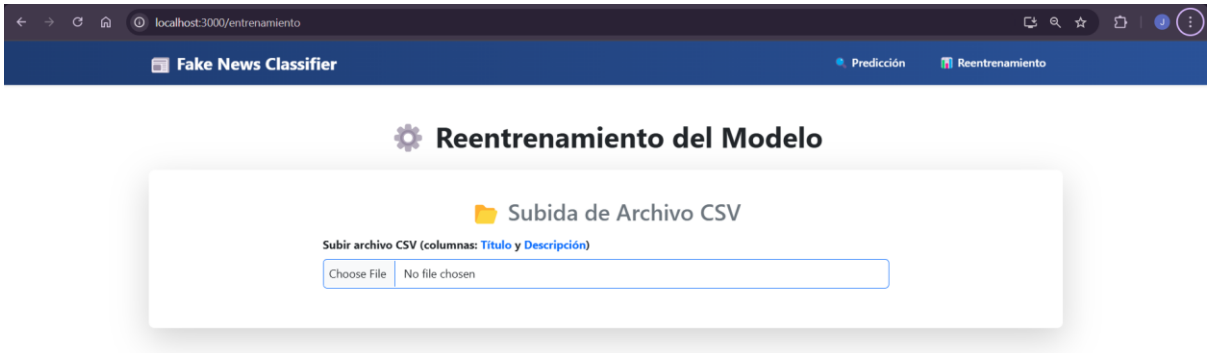
Esta es nuestra página principal. Aquí podemos clasificar los textos con registros individuales ingresando el Titulo, Descripción y Fecha de la noticia o subiendo un archivo CSV donde las columnas sean la de Titulo y Descripción.

Además, podemos reentrenar el modelo igualmente subiendo un archivo CSV donde las columnas sean la de Titulo y Descripción.

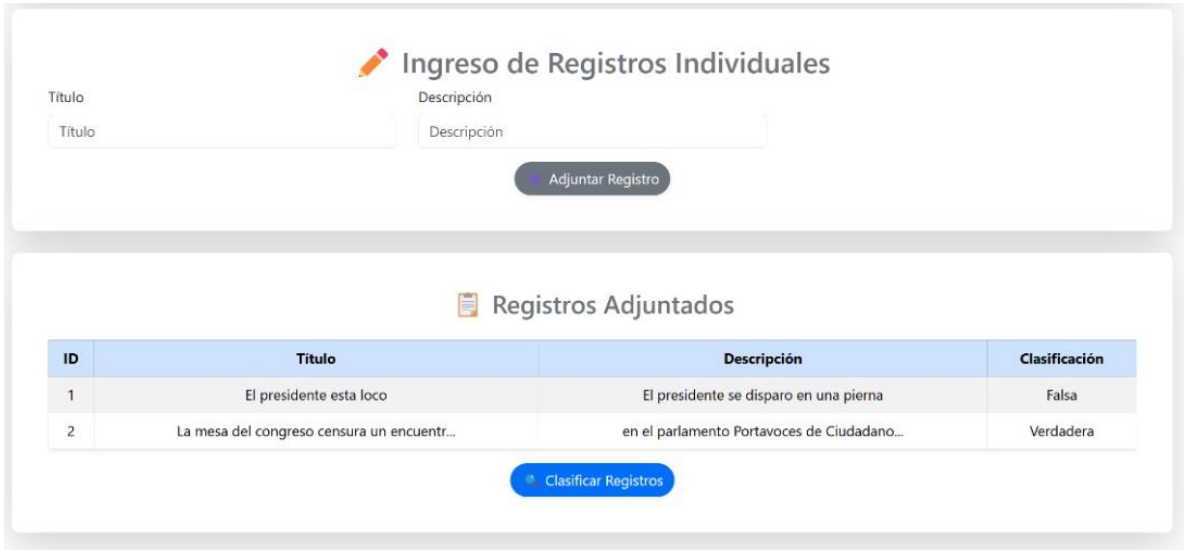


Predicción:

Reentrenamiento:



En la siguiente imagen podemos ver como un registro individual es clasificado:



Para poder correr la página web hay que tener en cuenta:

1. Tener instalado Node.js
2. Correr en la terminal npm install
3. Correr en la terminal npm start

Resultados

Se logró implementar todas las funcionalidades del frontend y el backend previstas, de acuerdo con los diseños y la elección de tecnologías. También se puede reentrenar el modelo sobre información existente, este reentrenamiento también sobrescribe el archivo del modelo como es esperado.

Impacto de la página web al negocio

La aplicación ayuda en gran medida al negocio ya que tenemos una interfaz bastante amigable para el usuario, por lo cual, es fácil para el negocio analizar una gran cantidad de textos y poderlos clasificar con un alto grado de precisión entre noticias falsas o verdaderas. Esta página ayuda a disminuir el tiempo de análisis de textos, así mismo disminuyendo los costos de la empresa. Analizando los resultados, el negocio puede tomar diferentes decisiones que aumenten la confianza en las instituciones democráticas y eviten la divulgación de noticias falsas.

Pruebas de uso

Se realizaron pruebas de uso para verificar que todo funcionara correctamente, durante el desarrollo se priorizó la usabilidad y que la página web fuese intuitiva e interactiva, tanto en diseño de front como en la forma de presentar los resultados.

Enlace del video

Enlace público que redirige al video:

https://www.canva.com/design/DAGjJc5clRg/SY9-6SFgy7_NP6ESkOaj5w/watch?utm_content=DAGjJc5clRg&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=ha0f90ad403

En el desarrollo del proyecto, nuestro equipo estuvo conformado por tres integrantes: Juan Sebastián Sánchez, Juan David Duarte y Luisa Hernandez. A continuación, se detallan los roles asumidos, las tareas realizadas por cada integrante, los tiempos dedicados, los retos enfrentados y la manera en que se resolvieron.

Trabajo en equipo

Roles y responsabilidades

Juan Sebastián (Líder de proyecto e ingeniero de datos)

- Coordinación general del proyecto y gestión de reuniones.
- Definición de fechas clave para entregables y seguimiento de avances.
- Diseño e implementación del pipeline de datos.
- Desarrollo y entrenamiento del modelo de analítica de textos.

- Implementación de la API REST con FastAPI.
- Persistencia del modelo en un repositorio para futuras predicciones y reentrenamiento.
- Validación de eficiencia del modelo mediante pruebas y ajustes.
- Encargado de la entrega del grupo
- Horas dedicadas: 18 horas.

Juan David (Ingeniero de software responsable de desarrollar la aplicación final)

- Desarrollo de la aplicación web para la interacción con la API.
- Integración de los endpoints de predicción y reentrenamiento en la interfaz.
- Implementación de validaciones para asegurar la entrada correcta de datos.
- Optimización del rendimiento de la aplicación mediante buenas prácticas de desarrollo.
- Pruebas unitarias y de integración de la aplicación.
- Horas dedicadas: 20 horas.

Luisa (Ingeniero de software responsable del diseño de la aplicación y resultados)

- Diseño de la interfaz de usuario y experiencia de usuario (UI/UX).
- Generación del video de presentación con la explicación de los resultados.
- Documentación de la aplicación y guía de uso para los usuarios finales.
- Validación de accesibilidad y facilidad de uso.
- Horas dedicadas: 22 horas.

Retos enfrentados y soluciones

1. **Integración del modelo con la API:** Se requirió adaptar el formato de datos para garantizar compatibilidad con la API REST. Se solucionó con preprocesamiento y esquemas JSON bien definidos.

2. **Optimización del rendimiento del modelo:** Se encontró que el modelo inicial tenía tiempos de respuesta elevados. Se resolvió optimizando el pipeline de inferencia y reduciendo la complejidad del modelo.

3. **Interfaz de usuario intuitiva:** Se presentaron dificultades en el diseño de la UI para ingresar datos de forma amigable. Se realizaron pruebas de usuario para ajustar la experiencia.

4. **Gestión del re-entrenamiento del modelo:** Se evaluaron tres enfoques distintos de re-entrenamiento y se optó por un esquema incremental para actualizar el modelo sin perder conocimiento previo.

5. **Coordinación y división equitativa del trabajo:** La carga de trabajo inicial no estaba bien distribuida, lo que se ajustó con reuniones periódicas y reasignación de tareas según el avance.

Distribución de puntos y mejoras para la próxima entrega

Juan Sebastián	33
Juan David	34
Luisa	33

Puntos por mejorar para la siguiente entrega

1. Mayor planificación en la integración del modelo con la aplicación desde el inicio del proyecto.

2. Optimización de tiempos en el desarrollo de la API y la aplicación para evitar retrasos en la integración.

3. Asegurar una documentación más detallada desde las primeras etapas del proyecto para facilitar futuras iteraciones.