Inteligencia de Negocios
Juan Manuel Ramírez Tamayo, Santiago Celis Rengifo, Luis Alfredo Borbón Holguín

PROYECTO 1 - ETAPA 2
GRUPO 22

Sección 1. Proceso de automatización del proceso de preparación de datos, construcción del modelo, persistencia del modelo y acceso por medio de API.

En el proyecto se hizo pipeline donde se automatiza el proceso de transformación de datos y clasificación de textos. Para ello, se emplea la librería Scikit-learn, integrando un Tfidf Vectorizer que convierte los textos en vectores numéricos basados en la frecuencia e importancia de las palabras. Posteriormente, estos vectores son clasificados mediante un Random Forest Classifier, un modelo supervisado que permite identificar si un texto es una noticia falsa o no.
Después, el modelo entrenado es guardado en un archivo .pkl.

- Código del Pipeline:

```python
def create_pipeline():
    return Pipeline([
        ('vectorizer', TfidfVectorizer(max_features=10000)),
        ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
    ])
```

Con el modelo entrenado y almacenado, se implemeenta una API REST desarrollada con FastAPI. Este componente es clave para que el modelo pueda ser consumido por otras aplicaciones de manera rápida y eficiente. La API cuenta con dos endpoints principales:

- Código del API rest:

```python
app = FastAPI()

MODEL_PATH = "model/model.pkl"

class PredictionRequest(BaseModel):
    text: str

class RetrainRequest(BaseModel):
    data: List[str]
    target: List[int]

def load_model():
    if os.path.exists(MODEL_PATH):
        model = joblib.load(MODEL_PATH)
        if not isinstance(model, Pipeline):
            logger.error("El modelo cargado no es un Pipeline válido.")
            raise HTTPException(
                status_code=500,
                detail="El modelo cargado no es un Pipeline válido. Asegúrate de haber entrenado y guardado un pipeline."
            )
        logger.info(f"Modelo cargado exitosamente desde {MODEL_PATH}.")
        return model
    logger.error(f"Modelo no encontrado en {MODEL_PATH}.")
    raise HTTPException(status_code=404, detail="Modelo no encontrado")

def save_model(model):
    try:
        joblib.dump(model, MODEL_PATH)
        logger.info(f"Modelo guardado correctamente en {MODEL_PATH}.")
    except Exception as e:
        logger.error(f"Error al guardar el modelo: {e}")
        raise HTTPException(status_code=500, detail="Error al guardar el modelo.")
```

La finalidad del primer Endpoint permite que se envíe un texto y reciba como respuesta la predicción realizada por el modelo, así como la probabilidad asociada. La API carga el modelo guardado y procesa la solicitud de manera inmediata.

- Código del Primer Endpoint (predicción) :

```python
@app.post("/predict")
async def predict_endpoint(request: PredictionRequest):
    model = load_model()
    try:
        y_pred = model.predict([request.text])
        probabilities = model.predict_proba([request.text])[0]
        max_probability = max(probabilities)
        return {
            "prediction": int(y_pred[0]),
            "probability": float(max_probability),
            "class_probabilities": probabilities.tolist()
        }
    except Exception as e:
        logger.error(f"Error en predict_endpoint: {e}")
        raise HTTPException(status_code=400, detail=f"Error en la predicción: {str(e)}")
```

El segundo endpoint tiene como finalidad permitir la actualización del modelo con nuevos datos y la API se encarga de reentrenar el pipeline desde cero, considerando nuevos datos.

- Código del Segundo Endpoint (reentrenamiento):

```python
@app.post("/retrain")
async def retrain_endpoint(request: RetrainRequest):
    try:
        if len(request.data) != len(request.target):
            logger.error("El tamaño de los datos y las etiquetas no coincide.")
            raise HTTPException(status_code=400, detail="El tamaño de los datos y las etiquetas no coincide.")

        X = pd.DataFrame(request.data, columns=["text"])
        y = request.target

        pipeline = create_pipeline()
        pipeline.fit(X["text"], y)
        save_model(pipeline)

        y_pred = pipeline.predict(X["text"])
        precision = precision_score(y, y_pred, average='weighted')
        recall = recall_score(y, y_pred, average='weighted')
        f1 = f1_score(y, y_pred, average='weighted')

        logger.info("Modelo reentrenado exitosamente.")
        return {
            "precision": precision,
            "recall": recall,
            "f1_score": f1
        }
    except Exception as e:
        logger.error(f"Error en retrain_endpoint: {e}")
        raise HTTPException(status_code=400, detail=f"Error al reentrenar el modelo: {str(e)}")
```

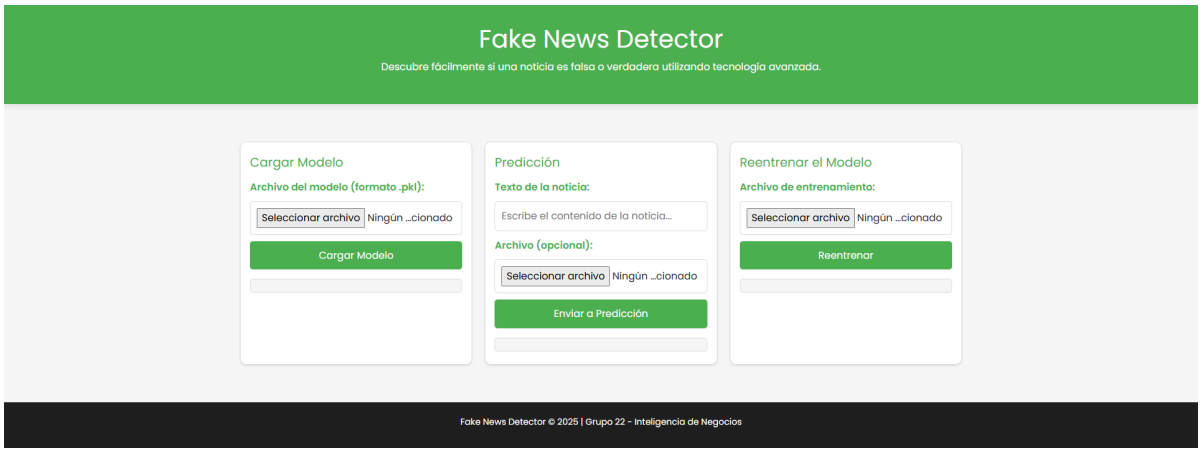
Sección 2. Desarrollo de la aplicación y justificación.

Descripción del usuario/rol de la organización que va a utilizar la aplicación, la conexión entre esa aplicación y el proceso de negocio que va a apoyar (si aplica), y la importancia que tiene para ese rol la existencia de esta aplicación.

- La aplicación está diseñada para ser utilizada por roles dentro de una organización que se encargan de supervisar la difusión de información, como analistas de contenidos digitales, departamentos de comunicación o equipos de monitoreo de noticias. Su objetivo es apoyar el proceso de identificación y prevención de la propagación de noticias falsas (fake news) en tiempo real, proporcionando herramientas que permiten analizar noticias individuales o en conjunto (usando un archivo .csv) para determinar su veracidad. La existencia de esta aplicación es fundamental para este rol, ya que reduce las decisiones subjetivas y permite un análisis más rápido y basado en datos, lo que ayuda a proteger la reputación de la organización, mantener la credibilidad en sus canales de información y prevenir posibles conflictos derivados de la propagación de noticias falsas.

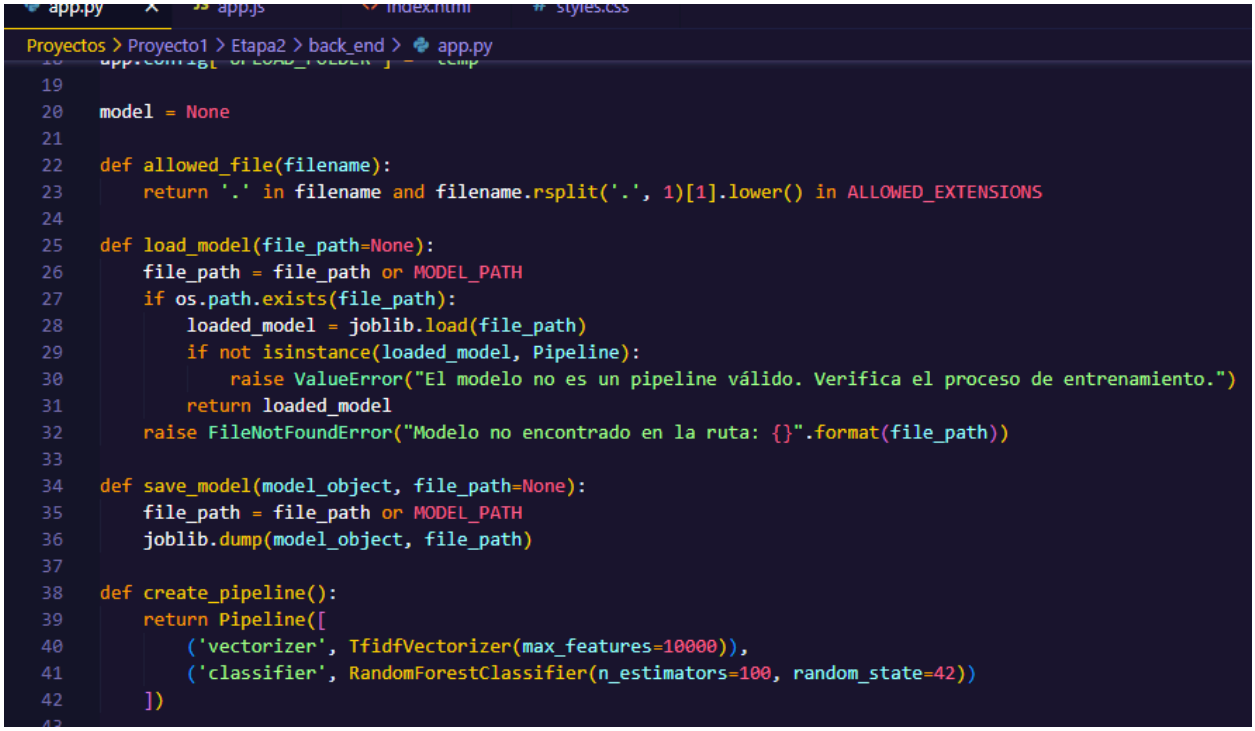
Desarrollo de una aplicación web o móvil para interactuar con el resultado del modelo analítico a partir de un texto o textos dados por el usuario. Recuerde que la interacción incluye tanto la predicción como la probabilidad asociada a la misma. A nivel de usuario experto en modelos analíticos, debe brindar la opción de re-entrenar el modelo, según la semántica que seleccionó.

Capturas del desarrollo:



Archivos importantes:

- app.py: En este archivo se tiene la lógica principal y se conecta el back con el front.



```python
app.py  app.js  index.html  styles.css
Proyectos > Proyecto1 > Etapa2 > back_end > app.py

model = None

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def load_model(file_path=None):
    file_path = file_path or MODEL_PATH
    if os.path.exists(file_path):
        loaded_model = joblib.load(file_path)
        if not isinstance(loaded_model, Pipeline):
            raise ValueError("El modelo no es un pipeline válido. Verifica el proceso de entrenamiento.")
        return loaded_model
    raise FileNotFoundError("Modelo no encontrado en la ruta: {}".format(file_path))

def save_model(model_object, file_path=None):
    file_path = file_path or MODEL_PATH
    joblib.dump(model_object, file_path)

def create_pipeline():
    return Pipeline([
        ('vectorizer', TfidfVectorizer(max_features=10000)),
        ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
    ])
```

- index.html: En este archivo está la lógica de la estructura html de la página web.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Fake News Detector</title>
    <link
      href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600&display=swap"
      rel="stylesheet"
    />
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
      rel="stylesheet"
    />
    <link rel="stylesheet" href="/static/styles.css" />
  </head>
  <body>
    <header class="header">
      <h1>Fake News Detector</h1>
      <p>
        Descubre fácilmente si una noticia es falsa o verdadera utilizando
        tecnología avanzada.
      </p>
    </header>

    <main class="container">
      <!-- Cargar el modelo -->
      <section class="card">
        <h2>Cargar Modelo</h2>
        <form id="load-model-form">      You, 2 days ago • SC: Ajustes
          <label for="model_file">Archivo del modelo (formato .pkl):</label>
          <input type="file" id="model_file" name="model_file" />
          <button type="submit" class="btn-primary">Cargar Modelo</button>
```

- app.js: En este archivo está la lógica en JavaScript para cada endpoint y como se realiza la carga de archivos y del modelo .pkl.

```
document
  .getElementById("load-model-form")
  .addEventListener("submit", async (e) => {
    e.preventDefault(); // Evitar recarga de la página
    const formData = new FormData(e.target);

    try {
      const response = await fetch("/load_model", {
        method: "POST",
        body: formData,
      });
      const result = await response.json();
      document.getElementById("load-model-result").textContent = JSON.stringify(
        result,
        null,
        2
      );
    } catch (error) {
      console.error("Error al cargar el modelo:", error);
      document.getElementById("load-model-result").textContent =
        "Ocurrió un error al cargar el modelo.";
    }
  });

// Manejo del formulario de predicción
document
  .getElementById("predict-form")
  .addEventListener("submit", async (e) => {
    e.preventDefault();
    const formData = new FormData(e.target);

    try {
```

Resuelva las siguientes preguntas con respecto a la aplicación a desplegar: ¿Qué recursos informáticos requiere para entrenar, ejecutar, persistir el modelo analítico y desplegar la aplicación? ¿Cómo se integrará la aplicación construida a la organización, estará conectada con algún proceso del negocio o cómo se pondrá a disposición del usuario final? ¿Qué riesgos tiene para el usuario final usar la aplicación construida?

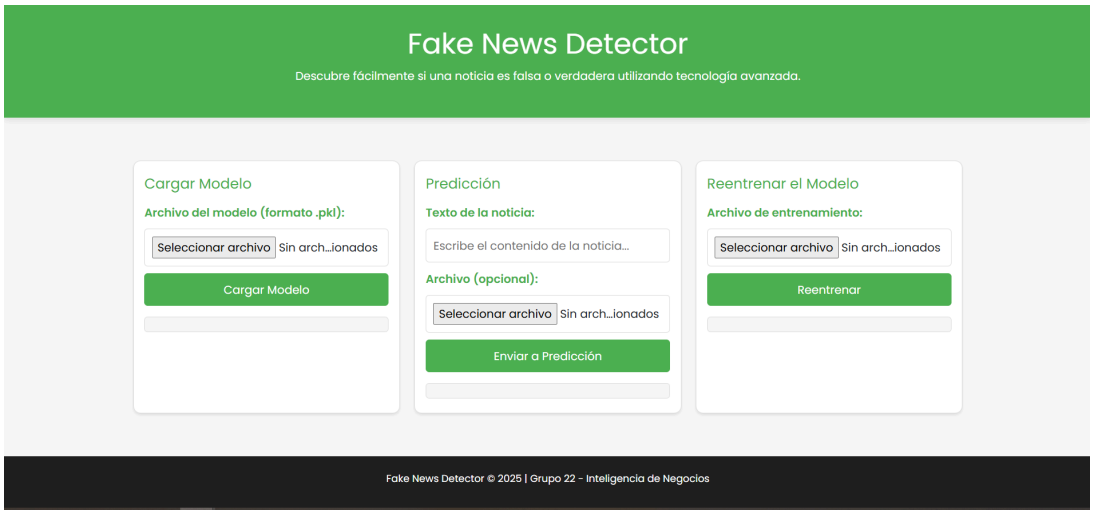
- Para entrenar y ejecutar el modelo analítico, se requiere un entorno informático que pueda manejar bibliotecas como scikit-learn y almacenar los datos etiquetados necesarios para el entrenamiento. Esto incluye acceso a un equipo con buena memoria RAM, almacenamiento disponible para persistir el modelo y una conexión estable a internet en caso de necesitar integraciones externas (por ejemplo, Google Colab para entrenamientos más pesados). En su despliegue, la aplicación se integrará como una herramienta web accesible, que estará disponible ya sea en el navegador interno de la organización o como una plataforma pública con usuarios limitados según permisos. Respecto a los riesgos, el usuario se enfrenta a posibles errores en las predicciones debido a la calidad del modelo analítico, a la lentitud en tiempos de respuesta por sobrecarga del servidor en caso de múltiples solicitudes concurrentes, y a

preocupaciones relacionadas con la privacidad, especialmente si se comparte información sensible o confidencial para el análisis. Es importante tener mecanismos de mitigación para estos casos.

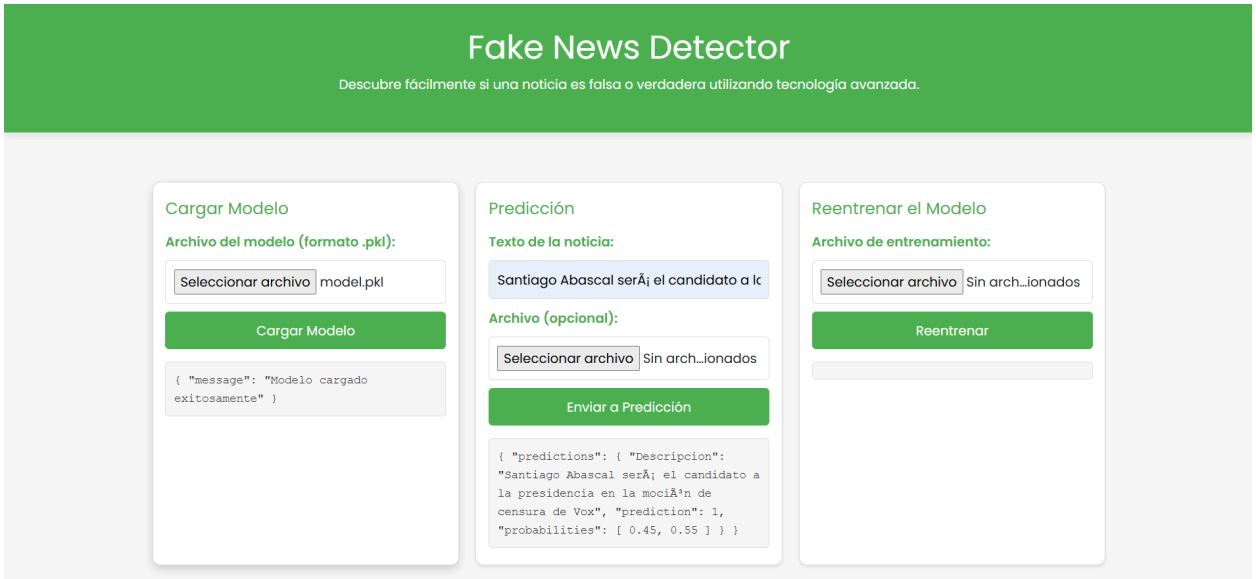
Sección 3. Resultados.

Los resultados obtenidos en la aplicación están explicados en el video publicado en el repositorio, a continuación se mostrarán las imágenes que muestran cada funcionalidad de la aplicación web desarrollada:

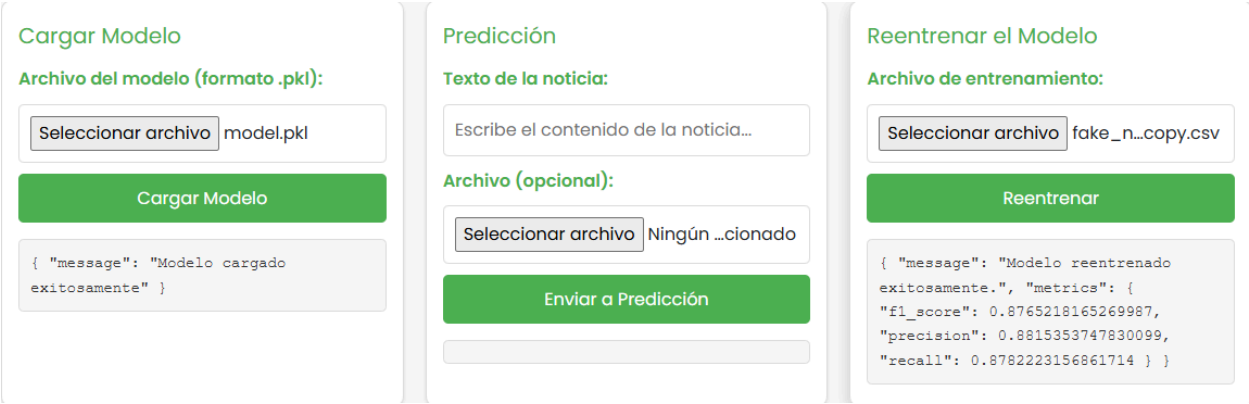
Página principal: Aquí se muestran todas las opciones que brinda la aplicación



Clasificación de las noticias: Aquí se muestra la predicción y las probabilidades de las noticias



Re-entrenamiento del modelo: Aquí se muestran las métricas necesarias para la aplicación



Sección 4. Trabajo en equipo.

Para el trabajo en equipo se llevaron a cabo varias reuniones con el fin de establecer las entregas a corto y largo plazo. Se dividió el trabajo de manera equitativa, y se fueron comunicando los avances a medida que se obtenían. Por otro lado, se definieron los roles y tareas críticas para ciertas fechas.

Principales retos:

Uno de los principales retos fue realizar los endpoints necesarios para cumplir el entrenamiento y reentrenamiento del modelo con los datos proveídos. A su vez, la integración de esta API con la aplicación representó un reto significativo. Por otro lado, el diseño de la interfaz fue complejo de realizar debido a la falta de habilidades en diseño. Sin embargo, estos retos se fueron resolviendo a medida que se avanzaba en el proyecto.

Distribución de puntos:

Nosotros nos repartimos los 100 puntos de la siguiente manera debido a que consideramos que cada estudiante aportó de manera equitativa a las tareas del proyecto:

1. Santiago Celis Rengifo - 33.3 puntos
2. Luis Borbon Holguin - 33.3 puntos

3. Juan Manuel Ramírez Tamayo - 33.3 puntos

ROLES:

Se definieron los siguientes roles para poder gestionar las tareas y los distintos entregables que se necesitaban (cada líder organizó la división de trabajo en su área específica de tal manera que fuera lo más balanceado posible):

- **Líder de Proyecto:** Santiago Celis Rengifo
- **Ingeniero de datos:** Juan Manuel Ramírez Tamayo
- **Ingeniero de software responsable del diseño de la aplicación y resultados:** Luis Borbon Holguin
- **Ingeniero de software responsable de desarrollar la aplicación final:** Santiago Celis Rengifo

Tareas realizadas:

Cada estudiante realizará las siguientes tareas:

1. Santiago Celis Rengifo:
    a. Realización del diseño de la aplicación
    b. Desarrollo de la aplicación

2. Luis Borbon Holguin:
    a. Creación del pipeline para el modelo
    b. Creación del API con los endpoints solicitados

3. Juan Manuel Ramírez Tamayo:
    a. Presentación de los resultados

Sección 6. Otros entregables y espacios de evaluación

Link del repositorio: https://github.com/NEXAF7/Grupo22-BI-20251
Los demás archivos están cargados en el repositorio.
Nótese que en la wiki del repositorio está el paso a paso para correr la aplicación.