UNIVERSIDAD DE LOS ANDES

INTELIGENCIA DE NEGOCIOS

PROYECTO 1, ETAPA 2

PROFESORA

MARÍA DEL PILAR VILLAMIL

ESTUDIANTE

BORIS N. REYES R.

OCTUBRE, 2024

TABLA DE CONTENIDO

DESCRIPCIÓN DE LA ETAPA

Esta etapa se centra en el rol de ingeniero de datos y hay actividades del ingeniero de software. El ingeniero de datos tiene la responsabilidad de tomar el resultado entregado por el científico de datos (etapa 1), completarlo incluyendo elementos como pruebas, organización de código, validación de atributos de calidad (e.g., eficiencia), además de persistir el modelo generado en un repositorio específico para este fin. Adicionalmente, deben extender el código para incluir el registro o log, que permita en caso de falla, revisar las entradas al modelo, los datos utilizados para el entrenamiento y prueba del modelo y el resultado de la preparación. El ingeniero de software toma ese resultado como entrada de la aplicación web o móvil que debe diseñar y construir, la cual es utilizada por el usuario final para ingresar su solicitud al modelo, interactuar con el resultado y apoyar sus acciones o decisiones.

En esta etapa 2 deben **automatizar el desarrollo del modelo de analítica de textos creado en la etapa 1 del proyecto y desarrollar una aplicación orientada a un usuario con sentido y apoyo a cualquier acción o decisión de este.**

AUTOMATIZACIÓN DEL PROCESO DE PREPARACIÓN DE DATOS, CONSTRUCCIÓN DEL MODELO, PERSISTENCIA DEL MODELO Y ACCESO POR MEDIO DE API

El proceso de automatización para la preparación de datos, construcción del modelo y su acceso mediante una API incluyó la implementación de un pipeline que se encarga de realizar el preprocesamiento de los datos textuales, la vectorización y la clasificación utilizando un modelo de Máquina de Soporte Vectorial (SVM). A continuación, se describen las fases clave de este proceso:

1. Construcción del Pipeline

La construcción del pipeline fue realizada en Google Colab utilizando scikit-learn. El pipeline incluye tres pasos principales:

- Preprocesamiento de Texto: Se implementó un transformador personalizado (FunctionTransformer) para procesar el texto, donde se aplicaron técnicas de PLN como normalización (minúsculas), eliminación de caracteres no ASCII, tokenización y lematización utilizando nltk. Este preprocesamiento asegura que los textos sean limpiados y transformados en una forma apta para el análisis.
- Vectorización del Texto: Se utilizó TfidfVectorizer para convertir los textos preprocesados en representaciones numéricas (matrices TF-IDF), limitando las características a las 5000 más comunes para reducir la dimensionalidad y mejorar la eficiencia.

- Clasificación: Finalmente, se utilizó un modelo SVM configurado con la opción de calcular probabilidades (probability=True), lo cual es útil para obtener estimaciones de confianza sobre las predicciones.

Este pipeline fue entrenado con el conjunto de datos de texto del proyecto y exportado utilizando joblib en un archivo .pkl para su posterior uso en una API.

```
# Snippet del pipeline
pipeline = Pipeline([
    ('preprocessing', FunctionTransformer(preprocess_text_data, validate=False)),
    ('vectorizer', TfidfVectorizer(max_features=5000)),
    ('classifier', SVC(probability=True))  # Usar probabilidad para obtener estimaciones de confianza
])

pipeline.fit(X_train, y_train)
joblib.dump(pipeline, 'unfpa_text_classification_pipeline.pkl')

```

2. Construcción del API

La API fue implementada utilizando FastAPI con dos endpoints principales:

Endpoint de Predicción: Este endpoint recibe un texto en español y devuelve la categoría de Objetivo de Desarrollo Sostenible (ODS) que mejor representa ese texto, junto con la probabilidad de la predicción.

Endpoint de Reentrenamiento: Permite al usuario enviar nuevos datos etiquetados para actualizar el modelo. Una vez reentrenado, el pipeline actualizado se guarda nuevamente en un archivo .pkl para asegurar que las futuras predicciones utilicen el modelo reentrenado.

El pipeline es gestionado por la clase Model dentro de PredictionModel.py, donde se carga, utiliza para hacer predicciones, y se reentrena si es necesario. Para centralizar el manejo del pipeline, se instanció la clase Model, la cual carga el archivo .pkl y realiza las operaciones directamente.

```
#Snippet de PredictionModel.py


class Model:
    def __init__(self):
        try:
            self.pipeline = load("unfpa_text_classification_pipeline.pkl")
        except Exception as e:
            raise Exception(f"Error loading pipeline: {str(e)}")

    def make_predictions(self, data):
        prediction = self.pipeline.predict(data['Textos_espanol'])
        probability = self.pipeline.predict_proba(data['Textos_espanol'])
        return {"prediction": int(prediction[0]), "probability": probability[0].max()}

```

```
    def retrain_model(self, data, labels):
        self.pipeline.fit(data['Textos_espanol'], labels)
        dump(self.pipeline, 'unfpa_text_classification_pipeline.pkl')

```

3. Persistencia del Modelo

El modelo es persistido utilizando joblib. Se guarda en un archivo .pkl después de su entrenamiento inicial y también después de cada reentrenamiento. Este archivo es reutilizado en cada solicitud de predicción o reentrenamiento.

4. Acceso mediante API

El acceso a las predicciones y el reentrenamiento del modelo se realiza mediante dos endpoints HTTP en la API de FastAPI. La API recibe los datos en formato JSON, los transforma en DataFrame y luego utiliza el pipeline cargado para realizar las predicciones o reentrenamientos según sea necesario.

DESARROLLO DE LA APLICACIÓN Y JUSTIFICACIÓN

La aplicación está diseñada para un analista de datos en el UNFPA, quien se encarga de procesar grandes volúmenes de información textual con el fin de generar predicciones rápidas y precisas que respalden la toma de decisiones. Este rol es crucial en el desarrollo de modelos predictivos que permiten a la organización identificar patrones en los datos y realizar ajustes continuos. La aplicación facilitará reentrenamientos frecuentes de los modelos analíticos, asegurando que estos estén siempre actualizados y optimizados para ofrecer *insights* relevantes.

Además, la aplicación automatiza el proceso de análisis de opiniones textuales, lo que mejora la precisión en las predicciones y garantiza que las decisiones se basen en análisis robustos. Al permitir reentrenamientos rápidos y eficientes, se mantendrá la relevancia de los modelos a lo largo del tiempo, impactando positivamente en la toma de decisiones estratégicas del UNFPA. Esta capacidad de mejorar continuamente los modelos predictivos optimiza la asignación de recursos hacia intervenciones prioritarias y reduce la dependencia de análisis manuales extensos.

Interfaz y Funcionalidad de la Aplicación

La aplicación está estructurada en dos secciones principales: **Predicción** y **Reentrenamiento del Modelo**.

1. **Predicción de Textos**: En esta sección, el usuario puede ingresar un texto o una serie de textos en un formato sencillo. El sistema procesa los datos ingresados mediante el pipeline del modelo, y devuelve no solo la categoría

predicha (en este caso, el SDG correspondiente), sino también la probabilidad asociada a cada predicción, lo que añade transparencia y confiabilidad al proceso de toma de decisiones.

```
const handlePredict = () => {
    const inputText = textInput;
    fetch('/api/predict', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ text: inputText }),
    })
     .then(response => response.json())
     .then(data => setPredictionResult(data))
     .catch(() => setPredictionError("Invalid input format"));
};

```

2. **Reentrenamiento del Modelo**: En esta sección, el usuario puede ingresar un conjunto de datos en formato JSON que contiene tanto las características (en este caso, el texto) como la variable objetivo (la categoría SDG). Al procesar estos datos, el sistema reentrena el modelo y devuelve métricas clave como precisión, recall y F1-score, lo que permite evaluar el desempeño del modelo actualizado.

```
const handleRetrain = () => {
    const retrainData = retrainInput;
    fetch('/api/retrain', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(retrainData),
    })
     .then(response => response.json())
     .then(data => setRetrainResult(data))
     .catch(() => setRetrainError("Invalid JSON format for re-training data"));
};

```

RESULTADOS

Durante el desarrollo, uno de los principales problemas fue la integración del pipeline entrenado en Colab con la API de FastAPI. Aunque el pipeline se entrenaba y funcionaba correctamente dentro del entorno de Colab, al intentar cargarlo en el entorno local de la API, surgieron problemas de deserialización debido a funciones personalizadas dentro del pipeline, como el preprocesamiento de texto. Estas funciones personalizadas causaban errores cuando se intentaba cargar el archivo .pkl en un entorno diferente al original.

Posibles causas del fallo

Incompatibilidad de Entornos: Las versiones de las librerías (como scikit-learn y joblib) diferían entre Colab y el entorno local, lo que pudo haber causado problemas de compatibilidad. (Se realizaron ajustes y no se obtuvieron resultados positivos)

Serialización de Funciones Personalizadas: El uso de transformadores personalizados dentro del pipeline (como el preprocesamiento de texto) pudo haber causado problemas durante la serialización y deserialización del pipeline.

Acciones para resolver el problema:

Se optó por centralizar la carga y gestión del pipeline dentro de la clase PredictionModel para simplificar el flujo y evitar problemas de deserialización. Sin embargo, los problemas persisten en algunos casos debido a la complejidad del preprocesamiento personalizado.

Actualmente se están explorando opciones para reconfigurar el pipeline utilizando funciones y transformadores estándar de scikit-learn para asegurar que el archivo .pkl funcione en diferentes entornos sin problemas.

Se sigue trabajando para solucionar estos problemas de integración y garantizar que la API funcione de manera óptima con el pipeline de clasificación de texto.

TRABAJO EN EQUIPO

Al trabajar individualmente, se funge los roles de líder, ingeniero de datos, ingeniero de software responsable del diseño de la aplicación y resultados, e ingeniero de software responsable de desarrollar la aplicación final por parte del estudiante Boris N. Reyes R. Las ceremonias se plantean como revisiones espaciadas a lo largo de la duración del sprint (1 semana) con el objetivo de identificar y realizar mejoras y correcciones a las diferentes secciones que constituyen la Etapa 2 del Primer Proyecto. Se le asocian todas las responsabilidades y sus respectivos valores cuantitativos al momento de evaluar el proyecto.