Trabajo en equipo - Descripción de roles y tareas

Integrantes:

- **Andrés Pérez**
- **Juana Morales**
- **John Serrato**

Descripción del código:

El código implementado para el desarrollo de la aplicación implementa una API utilizando FastAPI para el modelo analítico basado en aprendizaje automático creado en la etapa 1, el cual clasifica textos en categorías de los Objetivos de Desarrollo Sostenible (ODS). De esta forma, la API tiene dos funcionalidades o endpoints principales:

1. **Predicción de categorías ODS**:

```python
# Endpoint de predicción con cargue de archivo CSV
@app.post("/prediccion/")
async def predecir(file: UploadFile = File(...)):
    try:
        # Cargar el archivo CSV
        df = pd.read_csv(file.file, sep=',', encoding = "ISO-8859-1")

        textos = df['Textos_espanol'].tolist()

        # Se preprocesan los textos antes de predecir
        textos_preprocesados = preprocess_texts(textos)

        # Se realiza la predicción con el modelo
        predicciones = modelo_entrenado.predict(textos_preprocesados)
        probabilidades = modelo_entrenado.predict_proba(textos_preprocesados)

        resultado = []
        for i in range(len(predicciones)):
            resultado.append({
                "prediccion": int(predicciones[i]),
                "probabilidad": float(max(probabilidades[i]))*100
            })

        return {"resultado": resultado}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Descripción y explicación paso a paso del código

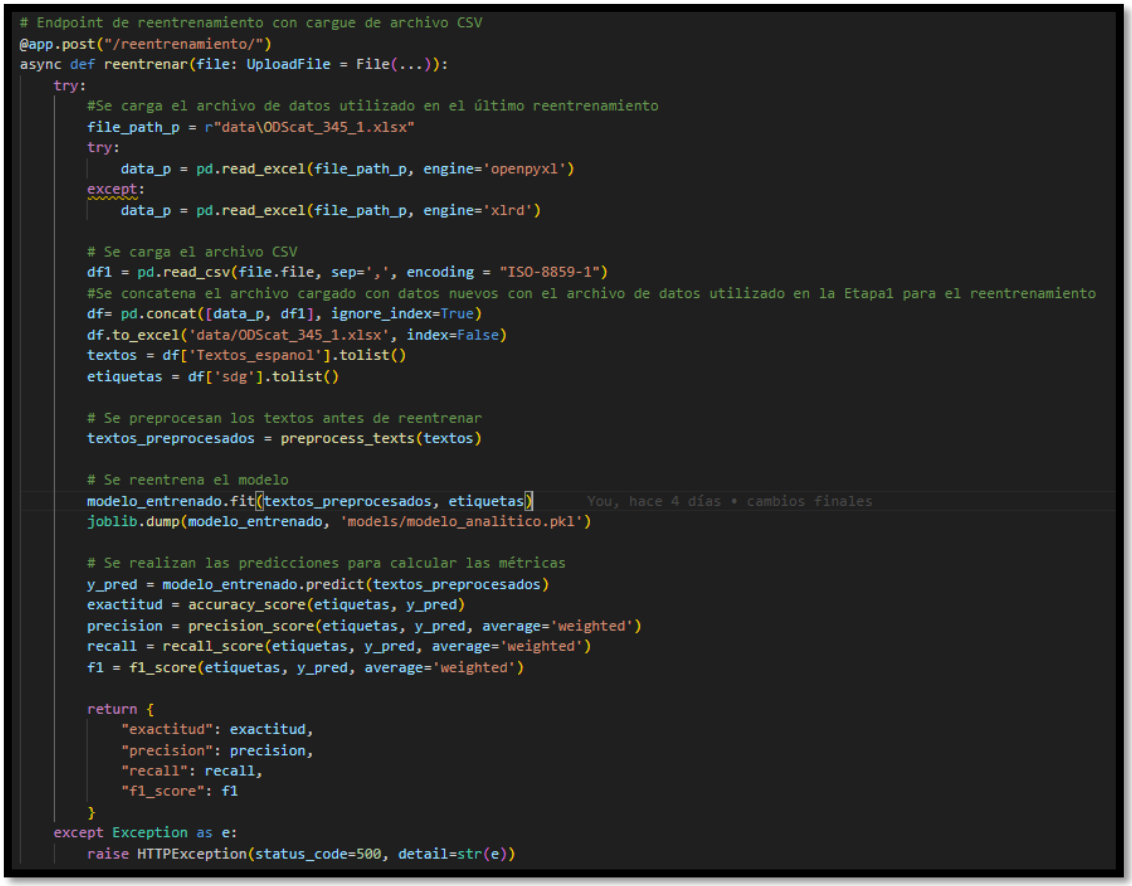
1. Se recibe un archivo CSV mediante una solicitud POST. Este archivo contiene los textos a predecir en la columna Textos_espanol.
2. El archivo CSV se lee con pandas (pd.read_csv()), extrayendo los textos a predecir de la columna Textos_espanol y se almacenan en una lista.
3. Los textos extraídos pasan por una función de preprocesamiento (preprocess_texts()), que normaliza y limpia los textos para asegurar que estén listos para la predicción.

```python
# Se adapta la función replace_special_chars para listas de textos
def replace_special_chars(texts):
    replacements = {
        'Ã¡': 'á', 'Ã©': 'é', 'Ã': 'í', 'Ã³': 'ó', 'Ãº': 'ú',      You, hace 5 días • API-LIS
        'Ã': 'Á', 'Ã': 'É', 'Ã': 'Í', 'Ã': 'Ó', 'Ã': 'Ú',
        'Ã±': 'ñ', 'Ã': 'Ñ', 'Ã¼': 'ü', 'Ã': 'Ü', 'ã³':'ó', 'ã':'í',
        'ã©':'é', 'ã¡':'á', 'ãº':'ú', 'ã°':'ó', 'ã':'ñ', 'ã':'ü'
    }

    # Se reemplazan los caracteres especiales en cada texto
    texts_replaced = []
    for text in texts:
        for old, new in replacements.items():
            text = text.replace(old, new)
        texts_replaced.append(text)

    return texts_replaced

# Función para normalizar documentos
def normalize_documents(doc):
    """Normaliza un documento preservando acentos y capitalización adecuada"""
    doc = re.sub(r'[^\w\sáéíóúÁÉÍÓÚñÑ]', '', doc)  # Mantiene tildes y caracteres como ñ
    words = doc.split()
    new_words = []
    capitalize_next = True

    for word in words:
        if capitalize_next:
            new_word = word
            capitalize_next = False
        else:
            if word[0].isupper() and word[1:].islower():
                new_word = word
            else:
                new_word = word.lower()

        if word.endswith('.'):
            capitalize_next = True

        new_words.append(new_word)

    doc = ' '.join(new_words)
    tokens = wpt.tokenize(doc)
    filtered_token = [ps.stem(token) for token in tokens if token.lower() not in stop_words]
    doc = ' '.join(filtered_token)
    return doc
```

```python
# Se normaliza cada texto individualmente
def normalize_corpus(texts):
    return [normalize_documents(text) for text in texts]

# Función para preprocesar los textos
def preprocess_texts(texts):
    # Se reemplazan los caracteres especiales
    textos_reemplazados = replace_special_chars(texts)
    # Se aplica normalización y demás procesamiento a los datos
    textos_normalizados = normalize_corpus(textos_reemplazados)
    return textos_normalizados
```

4. El modelo entrenado utiliza los textos preprocesados para predecir la categoría o etiqueta asociada a cada texto usando el método predict().
5. El modelo calcula la probabilidad de que cada texto pertenezca a su categoría predicha, usando predict_proba(). Esto indica la confianza del modelo en la predicción.
6. Se organizan los resultados en una lista, donde cada entrada contiene la predicción y su probabilidad correspondiente en formato de porcentaje.
7. Finalmente, se devuelve una respuesta JSON con la lista de predicciones y las probabilidades asociadas.

2. **Reentrenamiento del modelo**:



```python
# Endpoint de reentrenamiento con cargue de archivo CSV
@app.post("/reentrenamiento/")
async def reentrenar(file: UploadFile = File(...)):
    try:
        #Se carga el archivo de datos utilizado en el último reentrenamiento
        file_path_p = r"data\ODScat_345_1.xlsx"
        try:
            data_p = pd.read_excel(file_path_p, engine='openpyxl')
        except:
            data_p = pd.read_excel(file_path_p, engine='xlrd')

        # Se carga el archivo CSV
        df1 = pd.read_csv(file.file, sep=',', encoding = "ISO-8859-1")
        #Se concatena el archivo cargado con datos nuevos con el archivo de datos utilizado en la Etapa1 para el reentrenamiento
        df= pd.concat([data_p, df1], ignore_index=True)
        df.to_excel('data/ODScat_345_1.xlsx', index=False)
        textos = df['Textos_espanol'].tolist()
        etiquetas = df['sdg'].tolist()

        # Se preprocesan los textos antes de reentrenar
        textos_preprocesados = preprocess_texts(textos)

        # Se reentrena el modelo
        modelo_entrenado.fit(textos_preprocesados, etiquetas)      You, hace 4 días • cambios finales
        joblib.dump(modelo_entrenado, 'models/modelo_analitico.pkl')

        # Se realizan las predicciones para calcular las métricas
        y_pred = modelo_entrenado.predict(textos_preprocesados)
        exactitud = accuracy_score(etiquetas, y_pred)
        precision = precision_score(etiquetas, y_pred, average='weighted')
        recall = recall_score(etiquetas, y_pred, average='weighted')
        f1 = f1_score(etiquetas, y_pred, average='weighted')

        return {
            "exactitud": exactitud,
            "precision": precision,
            "recall": recall,
            "f1_score": f1
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Descripción y explicación paso a paso del código:

1. Se carga un archivo Excel que contiene los datos previos utilizados en el reentrenamiento anterior. Se usa openpyxl como motor por defecto y xlrd como alternativa si openpyxl falla.
2. El archivo CSV enviado por el usuario se carga utilizando pandas.read_csv(), especificando el formato de codificación como "ISO-8859-1" para manejar caracteres especiales.
3. Los datos del archivo recién cargado se combinan con los datos previos que ya están en el archivo Excel. Se utiliza pd.concat() para unir ambos DataFrames. El archivo combinado se guarda nuevamente en un Excel para asegurar que los datos se mantengan actualizados.
4. Se extraen los textos de la columna 'Textos_espanol' y las etiquetas de la columna 'sdg' del DataFrame combinado.
5. Los textos extraídos se preprocesan usando una función que limpia y normaliza los datos antes de pasarlos al modelo. Esto asegura que el modelo reciba entradas consistentes.

6. El modelo cargado se reentrena utilizando el método fit() con los textos preprocesados y las etiquetas correspondientes.
7. Después de reentrenar el modelo con los nuevos datos, el modelo actualizado se guarda en un archivo usando joblib.dump() para su reutilización.
8. Se realizan predicciones sobre los textos preprocesados para evaluar el rendimiento del modelo reentrenado. Se calculan las métricas de exactitud, precisión, recall y f1-score.
9. Las métricas de rendimiento (exactitud, precisión, recall y f1-score) se devuelven en formato JSON para que el usuario pueda ver los resultados del reentrenamiento.

Adicionalmente, en la creación del API, el código (por fuera del código de los endpoints explicado anteriormente):

```python
#Se carga el archivo de datos utilizado en la Etapa1
file_path = r"data\ODScat_345.xlsx"
try:
    data_t = pd.read_excel(file_path, engine='openpyxl')
except:
    data_t = pd.read_excel(file_path, engine='xlrd')

#Se adapta la función replace_special_chars para los datos utilizados en la Etapa1
def replace_special_chars(df):
    replacements = {
        'Ã¡': 'á', 'Ã©': 'é', 'Ã': 'í', 'Ã³': 'ó', 'Ãº': 'ú',
        'Ã': 'Á', 'Ã': 'É', 'Ã': 'Í', 'Ã': 'Ó', 'Ã': 'Ú',
        'Ã±': 'ñ', 'Ã': 'Ñ', 'Ã¼': 'ü', 'Ã': 'Ü'
    }
    for col in df.select_dtypes(include=['object']).columns:
        for old, new in replacements.items():
            df[col] = df[col].str.replace(old, new)
    return df
#Se envia a un excel los datos que se van a utilizar para el reentrenamiento
data_t.to_excel('data/ODScat_345_1.xlsx', index=False)
# Se aplica la función a los datos cargados
data_t = replace_special_chars(data_t)

# Se devide el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(data_t['Textos_espanol'], data_t['sdg'], test_size=0.2, random_state=42)

# Se crea el pipeline para automatizar el proceso
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=5000)),  # Vectorización con TF-IDF
    ('svm', SVC(kernel='rbf', gamma='scale', C=1, probability=True))  # Clasificador SVM
])

# Se entrena el pipeline
pipeline.fit(X_train, y_train)

# Se guarda el pipeline completo (preprocesamiento + modelo entrenado)
joblib.dump(pipeline, 'models/modelo_analitico.pkl')
```

```python
# Se ejecuta la api
if __name__ =="__main__":
    uvicorn.run("api:app", host="0.0.0.0", port=8000, reload=True)
```

- Utiliza un pipeline de preprocesamiento con TF-IDF y un clasificador SVM con los hiperparámetros escogidos durante la primera etapa del proyecto.
- Implementa funciones para preprocesar los textos para entrenar el modelo, manejar la carga de archivos a utilizar, y actualizar el modelo.
- Utiliza la librería joblib para persistir el modelo entrenado y uvicorn para ejecutar la API.

Definiciones de reentrenamiento:

1. **Reentrenamiento incremental:**

Este método actualiza el modelo solo con los nuevos datos proporcionados, sin empezar desde cero. Es eficiente en términos de tiempo y recursos, ya que no se necesita procesar todo el conjunto de datos desde el principio.

- **Ventajas**: Su eficiencia permite actualizaciones rápidas y frecuentes, ideal para sistemas de producción que requieren adaptarse rápidamente a nuevos datos.
- **Desventajas**: Puede generar un desbalance si los nuevos datos no son representativos de la distribución completa o son muy diferentes de los datos originales.

2. **Reentrenamiento completo:**

El modelo se entrena nuevamente desde cero, usando tanto los datos antiguos como los nuevos, procesando todo el conjunto de datos desde el principio.

- **Ventajas**: Asegura que el modelo tenga una visión completa de los datos, evitando sesgos hacia los datos nuevos y corrigiendo posibles errores acumulados.
- **Desventajas**: Es más costoso en términos de tiempo y recursos computacionales, ya que requiere un entrenamiento completo cada vez que se añaden nuevos datos.

3. **Reentrenamiento con selección de muestras:**

Se selecciona una muestra representativa de los datos antiguos junto con los nuevos para reentrenar el modelo, manteniendo un balance en la distribución de los datos.

- **Ventajas**: Permite un reentrenamiento más rápido que el completo, manteniendo una representación equilibrada de los datos originales y los nuevos.
- **Desventajas**: La selección de las muestras puede no siempre ser representativa, lo que podría afectar la precisión del modelo.

Decisión de uso:

En esta etapa, se implementó el reentrenamiento incremental por su eficiencia en términos de recursos y tiempo, lo cual es clave en sistemas de producción que requieren actualizaciones frecuentes con nuevas instancias de datos.

2. Descripción usuario/rol:

El usuario es un ingeniero de sostenibilidad o un especialista en desarrollo sostenible dentro de una organización. Su función principal es alinear las operaciones y proyectos de la organ ización con los Objetivos de Desarrollo Sostenible (ODS).

Conexión con el proceso de negocio:

Este usuario trabaja para garantizar que las actividades y proyectos de la organización sean sostenibles y cumplan con los ODS. La aplicación les ayuda a analizar grandes volúmenes de texto, como informes de sostenibilidad y evaluaciones de impacto, identificando automát icamente los ODS relevantes.

Importancia de la aplicación:

Esta herramienta es crucial para los ingenieros de sostenibilidad, ya que automatiza el análi sis y la clasificación de textos según los ODS. Esto reduce significativamente la necesidad de trabajo manual y agiliza la toma de decisiones basadas en datos.

Beneficios:

El uso de la aplicación permite tomar decisiones más informadas sobre proyectos prioritario s y asignación de recursos. Además, ayuda a asegurar que la organización esté en el camino correcto para cumplir con sus metas de sostenibilidad de manera efectiva y eficiente.

3. Roles asignados:

1. **Líder de proyecto (Andrés Pérez)**

   Responsabilidades: Definió las fechas clave, como reuniones y avances. Adicional a ello, realizó la gestión de asignación de tareas asegurando una carga de trabajo equilibrada entre todos los integrantes. Coordinó las reuniones del equipo y toma de decisiones en situaciones donde no se llegara a un consenso. Por último se aseguró de la subida del proyecto final.

   Tiempo dedicado: 1 hora y media

   Retos enfrentados: Gestionar la disponibilidad de los integrantes y ajustar las tareas cuando algunos miembros enfrentaban dificultades técnicas, como con Google Colab.

   Solución: Se ajustaron las tareas y tiempos para que cada integrante pudiera cumplir con sus responsabilidades.

2. **Ingeniero de datos (Juana Morales):**

   Responsabilidades: realizó la supervisión de la calidad en la automatización del proceso relacionado con la construcción del modelo analítico. Adicional a ello, aportó en el preprocesamiento de los datos para asegurar que el modelo analítico funcione correctamente. Posteriormente, desarrollo de la primera parte del API junto a Andrés. Junto con la implementación de los pipelines necesarios para garantizar un flujo de trabajo eficiente.

   Tiempo dedicado: 2 horas y media

   Retos enfrentados: Problemas con la implementación de la API en Google Colab debido a errores constantes.

   Solución: Optó por cambiar el entorno de trabajo y colaboró en la implementación del API en Visual Studio, mejorando la estabilidad del proyecto.

3. **Ingeniero de software - Diseño de la aplicación y resultados (Andrés Pérez)**

   Responsabilidades: Diseño la aplicación web a nivel general junto a Juana. Adicional a ello, realizó la implementación de la funcionalidad de la API para hacer predicciones basadas en los datos introducidos por los usuarios. También, desarrollo de la interfaz de usuario para la interacción con los resultados del modelo analítico. Y realizó las debidas pruebas que evidenciaran la correcta funcionalidad.

   Tiempo dedicado: 3 horas

   Retos enfrentados: Dificultades técnicas en la integración del API con la interfaz.

Solución: Ajustes en los endpoints y en la estructura del código para garantizar una correcta comunicación entre la API y la aplicación.

4. **Ingeniero de software - Desarrollo de la aplicación final (John Serrato):**

   Responsabilidades: Pulió la parte final de la aplicación, asegurando la funcionalidad completa de la API. También realizó del video de presentación de resultados. Adicionalmente, realizó nuevas pruebas de facilidad de uso para garantizar la fluidez de la interacción con la aplicación. Y por último, hizo una simulación de dos acciones que el usuario final podría realizar con la aplicación, explicando cómo los resultados del modelo aportan a la toma de decisiones.

   Tiempo dedicado: 2 horas y media

   Retos enfrentados: No hubo fallas significativas, solo al inicio hubo pequeños errores en el uso de comandos para la ejecución de app.

   Solución: Se adaptó el entorno de desarrollo a Visual Studio con la instalación de las librerías correspondientes y que funcionará todo adecuadamente.

Reuniones del grupo:

1. **Reunión de lanzamiento y planeación:** se definieron los roles y forma de trabajo, así como brainstorming de ideas para abordar el proyecto. Como resultado, se distribuyeron los roles de acuerdo con las fortalezas de cada integrante, estableciendo plazos internos para los entregables. En total duro aproximadamente 1 hora.
2. **Reunión de seguimiento:** esto lo realizamos vía mensaje (WhatsApp), con el objetivo de revisa e informar avances, identificar obstáculos y ajustar tareas según el progreso de cada integrante. Se realizó de manera constante (a diario).
3. **Reunión de finalización:** se consolidó el trabajo final y se verificó cada parte de la entrega, y se analizaron los puntos de mejora para el próximo proyecto. Tuvo una duración de 2 horas.

Distribución de puntos:

Basándonos en la contribución de cada miembro, la siguiente es la distribución de los 100 puntos del proyecto:

- **Andrés Pérez:** 33,3 puntos
- **Juana Morales:** 33,3 puntos
- **John Serrato:** 33,3 puntos

Reflexión y puntos de mejora:

En futuros proyectos, sería útil dividir más equitativamente el trabajo relacionado con la parte técnica y la generación de resultados, para evitar sobrecargar a ciertos miembros. Resaltando de igual forma, que para esta entrega se trató de mejorar significativamente ello y todo el trabajo fluyó de manera más acertada.