Entrega 2 – Grupo 36

Integrantes:

Nombre: Sergio Oliveros **Código:** 202123159

Nombre: Camilo Daza **Código:** 201416461

Colab del Proyecto: https://colab.research.google.com/drive/1Wr3x-9NunlgCAGevXqzAaL_kVbDU0hEc?usp=sharing

Github del Proyecto: https://github.com/ElSergioOliveros/Proyecto1-BI

Sección 1. (20%) Proceso de automatización del proceso de preparación de datos, construcción del modelo, persistencia del modelo y acceso por medio de API.

La sección 1 de este proyecto consta en la implementación de una API REST con 2 funcionalidades principales. La primera es la automatización del proceso de preparación de datos, que se trata de realizar los preprocesamientos necesarios para poder realizar predicciones sobre uno o más documentos. La segunda trata del reentrenamiento del modelo de análisis utilizado para generar producciones sobre un conjunto de textos. Para cumplir con estos requisitos se utilizó la librería *Flask* para crear los Endpoints que permitiesen el acceso a estas funcionalidades y, para los modelos de análisis, se utilizó *Sklearn*.

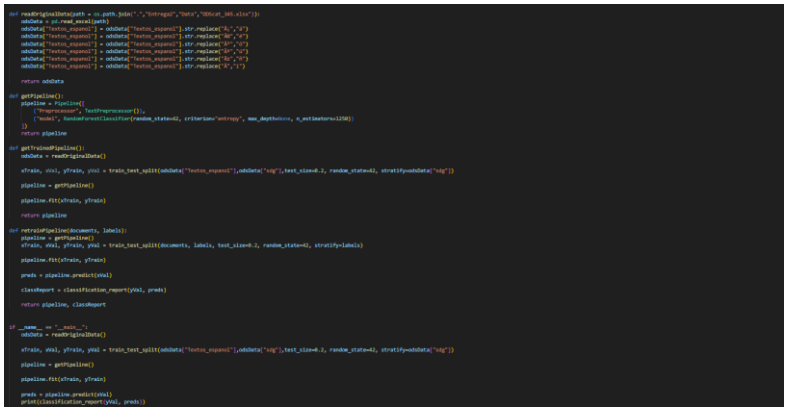
De manera que en conjunto se logra tener el Backend de una aplicación web funcional. A grandes rasgos la forma en que funciona la arquitectura implementada es que se tiene un archivo principal que se encarga de manejar las peticiones que le llegan y, de manejar el estado y la persistencia del modelo de análisis.

a. Automatización del proceso de preparación de datos (pipeline) y documentación:

TextPreprocessor.py

```python
class TextPreprocessor(BaseEstimator, TransformerMixin):
    def __init__(self, min_df=0.017):
        nltk.download('stopwords')
        nltk.download('punkt')
        self.spanish_stopwords = set(stopwords.words('spanish'))
        self.spanish_stopwords.add("ello")
        self.regexStopWords = r'\b(' + '|'.join(self.spanish_stopwords) + r')\b'

        self.stemmer = SnowballStemmer("spanish")
        self.tfidfVectorizer = TfidfVectorizer(min_df=min_df)

    def fit(self, textsDf, y=None):
        textsDf = textsDf.str.replace(self.regexStopWords, "", regex=True)
        textsDf = textsDf.str.replace(r'[^\w\s]', '', regex=True)
        textsDf = textsDf.str.replace(r'\d+|_', '', regex=True)

        stems = textsDf.apply(lambda text : [self.stemmer.stem(token) for token in word_tokenize(text)])

        self.tfidfVectorizer.fit(stems.astype(str))

        return self

    def transform(self, textsDf):
        textsDf = textsDf.str.replace(self.regexStopWords, "", regex=True)
        textsDf = textsDf.str.replace(r'[^\w\s]', '', regex=True)
        textsDf = textsDf.str.replace(r'\d+|_', '', regex=True)

        stems = textsDf.apply(lambda text : [self.stemmer.stem(token) for token in word_tokenize(text)])
        tfidfDocumentsMat = self.tfidfVectorizer.transform(stems.astype(str)).toarray()

        return tfidfDocumentsMat

    def fit_transform(self, textsDf, y=None):
        return self.fit(textsDf).transform(textsDf)
```

PipelineController.py



```python
def readOriginalData(path = os.path.join(".","Entrega1","Data","OSScat_185.xlsx")):
    osbData = pd.read_excel(path)
    osbData["Textos_espanol"] = osbData["Textos_espanol"].str.replace("Ã¡","á")
    osbData["Textos_espanol"] = osbData["Textos_espanol"].str.replace("Ã©","é")
    osbData["Textos_espanol"] = osbData["Textos_espanol"].str.replace("Ã","í")
    osbData["Textos_espanol"] = osbData["Textos_espanol"].str.replace("Ã³","ó")
    osbData["Textos_espanol"] = osbData["Textos_espanol"].str.replace("Ãº","ú")
    osbData["Textos_espanol"] = osbData["Textos_espanol"].str.replace("Ã±","ñ")

    return osbData

def getPipeline():
    pipeline = Pipeline([
        ("Preprocessor", TextPreprocessor()),
        ("model", RandomForestClassifier(random_state=42, criterion="entropy", max_depth=None, n_estimators=1250))
    ])
    return pipeline

def getTrainedPipeline():
    osbData = readOriginalData()

    xTrain, xVal, yTrain, yVal = train_test_split(osbData["Textos_espanol"],osbData["sdg"],test_size=0.2, random_state=42, stratify=osbData["sdg"])

    pipeline = getPipeline()

    pipeline.fit(xTrain, yTrain)

    return pipeline

def retrainPipeline(documents, labels):
    pipeline = getPipeline()
    xTrain, xVal, yTrain, yVal = train_test_split(documents, labels, test_size=0.2, random_state=42, stratify=labels)

    pipeline.fit(xTrain, yTrain)

    preds = pipeline.predict(xVal)

    classReport = classification_report(yVal, preds)

    return pipeline, classReport

if __name__ == "__main__":
    osbData = readOriginalData()

    xTrain, xVal, yTrain, yVal = train_test_split(osbData["Textos_espanol"],osbData["sdg"],test_size=0.2, random_state=42, stratify=osbData["sdg"])

    pipeline = getPipeline()

    pipeline.fit(xTrain, yTrain)

    preds = pipeline.predict(xVal)
    print(classification_report(yVal, preds))
```

Para la automatización del proceso de preparación de datos se crearon dos archivos: *PipelineController.py* y *TextPreprocessor.py.* El primero se encarga de manejar los pipelines que representan los modelos de análisis. Por lo que cada vez se inicia la aplicación se revisa si existe un modelo, en caso de haberlo, dicho modelo se carga. De lo contrario, el *PipeLineController.py* utiliza los datos de la Entrega 1 para crea un modelo que luego es persistido. El segundo es la implementación de una clase para ser utilizada dentro de un Pipeline, donde se hace una extensión e implementación de las funciones que el Pipeline espera para utilizar dicha clase.

Una vez con el modelo, se prosigue a recibir los datos sobre los cuales se quiere hacer una predicción, se les aplican las transformaciones necesarias a través del pipeline, y finalmente se obtiene la probabilidad de pertenencia a cada clase. Resultados que son la respuesta a la petición de sobre la API. Por último, el estado y la persistencia del modelo de análisis se manejan dentro de este archivo principal. Esto asegura que el modelo se mantenga disponible entre las distintas ejecuciones del sistema, evitando la necesidad de recalibrar o reentrenar el modelo con cada nueva solicitud.

b. Reentrenamiento de modelo de análisis:

Para el reentrenamiento del modelo de análisis, se idearon 3 posibles implementaciones:

1. **Reentrenamiento completo:** Esta opción implica que el modelo de análisis resultante se entrena exclusivamente con el nuevo conjunto de datos proporcionado en la petición, ignorando por completo el conjunto de datos original. Se utiliza en situaciones donde el nuevo conjunto de datos es considerablemente diferente del original, lo que impide combinarlos para formar un conjunto de datos mayor.

   La principal ventaja de esta estrategia es que permite al Fondo de Población de las Naciones Unidas entrenar rápidamente modelos de análisis de texto sobre bases de datos muy dispares. No obstante, tiene como desventaja que el reentrenamiento completo es inherentemente lento, ya que el tiempo requerido dependerá del tamaño del nuevo conjunto de datos. Además, se asume que el conjunto de datos previo ya no es relevante, lo que significa que su información no se aprovecha en el proceso.

2. **Reentrenamiento extendido**: Desde la perspectiva de los datos, esta opción implica que los nuevos datos que llegan a través de la petición se utilizan para ampliar el conjunto de datos existente. De esta manera, al entrenar un modelo desde cero sobre este conjunto de datos ampliado, se espera obtener un modelo más robusto, ya que se dispone de más información para el entrenamiento.

   Esta es la segunda opción disponible en la aplicación propuesta y se basa en la premisa de que, a mayor cantidad de datos, mejores serán los resultados del modelo entrenado.

   Entre las ventajas de esta estrategia se destaca que, si los datos son altamente coherentes entre sí, cada reentrenamiento contribuirá a mejorar el rendimiento del modelo. Esto convierte al modelo en un activo valioso para el Fondo de Población de las Naciones Unidas, cuyo valor se depreciará muy poco con el tiempo, e incluso podría aumentar. Sin embargo, una de las desventajas es que esta implementación es extremadamente sensible a la calidad de los datos. Si se reentrena el modelo con datos de baja calidad o que no están bien relacionados con la tarea original, el desempeño del modelo se verá comprometido.

3. **Basado en el concepto de aprendizaje por transferencia proveniente del deeplearning**: Los modelos se entrenan (ya sea de manera supervisada o auto-supervisada) sobre conjuntos de datos inmensos con el objetivo de aprender a capturar las características más relevantes de un conjunto de datos extremadamente diverso. Posteriormente, los

parámetros del modelo se ajustan para especializarlo en una tarea más concreta, pero relacionada con la original.

En nuestro caso, debido a las limitaciones de *sklearn*, no es posible ajustar los parámetros de un modelo que ya ha sido entrenado. Por lo tanto, la alternativa es ajustar los hiperparámetros. Este enfoque consiste en aprovechar los hiperparámetros del mejor modelo entrenado con el conjunto de datos inicial y luego entrenar el modelo únicamente con los nuevos datos. Sin embargo, en lugar de solo entrenar, también se realiza una búsqueda de hiperparámetros para encontrar la mejor configuración posible.

Este enfoque de reentrenamiento asume que los datos originales y los nuevos están correlacionados, aunque no de manera tan estrecha como en el caso de un reentrenamiento extendido. En este caso, los hiperparámetros del modelo anterior sirven como punto de partida, y luego se ajustan para optimizar el rendimiento del nuevo modelo. La ventaja de este método es que permite realizar una búsqueda de hiperparámetros más limitada, partiendo de la premisa de que los hiperparámetros anteriores eran óptimos para la tarea original, lo que implica que los nuevos ajustes estarán cercanos a ellos. No obstante, dado que se incluye una búsqueda de hiperparámetros, la ejecución de este proceso puede ser larga y demorada.

Al evaluar las diferentes alternativas de reentrenamiento, encontramos que la que es más aplicable para un gran número de casos y dadas las restricciones d

c. **Acceso por medio de API:**
   Para facilitar el uso del modelo analítico implementado en la aplicación, se desarrollaron dos endpoints que permiten interactuar de manera directa con el modelo, tanto para realizar predicciones como para reentrenar el modelo con nuevos datos. Estos endpoints aseguran que los usuarios puedan acceder a las funcionalidades clave del sistema de manera eficiente y escalable. Por otro lado, el framework seleccionado fue FLASK, un framework ligero y flexible de Python que es ampliamente utilizado en el desarrollo de APIs REST. El cual cuenta con 2 ventajas principales, su simplicidad, permitiendo un desarrollo rápido y sencillo de aplicaciones web, lo que lo hace ideal para crear APIs con funcionalidades específicas como las necesarias para este proyecto. Y segundo, presenta facilidades modulares que permite añadir o modificar funcionalidades de manera sencilla, lo que facilita el crecimiento y desarrollo de las dos secciones del proyecto.

Sección 2. (40%) Desarrollo de la aplicación y justificación

a. Aplicación y proceso de negocio

La aplicación desarrollada será utilizada principalmente por el equipo o Dirección de temas sociales del Fondo de Poblaciones de las Naciones Unidas (UNFPA), quienes buscan relacionar las opiniones de los ciudadanos con los Objetivos de Desarrollo Sostenible (ODS) 3,4 y 5. El usuario principal de esta aplicación será un analista de datos (con poca experiencia, dada la fluidez de la aplicación y el análisis que la respalda) dentro del UNFPA, cuya responsabilidad es evaluar las opiniones de los ciudadanos y vincularlas con las problemáticas relevantes. Estos analistas podrán automatizar el proceso de clasificación de opiniones, lo que ahorrará tiempo y recursos al evitar el análisis manual.

La importancia de la aplicación radica en que facilita la automatización de este proceso de análisis textual, mejorando la capacidad del UNFPA para tomar decisiones más informadas y rápidas con respecto a sus políticas y proyectos relacionados con los ODS. A través de la aplicación, se reduce el tiempo necesario para evaluar grandes cantidades de datos y se asegura que las decisiones estén alineadas con los problemas identificados por los ciudadanos.

Conexión entre la aplicación y el proceso de negocio:

El proceso de negocio que esta aplicación respalda es la evaluación de opiniones ciudadanas en relación con los ODS. El proceso consiste en la recolección de datos a través de herramientas de participación ciudadana, y la aplicación permite que estas opiniones se clasifiquen automáticamente según su relevancia con los ODS mencionados. Este proceso es clave para que el UNFPA pueda evaluar el progreso hacia el cumplimiento de los objetivos y ajustar sus programas de acuerdo con las necesidades y preocupaciones expresadas por los ciudadanos. Esta aplicación facilitará en gran medida a un analista de datos o persona de negocio con poco entendimiento de datos, el poder darle provecho a la información recogida de opiniones, por medio de este sistema sencillo, obtener clasificados los textos de manera rápida y poder entregar o usar esta información en luego el entendimiento de lo que buscan los ciudadanos en cuanto a los ODS de estudio.

b. Elaboración de la aplicación

Aplicación subida al padlet y al github del proyecto asociado.

c. Relación con el grupo de estadística

Se hicieron ajustes en la tabla de actores, no en los roles, pero si en su beneficio a partir de la aplicación planteada, los cambios principales se presentan a continuación:

- Ciudadanos: Se permite que sus opiniones sean tomadas en cuenta en las decisiones de los gobiernos a nivel internacional, mejorando la

implementación de políticas públicas asociadas a los ODS y que se haga de forma rápida.

- Dirección de temas sociales: El equipo podrá identificar las necesidades de las personas en los 3 ODS, facilitando la priorización de proyectos e implementación de nuevas iniciativas. Esto lo podrá hacer asignando una persona de negocio a tomar los resultados de la aplicación y segmentar por los ODS de investigación.

Sección 3. (18%) Resultados

a. Descripción y visualización de la app

La aplicación cuenta con una Interfaz de Predicción de Texto diseñada para que los usuarios ingresen texto y reciban predicciones basadas en el procesamiento de ese texto. La aplicación tiene los siguientes elementos clave:

1. **Campo de texto para ingresar o pegar el texto:**

El usuario puede escribir o copiar un texto en este campo. Este será el insumo principal que la aplicación utilizará para generar predicciones.

2. **Botón "Enviar para Predicción":**

Al hacer clic en este botón, la aplicación procesa el texto ingresado y devuelve una predicción, junto con una probabilidad asociada, que aparecerá en la tabla debajo.

3. **Tabla de resultados (Texto, Predicción, Probabilidad):**

Aquí se mostrarán los resultados de las predicciones, con tres columnas: el texto ingresado, la predicción realizada por la aplicación, y la probabilidad asignada a esa predicción.

4. **Sección de "Reentrenamiento del Modelo":**

Por otro lado, la plataforma le permitirá una segunda acción al usuario. Esta corresponde al reentramiento del modelo, la cual se usará para darle más información a este y optimizar sus resultados:

Esta parte permite que el usuario cargue un archivo para reentrenar el modelo de predicción.

El usuario también puede seleccionar el tipo de reentrenamiento deseado (en este caso, "Reentrenamiento tipo 2") y luego presionar el botón "Subir y Reentrenar" para actualizar el modelo con nuevos datos.

Sección 4. (10%) Trabajo en equipo

Roles del proyecto:

Líder del proyecto e Ingeniero de software responsable del diseño de la aplicación y resultados: Camilo Daza

Ingeniero de datos e Ingeniero de software responsable del diseño de la aplicación y resultados: Sergio Oliveros

Reuniones

1. **Planeación:** Lunes 30 de Septiembre

El equipo se reunió para llevar a cabo la primera sesión de planeación del proyecto. En esta sesión principalmente nos dedicamos entender la necesidad del cliente en esta segunda etapa y las actividades que se iban a requerir para poder cumplir y llevarlas a cabo. Luego de determinar las actividades, las organizamos por nivel de dificultad y tiempo de implementación y entre los participantes del grupo, dividimos las tareas. Para esta etapa mantuvimos los

roles definidos en la etapa 1 del proyecto, teniendo en cuenta las habilidades de cada integrante y los buenos resultados obtenidos en la primera etapa. De esta manera, Camilo se encargaría de asignar las actividades, enfocarse en las tareas de impacto y conexión con el negocio, planteamiento del front y asegurar el cumplimiento de la entrega planteada al cliente. Al igual, se decidió que Sergio liderara la automatización y conexiones a las APIs para el reentranamiento del modelo y el back necesario para entregar los resultados al cliente.

2. **Seguimiento:** jueves 5 de octubre

El equipo se reunió para solucionar dudas del análisis realizado hasta el momento, apoyarse en puntos de dificultad en la implementación del back y lo que se esperaba para cliente. El líder de datos mostró la mayoría de los avances y se alinearon puntos faltantes de la implementación para asegurar que el cliente tuviera acceso a todo lo que fuera a necesitar.

3. **Finalización:** Sábado 12 de Octubre

El equipo se reunió el sábado para realizar pruebas sobre el sistema y ver que todo estuviera funcionando bien. Se hizo un entendimiento general de las implicaciones sobre stakeholders específicos y se realizó el video de las capacidades del sitio web.

Retos enfrentados por el equipo y solución

En esta segunda etapa, ya se entendía lo que se buscaba con el modelo, los principales retos fueron de implementación. En donde se tuvieron dificultades en los siguientes puntos:

- **Diseño de la API:** El primer reto fue definir claramente los endpoints para garantizar que tanto el front-end como el back-end estuvieran alineados. Teníamos que asegurarnos de que la API proporcionara los datos correctos en el formato adecuado, y que los endpoints estuvieran bien documentados para que el equipo del front pudiera interactuar con ellos sin confusión. **Solución:** Para solucionarlo, decidimos crear dos endpoints clave como se sugería en el enunciado. Uno para recibir los datos y resolver predicciones y el otro para enviar datos en bulk que permitieran actualizar el modelo y actualizar su rendimiento.
- **Error en comunicación de los esperado en el front vs la implementación en el back:** Durante el proceso de integración, hubo momentos en los que el front-end esperaba ciertas respuestas en un formato específico, mientras que el back-end devolvía los datos en un formato diferente, lo que generaba errores. Por ejemplo, el front-end esperaba un objeto JSON con nombres de claves diferentes a los que el back-end estaba utilizando, o el manejo de errores no estaba bien alineado. **Solución:** Como equipo, nos reunimos para coordinarnos en las necesidades de la API, asegurándonos de que ambos equipos

comprendieran exactamente el formato de las solicitudes y las respuestas. Esto conllevó que estuviéramos reunidos durante un largo tiempo de la implementación de la unión, pero se aseguró un buen resultado.

- **Pruebas:** Uno de los principales desafíos fue simular los diferentes escenarios de uso que la aplicación enfrentaría su día a día. Durante las pruebas iniciales, algunas de las solicitudes de datos grandes causaban que la API fallara debido a limitaciones en el tamaño de las cargas de datos que no habíamos previsto. **Solución**: optimizamos la API para manejar de manera más eficiente grandes volúmenes de datos, ajustando la configuración del servidor. Adicionalmente, al final del proyecto una de las personas se dedicó a hacer pruebas y confirmar el correcto funcionamiento