

## Inteligencia de Negocios

### Proyecto 1

#### Etapla 2: Automatización y uso de modelos de analítica de textos

**Proceso de automatización del proceso de preparación de datos, construcción del modelo, persistencia del modelo y acceso por medio de API:**

Se creó un pipeline que transforma la información obtenida utilizando el mismo procesamiento de texto que el de la etapa 1, el cual es el siguiente:

```
def nlp_process(df, text, target):
    def recover(data):
        split = data.split(" ")
        new = []
        for x in split:
            if "Ã" in x:
                new.append(x.encode('latin1', errors='ignore').decode('utf-8', errors='ignore'))
            else:
                new.append(x)
        return " ".join(new).strip()
    def replace_data():
        for key in conversiones:
            data = data.replace(key, conversiones[key])
        return data.strip()
    def no_spaces(data):
        return [x.replace(" ", "") for x in data]
    def stopword_removal(data):
        return [x for x in data if x not in stps and "ã" not in x]
    def number_to_word(data):
        new = []
        for x in data:
            if x.isnumeric():
                new.append("num")
            else:
                new.append(x)
        return new
    def process_word(data):
        return [spanish_stemmer.stem(x) for x in data]
    def concat(data):
        return " ".join(data).strip()
```

Se implementó un procesamiento de lenguaje natural para preparar los datos con el objetivo de limpiarlos, normalizarlos y transformarlos antes de ser utilizados en el modelo de ML. El proceso comienza recuperando caracteres mal codificados (codificados con la letra "Ã") y corrigiéndolos. Luego, convierte el

texto a minúsculas y reemplaza ciertos patrones específicos definidos en el diccionario `conversiones` (por ejemplo, cambia el carácter "á" por el carácter "a"). A continuación, tokeniza el texto (divide en palabras) y elimina espacios dentro de las palabras. Después, remueve las palabras vacías (stopwords) y cualquier palabra que contenga el carácter "ã". Los números son reemplazados por la palabra "num". Posteriormente, se aplica un stemming para reducir las palabras a sus raíces. Finalmente, las palabras procesadas se concatenan de nuevo en una cadena usando la función de Python join.

Usar este procesamiento permite que un texto como este: “Por ejemplo, el nÃºmero de consultas externas”.

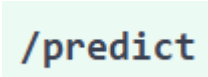
Sea transformado a este: “ejempl numer consult extern”.

Después de transformar la información, se pasa a través de un modelo de clasificación de Random Forest, y retorna el ODS predicho por el modelo.

Este pipeline persiste utilizando la librería joblib, que permite convertir este pipeline en un archivo que se mantiene dentro del repositorio de la aplicación, lo que permite utilizarlo en cualquier momento que sea necesario.

El back de la aplicación tiene un total de cuatro endpoints. Sin embargo, para usar la aplicación solo se necesitan dos (uno para predecir y el otro para reentrenar), pero se ofrecen tres formas de reentrenar el modelo (que se explicarán más adelante) por lo que cada una hace uso de un endpoint distinto.

Endpoint para predecir:



```
@app.post("/predict")
def predict(request: PredictionRequest):
    if len(request.texts) == 0:
        raise HTTPException(status_code=400, detail="No se proporcionaron instancias para predecir.")

    texts_series = pd.Series(request.texts)

    predictions = pipeline.predict(texts_series)
    probabilities = pipeline.predict_proba(texts_series)

    response = []

    for prediction, probability in zip(predictions, probabilities):
        response.append({
            "prediction": int(prediction),
            "probability": probability
        })

    return {"predictions": predictions.tolist(),
            "probabilities": probabilities.tolist(),
            "classes": pipeline.classes_.tolist()}
```

Este endpoint solo espera un body con el atributo “texts”, el cual es una lista de los textos a predecir.

Este es un ejemplo de body:

```
{
  "texts": [
    "La educación en Colombia es importante", "Las mujeres son un pilar fundamental de la sociedad", "Los hospitales y clínicas requieren de más financiación"
  ]
}
```

Este es un ejemplo de la respuesta del back:

```
{
  "predictions": [
    4,
    5,
    3
  ],
  "probabilities": [
    [
      0.15,
      0.69,
      0.16
    ],
    [
      0.11333333333333333,
      0.02,
      0.8666666666666667
    ],
    [
      0.78,
      0.13666666666666666,
      0.08333333333333331
    ]
  ],
  "classes": [
    3,
    4,
    5
  ]
}
```

Como se puede apreciar, retorna una lista de las predicciones para cada texto, también retorna una lista de las probabilidades de cada texto y por último retorna la lista de clasificaciones posibles del modelo, las cuales en orden corresponden con los valores de probabilidad.

En la aplicación esta información se muestra de la siguiente manera:

Texto	Predicción	Probabilidades
La educación en Colombia es importante		 3 4 5
Las mujeres son un pilar fundamental de la sociedad		 3 4 5
Los hospitales y clínicas requieren de más financiación		 3 4 5

## Endpoints de reentrenamiento:

Se implementaron tres endpoints distintos que representan tres alternativas de reentrenamiento:

- Entrenamiento completo: Los datos ingresados se agregan al total de datos que posee el modelo.
- Entrenamiento independiente: El modelo se entrena solo con los datos ingresados
- Entrenamiento parcial: Los datos ingresados se agregan a una mitad aleatoria de los datos que posee el modelo.

Los body de entrada y salida son iguales, lo que cambia es el proceso interno a la hora de reentrenar el modelo. Como body se esperan dos listas, una lista “texts” que posee los textos a predecir y una lista “labels” con los labels de cada texto.

```
@app.post("/retrain/completo")
def retrain(request: RetrainingRequest):
    if len(request.texts) == 0 or len(request.labels) == 0:
        raise HTTPException(status_code=400, detail="No se proporcionaron instancias para reentrenar.")
    if len(request.texts) != len(request.labels):
        raise HTTPException(status_code=400, detail="El número de textos y etiquetas no coincide.")

    new_data = pd.DataFrame({'Textos_espanol': request.texts, 'sdg': request.labels})

    existing_data = pd.read_excel(train_data_path)

    combined_data = pd.concat([existing_data, new_data], ignore_index=True)

    combined_data.to_excel(train_data_path, index=False)

    X_train = combined_data['Textos_espanol']
    y_train = combined_data['sdg']

    pipeline.fit(X_train, y_train)

    test_data = pd.read_excel(test_data_path)
    X_test = test_data['Textos_espanol']
    y_test = test_data['sdg']

    predictions = pipeline.predict(X_test)
    precision = precision_score(y_test, predictions, average='weighted')
    recall = recall_score(y_test, predictions, average='weighted')
    f1 = f1_score(y_test, predictions, average='weighted')
    accuracy = accuracy_score(y_test, predictions)

    joblib.dump(pipeline, pipeline_path)

    return {
        "precision": round(precision, 2),
        "recall": round(recall, 2),
        "f1_score": round(f1, 2),
        "accuracy": round(accuracy, 2)
    }
```

**/retrain/completo**

## /retrain/independiente

```
@app.post("/retrain/independiente")
def retrain_independiente(request: RetrainingRequest):
    if len(request.texts) <= 1 or len(request.labels) <= 1:
        raise HTTPException(status_code=400, detail="No se proporcionaron suficientes instancias para reentrenar.")
    if len(request.texts) != len(request.labels):
        raise HTTPException(status_code=400, detail="El número de textos y etiquetas no coincide.")

    new_data = pd.DataFrame({'Textos_espanol': request.texts, 'sdg': request.labels})
    new_data.to_excel(train_data_path, index=False)
    X_train = new_data['Textos_espanol']
    y_train = new_data['sdg']

    pipeline.fit(X_train, y_train)

    test_data = pd.read_excel(test_data_path)
    X_test = test_data['Textos_espanol']
    y_test = test_data['sdg']

    predictions = pipeline.predict(X_test)
    precision = precision_score(y_test, predictions, average='weighted')
    recall = recall_score(y_test, predictions, average='weighted')
    f1 = f1_score(y_test, predictions, average='weighted')
    accuracy = accuracy_score(y_test, predictions)

    joblib.dump(pipeline, pipeline_path)

    return {
        "precision": round(precision, 2),
        "recall": round(recall, 2),
        "f1_score": round(f1, 2),
        "accuracy": round(accuracy, 2)
    }
```

## /retrain/parcial

```
@app.post("/retrain/parcial")
def retrain_parcial(request: RetrainingRequest):
    if len(request.texts) == 0 or len(request.labels) == 0:
        raise HTTPException(status_code=400, detail="No se proporcionaron instancias para reentrenar.")
    if len(request.texts) != len(request.labels):
        raise HTTPException(status_code=400, detail="El número de textos y etiquetas no coincide.")

    new_data = pd.DataFrame({'Textos_espanol': request.texts, 'sdg': request.labels})

    existing_data = pd.read_excel(train_data_path)

    historical_sample = existing_data.sample(frac=0.5, random_state=42)

    combined_data = pd.concat([historical_sample, new_data], ignore_index=True)

    combined_data.to_excel(train_data_path, index=False)

    X_train = combined_data['Textos_espanol']
    y_train = combined_data['sdg']

    pipeline.fit(X_train, y_train)

    test_data = pd.read_excel(test_data_path)
    X_test = test_data['Textos_espanol']
    y_test = test_data['sdg']

    predictions = pipeline.predict(X_test)
    precision = precision_score(y_test, predictions, average='weighted')
    recall = recall_score(y_test, predictions, average='weighted')
    f1 = f1_score(y_test, predictions, average='weighted')
    accuracy = accuracy_score(y_test, predictions)

    joblib.dump(pipeline, pipeline_path)

    return {
        "precision": round(precision, 2),
        "recall": round(recall, 2),
        "f1_score": round(f1, 2),
        "accuracy": round(accuracy, 2)
    }
```

Como se aprecia en los códigos, el retorno son 4 métricas de desempeño del nuevo modelo, calculadas bajo unos datos etiquetas independientes de los de entrenamiento los cuales son siempre los mismos.

Ventajas y desventajas de cada tipo de reentrenamiento:

Método de Entrenamiento	Ventajas	Desventajas
Entrenamiento completo	- Aprovecha todo el historial de datos para ajustarse mejor a patrones a largo plazo.	- Consume más tiempo y recursos (computación, almacenamiento).
	- Aprende de la diversidad de opiniones pasadas y presentes.	- Problemas de escalabilidad a medida que el conjunto de datos crece.
	- Incrementa el contexto, especialmente en temas complejos como los ODS.	- Riesgo de sobreajuste si los datos anteriores pesan demasiado.
Entrenamiento independiente	- Proceso más rápido y menos costoso en términos de recursos computacionales.	- Pierde el contexto acumulado de datos históricos, importante en análisis a largo plazo.

	- Se ajusta rápidamente a nuevas tendencias y opiniones actuales.	- Menor estabilidad si los nuevos datos no son representativos.
	- Reduce la complejidad de manejar grandes volúmenes de datos históricos.	- Riesgo de desvirtuar el modelo con datos atípicos o sesgados.
<b>Entrenamiento parcial</b>	- Balancea el aprendizaje de nuevos datos sin perder el historial previo.	- Resultados menos consistentes debido a la aleatoriedad en la selección de datos antiguos.
	- Menos intensivo en computación que el entrenamiento completo.	- No aprovecha toda la información histórica, lo que puede reducir el rendimiento.
	- Reduce el riesgo de que los nuevos datos dominen el modelo.	- Inconsistencia si las mitades seleccionadas varían en calidad o representatividad.

Dado el contexto del UNFPA y el objetivo de relacionar las opiniones ciudadanas con los ODS, el entrenamiento completo sería el más robusto en cuanto a aprovechar todo el contexto de datos, lo que es crucial para un análisis que debe considerar información a largo plazo y diversidad de opiniones. Sin embargo, si la agilidad es una prioridad, el entrenamiento independiente podría ser útil en casos donde se necesite rapidez para ajustar el modelo a nuevas opiniones, mientras que el entrenamiento parcial ofrece un balance entre ambas opciones.

La información de entrenamiento se persiste en formato Excel dentro del repositorio del back.

### **Desarrollo de la aplicación y justificación:**

La UNFPA y las entidades públicas serán las que harán uso de la aplicación. Ya que estas organizaciones están interesadas en la identificación de problemas y la evaluación las soluciones actuales, podrán mediante la aplicación, relacionar la información dada por los ciudadanos con los diferentes Objetivos del Desarrollo Sostenible.

El objetivo de la UNFPA y las demás entidades es automatizar este proceso donde se relacionan las opiniones de los ciudadanos con los ODS 3, 4 y 5. La aplicación desarrollada resuelve esta necesidad y ofrece a los interesados la posibilidad de identificar aquellos ODS que resultan más problemáticos en una determinada región.

La existencia de la aplicación impacta directa y positivamente a las entidades involucradas, ya que les permite concentrar sus esfuerzos y recursos en aquellos problemas que requieren una atención más prioritaria, mediante políticas públicas más concretas y eficientes. De esta forma, se impacta también la

calidad de vida de aquellas personas que se benefician de la implementación de estas políticas.

### Resultados:



*Foto 1 Foto de uno de los usuarios de prueba usando la aplicación*

Las pruebas se llevaron a cabo involucrando a 3 personas cercanas, quienes cumplieron el perfil de usuario final para la aplicación. Se evaluaron los siguientes aspectos:

- Facilidad de uso: Intuición y simplicidad de la interfaz, tiempo requerido para aprender a utilizar la aplicación y navegación fluida.
- Utilidad: Capacidad de la aplicación para cumplir con las tareas descritas en el enunciado del proyecto.

#### Facilidad de Uso:

Los participantes destacaron que la aplicación es fácil de usar y tiene una interfaz intuitiva. En general, la curva de aprendizaje fue mínima, y los usuarios pudieron utilizar las dos funcionalidades principales sin problemas. La navegación fue fluida y no se presentaron confusiones ni errores de compilación al interactuar con las distintas funcionalidades.

- Puntuación media: 9/10

- Comentarios: Se sugirió la implementación de otros tipos de archivos para leer los datos (actualmente solo acepta formato Excel).

#### Utilidad:

La aplicación demostró ser útil para los objetivos planteados, cumpliendo con las expectativas de los usuarios finales. Los participantes lograron realizar predicciones y reentrenar el modelo de manera eficiente, resaltando que las funcionalidades clave de la aplicación respondieron a sus necesidades.

- Puntuación media: 8.5/10
- Comentarios: Los usuarios expresaron que la aplicación podría explicar mejor los resultados que arroja la página al realizar el entrenamiento, ya que un usuario sin conocimientos de Machine Learning podría no poder interpretarlos propiamente.

Las pruebas realizadas fueron exitosas. La aplicación cumplió con los criterios de facilidad de uso y utilidad esperados, lo que sugiere que está lista para su implementación o continuación del desarrollo por parte de UNFPA para implementar las sugerencias de mejora que arrojaron las pruebas.

#### **Trabajo en equipo:**

##### **Roles:**

**Nicolás Ruíz:** Líder del proyecto e ingeniero de software.

**Harold Nicolás Coca:** Ingeniero de datos.

Tareas realizadas con tiempos de cada uno y la distribución:

##### **Nicolás Ruiz Pérez:**

- Diseñar los mockups de la aplicación (3 horas).
- Implementar los mockups en JavaScript (5 horas).
- Hacer la conexión entre la aplicación web y el back (30 minutos).
- Implementar la lógica (lecturas de archivos, manejo de listas, etc.) en el front (2 horas).

##### **Harold Nicolás Coca:**

- Transformar la preparación de los datos de la entrega 1 del proyecto en un pipeline. (4 horas)
- Implementación de clases auxiliares para la preparación de los datos. (2 horas)
- Construcción e implementación del API usando el framework FAST API. (3 horas)

- Implementación de las tres opciones de reentrenamiento del modelo. (2 horas)

Retos enfrentados:

- Aprender nuevas tecnologías a la hora de hacer el front (lectura de archivos, librerías de componentes de front)

Repartición de 100 puntos:

- Nicolás Ruiz: 50 puntos.
- Nicolás Coca: 50 puntos.

Puntos por mejorar:

- Realizar más reuniones para aclarar dudas del enunciado.
- Comunicar los avances de cada integrante más seguido.