PROYECTO ETAPA 2

Automatización y uso de modelos de analítica de textos

ISIS3301

Juan Camilo Beltrán Garnica
Leidy Johana Lozano Florez
Juan Andrés Reyes

Índice

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

Para realizar el proceso de automatización del modelo analítico y acceso por medio de la Api se realizaron dos etapas:

Modificación de la exportación del modelo original.

Para poder exportar el modelo automatizado de forma apropiada, se tuvo que modificar el pipeline en el Jupyter Notebook de exportación original, se crearon clases personalizadas para realizar el preprocesamiento necesario del texto antes de ajustarlo al modelo y se añadieron al pipeline con el objetivo de que el modelo generado fuera capaz de realizar automáticamente todo el procesamiento

de los datos y dar un resultado una vez fuera cargado en el API. También se realizaron algunos cambios a los métodos de preprocesamiento de datos para mejorar la calidad del modelo, en particular, las modificaciones en el proceso de Stemming permitieron obtener métricas de calidad mucho mejores para el modelo exportado.

Implementación de la API.

La API fue construida usando el framework FastAPI y se configuró un ambiente virtual con todas las dependencias necesarias para ejecutarlo. La API consta de dos endpoints:

Endpoint 1

Este primero es el encargado de realizar predicciones, por lo que se accede con la ruta */predict/* . Recibe del front un archivo Excel que contiene los registros que se desean predecir. En el API, la función asociada a la ruta genera a partir del archivo subido un DataFrame con solo una columna "Textos_espanol", luego carga el modelo a partir de un archivo "model.joblib" y realiza la predicción con los datos de entrada. El API retorna un diccionario con el siguiente formato JSON:

```
{
   "prediction": [
      3,
      4,
      5
   ]
}
```

Esta respuesta contiene una lista de todas las predicciones hechas en orden para cada texto de entrada.

Endpoint 2

Este tiene la función de reentrenar el modelo con nuevos datos etiquetados. Se accede con la ruta */retrain/* . Recibe del front un archivo de Excel que contiene los registros etiquetados con los que se desea reentrenar el modelo. En el API, la función asociada a la ruta genera a partir del archivo subido un DataFrame con dos columnas, "Textos_espanol" y "sdg" y realiza el entrenamiento del modelo con el mismo proceso con el que se exporto el modelo original. Al terminar, se exporta el nuevo "model.joblib" y el API retorna en formato JSON un diccionario con todas las métricas de calidad asociadas al entrenamiento del modelo similar al siguiente:

```
{'label 1': {'precision':0.5,
             'recall':1.0,
             'f1-score':0.67,
             'support':1},
 'label 2': { ... },
  ...
}
```

Tipos de reentramiento

- Adición a los datos originales: Una forma de reentrenar el modelo puede ser que adicionar los datos que lleguen al API a los datos ya existentes y reentrenar el modelo con los datos completos.

    - Caso de aplicación: La organización ha estado guardando datos entrantes por un periodo de tiempo y desea entrenar el modelo para que tenga en cuenta los datos antiguos y los nuevos.
    - Ventaja: Permite que el modelo sea actualizable en el tiempo para casos de aplicación en los que se recopilan nuevos datos constantemente.
    - Desventajas: No permite eliminar datos antiguos del modelo, solo añadir nuevos.

- Reentrar para otra tarea (Transfer Learning): Otra forma de reentrenar el modelo es reutilizarlo para una tarea similar, sin embargo, creemos que este método de reentramiento se sale de los objetivos de la institución, además que requerirá mucha información adicional que el Api no fue diseñado para manejar.

    - Caso de aplicación: La organización ya no desea clasificar comentarios sino correos.
    - Ventajas: Permite que el modelo adquiera nuevas funcionalidades y sea más útil para la organización en otros ámbitos.
    - Desventajas: Elimina completamente el modelo antiguo, por lo que puede no llegar a ser practico. Las aplicaciones adicionales pueden ser muy limitadas.

- Reentrenar solo con los datos nuevos (elegido): Este método fue el que decidimos implementar, consta de solo utilizar los datos nuevos de llegada para entrenar el nuevo modelo, la razón por la que se decidió implementar este es porque es también una implementación del primer método, adición a los datos originales, ya que el usuario puede adicionar los datos nuevos a los antiguos manualmente y luego reentrenar el modelo con estos.

    - Caso de aplicación: La organización realizo cambios estructurales a sus comentarios o a los objetivos de desarrollo sostenible, así que necesita solo entrenar el modelo con datos adquiridos desde que se realizaron los cambios.
    - Ventajas: Permite que el modelo se pueda entrenar desde 0 con una lógica de negocio diferente. Permite que se adicionen mas datos.
    - Desventajas: El usuario o la organización siempre debe mantener una copia local de los datos antiguos si no desea perder esta información, tienen que realizar adiciones de información manualmente a los archivos que desean subir.

Sección 2. Desarrollo de la aplicación y justificación

El rol dentro de la UNFPA que haría uso de esta aplicación sería el de Coordinador/a de respuesta humanitaria, que normalmente tiene la responsabilidad de leer los comentarios y asignarlos a un objetivo de desarrollo de manera manual. Esta tarea puede tomar mucho tiempo, ya que el volumen de comentarios que recibe la organización puede llegar a ser bastante alto, por lo tanto, la aplicación es de vital importancia para su rol ya que puede significar un gran aumento en su rendimiento y en general puede ser una gran ventaja para la organización no tener que dedicar mucho tiempo a esta tarea.

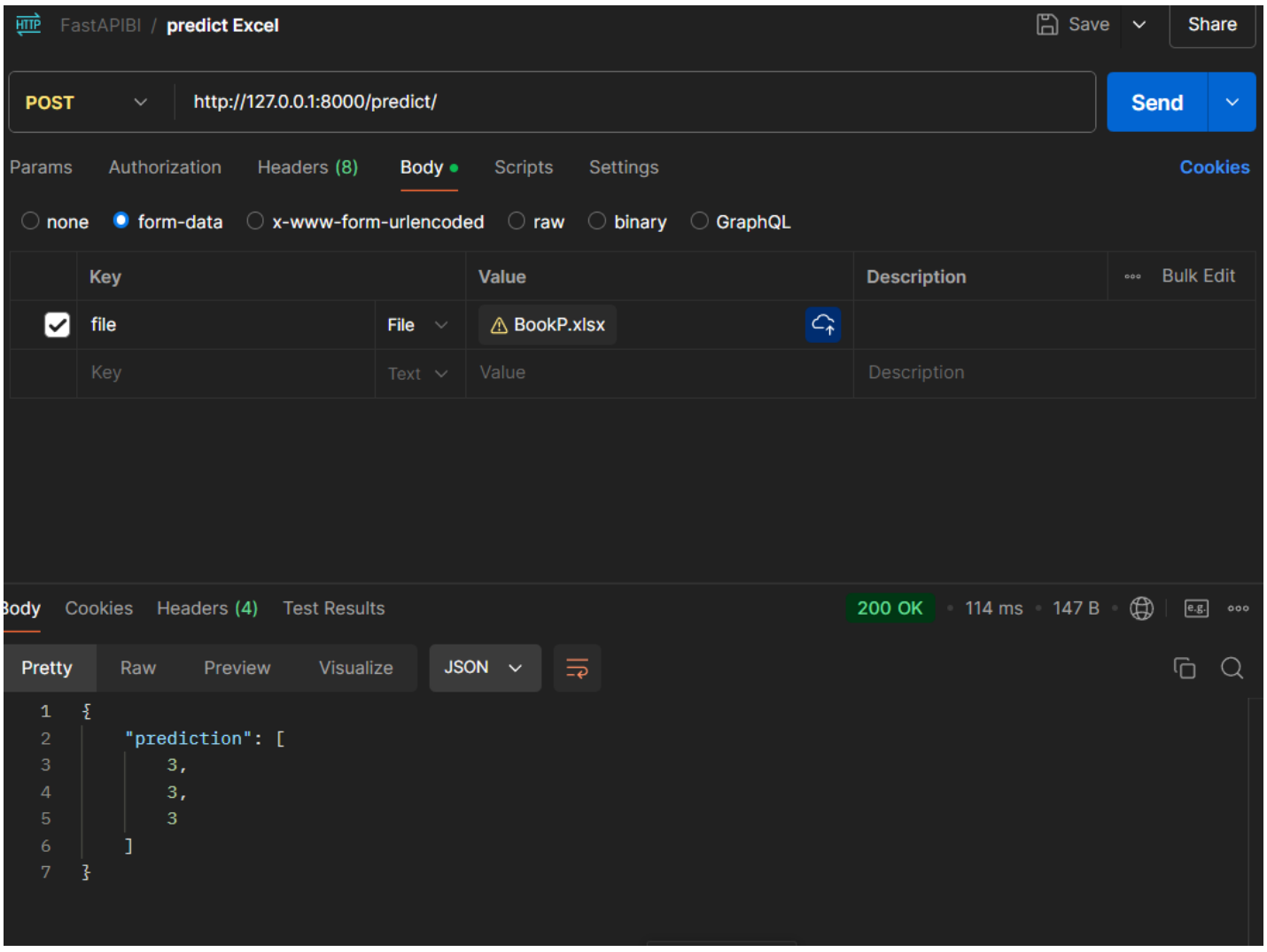
En el caso del endpoint 2, ninguno de los actores que consultamos para la etapa 1 haría uso de él, ya que estos están enfocados en los objetivos de desarrollo 3, 4 y 5. Consideramos que los textos no van a cambiar su estructura, por lo que el modelo original sigue siendo de ayuda para los actores.

La aplicación web se decidió hacer sobre el *framework* React, el cual le permitió al equipo un desarrollo sencillo, fácilmente enlazable con los endpoints de la API, esto se realiza a través de la asincronía de JavaScript. Se decidió que el usuario pudiera subir los datos directamente en formato Excel, ya que es típicamente un formato muy manejado y es muy intuitivo para el usuario ya que implica que no tiene que subir la información manualmente o cambiar el formato del archivo en el que ya tiene la información.

Internamente, la API se encarga de convertir el archivo de Excel seleccionado en un archivo JSON. Posteriormente, se lleva a cabo la predicción o el reentrenamiento según sea el caso. Finalmente, la respuesta es enviada como un JSON a la interfaz, la cual se encarga de organizar la información en tablas.

Sección 3. Resultados

Se realizaron pruebas por medio de Postman al API con archivos de Excel de prueba que, las cuales daban resultados correctos:



En este caso, se envió un Excel de prueba con tres textos distintos. En este caso, el modelo predijo que todos los archivos eran del ODS 3. Cabe aclarar que, para propósitos de la exposición de resultados, se usó un archivo con información no relevante. En la aplicación web, se evidencia la siguiente respuesta.

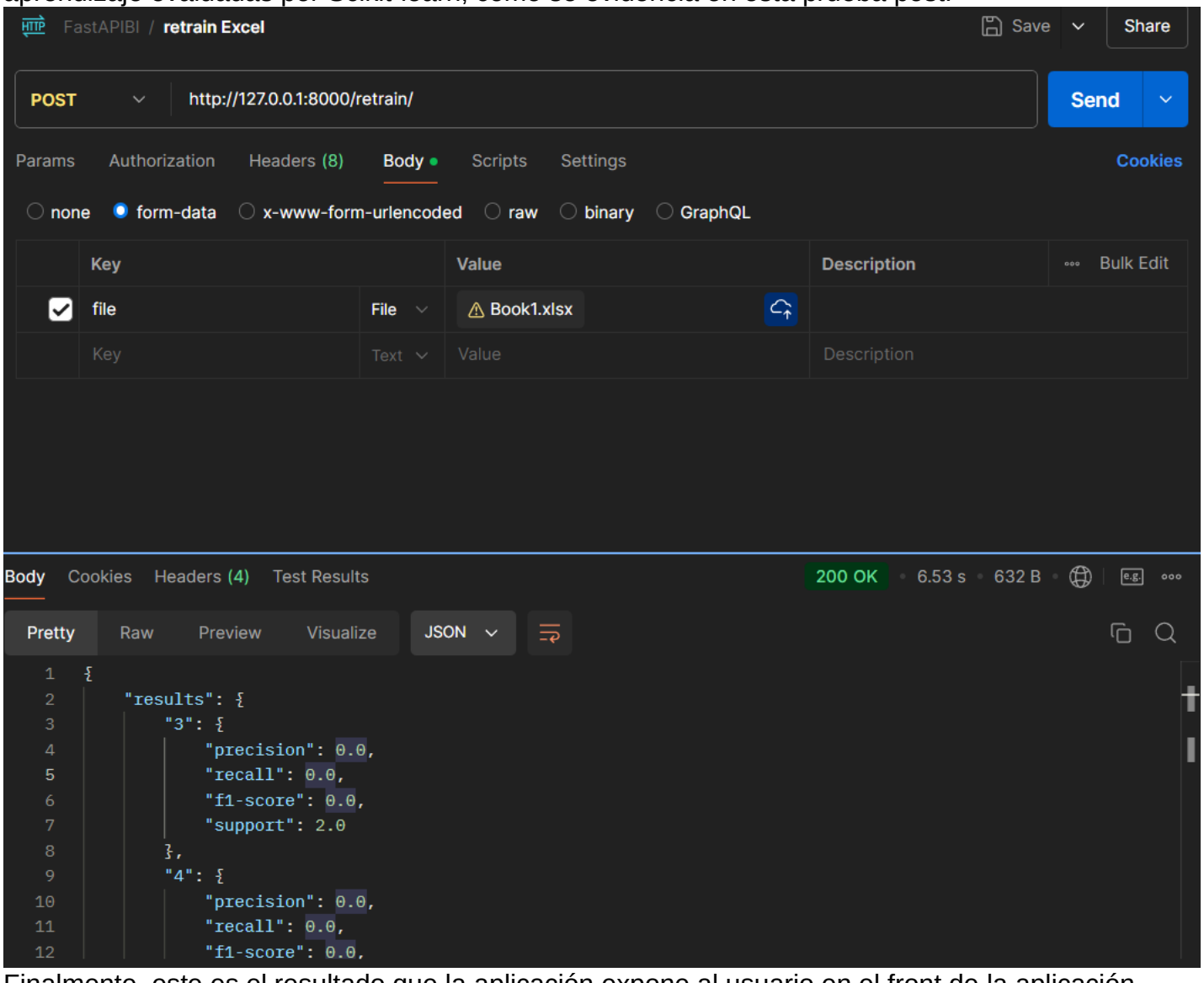
Predecir una muestra nueva de información

Carga un nuevo archivo

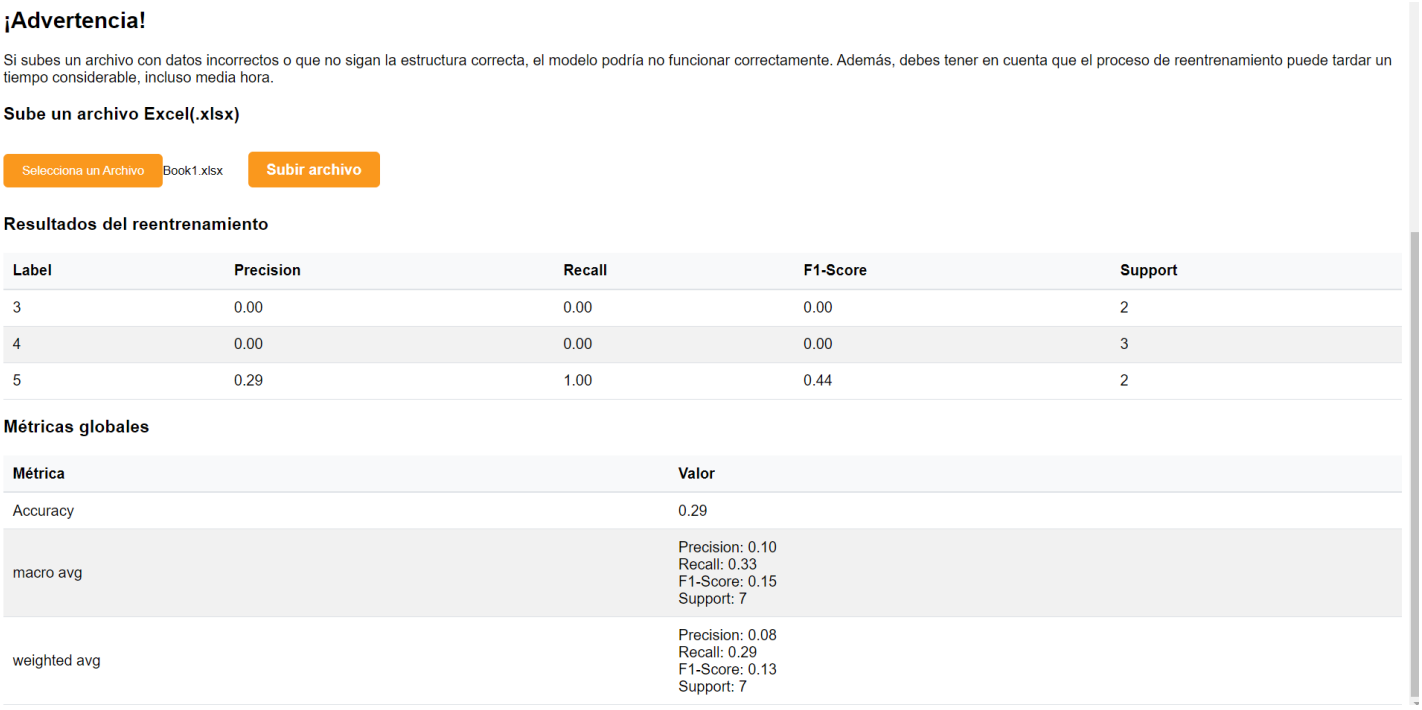
Choose File BookP.xlsx Subir archivo

Textos_espanol	Predicción
texto1	5
texto2	5
texto3	5
texto1	5
texto2	5
texto3	5
texto1	5
texto2	5
texto3	5
texto1	5
texto2	5
texto3	5
texto1	5

Para el segundo endpoint, se realiza el reentrenamiento del modelo y se retornan las métricas de aprendizaje evaluadas por Scikit-learn, como se evidencia en esta prueba post.



Finalmente, este es el resultado que la aplicación expone al usuario en el front de la aplicación.



¡Advertencia!

Si subes un archivo con datos incorrectos o que no sigan la estructura correcta, el modelo podría no funcionar correctamente. Además, debes tener en cuenta que el proceso de reentrenamiento puede tardar un tiempo considerable, incluso media hora.

Sube un archivo Excel(.xlsx)

Selecciona un Archivo Book1.xlsx Subir archivo

Resultados del reentrenamiento

Label	Precision	Recall	F1-Score	Support
3	0.00	0.00	0.00	2
4	0.00	0.00	0.00	3
5	0.29	1.00	0.44	2

Métricas globales

Métrica	Valor
Accuracy	0.29
macro avg	Precision: 0.10 Recall: 0.33 F1-Score: 0.15 Support: 7
weighted avg	Precision: 0.08 Recall: 0.29 F1-Score: 0.13 Support: 7

Sección 4. Trabajo en equipo

Nombre integrante y rol	Tareas Asignadas	Tiempo dedicado	Retos encontrados	Solución Propuesta	Puntos Asignados
Juan Camilo Beltrán – Ingeniero de datos	- Realización del API. - Análisis del usuario - Descripción del API en el documento - Descripción de los tipos de reentramiento.	6 Horas	- Hubo muchos problemas con el manejo de librerías y dependencias . - Fue un poco difícil encontrar un tercer método de reentramiento y luego elegir. - La comunicación entre el front y el API presento muchos desafíos de formato.	- Usando múltiples herramientas se corrigieron todos los errores de dependencias. - Se encontró un tercer método sin problemas. - Se decidió que el front y el API se comunicaran por medio de envíos de archivos Excel.	33
Andrés Reyes – Responsable del diseño de la aplicación y resultados	- Realización de Presentación y Video - Complementación de la wiki del proyecto	5 horas	- Hubo muchos errores con las dependencias en las apps, tanto API como frontend - Errores con el entorno virtual	- Revisar poco a poco qué dependencias hacían falta para añadirlas requirements. txt - Buscar a qué errores se referían los errores y modificar	33

				permisos para scripts	
Leidy Lozano – Líder de proyecto, Responsable de desarrollar la aplicación final	- Corrección del modelo de calificación de la entrega 1. - Generación del pipeline. - Desarrollo del frontend de la aplicación y su conexión con FastApi.	7 horas	- No se sabía cómo encapsular el proceso de preparación de los datos en el pipeline. - El desarrollo del frontend fue un poco complicado, porque si bien la parte gráfica estaba cubierta, no se tenía muy claro el manejo de promesas y asincronía de JS.	- Se consulto con un compañero del curso que contaba con mayor entendimiento del pipeline. - Se hizo el mejor esfuerzo por realizar la parte gráfica de forma estática y luego se consultó ayuda con otros compañeros para completar la asincronía.	34