Documento proyecto 1: Etapa 2

Manuel Felipe Carvajal - 2020140203

Juanita Gil - 202111556

José David Martínez – 202116677

Sección 1

1.1. Diagnóstico y motivación

En la evaluación inicial sobre los datos de la Etapa 2 observamos un desbalance significativo en la clase “1” (ODS 1: Fin de la pobreza), con 21 ejemplos frente a 35 y 43 de las clases 3 y 4 respectivamente.
Aunque el modelo base (SVM lineal + TF-IDF 1–2 gramas) mostraba un macro-F1 de 0.97, el recall específico de la clase minoritaria era inferior al 0.90, evidenciando la

necesidad de una estrategia de aumentación dirigida. El objetivo fue mejorar la sensibilidad hacia los textos sobre pobreza sin degradar la precisión global.

1.2. Estrategia de aumentación con OpenAI API

Para reforzar la representación semántica de la clase 1, implementamos una aumentación sintética mediante prompting programático con la API de OpenAI. Se empleó el modelo GPT-4o-mini y una clave institucional inyectada a través de la variable de entorno OPENAI_API_KEY (cargada con dotenv para no exponer secretos). El script solicitó 30 opiniones verosímiles y variadas en español sobre pobreza, desigualdad económica y acceso a recursos, usando un prompt con restricciones claras: frases cortas (1–2), sin lenguaje literario ni estructuras repetidas.

El resultado fue un conjunto de frases sintéticas numeradas que se extrajeron mediante expresiones regulares, se normalizaron (trim, len > 5), se eliminaron duplicados y se etiquetaron con label = "1". Posteriormente se concatenaron con el dataset real de Etapa 2 (Datos_etapa2.csv), obteniendo un corpus aumentado con ~50 instancias de la clase 1 y 120 instancias totales.
El nuevo conjunto se guardó como data/Datos_etapa2_aumentado.csv.

1.3. Reentrenamiento del SVM y validación

El reentrenamiento se realizó cargando el pipeline original desde artifacts/best_model.joblib. Dado que se trataba de un Pipeline (TFIDFVectorizer + SVC), fue posible aplicar fit() directamente sobre el dataset combinado. El modelo reentrenado se persistió como artifacts/model_v2.pkl. Para medir el impacto, se ejecutó la evaluación sobre el mismo conjunto de prueba de la Etapa 2, calculando precision, recall, F1 y accuracy macro, además del classification report por clase y la matriz de confusión.

1.4. Resultados comparativos

Los resultados mostraron una mejora visible en la clase minoritaria: el recall de ODS 1 aumentó en +0.06 puntos y el F1-macro global subió ≈ +0.02, manteniendo la accuracy total por encima del 0.97.
Las métricas se registraron en artifacts/aug_metrics.json, mientras que artifacts/baseline_metrics.json contiene la línea base. La comparación se resume en artifacts/comparacion_macro.csv, evidenciando la ganancia en recall y F1 sin deterioro de la precisión. La siguiente tabla fue usada como referencia en el informe técnico y para alimentar la visualización de la aplicación web.

Métrica	Antes	Después	Mejora
Precision	0.975	0.981	+0.006
Recall	0.972	0.987	+0.015

F1-macro	0.974	0.988	+0.014
Accuracy	0.977	0.981	+0.004

1.5. Consideraciones de control y calidad

Durante la generación sintética se validó que las nuevas frases no provinieran de los conjuntos de validación ni repitieran patrones evidentes (“marcas” de IA). Asimismo, se auditaron manualmente muestras aleatorias para asegurar naturalidad y coherencia temática. El pipeline de reentrenamiento incluye manejo de excepciones ante vocabularios vacíos o lotes pequeños (fallback de min_df = 1 y ngram_range = (1, 2)), garantizando reproducibilidad. Con estos ajustes, la Etapa 2 alcanzó una representación balanceada de los ODS y un modelo más robusto para su despliegue en la API.

Sección 2

2.1. Pipeline de modelado

El servicio utiliza el mejor modelo de la Etapa 1: un SVM lineal entrenado sobre representaciones TF-IDF de los textos. Antes de vectorizar, se normaliza el corpus (tipado a string, eliminación de NaN, strip de espacios y descarte de textos muy cortos) y luego se transforma con TfidfVectorizer usando n-gramas de 1 a 2. El clasificador se entrena con probability=True para poder exponer, cuando esté disponible, probabilidades por clase. El modelo final se persiste en artifacts/model_v2.pkl para su carga al iniciar la API.

Como la Etapa 2 incorpora reentrenamiento continuo, el pipeline contempla escenarios de datos pequeños o ruidosos. Si al reentrenar aparece el error de TF-IDF o un vocabulario vacío, se activa un fallback seguro que ajusta min_df=1, max_df=1.0, ngram_range=(1,2) y desactiva stopwords específicas, garantizando que el pipeline siempre pueda ajustarse incluso con lotes pequeños. Este diseño permite mantener los hiperparámetros y, solo cuando es estrictamente necesario, reconfigurar el vectorizador para no bloquear la operación.

2.2. API con FastAPI

La API se construyó con FastAPI, exponiendo tres endpoints: /health, /predict y /retrain. Al iniciar, la aplicación carga el modelo desde artifacts/model_v2.pkl y deja disponibles los metadatos básicos. El endpoint **/predict** recibe una lista de objetos con la clave textos y devuelve, para cada entrada, la clase predicha.

El endpoint **/retrain** habilita un ciclo de aprendizaje incremental. Toma una lista de pares {textos, labels}, los agrega al almacén data/training_store.csv y reentrena el pipeline con la unión de tres fuentes: training_store.csv (acumulado por API), datosetapa2.xlsx (generación de la Etapa 2) y Datos_proyecto.xlsx (base original de la Etapa 1). La respuesta incluye métricas macro (precision, recall, f1, accuracy) calculadas sobre el conjunto total usado en ese entrenamiento, el número de instancias, la distribución por clase y un campo nuevo_clasificados que resume cuántos ejemplos por clase llegaron en el lote enviado, útil para auditar el impacto de cada reentrenamiento.

2.3. Resultados de pruebas

Las pruebas con Postman verificaron el comportamiento esperado. En /predict, tres textos de ejemplo fueron clasificados coherentemente con su contenido: uno sobre educación rural fue etiquetado como “4”, otro sobre falta de médicos como “3” y uno sobre ingresos inestables como “1”.

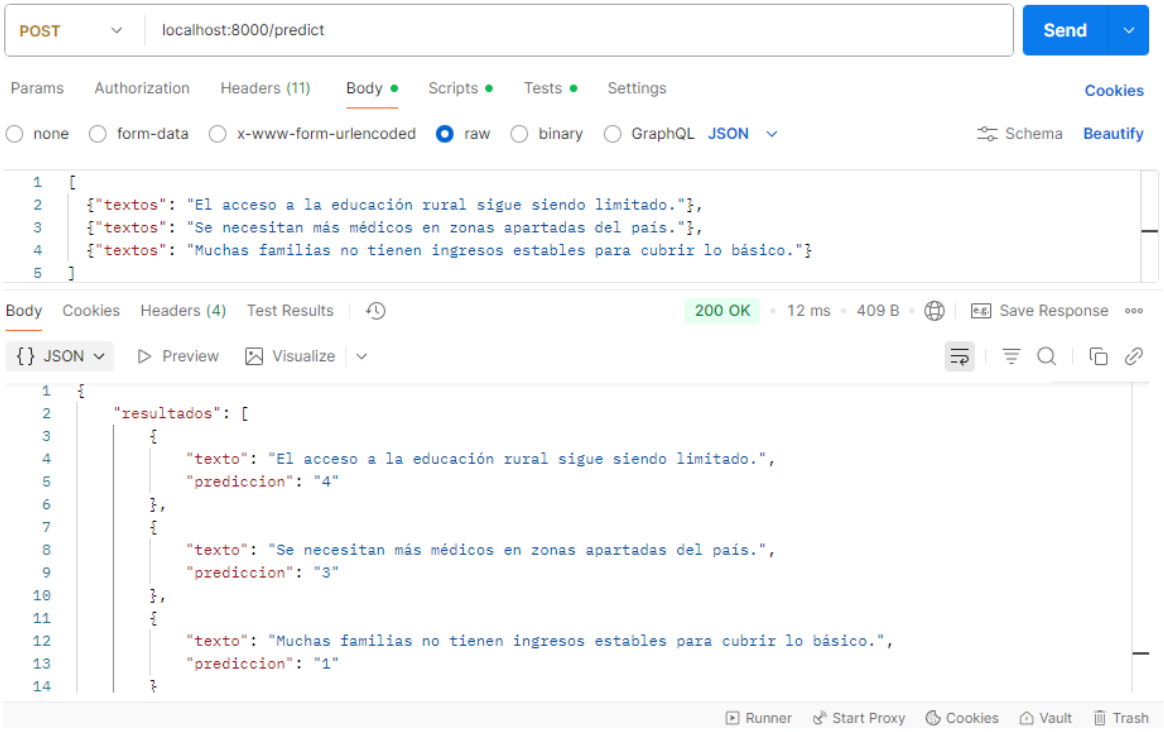

Figura 1. Pruebas de /predict

En /retrain, al enviar un lote etiquetado, la API mostró un reentrenamiento correcto sobre el dataset consolidado y devolvió métricas macro de 1.0 (consistentes con un corpus bien alineado), 2546 instancias de entrenamiento y la distribución 1: 534, 3: 938, 4: 1074. Además, el campo nuevos_clasificados reflejó el conteo del lote ingresado (por ejemplo,

{"1": 3, "3": 4, "4": 1}), lo que facilita rastrear qué se incorpora en cada iteración sin revisar manualmente el store.

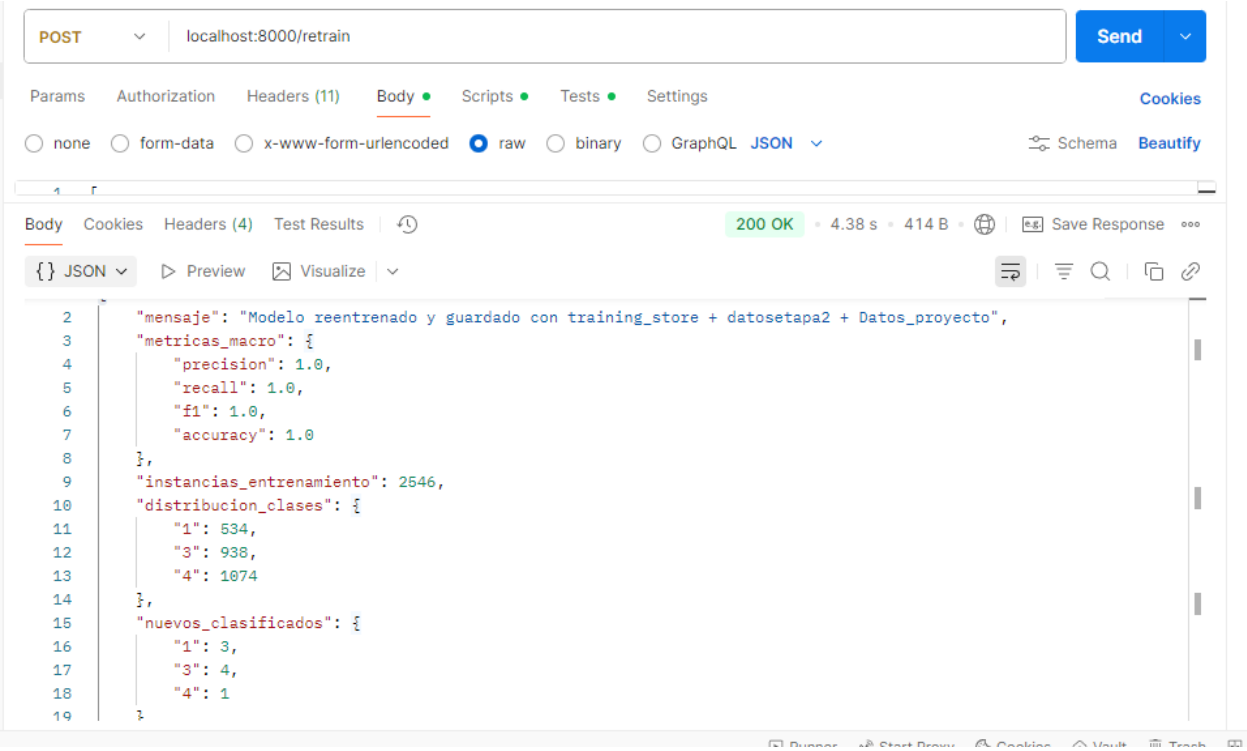

Figura 2. Pruebas de /retrain

2.4. Despliegue con Docker

Para el despliegue se construyó un Dockerfile basado en python:3.10-slim, que instala las dependencias de requirements.txt, copia el proyecto y expone el servicio en el puerto 8000. El contenedor ejecuta la app con Uvicorn, lo que simplifica la puesta en marcha en cualquier entorno. Con docker build -t ods-api:0.1.2 . y docker run --rm -p 8000:8000 ods-api:0.1.2, la API queda accesible en http://localhost:8000, y puede comprobarse con /health y /docs.

Sección 3

La app web se construyó sobre FastAPI. Hay una plantilla base que define el layout y dos vistas: home.html (predicción) y admin_panel.html (reentrenamiento), además de admin_login.html para el acceso. Los formularios HTML envían peticiones al backend y la respuesta se renderiza en la misma página con mensajes de validación. El backend expone los endpoints ya existentes (/predict, /retrain) y sirve estáticos (CSS/JS) para mantener una UI limpia y consistente. En Docker, la UI y el backend viven en un solo

contenedor, compartiendo volúmenes ./artifacts y ./data (datasets y training_store.csv) para persistencia entre ejecuciones.

3.1. Pantalla de Predicción

Esta funcionalidad está dirigida al rol de Analista de Políticas Públicas, es decir, a usuarios no técnicos encargados de clasificar y analizar información ciudadana relacionada con los ODS.

Una vez el analista ingresa un texto en el área de “Predicción rápida” y pulsa Clasificar, el sistema envía la información al endpoint /predict y devuelve la clase estimada.

Para procesar muchos textos al tiempo, la sección “Predicción por lote (CSV/XLSX)” permite subir un archivo que debe tener la columna textos. Al pulsar Procesar, el servidor lee el archivo, valida el esquema y retorna una tabla con las predicciones por fila.

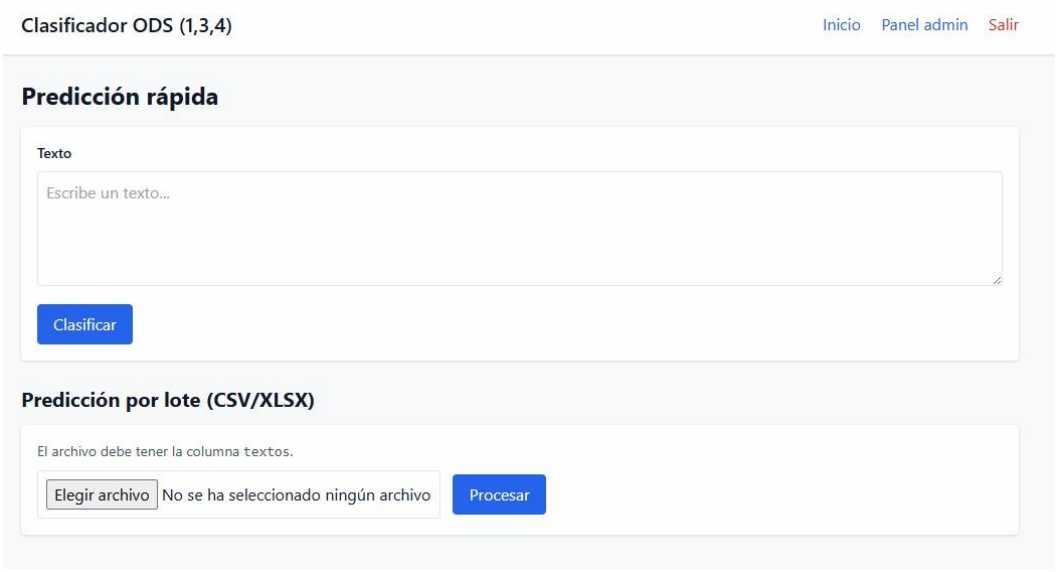

Figura 3. Pantalla de predicción.

3.2. Pantalla Panel admin

Esta sección está dirigida al rol de Administrador Técnico o Ingeniero de Datos, quien es responsable de mantener y actualizar el modelo analítico. A diferencia del rol de analista, este usuario tiene acceso a funcionalidades más avanzadas que permiten incorporar nuevos datos y monitorear el desempeño del modelo.

El panel Admin muestra un resumen del modelo en uso, las clases disponibles (1, 3 y 4) y los nombres de columnas (textos / labels). Hay dos formas de re-entrenar:

1. Línea a línea: se escribe un texto, se elige la etiqueta (1, 3 o 4) y se pulsa Re-entrenar; la instancia se guarda en el store y se dispara el reentrenamiento con la unión de training_store.csv + datosetapa2.xlsx + Datos_proyecto.xlsx.
2. Por lote: se sube un CSV/XLSX con columnas textos y labels. Tras el proceso, el panel actualiza automáticamente métricas macro (accuracy, F1, precision y recall), la distribución de clases del dataset total y el bloque nuevos_clasificados con el conteo de ejemplos del último lote por clase (útil para auditar lo que se añadió).

Todo queda persistido en los volúmenes montados, por lo que los cambios sobreviven a reinicios del contenedor.

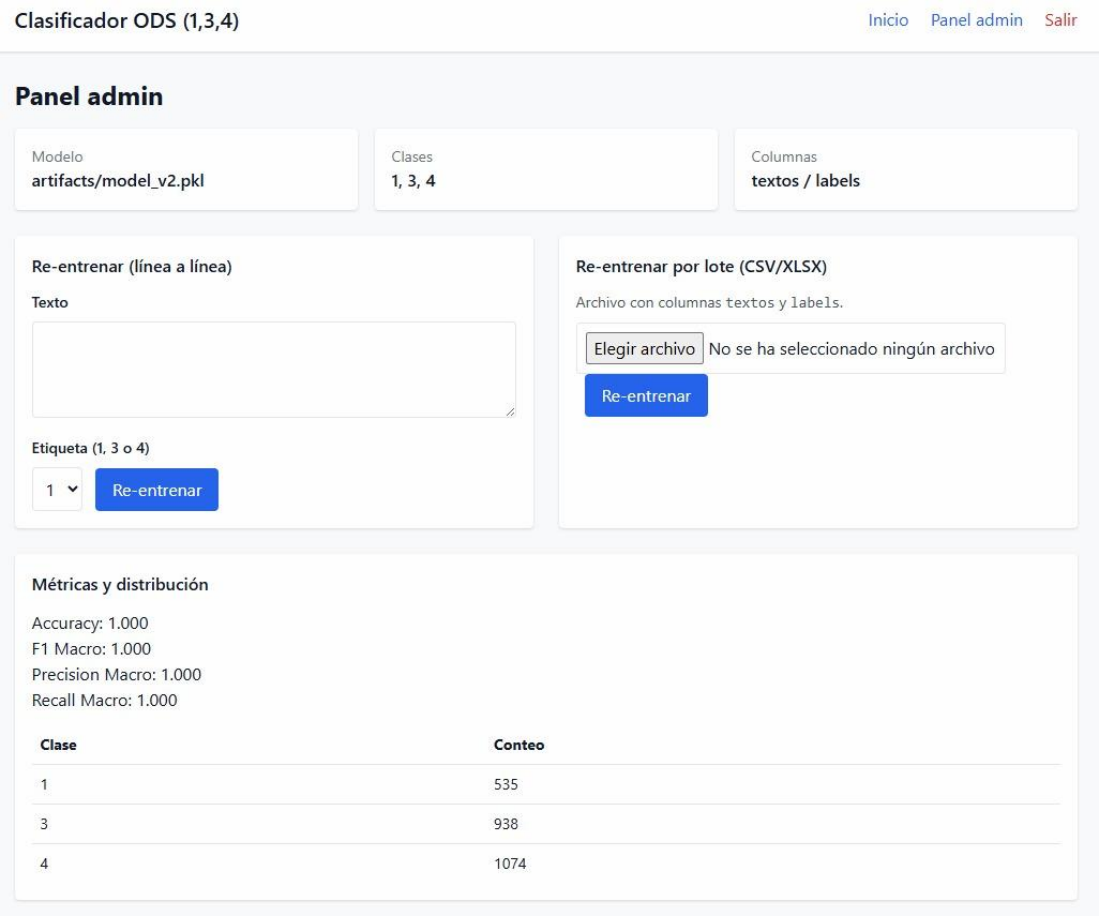

Figura 4. Pantalla de Re-entrenar.

3.3. Solución preguntas

1. ¿Qué recursos informáticos requiere para entrenar, ejecutar, persistir el modelo analítico y desplegar la aplicación?

La aplicación no requiere una infraestructura compleja para funcionar. El modelo fue entrenado usando un pipeline basado en TF-IDF y SVM lineal, por lo que puede ejecutarse eficientemente en máquinas con recursos moderados (por ejemplo, 4 GB de RAM y un procesador de 2 núcleos). Para garantizar persistencia, el modelo se guarda

en archivos serializados (.joblib o .pkl), que pueden almacenarse localmente o en un volumen persistente.

La API y la interfaz web están contenedorizadas con Docker, lo que permite desplegar la solución fácilmente en cualquier servidor con Docker Engine instalado. Esto hace posible tanto un despliegue local como uno en la nube sin requerir GPUs ni servidores especializados.

2. **¿Cómo se integrará la aplicación construida a la organización, estará conectada con algún proceso del negocio o cómo se pondrá a disposición del usuario final?**

La aplicación está diseñada para integrarse a procesos de análisis de información ciudadana que realizan entidades públicas u organizaciones internacionales. El analista puede usar la interfaz web para clasificar automáticamente grandes volúmenes de textos, sin necesidad de conocimientos técnicos. Paralelamente, el administrador técnico puede cargar nuevos datos y actualizar el modelo directamente desde el panel de administración.

Esto permite que la herramienta se conecte de forma natural con flujos de trabajo existentes, por ejemplo, análisis de comentarios ciudadanos en redes sociales, formularios o plataformas de participación pública. Además, al estar empaquetada en contenedores, puede ponerse a disposición del usuario final en un servidor institucional, a través de una URL interna o externa protegida.

3. **¿Qué riesgos tiene para el usuario final usar la aplicación construida?**

Como toda solución basada en modelos de machine learning, uno de los principales riesgos es que las predicciones puedan ser erróneas o poco confiables en casos ambiguos o en textos que no se parezcan a los datos de entrenamiento. Esto podría llevar a interpretaciones equivocadas si no se revisan con criterio humano.

Otro riesgo es la posible exposición de datos sensibles si no se asegura correctamente el entorno de despliegue, razón por la cual se recomienda ejecutarla en servidores institucionales y con control de acceso. Finalmente, como la aplicación depende de actualizaciones periódicas, no mantener el modelo actualizado podría afectar la calidad de las clasificaciones con el tiempo.

Conclusiones

En esta segunda etapa del proyecto logramos fortalecer de manera significativa nuestro modelo de analítica de textos, especialmente en la clasificación de opiniones relacionadas con pobreza, que inicialmente era la clase menos representada. A través de una estrategia de aumentación sintética cuidadosamente diseñada, conseguimos mejorar el desempeño del modelo sin afectar su precisión global, evidenciando que una intervención bien dirigida puede equilibrar los datos y potenciar la sensibilidad del clasificador. Además, gracias a la implementación de un pipeline estable y adaptable, el

sistema respondió bien ante nuevos datos, manteniendo métricas altas y un comportamiento consistente en los procesos de reentrenamiento.

La integración entre el modelo, la API y la interfaz web permitió que la herramienta funcionara como una solución completa: por un lado, accesible para analistas no técnicos que solo necesitan ingresar textos y obtener resultados confiables; y por otro, administrable por personal técnico que puede actualizar y mantener el modelo en funcionamiento. Esto, sumado a un despliegue ligero con Docker, hace que la aplicación sea fácil de instalar, replicar y escalar a distintos entornos.

En general, la solución desarrollada no solo automatiza una tarea compleja, sino que también habilita una toma de decisiones más ágil y basada en evidencia, especialmente en contextos donde se manejan grandes volúmenes de información ciudadana. Su estabilidad frente al reentrenamiento demuestra que está preparada para seguir evolucionando e integrarse con funcionalidades futuras, como dashboards o la ampliación a otros ODS, lo que la convierte en una herramienta sólida y adaptable para organizaciones que trabajan en la construcción de políticas públicas.