Proyecto 1 Etapa 2: Ajuste, automatización y uso de modelos de analítica de textos

Universidad de los Andes

Curso: ISIS-3301 – Inteligencia de Negocios

Proyecto: Analítica de Textos – Etapa 2

Semestre: 2025-20

Grupo: 1

Integrantes:

Gabriel Aristizábal – Ingeniero de datos / Líder de proyecto

Pablo Lara – Ingeniero de software responsable de desarrollar la aplicación final

Juan Gordillo – Ingeniero de software responsable del diseño de la aplicación y resultados

Tabla de contenido

1. Aumentación de datos y reentrenamiento del modelo (20%)

   El nuevo conjunto Datos_etapa2 contiene 100 opiniones ciudadanas sobre los ODS 1, 3 y 4, con mayor representación del ODS 4 (Educación de calidad). Esta distribución generó un leve desbalance de clases, además de incorporar nuevas expresiones y vocabulario regional, lo que aportó diversidad al modelo. Al evaluar el modelo de la Etapa 1 sobre este conjunto, se obtuvieron las siguientes métricas:

   Accuracy: 0.90 – Precision: 0.91 – Recall: 0.90 – F1-score: 0.8867.

   El rendimiento disminuyó frente al obtenido inicialmente, debido a:

   - El desbalance en las clases, con más ejemplos del ODS 4.
   - La variación lingüística de los nuevos textos.

- Un leve sobreajuste al dataset original.

En conjunto, estos resultados evidencian la necesidad de un reentrenamiento con datos aumentados para mejorar la generalización y robustez del modelo.

Para corregir el desbalance de clases identificado en el conjunto de datos de la Etapa 2, se aplicó una técnica de aumentación mediante prompting usando el modelo gpt-4o-mini de OpenAI. Primero, se identificó la clase minoritaria dentro del dataset (labels) y se seleccionaron algunos textos reales como ejemplos de referencia. Luego, se diseñó un prompt en español que instruyó al modelo a generar 200 opiniones ciudadanas breves y realistas relacionadas únicamente con el ODS minoritario (1, 3 o 4), garantizando diversidad temática y neutralidad.

El modelo devolvió los resultados en formato JSON con las columnas "textos" y "labels", los cuales se validaron y agregaron al dataset original para balancear las clases y aumentar la cobertura semántica. Este enfoque permitió ampliar el conjunto de entrenamiento sin introducir ruido, mejorando la capacidad del modelo para generalizar frente a nuevas expresiones y contextos.

Tras generar los ejemplos sintéticos mediante prompting, se aplicó una segunda técnica de aumentación con la librería nlpaug, la cual utiliza sinónimos de WordNet para crear variaciones semánticas de los textos. Con esto se generaron nuevas opiniones para las clases minoritarias, hasta equilibrar la distribución entre los tres ODS.

Los datos aumentados se unieron al conjunto original para conformar un dataset balanceado, verificado mediante conteo porcentual y visualización de barras. El resultado se almacenó en Google Drive bajo el archivo Datos_balanceados_etapa2.xlsx.

Posteriormente, se dividieron los datos en entrenamiento (80%) y prueba (20%), y se reentrenó el modelo usando un pipeline de Regresión Logística con TF-IDF, lo que automatizó la limpieza, vectorización y clasificación de los textos. El modelo fue ajustado mediante GridSearchCV para optimizar hiperparámetros y luego guardado como modelo_reentrenado.pkl.

Finalmente, se evaluó su desempeño en el conjunto de prueba, obteniendo una precisión de 0.99, con mejoras en el equilibrio entre clases y una matriz de confusión más uniforme respecto al modelo inicial.

	Accuracy	Precision (macro)	Recall (macro)	F1-score (macro)
Modelo Original	0.971564	0.972894	0.970569	0.971227
Modelo Reentrenado	0.988942	0.988754	0.989378	0.988973

El rendimiento del modelo mejoró tras el reentrenamiento porque la aumentación de datos redujo el desbalance entre clases y amplió la variedad lingüística de los ejemplos. Esto permitió que el modelo aprendiera mejor los patrones asociados a los ODS menos representados y generalizara con mayor precisión ante nuevas opiniones. En consecuencia, aumentó el recall y el F1-score, mostrando un comportamiento más estable y equilibrado entre todas las categorías.

2. Automatización con pipeline y API REST (15%)

   Pipeline

   El pipeline fue desarrollado con la librería scikit-learn y está compuesto por tres etapas principales:

   Limpieza de texto: Se aplica la función clean_text() para eliminar URLs, menciones, números, caracteres especiales y normalizar el texto en minúsculas.

   Vectorización: Se utiliza un TfidfVectorizer con n-gramas (1, 2) y ponderación sublineal (sublinear_tf=True) para convertir los textos en representaciones numéricas basadas en su relevancia semántica.

   Clasificación: Se implementa una Regresión Logística (OvR) con optimización de hiperparámetros mediante GridSearchCV y validación cruzada estratificada.

   El modelo resultante se guarda como modelo_reentrenado.pkl y puede cargarse directamente en la API. Este pipeline garantiza consistencia entre las etapas de entrenamiento, validación y predicción, asegurando reproducibilidad y mantenimiento del flujo completo.

   API REST

   La API implementa tres endpoints principales que permiten interactuar con el modelo:

   POST /predict: Recibe textos en formato JSON y devuelve la predicción del ODS más probable para cada entrada.

   Ejemplo:

   {"instances":

   [{"textos": "Estamos implementando programas de empleo para

   reducir la pobreza."},

{"textos": "La contaminación en la ciudad afecta más la salud de los habitantes."}]

}

Respuesta:

{ "predictions":

[1,

3]

}

POST /retrain

Permite enviar nuevos datos etiquetados (textos, labels) para reentrenar el modelo.

Durante este proceso, la API limpia y valida los textos, balancea las clases mediante aumentación híbrida (GPT-4o-mini + sinónimos de WordNet), reentrena y reemplaza el modelo guardado y devuelve métricas actualizadas de desempeño.

POST /predict_proba

Similar al endpoint /predict, pero devuelve las probabilidades asociadas a cada clase (ODS 1, 3 y 4), en lugar de una sola etiqueta. Este endpoint es útil para análisis comparativos o calibración del modelo, ya que muestra el grado de confianza de la predicción.

Ejemplo:

{"instances":

[{"textos": "Estamos implementando programas de empleo para

reducir la pobreza."},

{"textos": "La contaminación en la ciudad afecta más la salud de los habitantes."}]

}

Respuesta:

{ "probabilities":

[{ "1": 0.9677539382007577,

"3": 0.013757722983493266,

"4": 0.018488338815748937

},

{"1": 0.06705004324209507,

"3": 0.8745573997577982,

"4": 0.05839255700010673

}]

}

En conjunto, el pipeline y estos tres endpoints permiten automatizar todo el ciclo de vida del modelo, desde la limpieza y entrenamiento, hasta la predicción, evaluación y despliegue, garantizando coherencia y facilidad de integración con otras aplicaciones.

3. Desarrollo y justificación de la aplicación (25%)

   El principal usuario de la aplicación es un analista de sostenibilidad o funcionario del área de responsabilidad social dentro de una organización que evalúa las percepciones ciudadanas sobre los Objetivos de Desarrollo Sostenible (ODS). Este rol necesita identificar rápidamente a qué ODS se asocia una opinión o comentario, con el fin de orientar decisiones, reportes y estrategias institucionales.

   La aplicación web desarrollada permite que este usuario ingrese uno o varios textos y obtenga de forma automática:
   - La predicción del ODS correspondiente (1, 3 o 4).
   - La probabilidad asociada a cada clase (vía /predict_proba).

   Además, si el usuario cuenta con permisos avanzados, puede reentrenar el modelo incorporando nuevas opiniones desde la interfaz, manteniendo actualizado el análisis frente a nuevos contextos lingüísticos o territoriales. La aplicación se integra con el proceso de análisis de percepciones ciudadanas y gestión de proyectos alineados con los ODS, facilitando la clasificación automática de grandes volúmenes de texto provenientes de encuestas, redes sociales o formularios institucionales. De este modo, la

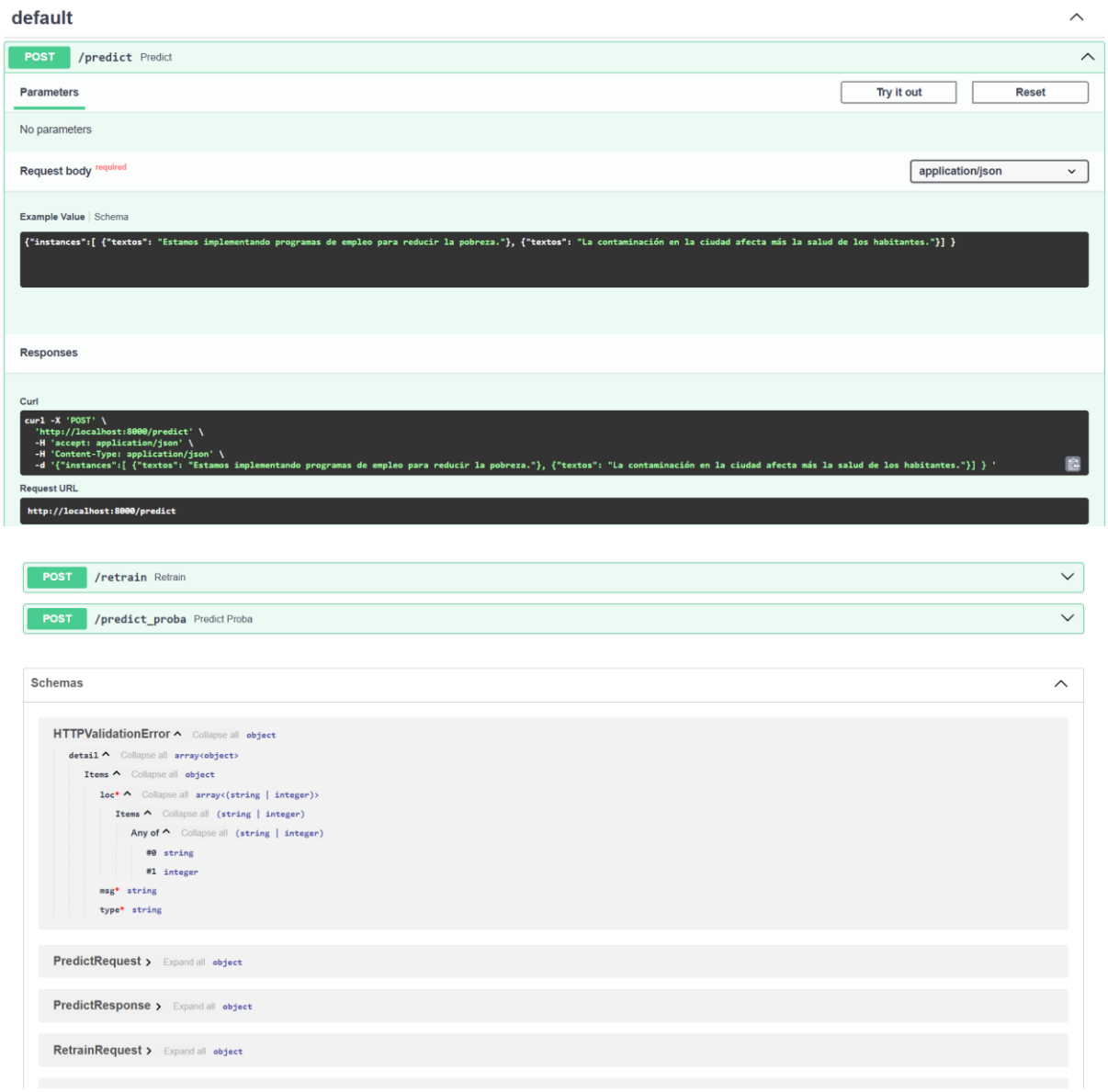
herramienta reduce el tiempo de análisis manual, mejora la consistencia en la categorización y apoya la toma de decisiones basada en datos.

Para operar correctamente, el sistema requiere:
- CPU o GPU con soporte para librerías de machine learning (scikit-learn, pandas, nlpaug, OpenAI SDK).
- Un servidor o contenedor Docker con entorno Python 3.10+, donde se despliegue la API REST.
- Almacenamiento de modelos (.pkl) y datasets balanceados (.xlsx) en servicios como Google Drive, S3 o almacenamiento local del servidor.

La aplicación se pondrá a disposición mediante un portal interno o enlace web, donde los analistas o gestores de sostenibilidad puedan cargar textos, revisar predicciones y decidir cuándo reentrenar el modelo. También puede conectarse con sistemas institucionales de recopilación de opiniones ciudadanas, integrándose en el flujo de gestión y monitoreo de los ODS.

Los principales riesgos identificados son:
- Errores de predicción pueden influir en interpretaciones inadecuadas si el modelo no se actualiza con frecuencia.
- Si los textos nuevos no representan bien la realidad lingüística, el modelo puede reforzar patrones erróneos.
- Permitir reentrenamientos sin control podría degradar el desempeño si se incorporan datos no validados.

Interfaz:



4. Resultados (18%)

   En el video

5. Trabajo en equipo (10%)

Integrante	Rol	Horas dedicadas	Tareas realizadas
Gabriel Aristizábal	Ingeniero de datos / Líder de proyecto	25 horas	Coordinación general del proyecto y asignación de responsabilidades

			Limpieza, integración y validación del conjunto de datos inicial. Implementación del pipeline de entrenamiento (TF-IDF + Regresión Logística). Ejecución del proceso de aumentación de datos con nlpaug y el modelo GPT-4o-mini. Análisis de métricas, generación de reportes de rendimiento y documentación técnica. Consolidación final del informe y revisión de cumplimiento con el enunciado.
Pablo Lara	Ingeniero de software responsable de desarrollar la aplicación final	22 horas	Desarrollo de la API REST con los endpoints /predict, /predict_proba y /retrain. Integración del modelo entrenado con la API usando joblib. Implementación de validaciones, manejo de errores y formato de respuesta JSON. Despliegue de la aplicación en entorno web o contenedor Docker. Pruebas funcionales y validación del flujo completo de inferencia y reentrenamiento. Documentación de endpoints y conexión con la interfaz de usuario.
Juan Gordillo	Ingeniero de software responsable del diseño de la aplicación y resultados	20 horas	Diseño de la interfaz web para ingreso de textos y visualización de resultados. Conexión del frontend con los endpoints /predict y /predict_proba. Presentación de resultados y métricas en formato visual

			(gráficas, reportes, probabilidades). Redacción de los apartados de análisis de resultados, interpretación de métricas y explicación del desempeño del modelo. Apoyo en la estructuración del documento final y revisión del formato de entrega.

Los 100 puntos del proyecto se repartirán en partes iguales entre los tres integrantes: Gabriel Aristizábal – 33.3 puntos, Pablo Lara – 33.3 puntos, Juan Gordillo – 33.3 puntos

Durante el desarrollo se presentaron varios retos, entre ellos el desbalance de clases en los datos, solucionado mediante aumentación con nlpaug y GPT-4o-mini. Dificultades para integrar el modelo con la API, resueltas usando un pipeline unificado y persistencia con joblib. Problemas de entorno y dependencias, corregidos mediante el uso de contenedores Docker.

Durante el desarrollo y documentación del proyecto se emplearon herramientas de inteligencia artificial para corregir la ortografía y redacción del informe, así como para agilizar la búsqueda de funciones, librerías y ejemplos de implementación relacionados con el modelo, el pipeline y la aplicación web. Esto permitió optimizar el tiempo de desarrollo y asegurar una presentación técnica clara y coherente.

Gracias a la coordinación y división de tareas, el equipo logró una solución funcional y estable que cumple con los objetivos de la Etapa 2.

6. Referencias

- Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media.

- Chollet, F., & Allaire, J. J. (2022). Deep Learning with Python (2nd ed.). Manning Publications.
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

- McKinney, W. (2010). Data structures for statistical computing in Python. Proceedings of the 9th Python in Science Conference, 51–56.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.
- Rajan, V. (2020). nlpaug: Data augmentation for NLP. Recuperado de https://github.com/makcedward/nlpaug
- OpenAI. (2024). OpenAI API documentation. Recuperado de https://platform.openai.com/docs
- Python Software Foundation. (2024). Python 3.10 documentation. Recuperado de https://docs.python.org/3/
- Seaborn Developers. (2023). Seaborn statistical data visualization. Recuperado de https://seaborn.pydata.org
- Docker Inc. (2023). Docker Documentation. Recuperado de https://docs.docker.com
- Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media.
- Tiangolo, S. (2018). FastAPI: Modern, fast (high-performance), web framework for building APIs with Python 3.7+. Recuperado de https://fastapi.tiangolo.com
- Streamlit Inc. (2023). Streamlit Documentation. Recuperado de https://docs.streamlit.io