Proyecto 1 – Etapa 2
Documento de Análisis

Integrantes:
Carlos Vargas - 202220064
Silvana Echeverry - 202310470
David Mora – 202226269

Link al repositorio del proyecto: https://github.com/silvanaecheverry/Proyecto1-BI

1. **Distribución del trabajo (se incluye en la wiki del repositorio del proyecto)**

2. **Sección 1**

Justificación del Algoritmo:

En la Etapa 1 se evaluaron distintos algoritmos de clasificación para resolver el problema planteado en la analítica de textos. Tras el proceso de comparación, el modelo seleccionado fue Logistic Regression, debido a los siguientes motivos principales:

1. Consistencia entre clases: La matriz de confusión de Logistic Regression mostró una mejor distribución de los aciertos entre las diferentes clases, evidenciando menor sesgo hacia una clase específica en comparación con otros algoritmos.
2. Excelente capacidad de discriminación (ROC-AUC = 0.99): El área bajo la curva ROC alcanzó un valor de 0.99, lo que indica una alta capacidad del modelo para diferenciar correctamente entre las clases, incluso en escenarios con datos desbalanceados.
3. Interpretabilidad: Logistic Regression permite interpretar la importancia relativa de cada característica (en este caso, palabras o tokens) en la clasificación, lo que aporta valor adicional al negocio al ofrecer explicabilidad en la toma de decisiones. Este nivel de interpretabilidad no se logra de forma tan directa en algoritmos como Naive Bayes.
4. Rendimiento cuantitativo
    a. Accuracy: 0.96
    b. F1-Score: 0.95
    c. ROC-AUC global: 0.99

    Estos resultados evidencian que Logistic Regression logra un equilibrio entre precisión y recall, maximizando la capacidad de clasificación de manera robusta.
5. Visualización de resultados
    a. La matriz de confusión reflejó un bajo nivel de errores de clasificación.
    b. Las curvas ROC por clase muestran valores cercanos a 1 en todas las categorías (AUC = 0.997–0.999), confirmando la confiabilidad del modelo en cada una de las clases.

En conclusión, Logistic Regression fue seleccionado como el mejor algoritmo de la Etapa 1 por su alto rendimiento, robustez, baja tendencia al sesgo y facilidad de interpretación, características que lo convierten en la mejor opción para ser reentrenado en la Etapa 2 con datos aumentados.

Comparación de resultados Etapa 1 y 2

Métrica	Etapa 1 – Datos viejos (485 test)	Etapa 2 – Datos nuevos (20 test)
Accuracy	0.961	0.900
F1-macro	0.956	0.856
ROC-AUC	0.997–0.999 por clase	0.703 – 1.000 por clase
Precisión	0.960 (macro)	0.939 (macro)
Recall	0.953 (macro)	0.833 (macro)

Etapa 1 (datos viejos)

- El modelo logra un alto rendimiento con F1-macro de 0.956 y Accuracy de 0.961.
- La matriz de confusión muestra muy pocos errores en las clases 1, 3 y 4.
- Las curvas ROC están prácticamente perfectas (AUC ≈ 1.0 en todas las clases).
- Conclusión: el modelo está muy bien ajustado a los datos viejos, pero esto también indica posible sobreajuste (overfitting), ya que en entrenamiento las métricas fueron perfectas (AUC = 1.0).

Etapa 2 (datos nuevos)

- La Accuracy nos da 0.90 y el F1-macro nos da 0.856.
- El recall de la clase 1 (solo 4 ejemplos) fue bajo (0.5), lo que afecta el promedio macro.
- La clase 3 fue clasificada perfectamente (precisión y recall = 1.0).
- La clase 4 tuvo un rendimiento alto (recall = 1.0, precisión = 0.818).
- El ROC-AUC refleja variabilidad: Clase 1 con 0.703, Clase 3 con 1.0 y Clase 4 con 0.929.
- Conclusión: el modelo mantiene buen desempeño, pero se observa mayor variabilidad debido al tamaño pequeño del conjunto de test

Logistic Regression generaliza bien, aunque en datos nuevos las métricas bajan como es esperado. La caída de métricas está asociada a la escasez de ejemplos en las clases minoritarias (especialmente clase 1). Esto justifica la aumentación de datos en la Etapa 2: al generar ejemplos adicionales de las clases con menos representación, se espera mejorar el recall y la robustez del modelo.

Comparación de resultados Etapa 1, 2 y ambas con los datos imputados por la IA

Métrica	Etapa 1 – Datos viejos	Etapa 2 – Datos nuevos	Modelo con datos imputados
Accuracy	0.961	0.900	0.959
F1-macro	0.956	0.856	0.954
ROC-AUC	0.997–0.999 por clase	0.703 – 1.000 por clase	0.996–0.997 por clase
Precisión	0.960 (macro)	0.939 (macro)	0.959 (macro)
Recall	0.953 (macro)	0.833 (macro)	0.949 (macro)

Etapa 1 – Datos viejos

- El modelo alcanza un rendimiento muy alto: F1-macro = 0.956, Accuracy = 0.961.
- Las curvas ROC fueron prácticamente perfectas (AUC ≈ 1.0).

- Hay una posible limitación en este modelo gracias a el excelente rendimiento el cual puede estar influenciado por un sobreajuste, ya que las métricas en entrenamiento también fueron casi perfectas.

Etapa 2 – Datos nuevos

- El rendimiento baja a F1-macro = 0.856 y Accuracy = 0.90.
- El bajo tamaño de muestra genera alta variabilidad, especialmente en la clase 1, que tuvo recall bajo (0.50).
- El AUC refleja esta inestabilidad: desde 0.703 hasta 1.0 según la clase.
- El modelo sigue funcionando, pero los resultados son menos confiables por la escasez de ejemplos.

Modelo con datos imputados

- Con la integración y aumentación de datos, el modelo recupera estabilidad: F1-macro = 0.954, Accuracy = 0.959, muy cercanos a Etapa 1.
- El ROC-AUC macro = 0.996, mostrando que el modelo logra discriminar muy bien entre clases en un conjunto de prueba más amplio.
- La matriz de confusión refleja menor sesgo y más consistencia en las tres clases.
- Conclusión: el reentrenamiento con imputación y aumento de datos permitió consolidar el desempeño, alcanzando resultados robustos y comparables a los de Etapa 1, pero ahora con mayor capacidad de generalización.

El modelo de Etapa 1 parecía ideal, pero podía estar sobreajustado. En Etapa 2 se evidenció la caída de rendimiento por la falta de representatividad en el set de prueba. Con el modelo imputado, se logra un balance óptimo: alto rendimiento y robustez frente a más datos, confirmando que la estrategia de aumentación y reentrenamiento mejora la capacidad de generalización del clasificador.

3. Sección 2

Proceso de implementación del pipeline para el procesamiento de los datos:

En el proceso de construcción del modelo consistió basicamente en encapsular todas las etapas de preparación, vectorización y entrenamiento en un Pipeline de scikit-learn.

La implementación de este pipeline permite organizar y automatizar el flujo de trabajo de manera reproducible, garantizando que tanto el preprocesamiento como el modelo de clasificación puedan ejecutarse de forma consistente cada vez que se utilicen datos nuevos o aumentados.

El flujo comienza con el preprocesamiento de los textos, cuyo objetivo es asegurar la calidad de la información utilizada para entrenar el modelo. En primer lugar, los textos se transformaron a minúsculas y se eliminaron acentos y caracteres especiales, normalizando así la entrada. Posteriormente, se aplicó un proceso de lematización con la librería spaCy, que reduce cada palabra a su forma base, lo cual ayuda a unificar variaciones morfológicas. A continuación, se realizó la tokenización, que separa los textos en palabras individuales preservando caracteres acentuados y números, y finalmente se aplicó una normalización para eliminar palabras vacías o stopwords en español, es decir, términos de alta frecuencia y bajo contenido semántico como

“de”, “que” o “la”. Este conjunto de pasos garantizó que los textos estuvieran limpios, normalizados y con la información lingüística esencial para el aprendizaje automático.

Una vez preprocesados los textos, se procedió a la vectorización mediante TF-IDF, que representa cada documento en forma numérica según la importancia relativa de las palabras. Para evitar ruido en los datos, se configuró el vectorizador con parámetros que descartan palabras demasiado raras (menos de cinco apariciones en el corpus) y términos excesivamente frecuentes (presentes en más del 90% de los documentos). Asimismo, se trabajó con unigramas (palabras individuales) y se aplicó normalización sublineal a la frecuencia, con el fin de que documentos largos no dominen el modelo. Este enfoque permitió obtener representaciones robustas de los textos que capturan la relevancia de los términos en cada clase.

Para la fase de clasificación se optó por utilizar un modelo de Regresión Logística, elegido tras comparar su rendimiento con otros algoritmos durante la primera etapa. Este modelo demostró un mejor equilibrio entre precisión, recall y F1-score, lo que lo hizo el algoritmo más adecuado para el problema planteado. Una de sus principales ventajas es que, además de ofrecer un alto nivel de desempeño en escenarios de clasificación multiclase, permite interpretar de manera más clara el peso que tiene cada palabra en el resultado final. Esto aporta un valor diferencial frente a otros enfoques más complejos, donde el proceso de decisión suele ser menos transparente.

Finalmente, todas estas etapas se integraron en un Pipeline de scikit-learn, que incluye dos pasos principales: el vectorizador TF-IDF y el clasificador de Regresión Logística. Este pipeline fue entrenado sobre los datos aumentados, que incorporan tanto el conjunto inicial como ejemplos generados para mejorar el balance de clases. De esta forma, se logra un proceso automatizado que recibe como entrada directamente el texto limpio y produce como salida la predicción de clase. Entre las principales ventajas de esta implementación se encuentran la automatización del flujo de trabajo, la replicabilidad de los experimentos, la escalabilidad hacia entornos de producción y la facilidad de mantenimiento, dado que cualquier ajuste en el preprocesamiento o en el modelo se realiza de manera centralizada en el pipeline.

En conclusión, el trabajo de la construcción del pipeline consistió en diseñar un proceso completo y robusto que integra el preprocesamiento, la vectorización y la clasificación en un solo flujo reproducible. Gracias a este pipeline, el modelo puede ser entrenado o reentrenado con nuevos datos de manera sencilla, asegurando consistencia en los resultados y facilitando su despliegue futuro en una API o aplicación orientada al usuario final.

Estrategias de Reentrenamiento:

1. **Reentrnamiento completo del modelo desde 0:** Esta estrategia consiste en volver a entrenar el modelo cada vez que se incorporan nuevos datos, combinando el conjunto original con los ejemplos recientes. Su principal ventaja es que garantiza que el modelo se actualice con toda la información disponible, evitando así, la acumulación de errores o sesgos previos.Sin embargo, presenta un mayor costo computacional y puede requerir tiempos de procesamiento más prolongados, especialmente cuando el volumen de datos crece. Esta estrategia, es

ideal cuando se busca máxima estabilidad y precisión, aunque no es la opción más eficiente para actualizaciones frecuentes.

2. **Entrenamiento incremental o aprendizaje continuo:** En esta estrategia, el modelo actualiza sus parámetros progresivamente sin necesidad de ser reentrenado desde cero. Se utiliza en escenarios donde los datos llegan de forma continua o en streaming, permitiendo mantener el modelo actualizado con un bajo costo de cómputo. Su desventaja principal es que puede acumular errores si los nuevos datos no son representativos o si hay un cambio drástico en la distribución del conjunto. Además, no todos los modelos de scikit-learn (como la Regresión Logística estándar) soportan este tipo de actualización incremental.
3. **Reentrenamiento por reemplazo parcial del dataset:** Esta técnica combina lo mejor de ambos mundos: en lugar de reentrenar desde cero, se conserva una parte del dataset original y se agregan los nuevos ejemplos más recientes, descartando datos sin sentido o redundantes. La ventaja es que mantiene la eficiencia y la relevancia del modelo frente a cambios graduales. No obstante, requiere un control cuidadoso del tamaño y calidad de los datos retenidos para no afectar la estabilidad del modelo.

Entonces, en el proyecto implementamos la estrategia de reentrenamiento completo del modelo, debido a su simplicidad, estabilidad y compatibilidad con el enfoque Pipeline de scikit-learn. Esta decisión garantiza que cada actualización incorpore la totalidad de los datos disponibles, tanto originales como aumentados, manteniendo la coherencia del modelo y evitando pérdida de información. Aunque demanda más tiempo de ejecución, resulta la opción más confiable para mantener un desempeño consistente en escenarios de clasificación de texto con volúmenes moderados de datos.

4. Sección 3

La aplicación está dirigida a analistas de datos y equipos de inteligencia de negocios que requieren automatizar la clasificación y análisis de textos dentro de la organización. Su valor radica en reducir el tiempo y esfuerzo del procesamiento manual, ofreciendo resultados consistentes mediante un modelo de Regresión Logística desplegado en un API REST. Gracias a su integración con sistemas internos de monitoreo y gestión documental, la herramienta se incorpora directamente en los procesos de análisis y toma de decisiones, fortaleciendo la eficiencia y la trazabilidad de la información.

¿Qué recursos informáticos requiere para entrenar, ejecutar, persistir el modelo analítico ydesplegar la aplicación? El entrenamiento, ejecución y despliegue del modelo analítico requieren recursos de cómputo de nivel medio, adecuados para entornos de desarrollo o producción ligera. Por el lado del entrenamiento del modelo, se necesita un entorno con capacidad suficiente de procesamiento dado que las operaciones de vectorización TF-IDF y el modelo de Regresión Logística implican el manejo de matrices de tamaño considerable. En caso de tener reentrenamientos frecuentes o datos mucho mas amplios que requieran un manejo de matrices mas grande, se deberia utilizar infrastructura en la nube. Para la ejecución del modelo y la persistencia, se requiere un entorno Python con las librerías scikit-learn, pandas, joblib, fastapi y uvicorn. El modelo se almacena en formato .pkl, lo que facilita su carga rápida en memoria y

reduce los tiempos de respuesta en predicciones. La aplicación web se despliega utilizando FastAPI, un framework que permite exponer el modelo a través de los endpoints REST.

¿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 desarrollada está pensada para integrarse dentro de los procesos de análisis y soporte a la toma de decisiones de la organización. El modelo, al estar encapsulado en una API REST, permite que otros sistemas internos; como plataformas de monitoreo, aplicaciones de inteligencia de negocios o herramientas de gestión documental. Puedan conectarse a él mediante solicitudes HTTP, enviando textos y recibiendo predicciones automatizadas.

De esta manera, la aplicación puede convertirse en un componente importante dentro de la arquitectura de la organización, facilitando así la automatización de tareas de clasificación de texto, análisis de opinión o segmentación de información. Además, al contar con el endpoint de reentrenamiento dinámico, los analistas de datos pueden actualizar el modelo con nuevos ejemplos sin necesidad de intervención técnica, lo que garantiza su evolución y mejora continua. Para el usuario final, el acceso se puede habilitar a través de una interfaz web, un panel interno o mediante integración directa con otros sistemas existentes.

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

Consideramos que el principal riesgo asociado al uso de la aplicación se relaciona con la interpretación incorrecta de los resultados del modelo. Dado que la Regresión Logística, aunque interpretable, depende de los patrones presentes en los datos de entrenamiento, existe la posibilidad de sesgos o errores.

Otro riesgo potencial es el mal uso de los endpoints, por ejemplo, el envío de datos en formatos incorrectos o el acceso no autorizado al API. Para mitigar esto, se recomienda implementar mecanismos de autenticación, validación de entradas y control de versiones del modelo.

5. **Bibliografia:**

scikit-learn. (n.d.). *Scaling strategies — scikit-learn 0.15 documentation.* Recuperado de https://scikit-learn.org/0.15/modules/scaling_strategies.html

Neptune.ai. (2022, octubre 24). *How to retrain your model during deployment: Continuous training & continuous testing.* Recuperado de https://neptune.ai/blog/retraining-model-during-deployment-continuous-training-continuous-testing

scikit-learn. (n.d.). *Pipeline — scikit-learn 1.5 documentation.* Recuperado de https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Daily Dev. (2023, julio 5). *Scikit-learn pipelines: Build, optimize, and explain.* Recuperado de https://daily.dev/blog/scikit-learn-pipelines-build-optimize-explain