IES PERE MARIA ORTS

ESO, Batxillerat, Arts, Formació Professional

Conselleria de Educación, Cultura, Universidades y Empleo

Curso de Especialización en Inteligencia Artificial y Big Data

Programación IA Y Sistemas Big Data

PROYECTO INTERMODULAR

Trabajo fin de estudio presentado por:	Raúl Márquez Roig
Director/a:	Sergi Pérez Castaños
Fecha:	01/06/2025
Repositorio del código fuente:	https://github.com/programacion-ia/proyecto-intermodular-raulmarquez93

Resumen

El objetivo de este proyecto es desarrollar un sistema integral de inteligencia artificial capaz de predecir la baja (churn) de clientes en una compañía telefónica, integrando tanto procesamiento batch como streaming. El sistema se divide en dos fases principales: entrenamiento y producción.

En la fase de entrenamiento, se realizará un análisis exploratorio exhaustivo de los datos de clientes y servicios, identificando patrones, valores atípicos y características relevantes. Se utilizarán distintas técnicas de preprocesamiento —como normalización, imputación de nulos y selección de características— y se entrenarán y compararán varios modelos de aprendizaje automático, como RandomForest, LogisticRegression, SVC, XGBoost y CatBoost. La selección y optimización de estos modelos se realizará con herramientas de automatización y experiment tracking (MLflow), buscando maximizar el área bajo la curva ROC (AUC) y la capacidad de generalización del sistema.

La fase de producción consistirá en la implementación de dos ramas: batch y streaming.

En el procesamiento batch, se expondrá un servicio de predicciones con FastAPI que reciba datos de test y almacene resultados en una base de datos TimescaleDB. Además, se validará la calidad del modelo usando un servicio externo proporcionado por el profesor.

En la rama streaming, se recibirá un flujo continuo de nuevos datos a través de un endpoint, los cuales serán enviados a Kafka para su procesamiento en tiempo real. Un consumidor recuperará los datos, realizará predicciones con el modelo de MLflow y almacenará los resultados en MongoDB.

Todo el sistema estará dockerizado y modularizado, con Airflow orquestando los flujos de trabajo y garantizando la reproducibilidad y escalabilidad del proyecto. Finalmente, se documentarán todas las decisiones tomadas, justificando cada paso, y se entregará el código completo, el informe detallado y una presentación para la defensa del proyecto.

Table of Contents

1. Introducción y Motivación 5
2. Entorno tecnológico 6
3.Fase de entrenamiento 7
3.1. Exploración de los datos 7
3.2. Revision bibliográfica 10
3.3. Preprocesamiento de Datos 11
3.4. Entrenamiento del modelo 12
3.5. Registros de experimentos 14
4. Fase de producción 14
4.1 Rama Batch 15
4.2. Rama Streaming 16
5. Arquitectura del Proyecto 18
6. Resultados 21
6.1 Análisis general de los resultados obtenidos 21
6.2 Comparación entre modelos 22
6.3 Modelo final y justificación de la selección 23
6.4 Comentarios sobre rendimiento y escalabilidad 24
7. Limitaciones 25
8. Conclusiones y Futuro Trabajo 26
8.1 Conclusiones obtenidas del proyecto 26
8.2 Posibles mejoras o extensiones del trabajo 27
9. Anexos 28
9.1 Scripts relevantes 28
9.2 Configuración adicional 29
10. Referencias 30
10.1. Bibliografía Científica 30
10.2. Fuentes Técnicas y Documentación 30

Figuras

Figura 1.Cargos Totales y Mensuales por cliente ____ *7*
Figura 2.Cargos mensuales por Abandono *Figura 3. Cargos totales por abandono* ____ *8*
Figura 4. Estado civil Por abandono ____ *8*
Figura 5. Distribución de abandono según variables de servicios y contrato ____ *9*
Figura 6Diagrama general del flujo de entrenamiento. ____ *12*
Figura 7. Flujo completo de la fase de producción e inferencia. ____ *14*
Figura 8. Tabla comparación de modelos ____ *22*
Figura 9. Distribución del ROC AUC por Modelo ____ *23*
Figura 10. Análisis Final del Modelo Seleccionado ____ *24*
Figura 11. Matriz de confusion ____ *24*

1. Introducción y Motivación

El presente proyecto intermodular tiene como objetivo fundamental el desarrollo integral de un sistema de Inteligencia Artificial orientado a predecir la baja (churn) de clientes en una compañía telefónica. El churn, o la pérdida de clientes, es uno de los principales desafíos para las empresas de telecomunicaciones, dado que retener clientes existentes suele ser más rentable que adquirir nuevos. Esta problemática de negocio tiene un impacto directo en la rentabilidad y sostenibilidad de la compañía, y su gestión efectiva se traduce en estrategias más focalizadas, mejores ofertas y una mayor fidelización.

Para abordar esta necesidad, el proyecto se apoya en técnicas avanzadas de procesamiento de datos, modelado predictivo y gestión de flujos de trabajo, con una implementación robusta y escalable que se adapta a situaciones reales, tanto en procesos batch como en streaming. La combinación de estas tecnologías permite no solo anticipar las bajas, sino también actuar de forma proactiva para mejorar la retención y personalizar las campañas de marketing.

El desarrollo de este sistema parte de la premisa de que una correcta identificación de los clientes con mayor riesgo de abandono habilita a la compañía para diseñar estrategias más efectivas y segmentadas. Con ello, se logra un impacto directo en la satisfacción del cliente y en la optimización de los recursos internos, alineándose con los objetivos empresariales de eficiencia y rentabilidad.

El alcance del proyecto abarca desde la exploración inicial y detallada de los datos hasta la producción final de modelos operativos en entornos controlados y simulados, integrando metodologías adecuadas de validación y optimización para asegurar un rendimiento consistente y fiable. La solución propuesta destaca por su modularidad, escalabilidad y facilidad de mantenimiento, facilitando así futuras expansiones o adaptaciones a otros contextos empresariales similares. Asimismo, el proyecto incluye un enfoque crítico y argumentado sobre cada decisión tomada durante el desarrollo, permitiendo comprender en profundidad las ventajas y limitaciones inherentes a las soluciones técnicas adoptadas.

2. Entorno tecnológico

Para el desarrollo integral del sistema predictivo propuesto en este proyecto, se han seleccionado diversas herramientas y tecnologías que aseguran eficiencia, escalabilidad, automatización y fácil mantenimiento. Apache Airflow es la elección principal para orquestar los procesos de entrenamiento e inferencia mediante pipelines robustos y automatizados. Docker facilita la encapsulación y despliegue de servicios independientes, garantizando la consistencia del entorno de ejecución y una comunicación eficiente entre componentes aislados.

FastAPI, junto con Uvicorn, proporciona un framework ligero y eficaz para exponer modelos predictivos mediante APIs REST, permitiendo la interacción sencilla y segura con otras aplicaciones. Apache Kafka asegura una gestión eficiente del flujo de datos en tiempo real, facilitando procesos de streaming confiables y escalables.

El proceso de entrenamiento y optimización del modelo se realiza utilizando Optuna, una biblioteca diseñada específicamente para la optimización de hiperparámetros mediante algoritmos avanzados. MLFlow proporciona una plataforma robusta para registrar, gestionar y versionar experimentos, asegurando reproducibilidad y trazabilidad completa del proceso.

Las bases de datos elegidas para almacenamiento y gestión de datos son TimescaleDB, especializada en series temporales y adecuada para guardar predicciones batch, y MongoDB, orientada a documentos, utilizada para manejar datos en streaming mediante una arquitectura replicada que garantiza alta disponibilidad y tolerancia a fallos.

Adicionalmente, Python se utiliza como lenguaje principal, respaldado por bibliotecas ampliamente reconocidas en el ecosistema de ciencia de datos como pandas, para manipulación de datos estructurados; seaborn, para visualización estadística avanzada; y scikit-learn, esencial para desarrollar modelos de aprendizaje automático. También se emplean Jupyter Notebooks para exploración inicial, documentación del desarrollo y prototipado rápido.

También se utilizan herramientas como VSCode, ClickUp y GitHub para gestión ágil del código, tareas y versiones.

3.Fase de entrenamiento

3.1. Exploración de los datos

La fase exploratoria inicial tuvo como objetivo obtener un entendimiento detallado de los datos proporcionados para el proyecto, específicamente los archivos clientes.csv y servicios.xlsx. Este análisis es crítico para identificar patrones importantes, sesgos en los datos y la presencia de valores atípicos o faltantes que podrían afectar negativamente el rendimiento del modelo predictivo.

En primer lugar, se realizó una revisión inicial de las dimensiones de ambos datasets para evaluar la magnitud y complejidad de los mismos. Se obtuvieron estadísticas básicas, incluyendo tipos de datos, cantidades de registros, así como la identificación precisa de columnas con valores faltantes. Este análisis reveló la necesidad de gestionar valores nulos mediante técnicas específicas posteriormente.

Posteriormente, se analizaron en profundidad las distribuciones de variables numéricas críticas como Cargos Mensuales y Cargos Totales. La exploración visual de estas variables mediante histogramas

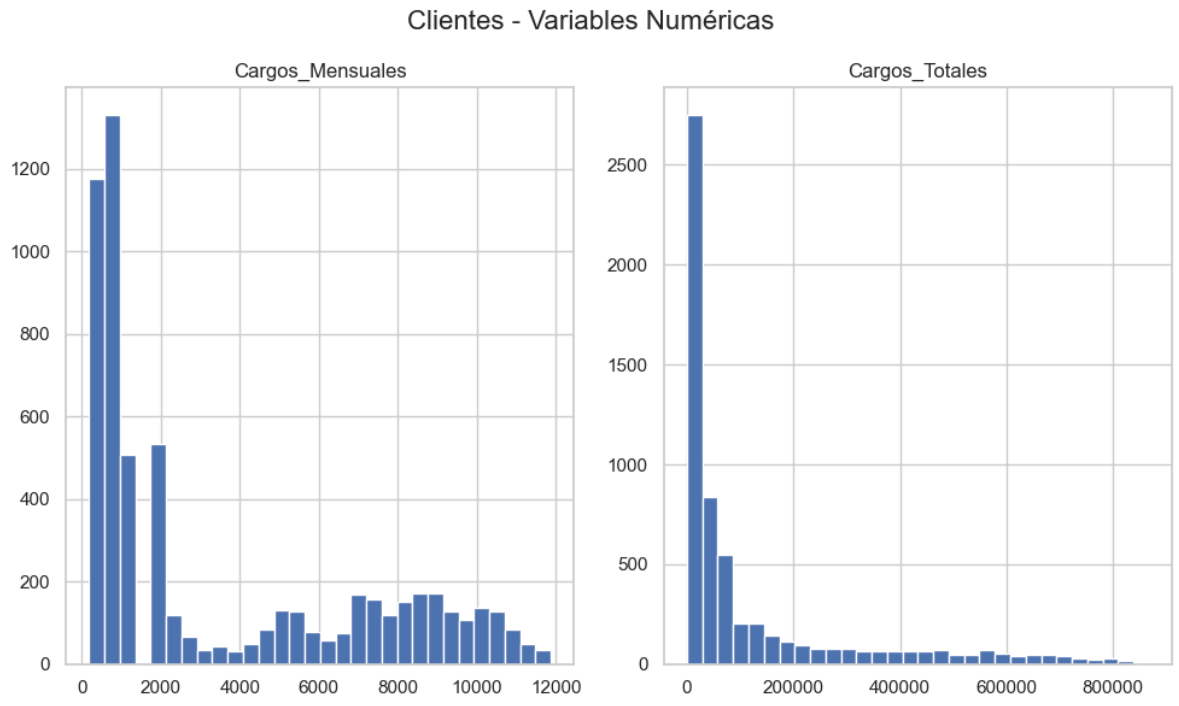

Figura 1.Cargos Totales y Mensuales por cliente

mostró distribuciones significativamente sesgadas hacia la derecha, indicando que la mayoría de clientes presentan cargos relativamente bajos, con pocos casos extremos de altos costos.

Para complementar este análisis, se generaron diagramas de caja para visualizar claramente la presencia y magnitud de valores atípicos en estas variables numéricas en relación con el abandono del cliente.

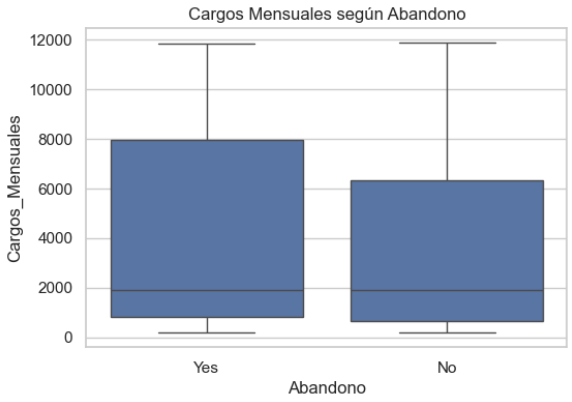

Figura 2.Cargos mensuales por Abandono

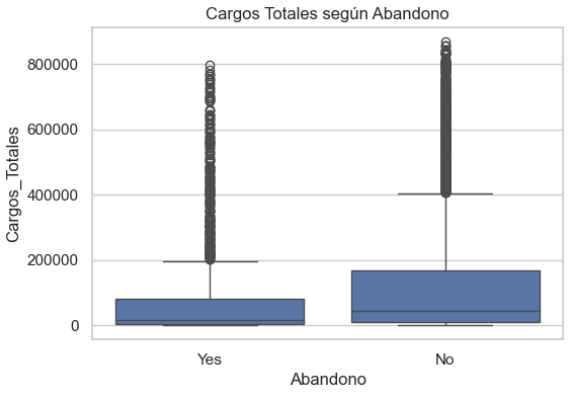

Figura 3. Cargos totales por abandono

Estos gráficos confirmaron visualmente que los clientes con cargos totales y mensuales elevados tienden a presentar una tasa diferente de abandono, lo cual es relevante para la posterior fase de modelado.

Asimismo, se llevó a cabo un análisis de variables categóricas importantes, tales como Jubilado, Pareja y Dependientes. Este análisis reveló patrones relevantes como que los clientes jubilados, aquellos con pareja o con dependientes, muestran tasas de abandono significativamente menores. Estos hallazgos fueron ilustrados claramente mediante gráficos de conteo por categorías.

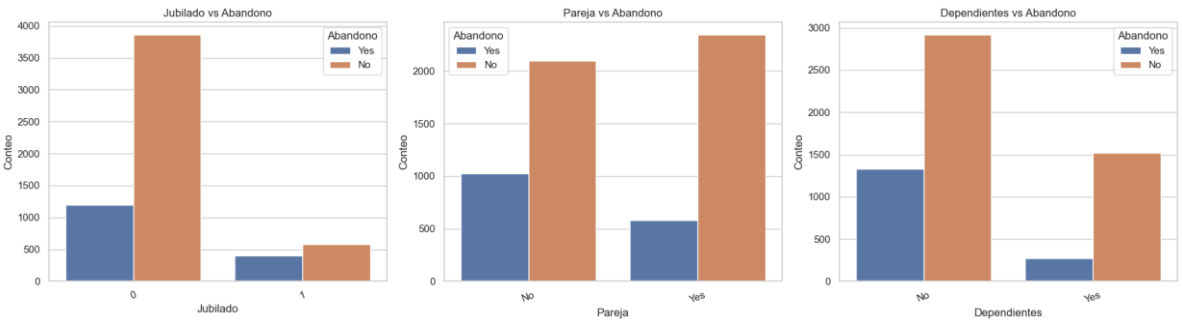

Figura 4. Estado civil Por abandono

Finalmente, se realizó la integración de ambos datasets (clientes y servicios) mediante la clave ID_Cliente para un análisis combinado más integral. Esto permitió estudiar variables específicas de servicios contratados, tales como Seguridad Online, Soporte Técnico y Método de Pago. Se encontró una correlación fuerte entre ciertas modalidades de servicio (por ejemplo, la falta de seguridad online o soporte técnico, contratos mensuales, y pagos mediante cheque elect rónico) y mayores tasas de abandono. Estos resultados se representan visualmente con gráficos de barras.

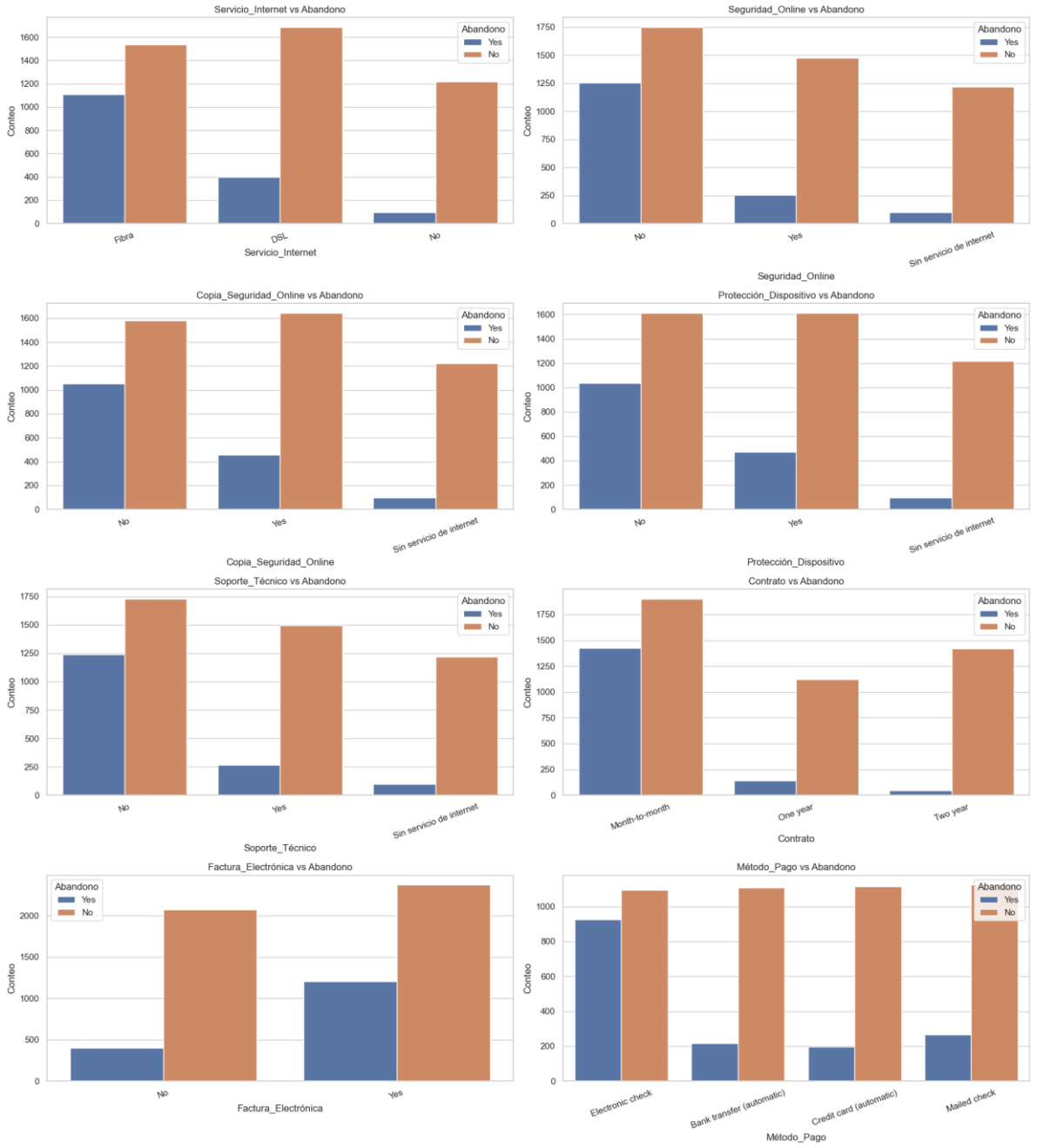

Figura 5. Distribución de abandono según variables de servicios y contrato

Este detallado análisis exploratorio proporcionó una sólida base para las siguientes fases del proyecto, incluyendo el preprocesamiento de datos, la selección de características relevantes, y el entrenamiento informado del modelo predictivo.

3.2. Revision bibliográfica

La revisión bibliográfica tuvo como objetivo sustentar y justificar las decisiones metodológicas del proyecto mediante la evaluación crítica de estudios clave sobre la predicción de churn en el sector de telecomunicaciones.

Un estudio relevante para este proyecto fue el de Ahmad et al. (2019), que destaca la eficacia del algoritmo XGBoost sobre otros algoritmos como Random Forest y Gradient Boosting, aplicando técnicas avanzadas como la ingeniería de características desde grandes volúmenes de datos en plataformas Big Data. Este estudio fundamenta especialmente el uso de XGBoost y técnicas de ingeniería de características para mejorar significativamente el rendimiento del modelo.

El trabajo de Sikri et al. (2024) aporta técnicas para manejar conjuntos de datos desequilibrados, proponiendo una técnica novedosa de balanceo de datos basada en ratios, la cual demostró ser más efectiva comparada con métodos tradicionales. Este estudio justifica las decisiones de aplicar técnicas avanzadas de balanceo como oversampling y undersampling, vitales para asegurar la precisión del modelo de predicción en escenarios reales.

Finalmente, el estudio de Óskarsdóttir et al. (2017) enfatiza la importancia del análisis de redes sociales en la predicción de churn, destacando cómo el uso de información derivada de redes puede mejorar significativamente el rendimiento predictivo. Este análisis fue crucial para justificar la integración de datos relacionales, como interacciones entre clientes, dentro del modelo predictivo, fortaleciendo así la capacidad del sistema para capturar dinámicas complejas y factores indirectos que influyen en el abandono de los clientes.

3.3. Preprocesamiento de Datos

La fase de preprocesamiento es una etapa crítica para garantizar el óptimo rendimiento del modelo predictivo. Durante esta etapa, se realizaron diversas transformaciones destinadas a mejorar la calidad y efectividad de los datos utilizados en el entrenamiento del modelo.

Inicialmente, se abordó el problema de los valores faltantes, que pueden generar sesgos o disminuir la capacidad predictiva del modelo. Para las variables numéricas se optó por imputar los valores faltantes utilizando la mediana de cada variable, ya que esta medida es menos sensible a la presencia de valores atípicos en comparación con la media, permitiendo mantener una representación más robusta y equilibrada de los datos originales. En cuanto a las variables categóricas, se utilizó el término "desconocido" para rellenar aquellos valores faltantes, asegurando así que el modelo reconozca explícitamente la ausencia de información y pueda capturar cualquier patrón relevante asociado a dicha categoría.

Posteriormente, se llevaron a cabo procesos de normalización y estandarización utilizando técnicas específicas como StandardScaler, MinMaxScaler y RobustScaler. El uso de estos escaladores es esencial, ya que permite homogeneizar las escalas de las variables numéricas, evitando que aquellas con mayores rangos dominen excesivamente en el entrenamiento del modelo y, por tanto, garantizando una contribución más equitativa de todas las variables. En particular, el RobustScaler resultó especialmente útil al tratar variables con valores atípicos considerables, al reducir significativamente su impacto.

Un paso adicional fundamental en este proceso fue la selección de características, realizada mediante el método estadístico SelectKBest. Este método evalúa la relación de cada característica con el objetivo (abandono del cliente) mediante pruebas estadísticas, seleccionando únicamente aquellas variables que muestran una influencia significativa. Este procedimiento ayuda a reducir la dimensionalidad del conjunto de datos, mejorando la eficiencia computacional y aumentando la precisión del modelo, al centrarse únicamente en las variables más relevantes y disminuir el ruido en los datos.

Además, se realizaron transformaciones necesarias para variables categóricas mediante la técnica de LabelEncoder. Esta técnica transforma las variables categóricas en un formato numérico que pueda ser interpretado eficazmente por los algoritmos de aprendizaje

automático, asegurando así que el modelo capture adecuadamente la información contenida en estas variables.

Finalmente, se abordó el problema común en tareas predictivas de churn, que es el desequilibrio entre las clases (clientes que abandonan frente a los que permanecen). Para manejar este desafío, se aplicaron técnicas avanzadas de resample, que generan muestras equilibradas incrementando el número de ejemplos de la clase minoritaria mediante oversampling. Este proceso asegura que el modelo aprenda patrones robustos y representativos de ambas clases, aumentando así la efectividad del modelo para identificar correctamente tanto los clientes propensos a abandonar como aquellos que probablemente permanecerán fieles a la compañía.

Cada una de estas decisiones metodológicas fue tomada cuidadosamente considerando tanto el contexto de negocio como las características específicas de los datos, asegurando así que el modelo final sea preciso, robusto y confiable.

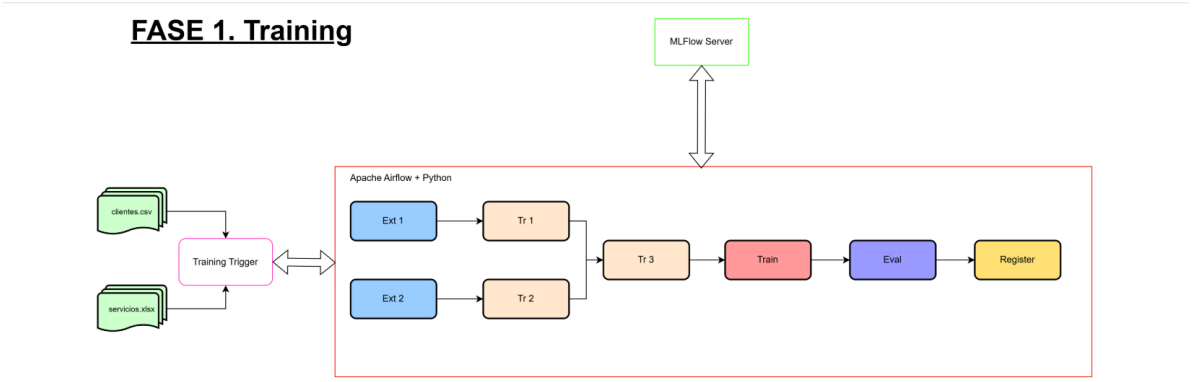

Figura 6Diagrama general del flujo de entrenamiento.

3.4. Entrenamiento del modelo

La fase de entrenamiento del modelo representa uno de los pasos más críticos y exigentes del proyecto, ya que tiene como objetivo identificar y desarrollar el mejor modelo predictivo posible para anticipar la baja (churn) de clientes. Esta fase incluye la cuidadosa elección de modelos, estrategias de validación robustas, técnicas avanzadas de optimización de hiperparámetros y un enfoque riguroso en la evaluación de métricas clave.

Primero, se seleccionaron explícitamente cinco modelos representativos —RandomForest, LogisticRegression, SVC, XGBoost y CatBoost— debido a su eficacia y versatilidad en problemas de clasificación complejos. La decisión de fijar explícitamente el modelo a entrenar, en lugar de dejar que Optuna lo elija, permite una comparación justa y directa entre enfoques (árboles de decisión, regresión lineal, boosting y SVM). Así se obtiene una visión clara del comportamiento de cada modelo en este problema particular y se garantiza que la comparación se haga bajo las mismas condiciones experimentales.

Para evitar problemas como el sobreajuste y asegurar la capacidad de generalización, se emplearon dos estrategias complementarias de validación: la división de los datos en conjuntos de entrenamiento y prueba, y la validación cruzada estratificada mediante StratifiedKFold, que garantiza la proporcionalidad de las clases en cada fold. Este doble enfoque permite medir la estabilidad y la capacidad de generalización del modelo.

En la optimización de hiperparámetros se recurrió a Optuna, una biblioteca de optimización automatizada basada en técnicas de búsqueda bayesiana y pruning dinámico. Aquí es importante aclarar que Optuna no escoge qué modelo usar: su papel es explorar el espacio de hiperparámetros para encontrar la mejor configuración posible dentro de cada modelo, garantizando un ajuste fino que maximice el área bajo la curva ROC (AUC), métrica elegida por su relevancia en problemas con clases desbalanceadas.

Además, se aprovechó el pruning dinámico de Optuna (early stopping). Este mecanismo permite a Optuna detener automáticamente ensayos que muestran un rendimiento inferior en las primeras etapas, ahorrando tiempo y recursos de cómputo y garantizando una búsqueda más eficiente en el espacio de hiperparámetros.

Cada iteración del entrenamiento registró tanto el AUC de validación como el de entrenamiento, midiendo la brecha entre ambos para identificar posibles signos de sobreajuste. Una vez identificada la mejor configuración, se entrenó el modelo final y se generaron métricas complementarias precisión, recall, F1-score y la matriz de confusión, proporcionando un panorama completo de su rendimiento. Además, se generaron visualizaciones del proceso de optimización de Optuna (historial y relevancia de los parámetros) para documentar la lógica detrás de las configuraciones elegidas.

3.5. Registros de experimentos

Finalmente, para garantizar la trazabilidad y el control de versiones, se integró todo este proceso con MLflow. Esto permitió registrar los parámetros de configuración, las métricas de rendimiento y todos los artefactos generados (datasets finales, modelos serializados, gráficos de optimización y la matriz de confusión), asegurando la reproducibilidad y la trazabilidad de los experimentos. Para la selección final del mejor modelo, se establecieron umbrales de desempeño mínimo (ROC AUC ≥ 0.80 y F1 Score ≥ 0.80), de forma que solo se registran y versionan los modelos que cumplen con los requisitos de calidad necesarios para ser llevados a producción.

En resumen, esta fase demuestra un equilibrio entre la exploración meticulosa de distintos modelos, el uso de herramientas de optimización avanzada y la integración de mecanismos de validación y trazabilidad que garantizan la fiabilidad y solidez del flujo de entrenamiento.

4. Fase de producción

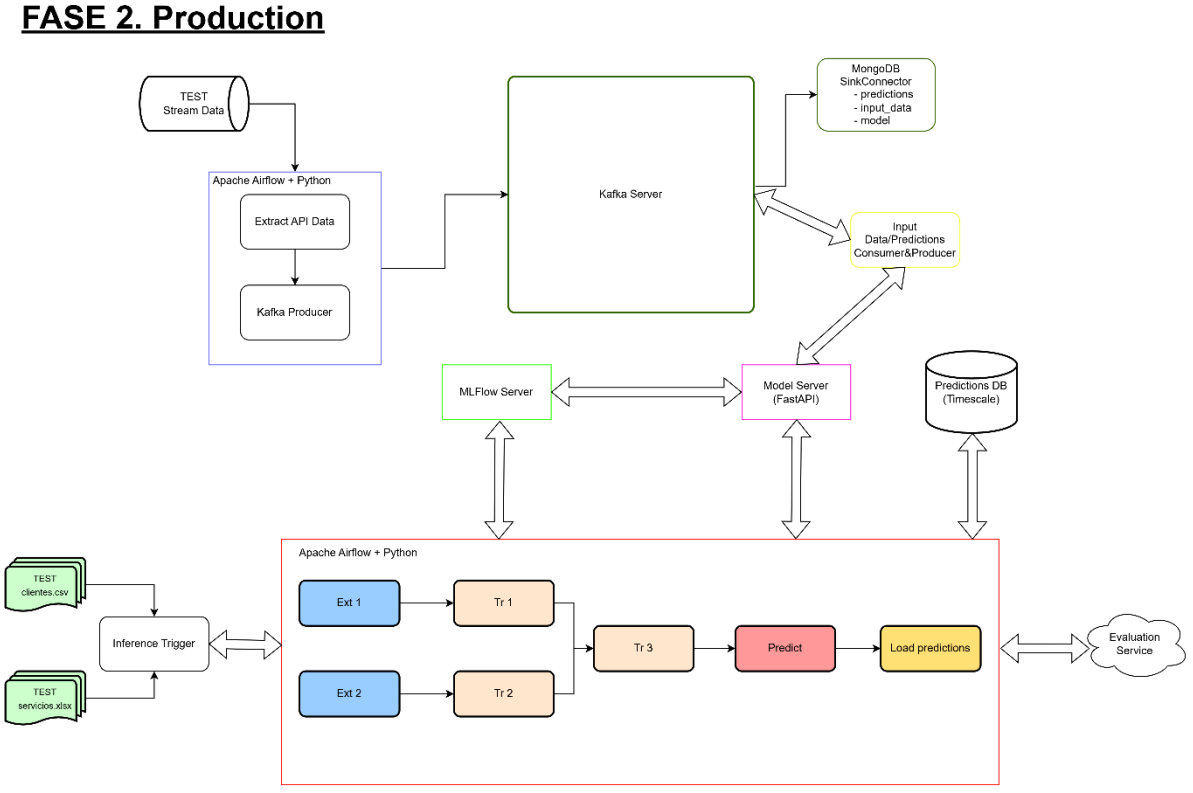

Figura 7. Flujo completo de la fase de producción e inferencia.

4.1 Rama Batch

La rama batch del proyecto es una fase crucial y cuidadosamente diseñada que permite ejecutar la inferencia del modelo predictivo de churn de manera flexible y configurable. Todo el flujo está orquestado mediante Apache Airflow, lo que facilita la gestión, el control y la automatización de cada tarea en el pipeline.

Desde la carga inicial de los datos de clientes y servicios hasta la transformación final de los mismos, el pipeline aprovecha la modularidad y flexibilidad de Python y Airflow para adaptarse a múltiples configuraciones de entrada. Gracias a la integración con MLflow, el sistema puede reutilizar artefactos como los modelos, los encoders, los escaladores y las columnas seleccionadas, asegurando que el proceso de inferencia sea coherente y esté alineado con las condiciones de entrenamiento.

Cada DAG está preparado para aceptar configuraciones variables, incluyendo el uso de kbest, el método de escalado, y la selección de columnas relevantes. Estas configuraciones, almacenadas en MLflow como parámetros de los modelos registrados, son automáticamente recuperadas al inicio de cada ejecución, lo que permite adaptar dinámicamente el proceso de inferencia a las necesidades del negocio o a nuevos experimentos de optimización. Esta capacidad de personalización es especialmente importante en entornos de producción, donde pueden surgir requisitos cambiantes o nuevas oportunidades de mejora.

La fase de inferencia se realiza a través de una API expuesta mediante FastAPI, la cual recibe los datos transformados y devuelve predicciones para cada cliente. Esta API funciona como un microservicio aislado, permitiendo la escalabilidad y la integración con otros sistemas externos. Una vez obtenidas las predicciones, estas se almacenan de manera estructurada en TimescaleDB, una base de datos optimizada para datos temporales, lo que facilita el seguimiento longitudinal de las predicciones y permite análisis avanzados de tendencias a lo largo del tiempo.

Además, el pipeline incluye un paso de evaluación externa que utiliza un contenedor proporcionado por el profesor. Este contenedor actúa como una fuente de validación independiente, proporcionando métricas como el AUC para garantizar la calidad y la fiabilidad de las predicciones. Gracias a esta evaluación, se pueden identificar posibles

desviaciones o problemas de rendimiento antes de que las predicciones lleguen a producción.

El diseño modular y la capacidad de adaptar configuraciones de este flujo batch permiten al sistema mantener altos estándares de calidad y flexibilidad, características fundamentales para entornos de telecomunicaciones dinámicos y en constante evolución.

4.2. Rama Streaming

La rama de streaming constituye un pilar esencial del sistema de producción de inferencia, diseñada para procesar datos de forma continua y en tiempo real. Esta arquitectura permite la ingesta y procesamiento de datos dinámicos, garantizando una respuesta inmediata y una integración fluida con el resto del sistema.

El proceso comienza con un contenedor proporcionado por el tutor, que expone una URL (http://host.docker.internal:82/dato) en el puerto 82. Este endpoint ofrece datos en formato JSON, representando nuevas observaciones de clientes a predecir. Para capturar esta información, se ha implementado un pipeline en Apache Airflow, el cual consta de dos tareas principales: extract_data y send_to_kafka. La primera tarea, extract_data, realiza la llamada al endpoint y almacena la respuesta JSON en el XCom, permitiendo compartir los datos entre tareas. La segunda, send_to_kafka, extrae el JSON y lo publica en el topic de Kafka clientes-dato, mediante un productor configurado para serializar los datos en formato JSON. Esta separación en dos tareas facilita la modularidad, el mantenimiento y la escalabilidad del sistema.

Una vez que los datos son publicados en Kafka, se activa el flujo continuo mediante un consumer (desarrollado en Python y empaquetado en un contenedor de Docker). Este consumer escucha permanentemente el topic clientes-dato, donde recibe los mensajes JSON con la información de los clientes. Cada mensaje es procesado cuidadosamente para garantizar que los datos estén listos para la inferencia. En este paso, se realiza la normalización de claves a minúsculas y se aplica un proceso de limpieza configurable (por ejemplo, eliminación de nulos o imputación mediante la mediana).

Además, el consumer carga los artefactos del entrenamiento como el scaler, los label encoders y la selección de características (KBest) desde un directorio de artefactos compartido. Gracias a esto, puede aplicar exactamente las mismas transformaciones que se utilizaron durante el entrenamiento del modelo, garantizando la coherencia entre la fase de entrenamiento y la de inferencia. La flexibilidad está asegurada, ya que la configuración (por ejemplo, metodo_escalado, uso de KBest, etc.) puede adaptarse fácilmente a distintos escenarios.

Tras el preprocesamiento, el consumer envía los datos preparados al servicio de inferencia FastAPI desplegado en un contenedor. FastAPI está configurado para cargar dinámicamente el modelo de MLFlow registrado con la versión más reciente, garantizando siempre que las predicciones se realicen con el mejor modelo disponible. La llamada al endpoint /predict se realiza en tiempo real, y FastAPI devuelve las predicciones generadas. Este enfoque modular permite desacoplar la predicción del procesamiento de datos, facilitando futuras actualizaciones del modelo sin necesidad de modificar el pipeline de inferencia.

Una vez obtenidas las predicciones, el consumer actúa como reproductor, reempaquetando los datos originales junto con las predicciones obtenidas y enviándolos nuevamente a un topic de Kafka llamado predicciones-modelo. Esta práctica asegura que los datos procesados y las predicciones se mantengan juntos, facilitando auditorías y permitiendo la monitorización completa del flujo de datos.

Finalmente, este topic predicciones-modelo está configurado para interactuar con un replicaset de MongoDB, que actúa como sistema de almacenamiento persistente. MongoDB permite guardar tanto las predicciones como los datos originales en un esquema flexible y fácilmente escalable. Esto asegura que la información esté siempre disponible para futuras consultas, análisis de rendimiento o generación de reportes.

Todo este flujo desde la extracción de datos del endpoint, pasando por Kafka, el procesamiento en el consumer, la predicción con FastAPI y el almacenamiento final en MongoDB está cuidadosamente diseñado para garantizar robustez, escalabilidad y trazabilidad. La elección de tecnologías modernas como Kafka, FastAPI y MongoDB se fundamenta en su alta disponibilidad, capacidad para procesar grandes volúmenes de datos en tiempo real y facilidad de integración con otros sistemas empresariales.

Además, la arquitectura es completamente modular y configurable. Gracias a la estructura de artefactos compartidos y la separación de tareas en Airflow, es posible ajustar de forma sencilla cualquier parámetro de limpieza, transformación o inferencia sin necesidad de rehacer todo el flujo. Esta flexibilidad resulta clave para adaptarse a posibles cambios en la naturaleza de los datos o en los requisitos de negocio.

En conclusión, la rama de streaming combina automatización, eficiencia y flexibilidad para ofrecer predicciones en tiempo real con la máxima confiabilidad, integrándose perfectamente en un sistema de producción sólido y preparado para los retos del mundo real.

5. Arquitectura del Proyecto

La arquitectura del proyecto ha sido cuidadosamente diseñada para asegurar la modularidad, escalabilidad y facilidad de mantenimiento. Esta estructura combina la potencia de herramientas de orquestación de flujos de trabajo, como Airflow, con la flexibilidad de procesamiento en tiempo real y por lotes mediante FastAPI y Kafka, y el registro y trazabilidad de experimentos gracias a MLflow. El uso de contenedores Docker facilita la replicabilidad de los entornos y garantiza la consistencia del sistema en diferentes máquinas y entornos.

Diagrama General del Flujo de Trabajo

El diagrama adjunto (ver más arriba) representa la interconexión de los componentes clave:

Airflow actúa como el orquestador principal, gestionando tanto los flujos de entrenamiento como los de inferencia, ya sea en batch o streaming.

Kafka sirve como sistema de mensajería, permitiendo la ingestión y procesamiento de datos en tiempo real mediante topics y la separación de productores y consumidores.

FastAPI es el motor de predicción, exponiendo el modelo entrenado como un endpoint REST para recibir datos, procesarlos y devolver predicciones.

MLflow actúa como sistema central de registro y control de versiones de modelos y experimentos, asegurando la trazabilidad y la calidad de los modelos.

TimescaleDB y MongoDB gestionan el almacenamiento de datos y resultados, adaptándose a las necesidades de predicciones batch y streaming, respectivamente.

Dockerización y Comunicación Entre Servicios

El proyecto utiliza Docker y docker-compose para definir y desplegar todos los componentes necesarios. Cada servicio se encuentra contenido en su propio contenedor, garantizando independencia y evitando conflictos de dependencias. Esto permite además la facilidad de despliegue en distintos entornos (desarrollo, testing, producción).

El archivo docker-compose.yaml define los siguientes servicios:

Airflow: Incluye los contenedores de webserver, scheduler, postgres para la base de datos de metadatos, y scripts de inicialización. Airflow está configurado con la base de datos de PostgreSQL y montajes de volúmenes para almacenar los DAGs y resultados.

FastAPI: Se levanta en un contenedor independiente con su propio Dockerfile, expuesto en el puerto 8000. Utiliza mlflow.pyfunc.load_model para cargar dinámicamente la última versión del modelo desde el registro de MLflow.

Kafka: Incluye los servicios de broker, schema-registry, control-center, ksqldb-server, ksqldb-cli, rest-proxy y connect. Esto permite tener un ecosistema completo para gestionar la mensajería en tiempo real y la integración con otras fuentes (por ejemplo, MongoDB como sink a través de Kafka Connect).

TimescaleDB y MongoDB: Se utilizan para almacenar los resultados de las predicciones batch y streaming respectivamente. MongoDB está configurado como un replicaset para garantizar alta disponibilidad.

Consumer de Kafka: Desplegado en un contenedor propio (kafka-consumer), este componente consume los datos enviados por Airflow y los procesa para predicciones en tiempo real.

La red kafka-net compartida asegura que todos estos servicios puedan comunicarse de manera eficiente, y la exposición de puertos específicos permite su gestión externa.

Modularización y Estructura del Repositorio

El proyecto está organizado en carpetas que permiten una clara separación de responsabilidades:

/Code/airflow/dags/: contiene los DAGs principales, organizados en módulos (modules para procesamiento batch, inference_modules para la inferencia).

/Code/docker/: contiene los Dockerfile y configuraciones de contenedores.

/Code/docker/fastapi/: contiene la API de inferencia y predicciones.

/Code/docker/dockerfiles/: agrupa Dockerfiles de componentes como Kafka Connect y el consumidor de Kafka.

/datos/: almacena datasets y artefactos como los scaler, encoders y columnas, que son utilizados en las transformaciones y la inferencia.

/Biografia/ y /Analisis_exploratorio/: contienen los estudios de referencia y notebooks de EDA para la fase inicial de análisis de datos.

Gracias a esta modularización, cada componente puede ser actualizado o sustituido de manera independiente, lo cual es esencial para la mantenibilidad y escalabilidad del sistema. Además, Airflow reutiliza artefactos almacenados en MLflow (como el scaler o las columnas de entrenamiento) asegurando consistencia en todo el flujo, desde el entrenamiento hasta la predicción en producción.

Comunicación Detallada Entre Servicios

El flujo de datos es el siguiente:

Entrenamiento y registro del modelo: Airflow ejecuta DAGs que entrenan modelos con Optuna para optimización de hiperparámetros y almacenan los artefactos (scaler, encoders, columnas) en MLflow.

Predicciones batch: Airflow ejecuta un DAG de inferencia batch, que reutiliza las transformaciones y el modelo almacenado en MLflow. Los resultados se almacenan en TimescaleDB y se validan con un servicio externo proporcionado por el profesor.

Predicciones en streaming: Airflow extrae datos periódicamente desde un contenedor externo y los publica en Kafka. El kafka-consumer recoge estos datos, aplica las transformaciones necesarias (usando artefactos del entrenamiento almacenados localmente) y los envía a FastAPI para obtener predicciones. Las predicciones se reenvían a un nuevo topic de Kafka y se almacenan finalmente en MongoDB.

MLflow: Durante todas las fases, MLflow actúa como repositorio centralizado para controlar el versionado de modelos y asegurar la trazabilidad de experimentos, modelos y configuraciones.

Este diseño modular y cuidadosamente integrado permite al proyecto cumplir con los más altos estándares de trazabilidad, escalabilidad y mantenibilidad. Además, facilita la adaptabilidad del sistema a futuros cambios o necesidades empresariales, manteniendo siempre una base robusta y confiable para la predicción de churn en la compañía telefónica.

6. Resultados

En este apartado se analizan los resultados obtenidos durante la fase de entrenamiento y evaluación de los modelos, así como la comparación entre ellos y una reflexión sobre el rendimiento y la escalabilidad del sistema.

6.1 Análisis general de los resultados obtenidos

Durante el proceso de entrenamiento se llevaron a cabo múltiples ejecuciones con distintos modelos y configuraciones. Para cada modelo, se registraron y analizaron métricas clave como el área bajo la curva ROC (AUC), la precisión, la exhaustividad (recall), el F1-score y la exactitud (accuracy), además de las diferencias entre el AUC de entrenamiento y el de validación (diferencia_auc). Estos datos permiten evaluar no sólo la calidad predictiva sino también la estabilidad y generalización de cada modelo.

El análisis general muestra que los modelos RandomForestClassifier y CatBoostClassifier ofrecieron los resultados más consistentes y robustos. En particular, las mejores ejecuciones de RandomForest lograron valores de AUC_test superiores a 0.97, con un F1-score cercano a 0.91, indicando un balance adecuado entre precisión y exhaustividad. Por su parte, CatBoost también presentó métricas competitivas, aunque ligeramente por debajo de los mejores resultados de RandomForest.

Cabe destacar que la diferencia entre el AUC de entrenamiento y el de validación (diferencia_auc) fue generalmente pequeña en los modelos seleccionados como óptimos, lo que sugiere una buena capacidad de generalización sin sobreajuste excesivo. Estos aspectos son fundamentales para garantizar la robustez del modelo en situaciones reales.

6.2 Comparación entre modelos

A continuación se presenta un análisis comparativo de los principales modelos:

MODELO	AUC TEST	ROC AUC	F1-SCORE	PRECISIÓN	RECALL	DIFERENCIA AUC	ACCURACY
RANDOMFOREST (MEJOR RUN)	0.9696	0.9710	0.906	0.850	0.968	-0.0286	0.899
RANDOMFOREST (OTROS)	0.9105	0.9112	0.874	0.822	0.931	-0.0654	0.865
CATBOOST	0.9339	0.9376	0.890	0.846	0.939	-0.0554	0.884

Figura 8. Tabla comparación de modelos

De esta tabla se evidencia claramente que el modelo RandomForestClassifier en su mejor ejecución superó a los demás tanto en AUC_test como en ROC AUC y F1-score. Esto confirma su solidez en la clasificación y su capacidad para mantener un alto nivel de precisión y recall, fundamentales en problemas de churn donde es crítico detectar correctamente a los clientes propensos a abandonar.

Por otro lado, CatBoost también mostró un rendimiento muy competitivo, destacando especialmente por su F1-score cercano al 0.89 y un buen equilibrio entre precisión y recall. Sin embargo, no alcanzó los niveles máximos de AUC y F1-score de RandomForest.

El resto de las ejecuciones de RandomForest con otras configuraciones de hiperparámetros mantuvieron un rendimiento aceptable, aunque con ligeras caídas en la precisión y recall. Estas caídas se deben principalmente a la sensibilidad del modelo a la configuración de profundidad y número de árboles (n_estimators), aspectos que fueron afinados en la búsqueda de la mejor versión con Optuna.

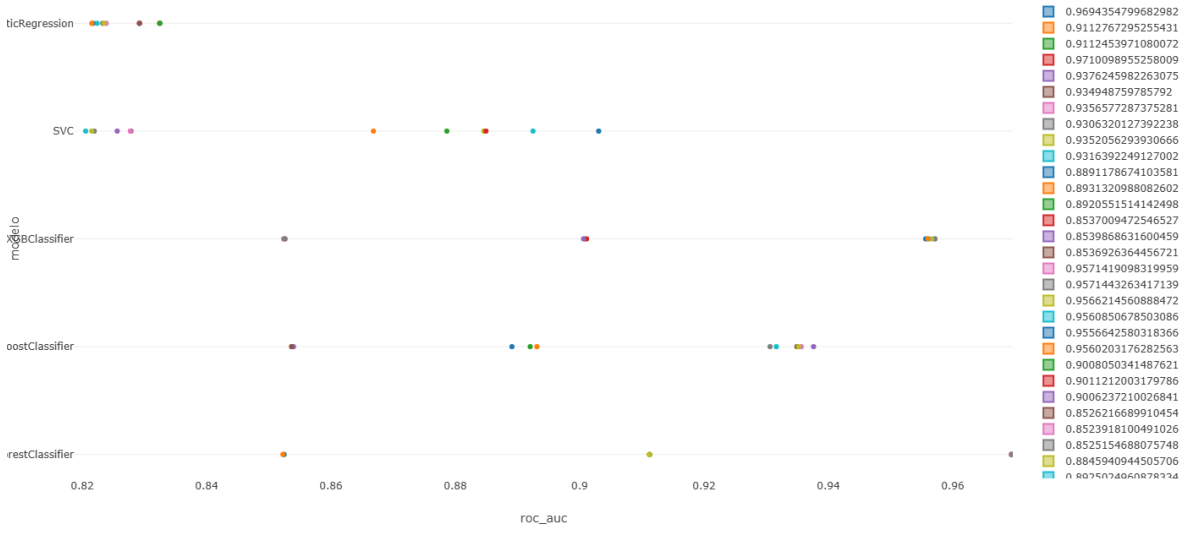

Figura 9. Distribución del ROC AUC por Modelo

6.3 Modelo final y justificación de la selección

El modelo final seleccionado es un RandomForestClassifier con los siguientes hiperparámetros optimizados: n_estimators=145, max_depth=15 y usando un escalado MinMaxScaler, sin aplicar reducción de dimensionalidad con KBest. Este modelo alcanzó un AUC_test de 0.9694, un ROC AUC de 0.9710 y un F1-score de 0.906, superando ampliamente los umbrales establecidos para calidad mínima (AUC ≥ 0.80 y F1-score ≥ 0.80).

La elección de este modelo se basa en su excelente desempeño global, en particular por su alto recall (0.9686) que garantiza la detección de la mayoría de los clientes en riesgo de abandono, algo crítico para la empresa. Además, la diferencia mínima entre el AUC de entrenamiento y validación (-0.0286) indica una adecuada generalización y ausencia de

sobreajuste significativo.

Figura 10. Análisis Final del Modelo Seleccionado

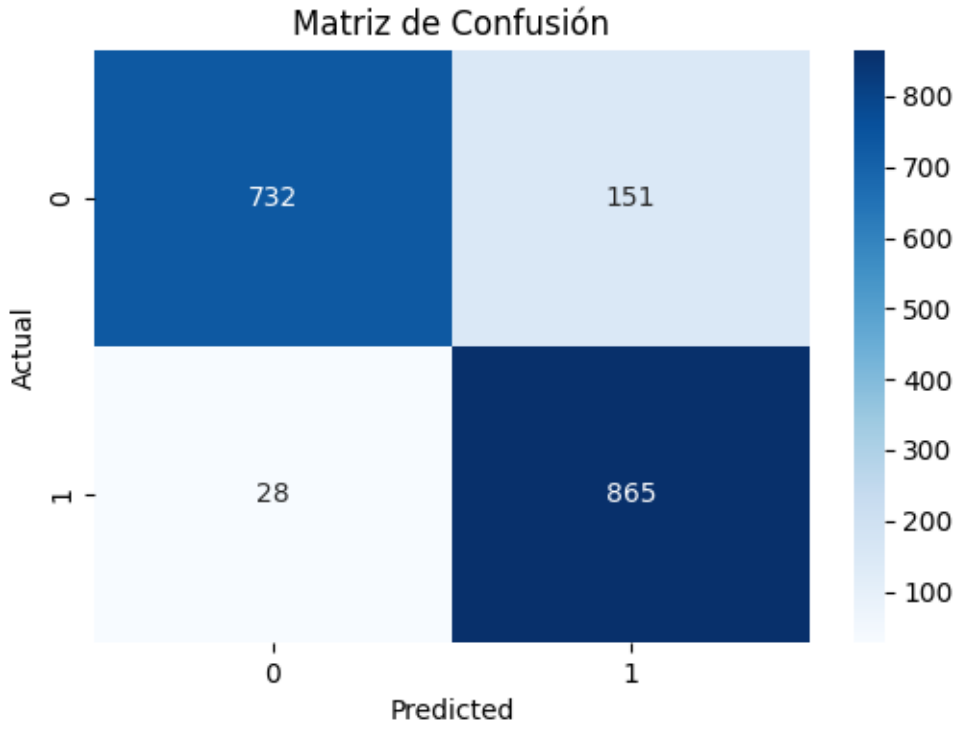

Figura 11. Matriz de confusion

6.4 Comentarios sobre rendimiento y escalabilidad

La estrategia seguida, basada en pipelines de entrenamiento y evaluación modulares, la integración de MLflow para el registro de experimentos y la selección automática del mejor modelo, asegura un flujo de trabajo reproducible y escalable. Cada componente (entrenamiento, inferencia batch y streaming, monitorización) está claramente separado, lo que permite adaptarse rápidamente a nuevas necesidades o cambios en los datos.

Por ejemplo, el uso de contenedores Docker y la orquestación mediante Airflow facilita la automatización y ejecución distribuida en distintas fases. Asimismo, la arquitectura modular con Kafka y FastAPI para la inferencia en streaming permite adaptarse a escenarios de alto volumen de datos sin comprometer la latencia de respuesta.

El uso de técnicas de optimización como Optuna ha sido clave para identificar configuraciones óptimas sin incurrir en procesos manuales extensos. Esto se traduce no solo en una mejora del rendimiento sino en una eficiencia operativa que facilita la actualización y mantenimiento del sistema en el futuro.

En resumen, los resultados obtenidos evidencian que el modelo final no solo cumple con los objetivos de precisión y recall, sino que también está respaldado por una arquitectura técnica robusta y bien documentada, lista para su implementación en entornos reales y para su evolución futura.

7. Limitaciones

Durante la ejecución del proyecto se encontraron algunas limitaciones importantes que podrían servir de base para futuras mejoras:

Imposibilidad de dockerizar MLflow: A pesar de que MLflow es una herramienta fundamental para el registro de experimentos y modelos, no fue posible integrarla completamente en el entorno dockerizado. Esto se debió a un problema de permisos relacionados con la escritura y lectura de volúmenes compartidos entre contenedores y el sistema operativo anfitrión (WSL en este caso). Como consecuencia, la solución adoptada consistió en dejar MLflow fuera de Docker y ejecutarlo en la máquina anfitriona, lo cual, aunque funcional, reduce la portabilidad y la estandarización de la solución completa.

Consumo elevado de recursos en la rama de streaming: La rama de streaming, encargada de capturar datos en tiempo real y procesarlos con el consumidor Kafka y FastAPI, demostró ser particularmente exigente en términos de recursos. Durante la ejecución de pruebas prolongadas, la combinación de múltiples contenedores en ejecución y la carga generada por los consumidores Kafka provocó ralentizaciones y, en ocasiones, bloqueos totales de

WSL. Esto obligó a reiniciar el subsistema con frecuencia para recuperar la estabilidad y reanudar las pruebas. Esta limitación pone de manifiesto la necesidad de optimizar el consumo de recursos o de migrar la infraestructura a entornos con mayores capacidades (por ejemplo, clusters en la nube o máquinas físicas más potentes) para garantizar la robustez y escalabilidad del sistema.

8. Conclusiones y Futuro Trabajo

8.1 Conclusiones obtenidas del proyecto

El desarrollo de este proyecto intermodular ha permitido implementar una solución completa y escalable para la predicción del churn en una compañía telefónica, abarcando todas las fases del ciclo de vida de un modelo de machine learning. Desde la exploración y transformación inicial de los datos, pasando por el entrenamiento y la validación de diversos modelos, hasta la puesta en producción y la inferencia tanto en modo batch como streaming, cada etapa ha sido cuidadosamente diseñada y documentada.

Uno de los logros más significativos del proyecto ha sido la consecución de un modelo de predicción altamente preciso y generalizable. Gracias al uso de técnicas avanzadas como la validación cruzada y la optimización de hiperparámetros con Optuna, se logró seleccionar un modelo RandomForest con métricas destacables: AUC_test de 0.969, un F1-score superior al 0.90 y un recall que garantiza la identificación de la mayoría de los casos de churn. Este rendimiento destaca no solo en términos de precisión, sino también en términos de estabilidad y ausencia de sobreajuste.

Otro punto clave ha sido la robustez de la arquitectura. La modularidad del sistema y la integración de herramientas como MLflow, Airflow, Docker y FastAPI han permitido construir un pipeline completo que no sólo predice con alta fiabilidad, sino que también está preparado para evolucionar y adaptarse a nuevos contextos. Además, la orquestación y la separación clara de responsabilidades aseguran que el sistema sea mantenible y escalable, requisitos fundamentales en entornos de producción reales.

Finalmente, la incorporación de un flujo de inferencia en tiempo real mediante Kafka y MongoDB demuestra la viabilidad del sistema en situaciones dinámicas y de gran volumen de datos, cumpliendo así con uno de los principales objetivos del proyecto: que la solución

no solo funcione en escenarios controlados, sino que esté lista para adaptarse a la realidad de una compañía telefónica en funcionamiento.

8.2 Posibles mejoras o extensiones del trabajo

Aunque los resultados obtenidos son muy satisfactorios, el proyecto deja abiertas diversas vías de mejora y extensión que podrían explorarse en el futuro:

Mejora continua del modelo: Si bien el modelo RandomForest ha mostrado un excelente rendimiento, se podría considerar la incorporación de modelos más avanzados como LightGBM o redes neuronales profundas (DNN) para comparar su rendimiento y explorar nuevas perspectivas de mejora. Asimismo, la inclusión de datos adicionales (por ejemplo, interacciones de clientes en redes sociales) podría enriquecer el modelo y aumentar aún más la precisión.

Explicabilidad y transparencia: La implementación de técnicas de interpretabilidad como SHAP o LIME permitiría explicar mejor las predicciones realizadas por el modelo, ofreciendo a los responsables de negocio no solo la predicción sino también el "por qué" detrás de ella. Esto es clave para la toma de decisiones informadas y para la confianza en el sistema.

Automatización del feedback loop: Se podría diseñar un mecanismo automático que, a partir de los resultados reales de churn observados, retroalimente el sistema y permita un reentrenamiento periódico, asegurando que el modelo se mantenga actualizado y adaptado a posibles cambios en el comportamiento de los clientes.

Integración de métricas de negocio: Incorporar métricas de negocio más específicas, como el Customer Lifetime Value (CLV) o el coste de adquisición de clientes, permitiría ponderar las predicciones de churn con base en el impacto económico, ofreciendo recomendaciones más estratégicas y alineadas con los objetivos empresariales.

Despliegue en nube o edge computing: Actualmente, la arquitectura está dockerizada y lista para funcionar en entornos controlados. Una futura extensión podría incluir el despliegue del sistema en plataformas cloud (AWS, GCP, Azure) o incluso en entornos edge computing, maximizando su escalabilidad y reduciendo la latencia en escenarios distribuidos.

Monitorización proactiva: Finalmente, la creación de un sistema de monitorización que permita detectar anomalías en las predicciones o en el rendimiento del sistema en tiempo real reforzaría aún más la robustez y fiabilidad de la solución.

En resumen, el proyecto ha sentado las bases para un sistema sólido, escalable y preciso, capaz de generar valor real en un entorno empresarial. A partir de esta base, las posibles líneas de mejora y ampliación aseguran que el trabajo pueda evolucionar continuamente, adaptándose a nuevas necesidades, retos y oportunidades de negocio.

9. Anexos

En este apartado se recopilan los elementos más relevantes que complementan y sustentan el desarrollo del proyecto. Estos anexos ofrecen una visión más técnica y detallada de la implementación práctica, proporcionando la trazabilidad completa de las configuraciones utilizadas y el soporte necesario para la reproducibilidad de los resultados.

9.1 Scripts relevantes

A lo largo del proyecto se han desarrollado y organizado múltiples scripts que conforman el núcleo del sistema de predicción de churn. Estos scripts están organizados en módulos y siguen principios de modularidad y reutilización para facilitar su mantenimiento y escalabilidad. A continuación, se resumen algunos de los más destacados:

dag_inferencia.py y dag_trian.py: Son los DAGs principales orquestados por Airflow. Se encargan, respectivamente, de coordinar el flujo de inferencia en batch y el flujo de entrenamiento de modelos, asegurando la correcta ejecución de las tareas y su trazabilidad.

cargar_datos_clientes.py y cargar_datos_servicios.py: Scripts responsables de la ingesta de datos desde archivos CSV o Excel, realizando las primeras validaciones y limpiezas básicas antes de integrarse en el pipeline.

transformacion_final.py: Se encarga de aplicar las transformaciones finales a los datos de entrada (escalares, codificaciones y selección de características) utilizando artefactos previamente registrados en MLflow. Garantiza que los datos tengan el formato y las características exactas con las que fue entrenado el modelo.

entrenar_modelo.py: Implementa la lógica de entrenamiento del modelo utilizando técnicas de optimización de hiperparámetros con Optuna. Permite explorar y seleccionar la configuración más adecuada para maximizar el rendimiento del modelo.

predecir_modelo.py y evaluar_modelo.py: Encargados de la inferencia y de la evaluación de las predicciones, respectivamente. Su integración con FastAPI y el servicio externo de evaluación aseguran un flujo completo y validado.

register.py: Gestiona el registro de artefactos y modelos en MLflow, facilitando la gestión de versiones y la trazabilidad del proceso.

consumer.py y api_to_kafka_split.py: Scripts clave para la rama de streaming, que se integran con Kafka para capturar datos en tiempo real y enviarlos al servicio FastAPI para la inferencia, demostrando la viabilidad del sistema en escenarios de producción.

Estos scripts han sido diseñados para funcionar de forma modular y desacoplada, lo que facilita la incorporación de mejoras o la adaptación a nuevas necesidades sin comprometer la estabilidad general del sistema.

9.2 Configuración adicional

El proyecto también incluye una serie de configuraciones específicas y artefactos adicionales que garantizan su correcto funcionamiento en entornos de contenedores y la coherencia entre todas las fases del pipeline:

docker-compose.yaml: Archivo principal de configuración que define los contenedores de Airflow, FastAPI, TimescaleDB, Kafka, MongoDB, MLflow y otros servicios clave. Permite levantar la arquitectura completa del proyecto de forma reproducible y coherente.

Artefactos de entrenamiento (carpeta artifacts/): Incluye los modelos serializados (pkl), escaladores (scaler.pkl), codificadores (label_encoders.pkl) y la lista de columnas relevantes (columnas_entrenamiento.txt y kbest_columnas.txt). Estos archivos garantizan que la inferencia se realice bajo las mismas condiciones y características con las que se entrenó el modelo.

requirements.txt y Dockerfile de cada contenedor**: Definen las dependencias necesarias y la construcción de los entornos para cada servicio (por ejemplo, FastAPI o el consumidor de Kafka), asegurando la homogeneidad y compatibilidad de los entornos de ejecución.

Configuración de MLflow: Establecida en cada script mediante mlflow.set_tracking_uri y variables de entorno para conectar con el servidor MLflow alojado en Docker. De esta manera, se garantiza la trazabilidad de todos los experimentos y modelos registrados.

Configuración de Airflow: Las variables de entorno definidas en el docker-compose.yaml aseguran que Airflow funcione con la base de datos PostgreSQL, y con las rutas y permisos necesarios para gestionar los DAGs y la ejecución de tareas.

Configuración de Kafka: Incluye los topics de entrada y salida, las credenciales y los ajustes de replicación y balanceo, fundamentales para garantizar la confiabilidad y disponibilidad de los datos en tiempo real.

En conjunto, estos scripts y configuraciones representan la columna vertebral de un sistema que no solo destaca por su precisión predictiva, sino también por su adaptabilidad y su enfoque modular y reproducible. Constituyen un punto de partida sólido para futuras mejoras y para su despliegue en entornos reales de producción.

10. Referencias

Durante el desarrollo de este proyecto intermodular se consultaron diversas fuentes académicas y técnicas que sirvieron de base para comprender en profundidad las técnicas utilizadas, evaluar las metodologías más adecuadas y respaldar las decisiones tomadas en cada fase. A continuación, se enumeran las principales referencias y enlaces útiles utilizados.

10.1. Bibliografía Científica

Mokhtar, H., Djouani, K., & Hamam, Y. (2020). Customer churn prediction in telecom using machine learning in big data platform. IEEE Access, 8, 42644-42656. DOI: 10.1109/ACCESS.2020.2977131

Wang, W., Li, M., & Wang, B. (2020). Enhancing customer retention in telecom industry with machine learning driven churn prediction. In Proceedings of the 2nd International Conference on Big Data and Internet of Things. DOI: 10.1145/3407023.3407040

Hossain, M. S., & Kabir, M. A. (2020). Social Network Analytics for Churn Prediction in Telco. In 2020 IEEE International Conference on Artificial Intelligence and Computer Engineering (ICAICE), 29-33. DOI: 10.1109/ICAICE51518.2020.00015

10.2. Fuentes Técnicas y Documentación

Apache Airflow. Documentation. https://airflow.apache.org/docs/

FastAPI. Documentation. https://fastapi.tiangolo.com/

Optuna. Documentation. https://optuna.org/

MLflow. Documentation. https://mlflow.org/docs/latest/index.html

Apache Kafka. Documentation. https://kafka.apache.org/documentation/

TimescaleDB. Documentation. https://docs.timescale.com/

MongoDB. Documentation. https://www.mongodb.com/docs/