INFORME MINERIA DE DATOS

Integrantes: Gamaliel Moya, Erika Aristizábal, Leonardo Miranda, Matías Baeza, Luis Tobar

22 DE NOVIEMBRE DE 2025

Contenido

1. El Problema

1.1 Contexto Real del Problema

Actualmente, los Centros de Salud Familiar (**CESFAM**) en Chile enfrentan una saturación crónica en su sistema de agendamiento de horas médicas y de procedimientos. La asignación de horas se realiza a menudo mediante métodos manuales, llamadas telefónicas o sistemas de "primero en llegar" (filas presenciales), lo que resulta en una distribución ineficiente de los cupos disponibles y una alta frustración en la población. Los pacientes deben esperar largos periodos o presentarse físicamente muy temprano para conseguir una hora, y existe una alta tasa de inasistencia (***no-show rate***) debido a la rigidez del sistema y la falta de recordatorios efectivos.

Justificación Técnica de la Relevancia

El problema es técnicamente relevante porque la asignación actual de horas no utiliza la data histórica para optimizar la distribución de los cupos. Un modelo de Machine Learning puede analizar patrones de demanda (por día, mes, especialidad, edad, sector, etc.) y de inasistencia. La solución propuesta se justifica por la capacidad de implementar un **modelo predictivo de Machine Learning** para predecir la demanda de horas y la probabilidad de *no-show* de un paciente, lo que permite la **sobreagendamiento inteligente** o la **reprogramación proactiva** de horas vacantes.

2. Impacto y Variables

2.1 Impacto en la Operación o Negocio (Salud Pública)

El impacto de resolver la ineficiencia en el agendamiento es significativo y se manifiesta en varios frentes:

- **Mejora de la Eficiencia Operacional:** Se logra maximizar la utilización de los cupos disponibles (reducir el tiempo ocioso del personal) y optimizar la dotación de recursos humanos y físicos.
- **Reducción de la Tasa de Inasistencia (*No-Show*):** Al predecir qué pacientes tienen alta probabilidad de faltar, se pueden implementar estrategias de recordatorio específicas o sobreagendar con un riesgo calculado.
- **Mejora en la Satisfacción del Usuario:** Se disminuyen las filas y los tiempos de espera para la obtención de una hora, mejorando la percepción de la calidad del servicio.
- **Gestión Basada en Evidencia:** El sistema pasa de una gestión reactiva a una **proactiva** y basada en datos históricos.

2.2 Variables Involucradas y Restricciones

Variables Involucradas (Features Potenciales para el Modelo):

Tipo de Variable	Ejemplos de Variables
Demográficas del Paciente	Edad, sexo, ubicación geográfica (sector o comuna), previsión de salud.
Histórico de Agendamiento	Número de inasistencias previas, tiempo promedio entre solicitud y atención, tipo de atención (primera vez o control).
De la Hora/Cita	Día de la semana, hora del día, especialidad médica solicitada, tipo de profesional de la salud.
Contextuales	Mes del año (picos de demanda), días festivos cercanos.

Restricciones del Proyecto:

- **Acceso a Datos:** Restricción fundamental, se requiere acceso a datos históricos de agendamiento y *no-shows* anonimizados, cumpliendo con la Ley de Protección de Datos Personales (LPD).
- **Infraestructura Existente:** El sistema propuesto debe ser capaz de integrarse con los sistemas de gestión de pacientes (GIS) que actualmente usa el CESFAM, los cuales pueden ser tecnológicamente limitados.
- **Restricciones Clínicas:** El modelo no debe interferir con la asignación de horas de urgencia o con protocolos clínicos específicos que exigen atención prioritaria a ciertos grupos de riesgo.
- **Imbalance de Clases:** La variable objetivo (*no-show*) puede estar desbalanceada (la mayoría de los pacientes sí asiste), lo que requerirá técnicas de remuestreo o métricas específicas.

3. La Solución Propuesta: Sistema Predictivo de Agendamiento

3.1 Arquitectura General del Sistema

Proponemos una arquitectura de microservicio orientada al despliegue, que encapsula el modelo predictivo y lo expone a través de una API.

- **Capa de Datos:** Almacenamiento de datos históricos anonimizados del CESFAM (e.g., base de datos SQL/NoSQL).
- **Pipeline de ETL/ML:** Script de preprocesamiento, *feature engineering* y entrenamiento, ejecutado periódicamente (o bajo demanda) para generar y serializar el modelo entrenado (.pkl o similar).
- **API REST (Microservicio):** Desarrollada con un framework ligero (como **FastAPI** o **Flask**) que carga el modelo entrenado en memoria al inicio. Expone el *endpoint* /predict.
- **Dashboard de Visualización:** Interfaz web (usando **Streamlit** o **Dash**) que consume los resultados del *endpoint* de la API para visualizar métricas, predicciones en tiempo real y resultados del EDA.
- **Consumidor Final (Aplicación de Agendamiento):** El sistema de agendamiento del CESFAM llama a la API para obtener la predicción de riesgo de *no-show* antes de confirmar una hora.

3.2 Algoritmo de Machine Learning Elegido y Justificación Técnica

Algoritmo Propuesto

Para la predicción de inasistencia (*no-show*), que es un problema de **clasificación binaria** (Asiste: 0 / No Asiste: 1), se elegirá **Gradient Boosting (e.g., XGBoost o LightGBM)**.

Justificación Técnica

1. **Alto Rendimiento en Datos Tabulares:** Los modelos de *Boosting* suelen superar a modelos lineales y a *Random Forests* en la mayoría de los desafíos de clasificación con datos tabulares.
2. **Manejo de Features No Lineales:** Es capaz de capturar relaciones complejas y no lineales entre las variables (edad, hora del día, historial de faltas) sin requerir una ingeniería de *features* manual tan extensa.
3. **Tolerancia al Desbalance:** A diferencia de otros clasificadores, *Gradient Boosting* puede ser ajustado para poner mayor peso en la clase minoritaria (los *no-shows*), crucial debido al desbalance esperado en los datos.

3.3 Diseño del Pipeline de Datos (ML Pipeline)

El *pipeline* de datos garantizará la **reproducibilidad** y la **coherencia** de los datos entre las fases de entrenamiento y predicción (API).

1. **Ingesta y Validación:** Carga de datos crudos y validación inicial de esquema (revisión de tipos de datos, rangos válidos).
2. **Limpieza y Manejo de Nulos:** Imputación de nulos (e.g., con la media/mediana para numéricos, o con la moda/valor fijo para categóricos)[1].
3. **Feature Engineering:**
    - Creación de variables de tiempo (Día de la semana, fin de semana sí/no, Mes del año).
    - Cálculo de variables de riesgo (Tasa histórica de *no-show* del paciente, o de la especialidad).
4. **Codificación (Encoding):** Aplicación de **One-Hot Encoding** a variables categóricas con baja cardinalidad y **Target Encoding** o **Ordinal Encoding** a variables con alta cardinalidad (para evitar la explosión de *features*).
5. **Escalado:** Estandarización o normalización de las variables numéricas para el modelo.
6. **Serialización:** El *pipeline* completo (incluyendo limpieza, ingeniería y codificación) se serializa (generalmente con **joblib**) junto con el modelo entrenado, asegurando que la API aplique **exactamente las mismas transformaciones** a los datos de entrada antes de predecir.

3.4 Decisiones Técnicas y Métricas

Aspecto	**Decisión Técnica**	**Justificación**
Métrica Principal	**Recall** de la clase "No Asiste" (Positivo)	Maximizar el *Recall* minimiza los **falsos negativos** (pacientes que faltarán, pero el modelo predice que sí asistirán). Esto es crucial para identificar oportunidades de sobreagendamiento o intervención.
Métrica Secundaria	**Curva ROC AUC**	Evalúa el rendimiento general del clasificador en umbrales de decisión variables[3].
Manejo de Nulos	**Imputación Estratégica**	Se justifica técnicamente según el tipo de variable (e.g., imputar un valor específico 'Desconocido' si el nulo tiene significado propio).
Encoding	**One-Hot y Target Encoding**	Para evitar la pérdida de información de las categóricas (One-Hot) y manejar eficientemente las variables con muchos valores únicos (Target Encoding).

3.5 Integración del Modelo con Dashboard y API

- **API del Modelo (FastAPI):** Recibirá un JSON con las *features* del paciente y la hora (ej. /predict?edad=45&especialidad=medicina_general). La API deserializará el *pipeline*, lo aplicará a los datos de entrada, obtendrá la probabilidad de *no-show* y devolverá la predicción (0 o 1) y la probabilidad asociada.
- **Dashboard:** Consumirá la API para:
    - Mostrar el resultado en una interfaz amigable.
    - Visualizar cómo el modelo se desempeña con datos en tiempo real.
    - Mostrar los **gráficos del modelo** (e.g., Matriz de Confusión, Curva ROC) y los **gráficos EDA** para análisis continuo.

4. Testing y Evidencias

Esta sección valida que el código no solo funcione, sino que sea robusto y que la aplicación cumpla con su propósito en un flujo real.

4.1 Tests Unitarios

Los *tests* unitarios son pruebas automatizadas que verifican el correcto funcionamiento de unidades aisladas de tu código.

Requisito	**Descripción Técnica**
Cobertura	Debes crear *tests* unitarios para las funciones críticas del proyecto[33]:
	* **Preprocesamiento:** Verificar que las funciones de limpieza, imputación de nulos y *encoding* actúen correctamente sobre los datos de prueba.
	* **Predicción:** Verificar que la función de predicción cargue el modelo y devuelva un resultado esperado (0 o 1) con los datos de entrada correctos.
	* **API:** Testear que los *endpoints* de la API (especialmente /predict) respondan con el código de estado correcto (e.g., 200 para éxito) y manejen correctamente la **validación de datos** y los **errores**[4444].
Ejecución	Todos los *tests* deben **ejecutarse sin errores**[5].

4.2 Evidencias de Pruebas Funcionales

Las pruebas funcionales demuestran el flujo completo de la aplicación, desde la interfaz hasta la predicción, con datos que simulan el uso real.

Requisito	**Descripción Técnica**
Ubicación	Deben incluirse en la carpeta específica: /docs/pruebas[6].
Contenido	Las evidencias deben ser capturas de pantalla o grabaciones que muestren el **flujo real** de la aplicación[7]:
	* **API:** Prueba de la API (e.g., usando Postman o curl) mostrando la solicitud (JSON de entrada) y la respuesta del *endpoint* /predict[8].
	* **Dashboard:** Interacción con el *dashboard* y cómo muestra una predicción real (por ejemplo, al ingresar los datos de un paciente)[9].
Validación	Deben incluir de forma explícita los **resultados esperados** y los **resultados obtenidos** para cada caso de prueba[10].

5. Estructura de Archivos Recomendada

Una estructura ordenada y modular es un requisito clave[11]. Aquí tienes un esquema de carpetas para organizar el proyecto de forma profesional:

```
MachineLearning---Proyecto-M/
├── README.md              # Documentación general
├── requirements.txt       # Dependencias del proyecto
├── .streamlit/
│   ├── # config.toml # Modifica Fondo, letras y Tipografía
├── data/
│   ├── # Dataset generado (dataset_cesfam_v1.csv)
├── docs/ # Evidencias de pruebas funcionales y documentacion en general
├── models/
│   └── model_pipeline.pkl  # Modelo entrenado serializado
├── src/
│   ├── api/
│   │   ├── main.py        # API FastAPI (Endpoint /predict)
│   │   └── model_loader.py # Cargador del modelo
│   ├── dashboard/
│   │   └── dashboard.py    # Interfaz Streamlit
│   ├── data_prep/
│   │   └── stream_generator.py # Simulación de flujo en tiempo real para el Dashboard
│   │   └── data_generator.py # Script de generación de datos
│   └── modeling/
│       ├── pipeline.py     # Lógica de preprocesamiento
│       └── train.py        # Script de entrenamiento
└── tests/                 # Tests unitarios (pytest)
```