Instituto Tecnológico y de Estudios Superiores de Monterrey

Inteligencia artificial avanzada para la ciencia de datos I

Grupo 101

22 de Octubre 2023

Titanic - Machine Learning from Disaster

Autores:

Catherine Johanna Rojas Mendoza A01798149

Adrian Pineda Sánchez A00834710

Luis Maximiliano López Ramírez A00833321

Rogelio Lizárraga Escobar A01742161

Rodolfo Jesús Cruz Rebollar A01368326

Profesor:

Edgar Gonzalez Fernandez

Índice

Abstract

El reto "Titanic: Machine Learning from Disaster"busca predecir la supervivencia de los pasajeros del Titanic utilizando técnicas de machine learning. Se desarrollaron modelos predictivos a partir de datos históricos de pasajeros, que incluyen edad, género y clase socioeconómica. El proceso incluyó limpieza y transformación de datos, ingeniería de características y la aplicación de varios algoritmos de clasificación. Los modelos fueron optimizados mediante herramientas como GridSearchCV y Optuna, evaluando su desempeño a través de métricas de precisión, recall y la curva ROC. La meta fue construir un modelo robusto para predecir la supervivencia con alta precisión.

Introducción

El desafío "Titanic: Machine Learning from Disaster"tiene como objetivo predecir la supervivencia de los pasajeros del Titanic, un transatlántico que se hundió en 1912, causando la muerte de más de 1500 personas. El reto consiste en identificar los factores que influyeron en la supervivencia y desarrollar un modelo predictivo utilizando un conjunto de datos que incluye información sobre la edad, género, clase socioeconómica y otros atributos de los pasajeros. Este problema es un caso clásico de clasificación binaria en machine learning, donde el desafío es separar a los pasajeros en dos grupos: sobrevivientes y no sobrevivientes. La clave está en descubrir patrones en los datos para mejorar la precisión del modelo, aplicando técnicas de limpieza de datos, ingeniería de características y selección de modelos, demostrando cómo los datos históricos pueden utilizarse para realizar predicciones precisas en contextos de gran relevancia.

Objetivos

1. Desarrollar un modelo predictivo que clasifique a los pasajeros del Titanic en dos categorías: sobrevivientes y no sobrevivientes, utilizando un conjunto de datos históricos.

2. Identificar y analizar los factores más relevantes que influyeron en la supervivencia de los pasajeros, como la edad, género, y clase socioeconómica, mediante técnicas de ingeniería de características.

3. Aplicar técnicas de limpieza de datos y procesamiento para preparar el conjunto de datos, asegurando su calidad y adecuación para el entrenamiento del modelo.

4. Evaluar la precisión del modelo mediante métricas como la exactitud y la curva ROC, garantizando que el modelo alcance un rendimiento adecuado para su propósito

5. Optimizar el modelo mejorando su capacidad de generalización a través de la selección de características y la validación cruzada, alcanzando un nivel de precisión previamente definido como satisfactorio.

Herramientas y recursos utilizados

- **Software y Frameworks:** Para el desarrollo del modelo predictivo en el reto "Titanic: Machine Learning from Disaster", se utilizó Google Colab como entorno de desarrollo. Google Colab es una plataforma que permite la ejecución de código en notebooks de Python, facilitando la colaboración y el uso de recursos computacionales avanzados. Dentro de este entorno, se emplearon las siguientes bibliotecas y frameworks:

    1. Pandas: Para la manipulación y análisis de datos estructurados.
    2. NumPy: Para cálculos numéricos eficientes y operaciones con arrays.
    3. Matplotlib y Seaborn: Para la visualización de datos y gráficos estadísticos.
    4. Scikit-learn: Para la implementación de algoritmos de machine learning, incluyendo la construcción, evaluación y optimización del modelo predictivo.

- **Hardware y Servicios:** El trabajo se realizó íntegramente en Google Colab, aprovechando los recursos de hardware proporcionados por la plataforma, que incluyen:

    1. La CPU (Unidad Central de Procesamiento), la GPU (Unidad de Procesamiento Gráfico) y la TPU (Unidad de Procesamiento Tensorial) son diferentes tipos de unidades de hardware diseñadas para realizar tareas de cómputo, cada una optimizada para ciertos tipos de trabajo. Estas opciones son ofrecidas dentro del entorno de Google Colab, sin embargo no fue necesario aplicar GPU o TPU, pues los modelos no requirieron el uso intensivo de estos recursos.
    2. Almacenamiento en Google Drive: Para guardar los notebooks y los datos generados durante el proceso de análisis y modelado.

- **Fuentes de Información:** El proyecto utilizó dos conjuntos de datos principales proporcionados por Kaggle, estructurados en formato CSV:

    1. Conjunto de Entrenamiento (train.csv): Contiene los datos de 891 pasajeros, incluyendo sus características demográficas y socioeconómicas, como edad, género, clase socioeconómica, y el resultado de supervivencia (0 = No sobrevivió, 1 = Sobrevivió). Este conjunto de datos es la base para la construcción y entrenamiento del modelo, permitiendo la identificación de patrones y relaciones entre las variables que influyen en la supervivencia.
    2. Conjunto de Prueba (test.csv): Incluye información similar a la del conjunto de entrenamiento, con datos de 418 pasajeros. Sin embargo, en este caso, no se proporciona la etiqueta de supervivencia. El objetivo es utilizar el modelo entrenado para predecir la probabilidad de supervivencia de estos pasajeros, lo que permite evaluar el rendimiento del modelo en datos no vistos previamente.

- **Calidad:** Los datos han sido limpiados y procesados con el fin de estructurarlos específicamente para este reto, garantizando su relevancia y adecuación para el análisis predictivo.
- **Cantidad:** El dataset de entrenamiento contiene información de 891 pasajeros, mientras que el conjunto de prueba incluye datos de 418 pasajeros adicionales, sin la etiqueta de supervivencia.
- **Formato y Estructura**: Los datos están en formato tabular, con columnas que representan las diferentes variables. Los datos fueron manipulados y transformados para adecuarlos al proceso de modelado.

Etapas

Para abordar el reto "Titanic: Machine Learning from Disaster", se realizaron dos Google Colabs con enfoques distintos en la limpieza y transformación de los datos. La razón principal para crear dos notebooks separados fue explorar diferentes estrategias de preprocesamiento de datos, con el fin de evaluar cómo afectan estos enfoques al rendimiento del modelo predictivo, pues la calidad del preprocesamiento es crucial para garantizar que el modelo pueda capturar patrones significativos en los datos y generalizar bien en datos no vistos previamente.

Extracción

- **Primer Google Colab:** En esta etapa, se cargaron los conjuntos de datos train.csv y test.csv proporcionados por Kaggle utilizando la biblioteca Pandas. Ambas bases de datos fueron almacenadas previamente en un repositorio público de GitHub. Se realizó una inspección inicial de las primeras filas de los datos para comprender su estructura y verificar la existencia de valores nulos e inconsistencias.
- **Segundo Google Colab:** Similar al primer enfoque, los datos se cargaron utilizando Pandas.

Limpieza

Para ambos colabs, se renombraron las columnas del conjunto de datos a español, para facilitar la interpretación y manipulación de los datos durante el análisis.

Durante esta fase de limpieza de datos, se tomaron decisiones sobre qué variables eliminar para simplificar el modelo y mejorar su rendimiento, basadas en la relevancia y la cantidad de valores faltantes.

Aspecto	Primer Google Colab	Segundo Google Colab
Variables	ID, Clase Ticket, Nombre, Sexo, Edad, Hermanos_Esposos, Padres_Hijos, Tarifa, Sobrevivió	ID, Sobrevivió, Clase Ticket, Nombre, Sexo, Edad, Hermanos_Esposos, Padres_Hijos, Tarifa, Embarcación
Cabin	Eliminada por la alta cantidad de valores faltantes y baja relevancia en la predicción de supervivencia.	Eliminada por las mismas razones: alta cantidad de valores faltantes y baja relevancia.
Ticket	Eliminada porque no se identificó un patrón útil en los números de boleto y por su naturaleza categórica que requiere una ingeniería de características compleja.	Eliminada por las mismas razones: la falta de patrones útiles para la predicción de supervivencia.
Embarcación	Eliminada porque no se consideró relevante para la predicción de la supervivencia.	Se prueba una imputación de datos para esta variable con el fin de observar si aporta información adicional significativa para los modelos.

Cuadro 1

Comparación de las decisiones de eliminación de variables en los Google Colabs.

Transformación

- **Primer Google Colab**

  Codificación de Variables Categóricas: Para la columna `Sexo`, se aplicó One Hot Encoding. Este paso permitió transformar la variable categórica en valores numéricos binarios (0 y 1), donde: 'male' fue mapeado a 0. 'female' fue mapeado a 1. Esta conversión es esencial para permitir que los modelos de machine learning puedan interpretar esta variable correctamente.

  Creación de Nuevas Variables:

  1. **Clasificación de Edad:** Se creó una columna que clasifica a los pasajeros en grupos de edad como 'Bebé', 'Niño', 'Adulto', etc. Esta clasificación proporciona información adicional sobre la demografía de los pasajeros, lo que puede ser útil en los modelos predictivos.
  2. **Familiares:** Esta columna suma el número de `Hermanos/Esposos` y `Padres/Hijos` que el pasajero tenía a bordo, proporcionando una medida del número total de familiares presentes en el Titanic.

- **Segundo Google Colab**

  Codificación de Variables Categóricas: Se aplicó la técnica de One Hot Encoding, que convierte las categorías en múltiples columnas binarias. Las columnas resultantes fueron:

  1. Clase Ticket: Se crearon las columnas `Primera Clase`, `Segunda Clase` y `Tercera Clase`, representando las tres categorías de clase en el barco.
  2. Embarcación: Se generaron las columnas `S`, `Q` y `C` para indicar el puerto de embarque de los pasajeros.

3. Familiares: Se clasificaron los pasajeros en función del tamaño de su familia a bordo, creando las columnas `FamPequeña`, `FamMediana` y `FamGrande`.
4. Título: A partir de los títulos en los nombres de los pasajeros, se crearon las columnas `Mr`, `Miss`, `Mrs`, `Master` y `Otro` para capturar estas categorías en el análisis.
5. Sexo: Se crearon las columnas `Femenino` y `Masculino`, transformando la variable categórica de género en valores binarios.

Creación de Nuevas Variables: Para mejorar la capacidad predictiva del modelo y proporcionar un análisis más detallado, se generaron nuevas variables a partir de las columnas originales

1. Clasificación de Edad: Se creó esta columna, que agrupa a los pasajeros en diferentes rangos etarios, como 'Bebé', 'Niño', 'Adulto', y 'Adulto Mayor', entre otros, basándose en su edad. Esto permite analizar cómo la edad afecta la supervivencia de manera más precisa.
2. Familiares: Suma las columnas `Hermanos_Esposos` y `Padres_Hijos` para representar el número total de familiares que un pasajero tenía a bordo. Esta variable proporciona una visión más completa de la estructura familiar de los pasajeros.
3. Solo Viaje: A partir de la variable `Familiares`, se generó la columna `Solo_Viaje`, una variable binaria que indica si un pasajero viajaba solo (valor 1) o acompañado (valor 0). Esta variable es útil para determinar si viajar solo influía en la probabilidad de supervivencia.

Escalamiento de los Datos: Para garantizar que los datos numéricos estuvieran en un rango uniforme, se aplicó la técnica de escalamiento `MinMaxScaler` a columnas como `Edad`, `Hermanos_Esposos`, `Padres_Hijos` y `Tarifa`. Este método normaliza los valores, llevándolos a un rango de 0 a 1, lo que mejora el rendimiento de los algoritmos de Machine Learning que son sensibles a la escala de los datos.

Análisis de Componentes Principales (PCA): Dado que el conjunto de datos final contenía muchas columnas, se utilizó el Análisis de Componentes Principales (PCA) para reducir la dimensionalidad, conservando la mayor parte de la varianza de los datos. Tras aplicar PCA, se seleccionaron 6 componentes principales que explicaban aproximadamente el 90 % de la variabilidad total del dataset. Estas componentes simplificaron el modelo al reemplazar el conjunto original de variables con un número reducido de dimensiones, optimizando así la eficiencia sin perder información clave.

Imputación de datos

La imputación de datos es un paso crucial en la limpieza y preparación de los datos, especialmente cuando se trabaja con conjuntos de datos incompletos. En este primer Google Colab, se implementaron

Aspecto	Primer Google Colab	Segundo Google Colab
Codificación de Variables Categóricas	One Hot Encoding en columna Sexo (male: 0, female: 1)	One Hot Encoding en Clase Ticket, Embarcación, Familiares, Título, y Sexo
Creación de Nuevas Variables	Clasificación de Edad (Bebé, Niño, Adulto) y Familiares (suma de Hermanos/Esposos y Padres/Hijos)	Clasificación de Edad (Bebé, Niño, Adulto, Adulto Mayor), Familiares (suma), Solo Viaje (binaria)
Escalamiento de los Datos	MinMaxScaler en Edad, Familiares y Clase ticket	MinMaxScaler en Edad, Hermanos_Esposos, Padres_Hijos y Tarifa
Análisis de Componentes Principales (PCA)	-	PCA para reducir a 6 componentes principales, explicando el 90 % de la variabilidad

Cuadro 2

Comparación de las actividades en el primer y segundo Google Colab

diversas estrategias para abordar los valores faltantes en dos columnas clave: `Edad` y `Tarifa`.

- **Primer Google Colab**

1. **Imputación de la Columna Edad** La columna `Edad` contenía un número significativo de valores nulos y dado que la edad es una variable importante para predecir la supervivencia de los pasajeros, se imputaron estos valores faltantes utilizando una técnica basada en los títulos extraídos de la columna `Nombre`.

   Extracción de Títulos: Se extrajeron los títulos (e.g., Mr, Mrs, Miss, Master) de los nombres de los pasajeros, ya que estos títulos proporcionan una indicación aproximada de la edad del pasajero. Por ejemplo, Master generalmente se refiere a niños, mientras que Mr y Mrs corresponden a adultos.

   Cálculo de Medias y Desviaciones Estándar: Para cada título, se calcularon la media y la desviación estándar de la edad entre los pasajeros que compartían ese título. Esto permitió capturar la variabilidad natural de las edades dentro de cada grupo.

   Imputación Probabilística: Utilizando las medias y desviaciones estándar calculadas, se imputaron los valores nulos de `Edad` mediante una distribución normal específica para cada título. Este enfoque probabilístico aseguraba que los valores imputados reflejaran la distribución observada en los datos originales, reduciendo el riesgo de introducir sesgos.

   Manejo de Casos Especiales: En un caso específico, un pasajero con el título Ms tenía un valor nulo en Edad. Dado que había pocos registros con este título, se utilizó una imputación basada en un promedio ponderado de los títulos Mrs y Miss, considerando que Ms es una variante de estos títulos. Esto garantizó una imputación razonable y coherente con la tendencia observada en los datos.

2. **Imputación de la Columna Tarifa**

La columna `Tarifa` también presentaba valores nulos en algunos registros del conjunto de datos de prueba. Dado que `Tarifa` es una variable continua que podría influir en la predicción de la supervivencia, se llevó a cabo una imputación cuidadosa:

Análisis por Clase Socioeconómica: Se observó que el valor nulo pertenecía a un pasajero de la clase socioeconómica 3 (Clase Ticket = 3). Por lo tanto, se calculó la media y desviación estándar de las tarifas dentro de esta clase.

Imputación Basada en Distribución: Utilizando la media y la desviación estándar de las tarifas en la clase 3, se generó un valor imputado dentro de un rango razonable utilizando una distribución normal. Esta estrategia aseguró que el valor imputado estuviera alineado con la distribución real de las tarifas en esa clase, minimizando la introducción de sesgos.

- **Segundo Google Colab**

1. **Imputación de la Columna Edad**

   Similar al primer colab, la imputación de la columna `Edad` se realiza mediante un enfoque detallado que utiliza los títulos de los pasajeros extraídos de la columna `Nombre`. Sin embargo se realizó una imputación diferente.

   Extracción de Títulos: Se extraen los títulos de los nombres de los pasajeros utilizando la expresión regular `([A-Za-z]+)`. Estos títulos incluyen Mr, Mrs, Miss, Master, y otros menos comunes como Lady, Countess, Capt, etc. Los títulos raros se agrupan bajo una categoría común llamada Otro, y algunos títulos específicos se normalizan (Mlle se convierte en Miss, Ms se convierte en Miss, y Mme se convierte en Mrs).

   Agrupación de Datos y Cálculo de Estadísticas: Después de normalizar los títulos, se agrupan los pasajeros en categorías basadas en estos títulos (Mr, Mrs,Miss, Master, Otro). Para cada grupo de títulos, se calculan la media (mean) y la desviación estándar (std) de la edad. Esto permite capturar la distribución de las edades dentro de cada grupo, que se usará para la imputación.

   Imputación Probabilística: Se define una función llamada `assign_age` que se aplica a cada fila del conjunto de datos. Esta función verifica si el valor de `Edad` es nulo (`pd.isnull(row[Edad])`).

   - Si la `Edad` es nula, la función determina el título del pasajero (`row[Titulo]`) y utiliza la media y la desviación estándar calculadas previamente para generar un valor de edad imputado.
   - El valor de edad imputado se genera utilizando una distribución normal (`np.random.normal(mean_age, std_age)`) con la media y desviación estándar correspondientes al título del pasajero. Esta distribución normal permite que el valor

imputado esté dentro de un rango realista, reflejando la variabilidad observada en el grupo.

Aplicación de la Imputación: La función `assign_age` se aplica a los conjuntos de datos de entrenamiento (`df_train`) y de prueba (`df_test`), imputando los valores faltantes en la columna `Edad`. Tras la imputación, se verifica la cantidad de valores nulos restantes en ambos conjuntos de datos. Inicialmente, había 177 valores nulos en `df_train` y 86 en `df_test`, pero después de la imputación, estos se reducen significativamente.

Manejo de Casos Especiales: Aunque el enfoque general se basa en el título, en casos donde un título tiene muy pocos registros (e.g., Ms), se utiliza un enfoque ponderado para imputar la edad. En este caso, se combinan las medias y desviaciones estándar de títulos similares (Mrs y Miss) para generar un valor de imputación más robusto. Esto asegura que incluso en categorías con pocos registros, los valores imputados sean consistentes y reflejen adecuadamente la distribución esperada de las edades.

2. **Imputación de la Columna Tarifa**

   La imputación de la columna `Tarifa` en el segundo Colab sigue un enfoque muy similar al del primer Colab, utilizando la clase socioeconómica del pasajero para guiar la imputación. La principal diferencia radica en la utilización de un generador de números aleatorios con semilla para asegurar la reproducibilidad: `(np.random.RandomState(22))`, lo que asegura que los valores imputados sean consistentes cada vez que se ejecute el código.

3. **Imputación de la Columna Embarcación**

   Dado que la columna `Embarcación` es categórica (con posibles valores como Ç", "Q", "S", que representan diferentes puertos), se optó por imputar los valores nulos usando la moda. La moda es el valor más frecuente, y se considera una buena práctica para imputar datos categóricos cuando faltan pocos valores, ya que se asume que la mayoría de los pasajeros embarcaron desde el mismo puerto.

Aspecto	Primer Google Colab	Segundo Google Colab
Imputación de Edad	Títulos extraídos (Mr, Mrs, Miss, Master). Imputación probabilística basada en media y desviación estándar. Manejo especial para título Ms.	Títulos extraídos y normalizados (Mlle, Mme, Ms). Función `assign_age` con imputación probabilística. Manejo especial para títulos raros.
Extracción de Títulos	Extraídos directamente de la columna Nombre.	Expresión regular utilizada para extraer títulos (`([A-Za-z]+)`).
Manejo de Casos Especiales	Título Ms imputado con un promedio ponderado de Mrs y Miss.	Títulos raros (e.g., Ms) combinados con medias de títulos similares para imputar edad.
Imputación de Tarifa	Imputación basada en la clase socioeconómica. Media y desviación estándar por clase.	Similar al primer colab, pero con generador de números aleatorios (`RandomState(22)`).
Imputación de Embarcación	No se imputa la columna Embarcación.	Imputación de valores nulos en la columna Embarcación utilizando la moda.

Cuadro 3

Comparación de las estrategias de imputación entre los dos Google Colabs

Generación de modelos

- **Primer Google Colab**

  Variables Predictoras: Las variables predictoras usadas en todos los siguientes modelos del Primer Google Colab fueron `Clase Ticket`, `Sexo`, `Edad` y `Familiares`.

  Primeros Modelos: Los primeros modelos que se probaron fueron de la librería `scikit-learn` estableciendo hiperparámetros arbitrarios. Los modelos utilizados fueron:

  1. **Regresión Logística:** Este modelo es adecuado para problemas de clasificación binaria y es especialmente útil cuando se espera una relación lineal entre las características (como Edad, Sexo, Clase Ticket, Familiares) y la probabilidad de supervivencia. Además, la regresión logística es interpretable, lo que permite entender el impacto de cada característica en las predicciones. (IBM, n.d.-a)
  2. **Árboles de Decisión:** Los árboles de decisión son útiles para capturar relaciones no lineales y no requieren escalar los datos o manejar las variables categóricas mediante encoding. Son adecuados para este conjunto de datos mixto (numérico y categórico). (IBM, 2021)
  3. **Random Forest:** Como un conjunto de árboles de decisión, el Random Forest mejora la estabilidad y generalización del modelo. Es especialmente útil para manejar grandes conjuntos de datos con alta dimensionalidad y ruido. (IBM, n.d.-d)
  4. **K-Nearest Neighbors (KNN):** Es un modelo simple que clasifica basándose en la similitud entre los datos. Es adecuado cuando las relaciones entre las variables no son

lineales y cuando el conjunto de datos no es excesivamente grande. (IBM, n.d.-b)

5. **Naive Bayes:** Este modelo es particularmente útil para problemas de clasificación donde la independencia de las características se asume o se aproxima. Funciona bien con conjuntos de datos pequeños y puede ser eficiente incluso con muchas variables predictoras. Es ideal cuando se desea una clasificación rápida y la interpretabilidad no es la prioridad principal. (IBM, n.d.-c)
6. **Red Neuronal Feedforward (FNN):** Se implementó una red neuronal con capas densas (fully connected) para predecir la supervivencia, pues las redes neuronales son potentes para capturar patrones complejos en los datos, aunque requieren más datos y poder de cómputo. Son útiles cuando se espera que las relaciones entre las variables sean no lineales y complicadas. (Education, 2023)
7. **LSTM (Long Short-Term Memory):** Se probó una arquitectura LSTM para ver si podría capturar patrones que otros modelos no logran, y aunque las LSTM son generalmente usadas para secuencias temporales, su capacidad de recordar y aprender patrones complejos en los datos las hace útiles en este caso. (GeeksforGeeks, 2024)
8. **XGBoost con Optuna:** Se optimizó el modelo XGBoost usando Optuna, probando varias configuraciones de hiperparámetros para encontrar el mejor rendimiento, pues esta es una potente técnica de boosting que maneja bien datos numéricos y categóricos. Es especialmente útil en competencias como Kaggle debido a su capacidad para maximizar la precisión a través de una combinación de árboles de decisión. Optuna se utilizó para optimizar hiperparámetros de manera automática. (Para, 2021)

A continuación se muestran la generación de estos modelos con distintas configuraciones de hiperparámetros.

1. **Regresión Logística:** Se probaron 5 configuraciones distintas en los hiperparámetros modificando los valores de `C`, `solver`, `max iter` y `penalty`. La configuración de hiperparámetros con los mejores resultado fue `C = 0.01`, `solver = liblinear` y `max iter = 1000`. La evaluación se encuentran en la tabla 4.
2. **Árboles de decisión:** Se probaron 3 configuraciones distintas en los hiperparámetros modificando los valores de `max depth`, `min samples split` y `criterion`. La configuración de hiperparámetros con los mejores resultado fue `max depth = 100`, `min samples split = 10` y `criterion = entropy`. La evaluación se encuentran en la tabla 4.
3. **Bosques Aleatorios:** Se probaron 5 configuraciones distintas en los hiperparámetros modificando los valores de `n estimators`, `max features`, `max depth`, `min samples split` y `min samples leaf`. La configuración de hiperparámetros con los mejores resultado fue `n`

`estimators = 300`, `max features = log2`, `max depth = 300` y `min samples leaf = 2`. La evaluación se encuentran en la tabla 4.

4. **K Vecinos más Cercanos (KNN):** Se probaron 3 configuraciones distintas en los hiperparámetros modificando los valores de `n neighbors` y `weight`. La configuración de hiperparámetros con los mejores resultado fue `n neighbors = 15` y `weight = uniform`. La evaluación se encuentran en la tabla 4.
5. **Naive Bayes:** Se probaron 3 configuraciones distintas en los hiperparámetros modificando los valores de `var smoothing`. La configuración de hiperparámetros con los mejores resultado fue `var smoothing = 1e-9`. La evaluación se encuentran en la tabla 4.

GridSearchCV: GridSearchCV busca los mejores parámetros para un modelo probando diferentes combinaciones de forma automática. Divide los datos de entrenamiento en varios `conjuntos de validación`, entrena y evalúa el modelo en cada uno para medir su rendimiento. Al final, selecciona la mejor combinación de parámetros y evalúa el modelo final en el `conjunto de prueba` para comprobar su desempeño real. Aplicando esta herramienta la evaluación se encuentran en la tabla 5. (Tuning the hyper-parameters of an estimator, s.f.)

Optuna: Optuna es una herramienta que optimiza los hiperparámetros de un modelo de forma inteligente y rápida. A diferencia de métodos tradicionales, utiliza algoritmos avanzados para explorar el espacio de parámetros de manera eficiente, probando solo las combinaciones más prometedoras. Esto acelera la búsqueda de la mejor configuración y mejora el rendimiento del modelo sin necesidad de probar todas las combinaciones posibles. En este caso se realizaron estudios para maximizar el accuracy del modelo y se hicieron 5,000 pruebas obteniendo que el mejor modelo es el que se encuentran en la tabla 6. (Torres, 2024)

Regresión Logística sin framework: Este código implementa un algoritmo de regresión logística desde cero, sin usar frameworks, como Scikit-learn. Para ello, se extraen los conjuntos de entrenamiento de X y Y, se definen valores iniciales para los coeficientes de cada $theta_i$ (2, -2, 2, -2, -2) con un learning rate de 0.05. Se ejecutan 15,000 iteraciones para ajustar los coeficientes, donde se utiliza una función sigmoide y se actualizan los coeficientes para minimizar el error.

Regresión Logística sin framework con funciones: Este código implementa un algoritmo de regresión logística desde cero, por medio de funciones y el uso de álgebra lineal. Se realiza lo mismo que la regresión logística sin framework original, pero se inicializan los coeficientes en ceros con un learning rate de 0.05 y 15000 iteraciones, donde los cálculos de los coeficientes se realizan mediante un producto punto entre estos y los datos X ($X \cdot theta_i \forall i$), se le aplica la función sigmoide y se calcula el gradiente por medio de un producto punto entre la transpuesta de X y la

h - y, lo cual se divide entre el total de datos:

$$gradient = \frac{X^T \cdot (h - y)}{n}$$

De tal manera, se actualizan los pesos restándole a θ_i, el valor total de $\theta_i \cdot \alpha$

Modelo FNN: Se realizó una red sequencial con una capa densa de 64 unidades con activación ReLU, una segunda capa densa de 32 unidades con ReLU y una última capa de 1 unidad que regresa con la activación sigmoide la predicción binaria (si sobrevivió o no). Lo anterior se corre en 1000 épocas con un batch size de 32 y un validation split del 10 % y con un earlystopping si no mejora en 50 épocas. Lo anterior se observa en la tabla 7.

Modelo LSTM: Se realizó una red sequencial con una capa LSTM de 64 unidades donde regresan las secuencias, una segunda capa LSTM de 32 unidades sin que las secuencias regresen, una tercera capa de 16 unidades densa con activación ReLU y una última capa de 1 unidad que regresa con la activación sigmoide la predicción binaria (si sobrevivió o no). Lo anterior se corre en 1000 épocas con un batch size de 32 y un validation split del 10 % y con un earlystopping si no mejora en 50 épocas. Lo anterior se observa en la tabla 7.

XGBOOST con Optuna: Se implementó un modelo XGBoost el cual tuvo optimización de hiperparámetros con optuna. Los parámetros optimizados para los últimos modelos se observan en la tabla 8.

- **Segundo Google Colab**

Variables Predictoras: Las variables predictoras usadas en todos los siguientes modelos del Primer Google Colab fueron `PC1`, `PC2`, `PC3`, `PC4`, `PC5` y `PC6` obtenidas a partir del Análisis de Componentes Principales.

Primeros Modelos: Los primeros modelos que se probaron fueron de la librería `scikit-learn` estableciendo hiperparámetros arbitrarios. Los modelos utilizados fueron:

1. **Regresión Logística:** Se probaron 5 configuraciones distintas en los hiperparámetros modificando los valores de `C`, `solver`, `max iter` y `penalty`. La configuración de hiperparámetros con los mejores resultado fue `C = 0.10`, `solver = newton-cg` y `max iter = 3000`. La evaluación se encuentran en la tabla 10.
2. **Árboles de decisión:** Se probaron 3 configuraciones distintas en los hiperparámetros modificando los valores de `max depth`, `min samples split` y `criterion`. La configuración de hiperparámetros con los mejores resultado fue `max depth = 150`, `min samples split = 5` y `criterion = gini`. La evaluación se encuentran en la tabla 10.

3. **Bosques Aleatorios:** Se probaron 5 configuraciones distintas en los hiperparámetros modificando los valores de `n estimators`, `max features`, `max depth`, `min samples split` y `min samples leaf`. La configuración de hiperparámetros con los mejores resultado fue `n estimators = 300`, `max features = log2`, `max depth = 300` y `min samples leaf = 2`. La evaluación se encuentran en la tabla 10.
4. **K Vecinos más Cercanos (KNN):** Se probaron 3 configuraciones distintas en los hiperparámetros modificando los valores de `n neighbors` y `weight`. La configuración de hiperparámetros con los mejores resultado fue `n neighbors = 15` y `weight = uniform`. La evaluación se encuentran en la tabla 10.
5. **Naive Bayes:** Se probaron 3 configuraciones distintas en los hiperparámetros modificando los valores de `var smoothing`. La configuración de hiperparámetros con los mejores resultado fue `var smoothing = 1e-9`. La evaluación se encuentran en la tabla 10.

GridSearchCV y Optuna: De nueva cuenta se utilizaron las herramientas de GridSearchCV y Optuna para buscar la mejor configuración de hiperparámetros que maximice la precisión del modelo. Las evaluaciones se encuentras en las siguientes tablas 11 12.

Evaluación

- **Primer Google Colab**

Cuadro 4

Modelos con hiperparámetros arbitrarios

Modelo		**Accuracy**	**Recall**	**Precision**	**F1-Score**	**ROC AUC**
Regresión Logística	**Entrenamiento**	0.787	0.947	0.471	0.629	0.849
	Prueba	0.775	—-	—-	—-	—-
Árboles de Decisión	**Entrenamiento**	0.89	0.924	0.778	0.844	0.958
	Prueba	0.763	—-	—-	—-	—-
Bosques Aleatorios	**Entrenamiento**	0.883	0.894	0.789	0.839	0.959
	Prueba	0.751	—-	—-	—-	—-
K Vecinos más Cercanos (KNN)	**Entrenamiento**	0.833	0.821	0.722	0.768	0.902
	Prueba	0.766	—-	—-	—-	—-
Naive Bayes	**Entrenamiento**	0.799	0.764	0.69	0.725	0.85
	Prueba	0.782	—-	—-	—-	—-

Cuadro 5

Resultados del GridSearchCV

Modelo	Mejor Accuracy en Validación	Mejores Hiperparámetros	Accuracy en Prueba
Logistic Regression	0.7935	{'C': 0.001, 'l1_ratio': 0.0, 'max_iter': 100, 'penalty': 'none', 'solver': 'newton-cg'}	0.7632
Decision Tree	0.8305	{'class_weight': None, 'criterion': 'gini', 'max_depth': 10, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'random'}	0.7656
Random Forest	0.8216	{'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 10}	0.7488
Naive Bayes	0.7857	{'var_smoothing': 1e-09}	0.7823
K-Nearest Neighbors (KNN)	0.8238	{'algorithm': 'brute', 'leaf_size': 10, 'metric': 'manhattan', 'n_neighbors': 3, 'p': 1, 'weights': 'uniform'}	0.7392

Cuadro 6

Mejor modelo con Optuna

Parámetro	Valor
Modelo	Árboles de Decisión
`criterion`	entropy
`max depth`	6
`min samples split`	17
`min samples leaf`	3
`max leaf nodes`	27
`Accuracy en Prueba`	78.94 %

Cuadro 7

Redes Neuronales realizadas

Modelo	Conjunto	Accuracy	Recall	Precision	F1-Score
FNN	**Entrenamiento**	0.83	0.657	0.868	0.748
	Prueba	0.772	—-	—-	—-
LSTM	**Entrenamiento**	0.8002	0.605	0.828	0.6993
	Prueba	0.7583	—-	—-	—-

Cuadro 8

Resultados de Optuna 2

Modelo	**Mejores Hiperparámetros**	**Accuracy en Prueba**
LSTM	{'hidden_size': 103, 'num_layers': 2, 'dropout': 0.04086814455069222, 'learning_rate': 0.004559194178514386}	0.7655
FNN	{'class_weight': None, 'criterion': 'gini', 'max_depth': 10, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'random'}	0.7799
XGBoost	{'alpha': 0.9980300864205861, 'base_score': None, 'booster': 'dart', 'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': 0.8511428216373328, 'device': None, 'early_stopping_rounds': None, 'enable_categorical': False, 'eta': 0.9999737834170497, 'eval_metric': None, 'feature_types': None, 'gamma': None, 'grow_policy': None, 'importance_type': None, 'interaction_constraints': None, 'lambda': 0.0056152907294103015, 'learning_rate': None, 'max_bin': None, 'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None, 'max_depth': 2, 'max_leaves': None, 'min_child_weight': 5.134034658985146e-07, 'missing': nan, 'monotone_constraints': None, 'multi_strategy': None, 'n_estimators': None}	0.7587

Cuadro 9

Mejor modelo con Optuna

Parámetro	**Valor**
Modelo	FNN
`criterion`	gini
`max depth`	10
`max features`	None
`min samples leaf`	1
`min samples split`	5
`splitter`	random
`Accuracy en Prueba`	77.99 %

- **Segundo Google Colab**

Cuadro 10

Modelos con hiperparámetros arbitrarios

Modelo		**Accuracy**	**Recall**	**Precision**	**F1-Score**	**ROC AUC**
Regresión Logística	**Entrenamiento**	0.788	0.737	0.696	0.716	0.851
	Prueba	0.775	—-	—-	—-	—-
Árboles de Decisión	**Entrenamiento**	0.963	0.975	0.927	0.951	0.997
	Prueba	0.718	—-	—-	—-	—-
Bosques Aleatorios	**Entrenamiento**	0.929	0.949	0.863	0.904	0.989
	Prueba	0.734	—-	—-	—-	—-
K Vecinos más Cercanos (KNN)	**Entrenamiento**	0.825	0.839	0.673	0.747	0.896
	Prueba	0.746	—-	—-	—-	—-
Naive Bayes	**Entrenamiento**	0.8	0.741	0.737	0.739	0.842
	Prueba	0.739	—-	—-	—-	—-

Cuadro 11

Resultados del GridSearchCV

Modelo	Mejor Accuracy en Validación	Mejores Hiperparámetros	Accuracy en Prueba
Logistic Regression	0.8024	{'C': 0.01, 'l1_ratio': 0.0, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}	0.7512
Decision Tree	0.8227	{'class_weight': None, 'criterion': 'entropy', 'max_depth': 50, 'max_features': None, 'min_samples_leaf': 8, 'min_samples_split': 2, 'splitter': 'random'}	0.7440
Random Forest	0.8227	{'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 50}	0.7464
Naive Bayes	0.7968	{'var_smoothing': 1e-09}	0.7392
K-Nearest Neighbors (KNN)	0.8238	{'algorithm': 'auto', 'leaf_size': 10, 'metric': 'chebyshev', 'n_neighbors': 19, 'p': 1, 'weights': 'distance'}	0.7368

Cuadro 12

Mejor modelo con Optuna

Parámetro	Valor
Modelo	K Vecinos más Cercanos (KNN)
`n neighbors`	20
`weights`	uniform
`p`	1
`algorithm`	ball tree
`leaf size`	33
`Accuracy en Prueba`	77.51 %

Resultados

Se encontró que el mejor modelo realizado fue el de Árboles de decisión con un accuracy en prueba de 78.94 %. Los parámetros utilizados fueron los siguientes: se utilizó un criterio de entropía, una profundidad máxima de 6, un split mínimo de muestras de 17, un mínimo de muestras de nodos externos de 3 y un máximo de muestras de nodos externos de 27.

Conclusiones

Este reto nos permitió desarrollar y evaluar diferentes modelos de machine learning, con el objetivo de predecir la supervivencia de los pasajeros del Titanic. El análisis mostró que el modelo de Árboles de Decisión optimizado mediante Optuna logró el mejor rendimiento, con un 78.94 % de precisión en el

conjunto de prueba, lo que indica un modelo robusto para este problema de clasificación binaria.

Nuestras principales conclusiones son:

- **Importancia de la Ingeniería de Características:** Las variables como la clase socioeconómica, el género y la edad fueron determinantes en la predicción de la supervivencia, lo que resalta la relevancia de la correcta selección y transformación de las características.
- **Optimización de Modelos:** La utilización de herramientas como GridSearchCV y Optuna resultó fundamental para identificar los mejores hiperparámetros en cada modelo, demostrando que la optimización es clave para mejorar la precisión y generalización de los modelos.
- **Desempeño de los Modelos:** Aunque varios modelos lograron un desempeño aceptable, el Árbol de Decisión mostró el mejor equilibrio entre simplicidad y precisión, con la posibilidad de interpretarlo fácilmente, algo valioso en este tipo de problemas.

Trabajo a Futuro

Existen varias áreas de mejora para optimizar el desempeño del modelo. En primer lugar, refinar la ingeniería de características explorando nuevas transformaciones y combinaciones de variables podría capturar relaciones más complejas en los datos, mejorando la precisión. Además, la exploración de modelos avanzados como redes neuronales profundas más sofisticadas permitiría identificar patrones no lineales con mayor eficacia.

Ampliar el conjunto de datos a través de técnicas de generación de datos sintéticos o incorporar información adicional también ayudaría a mejorar la capacidad de generalización del modelo. En cuanto a la imputación de datos faltantes, utilizar enfoques más avanzados para estimar los valores nulos podría reducir el impacto de los datos incompletos en el rendimiento del modelo.

Finalmente, sería útil implementar una validación cruzada más exhaustiva y usar diversas métricas, como AUC-ROC y sensibilidad, para evaluar los modelos de manera más robusta, con el fin de entender mejor las decisiones del modelo y optimizar aún más su desempeño.

Referencias

Education, T. P. (2023). Redes Neuronales: Aprendizaje y Resolución de Problemas [August 10]. https://thepower.education/blog/redes-neuronales

GeeksforGeeks. (2024). What is LSTM - Long Short Term Memory? [June 10]. https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

IBM. (2021). Decision Tree Models [August 18]. https://www.ibm.com/docs/es/spss-modeler/saas?topic=trees-decision-tree-models

IBM. (n.d.-a). ¿Qué es la regresión logística? https://www.ibm.com/mx-es/topics/logistic-regression

IBM. (n.d.-b). K-Nearest Neighbors (KNN). https://www.ibm.com/mx-es/topics/knn

IBM. (n.d.-c). Naive Bayes. https://www.ibm.com/mx-es/topics/naive-bayes

IBM. (n.d.-d). Random Forest. https://www.ibm.com/mx-es/topics/random-forest

Para. (2021). XGBoost Stepwise Tuning using Optuna [March 17]. https://www.kaggle.com/code/para24/xgboost-stepwise-tuning-using-optuna

Torres, L. (2024). Optuna: ¿El futuro de la búsqueda de hiperparámetros? [July 6]. https://www.themachinelearners.com/optuna-busqueda-hiperparametros/

Tuning the hyper-parameters of an estimator. (s.f.). Scikit-learn [(n.d.)]. https://scikit-learn.org/stable/modules/grid_search.html