Inteligencia artificial avanzada para la ciencia de datos I

Reporte final de Solución del Reto

Autores:

Jose Alfredo García Rodríguez | A00830952
Diego Alberto Baños Lopez | A01275100
Carlos David Lozano Sanguino | A01275316
Carol Arrieta Moreno | A01275465

Profesores:

Iván Mauricio Amaya Contreras, Dra. Blanca R. Ruiz Hernández,
Frumencio Olivas Alvarez, Antonio Carlos Bento

13 de septiembre de 2023

Índice

1. Introducción

El hundimiento del Titanic es uno de los naufragios más infames de la historia. El 15 de abril de 1912, durante su viaje inaugural, el RMS Titanic, ampliamente considerado ïnsumergible", se hundió después de chocar con un iceberg. Desafortunadamente, no había suficientes botes salvavidas para todos a bordo, lo que provocó la muerte de 1.502 de los 2.224 pasajeros y tripulantes.

Si bien hubo algún elemento de suerte involucrado en la supervivencia, parece que algunos grupos de personas tenían más probabilidades de sobrevivir que otros. En este reto, se construirá un o varios modelos predictivos que respondan a la pregunta: "¿qué tipo de personas tenían más probabilidades de sobrevivir?ütilizando datos de los pasajeros (es decir, nombre, edad, sexo, clase socioeconómica, etc.).

Este proyecto busca arrojar luz sobre los factores que influyeron en la supervivencia de los pasajeros del Titanic, explorando patrones y relaciones en los datos. A través de la construcción de un modelo predictivo utilizando técnicas de machine learning, buscamos comprender mejor las variables que desempeñaron un papel crucial en las posibilidades de supervivencia. Esta aplicación de modelos de machine learning permite obtener información valiosa que puede ayudar a honrar la memoria de aquellos que viajaron en el trágico viaje inaugural del Titanic y aportar conocimientos útiles en el campo de la ciencia de datos y la estadística.

2. Entendimiento del set de datos del Titanic

Link del dataset: https://www.kaggle.com/competitions/titanic

El conjunto de datos del Titanic es una colección de información recopilada sobre los pasajeros que abordaron el famoso transatlántico RMS Titanic en su viaje inaugural en abril de 1912. Este conjunto de datos se ha convertido en un referente en el mundo de la ciencia de datos y el aprendizaje automático, y se utiliza comúnmente para ejercicios de análisis y modelado predictivo. A continuación, se describen algunas de las principales características incluidas en el conjunto de datos:

- **PassengerId:** Este campo representa un número único asignado a cada pasajero. Se utiliza para identificar de manera exclusiva a cada persona en el conjunto de datos.
- **Survived:** Esta variable es binaria y refleja si un pasajero sobrevivió o no al desastre. Un valor de "1ïndica que el pasajero sobrevivió, mientras que un valor de "0ïndica que no lo hizo.
- **Pclass (Clase):** La variable de clase representa la clase socioeconómica del pasajero. Puede tomar uno de tres valores: "1"para Primera Clase, "2"para Segunda Clase y "3"para Tercera Clase.
- **Name (Nombre):** Esta columna contiene el nombre completo de cada pasajero.
- **Sex (Sexo):** La variable de sexo indica el género del pasajero y puede ser "male"para masculino o "female"para femenino.
- **Age (Edad):** Representa la edad del pasajero en años.
- **SibSp (Hermanos/Cónyuges a bordo):** Indica la cantidad de hermanos o cónyuges que el pasajero tenía a bordo del Titanic.
- **Parch (Padres/Hijos a bordo):** Muestra la cantidad de padres o hijos que el pasajero tenía a bordo.
- **Ticket (Boleto):** Esta variable contiene el número de boleto del pasajero.
- **Fare (Tarifa):** Indica la tarifa que el pasajero pagó por su boleto.
- **Cabin (Cabina):** Representa el número de cabina del pasajero, cuando se conoce.

- **Embarked (Puerto de Embarque):** Indica el puerto de embarque del pasajero y puede tomar uno de tres valores: Ç"para Cherbourg, "Q"para Queenstown y "S"para Southampton.

Estas variables proporcionan información detallada sobre los pasajeros a bordo del Titanic, la diferencia entre el csv de datos de entrenamiento y prueba es que los de entrenamiento tienen la variable adicional survived descrita anteriormente mientras que los datos de prueba no con el fin de que el modelo que se elija pueda trabajar con la división de datos y aprenda de estos. El dataset esta conformado por 10 variables mencionadas anteriormente teniendo en total 891 datos cada columna de tipo int o float excepto dos que tienen algunos valores o datos tipo object indicando que están presentes ciertos valores nulos, combinando el total de la cantidad de columnas por filas sin tomar los valores nulos el dataset tiene 9826 datos en total.

3. Parte 1. Limpieza de Datos

Primeramente antes de limpiar los datos, tenemos que observar cómo están conformados estos últimos en el data frame correspondiente, en este caso tenemos en el csv las columnas PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin y Embarked, las cuales contienen diferentes tipos de datos y estos datos se podría decir que dicha información están en crudo, haciendo que para poder trabajar en el reto, tenemos que someter a este data frame a una limpieza de datos.

Así que empezamos exportando el csv que contiene los datos a una libreta en python (Jupyter Notebook), en el cual pedimos al programa que lo convirtiera a un data frame, una vez realizado ello se pidió una mayor descripción del data frame, una vez observado los datos exportados se decidió tratar dos columnas que a nuestro criterio es importante convertirlo a un tipo de dato el cual se pueda trabajar más fácilmente para su posterior análisis, las cuales son la columna de sex y de embarked.

Para la columna de Sex al ver que solo tenía dos valores posibles (male, female) usando la librería de sklearn se convirtieron dichos valores a números binarios en donde 1 representa male y el 0 representa female.

Por el otro lado para la columna Embarked se tienen 3 valores los cuales son S, C y Q los cuales se contaron cuántas veces se repetían en Embarked y en base a ello y para evitar el problema de multicolinealidad perfecta se decidió eliminar en los parámetros dummies Embarked Q, creando y dejando las columnas Embarked S y Embarked C, en el cual 1 representa que pertenece al embarked de esa letra y 0 que no pertenece a este último.

Una vez realizado dichas acciones se decidió analizar la correlación de variables con un mapa de correlación, esto nos ayuda a identificar posibles variables que puedan resultar útiles para la parte dos y así poder observar de mejor forma la naturaleza de estos mismos.

Una vez realizado lo anterior se contaron la cantidad de datos no nulos que tenía cada columna, ninguna de las columnas tenía valores nulos a excepción de dos, Age y Cabin, Cabin contaba con 697 valores nulos de 891 valores, por lo que solo el 22 % de los datos contaban con un valor real, por esta razón decidimos que Cabin no tiene la fiabilidad necesaria que nos ayude al posterior análisis para este reto, mientras que para Age el 81 % de los valores no eran nulos, esto se traduce a que solo se contaba con 177 valores nulos de 891, por lo que decidimos reemplazar los valores vacíos.

Para reemplazarlos nos dimos cuenta que en la columna de Name la mayoría de los nombres contenían un título al inicio; Master, Mr, Miss. ó Mrs, por lo que basándonos en esto podríamos saber los rangos de edades que cada persona podría tener de acuerdo con cómo iniciaba su nombre, por lo que se obtuvo la media de edad por cada título, obteniendo una media para los niños, una para hombres y otra para mujeres, la media obtenida para mujeres incluye los valores de niñas y mujeres, esto debido a que no se pudo observar una distinción de edades entre Miss o Mrs, por lo que para las mujeres se obtuvo la media sin importar cual de estos dos títulos tenia, y esta se le asignó a los valores vacíos dependiendo del título que tuviera el nombre, de igual forma encontramos que no todos los nombres tenían estos títulos, habían algunos que o no tenían o que aparecían como Dr., por lo que se realizó el mismo proceso para este conjunto de datos. De esta manera ningún valor de Age se mantuvo vacío.

Por último en esta parte usando seaborn y matplotlib decidimos si había algún valor atípico dentro del data frame, y acorde a las gráficas y a nuestro análisis llegamos a la conclusión de que no hay algún valor que sea realmente atípico y que por lo tanto amerita que sea reemplazado o modificado, haciendo que nuestro data frame se mantenga intacto y hayamos dado por concluido la limpieza de datos. El archivo con la limpieza de datos completa se llama Reto_Kaggle_Titanic.ipynb en el repositorio de github en la carpeta de limpieza de datos.

4. Parte 2. Selección, configuración y entrenamiento del modelo

Como pudimos observar anteriormente el tipo de problema al que estamos enfrentando en este dataset es uno de clasificación que consiste en predecir si un pasajero sobrevivió (etiqueta "1") o no sobrevivió (etiqueta "0") al hundimiento del Titanic, teniendo definido el tipo de problema decidimos implementar múltiples modelos de aprendizaje con el fin de analizar y ver su comportamiento en la predicción y calificación de sobrevivientes para al final seleccionar el mejor modelo para este reto, a continuación se presentan los modelos elegidos y por que son aptos para este problema.

4.1. Árboles de Decisión (Decision Trees):

Los árboles de decisión son modelos de machine learning que se utilizan ampliamente para problemas de clasificación. Estos modelos funcionan creando un árbol de decisiones en el que cada nodo representa una pregunta o prueba sobre una característica del conjunto de datos. Los árboles se dividen en ramas basadas en las respuestas a estas preguntas, y finalmente se llega a una decisión (etiqueta de clase) en las hojas del árbol.

4.1.1. Justificación para el Titanic:

Los árboles de decisión son fáciles de interpretar y visualizar, lo que permite comprender las reglas de toma de decisiones.

Pueden manejar datos mixtos (categóricos y numéricos) y son adecuados para el conjunto de datos del Titanic, que contiene una variedad de tipos de características.

Los árboles de decisión pueden capturar relaciones no lineales en los datos, como la influencia de la clase socioeconómica, la edad y el género en la supervivencia de los pasajeros.

4.2. Random Forest:

Random Forest es una extensión de los árboles de decisión que combina múltiples árboles para mejorar la precisión y reducir el sobreajuste. Funciona entrenando una colección (ensemble) de árboles de decisión y promediando sus predicciones.

4.2.1. Justificación para el Titanic:

Random Forest es una opción sólida para el conjunto de datos del Titanic debido a su capacidad para manejar múltiples características y lidiar con datos faltantes.

Reduce el sobreajuste inherente a los árboles de decisión individuales, lo que puede mejorar la generalización del modelo.

Puede evaluar la importancia de las características, lo que ayuda a identificar qué características tienen un mayor impacto en la supervivencia de los pasajeros.

4.3. XGBoost (XGBClassifier):

XGBoost, abreviatura de Extreme Gradient Boosting, es un algoritmo de boosting que ha demostrado ser altamente efectivo en una variedad de problemas de clasificación. Utiliza árboles de decisión como componentes base y realiza un entrenamiento secuencial para mejorar el rendimiento del modelo.

4.3.1. Justificación para el Titanic:

XGBoost es conocido por su capacidad para manejar conjuntos de datos pequeños a grandes y su capacidad para tratar con datos desequilibrados, lo que es relevante para el conjunto de datos del Titanic. Ofrece una alta precisión en la predicción y es una elección sólida cuando se busca mejorar el rendimiento del modelo en comparación con otros algoritmos.

Puede manejar características categóricas de manera efectiva, lo que es importante dada la naturaleza mixta de las características en el conjunto de datos del Titanic.

5. Implementación de los Modelos

5.1. Decision Trees

Para la implementación del modelo decidimos usar los datos procesados referenciados en la Parte 1 de este reporte en el que los datos crudos con valores nulos son corregidos con la limpieza, con esta parte lista empezamos con la implementación del modelo de la siguiente manera: Primero transformamos la columnas de “Embarked C” y de “Embarked D” a variables dummies como números binarios para así poder usar el valor de estas variables de una forma más simplificada, se decidió en este modelo no aplicar una técnica de resampling pese al desequilibrio que se pudo notar entre los que sobrevivieron y los que no ya que al analizar los valores de survived 342 sobrevivieron y 549 no sobrevivieron pero la diferencia no es extreman y los valores son cercanos. Una vez considerado ello el algoritmo empieza con la división del dataset en varios subdatasets, uno de entrenamiento y otro de validación, el porcentaje de datos que se utilizara para el entrenamiento será del 80 % mientras que el de validación será 20 %, esta elección fue debido a que como se tienen un conjunto de datos grande teniendo más de 9,000 datos la mejor decisión fue usar 20 % para los de validación , si el conjunto fuera más grande se aplicaría 10 % a los de validación y 90 % a los de entrenamiento, por el contrario si fuera más pequeño el porcentaje de datos de entrenamiento reduciría(esto se encuentra en el hiper parámetro test size=0.2)

Tras haber hecho estas pruebas de hiper parámetros corrimos esto para posteriormente aplicar la validación cruzada para graficar y analizar cual seria la profundidad deseada del algoritmo donde alteramos el hiper parámetro cv en 0.5 que es elección común porque proporciona un buen equilibrio entre una evaluación robusta y la eficiencia computacional. Tras esto obtuvimos los siguientes resultados: -La precisión del conjunto de entrenamiento aumenta constantemente a medida que aumenta la profundidad del árbol. Esto es esperado ya que un árbol más profundo puede ajustarse mejor a los datos de entrenamiento. -La precisión del conjunto de validación, sin embargo, alcanza un pico y luego comienza a disminuir. Esto sugiere que el modelo comienza a hacer overfitting en los datos de entrenamiento después de cierta profundidad. -La profundidad que parece proporcionar la mejor precisión en el conjunto de validación está entre 3 y 5. Con base en este análisis, una profundidad de 4 parece una elección razonable.

Tras esto comenzamos a entrenar y evaluar el algoritmo con métricas de desempeño tanto para los datos de entrenamiento como los de validación o prueba.

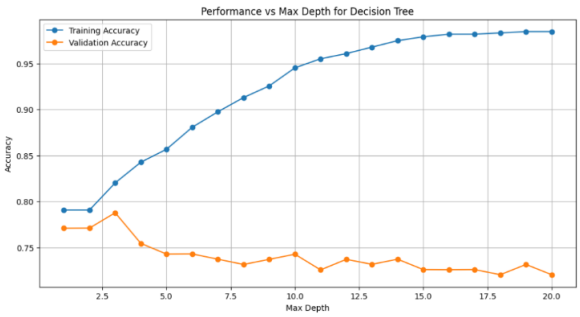

Figura 1: Greafico LC Decision Tree

```
Accuracy: 0.8100558659217877

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.93      0.86       112
           1       0.84      0.61      0.71        67

    accuracy                           0.81       179
   macro avg       0.82      0.77      0.78       179
weighted avg       0.81      0.81      0.80       179


Confusion Matrix:
 [[104   8]
 [ 26  41]]
```

Figura 2: Metricas de Desempeño y Resultados

De los resultados del accuracy scores se obtuvo que para la clase 0 (primera fila), la precisión es 0.80, lo que significa que el 80 % de las predicciones clasificadas como clase 0 fueron correctas. Para la clase 1 (segunda fila), la precisión es 0.84, lo que indica que el 84 % de las predicciones clasificadas como clase 1 fueron correctas. Para la clase 0, el recall es 0.93, lo que indica que el modelo identificó correctamente el 93 % de los casos reales de clase 0. Para la clase 1, el recall es 0.61, lo que significa que el modelo capturó el 61 % de los casos reales de clase 1. Al final se obtuvo un accuracy del modelo del 81 %, tras haber cambiado los parámetros como el tamaño de los datos de entrenamiento o el cv de validación cruzada vimos que el modelo puede bajar en precision y recall lo cual no es lo óptimo por eso llegamos a los mejores valores óptimos para el modelo obteniendo el accuracy de 81 % anteriormente mencionado.

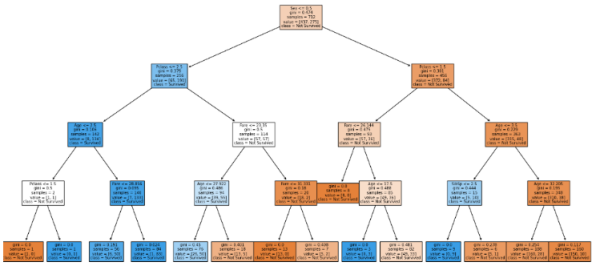

Figura 3: Visualización del Árbol de Decisiones

5.2. Random Forest

Random Forest es un algoritmo de aprendizaje automático que se utiliza comúnmente para resolver problemas de clasificación y regresión. En el contexto del desafío "Titanic - Machine Learning from Disaster", donde se busca predecir si los pasajeros del Titanic sobrevivieron o no al naufragio.

Antes de entrenar el modelo, notamos un desbalance en la cantidad de sobrevivientes y no sobrevivientes

en los datos del Titanic. Aplicamos SMOTE para equilibrar las clases. Luego, usamos el RandomForestClassifier de scikit-learn para entrenar el modelo, ya que puede lidiar con ruido y características irrelevantes, además de manejar relaciones no lineales y sobre ajustes.

Tras el equilibrio de las clases con SMOTE, dividimos el set de datos en subset de entrenamiento y prueba al igual que el decision tree por la cantidad de datos ya justificada se eligió 0.2 para el test size, con esto y las clases equilibradas se da paso la construcción de los árboles con GridSearch ya que es una estrategia sistemática utilizada para encontrar la mejor combinación de hiper parámetros que maximice el rendimiento del modelo de Random Forest.

Se define param grid para ajustar el modelo a 200 árboles y profundidad máx de 10 que se utilizarán para ajustar el modelo Random Forest. Estos hiper parámetros en la funcion incluye la cantidad de árboles (n estimators), la profundidad máxima de los árboles (max depth), los criterios de división (criterion), los parámetros relacionados con la división de nodos (min samples split, min samples leaf, min impurity decrease), el manejo del desequilibrio de clases (class weight), el uso de bootstrap (bootstrap), y la medida de impureza (criterion).

Se define un objeto de validación cruzada estratificada (kfold) con 5 divisiones para evaluar el rendimiento del modelo de manera robusta.Después se crea una instancia del modelo Random Forest (rf) con algunos valores predeterminados. Se utiliza GridSearchCV para buscar la combinación óptima de hiper parámetros dentro del espacio definido por param grid rf. La métrica de evaluación utilizada para seleccionar el mejor modelo es el F1-score ponderado (f1 macro). El GridSearchCV se ajusta a los datos previamente remuestreados (X resampled e y resampled), lo que indica que se está abordando un problema de desequilibrio de clases. Finalmente, se obtiene el mejor estimador (best rf), que es el modelo Random Forest con los hiper parámetros que proporcionaron el mejor rendimiento según el F1-score ponderado que se verán más adelante con las métricas de desempeño, se eligieron dos valores, 200 y 400, para encontrar la cantidad óptima de árboles en el bosque. Esto se hace porque un mayor número de árboles generalmente mejora la capacidad de generalización del modelo, reduciendo el riesgo de sobreajuste. En profundidad máxima se prueban tres valores, 5, 10 y 15, para limitar la profundidad máxima de los árboles. Para min impurity decrease se prueban tres valores, 0.0, 0.1 y 0.2, para controlar la disminución mínima requerida en la impureza para realizar una división en un nodo. Este parámetro afecta la estructura del árbol y puede ayudar a evitar divisiones insignificantes. Todos esta selección de hiperparametros sirven para evitar el sobreajuste que se verá a continuación con la Learning Curve que verá si no hay underfitting o overfitting.

Tras esto se graficó para el modelo la curva de aprendizaje en la que se obtuvieron resultados similares a los de decisión tree en el que el puntaje de entrenamiento alto indica que el modelo está realizando muy bien en el conjunto de datos de entrenamiento. Esto sugiere que el modelo es capaz de aprender y ajustarse bien a los datos de entrenamiento, lo que podría ser un indicativo de que tiene una capacidad suficiente para capturar la complejidad de los datos. Por otro lado, la curva de validación cruzada también está aumentando a medida que se aumenta el tamaño del conjunto de entrenamiento o se ajusta algún hiper parámetro del modelo. Esto es positivo, ya que muestra que el modelo está mejorando su capacidad de generalización a medida que se le proporcionan más datos o se realizan ajustes. Cuando las dos curvas se acercan entre sí, indica que el modelo está generalizando de manera efectiva y no está sobreajustado.

De esta learning curve y la ROC que es una métrica que cuantifica la calidad global del modelo en términos de su capacidad para discriminar entre las clases se obtiene un 0.88 o 88 % de AUC lo que indica que el modelo tiene un buen rendimiento en la tarea de clasificación, siendo capaz de distinguir efectivamente entre las clases positivas y negativas en el conjunto de datos.

Tras esto y la obtención de la gráfica ROC Curve que se puede ver en el código aplicamos una matriz de confusión para ver las métricas de desempeño para los datos de entrenamiento y prueba en el que obtuvimos que para la clase 0, la precisión es del 89 %, lo que significa que el 89 % de las predicciones de clase 0 fueron correctas. Para la clase 1, la precisión es del 68 %, lo que indica que el 68 % de las predicciones de clase 1 fueron correctas.En cuanto a recall que mide la proporción de ejemplos positivos reales que se predijeron correctamente, para la clase 0, el recall es del 83 %, lo que significa que el modelo identificó correctamente el 83 % de los ejemplos de la clase 0. Para la clase 1, el recall es del 79 %, indicando que el 79 % de los ejemplos

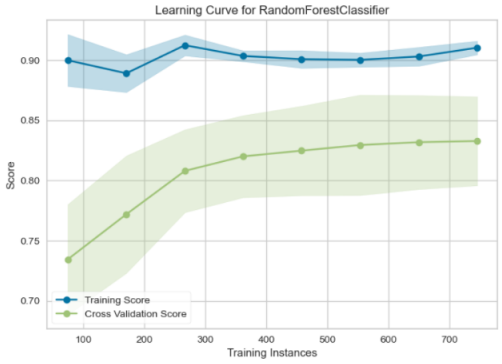

Figura 4: Learning Curve Random Forest

de la clase 1 se predijeron correctamente. Y la exactitud general del modelo es del 82 %, lo que significa que el 82 % de todas las predicciones fueron correctas.

```
Random Forest Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.83      0.86       123
           1       0.68      0.79      0.73        56

    accuracy                           0.82       179
   macro avg       0.79      0.81      0.79       179
weighted avg       0.83      0.82      0.82       179

Random Forest Classifier Confusion Matrix:
[[102  21]
 [ 12  44]]
```

Figura 5: Metricas de Desempeño Random Forest

Se obtuvo de lo anterior un buen desempeño del modelo sin embargo pensamos que podría ser mejor cambiando algunos parámetros como cantidad de árboles y profundidad por lo tanto el siguiente paso para este modelo fue usar una biblioteca llamada optuna para realizar una búsqueda de hiper parámetros óptimos.

Para esta búsqueda creamos un código en el que se definiera una función llamada objective que toma un objeto trial como argumento y representa el objetivo de optimización. Dentro de esta función, sugiere valores para varios hiper parámetros del modelo Random Forest, como el número de árboles, la profundidad máxima, el número mínimo de muestras requeridas para dividir un nodo y el número mínimo de muestras requeridas en un nodo hoja que es lo que nos interesa, tras esto se crea una instancia del modelo Random Forest (clf) con los hiper parámetros sugeridos por Optuna y lo entrena en los datos de entrenamiento, Tras esto empieza a generar predicciones en el conjunto de prueba utilizando el modelo ajustado y calcula el F1-score macro como métrica de evaluación para después se configura un estudio de Optuna (study) que se utilizará para llevar a cabo la optimización. La dirección de la optimización se establece en 'maximize', lo que significa que se busca maximizar el valor de la métrica objetivo.

Tras correr y pasar el proceso del código con la librería optuna se obtuvo que los mejores hiperparametros serian 600 n estimators, profundidad de 14, min samples split: 3, min samples leaf: 4 con el mejor accuracy de 81 % o 0.81 en resultados de código.

```
Best Hyperparameters: {'n_estimators': 600, 'max_depth': 14, 'min_samples_split': 3, 'min_samples_leaf': 4}
Best Accuracy: 0.8189838546570423
```

Figura 6: Resultado de Optuna para mejores hiper parametros

Con estos supuestos hiper parametros pondremos a prueba estos resultados de la misma manera que el primer intento con una learning curve y métricas de desempeño para obtener conclusiones de los resultados y precisión. Tras aplicar los mismos códigos que generan las gráficas con el nuevo modelo optimizado se obtuvo lo siguiente:

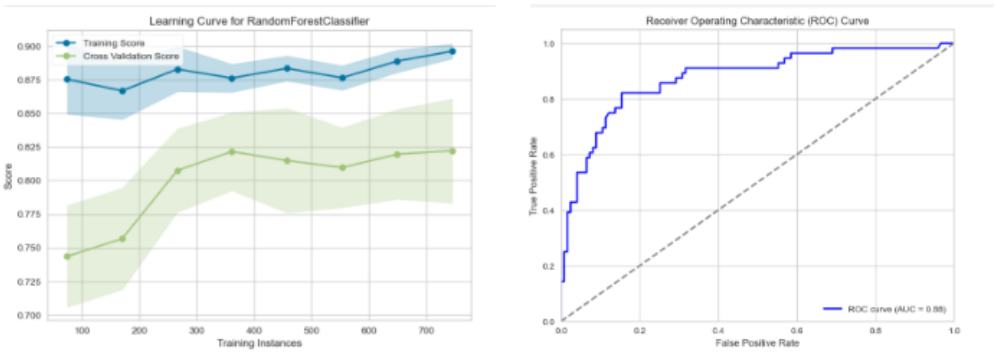

Figura 7: LC y ROC con nuevos hiper parametros

Al igual que el primer intento la recta de training score esta arriba de la de cross validation lo que significa que el modelo se está ajustando muy bien a los datos de entrenamiento como vimos anteriormente, y a medida que aumenta las instancias de entrenamiento la recta de cross validation aumenta y sube acercándose a la recta de training score lo que sugiere que a medida que se proporcionan más datos al modelo para entrenamiento, su capacidad de generalización mejora. En conclusión de esta gráfica se obtiene una curva de aprendizaje mucho mejor que la anterior con los nuevos hiperparametros.

El último paso para comprobar el cambio serian las métricas de desempeño en el que se obtuvieron mejores resultados como Para la clase 0 (etiqueta 0), la precisión es del 91 %, lo que significa que el 91 % de las predicciones de clase 0 fueron correctas. Para la clase 1 (etiqueta 1), la precisión es del 71 %, indicando que el 71 % de las predicciones de clase 1 fueron correctas. El recall es del 85 % para la clase 0 y del 82 % para la clase 1, lo que sugiere que el modelo identificó correctamente el 85 % de los ejemplos reales de la clase 0 y el 82 % de los ejemplos reales de la clase 1. El F1-score es del 0.88 para la clase 0 y del 0.76 para la clase 1, con un F1-score macro promedio del 0.82. La exactitud general del modelo es del 84 %, lo que significa que el 84 % de todas las predicciones fueron correctas en el conjunto de prueba. En resumen, los resultados indican que el modelo Random Forest Classifier tiene un buen rendimiento general en la tarea de clasificación, con un buen equilibrio entre precisión y capacidad de recuperación para ambas clases obteniendo la mejor precisión de las pruebas anteriores concluyendo así la comparación y pruebas de hiperparametros. Como extra graficamos con Feature Importances las variables que influyen más en la supervivencia o muerte de tripulantes dando como las más influyentes, la pclass, sex,fare, ,age,embarque etc.

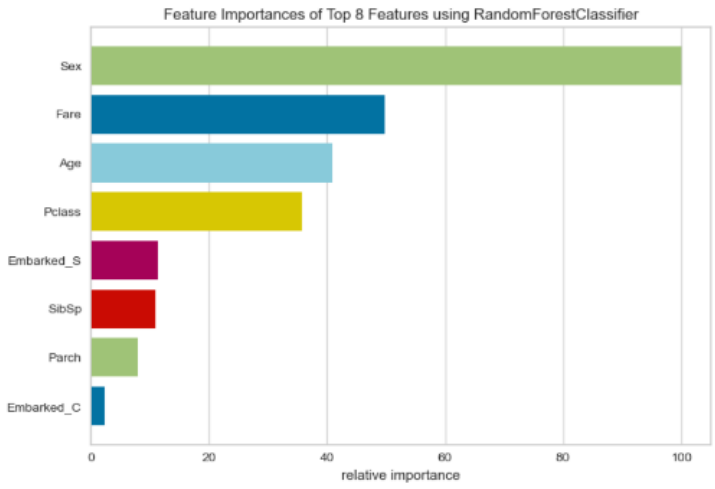

Figura 8: Variables de Importancia para el reto Titanic

5.3. XGBClassifier

XGBoost (Extreme Gradient Boosting) Classifier es un algoritmo de aprendizaje automático que pertenece a la familia de los métodos de boosting. Se utiliza principalmente para problemas de clasificación y regresión. XGBoost es una técnica avanzada que combina múltiples modelos de aprendizaje débiles, generalmente árboles de decisión simples, en un predictor más fuerte y preciso.

5.3.1. Metodología del Modelo

En este reto, se utiliza una metodología cuidadosamente diseñada para garantizar la reproducibilidad y la evaluación efectiva de los modelos. Antes de realizar cualquier prueba, se lleva a cabo un proceso de preparación de datos y se establecen parámetros clave para el modelo XGBoost.

1. Preparación de Datos

Para asegurar la confiabilidad de nuestros resultados, se utiliza la función 'train_test_split' de la biblioteca scikit-learn para dividir nuestro conjunto de datos en conjuntos de entrenamiento y prueba. La relación elegida es de 90 % para entrenamiento (datos de entrenamiento) y 10 % para prueba (datos de prueba), con un valor de 'random_state' establecido en 77. Esto garantiza que la división de datos se mantenga constante a lo largo de todas las pruebas, lo que facilita la comparación de los modelos.

2. Métricas de Desempeño

Dada la presencia de un ligero desbalance en las clases objetivo de nuestro problema, hemos seleccionado tres métricas principales para evaluar el rendimiento de nuestros modelos: precisión (precision), exhaustividad (recall) y el puntaje F1. Estas métricas son especialmente adecuadas para problemas de clasificación desequilibrados, ya que brindan una visión completa del rendimiento del modelo.

a. Precisión (Precision):

La precisión mide la proporción de predicciones positivas correctas entre todas las predicciones positivas. Se calcula mediante la fórmula:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Donde:

- True Positives (Verdaderos Positivos) son las muestras positivas que fueron correctamente clasificadas como positivas.
- False Positives (Falsos Positivos) son las muestras negativas que fueron incorrectamente clasificadas como positivas.

b. Exhaustividad (Recall):

La exhaustividad mide la proporción de instancias positivas correctamente clasificadas entre todas las instancias reales positivas. Se calcula mediante la fórmula:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Donde:

- True Positives (Verdaderos Positivos) son las muestras positivas que fueron correctamente clasificadas como positivas.

- False Negatives (Falsos Negativos) son las muestras positivas que fueron incorrectamente clasificadas como negativas.

c. Puntaje F1:

El puntaje F1 es una métrica que combina precisión y exhaustividad en una sola medida. Se calcula mediante la fórmula:

$$F1 = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall}$$

Donde:

- La precisión (precision) mide la proporción de predicciones positivas correctas entre todas las predicciones positivas.
- La exhaustividad (recall) mide la proporción de instancias positivas correctamente clasificadas entre todas las instancias reales positivas.

Estas métricas nos permiten evaluar de manera integral el rendimiento de nuestros modelos, considerando tanto la capacidad de identificar las instancias positivas correctamente (exhaustividad) como la precisión en la clasificación de las instancias positivas (precisión), además del equilibrio entre ambas métricas proporcionado por el puntaje F1.

3. Hiperparámetros de XGBoost

El modelo XGBoost se utiliza como algoritmo de aprendizaje automático en este ensayo. Se explorarán y variarán varios hiperparámetros clave para optimizar el rendimiento del modelo. Los hiperparámetros que serán objeto de variación son:

- ‘n_estimators‘: El número de árboles de decisión en el ensamble.
- ‘max_depth‘: La profundidad máxima de cada árbol en el ensamble.
- ‘learning_rate‘: La tasa de aprendizaje, que controla la contribución de cada árbol al modelo final.
- ‘subsample‘: La fracción de muestras utilizada para entrenar cada árbol.
- ‘colsample_bytree‘: La fracción de características utilizada para entrenar cada árbol.
- ‘lambda‘: La penalización L2 aplicada a los términos de regularización.
- ‘alpha‘: La penalización L1 aplicada a los términos de regularización.

La variación de estos hiperparámetros nos permitirá encontrar la combinación óptima que maximice el puntaje F1 en nuestro problema de clasificación.

En resumen, esta metodología establece una base sólida para la experimentación y evaluación de modelos, garantizando la consistencia en la división de datos, utilizando métricas adecuadas para problemas desequilibrados y explorando una variedad de hiperparámetros para optimizar el rendimiento del modelo XGBoost en nuestro reto que se hará usando el framework optuna variando los hiperparámetros previamente descritos.

5.4. Análisis de resultados

Tomando en cuenta que el modelo que estamos evaluando es el XGBClassifier sin ningún hiperparámetro ajustado, es decir, los hiperparámetros seleccionados fueron solamente unos básicos:

- n_estimators = 100
- max_depth = 3
- learning_rate = 0.1

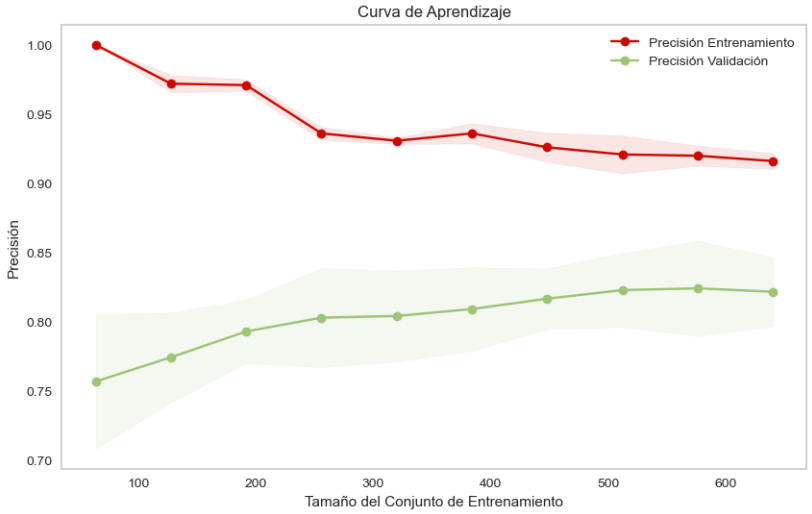

Figura 9: Curva de aprendizaje de XGB

La curva de aprendizaje nos muestra una mejora constante en la precisión indica que el modelo está aprendiendo de los datos y está mejorando su capacidad para predecir los resultados. Además la curva que es relativamente plana en la región de convergencia indica que el modelo ha alcanzado un punto en el que no está mejorando significativamente su precisión al agregar más iteraciones de entrenamiento. Esto sugiere que el modelo está generalizando bien a los datos nuevos y no está sobreajustándose a los datos de entrenamiento.

Una curva ROC como en la figura 10 con un AUC de 0.84 es una curva ROC decente. Esto significa que el modelo tiene buena capacidad para discriminar entre las dos clases. En otras palabras, el modelo es bueno para predecir si una muestra pertenece a la clase positiva (sobrevive) o a la clase negativa (no sobrevive).

En este caso en la Figura 11, el modelo predijo correctamente 49 muestras de la clase positiva y solo 27 muestras de la clase negativa. El modelo también predijo incorrectamente 9 muestras de la clase positiva y 5 muestras de la clase negativa. Resultados buenos considerando que es un modelo sin refinar.

En este caso, como podemos ver en la Figura 12 el modelo tiene una precisión de 0.84 para la clase 0, una precisión de 0.84 para la clase 1, una sensibilidad de 0.91 para la clase 0, una sensibilidad de 0.75 para la clase 1, y un f1-score de 0.88 para la clase 0 y 0.79 para la clase 1. La precisión ponderada es de 0.84 y el f1-score ponderado es de 0.83.

A pesar de que el modelo se desempeña bien, hay que tener en cuenta que es una aplicación muy básica del mismo, aún se pueden hacer alguna cosas para mejorar principalmente nuestro recall en los 1's principalmente, que en este caso resulta particularmente importante debido a la naturaleza de los datos, en donde predecir correctamente un 1 nos diría si la persona no tiene chances de sobrevivir.

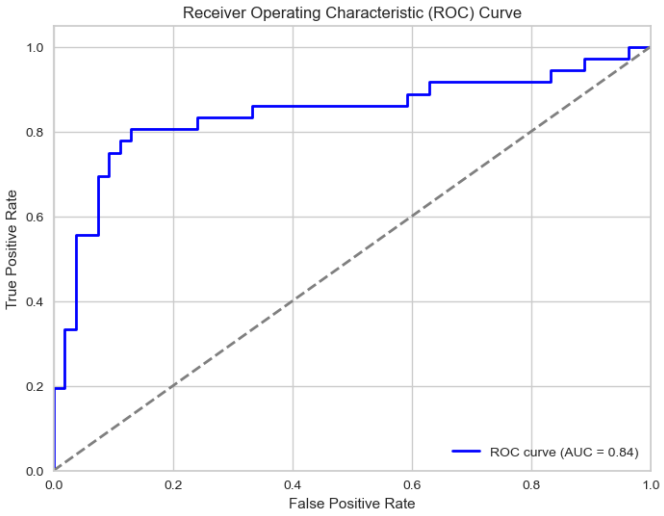

Figura 10: Curva ROC de XGB

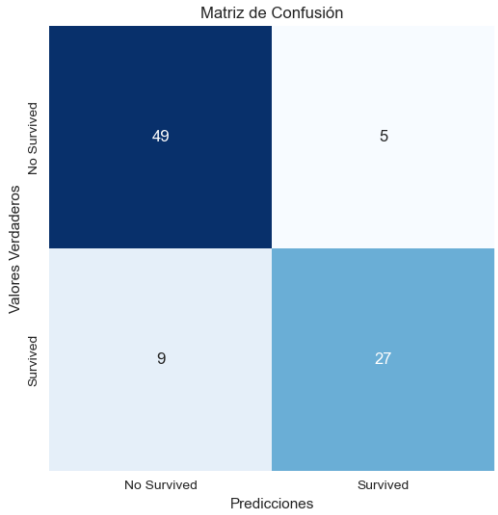

Figura 11: Matriz de confusión de XGB

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.91	0.88	54
1	0.84	0.75	0.79	36
accuracy			0.84	90
macro avg	0.84	0.83	0.83	90
weighted avg	0.84	0.84	0.84	90

Figura 12: reporte de clasificación de XGB

5.5. Elección del Modelo para la solución del reto y posterior evaluación/refinamiento

Tras haber ejecutado y obtenido resultados de los tres modelos en métricas y pruebas decidimos usar XGB como el modelo final para la solución del reto y refinamiento del mismo por las siguientes razones:

Comparación de Métricas de Desempeño

Se evaluaron los modelos en términos de F1-score, que combina precisión y recall, y proporciona una medida equilibrada del rendimiento en la clasificación:

- Decision Tree: Obtuvo un F1-score de 0.81 (sin optimizar).
- Random Forest: Obtuvo un F1-score de 0.79 (sin optimizar).
- XGBoost: Obtuvo un F1-score de 0.83 (sin optimizar).

Claramente, XGBoost logró el F1-score más alto, lo que sugiere un mejor equilibrio entre falsos positivos y falsos negativos.

Robustez y Regularización

XGBoost incluye técnicas de regularización que ayudan a prevenir el sobreajuste, lo que lo hace más robusto en comparación con Decision Tree y Random Forest, que pueden ser propensos al sobreajuste.

Capacidad de Ajuste de Hiperparámetros

XGBoost ofrece una amplia gama de hiperparámetros ajustables, lo que facilita la búsqueda de configuraciones óptimas y el afinamiento del modelo. Esto es más flexible que Decision Tree y Random Forest.

En conclusión, basándonos en las métricas de desempeño, la capacidad de regularización, la flexibilidad en la optimización de hiperparámetros, la eficiencia en el uso de recursos y la popularidad en la comunidad de aprendizaje automático, se selecciona XGBoost como el modelo preferido para resolver el reto del Titanic. Su alto F1-score y su capacidad de generalización lo hacen una elección sólida y confiable para problemas de clasificación como este.

6. Evaluación y Refinamiento del modelo

Como definimos anteriormente el modelo que obtuvo un mejor desempeño en nuestro problema de clasificación fue el XGBoost Classifier, es por eso que va a ser el modelo en que nos vamos a dedicar a refinar minuciosamente, mediante el proceso que se va a describir en la metodología que es similar a XGBoost porque se trata del mismo modelo.

6.1. Metodología del modelo

En este reto, se utiliza una metodología cuidadosamente diseñada para garantizar la reproducibilidad y la evaluación efectiva de los modelos. Antes de realizar cualquier prueba, se lleva a cabo un proceso de preparación de datos y se establecen parámetros clave para el modelo XGBoost.

1. Preparación de Datos

Para asegurar la confiabilidad de nuestros resultados, se utiliza la función 'train_test_split' de la biblioteca scikit-learn para dividir nuestro conjunto de datos en conjuntos de entrenamiento y prueba. La relación elegida es de 90 % para entrenamiento (datos de entrenamiento) y 10 % para prueba (datos de prueba), con un valor de 'random_state' establecido en 77. Esto garantiza que la división de datos se mantenga constante a lo largo de todas las pruebas, lo que facilita la comparación de los modelos.

Además para mejorar nuestro modelo, implementamos una ampliación en la feature engineering, cuyos detalles se encuentran detallados en XGB.ipynb. Aunque seguimos enfrentando el desafío del desequilibrio en las clases, esta vez optamos por no aplicar ningún resampling, esta decisión se basa en la observación previa de un sobreajuste en el modelo de Random Forest (random_forest.ipynb).

2. Métricas de Desempeño

Dada la presencia de un ligero desbalance en las clases objetivo de nuestro problema, hemos seleccionado tres métricas principales para evaluar el rendimiento de nuestros modelos: precisión (precision), exhaustividad (recall) y el puntaje F1. Estas métricas son especialmente adecuadas para problemas de clasificación desequilibrados, ya que brindan una visión completa del rendimiento del modelo.

a. Precisión (Precision):

La precisión mide la proporción de predicciones positivas correctas entre todas las predicciones positivas. Se calcula mediante la fórmula:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Donde:

- True Positives (Verdaderos Positivos) son las muestras positivas que fueron correctamente clasificadas como positivas.
- False Positives (Falsos Positivos) son las muestras negativas que fueron incorrectamente clasificadas como positivas.

b. Exhaustividad (Recall):

La exhaustividad mide la proporción de instancias positivas correctamente clasificadas entre todas las instancias reales positivas. Se calcula mediante la fórmula:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Donde:

- True Positives (Verdaderos Positivos) son las muestras positivas que fueron correctamente clasificadas como positivas.
- False Negatives (Falsos Negativos) son las muestras positivas que fueron incorrectamente clasificadas como negativas.

c. Puntaje F1:

El puntaje F1 es una métrica que combina precisión y exhaustividad en una sola medida. Se calcula mediante la fórmula:

$$F1 = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall}$$

Donde:

- La precisión (precision) mide la proporción de predicciones positivas correctas entre todas las predicciones positivas.
- La exhaustividad (recall) mide la proporción de instancias positivas correctamente clasificadas entre todas las instancias reales positivas.

Estas métricas nos permiten evaluar de manera integral el rendimiento de nuestros modelos, considerando tanto la capacidad de identificar las instancias positivas correctamente (exhaustividad) como la precisión en la clasificación de las instancias positivas (precisión), además del equilibrio entre ambas métricas proporcionado por el puntaje F1.

3. Hiperparámetros de XGBoost

El modelo XGBoost se utiliza como algoritmo de aprendizaje automático en este ensayo. Se explorarán y variarán varios hiperparámetros clave para optimizar el rendimiento del modelo. Los hiperparámetros que serán objeto de variación son:

- 'n_estimators': El número de árboles de decisión en el ensamble.
- 'max_depth': La profundidad máxima de cada árbol en el ensamble.
- 'learning_rate': La tasa de aprendizaje, que controla la contribución de cada árbol al modelo final.
- 'subsample': La fracción de muestras utilizada para entrenar cada árbol.
- 'colsample_bytree': La fracción de características utilizada para entrenar cada árbol.
- 'lambda': La penalización L2 aplicada a los términos de regularización.
- 'alpha': La penalización L1 aplicada a los términos de regularización.

La variación de estos hiperparámetros nos permitirá encontrar la combinación óptima que maximice el puntaje F1 en nuestro problema de clasificación.

4. Eliminación de variables irrelevantes

Para eliminar el posible ruido que tenia el modelo y mejorar aun mas los resultados obtenidos, decidimos eliminar las variables cuya importancia fuera menor a 0.05 del modelo para así quedarnos solamente con las variables que impacten a las predicciones como en la Figura 14 y ahorrar poder computacional. Esto se hizo con la ayuda de atributo del modelo XGB llamado feature_importance, luego de eliminar las variables que no nos impactan en nada, el análisis se hace de nuevo pero sin esas variables

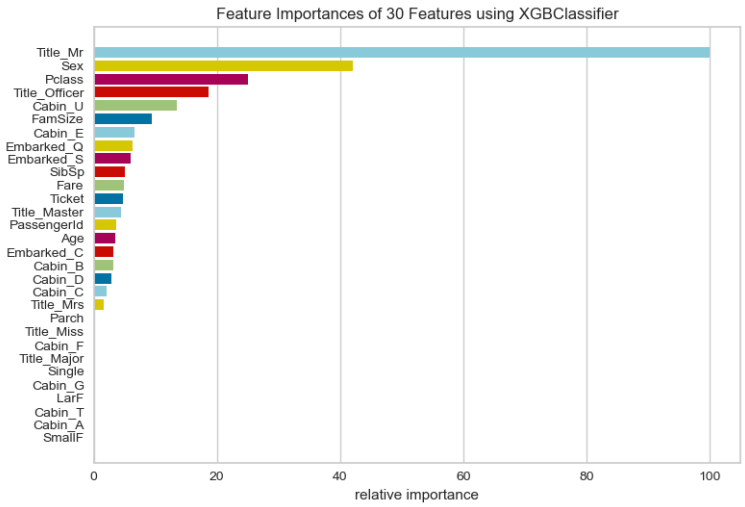

Figura 13: Importancia de variables pre-analisis

5. Optimización de hiperparámetros de XGBoost con Optuna

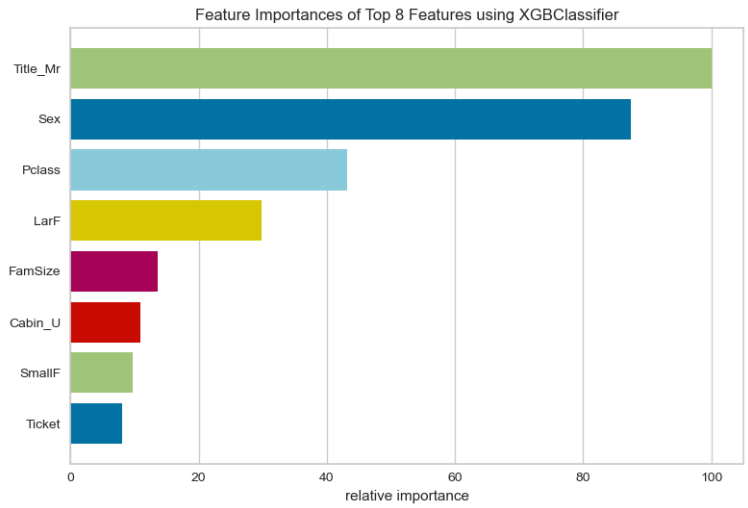

Figura 14: Importancia de variables post-analisis

Optuna es un framework de software de optimización automática de hiperparámetros. Nos ayuda a encontrar los 'mejores' hiperparámetros para tus modelos de aprendizaje automático.

Tenemos que definir la función objetivo que queremos optimizar. En este caso, estámos optimizando el F1 score de un modelo XGBClassifier. La función study.optimize() llama a Optuna para optimizar la función objetivo. El parámetro n_trials que dejamos con un valor de 100 especifica el número de pruebas que Optuna ejecutará.

En resumen, esta metodología establece una base sólida para la experimentación y evaluación de modelos, garantizando la consistencia en la división de datos, utilizando métricas adecuadas para problemas desequilibrados y explorando una variedad de hiperparámetros para optimizar el rendimiento del modelo XGBoost en nuestro ensayo que se hará usando el framework optuna variando los hiperparámetros previamente descritos.

6.1.1. Efecto de la regularización

La regularización de los datos es una técnica que se utiliza para reducir el sobreajuste en los modelos de aprendizaje automático. El sobreajuste ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento y no es capaz de generalizar bien a los datos nuevos. La regularización ayuda a evitar el sobreajuste al agregar una penalización a los pesos del modelo. Esta penalización hace que los pesos sean más pequeños, lo que ayuda a que el modelo sea menos sensible a los ruidos y a los artefactos en los datos de entrenamiento.

En la Figura 15 se muestra una curva de aprendizaje para un modelo de clasificación sin regularización. Como se puede ver, la precisión del modelo aumenta rápidamente al principio del entrenamiento y se mantiene hasta el 1 durante todos los CV. Esto se debe a que el modelo se está ajustando demasiado a los datos de entrenamiento.

En la Figura 15 se muestra una curva de aprendizaje para el mismo modelo, pero con regularización. Como se puede ver, la precisión del modelo es más constante a lo largo del entrenamiento y se puede ver como ambas curvas de aprendizaje convergen, es decir, se acercan más entre si a comparación cuando no usamos técnicas de regularización como en 15. Esto se debe a que la regularización ayuda a evitar que el modelo se ajuste demasiado a los datos de entrenamiento.

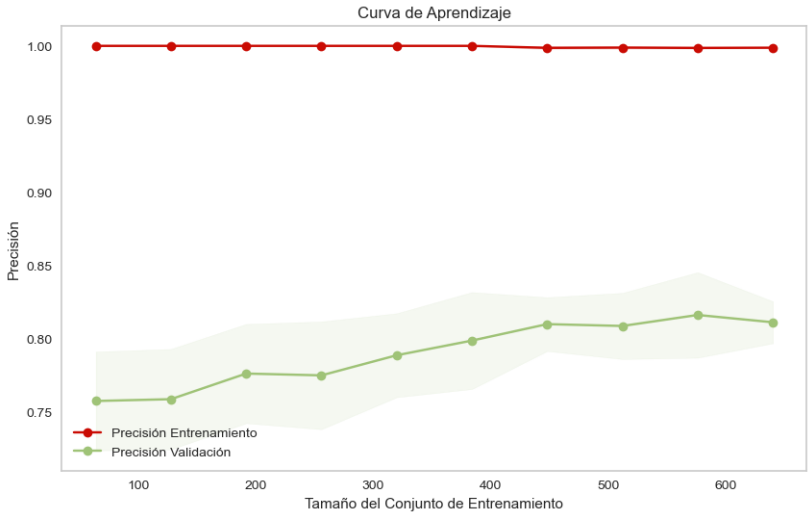

Figura 15: Cruva de aprendizaje sin regularización

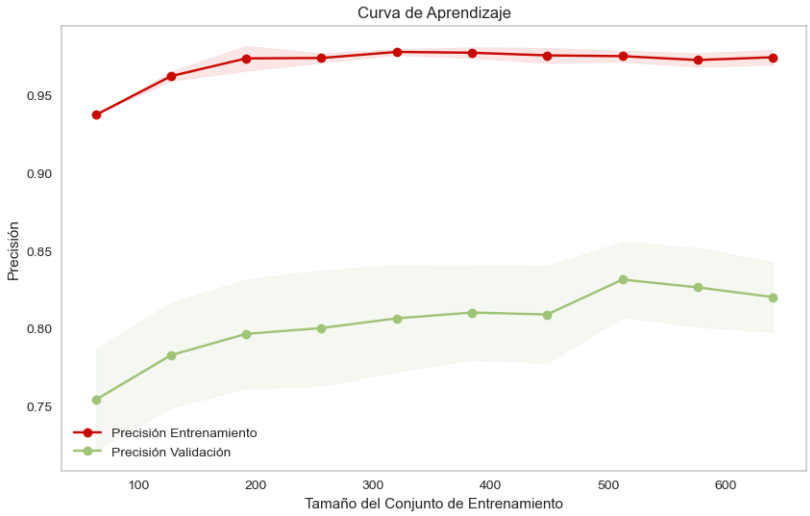

Figura 16: Cruva de aprendizaje con regularización L1 y L2

6.1.2. Análisis de resultados

Tomando como modelo a evaluar el modelo en el que obtuvimos un mejor f1 score y dónde las curvas de aprendizaje se veían mejor en cuestiones de ajuste del modelo, vamos a considerar en el que usamos técnicas de regularización.

La curva de aprendizaje nos muestra una mejora constante en la precisión indica que el modelo está aprendiendo de los datos y está mejorando su capacidad para predecir los resultados. Además la curva que es relativamente plana en la región de convergencia indica que el modelo ha alcanzado un punto en el que no está mejorando significativamente su precisión al agregar más iteraciones de entrenamiento. Esto sugiere que el modelo está generalizando bien a los datos nuevos y no está sobreajustándose a los datos de entrenamiento.

Una curva ROC como en la figura 18 con un AUC de 0.92 es una curva ROC muy buena. Esto significa que el modelo tiene una alta capacidad para discriminar entre las dos clases. En otras palabras, el modelo es muy bueno para predecir si una muestra pertenece a la clase positiva (sobrevive) o a la clase negativa (no

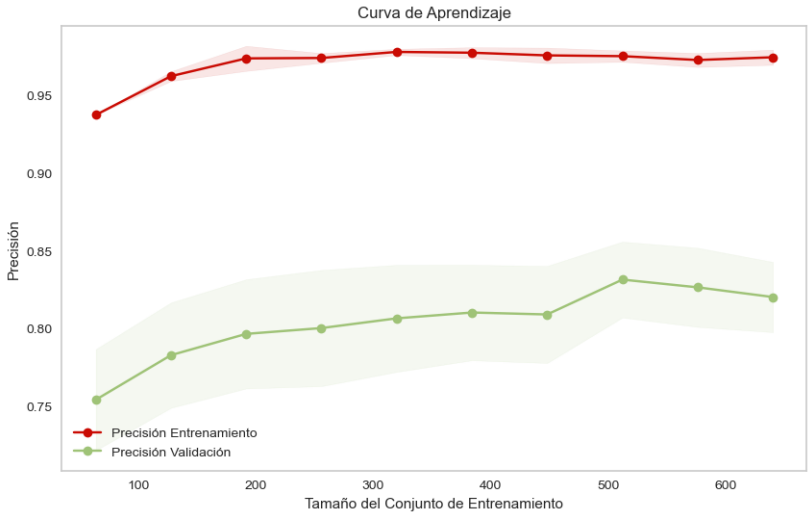

Figura 17: Curva de aprendizaje del mejor modelo

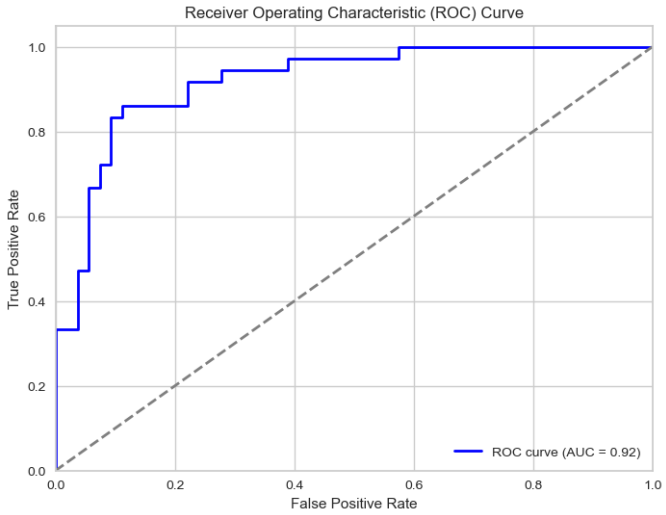

Figura 18: Curva ROC del mejor modelo

sobrevive).

En este caso, el modelo predijo correctamente 48 muestras de la clase positiva y 31 muestras de la clase negativa. El modelo también predijo incorrectamente 6 muestras de la clase positiva y 5 muestras de la clase negativa. Resultados bastante buenos considerando que como podemos ver en la Figura 19 los valores en la diagonal son muy superiores al resto

En este caso, como podemos ver en la Figura 20 el modelo tiene una precisión de 0.91 para la clase 0, una precisión de 0.84 para la clase 1, una sensibilidad de 0.89 para la clase 0, una sensibilidad de 0.86 para la clase 1, y un f1-score de 0.90 para la clase 0 y 0.85 para la clase 1. La precisión ponderada es de 0.88 y el f1-score ponderado es de 0.88.

En general, el classification report indica que el modelo tiene un rendimiento bueno. El modelo es capaz de clasificar correctamente el 88 % de las muestras, y es especialmente bueno para clasificar las muestras de la clase 0 (no sobrevive).

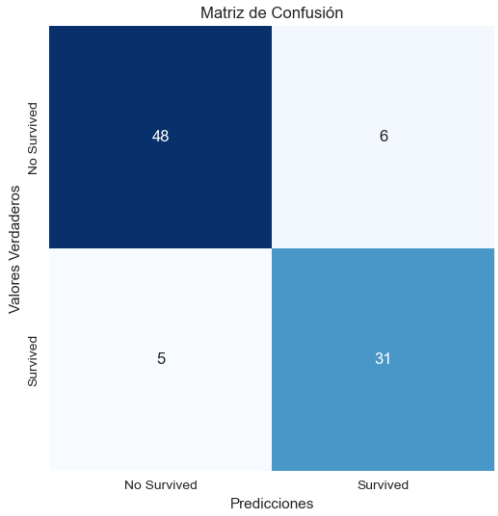

Figura 19: Matriz de confusión del mejor modelo

Classification Report:

	precision	recall	f1-score	support
0	0.91	0.89	0.90	54
1	0.84	0.86	0.85	36
accuracy			0.88	90
macro avg	0.87	0.88	0.87	90
weighted avg	0.88	0.88	0.88	90

Figura 20: reporte de clasificación del mejor modelo

Es evidente que hemos logrado una mejora en el F1-score, que era nuestro objetivo principal. Esto indica que este es el mejor modelo que hemos obtenido hasta la fecha. Es posible que valga la pena aplicar un enfoque similar a los demás modelos para explorar la posibilidad de mejorarlos aún más.

7. Interfaz

Para facilitar el proceso de predicción al usuario realizamos una interfaz web, para realizar dicha página web éramos conscientes que se tenía que dividir en dos partes, una era la parte interactiva con el usuario en donde se ingresa los datos que desea el usuario, podríamos decir que es la cara de la interfaz, que es el frontend, y la otra parte consiste en el manejo de los datos ingresados del usuario, procesarlo de una forma invisible, se puede decir que es el cerebro de la interfaz, que es el backend.

Una vez entendido ello observamos diferentes herramientas y en la parte de frontend sé decidió usar React, que tiene de base Javascript, ya que nos brinda una amplia opción de librerías a usar así como documentaciones sobre estas, ya decidido el framework del frontend.

Lo primero en lo que trabajamos fue el frontend, para el que decidimos que el usuario tuviera que ingresar los valores como si fuera un formulario, por lo que empezamos el diseño de esta manera, decidimos utilizar una librería llamada MUI debido a la gran opción de componentes que nos ofrece, una vez importada está librería colocamos un header pensando en un futuro crecimiento, en el cual después se puedan agregar diferentes opciones o un menú.

Después de esto fuimos agregando cuadros de textos cada uno con el nombre de la variable que requería el modelo para realizar la predicción, fuimos jugando con las medidas de estos cuadros hasta obtener el tamaño adecuado para que el diseño fuera estético, fácil de entender y que no ocupará más espacio del que

requería, después de esto agregamos un botón con el objetivo de que apretando este se muestre la predicción de acuerdo a los datos.

Para terminar el frontend se hicieron ajustes en los cuadros de textos, para el cuadro de tex de 'Sex' se añadió un cuadro desplegable el cual no permitía escribir cualquier valor, solo mostraba las opciones posibles, esto para asegurarnos que no afecte al modelo a la hora de realizar la predicción. Una vez realizado ello el frontend tiene un botón que cuando se usa cuando todos los datos están rellenados de manera adecuada hace un llamado al backend para que haga la predicción y una vez realizado ello el frontend muestra el resultado de la predicción.

Para el backend se utilizó Flask, el cual es un framework que permite crear aplicaciones web usando python como base, en este caso como el modelo seleccionado fue entrenado en python, se pensó que era prudente la idea de hacer el backend en el mismo lenguaje, para así exportar el modelo entrenado y simplemente ejecutarlo para hacer predicciones, dentro de lo que sería la parte lógica la página.

Lo que hace el backend a grandes rasgos es que carga el modelo del titanic seleccionado, y espera a que el frontend lo llame para hacer la predicción, una vez que el frontend hace el llamado al backend, el backend recolecta la información que le envió el frontend, hace una transformación de datos para que sean compatibles con el modelo exportado, una vez realizada la transformación de datos se usa el modelo que se seleccionó previamente para realizar la predicción, una vez realizada se devuelve la predicción al frontend, de esta forma este último pueda mostrar al usuario, acorde al modelo, si el pasajero sobrevivió.

Actualmente y hasta la fecha que data el documento, la versión construida de la interfaz de usuario (solución del reto) es accesible a través de un navegador web y está disponible en un servidor de azure, aquí uno puede entrar a probar tanto el modelo como la interfaz de usuario: https://titaniceq6ml.azurewebsites.net/

Aqui algunas imagenes de la interfaz de usuario:

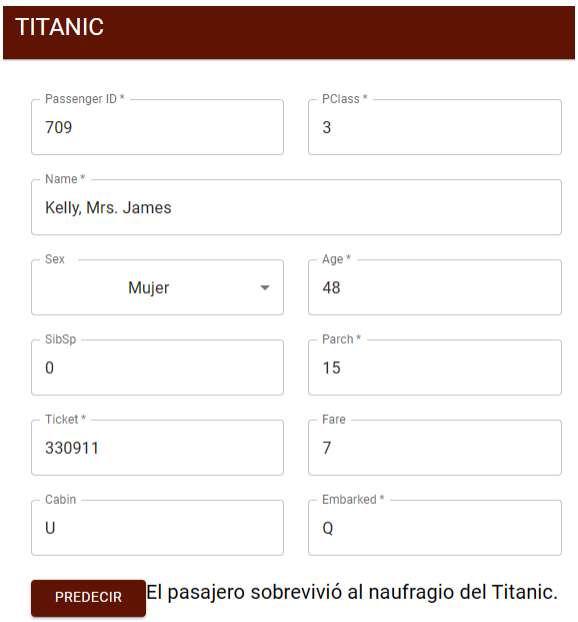

Figura 21: La interfaz con el resultado de sobrevivió en 1

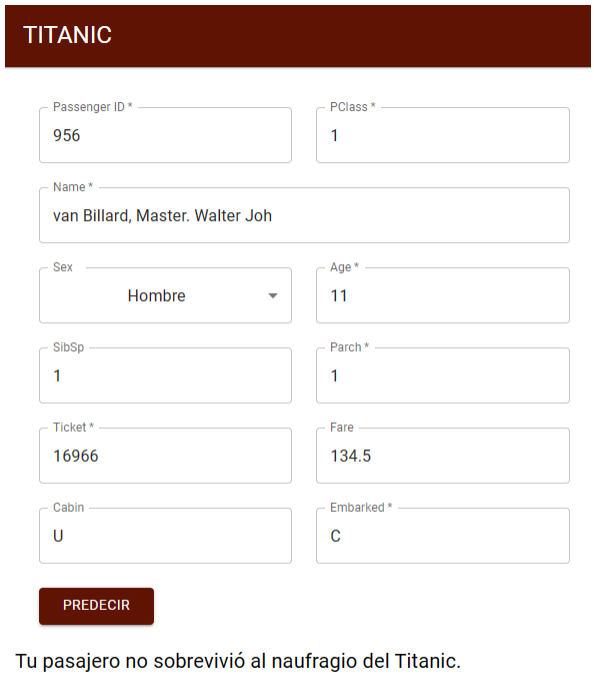

Figura 22: La interfaz con el resultado de sobrevivió en 0