Parte uno: 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 (Jupiter 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.

Árbol de decisiones

Los árboles de decisiones son modelos predictivos que mapean características que pueden ser relevantes para poder predecir un resultado, en el contexto del reto "Titanic - Machine Learning from Disaster" se busca poder identificar las variables que nos puedan ayudar a predecir quienes sobrevivieron y quienes no.

Primeramente decidimos usar los datos anteriormente ya limpios para poder usarlos en el diseño del modelo, los cuales decidimos transformar 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 no se consideró que fuera tan extrema que afecta a este modelo.

Una vez considerado ello y que tenemos un set de entrenamiento y un set de validación empezamos a experimentar con varias configuraciones del modelo del árbol de decisiones, en el cual se probaron múltiples valores de profundidad y como estos podrán afectar al modelo, una vez seleccionado el valor óptimo de 4 para nuestro modelo lo entrenamos con los datos y se llegó a que el modelo realizado en base al árbol de decisiones tiene una precisión aproximadamente del 80% para predecir quienes sobrevivieron al naufragio

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.

Optimizamos los hiper parámetros del RandomForest usando Optuna y fijamos el f1_score como la métrica principal, ya que se adapta bien a nuestros objetivos de clasificación.

Después de entrenar, evaluamos las predicciones y las comparamos con otros modelos probados. Esto nos permitió ver cómo el RandomForest se desempeñaba en comparación con las alternativas.

En resumen, manejamos el desbalance, entrenamos con RandomForest, optimizamos con Optuna y evaluamos con f1_score, obteniendo una perspectiva clara de cómo nuestro modelo aborda el desafío del Titanic.

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.

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 posible sobreajuste en el modelo de Random Forest.

En el proceso de entrenamiento del nuevo modelo, realizamos pruebas con distintos tamaños de división de datos para el conjunto de prueba. Además, empleamos Optuna para optimizar los hiperparámetros, maximizando el f1_score, que elegimos como métrica objetivo.

Tras analizar los resultados obtenidos, identificamos las características más relevantes y procedimos a eliminar aquellas con una importancia menor a 0.01. Este paso nos permitió reducir el ruido en el modelo.

Una vez completadas estas etapas, repetimos el proceso y logramos una mejora notable del 0.05 en la precisión del modelo en comparación con iteraciones anteriores.

Finalmente, para validar nuestro enfoque, evaluamos y realizamos predicciones en el conjunto de pruebas proporcionado por Kaggle. Esto nos permitió realizar una comparación con los otros modelos implementados.

Justificación del uso de múltiples modelos

Se decidió elegir varios modelos que acorde a nosotros creemos que pueden resolver de manera óptima el reto de Titanic, esto con el objetivo de ver cómo se comporta cada uno y analizar cuál modelo puede resultar conveniente para poder resolver este último de una manera satisfactoria y con el modelo que resulte más apto a las necesidades del reto.

Evaluación y Refinamiento del modelo

Hemos dividido nuestros datos en conjuntos de entrenamiento y prueba para evaluar y mejorar nuestro modelo, utilizando una proporción de 0.1 para el conjunto de prueba y el resto para el entrenamiento. Luego de experimentar con diversas proporciones de esta división, hemos determinado que esta configuración nos proporciona los mejores resultados.

Para evaluar nuestros modelos, hemos optado por utilizar la métrica F1-score. Esta elección se debe a la presencia de cierto desequilibrio entre las clases, lo que hace que el F1-score sea la métrica más adecuada para la evaluación de nuestro modelo.

Después de llevar a cabo una serie de análisis utilizando múltiples modelos de aprendizaje automático en nuestro conjunto de datos, hemos obtenido resultados prometedores en términos de precisión, con valores superiores al 80%. Sin embargo, es importante destacar que algunos modelos han demostrado un rendimiento superior a otros.

En este contexto, hemos decidido seleccionar el modelo XGBoost (XGB) de entre todos los modelos evaluados. Este modelo en particular ha destacado al lograr una precisión del 85.55%, superando a otros modelos como el árbol de decisiones (decision tree) y el bosque aleatorio (random forest). Cabe mencionar que este alto nivel de precisión nos brinda confianza en su capacidad para realizar predicciones precisas en datos del mundo real.

Además, tenemos la intención de optimizar aún más el rendimiento del modelo XGBoost con el objetivo de mejorar su eficiencia y su capacidad predictiva, sin incurrir en problemas de sobreajuste (overfitting) ni subajuste (underfitting). Esta optimización nos permitirá aprovechar al máximo el potencial de este modelo en futuras aplicaciones y escenarios.

Para optimizar los hiper parámetros, hemos empleado un algoritmo de optimización bayesiana llamado Optuna. Le proporcionamos un rango de valores para los hiper parámetros y realizamos 100 iteraciones, ajustando además las proporciones en la división de datos y eliminando variables que no contribuyeron significativamente al modelo, basándonos en la importancia de características (feature importance).

Antes de realizar la optimización con Optuna, nuestro modelo XGBoost Classifier tenía el siguiente rendimiento:

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.89	0.90	64
1	0.74	0.77	0.75	26
accuracy			0.86	90
macro avg	0.82	0.83	0.83	90
weighted avg	0.86	0.86	0.86	90

Después de afinar los hiper parámetros con Optuna, obtuvimos los siguientes resultados:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.87      0.90        54
           1       0.82      0.89      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
```

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.