THE BRIDGE DIGITAL TALENT ACCELERATOR

Bootcamp Data Science
Blanca Novella Serrano
2024

Machine Learning Supervisado

Predicción del nivel de engagement en juegos online

ÍNDICE

01 Introducción

En este proyecto enfocado a Machine Learning Supervisado, entrenaremos distintos modelos y compararemos sus resultados para determinar cuál de ellos obtiene mejores resultados en la predicción de el nivel de engagement en videojuegos online.

El dataset utilizado para el entrenamiento es el siguiente:
Rabie El Kharoua. (2024).
🎮 Predict Online Gaming Behavior Dataset [Data set].
Kaggle. https://doi.org/10.34740/KAGGLE/DSV/8742674.
https://www.kaggle.com/datasets/rabieelkharoua/predict-online-gaming-behavior-dataset

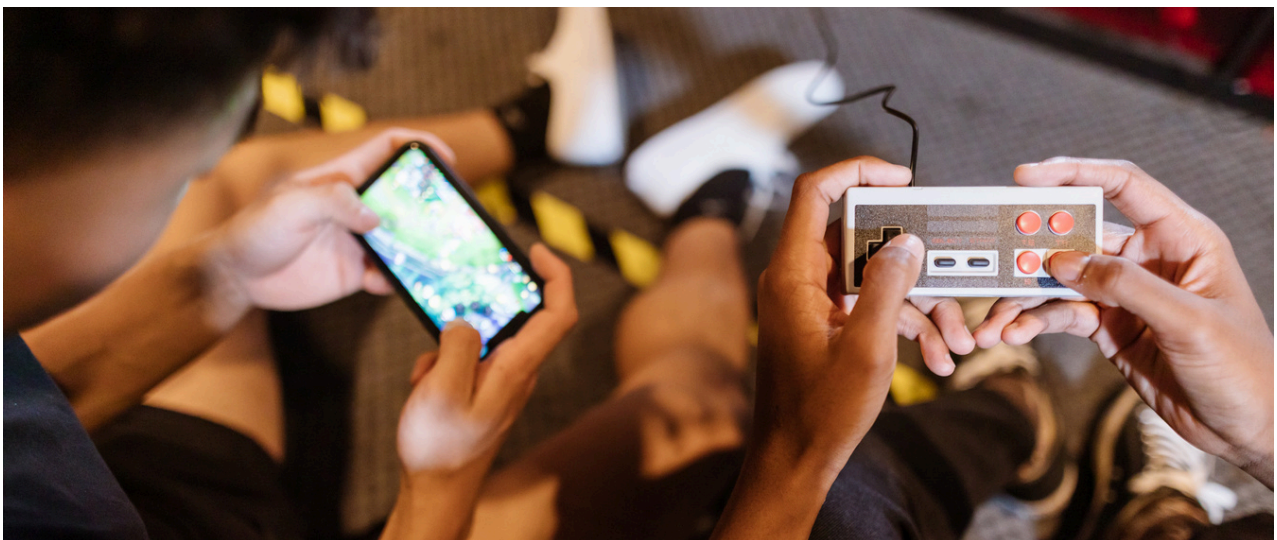
01.A Contexto

_Objetivo:
Predecir el **nivel de engagement** que tendrá un jugador teniendo en cuenta una serie de características dadas como las horas de juego empleadas, las sesiones iniciadas, tipo de juego, etc.

_Metodología:
Nos encontramos ante un **problema de clasificación** para el cual emplearemos algoritmos de clasificación no supervisada.

_Métricas:
Nos centraremos en el balanced accuracy y recall medio dada las características del target a predecir.

_Impacto de negocio:
- Ayudar a mejorar estrategias de retención de usuarios.
- Mejorar las estrategias de monetización.
- Ayudar en la toma de decisiones para el diseño de videojuegos.

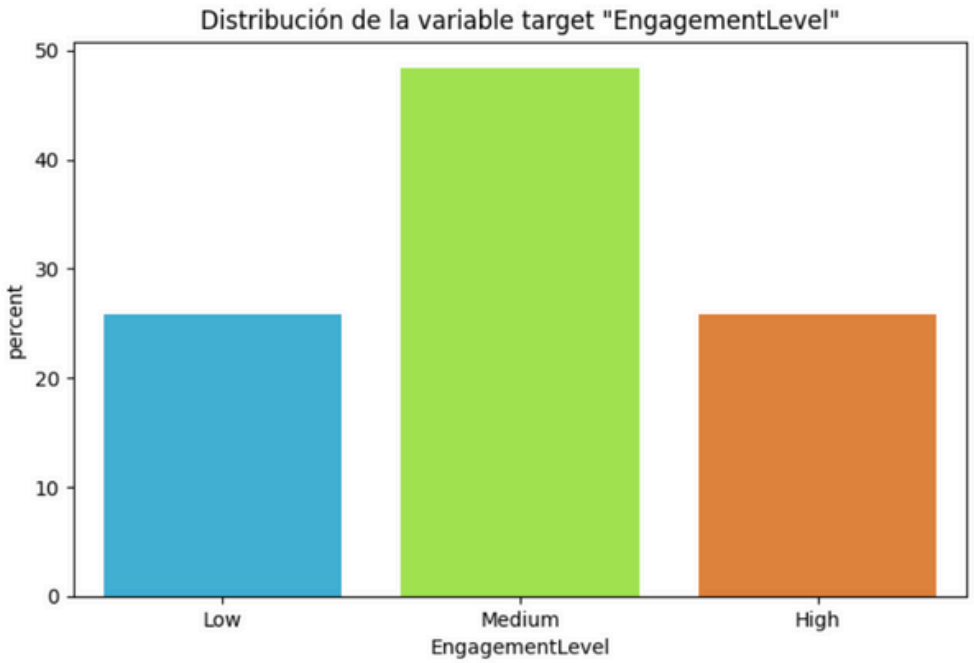


01.B Compresión de los datos (EDA)

Se parte del dataset (online_gaming_behavior_dataset.csv) que contiene una muestra de 40034 entradas y 13 columnas. Aparentemente el dataset ya ha sido tratado y acondicionado por el autor, aunque en un punto más avanzado del proyecto realizaremos transformación de los datos y nos desharemos de la columna 'PlayerID'.

Tras dividir el set de datos en train y test, visualizaremos nuestra variable **target ,** 'EngagementLevel' etiquetada en 3 clases (Medium, High y Low). Se encuentra desbalanceada, representando un 50% de las muestras la clase Medium y el otro 50% repartido equitativamente entre High y Low.

Fig.1 Distribución del target.



Por otro lado, para entender cómo se comportan el resto de variables en relación al target, realizamos una clasificación entre variables numéricas y categóricas de las cuales veremos su distribución y matriz de correlación en un mapa de calor en el caso del las variables numéricas.

En general, las variables presentan una distribución uniforme, no hay presencia de anomalías.

_Variables Numéricas:

`Age, PlayerLevel y AchievementsUnlocked`
Son variables continuas que bien podrían tratarse como categóricas si las agrupamos en intervalos. Por ejemplo, grupos de edad (menores de 20, etc) o niveles de jugador (alto, medio, bajo).
Pero en nuestro caso, probaremos a entrenar el modelo dejándolas tal cual porque los valores parecen estar bien distribuidos.

`SessionsPerWeek y AvgSessionsDurationMinutes`

Son variables que visualmente prometen ya que son las que más variabilidad presentan en función del target. (figura 3)

`PlaytimeHours`

Puede ser interesante aunque gráficamente no presente mucha variabilidad según el target.

_Variables Categóricas:

`InGamePurchases`

De naturaleza binaria, ya codificada como tal. También presenta una gran desigualdad entre sus categorías, siendo la más prominente 0 (no realiza compras).

`Gender, GameDifficulty`

Presentan categorías con una distribución algo más irregular que con respecto al resto de variables.

Location, GameGenre

Son las más regulares, sobre todo GameGenre cuya muestras se repartes de forma equitativa para cada clase. Por otro lado, Location, se reparte de forma proporcional aunque existen clases minoritarias.

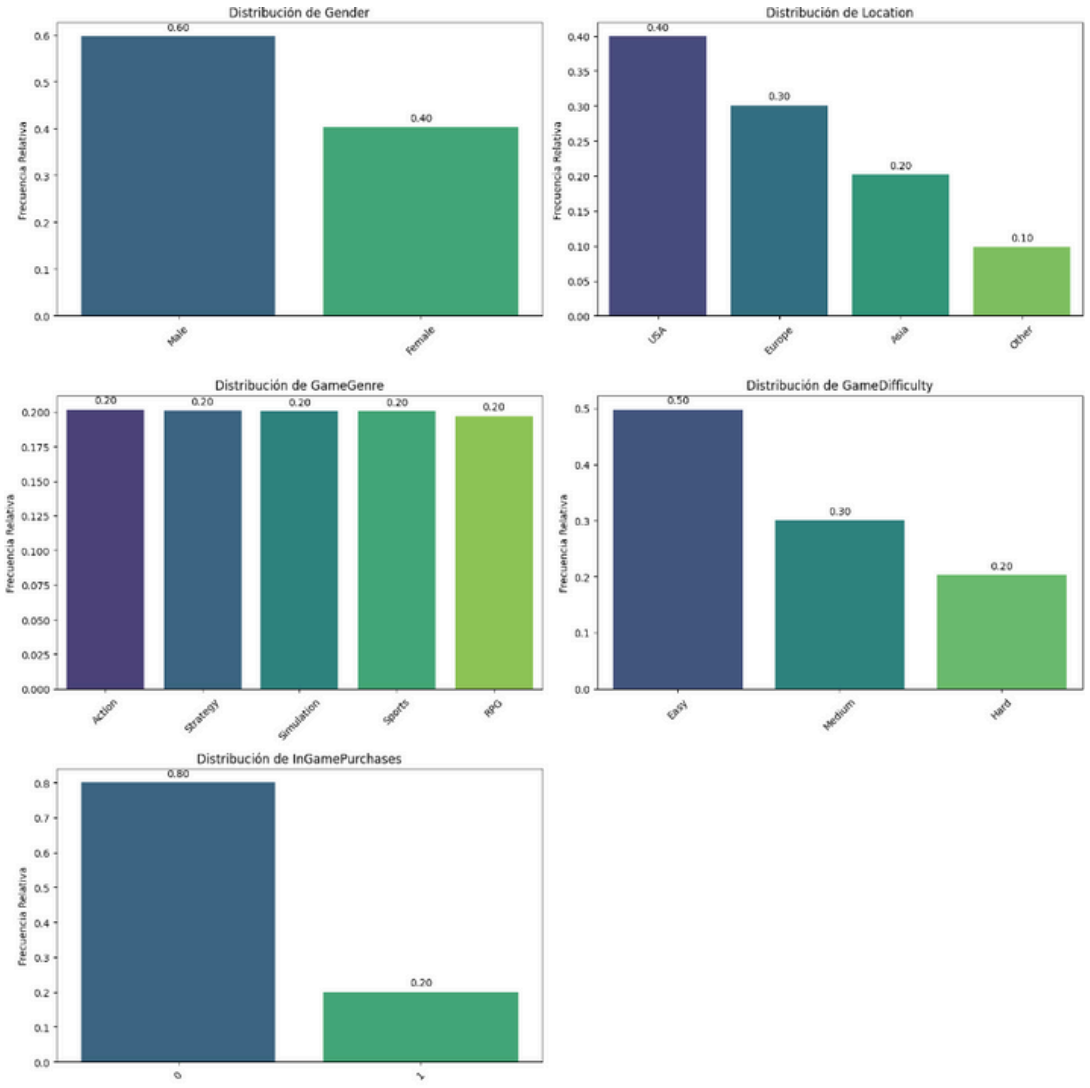


Fig.2 Distribución de variables categóricas.

Fig.3 Distribución de variables numéricas con respecto al target

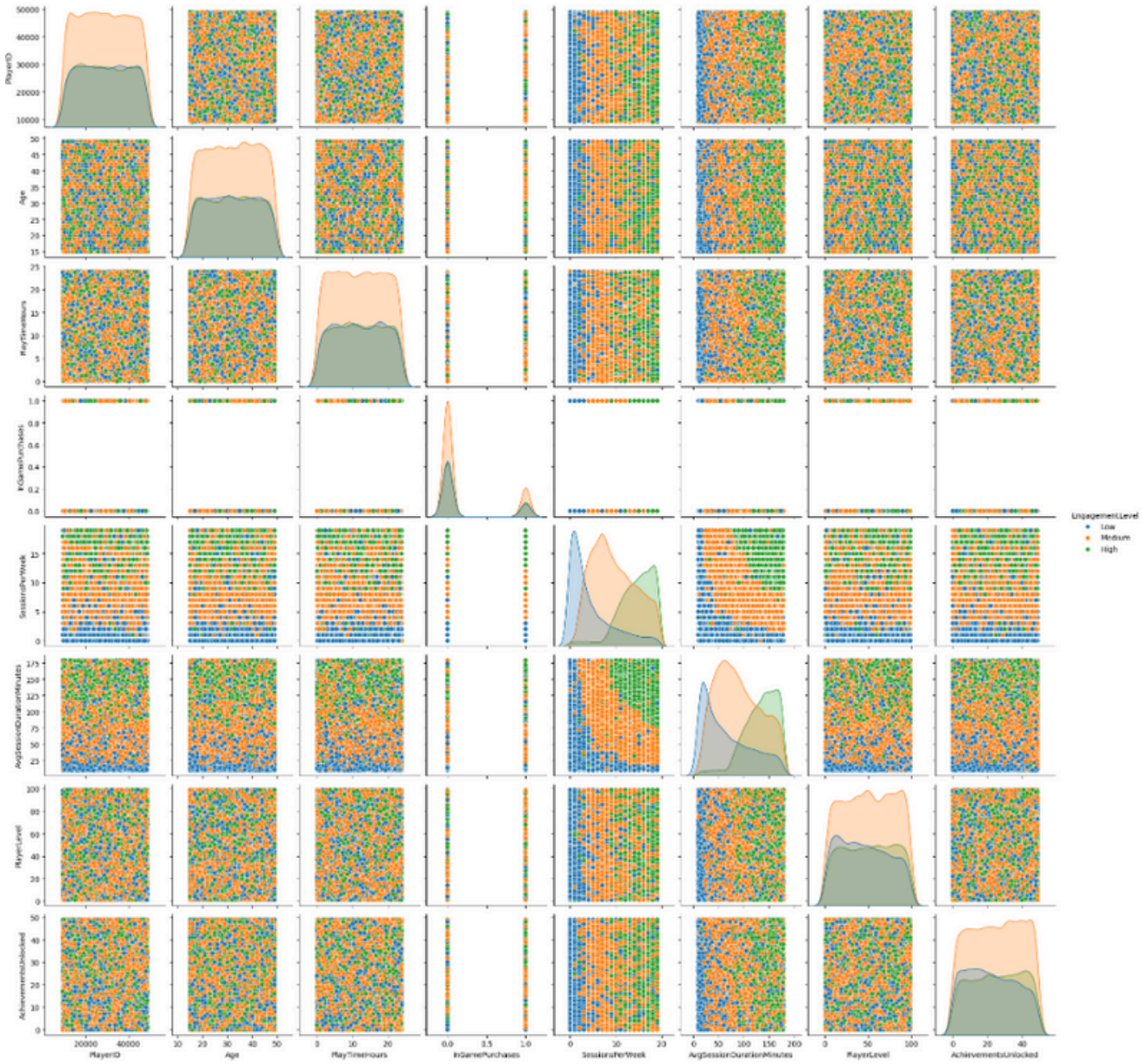


Fig.4 Distribución de variables numéricas

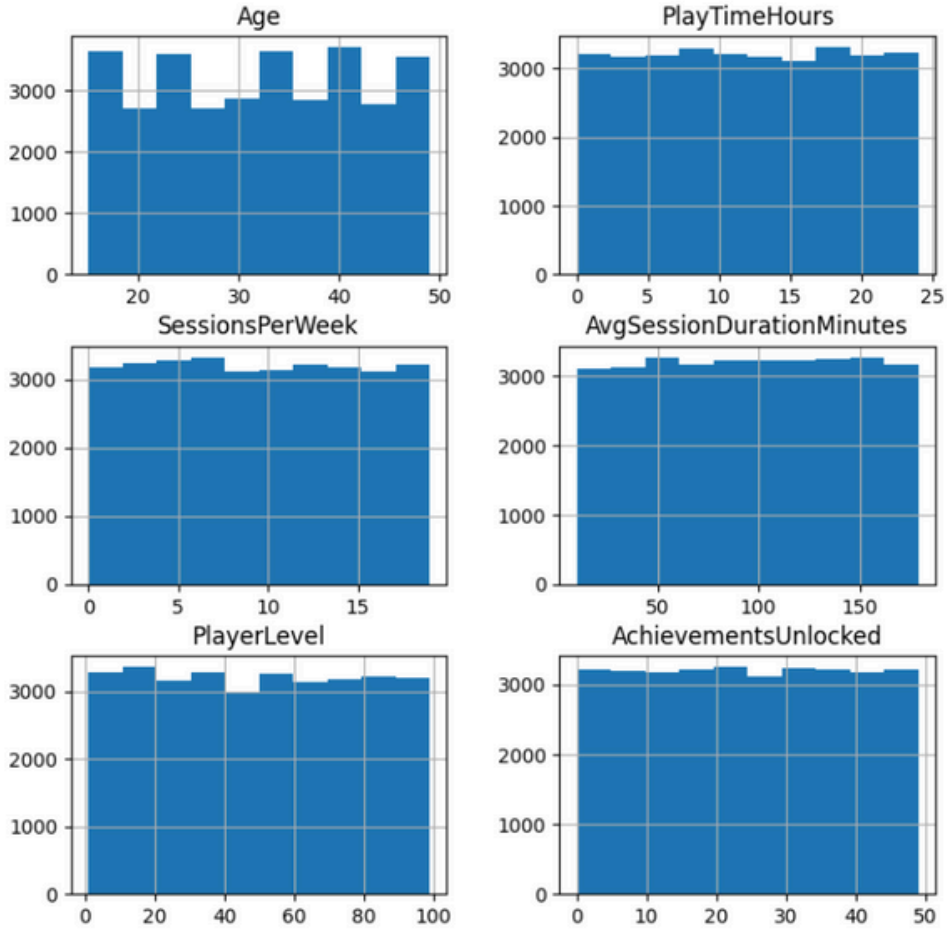
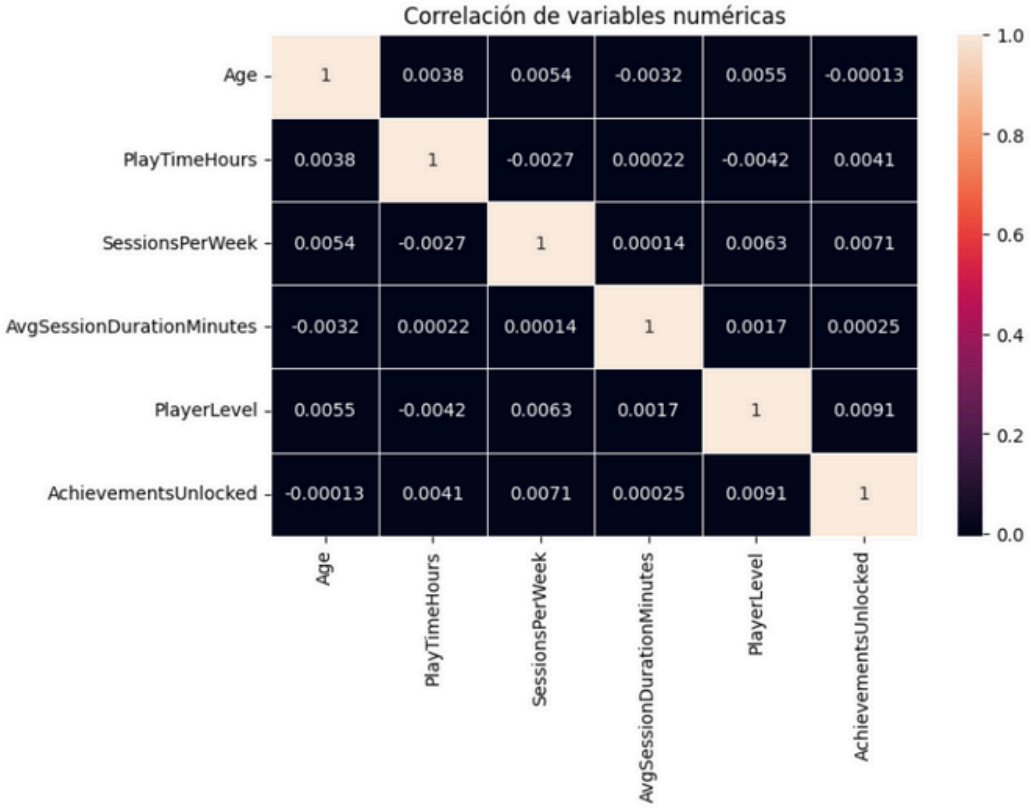


Fig.5 Mapa de calor. Matriz de correlación



02 Procesamiento de datos

Si observamos los datos, podemos afirmar que las variables numéricas en general presentan una distribución uniforme y no hay presencia de valores atípicos (outliers). Sin embargo tenemos valores en diferentes escalas, por ejemplo edad y frecuencia de juego. Para escalar los datos y no distorsionar la distribución, usaremos **MinMaxScaler().**

En cuanto a las variables categóricas, las procesaremos todas con **OneHotEncoder(),** ya que ninguna tiene un gran número de clases distintas y podremos binarizarlas sin generar un gran número de columnas por clase.

Realizaremos el preprocesado a través de un pipeline.

03 Baseline

Generaremos un pipeline con pre procesado por cada uno de los modelos cuyas métricas queremos poner a prueba. Realizaremos con **cross_validate** ya que queremos ver tanto métricas de entrenamiento y validación como los tiempos de ejecución.

Por otro lado, realizaremos la validación cruzada con **StratifiedKFold()** para que se mantenga la proporción de clases en cada fold dado que contamos con un target desbalanceado.

Probaremos los siguientes modelos; **DecisionTreeClassifier, RandomForestClassifier, LGBMClassifier, XGBClassifier, LogisticRegressor y KNN.** El objetivo es comparar a priori, el desempeño de modelos más potentes con respecto a modelos que no ofrecen tantas prestaciones pero que pueden tener un buen rendimiento como es el caso de LogisticRegressor.

Como resultado llama la atención que obtenemos de entrada muy buenas métricas de validación y entrenamiento, sobre todo en los modelos basados en árboles. Lo que sugiere que sin ningún tipo de penalización y con una distribución tan uniforme de los datos los modelos pueden estar sobre ajustados.

Mostramos a continuación las curvas de aprendizaje de cada Baseline:

Fig.6 Baseline DecisionTreeClassifier

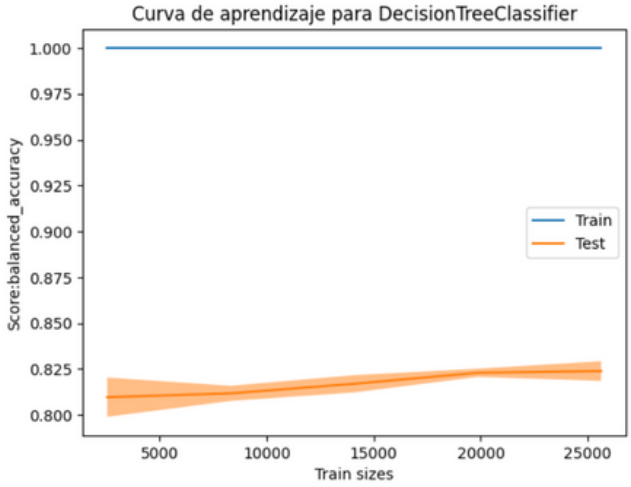


Fig.7 Baseline RandomForestClassifier

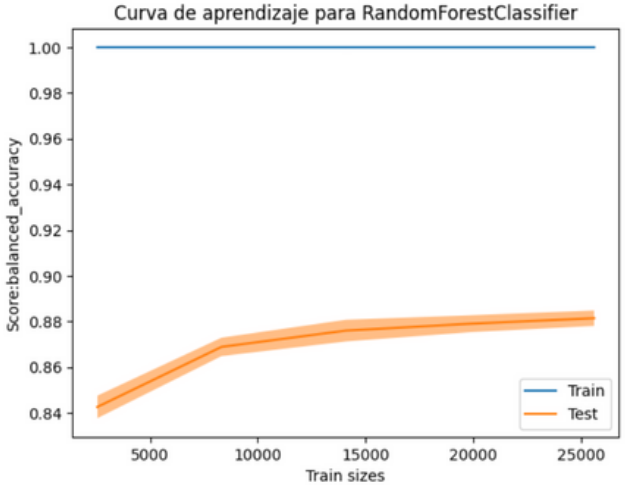


Fig.8 Baseline DecisionTreeClassifier

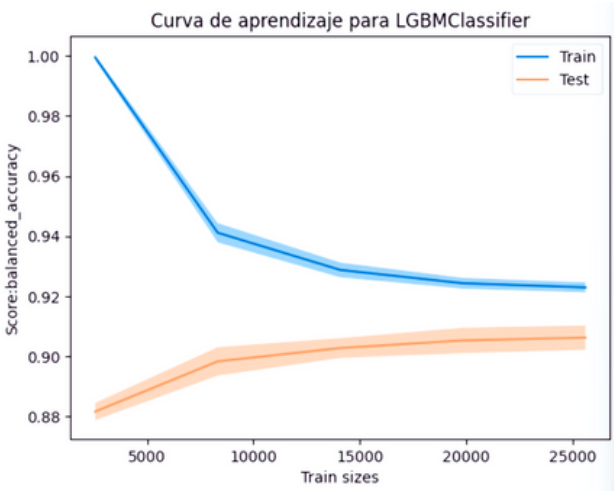


Fig.9 Baseline RandomForestClassifier

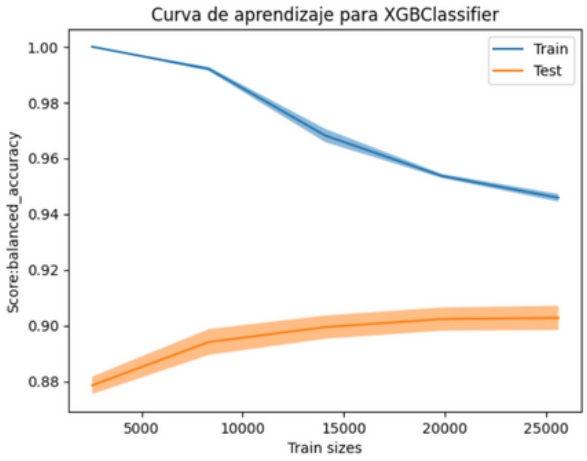


Fig.10 Baseline CatBoostClassifier

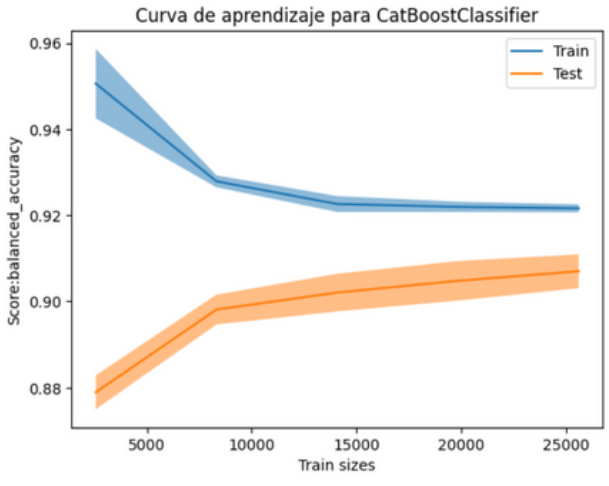
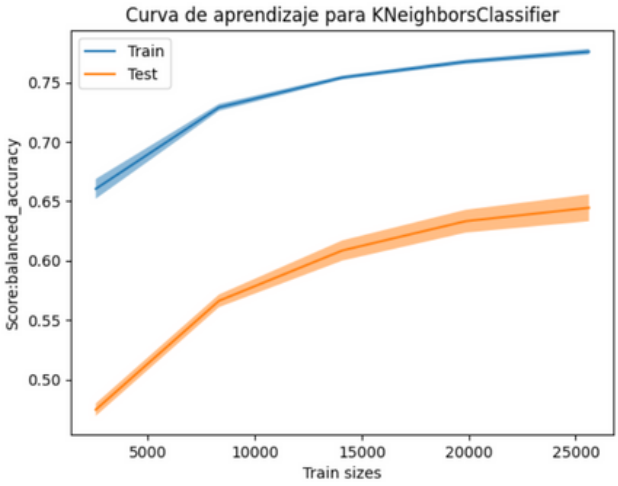


Fig.11 Baseline KNearestNeighbors



04 Comparativa de modelos: optimización, curvas de aprendizaje y evaluación

A partir de los resultados del baseline y teniendo en cuenta tanto métricas como tiempos de entrenamiento, se decide optimizar los siguientes modelos:

- LGBMClassifier: Como mejor modelo resultado de la validación y menor tiempo de entrenamiento, probaremos a optimizar hiperparámetros con GridSearch y seleccionar las features más relevantes para entrenarlo y compararlo con su baseline.
- LogisticRegression : Probaremos a aplicar penalizaciones y a restringir el número de features en el entrenamiento para comprobar si mejora con respecto al baseline.
- RandomForestClassifier: Probaremos a limitar el modelo y a aplicar penalizaciones para comprobar si podemos evitar el overfitting con este modelo y comparar resultados con el resto de modelos seleccionados.

Además probaremos también a entrenar con una **red neuronal simple (MLP)**, para comparar resultados y tiempo de entrenamiento.

04.A LGBMClassifier

Con el objetivo de optimizar los resultados, hemos entrenado dos modelos con este algoritmo:

- **Primero modelo:** recurrimos a la selección de features a través del método de selección del propio modelo. Por otro lado, incluimos en el pipeline de entrenamiento, técnicas de muestreo (RandomOverSampler() y realizamos una validación cruzada con GridSearchCV() para determinar los mejores parámetros de un grid establecido. Como resultado obtenemos una curva de aprendizaje que puede tender al sobreajuste.

- **Segundo modelo:** con el objetivo de corregir el posible sobreajuste del modelo, entrenamos con todas las features del dataset y sólo con el parámetro de balanceo del propio modelo (class_weight= 'balanced'). Por otro lado, restringimos la profundidad y el numero de hojas por árbol. Obtenemos como resultado una curva de aprendizaje mejor ajustada tanto en train como en validación. En cuanto a las métricas, recurrimos a la matriz de confusión para ver de forma gráfica los resultados, pudiendo observar que el modelo generaliza mejor sin sacrificar en calidad de predicción.

- **Conclusiones:**
  - El modelo generaliza mejor sin feature importance ni sampling. Además es necesario limitar el grid en cuanto a numero de árboles, profundidad y hojas, estas limitaciones no afectan a la calidad de los resultados en la evaluación contra test.
  - Las probabilidad de acierto del modelo optimizado para controlar el sobreajuste es equitativa en cada clase. En proporción no existe una clase cuya etiqueta se otorgue con mayor facilidad.

Fig.12 Curva de aprendizaje Primer modelo LGBM

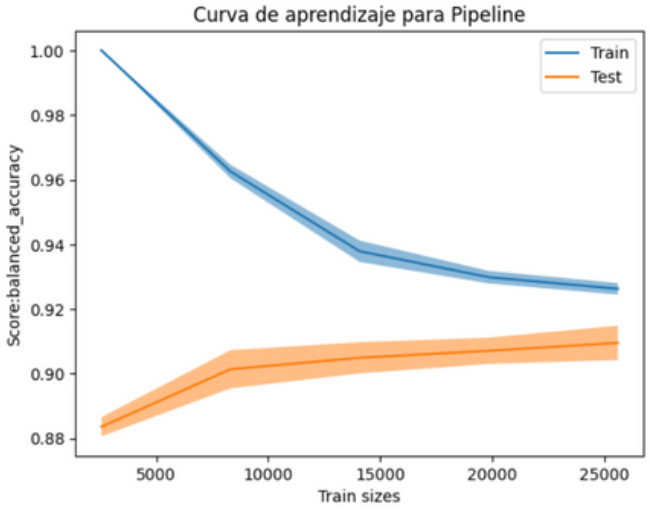


Fig.13 Matriz de Confusión Primer modelo LGBM

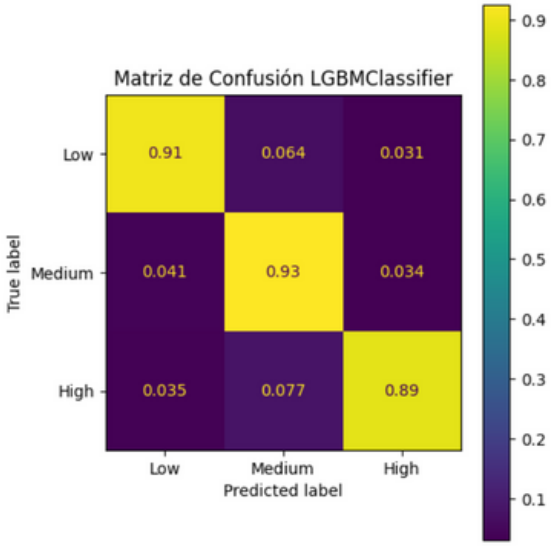


Fig.14 Curva de aprendizaje Segundo modelo LGBM

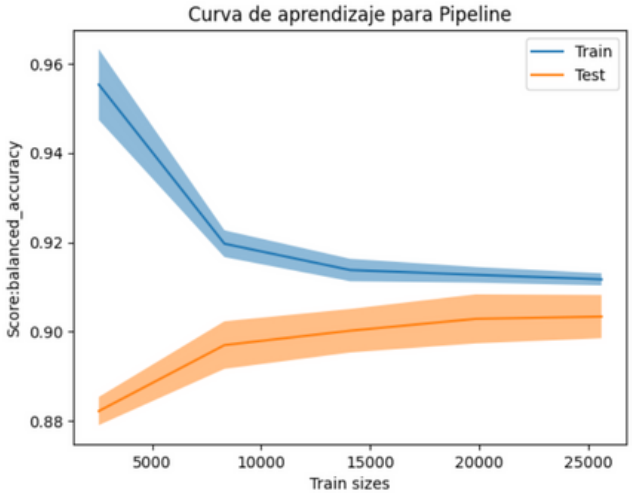


Fig.15 Matriz de Confusión Segundo modelo LGBM

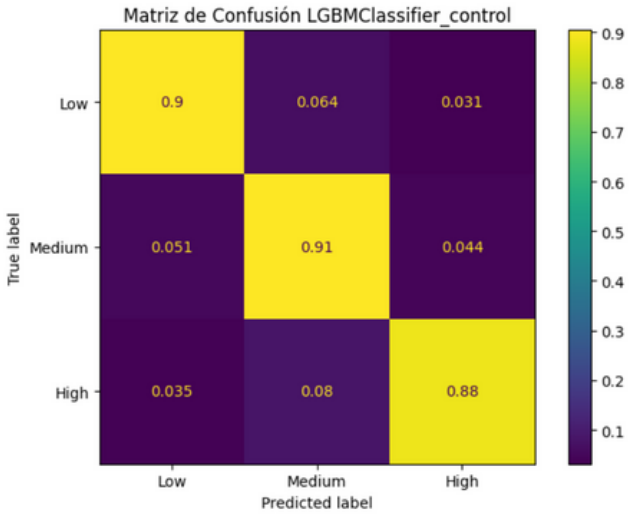
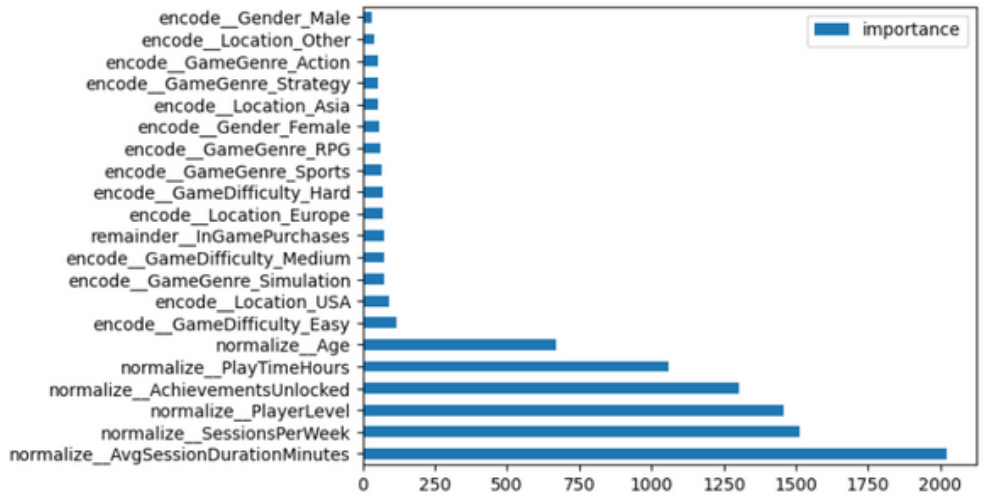


Fig.16 Feature Importance LGBMClassifier



04.B LogisticRegressor

Como partida, queremos comprobar si aplicando feature importance o penalizaciones el modelo es capaz de realizar mejores predicciones.

Desplegamos un grid donde incluimos regularización de tipo L1 y L2 para equilibrar los pesos. Sin embargo como resultado de la validación cruzada, el modelo no elige ningún tipo de regularización.

En conclusión, es un modelo mucho más sencillo que los basados en árboles que presenta sus limitaciones, pero que a pesar de ello, también realiza muy buenas predicciones sin sobreajustarse.
En cuanto a las predicciones, cabe resaltar que comete más errores en la clase mayoritaria.

Fig.17 Coeficientes Logisticregressor

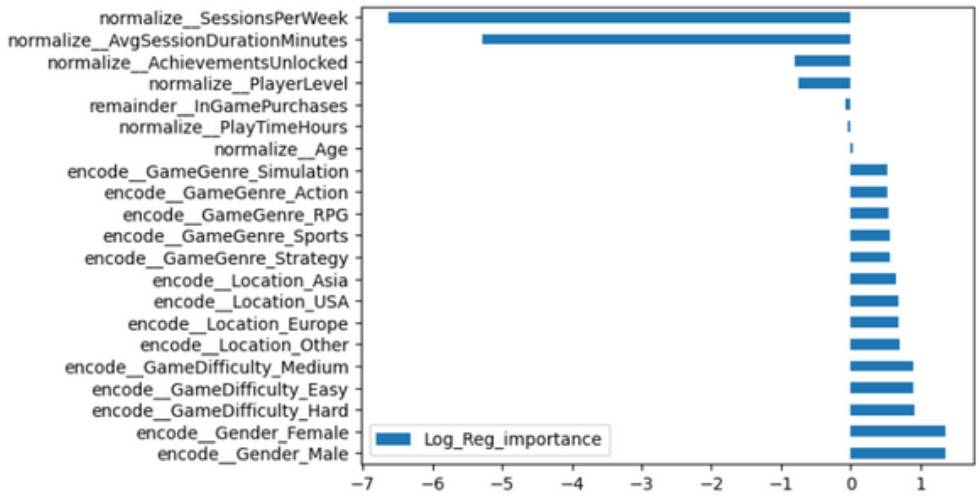


Fig.18 Curva de aprendizaje LogisticregRegressor Optimizado

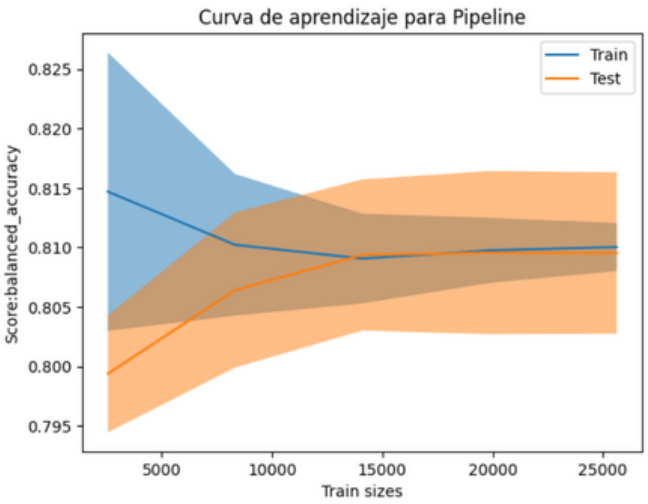
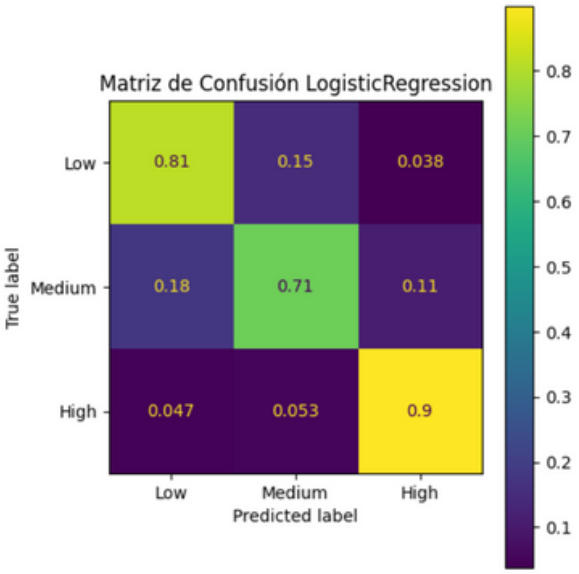


Fig.19 Matriz de Confusión LogisticregRegressor Opt.



04.C RandomForestClassifier

Es uno de de los modelos que presentaba un claro sobreajuste en el tras validación cruzada inicial. El objetivo es ajustar el modelo para encontrar el punto de equilibrio entre aprendizaje y predicciones, sin tender al sobre ajuste. Para ello hemos entrenado tres modelos:

- En la **primera** optimización, empleo **RandomOverSampler** como técnica de muestreo, estructurando un grid con cierta profundidad de árboles y nodos además de entrenar el modelo sólo con aquellas variables de mayor peso. Sin embargo, no hemos conseguido que generalice mejor y el tiempo de ejecución durante el entrenamiento del GridSearch ha sido bastante largo en comparación con otros.

- En la **segunda** optimización, el objetivo es 'dificultar' el aprendizaje. Pruebo a eliminar el sampling y que el modelo balancee las clases sólo tomando el hiperparámetro **class_weight**, lo que ha mejorado notablemente el rendimiento del entrenamiento. Por otro lado, hemos restringido drásticamente la profundidad de árboles y sus nodos además de entrenarlo con todas las variables. De esta forma sí hemos logrado conseguir que el modelo no tienda al sobreajuste a pesar de bajar sus métricas de predicción.

- En la **tercera** optimización, el objetivo ha sido comprobar si añadiendo profundidad y nodos, podemos mejorar las métricas anteriores o el modelo comienza a sobreajustarse. En este caso, a partir de 50 árboles el modelo comienza a presentar sobreajuste como hemos podido comprobar en las curvas de aprendizaje.

Por tanto, nos quedaremos con la segunda optimización como mejor opción para RandomForestClassifier.

Fig.18 Curva de aprendizaje RandomForest _1

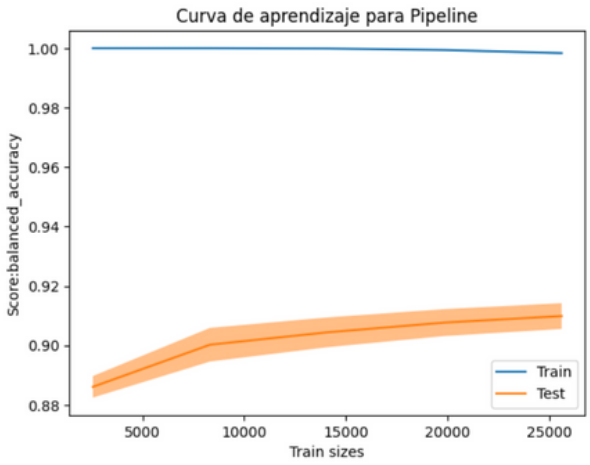


Fig.19 Curva de aprendizaje RandomForest _3

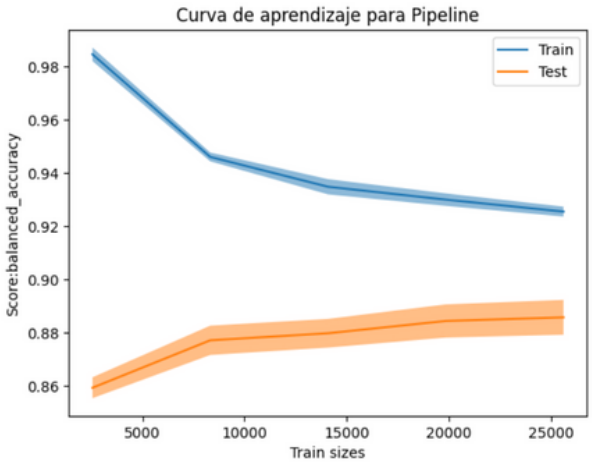


Fig.20 Curva de aprendizaje RandomForest _2

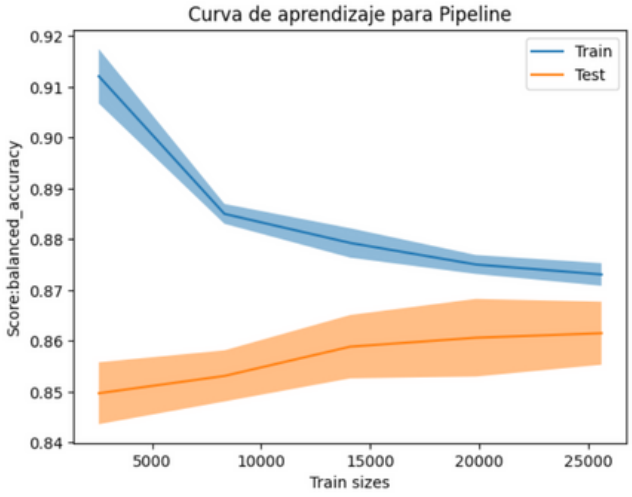
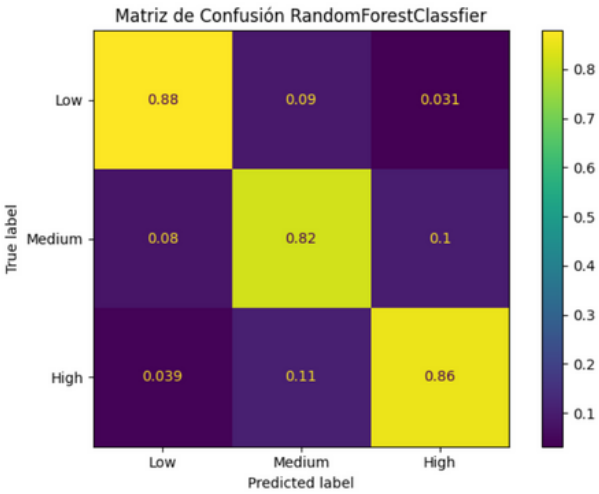


Fig.21 Matriz de Confusión RandomForest _2



04.C Red Neuronal Simple (MLP)

Para probar con deep learning y comprobar si existen ventajas en el entrenamiento con un perceptrón multicapa con respecto al modelo de machine learning con mejor métrica, en este caso LGBMClassifier, pasamos a diseñar una red neuronal de una sola capa oculta.

Para solventar el problema de desbalanceo de clases, recurrimos a la función **compile_sample_weight** y **compile_class_weight** para que otorgue de forma autómatica pesos equitativos a las diferentes clases.

Los resultados han sido bastante satisfactorios, obteniendo casi el mismo score en predicciones y probabilidades de clasificación de forma correcta.

El tiempo de ejecución es mucho menor comparado con el de LGBM y no tiende al overfitting.

Fig.22 Arquitectura MLP

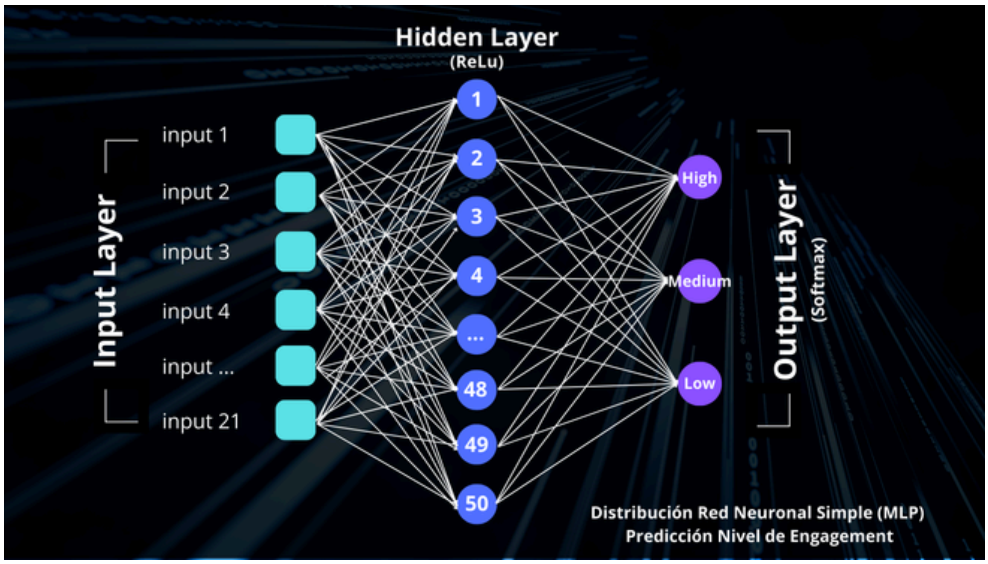


Fig.23 Curva Accuracy / Val_Accuracy

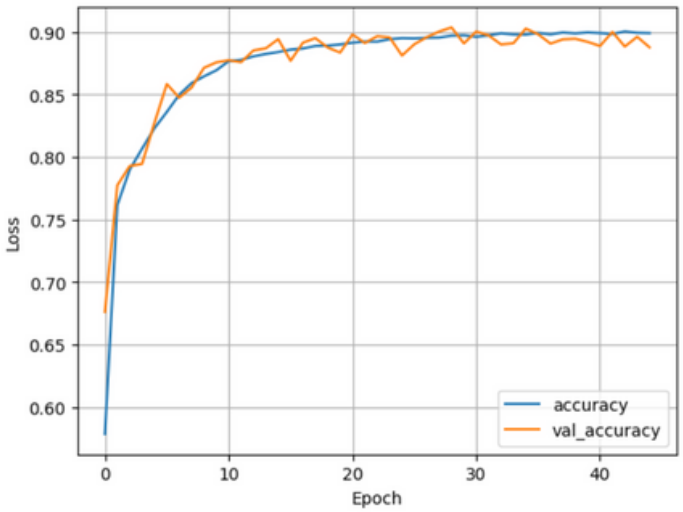


Fig.24 Curva Loss/ Val_loss

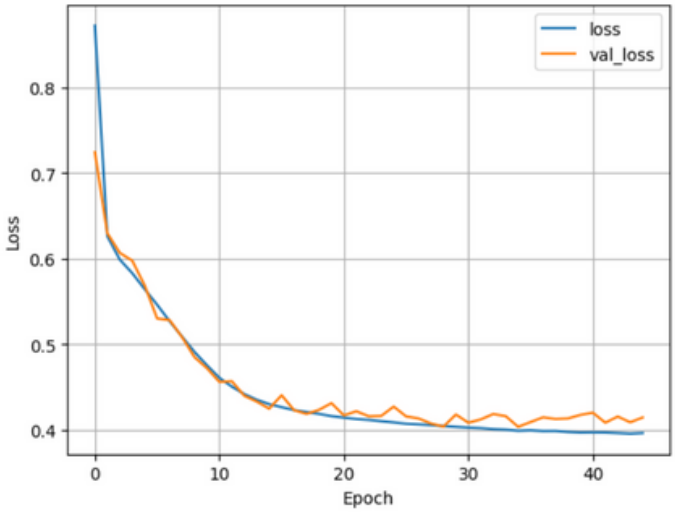
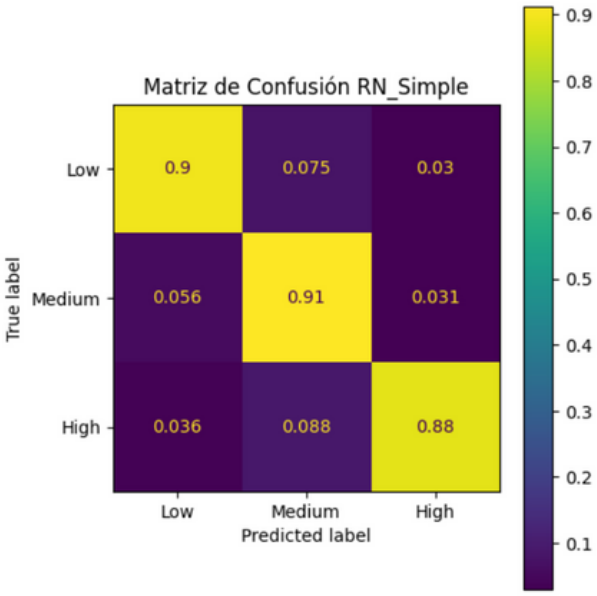


Fig.25 Matriz de Confusión MLP



05 Conclusiones

Tras optimizar y comparar distintos modelos, hemos enfrentado una dificultad presente en dos de los modelos seleccionados y es la gran capacidad de adaptación a los datos y generalización de los modelos basados en árboles. Es crucial limitar y probar distintos hiperparámetros para obtener mejores resultados.

Por otro lado, modelos simples como Regresión Logística, presentan buenos resultados con menor coste aunque tenga mayores limitaciones con respecto a otro modelos.

Otra de las particularidades de este proyecto han sido los datos, cuya distribución ha podido inclinar la balanza al sobreajuste.

En este caso en particular, entrenar con las features de mayor importancia para el modelo no ha sido beneficioso en términos de generalización del propio modelo ya que todos, han presentado mejores resultados entrenando con todas las características. A pesar de que podríamos pensar que aquellas con menor peso podían introducir ruido, han resultado beneficiosas en el entrenamiento.

Bootcamp Data Science
Blanca Novella Serrano
2024

Machine Learning Supervisado
Predicción del nivel de engagement en juegos online