Diego Velázquez Saldaña A01177877

Módulo 2 Análisis y Reporte sobre el desempeño del modelo

Introducción

En este proyecto, se implementa un modelo de Machine Learning utilizando el algoritmo XGBoost para resolver un problema de clasificación basado en datos deportivos. El objetivo principal es predecir el resultado de un partido de fútbol de la Premier League utilizando un conjunto de características como la posesión, los disparos, las faltas, entre otras. Para garantizar un análisis exhaustivo y robusto, se ha dividido el dataset en tres conjuntos: entrenamiento, prueba y validación. Esto permite evaluar el rendimiento del modelo en datos no observados y ajustar sus hiper parámetros de manera adecuada.

A lo largo de este reporte, se diagnosticarán los niveles de sesgo, varianza y el grado de ajuste del modelo (underfitting, fitting, overfitting). Además, se aplicarán técnicas de regularización y ajuste de parámetros para optimizar el desempeño, mejorando la precisión del modelo y minimizando errores de predicción.

Análisis y Limpieza de los Datos

Primeramente, es importante hacer un análisis de los datos que fueron encontrados. Estos datos fueron obtenidos de Kaggle, subidos por Mohamed Kardosha. Aquí la descripción y el tipo de dato que fueron encontrados:

Variable	Descripción	Tipo de dato
`day`	Día de la semana del partido	Categórico
`venue`	Sede del partido (casa o fuera)	Categórico
`result`	Resultado del partido (2: Victoria, 1: Empate, 0: Derrota)	Categórico (numérico)
`gf`	Goles a favor del equipo local	Numérico
`ga`	Goles en contra del equipo local	Numérico
`opponent`	Oponente del equipo local	Categórico
`xg`	Goles esperados para el equipo local	Numérico
`xga`	Goles esperados para el equipo contrario	Numérico
`poss`	Posesión del equipo local en porcentaje	Numérico
`sh`	Tiros del equipo local	Numérico
`sot`	Tiros a puerta del equipo local	Numérico
`dist`	Distancia media de los tiros del equipo local	Numérico
`fk`	Tiros libres del equipo local	Numérico
`pk`	Penaltis marcados por el equipo local	Numérico
`pka`	Penaltis intentados por el equipo local	Numérico
`season`	Año de la temporada del partido	Numérico
`team`	Equipo local	Categórico

Table 1: Descripción de las variables utilizadas en el modelo.

Antes de entrenar el modelo, es importante adaptar los datos a unos que un modelo pueda entender de mucho mejor manera, sobre todo las variables que son categóricas. Aún así, no fue el único cambio que se debió realizar, pues había más problemas con el conjunto de datos como tal. Entre los elementos clave:

- **Eliminación de variables irrelevantes:** Se descartaron columnas que, aunque formaban parte del dataset original, no eran pertinentes para el objetivo de predecir el resultado del partido. Algunas de estas variables incluyen:
    - Unnamed: 0, comp, round, attendance, match report, notes, y time: Estas columnas no aportan información relevante directa para predecir el resultado del partido.
    - captain, formation, referee: Aunque estas variables podrían influir en el resultado, se decidió no incluirlas en este análisis inicial para simplificar el modelo.
- **Transformación de fechas y temporadas:** Se observó una inconsistencia en las fechas y temporadas, por lo que se transformó la variable season para que coincidiera correctamente con el año de la temporada en curso. Se decidió que la temporada cambiaría al año siguiente si el partido se jugó después de agosto, reflejando la estructura típica de las temporadas deportivas.
- **Transformación de variables categóricas:** Las columnas categóricas como *venue*, *opponent*, *team*, y *result* fueron convertidas a variables numéricas utilizando la técnica de one-hot encoding. En el caso de *venue*, se descartó la categoría Away, ya que sería redundante. La variable result fue transformada en una variable categórica con valores 2 para victoria, 1 para empate, y 0 para derrota, lo que simplificó el problema en una tarea de clasificación multiclase.

- **Manejo de datos faltantes:** Se detectó una fila con un valor nulo en la variable de dist, esto debido a que dicho partido fue el único en el que el equipo de los datos no hizo ningún tiro en el partido, y por ende no existe un promedio de distancia de los disparos. Por ser solo un caso, se decidió eliminar la fila, ya que ponerla en cero podría confundir al modelo dada la naturaleza del dato.

Elección de Modelo

Para este proyecto, se decidió utilizar XGBoost como el algoritmo de clasificación. XGBoost fue elegido debido a varias razones que lo hacen adecuado para el tipo de problema presentado. Primeramente, XGBoost es conocido por su capacidad de manejar grandes conjuntos de datos de manera eficiente, lo cual es ideal cuando se tienen muchas variables y observaciones. En este caso, el dataset contiene múltiples características que pueden influir en el resultado, y XGBoost es capaz de procesarlas de manera efectiva gracias a su optimización interna.

Otro de los motivos por los que considero pertinente la elección de este modelo es debido a su capacidad para manejar variables categóricas. Aunque se hizo una transformación de las variables categóricas mediante técnicas como one-hot encoding con el fin de hacer los datos más sencillos, XGBoost puede lidiar adecuadamente con este tipo de variables, evitando el sobreajuste que podría ocurrir en otros modelos más simples.

Por último, la regularización es una de las características clave de XGBoost, tanto en términos L1 como L2, lo que ayuda a evitar el sobreajuste del modelo. En este proyecto, esta

característica es importante para controlar el comportamiento del modelo, especialmente cuando se tiene un conjunto de datos con múltiples características potencialmente correlacionadas.

Primeros Resultados

En esta etapa inicial, se optó por entrenar un modelo básico de XGBoost utilizando un conjunto de parámetros ajustados específicamente para la clasificación multiclase. El objetivo principal era evaluar cómo el modelo se comportaba sin demasiados ajustes ni optimizaciones. Los parámetros utilizados fueron los siguientes:

- *objective - multi*: El modelo está diseñado para una tarea de clasificación multiclase.
- *eval_metric - mlogloss*: Penaliza fuertemente predicciones incorrectas y se utiliza para medir la calidad del modelo durante el entrenamiento.
- *num_class - 3*: Indica que el modelo debe clasificar las observaciones en tres clases.
- *learning_rate - 0.01*: Se usó una tasa de aprendizaje baja para evitar que el modelo converja demasiado rápido y caiga en mínimos locales.
- *max_depth - 6*: La profundidad máxima de los árboles fue ajustada a 6. Un valor más alto podría causar sobreajuste, mientras que un valor más bajo podría llevar a un ajuste insuficiente.

```
# Parámetros de XGBoost para clasificación multiclase
params = {
    'objective': 'multi:softprob',  # Probabilidades para clasificación multiclase
    'eval_metric': 'mlogloss',      # Log Loss para multiclase
    'num_class': 3,                 # Tres clases: 0 (derrota), 1 (empate), 2 (victoria)
    'max_depth': 6,                 # Profundidad máxima de los árboles
    'learning_rate': 0.01           # Tasa de aprendizaje
}
```

El modelo fue entrenado utilizando la función xgb.train, que permite ajustar los hiperparámetros en el proceso de entrenamiento y definir cómo evaluar el rendimiento en cada iteración. Los puntos clave del código utilizado son:

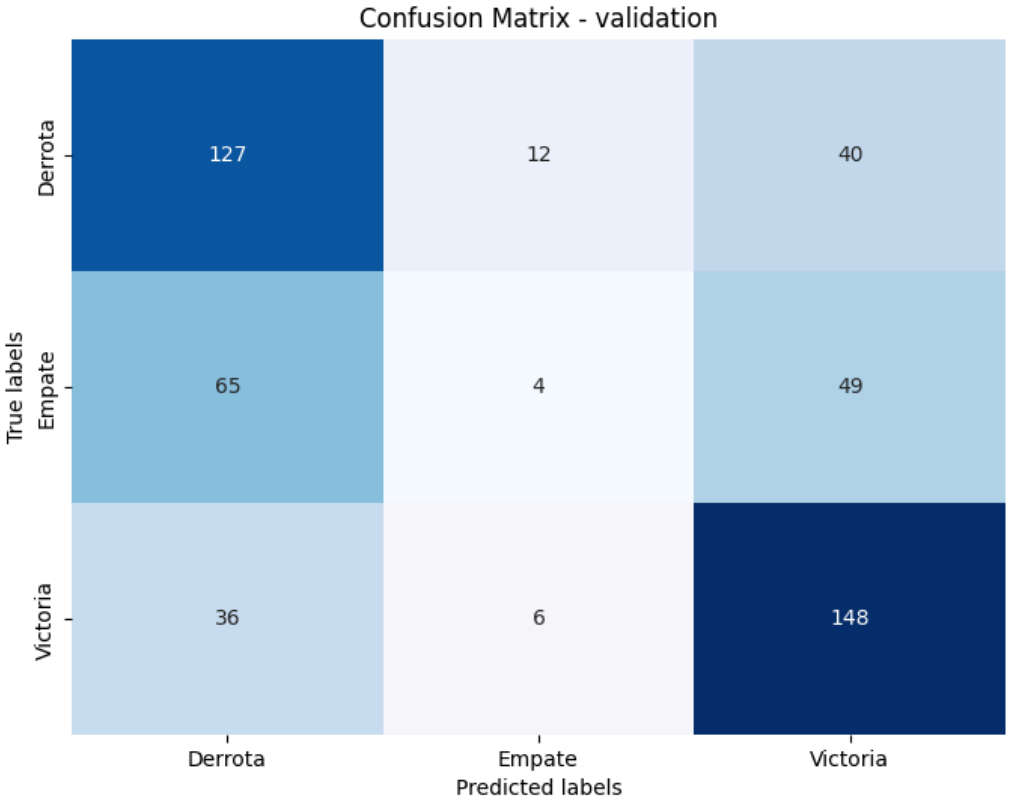
- *num_boost_round - 100*: Se realizaron 100 rondas de boosting. El boosting permite al modelo aprender de sus errores iterativamente, mejorando el ajuste progresivamente.
- *evals - [(train_dmatrix, 'train'), (val_dmatrix, 'eval')]*: Esta línea especifica que el rendimiento del modelo debe evaluarse tanto en los datos de entrenamiento como en el conjunto de validación en cada ronda.
- *evals_result*: Aquí se guardan los resultados de cada ronda para poder monitorear las métricas de evaluación en cada etapa del entrenamiento.

Por otra parte, las métricas de evaluación del modelo incluyen accuracy, precision, recall, y F1-score. Estas métricas son fundamentales para comprender el desempeño del modelo en cada clase.

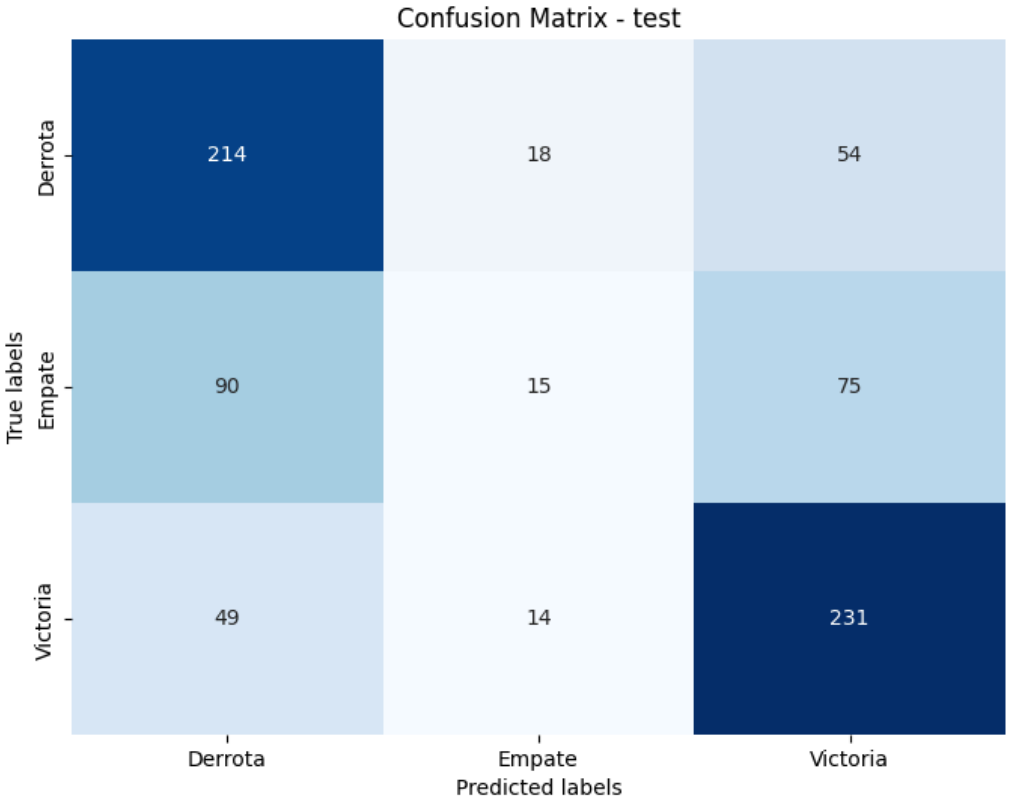
```
Validation Accuracy: 0.5728952772073922
Validation Confusion Matrix:
 [[127  12  40]
 [ 65   4  49]
 [ 36   6 148]]
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.71      0.62       179
           1       0.18      0.03      0.06       118
           2       0.62      0.78      0.69       190

    accuracy                           0.57       487
   macro avg       0.45      0.51      0.46       487
weighted avg       0.49      0.57      0.51       487

Test Accuracy: 0.6052631578947368
Test Confusion Matrix:
 [[214  18  54]
 [ 90  15  75]
 [ 49  14 231]]
Test Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.75      0.67       286
           1       0.32      0.08      0.13       180
           2       0.64      0.79      0.71       294

    accuracy                           0.61       760
   macro avg       0.52      0.54      0.50       760
weighted avg       0.55      0.61      0.56       760
```

Además, se incluyó la matriz de confusión obtenida también proporciona una visualización clara de las predicciones del modelo. Aquí se muestran las matrices para los datos de validación y de testeo:

Validación

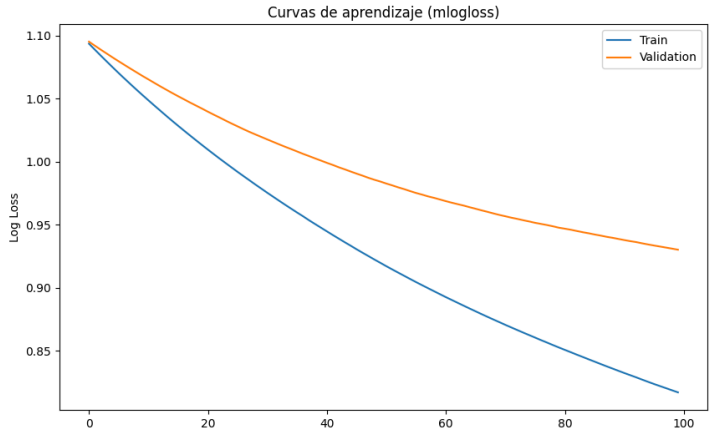

Confusion Matrix - validation
Derrota
Empate
Victoria
True labels
127
12
40
65
4
49
36
6
148
Derrota
Empate
Victoria
Predicted labels

Testeo



Como podemos observar en la matriz, el modelo tiene un buen desempeño prediciendo victorias y derrotas, pero presenta un área de oportunidad en los empates. Esto podría deberse a que los empates son menos frecuentes, lo que causa que el modelo los clasifique incorrectamente con más frecuencia.

Para finalizar, se incluyó una curva de aprendizaje para evaluar cómo mejora el modelo a medida que incrementa el número de iteraciones. La learning curve permite visualizar si el modelo está sobreajustando o subajustando, comparando el error de entrenamiento y validación.



En resumen, aunque los primeros resultados son prometedores, hay margen de mejora, particularmente en la clasificación de empates. Esto sugiere que en la siguiente etapa será clave ajustar parámetros o implementar técnicas de regularización.

Mejora del Modelo

En esta sección, se introdujeron nuevas variables de hiperparámetros en el modelo XGBoost con el objetivo de mejorar su desempeño y precisión. Las variables que se ajustaron son:

- *colsample_bytree*: Se refiere a la proporción de características (features) que serán utilizadas por cada árbol en el modelo. Un valor más bajo de este parámetro puede ayudar a reducir el sobreajuste, ya que los árboles individuales no utilizarán todas las características, lo que introduce más variabilidad en el proceso de entrenamiento.
- *subsample*: Controla la proporción de datos de entrenamiento que se utilizarán para cada árbol. Similar a colsample_bytree, un valor más bajo puede generar un modelo menos propenso al overfitting.
- *reg_alpha*: Este parámetro controla la regularización L1 (Lasso), que puede ayudar a reducir el número de características no importantes al aplicar un castigo a los coeficientes más pequeños, forzándolos a ser cero.
- *reg_lambda*: Controla la regularización L2 (Ridge), que ajusta el modelo penalizando grandes coeficientes. Es útil para controlar el sesgo y la varianza.

Estos ajustes fueron hechos para reducir el sobreajuste, mejorar la capacidad de generalización del modelo y permitir que el modelo sea más robusto al utilizar datos que no se han visto previamente.

Para encontrar los mejores valores de estos parámetros, y no solo a estos, sino también a los parametros presentados anteriormente, se utilizó la técnica de Grid Search. Esta técnica tiene como objetivo principal explorar de manera exhaustiva todas las combinaciones posibles de los hiperparámetros proporcionados y selecciona la mejor combinación en función del rendimiento del modelo en los datos de validación.

```
# Definir los parámetros que se quieren ajustar
param_grid = {
    'max_depth': [3, 6, 9],              # Profundidad del árbol
    'learning_rate': [0.01, 0.1, 0.2],   # Tasa de aprendizaje
    'n_estimators': [100, 200],          # Número de árboles
    'subsample': [0.8, 1.0],             # Proporción de muestras usadas en cada árbol
    'colsample_bytree': [0.8, 1.0],      # Proporción de features usadas por árbol
    'reg_alpha': [0, 0.1, 0.5],          # L1 regularización
    'reg_lambda': [0.1, 1.0, 10]         # L2 regularización
}
```

Tras la ejecución del Grid Search, se encontraron los mejores hiperparámetros para el modelo, lo que permitió optimizar aún más su rendimiento.

```
Best hyperparameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100, 'reg_alp
ha': 0, 'reg_lambda': 0.1, 'subsample': 1.0}
```

Resultados Finales

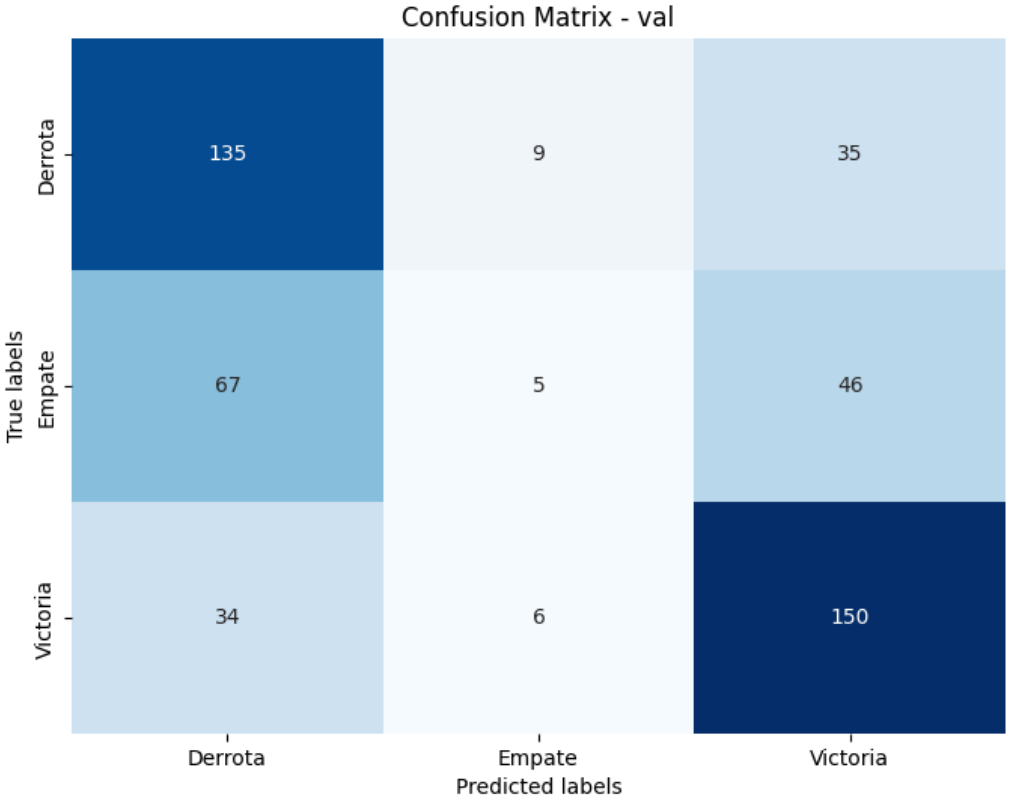
Después de aplicar los mejores hiperparámetros encontrados mediante Grid Search, se establecieron los parámetros finales del modelo. Estos ajustes permitieron mejorar ligeramente el rendimiento general, aunque los incrementos en precisión no fueron muy significativos.

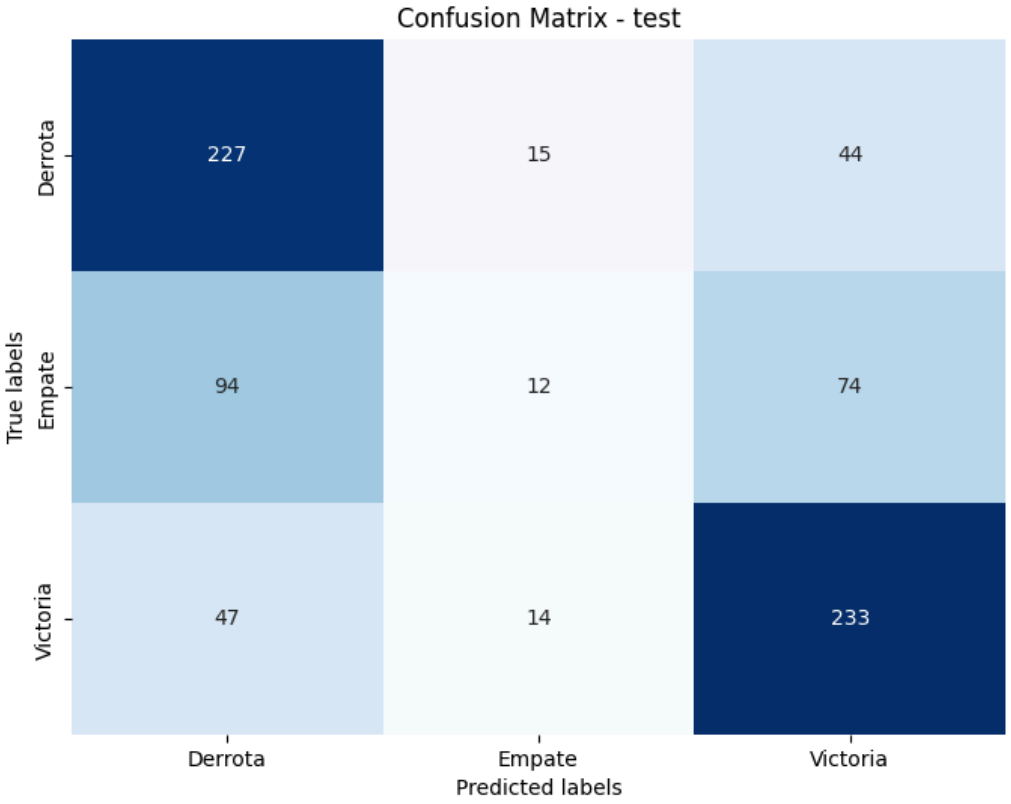
```python
# Parámetros de XGBoost con los mejores hiperparámetros
params = {
    'objective': 'multi:softprob',  # Probabilidades para clasificación multiclase
    'eval_metric': 'mlogloss',      # Log Loss para multiclase
    'num_class': 3,                 # Tres clases: 0 (derrota), 1 (empate), 2 (victoria)
    'learning_rate': 0.1,           # Tasa de aprendizaje ajustada
    'max_depth': 3,                 # Profundidad máxima ajustada
    'n_estimators': 100,            # Número de árboles
    'colsample_bytree': 1.0,        # Fracción de columnas por árbol ajustada
    'subsample': 1.0,               # Submuestreo ajustado
    'reg_alpha': 0,                 # Regularización L1 ajustada
    'reg_lambda': 0.1               # Regularización L2 ajustada
}
```

El resultado del modelo, tanto en el conjunto de validación como en el de prueba, mostró un aumento en la precisión (accuracy), pero el incremento fue de aproximadamente un 2% en ambos casos, lo cual, aunque positivo, no implica una mejora drástica.

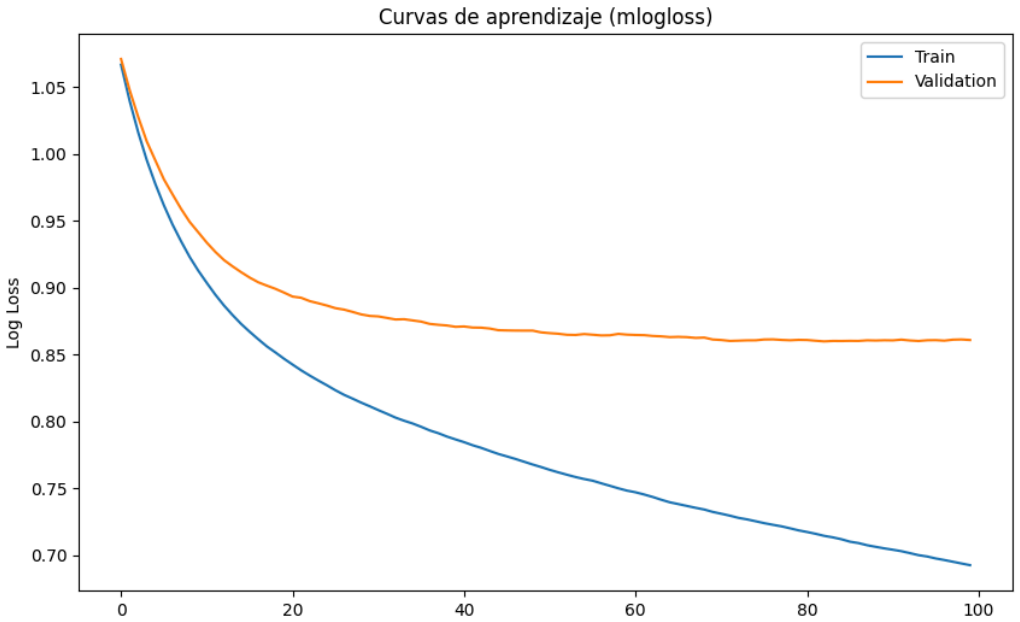
```
Validation Accuracy: 0.5954825462012321
Validation Confusion Matrix:
 [[135   9  35]
 [ 67   5  46]
 [ 34   6 150]]
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.75      0.65       179
           1       0.25      0.04      0.07       118
           2       0.65      0.79      0.71       190

    accuracy                           0.60       487
   macro avg       0.49      0.53      0.48       487
weighted avg       0.52      0.60      0.53       487

Test Accuracy: 0.6210526315789474
Test Confusion Matrix:
 [[227  15  44]
 [ 94  12  74]
 [ 47  14 233]]
Test Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.79      0.69       286
           1       0.29      0.07      0.11       180
           2       0.66      0.79      0.72       294

    accuracy                           0.62       760
   macro avg       0.52      0.55      0.51       760
weighted avg       0.56      0.62      0.57       760
```

Ahora bien, la matriz de confusión muestra que el modelo sigue prediciendo bien las victorias y derrotas, pero aún tiene dificultades en la clasificación de empates, lo cual sigue siendo un área de mejora.





Por último, tras los ajustes realizados, la curva de aprendizaje (learning curve) mostró una mejora en cuanto a la métrica log loss, indicando que el modelo está logrando una mejor convergencia durante el proceso de entrenamiento. Sin embargo, aunque la reducción en el log loss es notable, no representa un cambio abismal.



Esta mejora, aunque ligera, sugiere que el modelo es ahora más eficiente en la predicción de las clases, pero aún tiene margen para perfeccionarse, especialmente en lo referente a la clasificación de empates.

Conclusión

A lo largo de este proyecto, se implementó el modelo XGBoost para resolver un problema de clasificación multiclase en un conjunto de datos deportivos. Inicialmente, se trabajó con una versión básica del modelo, logrando una predicción adecuada para victorias y derrotas, aunque con limitaciones en los empates. Posteriormente, se aplicaron técnicas de mejora, como el ajuste de parámetros y la búsqueda por cuadrícula (Grid Search), lo que permitió optimizar el rendimiento del modelo.

Si bien las métricas, como la precisión y el log loss, mostraron una mejora moderada de aproximadamente un 2% en los conjuntos de validación y prueba, el cambio no fue lo suficientemente significativo como para considerarlo un salto drástico en el desempeño. La curva de aprendizaje también evidenció una disminución del log loss, lo que indica que el modelo se ajusta mejor, pero aún persiste una oportunidad de mejora, especialmente en la clasificación de empates.

En conclusión, aunque se lograron avances notables, el rendimiento del modelo podría optimizarse aún más mediante la exploración de nuevas técnicas, como un ajuste más fino de los parámetros o el uso de modelos complementarios. Este proceso de iteración y mejora continua es clave para obtener resultados más robustos en problemas de clasificación multiclase.