Boson de Higgs

TAA - proyecto 1
Grupo C

Magdalena Castrillo
Elias Courdin
Uriel Yaffé

08 mayo 2025

Índice

1. Objetivos

Familiarizarnos con los datos del desafío del Bossón de Higgs, el uso de la herramienta comet y la métrica del problema para finalmente diseñar e implementar un clasificador para el problema de detección de eventos *signal* y ser subido al kaggle del curso.

2. Introducción

El problema físico que convoca al desafío trata de la partícula Boson de Higgs que es una partícula fundamental de la física de partículas que se usa para explicar el origen de la masa de las partículas elementales. En 1964 se propuso la teoría pero no fue hasta 2012 que se comprobó la existencia de esta partícula. Tiene la particularidad de una muy rápida desintegración de unos 10^{-21} segundos [1] por lo tanto lo que se busca detectar son las partículas de su desintegración, su rastro. El experimento que permitió su detección fue realizado en el acelerador de partículas ATLAS el cual realiza choques de partículas los cuales son llamados eventos[3] y toma medidas de los mismos. La mayoría de estos eventos corresponde a otras partículas y son considerados *background* identificando a los eventos de interés como *signal*.

3. Análisis de los datos

Los 250.000 eventos que se encuentran en el conjunto de datos de *train* y los 550.000 del conjunto de *test* corresponden a simulaciones realizadas por físicos en el simulador ATLAS. Cada evento cuenta con 30 características, 17 tomadas de forma directa (PRI) y 13 derivadas de estas (DER). Los datos tienen identificados a los eventos con un número entero que se encuentra en la columna `EventId` y que será utilizada en este trabajo solamente a los efectos de conformar los archivos *csv* para cargar en kaggle y para evaluar la métrica AMS_metric. Los eventos se distinguen entre dos clases, por un lado *signal* que son los deseados y por otro lado los eventos *background*, en el conjunto de entrenamiento cada evento está marcado con su clase correspondiente en la columna `Label` como 's' ó 'b'. De los 250.000 eventos de *train*, 85.667 son *signal* y 164.333 son *background*, esto quiere decir que se cuenta con casi el doble de datos *background*. Cada uno de estos eventos tiene cierta probabilidad de ocurrencia en la vida real y es por eso que los datos de entrenamiento cuentan con un vector `Weight` que indica la frecuencia del suceso. Considerando los pesos de los eventos las clases están completamente desbalanceadas siendo un $0,17\,\%$ *signal* y $99,83\,\%$ *background*. Por el desbalance de clases podría querer encararse el problema como detección de anomalías, pero en este caso contamos con todos los datos de los eventos *background* los cuales están bien identificados y etiquetados, por lo cual no será este el abordaje. Los valores de las etiquetas fueron sustituidos por valores numéricos, siendo 's' remplazado por 1 y 'b' por 0.

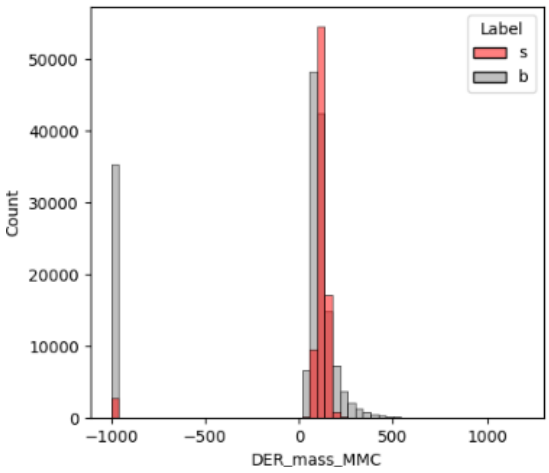

Figura 3.1: Distribución del conjunto de *train* para la *feature* `DER_mass_MMC`. La mayoría de los casos de masa faltante son eventos de fondo.

Este problema se caracteriza por tener muchos datos faltantes, simbolizados con el valor -999. En el conjunto de entrenamiento se visualiza una gran proporción de estos valores en 11 de las *features*, en los peores casos alcanza un $70\,\%$. En 10 de estas 11 *features* los valores faltantes se deben a que en algunos casos ciertas medidas no tienen sentido[2] por corresponder a *jets*[1] de valor 0 o 1, por lo tanto los valores -999 de estos eventos están relacionados con el valor de la característica `PRI_jet_num`. La otra característica que presenta valores -999 es `DER_mass_MMC` que toma este valor cuando la topología del evento es muy lejana de la topología esperada[2].

[1] Este término es físico y corresponde a un chorro de materia.

Entendiendo la diferencia entre estos valores faltantes se considera que la ausencia del dato de la masa puede llegar a ser un dato a considerar ya que en el conjunto de entrenamiento se visualiza que el 92,5 % de los eventos que no cumplen la topología son *background*, sin embargo esa relación no se cumple para los eventos en los que sí se cuenta con un valor de esta característica, donde el 61 % es *background*, esto puede verse en la figura 3.1. Otra observación que hicimos de los datos haciendo foco en los valores faltantes es que la característica `PRI_jet_num` es la única categórica y corresponde a 4 tipos de `jets`.

Visualizando la distribución de los datos para cada *feature*, tal como se puede ver para el caso de `DER_mass_MMC` en la figura 3.1, discriminando por clase se observa que las fronteras de decisión no son lineales en ningún caso, ya que siempre hay superposición, por lo tanto no parece conveniente abordar el problema con clasificadores lineales.

4. Metrica de evaluación - AMS

La métrica de evaluación sugerida para este problema es *approximate median significance* (AMS) que es una medida que toma en cuenta el peso de cada evento. La misma consiste en:

$$AMS = \sqrt{2\left((s+b+b_r)\log\left(1+\frac{s}{b+b_r}\right)-s\right)} \quad , \quad b_r = 10$$

Donde s corresponde a la cantidad de eventos *signal* clasificados como tales, es decir verdaderos positivos (TP por su sigla en inglés) y b son los eventos *background* que fueron clasificados como *signal*, es decir falsos positivos (FP). Estos valores de s y b se encuentran escalados por su peso, que se encuentra en la característica `Weight`.

Esta métrica fue utilizada para definir el umbral de clasificación de los desempeños de los mejores modelos obtenidos en validación (usando otra métrica como *score*) y para afinar los modelos previo a evaluar en el conjunto de *test*.

Un primer acercamiento al ver el gran desbalance de clases analizado en la sección 3 se puede ver al probar con un estimador *dummy* que marque a todos los eventos como *background*, es decir 0. Si bien esto nos llevaría a un *acurracy* muy bueno, en realidad el AMS sería nulo ya que ningún s es detectado, esto implica también un *recall* y un *precision* nulos. Esta es una forma de ver la importancia de esta métrica, ajusta mejor a la física detrás en este problema ya que no se trata de un simple caso de clasificación binaria sino que queremos detectar los eventos *signal* que son sumamente raros tratando de tener un alto valor de TP y un bajo valor de FP. Por otro lado, se probó con un estimador *dummy* que marca a todos los eventos como positivos, de esta forma seguro detectamos todos los eventos *signal*, claramente es un estimador malo porque hay muchos FP. Para este estimador se obtuvo un AMS de 0,48, lo cual sirve como partida de comparación de otros valores obtenidos con modelos más complejos los cuales deberán ser mayores a éste.

5. Conjuntos de entrenamiento, validación y prueba

Para todos los modelos entrenados, inicialmente se realizó una validación utilizando el 20 % del conjunto de *train*, reservando el 80 % restante para definir el preprocesamiento más adecuado, ajustar los hiperparámetros de cada modelo y determinar el umbral óptimo para el AMS. Esta partición se realizó mediante la función `train_test_split`, utilizando el parámetro `stratify` para preservar el balance de clases. Finalmente, para evaluar el rendimiento en el conjunto *test*, se entrenó cada modelo utilizando la totalidad del conjunto *train*.

6. Modelos intermedios y preprocesamiento

6.1. Método general

Se probaron y compararon 4 modelos diferentes, lo cual requirió preprocesamientos adecuados a las particularidades de cada uno. Sin embargo, para todos ellos se usó una base de pasos general en el entrenamiento para lidiar con

la particular métrica de evaluación que tiene el problema y el desbalance de las clases.

Seleccionamos los hiperparámetros mediante una búsqueda con `GridSearch`, para esto se utilizó como *score* la métrica roc_auc. Esta métrica fue elegida por tomar en cuenta las tasas de FP y TP y por lo tanto no verse afectada a priori por el desbalance[2]. Como preprocesamiento general de todos los modelos realizamos una reducción del conjunto de datos con el objetivo de construir una versión simplificada del *dataset*, con un número significativamente menor de características. La idea fue trabajar inicialmente con un conjunto básico pero representativo de los datos, de modo que los modelos e hiperparámetros seleccionados pudieran generalizar de forma robusta a su vez disminuir el costo computacional de los `GridSearch` con muchas combinaciones. Esto permite, por ejemplo, reutilizar dichos modelos si se quiere repetir el experimento con algunas modificaciones en las variables (como agregar, eliminar o sustituir algunas columnas), sin que el rendimiento se vea tan afectado. En cambio, si se ajustan hiperparámetros sobre un modelo entrenado con todas las *features* disponibles (más específico), estos parámetros podrían no acomodarse bien a configuraciones distintas del conjunto de datos.

Para construir esta versión reducida del *dataset*, inicialmente realizamos una etapa de preprocesamiento sobre las variables numéricas del conjunto de entrenamiento. Se aplicaron distintas estrategias de imputación y escalado en función del tipo de variable que se distinguieron en PRI y DER. Se optó por imputar los valores faltantes de las variables DER con la mediana por ser cálculos obtenidos a partir de otras variables y a las PRI se las imputó con 0 para indicar una ausencia de dato. Si se usara la misma estrategia para todas las columnas, se correría el riesgo de introducir sesgos innecesarios (llenando de datos un mismo valor) o distorsionar distribuciones que dificultarían al modelo aprender. Todas las características fueron normalizadas con la media y la varianza.

Luego se aplicó una función que eliminó algunas columnas en los casos de pares cuya correlación es superior a 0.8, con el fin de reducir la colinealidad de las *features*, evitar sobreajuste y eliminar cierta redundancia. Luego, utilizamos un Análisis de Componentes Principales (PCA) conservando el 99 % de la varianza.

Estas dos formas de "limpiar" el *dataset* permitieron reducir la dimensionalidad de 30 a 20 variables, manteniendo la mayor parte de la información relevante del conjunto original. De esta forma, se buscó mejorar la generalización de los modelos y facilitar su adaptabilidad a escenarios levemente modificados.

Aunque podría parecer redundante eliminar columnas altamente correlacionadas y luego aplicar PCA, en un conjunto de datos de gran tamaño esta estrategia resultó útil para depurar y ajustar el dataset, facilitando así la búsqueda de hiperparámetros óptimos utilizados en los distintos modelos y experimentos del estudio.

Una vez seleccionados los hiperparámetros, se predijeron las probabilidades de pertenecer a la clase *signal* para los datos del conjunto de validación, y se buscó fijar un umbral de forma de maximizar el AMS en este conjunto para cada uno de los modelos Para esto se grafica Treshold vs AMS y se observa su comportamiento, esto se profundizará en la descripción de cada modelo.

6.2. Random Forest

El primer modelo entrenado fue *Random Forest* que luego de realizar búsqueda de los mejores hiperparámetros como se explicó en el método general, se obtuvieron los de la tabla 6.1. Luego buscamos el mejor umbral y AMS de este modelo, para esto fue necesario realizar imputación de los datos faltantes, y se optó por realizar la misma que se usó para buscar los hiperparámetros probando con además imputar las *features* `PRI` con la mediana. Para encontrar el umbral que maximiza el AMS se evaluaron varios modelos modificando el *pipeline* principal probando realizar reducción de dimensionalidad eliminando *features* con alta correlación con y sin realizar PCA luego. También se probaron estos modelos agregando la característica *topology* que indica 0 si el valor de la masa es -999 y 1 en otro caso. Así como realizar un `OneHotEncoder` para la *feature* `PRI_jet_num`. Por otro lado como en este modelo existe la posibilidad de indicarle el parámetro `sample_weight` al momento de entrenar lo probamos agregando este paso al realizar el *fit* para los 4 mejores modelos encontrados previamente, se descartó este camino por obtener peores desempeños en validación.

Para todas estas combinaciones se realizaron 19 experimentos, los cuales fueron registrados en comet y se pueden revisar accediendo aquí. Para el resto de los modelos se procedió de forma similar realizando varias pruebas para encontrar con qué preprocesamiento se encontraba el mejor AMS pero a efectos de este informe sólo se detallará el mejor obtenido.

[2]Podríamos haber utilizado PR curve que es buena cuando el desbalance es desfavorable para la clase positiva.

Tal como se puede comprobar en comet el mejor ensámble de árboles fue obtenido realizando imputación nula de los valores faltantes de las `PRI` y `OneHotEncoder` de `PRI_jet_num`. En la figura 6.1 donde se puede observar la gráfica del AMS para varios umbrales. En este caso el mejor desempeño fue de un AMS=1.5415 con un *threshold* de 0.7373.

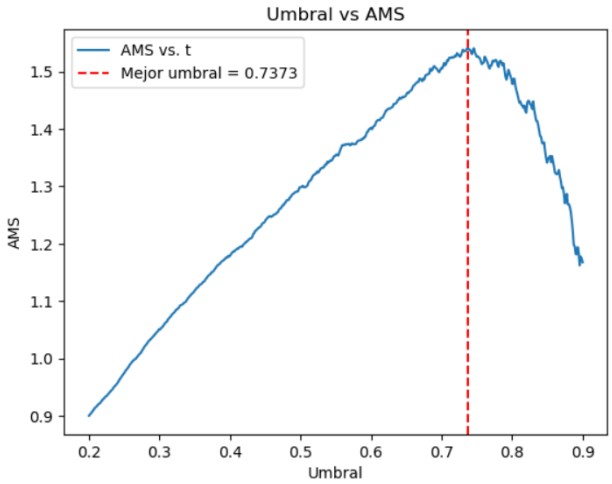

Figura 6.1: Treshold vs AMS para el modelo de RF

Hiperparámetro	Valor
max_depth	50
n_estimators	700
min_samples_leaf	2
min_samples_split	5

Cuadro 6.1: Mejores hiperparámetros encontrados para el modelo de RF

6.3. Gradient Boosting

6.3.1. XGBoost

Se entrenó un modelo XGBoost, de la familia de Gradient Boosting. El modelo fue seleccionado ya que interesaba la idea de combinar clasificadores débiles (en este caso árboles de decisión) y ver como se comporta para este problema. Además, la principal motivación para utilizar este tipo de modelos es que pueden manejar datos faltantes, es decir, no hace falta darles un valor numérico a los valores -999. Por esta razón, el pipeline de preprocesamiento utilizado es un `ColumnTransformer` que intercambia los valores faltantes por NaN, que significa *not a number*.
Al final del *pipeline* se entrena el modelo de Gradient Boosting sobre los datos de entrenamiento con los mejores hiperparámetros encontrados que se observan en el cuadro 6.2, se predicen las probabilidades para el conjunto de validación y se grafica la curva Treshold vs AMS, que se observa en la figura 6.2.

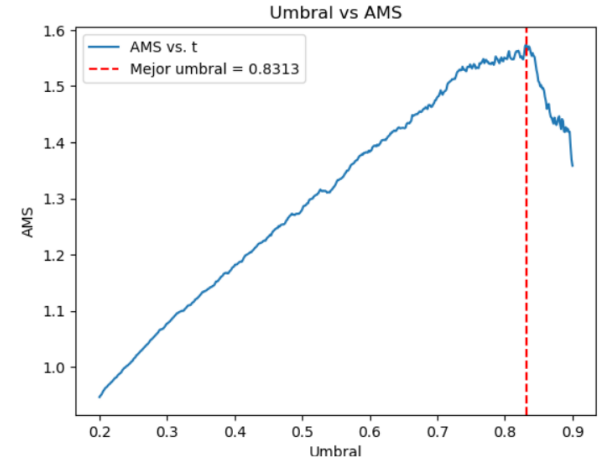

Figura 6.2: Treshold vs AMS para el modelo de XGBoost

Hiperparámetro	Valor
n_estimators	400
learning_rate	0.05
sub_sample	0.8
max_depth	7
gamma	0
colsample_bytree	0.8

Cuadro 6.2: Mejores hiperparámetros encontrados para el modelo de XGBoost

En la figura 6.2 se puede observar como el umbral óptimo se encuentra alrededor de 0,8. De la gráfica podemos ver que esta estrategia funciona bastante bien, alcanzando un AMS considerablemente alto para el conjunto de validación, lo que hace pensar que trabajar con datos NaN es una buena estrategia.

6.3.2. LGBM

LGBM es otro modelo basado en Gradient Boosting muy similar a XGBoost, pero con algunas diferencias. Es particularmente eficiente para grandes cantidades de datos aunque aveces es más sensible al sobreajuste.
El preprocesamiento para este modelo es el mismo que el descrito para XGBoost, por lo que solo se muestran los mejores hiperparámetros encontrados en el cuadro 6.3 y la curva Treshold vs AMS en la figura 6.3. Se puede apreciar que el desempeño entre los modelos de Gradient Boosting es muy similar.

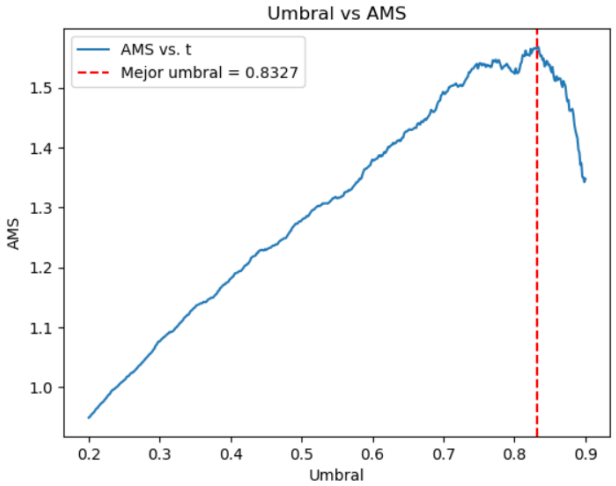

Figura 6.3: Treshold vs AMS para el modelo de LGBM

Hiperparámetro	Valor
n_estimators	600
learning_rate	0.05
max_depth	10

Cuadro 6.3: Mejores hiperparámetros encontrados para el modelo de XGBoost

6.4. Redes Neuronales

Se probó un modelo del tipo Red Neuronal buscando aprovechar la capacidad de este tipo de modelos de aprender en problemas complejos como el que se está tratando. Las redes neuronales se desempeñan bien en conjuntos con muchos datos, y pueden aprender relaciones no lineales entre características.
Para comenzar a trabajar con redes se hizo un *pipeline* de preprocesamiento particular para este tipo de modelos. Como las redes funcionan mejor si los datos están correctamente escalados, y observando los valores que toman diferentes columnas se puede apreciar que los rangos de variaciones difieren entre sí, de forma tal que se deben escalar los datos antes del entrenamiento.
Por otra parte, se debe hacer algo con los datos faltantes. Lo que se decidió hacer es crear, para cada columna con datos faltantes, una columna extra binaria llamada "$columnaX\_missing$", indicando si columnaX tiene valor faltante o no para cada dato. Para las columnas originales "$columnaX$", el valor -999 fue reemplazado por la media. De este modo se agregan 11 columnas al *Dataframe*.
Además, se usa `OneHotEncoder` para la columna `PRI_jet_num`, agregando 3 columnas más.
Finalmente se usa `StandardScaler` para las columnas numéricas, es decir, aquellas que no son `PRI_jet_num` ni las binarias agregadas.
Para entrenar la red se procesan los datos entrenamiento y validación, ya que éstos últimos se utilizan dentro del entrenamiento.
El método utilizado para hallar los mejores hiperparámetros para la red consiste en comenzar con una red grande y que busca sobreajustar a los datos, a partir de ahí comenzar a reducir el número de capas y neuronas, agregando

también métodos de regularización con el objetivo de maximizar la métrica `roc_auc` en validación. Se comenzó con una red con 4 capas y 400 neuronas por capa, con un learning rate de 0,001. En este caso se pudo observar como la red comienza a sobreajustar, como se muestra en la figura 6.4, se puede ver como la función de *loss*, que en este caso es "binary crossentrophy", continua reduciendose para el conjunto de *train* pero aumentando en el conjunto de validación.
Para combatir el *overfitting* se hizo uso de técnicas como el *EarlyStopping* y *Dropout*. El *EarlyStopping* permite que el entrenamiento frene cuando la métrica en validación no está mejorando, restaurando el estado de los pesos al mejor que se obtuvo durante el entrenamiento. El *Dropout* es un parámetro que controla la cantidad de neuronas que aprenden en una capa, apagando una cierta porción de ellas para evitar sobreajustar. También se utilizó una técnica que consiste en reducir el *learning rate* durante el entrenamiento cuando éste comienza a estancarse.
Dicho esto, se procedió reduciendo el numero de capas y neuronas, agregando las técnicas mencionadas, monitoreando el comportamiento de la red.
Junto con este mecanismo, se utilizó keras hipertuner para realizar una búsqueda en paralelo, variando el *learning rate*, numero de capas y numero de neuronas.
Al final de la búsqueda, los mejores resultados se obtuvieron para una red de 2 capas con 300 neuronas cada una, un *learning rate* de 0,0001 y un factor de Dropout de 0,3.
Luego de entrenar la red se grafica Treshold vs AMS que se observa en la figura 6.5.

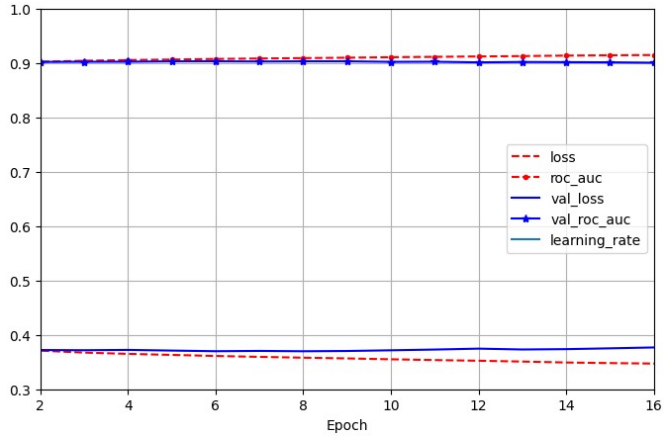

Figura 6.4: History de la red sobreajustada

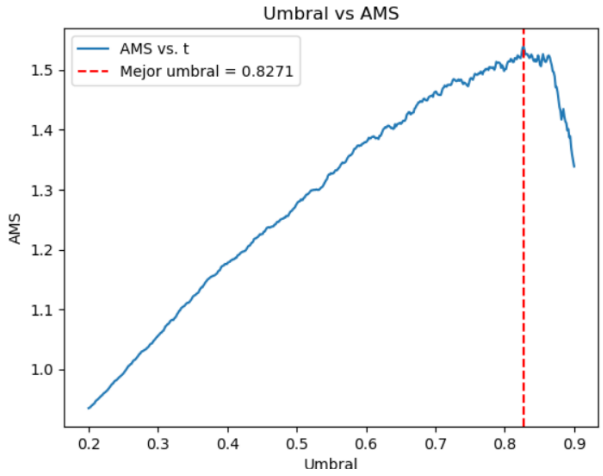

Figura 6.5: Treshold vs AMS para el modelo de Red Neuronal

7. Modelo final

Para la elección del modelo final, se optó por una estrategia de ensamble que combina varios modelos individuales con buen desempeño, en lugar de seleccionar uno solo. Esta decisión se basa en que modelos distintos tienden a cometer errores diferentes, por lo que al combinarlos se puede mejorar la capacidad de generalización del sistema. Además, se utilizaron distintos esquemas de preprocesamiento para cada uno de los modelos incluidos en el ensamble. De esta manera, cada modelo aprende diferentes representaciones del dataset, destacando distintas estructuras o relaciones entre variables.

La construcción del modelo final se optó por un ensamble que combina varios clasificadores con buen desempeño individual: un XGBoost, un *Random Forest*, un LightGBM y una red neuronal. Estos modelos se integran usando la idea de un *VotingClassifier* ponderando sus probabilidades, lo que permite aprovechar sus fortalezas combinadas y aporta mayor robustez a las predicciones. Todos los modelos se entrenaron con los mejores hiperparámetros obtenidos en las etapas previas.

La idea fue crear un sistema que mezcla modelos de distinta naturaleza, esta diversidad de modelos permite capturar diferentes aspectos del conjunto de datos, los árboles como RF , XGB y LGBM son eficaces para capturar relaciones no lineales en datos y son robustos ante *outliers*, mientras que la red neuronal puede capturar patrones más complejos e interacciones sutiles entre variables, lo que complementa lo aprendido por los otros modelos.

Como último detalle, dado que los distintos modelos tenían rendimientos dispares, decidimos no ponderar sus probabilidades de forma equitativa. En cambio, optamos por asignar mayor peso a algunos modelos sobre otros.

En particular, observamos que al darle más peso a las probabilidades del modelo XGBoost y de la red neuronal, en comparación con el *Random Forest* y el LGBM, el rendimiento final mejoraba.

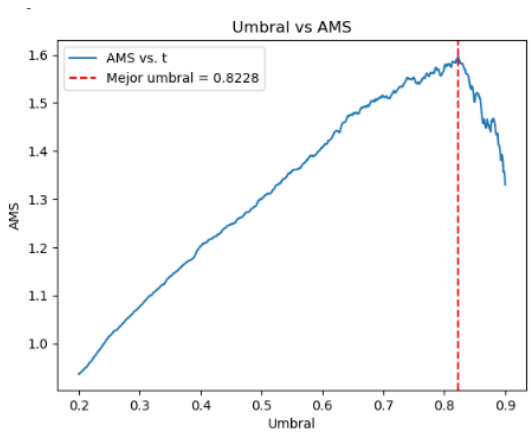

Figura 7.1: Threshold vs AMS para el modelo final con ponderación equiprobable con AMS = 1.597

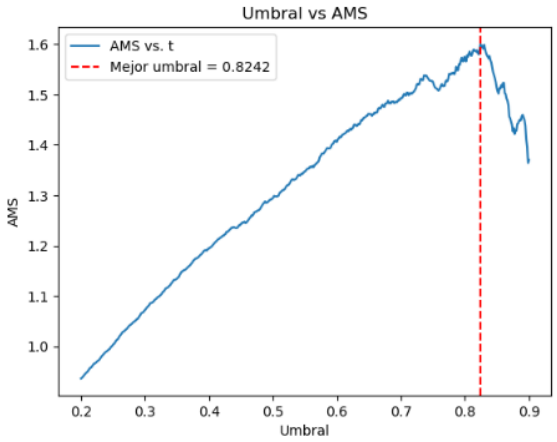

Figura 7.2: Threshold vs AMS para el modelo final con ponderación no equiprobable con AMS = 1.600

En la Figura 7.1 se aprecia el mejor *threshold* cuando a cada probabilidad por cada modelo se le asigna igual peso, $\frac{1}{4}$ a cada predicción de probabilidad, mientras que en la Figura 7.2 es la misma gráfica donde se le da mas peso a la red que a los otros modelos ($\frac{3}{5}$a la red y $\frac{2}{15}$ al RF,XGB y LGBM).

Para tratar de mejorar aún más, probamos correr un Optuna Study en las variables que indican que tanto peso le da a cada probabilidad dada por cada modelo para lograr un mejor valor de AMS.

Se llegó a los valores:

$$\begin{aligned} \text{peso}_{XGB} &= a = 0{,}4785 \\ \text{peso}_{LGBM} &= b = 0{,}2469 \\ \text{peso}_{RF} &= c = 0{,}0272 \\ \text{peso}_{RED} &= d = 1 - a - b - c = 0{,}2475 \end{aligned}$$

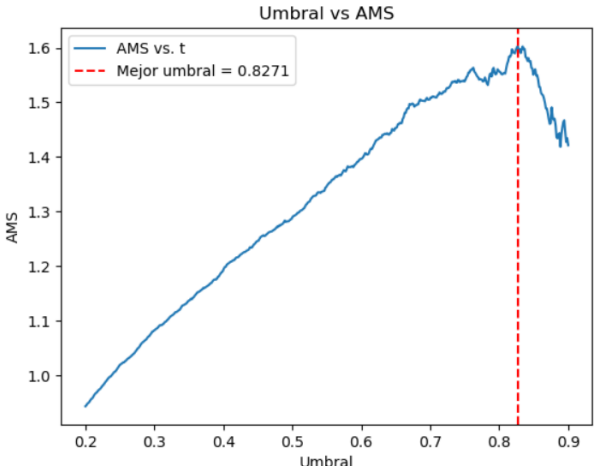

Figura 7.3: Threshold vs AMS para el modelo final con AMS = 1.603

8. Conclusiones

A lo largo del proyecto nos enfrentamos al problema del desbalance de clases y de una nueva métrica de evaluación lo que nos llevó a tomar múltiples decisiones dentro de la cantidad de herramientas posibles, incluso de forma algo arbitraria en algunos casos. Entendemos que esto representó un primer acercamiento a un problema real de aprendizaje automático, así como a pulir y tratar de maximizar el rendimiento de modelos vistos y tener una primera interacción con herramientas que permiten facilitar la interacción con modelos,como son kaggle y comet.

A lo largo del trabajo, se vio la importancia de adaptar la estrategia de entrenamiento a la métrica AMS. El ajuste del umbral de decisión resultó ser clave para optimizar este indicador.

Durante el desarrollo del proyecto, se hizo énfasis en la reproducibilidad de los experimentos mediante el uso de herramientas como Kaggle y Comet. Esto permitió registrar y observar los distintos entrenamientos, configuraciones de modelos y métricas obtenidas, facilitando la comparación entre enfoques y modelos. Además, el uso de los notebooks con seeds y pipelines ayudó a que los resultados pudieran ser replicados por otros integrantes del equipo.

Se eligieron modelos de ensamble de árboles del tipo *bagging* (RF) y *boosting* (XGB y LGBM) y una red neuronal abarcado así varios modelos vistos en el curso, que aprenden de forma diferente y aprovechando diferentes procesamientos de los datos. Por esto mismo, una combinación final de los 4 mejores modelos permitió aprovechar mejor cada aprendizaje lo cual nos permitió alcanzar un AMS mayor, en el siguiente enlace se puede ver los experimentos finales de comet. La submission a Kaggle del curso dió un AMS de 3.52820.

Referencias

[1] Wikipedia. `https://es.wikipedia.org/wiki/Bos%C3%B3n_de_Higgs`.

[2] Claire Adam-Bourdarios, Glen Cowan, Cecile Germain, Isabelle Guyon, Balazs Kégl, and David Rousseau. Learning to discover: the Higgs boson machine learning challenge. `https://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf`, 2014.

[3] Instituto de Ingeniería Eléctrica Facultad de Ingeniería. Taller de Aprendizaje Automático. Proyecto 1 - Bosón de Higgs. `https://eva.fing.edu.uy/pluginfile.php/332767/mod_resource/content/5/proyecto_1_2025.pdf`, 2025.