Universidad de la República

Facultad de Ingeniería

Taller de Aprendizaje Automático

Proyecto 1 | Bosón de Higgs

Autores:
Federico Bello
Gonzalo Chiarlone
Ernesto Rován

28 de septiembre de 2025

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Índice

1. Introducción y Objetivo

La detección del bosón de Higgs (o la partícula de Dios) es de vital importancia ya que es una pieza clave en la comprensión de cómo las partículas elementales obtienen su masa. Según la teoría del Modelo Estándar de la física de partículas, el campo de Higgs es responsable de dar masa a las partículas elementales, y el bosón de Higgs es la partícula asociada a este campo. Su descubrimiento confirmó la existencia del campo de Higgs y completó la teoría del Modelo Estándar.

Esta detección, se hace mediante aceleradores de partículas. Los mismos llevan las partículas a una gran energía, logrando así analizar como las mismas interactúan entre si. Sin embargo, debido a la naturaleza del problema, las medidas de estos experimentos tienen una baja precisión, complejizando el análisis del mismo. Es aquí, donde los modelos de aprendizaje automático entran en juego.

El objetivo del proyecto es poner en practica las herramientas vistas en el curso, buscando detectar la presencia del bosón de Higgs a partir de mediciones de otras magnitudes.

2. Exploración de los datos

Se cuenta con un conjunto de datos para Train de 250000 sucesos, cada uno con 30 *feature columns*, una columna de pesos para considerar el desbalance de las clases, una columna ID que identifica cada suceso, y una con las *labels*. Estas ultimas columnas serán quitadas de los datos de Train. Las *labels* toman los valores 's' (*signal*) y 'b' (*background*), por lo que es un problema de clasificación binario. Los datos son simulados, teniendo en entrenamiento un desbalance en un 65 % a favor de 'b', que a su vez es mucho mayor en los datos reales. Este desbalance se compensa con la columna *Weight*, la cual se utiliza para que la suma ponderada de cada clase con sus respectivos pesos dé como resultado la cantidad esperada de eventos en el tiempo que se adquirieron los datos. Además de esta normalización por clase, cada muestra tiene un peso asignado, indicando la importancia de la misma en el conjunto.

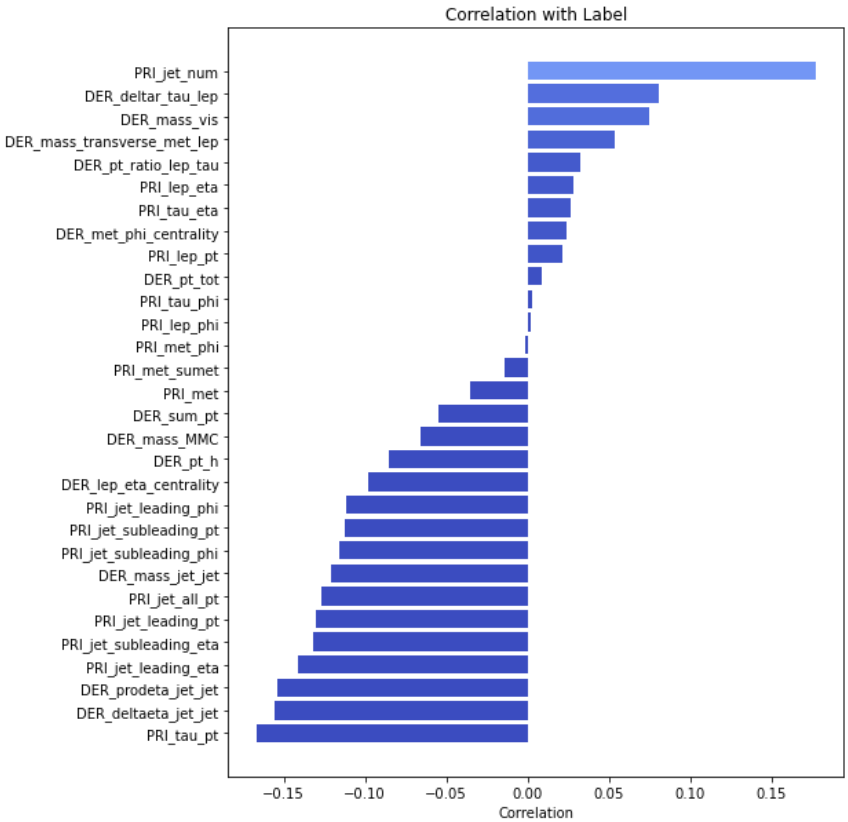


Figura 1: Correlaciones con el atributo *Label*

Una peculiaridad del problema, es que los datos vacíos son expresados con -999.000. Estos no solo representan la imposibilidad de haber realizado esa medición, sino que además se utilizan cuando la misma no pudo ser calculada o no tiene sentido calcularla o medirla para este caso particular. Por lo tanto, este valor tiene un significado por si mismo, y no necesariamente implica datos incompletos. De las columnas con estos valores, 7 están conformadas por más de un 70 % de datos null, 3 de ellas en más de un 40 % y una tiene aproximadamente 16 % null. La correlación entre estas columnas y la *label* no es tan alta, por lo que se evaluará en cada modelo si se obtiene un mejor resultado utilizando estas columnas o simplemente no aportan suficiente información al problema.

En la figura 1 se observan las correlaciones con la etiqueta *label*. Notar que en general, las correlaciones de todos los atributos son considerablemente bajas, ninguna supera el 0.20.

La figura 2 muestra los histogramas de los respectivos atributos. Notar como algunos atributos tienen una distribución normal (o parecen tenerla), mientras que otros tienen distribuciones menos amigables, en particular varias parecen tener distribuciones de poisson. Para estas ultimas puede llegar a ser beneficioso realizar alguna transformación logarítmica, y así asemejarlas a distribuciones normales.

Los datos de entrenamiento fueron separados en un conjunto de entrenamiento y otro de validación con una división de 90 %/10 %. Esta separación se mantiene en todos los modelos para evaluar sobre los mismos conjuntos de datos cada uno, consiguiendo así que los modelos sean mas comparables entre si. Para las *labels* se asigna un 1 para 's', y 0 para 'b'.

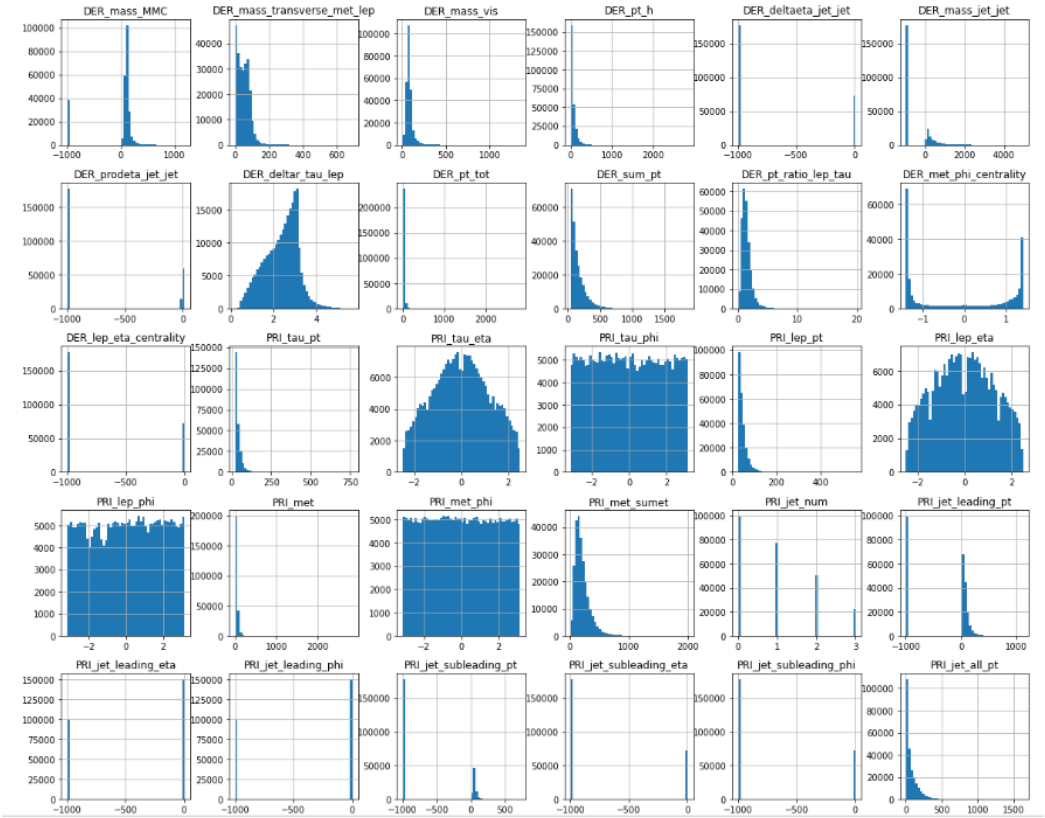


Figura 2: Histogramas

3. Métrica

La métrica a utilizar para la evaluación del modelo es la *Approximate Median Significance* (AMS). Esta métrica mide la capacidad de un modelo de distinguir correctamente entre clase positiva y clase negativa, teniendo en cuenta el compromiso *recall-precision*. A un mayor valor de la métrica, un mejor desempeño. La misma, se calcula como:

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

donde:

- s, b: las tasa de predicciones positivas correctas y falsos positivos sin normalizar respectivamente
- b_r**:** constante de regularización (10 en este caso)

Notar que la métrica recompensa principalmente la detección de los eventos *signal*, mientras que penaliza identificar incorrectamente eventos *background*. Teniendo esto en cuenta, la medida parece ser similar a *precision*, sin embargo, se diferencian en que la *precision* mide la proporción de predicciones positivas correctas, mientras que la AMS mide la capacidad del modelo de separar los verdaderos positivos de los falsos positivos, teniendo en cuenta los pesos de cada clase y muestra. Por ejemplo, la precisión es máxima (1) cuando no hay falsos positivos, mientras que en esta misma situación se puede obtener un mal valor de AMS (debido a que la tasa de predicciones positivas correctas sea baja). Por lo tanto, si bien las medidas son similares y, probablemente, para conseguir un buen valor de AMS sera necesario una buena precisión, mejorar esta ultima no necesariamente mejora la AMS.

En este caso, como las muestras *signal* y *backgorund* están normalizadas, se obtiene que:

$$s = \sum_{i=1}^{n} w_i \mathbb{1}\{y_i = s\}\mathbb{1}\{\hat{y}_i = s\}$$

$$b = \sum_{i=1}^{n} w_i \mathbb{1}\{y_i = b\}\mathbb{1}\{\hat{y}_i = s\}$$

siendo w_i el peso de cada muestra, $\mathbb{1}$ la función indicatriz, y_i y $\hat{y}_i$ la etiqueta real y la predicha respectivamente.

Una observación no menor es que al utilizar los valores desnormalizados, el valor de la métrica depende directamente de la cantidad de datos. Por lo tanto, sera necesario tener lineas bases para valores de validación y entrenamiento, manteniendo en todos los modelos la cantidad de datos en cada conjunto, para así poder compararlos.

4. Modelos

4.1. Regresión Logística

Inicialmente, se utilizó un modelo de regresión logística para tener una linea base sobre la cual comenzar. El objetivo de este primer modelo no es obtener un buen desempeño, sino conseguir el mínimo desempeño que se puede conseguir con algo simple, obteniendo así un resultado contra el que comprar.

Preprocesamiento

Para cumplir con el objetivo descrito anteriormente, se realizó un preprocesamiento simple. Se descartan todas las columnas con valores nulos, y luego se estandarizan los datos.

Entrenamiento y Regularización

A partir del modelo de regresión logística, se realiza la búsqueda del hiperparametro C con una *Grid Search* por validación cruzada y métrica de evaluación por defecto (*accuracy)*

Resultado

Siguiendo este procedimiento, se obtienen los resultados de la tabla 1 para las métricas de *accuracy, precision y recall*. Notar como no solo no se obtiene un buen resultado, sino que el resultado en entrenamiento y validación es casi igual indicando que el modelo no esta logrando ajustarse correctamente a los datos, está subajustado. Esto implica que es necesario aumentar la complejidad del modelo para lograr un mejor resultado.

Métrica \ Conjunto	Entrenamiento	Validación
Accuracy	0.738	0.744
Precision	0.646	0.645
Recall	0.526	0.525

Cuadro 1: Desempeño del modelo de Regresión Logística en los distintos conjuntos de datos

Evaluando el modelo con la métrica AMS, se obtiene un resultado de 1,885 en entrenamiento y 0,61 en validación. Estos son los valores base, en las próximas secciones se buscara que aumenten.

4.2. Modelos de Ensamble: Random Forest y XGBoost

4.2.1. Random Forest

Se comienza probando con un Random Forest (incluyendo los pesos en *fit*) simple sin regularización ni preprocesamiento dado que Random Forest maneja bien los datos de diferentes escalas, sin necesidad de estandarizar los valores.

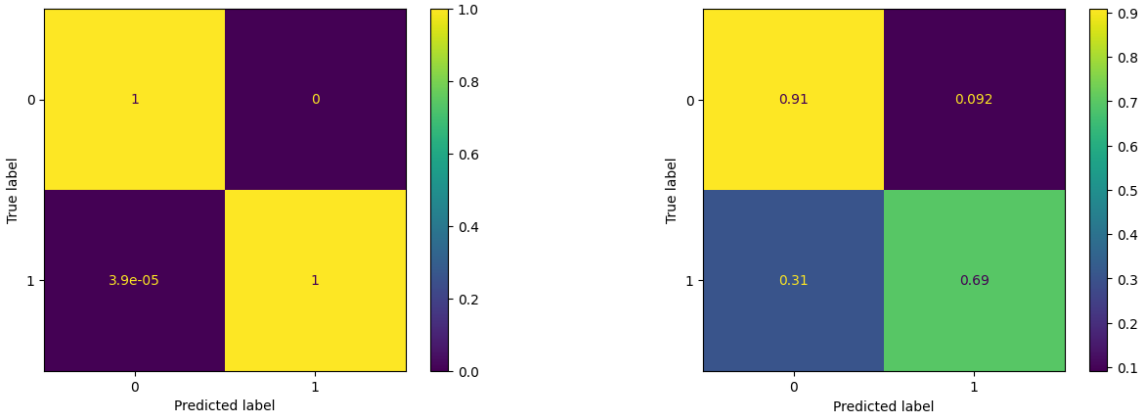
Como se observa en la sección de Resultados, el error de Train es casi nulo, mientras que en validación es un poco alto (particularmente en la predicción de positivos), pero el valor de la métrica AMS no es malo, sin dudas bastante mejor en relación con el modelo anterior (regresión logística). Comparando Train y validación, y consdierando que los valores de *accuracy, precision y recall* del entrenamiento son practicamente uno, es claro que el modelo se está sobreajustando a los datos, por lo que es conveniente regularizar.

Regularización y preprocesamiento

Se analiza la profundidad máxima de cada árbol del estimador entrenado y se osberva que se encuentra entre 47 y 62. Con *GridSearch* se prueban distintos valores del hiperparámetro *max_ depth* en el entorno mencionado, para que los Árbol entrenados se ajusten menos a los datos, y el resultado para el mejor estimador (*max_ depth*=44) es levemente mejor, como se observa en la sección de resultados.

Como preprocesamiento de los datos se remplazan los valores null con el criterio de la mediana por columna, al ver que mejora (muy poco) los resultados, aunque RandomForest no tiene problemas para tratar datos faltantes. A su vez, en el entrenamiento se asignan los *weights*, utilizando el parámetro *sample_ weight*, ya que mejora el desempeño. Como los métodos de ensamble de Decision Trees parecen tener buen rendimiento, a continuación se prueba con una variante para ver si mejora el panorama: XGBoost.

Resultados



(a) Matriz de confusión en Train (RF sin reg.)

(b) Matriz de confusión en Val (RF sin reg.)

Figura 3: Matrices de confusión para Random Forest sin Regularizar

Modelo \ Conjunto	Entrenamiento	Validación
Sin Regularizar ni Preprocesamiento	2651	0.897
Con Regularización y Preprocesamiento	2551	0.908

Cuadro 2: Desempeño de Random Forest con métrica AMS

4.2.2. XGBoost

Como próximo modelo a probar se decidió utilizar *Extreme Gradient Boosting* (XGBoost). El mismo tiene una gran capacidad de ajustarse a los conjuntos de datos, aunque tiene una tendencia a sobre ajustarse. Sin embargo, tiene una gran cantidad de hiperparametros para poder regularizar y, además, se tiene una gran cantidad de datos en el conjunto de entrenamiento.

Preprocesamiento

Primero, se probó el modelo sin ningún tipo de preprocesamiento. Como ya se mencionó, los modelos basados en árboles suelen comportarse bien tanto con los *outliers* como con los datos faltantes, por lo que sería esperable que el modelo funcione correctamente sin necesidad de aplicar preprocesamiento, al igual que con Random Forest. Sin embargo, se probaron algunos preprocesamientos para intentar conseguir una mejora respecto al modelo sin preprocesar.

Ya que parece ser el modelo con mejor desempeño hasta el momento, vale la pena realizar un ajuste más fino. Se probaron distintas alternativas, utilizando un subconjunto de los datos como validación para seleccionar la mejor de todas. El primer intento fue rellenar los datos faltantes con algún criterio (mediana o más frecuente), para luego estandarizar, verificando que efectivamente esto no tiene impacto positivo en el resultado. Por encima de esto, se probó eliminar los *outliers*, definiéndolos como datos que distarán a la media más de 4 veces la desviación estándar, consiguiendo así un peor resultado que sin preprocesar.

Teniendo en cuenta que las transformaciones anteriores a los datos no dieron resultados positivos, se decidió aplicar ingeniería de características para crear nuevas columnas con las ya existentes. Se seleccionaron las columnas que contienen datos estrictamente positivos, a los cuales se les aplicó el logaritmo al inverso del valor. Esto generalmente se utiliza con el objetivo de normalizar la distribución de los datos, además de reducir el impacto de *outliers*. A priori, no necesariamente seria beneficioso en los modelos

basados en árboles, aunque se verá más adelante que efectivamente resultó en una leve mejora de desempeño. Asimismo, se seleccionaron los atributos de ángulos, agregando por cada una de las tres las funciones trigonométricas (seno, coseno y tangente). Para que sea posible crear estas dos columnas, los datos no pueden tener valores nulos, por lo que previo a realizarlo fue necesario suplantar los mismos con algún criterio, en este caso, la mediana.

Por otro lado, se probó también utilizar la función *PolynomialFeatures* de *scikit-learn*, generando nuevas columnas con todas las combinaciones posibles de los atributos de grado menor o igual a dos. Esto no solo empeoró el resultado, sino que aumentó drásticamente el costo computacional del entrenamiento.

Entrenamiento y Regularización

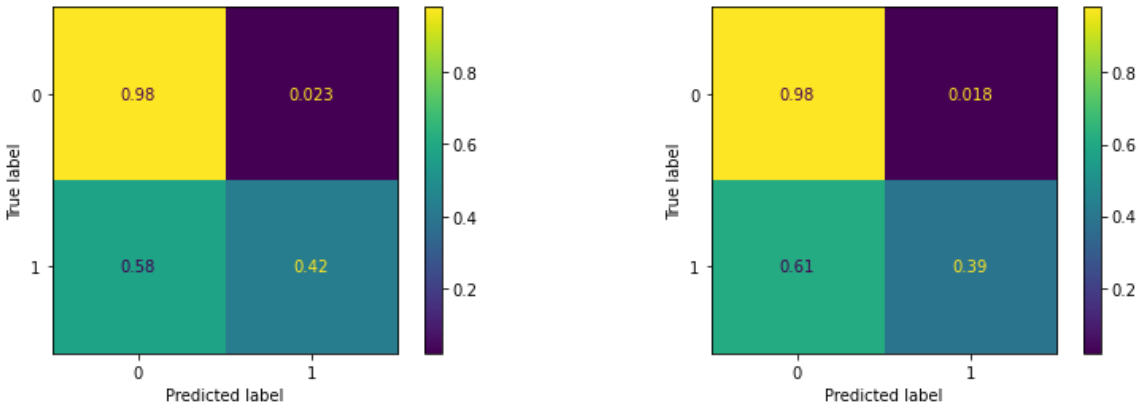
Este modelo permite fácilmente ajustar los pesos como parámetros, tanto los pesos para balancear las clases al nivel deseado como para los pesos de cada muestra. Matemáticamente, para balancear la clase positiva (*signal*) seria necesario multiplicar los pesos de esta por el cociente entre los pesos de la negativa sobre los positivos, para conseguir que la suma de los pesos de las respectivas clases sean iguales. Sin embargo, empíricamente se ve que cuando las clases son altamente desbalanceadas suele dar mejores resultados tomar la raíz cuadrada (o similar) del cociente antes mencionado. Este valor es el que se le pasa al XGBoost como *scale_pos_weight*. Además, para tener en cuenta el peso de cada muestra se utiliza el parámetro *sample_weight* al momento de entrenar el modelo, para asi tener en consideración que no todas las muestras son igual de importantes.

Con este método se probaron los distintos preprocesamientos antes descriptos, realizando la búsqueda de hiperparametros tanto con una *Grid Search* por validación cruzada como haciendo uso del conjunto de validación.

Resultado

Como se mencionó anteriormente, utilizar *PolynomialFeatures* no resulta en una mejora del modelo y aumenta el costo computacional, por lo que simplemente se comentarán los resultados del modelo sin preprocesar y el único preprocesamiento con resultado positivo, el segundo descripto.

En la figura 4 se observa como los mejores modelos priorizan la *precision* por sobre el *recall*, minimizando la cantidad de falsos positivos. Las matrices de confusión en conjunto de entrenamiento no se incluyen, aunque tienen resultados muy similares a las de validación, indicando que los modelos no se encuentran sobreajustados.



(a) Matriz de confusión para data sin preprocesar

(b) Matriz de confusión para data procesada

Figura 4: Matrices de confusión en validación para distintos preprocesamientos de XGBoost

El cuadro 3 muestra los valores de la métrica AMS para ambos modelos, tanto en conjunto de entrenamiento como en el de validación. Notar que este modelo con preprocesamiento obtiene claramente el mejor resultado en validación hasta ahora.

Datos \ Conjunto	Entrenamiento	Validación
Sin preprocesar	5.31	1.08
Con preprocesamiento	4.04	1.19

Cuadro 3: Desempeño del XGBoost con métrica AMS, los datos sin y con preprocesar

4.3. Red Neuronal

4.3.1. Preprocesamiento

En el caso de la red neuronal, no se requirió de mucho preprocesamiento de los datos. Solamente se utilizó la media para llenar los datos faltantes y se estandarizaron todos los datos. La estandarización es importante en la red, ya que sin ella se desperdician varias *epochs* para que las primeras capas de la red se encarguen de regularizar los datos. También se probó elimnando las columnas con valores nulos, obteniendo un mejor resultado al utilizar todas las columnas. Es importante mencionar que en este caso no tiene sentido generar nuevas columnas a partir de las que se tienen, ya que la red debería ser capaz de realizar este tipo de transformaciones.

4.3.2. Arquitectura

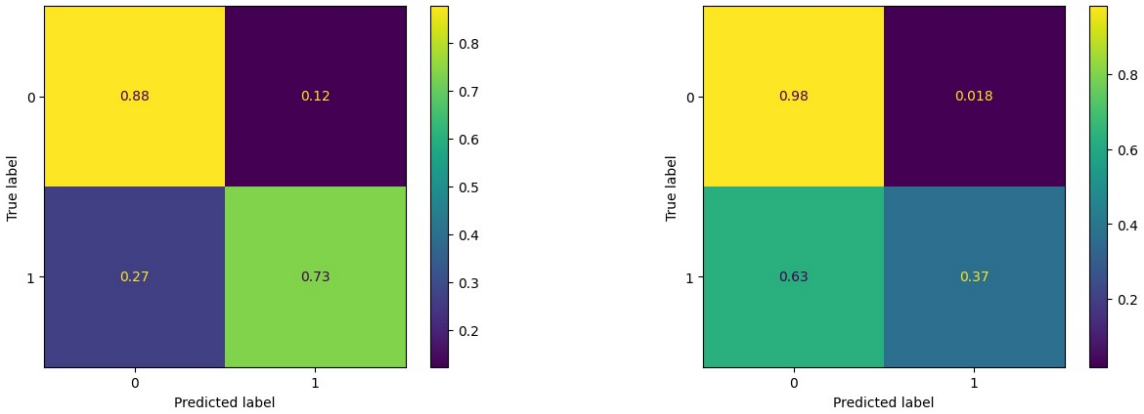
Si bien se probaron varias arquitecturas diferentes, todas se basaron en concatenar bloques de *Dense layers* y *batch normalization*, con distinta cantidad y tamaños de capas, siempre terminando con una ultima capa de clasificación (una sola dimensión de salida con un sigmoide). Sin embargo, como la red no estaba dando tan buenos resultados como se esperaba, se decidió probar algo diferente, usar la red como *features extractor*. La idea es entrenar una red neuronal para generar una mejor representación de los datos, para luego clasificarlos utilizando un *XGBoost*. Esto nace de la idea de que si bien la red es buena aprendiendo cómo representar los datos, no es tan buena a la hora de clasificarlos en comparación con un método de ensamble de modelos. Además, ya se vió que una buen preprocesamiento para *XGBoost* es generar nuevas columnas a partir de las otras, por lo que esto evita tener que hacerlo de forma manual. La arquitectura exacta del modelo puede observarse en el código, contando con un total de 37,089 parámetros entrenables.

4.3.3. Entrenamiento y Regularización

El modelo fue entrenado con la API de Keras de tensorflow. Todos los experimentos fueron entrenados por 1000 *epochs*, usando el *Adam optimizer* con un *learning rate* inicial de valor 0,01. Se utilizó un *scheduler ReduceLROnPlateau* para mejorar la convergencia de la *loss*. Además, se utiliza un checkpoint para guardar los pesos en la *epoch* que se tenga el mejor resultado en validación. Para regularizar el modelo se agrega regularización L1 y L2 en cada *layer*, además de Dropout 0.2 en la primer *layer*. Por el lado del XGboost, al igual que antes, se limita el *max_ depth* y el *n_ estimators*, esta vez con valores 3 y 120 respectivamente, hallados a través de una *grid search*.

4.3.4. Resultado

En la figura 5 se observan las matrices de confusión para ambos modelos de la red neuronal, mientras que en la tabla 4 los respectivos resultados de la metrica AMS. Notar como la red sin el *XGBoost* realiza una decisión más "justa", sin tener en cuenta la diferencia de las importancias entre las clases. En cambio, al agregar el *XGBoost*, si bien el *accuracy* disminuye, la precisión aumenta, al igual que el AMS, debido a tener en cuenta el desbalance entre clases.



(a) Matriz de confusión sin XGBoost

(b) Matriz de confusión con XGBoost

Figura 5: Matrices de confusión en validación para distintos modelos de la Red Neuronal

Modelo \ Conjunto	Entrenamiento	Validación
Sin XGBoost	2.659	0.826
Con XGBoost	4.518	1.101

Cuadro 4: Desempeño de la Red Neuronal con métrica AMS, con y sin mezcla del modelo XGBoost

5. Conclusión

Luego de abordar el problema con los distintos modelos, se optó por el que logre maximizar la métrica AMS en validación. Por lo tanto, se eligió el *XGBoost* con preprocesamiento, el cual fue reentrenado con todos los datos disponibles (entrenamiento y validación) para así obtener en test el desempeño de la figura 6, haciendo el submission de Kaggle.

Figura 6: Desempeño en conjunto de test