Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

Análisis de la Composición de Rayos Cósmicos mediante técnicas de Machine Learning usando eventos del Detector de Superficie del Observatorio Pierre Auger.

T E S I S

como requisito para la obtención del grado de

Licenciado en Física Aplicada

presenta

Andrea Corona Hernández

Asesores:
Dr. Enrique Varela Carlos
Dr. Humberto A. Salazar Ibargüen

Puebla, Puebla, México
Abril, 2025

Análisis de la Composición de Rayos Cósmicos mediante técnicas de Machine Learning usando eventos del Detector de Superficie del Observatorio Pierre Auger.

Andrea Corona Hernández

Comité

Dr. Iván Fuentecilla Carcamo
PRESIDENTE

Dr. Luis Manuel Villaseñor Cendejas
SECRETARIO

Dr.Jorge Velázquez Castro
VOCAL

Dr. Sebastián Rosado Navarro
SUPLENTE

Dr. Enrique Varela Carlos
ASESOR

Dr. Humberto Antonio Salazar Ibargüen
ASESOR

Agradecimientos

A mi familia.
A mamá, a papá y a mis hermanas (incluída Blacky), por ser mi hogar y mi sustento.

A mis asesores.
Por el apoyo académico y económico que me permitió crecer en el mundo de la ciencia.

Al CIIEC.
Por sentirse más que un espacio físico.

A mis amistades.
A Lupita, Moi, Jorge T. y Jorge A. porque desde aquel congreso en Chihuahua no han dejado de ser y estar para mí.

Índice general

Resumen

El principal objetivo de este trabajo consiste en desarrollar una metodología para predecir el parámetro X_{max} (profundidad atmosférica del máximo desarrollo de las lluvias) utilizando exclusivamente datos del Detector de Superficie (SD). Para ello, se implementó un modelo de aprendizaje automático basado en el algoritmo XGBoost, cuyo entrenamiento inicial se realizó con datos Golden que combinan información tanto del Detector de Fluorescencia (FD) como del Detector de Superficie (SD). Una vez optimizado el modelo, este fue almacenado y posteriormente aplicado para realizar predicciones empleando únicamente datos del SD, siguiendo los cortes energéticos y criterios de selección establecidos en el análisis. Los resultados obtenidos demuestran un comportamiento satisfactorio en las predicciones de $\langle X_{max} \rangle$, mostrando una notable concordancia con las tendencias físicas esperadas según los modelos teóricos y mediciones experimentales reportadas en la literatura. Este trabajo valida la eficacia del enfoque propuesto, destacando su potencial para superar las limitaciones operacionales del FD mediante el uso exclusivo de datos del SD, lo que representa un avance significativo en el análisis de rayos cósmicos de ultra alta energía, particularmente en condiciones donde la disponibilidad de datos de fluorescencia es limitada.

Introducción

X_{max} representa la profundidad atmosférica a la que una lluvia extensa, generada por la interacción de un rayo cósmico con la atmósfera terrestre, alcanza su máximo desarrollo. La predicción de X_{max} a altas energías (superiores de $10^{18.5}$) proporciona información sobre la energía y composición de los rayos cósmicos primarios, permitiendo saber información tanto sus propiedades como su interacción con la atmósfera.[36]

Históricamente, su determinación precisa ha requerido datos combinados de detectores de fluorescencia (FD) y superficie (SD). Los detectores de fluorescencia son altamente precisos, pese a eso, presentan limitaciones operativas significativas, como su dependencia de condiciones meteorológicas favorables (noches oscuras y despejadas) y su disponibilidad limitada durante el día. Estas restricciones reducen el volumen de datos disponibles y dificultan la obtención de mediciones continuas. Por otro lado, los detectores de superficie, han proporcionado estimaciones indirectas de X_{max} con mayor incertidumbre.[2] El contraste de ambos detectores, ha motivado la búsqueda de métodos alternativos que permitan estimar X_{max} de manera confiable utilizando datos provenientes exclusivamente de SD , maximizando la eficiencia.

El presente estudio propone un enfoque basado en aprendizaje automático (Machine Learning) para predecir X_{max} empleando exclusivamente datos del Detector de Superficie implementando un modelo XGBoost, entrenado inicialmente con datos Golden, que incluye mediciones simultáneas de FD y SD. Este entrenamiento le permite al modelo aprender las relaciones complejas entre las características observadas por los detectores de superficie y la profundidad atmosférica del máximo de la lluvia (X_{max}) proporcionada por el Detector de Fluorescencia. Una vez entrenado, el modelo se aplica para predecir $\langle X_{max} \rangle$ a partir de datos exclusivamente de SD y sus respectivos cortes, sin necesidad la necesidad de proporcionar información de FD.

Este planteamiento supera las limitaciones de la disponibilidad de datos de fluorescencia al optimizar el uso de los detectores de superficie para una estimación más accesible de X_{max}. Además, proporciona una alternativa para caracterizar eventos de rayos cósmicos, particularmente en condiciones donde los datos de FD son pocos. En este documento se detalla la metodología aplicada, se presentan los resultados obtenidos y se evalúa el rendimiento del modelo propuesto en términos de precisión, en comparación con enfoques tradicionales que integran información de ambos detectores.

Rayos cósmicos

Los rayos cósmicos son partículas que provienen del espacio exterior y han estado impactando continuamente la atmósfera terrestre[28], están compuestos principalmente por núcleos atómicos ionizados, donde aproximadamente el 90 % corresponden a protones (núcleos de hidrógeno), seguidos por núcleos de helio (9 %) y trazas de elementos más pesados como el hierro[15]. Un subconjunto de ellos, denominado rayos cósmicos de ultraalta energía, posee energías cinéticas que superan en órdenes de magnitud a las alcanzadas en colisionadores artificiales como el LHC, acelerándose hasta velocidades relativistas cercanas a la de la luz.[31] Su estudio es fundamental para comprender procesos astrofísicos y la composición de la materia interestelar.

1.1 Cascadas de partículas

Los rayos cósmicos primarios principalmente formado por protones energéticos tienen un espectro energético que abarca desde el rango de los TeV hasta energías ultraaltas superiores a 10^{20} eV. Al interactuar con los núcleos de nitrógeno y oxígeno de la atmósfera terrestre, estas partículas primarias generan cascadas de partículas secundarias, compuestas por fotones (γ), electrones, muones, neutrinos, hadrones, etc.[31] y a su vez producen luz fluorescente al interactuar con las moléculas de la atmósfera.[22]. Estas partículas secundarias producen nuevas interacciones con los átomos de la atmósfera, disipando progresivamente su energía hasta alcanzar un umbral crítico donde el proceso se detiene. Una sola cascada puede contener hasta $\sim 10^{11}$ partículas secundarias. En la Figura 1.1 se ilustra esquemáticamente la estructura de las componentes de estas cascadas atmosféricas.[31] Llegar a conocer la energía y dirección del rayo cósmico primario y cuál es la partícula que originó la cascada requiere de un análisis basado en simulaciones Monte Carlo del desarrollo de las cascadas, sin embargo, existe una considerable incertidumbre en este tipo de simulaciones para altas energías.[33]

1.2 Composición de rayos cósmicos ultra energéticos

La determinación de la composición de los rayos cósmicos de ultra alta energía es fundamental para entender su origen, las fuentes que los producen y los mecanismos de propagación a través del medio interestelar y extragaláctico. Para energías inferiores a 100 TeV, la detección directa revela que los rayos cósmicos primarios están dominados por protones. Sin embargo, para energías superiores a 100 TeV, la detección debe realizarse de manera indirecta a través del análisis de lluvias atmosféricas extensas, lo que introduce incertidumbres adicionales debido a la dependencia de los modelos de interacción hadrónica utilizados en la reconstrucción de los eventos.

Los modelos de interacción hadrónica describen las interacciones entre los núcleos pri-

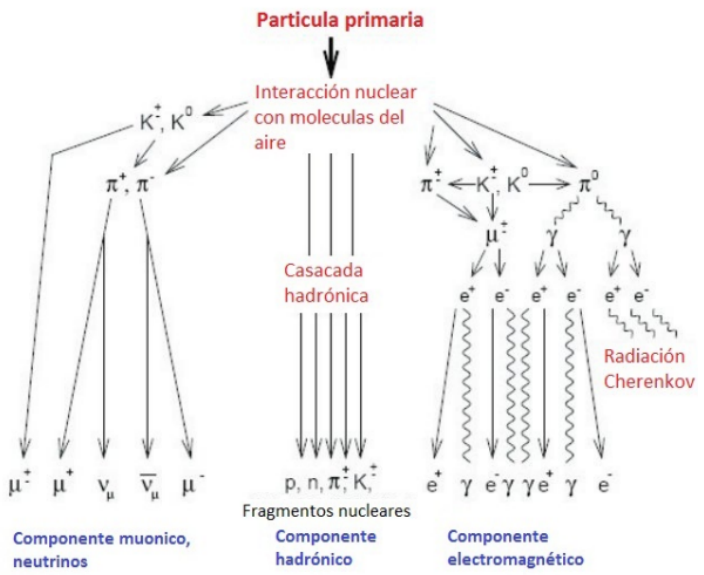


Figura 1.1: Componentes de una EAS (Extended Air Shower) generadas por la interacción de los rayos cósmicos con las capas altas de la atmósfera terrestre.

marios y la atmósfera terrestre.Entre los modelos más empleados se encuentran EPOS-LHC, QGSJET-II-04 y Sibyll 2.3. EPOS-LHC es un generador de eventos Monte-Carlo basado en la teoría Parton-Based Gribov Regge, diseñado para minimizar sesgos en la descripción de interacciones hadrónicas. Sibyll 2.3 combina el modelo minijets con el modelo de cuerda Lund para describir la producción de partículas secundarias y QGSJET-II-04 se basa en la teoría de campo Reggeon, modela las interacciones hadrónicas a través de múltiples dispersores partónicos. Estos tres modelos han sido calibrados y mejorados utilizando datos experimentales obtenidos en el Gran Colisionador de Hadrones (LHC), permitiendo optimizar su precisión en la descripción de las interacciones a altas energías.[36] Un parámetro clave para determinar la composición de los rayos cósmicos a partir de las lluvias atmosféricas es X_{max}.

1.2.1 Profundidad atmosférica del máximo desarrollo de partículas (X_{max})

La cascada producida al interaccionar una partícula con la atmósfera crece hasta alcanzar un número máximo de partículas, pero luego comienza a disminuir porque la energía se pierde principalmente por ionización. La profundidad atmosférica donde ocurre este máximo desarrollo se llama X_{max}, y se mide desde el borde de la atmósfera en unidades de gramos por centímetro cuadrado (g/cm^2). El X_{max} es medido directamente con los Detectores de Fluorescencia, y está relacionado de manera directa con la masa de la partícula primaria. Una cascada generada por un protón tiene un X_{max} mayor que una iniciada por un núcleo más pesado como el hierro.[16]

Las mediciones del Observatorio Pierre Auger muestran un cambio en la tendencia de X_{max} cerca de 3×10^{18} eV, indicando una transición de composición ligera a más pesada

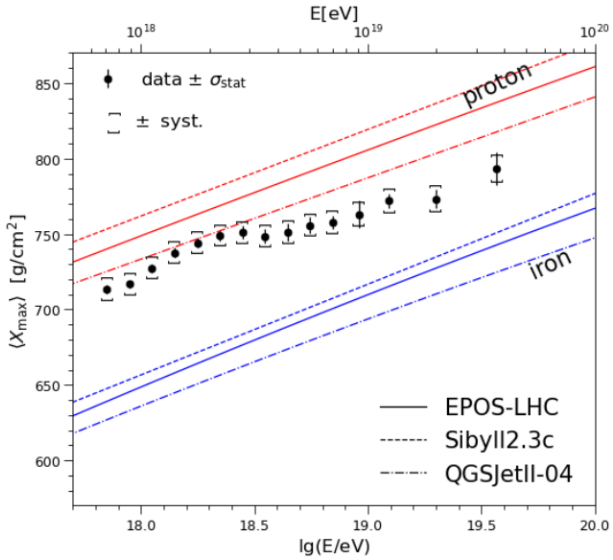


Figura 1.2: Medidas de X_{max} en el Observatorio Pierre Auger comparadas con los valores obtenidos por los modelos de interacción hadrónica EPOS-LHC, Sibyll 2.3c y QGSJETII-04. [5]

con el aumento de energía.[5]

X_{max} presenta una dependencia lineal con el logaritmo de la energía por nucleón:

$$X_{max} \propto ln(E/A)$$

donde E es la energía total del primario y A su número másico. De esta forma, las lluvias de partículas generadas por núcleos primarios pesados se desarrollan más rápidamente que las originadas por primarios ligeros, alcanzando su profundidad máxima a menores altitudes. En contraste, cuanto más ligera es la partícula primaria, mayor es la profundidad atmosférica esperada para el desarrollo máximo del chubasco.[16]

Observatorio Pierre Auger

El Observatorio Pierre Auger ubicado en la ciudad Malargüe, Mendoza Argentina pretende descubrir y comprender la fuente de los rayos cósmicos de mayor energía. La Colaboración Pierre Auger, formada por científicos de 18 países, diseñó el Observatorio para un estudio estadístico y poder determinar la energía y la dirección de llegada de cada rayo cósmico. [12] Este Observatorio es el primero diseñado para funcionar como un detector híbrido. Se combinan dos técnicas complementarias para la detección de las lluvias utilizando detectores de fluorescencia (FD) y de superficie (SD), el arreglo general puede verse en la figura 2.1. Los SD operan de forma continua, midiendo las densidades de partículas a medida que la lluvia de partículas impacta el suelo justo después de su máximo desarrollo. En noches oscuras, los FD registran el desarrollo de la lluvia de partículas mediante la cantidad de fluorescencia de nitrógeno producida a lo largo de su trayectoria.[12]

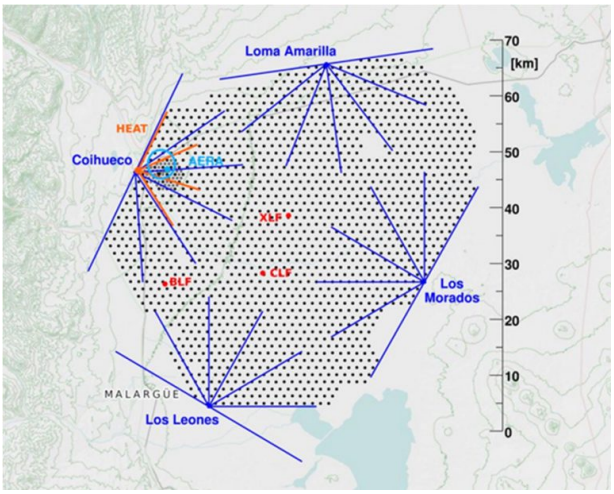


Figura 2.1: Mapa del arreglo de detectores de florescencia y superficie. Cada punto representa un SD ubicado a 1.5 km entre ellos y se encuentran rodeados de los cinco FD. También se muestran las dos instalaciones láser, CLF y XLF, cerca del centro del Observatorio.[4]

2.1 Detector de Fluorescencia

El Detector de Fluorescencia (FD) del Observatorio Pierre Auger está compuesto por cinco sitios de observación: Los Leones, Los Morados, Loma Amarilla, Coihueco y HEAT. Cada sitio cuenta con seis bahías (excepto Coihueco, que incluye tres detectores adicionales para HEAT), donde telescopios con campos de visión de 30° × 30° en acimut y elevación se orientan hacia el interior del arreglo de detectores de superficie (SD), logrando una cobertura combinada de 180° en acimut [3]. Estos detectores registran la fluorescencia ul-

travioleta producida cuando las partículas secundarias de los rayos cósmicos (con energías del orden de 10^{11} partículas) excitan moléculas de aire, emitiendo luz equivalente a $\backsim$25 W durante $\backsim$ 10^{-5} segundos, convirtiendo un 0.005 % de la energía del rayo cósmico en señal detectable [29].

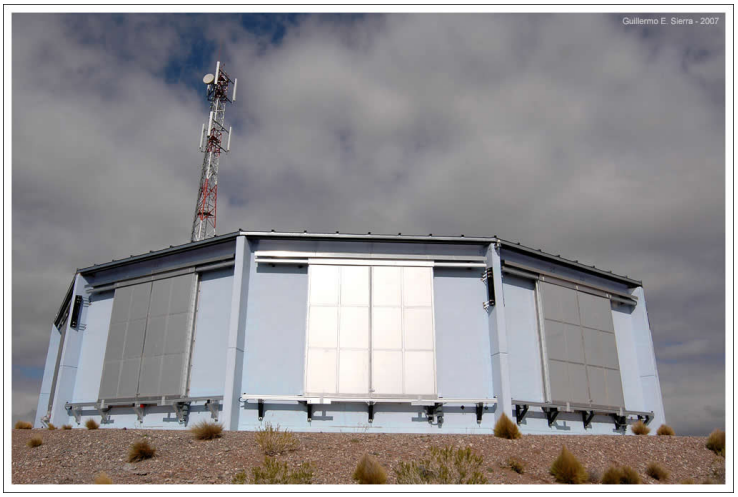


Figura 2.2: Detector de Fluorescencia en el observatorio Pierre Auger.[29]

2.1.1 Limitaciones del Detector de Flourescencia

Estos detectores deben cerrarse durante el día y también se cierran automáticamente por la noche cuando el viento es demasiado fuerte o se detecta lluvia[12], por lo que el funcionamiento de estos detectores depende de la hora del día, las condiciones atmosféricas y que haya noches sin Luna. Según las estimaciones, el tiempo de funcionamiento del FD representa de un 10 a un 15 por ciento del tiempo de SD.[17]

2.2 Detector de Superficie

El Detector de Superficie (SD) consta de 1660 detectores Cherenkov de agua (WCD) ubicados en una cuadrícula triangular con una separación de 1.5 km y una superficie de aproximadamente 3000 km^2.[12] Cada detector está compuesto por un revestimiento sellado de 3.6 m de diámetro y 1.2 m de altura, estos registran la radiación Cherenkov -luz ultravioleta emitida cuando partículas relativistas (con velocidades superiores a la de la luz en agua) atraviesan los tanques de agua ultrapura (12,000 L por detector). Cada tanque cuenta con tres fotomultiplicadores (PMT) de 23 cm que detectan los tenues pulsos de luz Cherenkov (amplificados mediante emisión secundaria de electrones) y un recubrimiento interno reflectante para optimizar la captación. La señal se transmite inalámbricamente a estaciones colectoras y luego al centro de datos en Malargüe, donde se reconstruye la dirección del rayo cósmico primario con precisión de $\backsim$20 ns (vía GPS) y su energía mediante comparación con simulaciones computacionales, analizando coincidencias entre

múltiples detectores.[30]

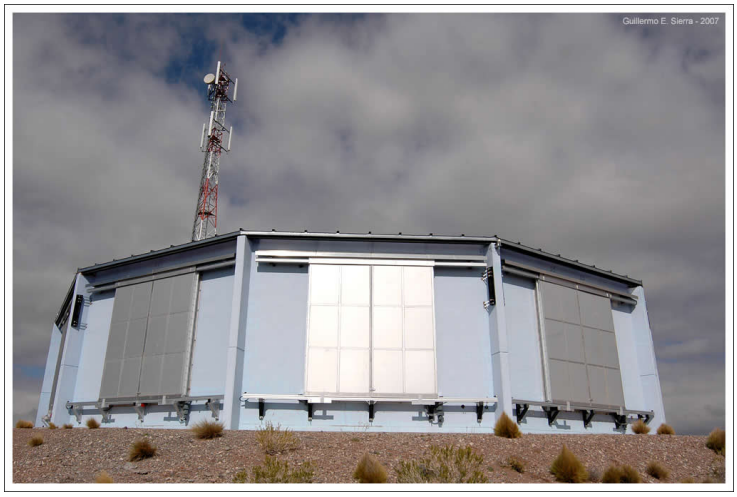


Figura 2.3: Detector de Superficie en el observatorio Pierre Auger.[30]

Aprendizaje Automático (Machine Learning)

3.1 Introducción al Aprendizaje Automático

Hace no mucho tiempo, las interacciones con sistemas inteligentes como solicitar direcciones a un teléfono parecían propias de la ciencia ficción. Hoy en día, el Aprendizaje Automático (Machine Learning) es una realidad cotidiana, el cuál es encontrado en diversas aplicaciones, aunque sus fundamentos se remontan a décadas atrás en dominios especializados, como el reconocimiento óptico de caracteres (OCR). Uno de los primeros casos de éxito global fue el filtro de spam en los años 90: un sistema que, sin ser autónomo, aprende de los datos hasta minimizar la intervención del ser humano. Desde entonces, el Aprendizaje Automático impulsa innumerables funcionalidades, desde traducción automática hasta recomendaciones personalizadas.[18]

3.1.1 ¿Qué es Machine Learning?

El Aprendizaje Automático (Machine Learning) es una rama de la Inteligencia Artificial que le concede a las máquinas aprender a partir de información que ya existe para aplicar ese conocimiento en la ejecución de tareas semejantes. Al recibir los datos, la maquina aprende he identifica patrones, tendencias o características esenciales de datos previos y utiliza ese aprendizaje para hacer predicciones sobre nuevos datos.[21]

3.1.2 Tipos de Aprendizaje Automático

Aprendizaje Supervisado

El aprendizaje supervisado se emplea cuando los datos consisten en variables de entrada y valores objetivo de salida. El algoritmo aprende la función de mapeo que relaciona las entradas con las salidas. La necesidad de disponer de grandes volúmenes de muestras etiquetadas lo convierte en un método costoso para tareas con escasez de datos.[35] Estos enfoques pueden catalogarse en dos categorías principales: clasificación y regresión. Cuando la variable objetivo que intentamos predecir es continua, se trata de un problema de regresión. Cuando solo puede tomar un número reducido de valores discretos, se denomina problema de clasificación.[27]

Aprendizaje No Supervisado

El aprendizaje no supervisado es utilizado cuando los datos solo están disponibles en forma de variables de entrada, sin contar con una variable de salida correspondiente. Estos algoritmos modelan los patrones que están implícitos en los datos para extraer información

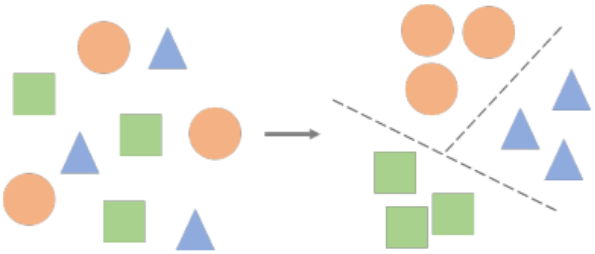


Figura 3.1: Descripción general del aprendizaje supervisado. Las entradas se clasifican en un conjunto conocido de clases.[35]

relevante sobre sus características. El clustering (agrupamiento), es una técnica que identifica grupos en los datos y los emplea para predecir la salida de entradas no observadas.[35]

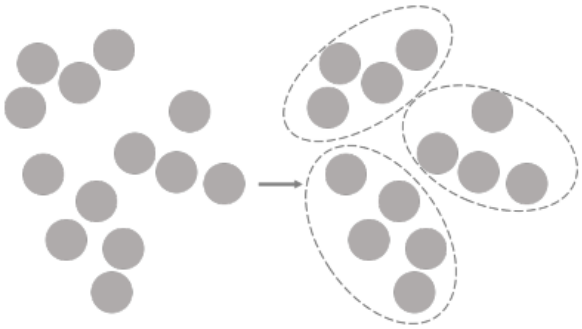


Figura 3.2: Descripción general del aprendizaje no supervisado. Las muestras de entrada se agrupan en grupos según los patrones subyacentes.[35]

Aprendizaje semi-supervisado

Como su nombre lo indica, representa un punto intermedio entre el aprendizaje supervisado y no supervisado. El entrenamiento de estos algoritmos se basan en la mezcla de datos etiquetados y no etiquetados, donde usualmente se dispone de un volumen reducido de ejemplos etiquetados junto con una gran cantidad de datos sin etiquetar.[35] El procedimiento consiste en agrupar los datos relevantes con técnicas de aprendizaje no supervisado, y usar las muestras etiquetadas proporcionadas para inferir la etiqueta de los otros datos.

3.1.3 Machine Learning en la Física de Partículas

Simultáneamente al auge de las técnicas de machine learning en aplicaciones industriales, la comunidad científica está participando más en la explotación del aprendizaje automático para la investigación, siendo la física un caso relevante. La física busca comprender los mecanismos fundamentales de la naturaleza, privilegiando modelos interpretables basados en el conocimiento teórico y la intuición humana, los científicos pueden emplear técnicas

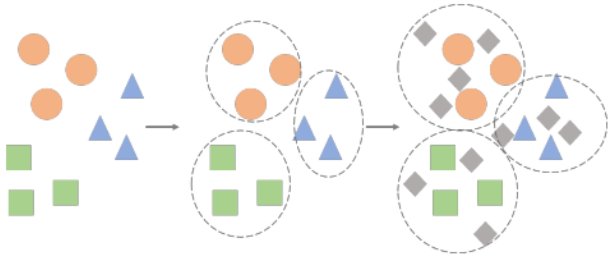


Figura 3.3: Descripción general del aprendizaje semi-supervisado. Los clústeres formados por una gran cantidad de datos sin etiquetar se utilizan para clasificar una cantidad limitada de datos etiquetados.[35]

de machine learning como herramientas complementarias para resolver problemas complejos, conservando la capacidad de interpretar los resultados mediante marcos teóricos establecidos. Estas disciplinas funcionan ya que ambas comparten metodologías y objetivos comunes: la recolección y análisis de datos para construir modelos predictivos de sistemas complejos.[8]

El Machine Learning se ha utilizado en física de altas energías durante más de una década marcando un cambio importante en el alcance de las investigaciones. La combinación de principios físicos con ideas innovadoras del aprendizaje estadístico será fundamental para analizar grandes volúmenes de datos y, poder reconocer la estructura de la naturaleza, algunos de esos ejemplos son experimentos del Gran Colisionador de Hadrones (LHC), búsquedas de eventos raros en materia oscura y detectores de neutrinos y rayos cósmicos. Los métodos supervisados de machine learning se emplean ampliamente para identificar partículas conocidas y diseñar búsquedas específicas de teorías de nueva física. Por otro lado, enfoques menos supervisados permiten realizar búsquedas menos dependientes de un modelo de señal específico.[23]

3.2 Árboles de Decisión y Métodos de Ensamble

3.2.1 Concepto de Árboles de Decisión

El árbol de decisión comienza con un nodo raíz, y se divide en el nodo de decisión donde se verifica y se toma una decisión y en el nodo hoja se obtiene un conjunto de datos con las mismas características y no se le pueden realizar divisiones adicionales y representan todos los resultados posibles dentro del conjunto de datos.

El algoritmo del árbol de decisión consta de dos tipos de nodos: el nodo de decisión y el nodo hoja, como se muestra en la Figura 3.4. Un nodo de decisión es un punto donde se verifica una condición y se toma una decisión, mientras que un nodo hoja es un punto donde se obtiene un conjunto de datos puro (es decir, con las mismas características), y no se pueden realizar divisiones adicionales.

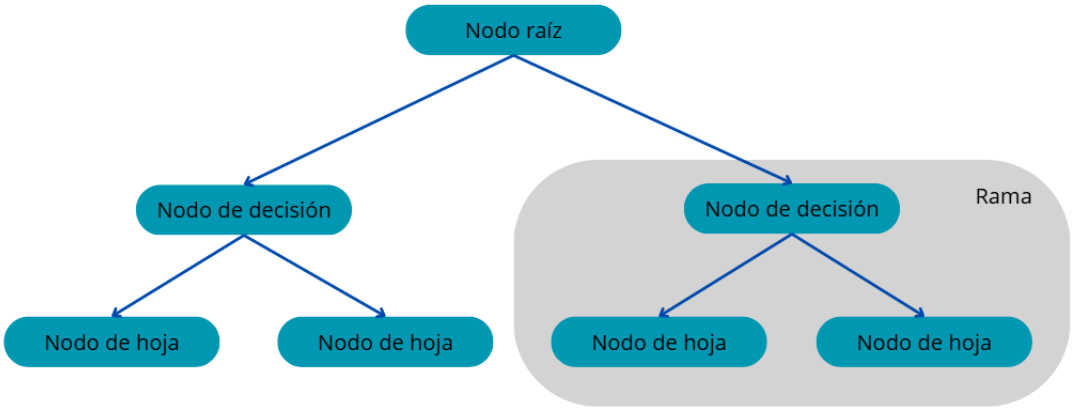


Figura 3.4: Esquema de un árbol de decisión

3.2.2 Método de Ensamble: Boosting

El Boosting surgió de un planteamiento teórico sobre la posibilidad de transformar cualquier algoritmo débil de clasificación definido como aquel que supera ligeramente el rendimiento de una elección aleatoria (>50 % de precisión en problemas binarios) en un clasificador fuerte capaz de alcanzar una precisión casi perfecta (ej. 99 %). Este término es abordado inicialmente por Schapire (1990) y Freund (1995), tenía gran relevancia práctica, dado que diseñar aprendices fuertes directamente resulta complejo, mientras que construir versiones débiles es considerablemente más sencillo. La respuesta, que fundamentó el desarrollo del boosting, demostró que cualquier aprendizaje base débil puede mejorarse iterativamente (boostearse) hasta alcanzar un rendimiento fuerte. Este principio teórico no solo resolvió un problema fundamental en aprendizaje automático, sino que también impulsó la creación de los primeros algoritmos de boosting, estableciendo las bases para métodos posteriores como AdaBoost y Gradient Boosting.[26]

3.3 XGBoost: Principios y Funcionamiento

3.3.1 Introducción a XGBoost

XGBoost fue creado por Tianqi Chen y Carlos Guestrin, estudiantes de doctorado en la Universidad de Washington en el 2014 y su lanzamiento al público en general fue en el 2016.[10] XGBoost (eXtreme Gradient Boosting) es un algoritmo de aprendizaje automático basado en el principio de Gradient Boosting(potenciación del gradiente), que forma parte del campo del aprendizaje supervisado, proporciona un alto rendimiento en tareas de clasificación, regresión y ranking[26]. XGBoost está diseñado para tener un alto rendimiento predictivo, buena capacidad para gestionar datos dispersos, gran implementación multinúcleo y eficientemente optimizado. [25]

3.3.2 Cómo Aprende XGBoost

1. Inicialización:
   Se comienza con una predicción inicial (usualmente la media de la variable objetivo en regresión o la probabilidad inicial en clasificación). Ejemplo para regresión:

$$F_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$$

   Donde L es la función de pérdida (ej: error cuadrático).

2. Construcción iterativa de árboles:
   En cada iteración t (para $t = 1$ a T):

   (a) Cálculo de residuos:
   Para cada muestra i, se calcula el residuo (error) entre la predicción actual y el valor real:

$$r_{it} = -\left[\frac{\partial L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)}\right]$$

   (Gradiente negativo de la función de pérdida).

   (b) Entrenamiento de un nuevo árbol:
   Se entrena un árbol f_t para predecir los residuos (no los valores originales). El árbol divide los datos en M nodos terminales (hojas), cada uno con un peso óptimo w_j.

   (c) Optimización de la estructura del árbol:
   XGBoost usa una pérdida regularizada para elegir la mejor división en cada nodo:

$$\text{Ganancia} = \frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} - \gamma$$

   Donde: - g_i y h_i son gradientes y hessianos de la pérdida. - λ (regularización L2) y γ (complejidad) evitan sobreajuste.

   (d) Actualización del modelo:
   Las predicciones se actualizan sumando el nuevo árbol escalado por una tasa de aprendizaje (η):

$$F_t(x) = F_{t-1}(x) + \eta f_t(x)$$

3. Predicción final:
   Después de T iteraciones, el modelo final es la suma de todos los árboles:

$$F(x) = \sum_{t=1}^{T} \eta f_t(x)$$

3.3.3 Parámetros Clave en XGBoost

En XGBoost debemos configurar tres tipos de parámetros[14]:

- **Parámetros generales:** Definen el tipo de booster, como árbol o modelo lineal.
- **Parámetros del booster:** Específicos según el tipo seleccionado
- **Parámetros de tarea:** Determinan el objetivo de aprendizaje, como regresión o ranking, cada uno con ajustes particulares

Enfatizando en nuestra investigación, el tipo de booster a utilizar es "Tree Boosterz dentro de este podremos modificar parámetros importantes tales como:

- `eta`[valor predeterminado=0.3, alias: `learning_rate`]: Reducción del tamaño de paso utilizada en la actualización para evitar el sobreajuste. Tras cada paso de refuerzo, podemos obtener directamente los pesos de las nuevas características, y eta reduce los pesos de las características para que el proceso de refuerzo sea más conservador, puede tomar valores de $[0, 1]$.
- `max_depth`[valor predeterminado=6]:Profundidad máxima de un árbol. Si se aumenta este valor hará que el modelo sea más complejo y más propenso a sobreajustarse y enter mayor sea se va a consumir mñas memoria al entrenar el árbol, el número debe estar en un rango de $[0, \infty]$.
- `subsample`[valor predeterminado=1]:Relación de submuestreo de las instancias de entrenamiento. Establecerla en 0.5 significa que XGBoost muestreará aleatoriamente la mitad de los datos de entrenamiento antes de generar árboles, lo que evitará el sobreajuste. El submuestreo se realizará una vez en cada iteración de refuerzo, puede tomar valores de $(0, 1]$.
- `colsample_bytree`: Es la proporción de submuestras de columnas al construir cada árbol. El submuestreo se realiza una vez por cada árbol construido.
- `alpha`[valor predeterminado=0, alias: `reg_alpha`]: Término de regularización L1 en ponderaciones. Aumentar este valor hará que el modelo sea más conservador, el número debe estar en un rango de $[0, \infty]$.
- `lambda`[valor predeterminado=1, alias: `reg_lambda`]: Término de regularización L2 en ponderaciones. Aumentar este valor hará que el modelo sea más conservador, el número debe estar en un rango de $[0, \infty]$.
- `objective:` Usa como valor predeterminado `reg:squarederror` que es la regresión con pérdida al cuadrado.
- `eval_metric:` La métrica de evaluación para los datos de validación es asignada según el objetivo, para regresión se usa `rmse` y para clasificación `logloss`.
- `seed:` Semilla con un número entero aleatorio.

3.3.4 Limitaciones de XGBoost

Pese a su eficiencia superior frente a otras alternativas de Gradient Boosting, XGBoost presenta un alto costo computacional, hay tareas comunes que pueden demorar horas o incluso días y es necesario ejecutar numerosas veces para explorar el efecto de parámetros como la tasa de aprendizaje y los términos de regularización L1/L2 para maximizar la precisión en validación cruzada.[26]

Métodos para reconstruir X_{max}

Anteriormente, la determinación de X_{max} dependía exclusivamente de las mediciones del detector de fluorescencia. Por lo que usar detectores de superficie representa una solución viable para mejorar las estadísticas en estudios de composición de rayos cósmicos ultra-energéticos. Sin embargo, a diferencia de FD que miden directamente el perfil longitudinal de la lluvia atmosférica, los SD solo registran un muestreo espacial de la densidad de partículas en tierra. Para superar esta limitación, es necesario analizar los patrones temporales en la distribución de partículas secundarias, los cuales contienen información implícita sobre el desarrollo de la cascada atmosférica.

4.1 Usando datos híbridos

Como se ha establecido, el Observatorio Pierre Auger utiliza un sistema de detección híbrida (FD + SD) para mejorar la precisión en la reconstrucción de eventos. Este proceso se divide en tres fases: la primera se centra en la reconstrucción geométrica, posteriormente en la reconstrucción del perfil de la lluvia y finalmente el cálculo de la energía primaria (E) del rayo cósmico, siendo estas dos últimas basadas predominantemente en el FD. La reconstrucción geométrica incluye los datos de tiempo obtenidos por las estaciones SD que detectaron señal de esta lluvia.

La precisión con la que se realiza el primer paso de la reconstrucción es importante debido a su implicancia directa en la búsqueda de fuentes de rayos cósmicos, su influencia en el cálculo de E y de la profundidad máxima de la lluvia (X_{max}), así como su aplicación en la calibración de las lluvias detectadas en el SD.

4.2 Usando distribuciones temporales

Según M. Ave y M. Roth y A. Schulz en su investigación llamada Ün descripción generalizada de las señales dependientes del tiempo en detectores de lluvia de aire extensivos y sus aplicaciones"[6], para reconstruir (X_{max}), buscan analizar cómo llegan las partículas secundarias al suelo en diferentes tiempos. El modelo aprovecha estas diferencias temporales para estimar X_{max}. Primero, divide las partículas detectadas en cuatro grupos principales: muones, partículas electromagnéticas puras, partículas electromagnéticas producidas por muones, y partículas residuales de baja energía. Luego, compara cómo se distribuyen en el tiempo estas partículas con patrones precalculados basados en simulaciones. Si la lluvia alcanza su máximo desarrollo (X_{max}) más cerca del suelo, las partículas electromagnéticas llegan con menos retraso; si el máximo ocurre más arriba, el retraso es mayor. Esta técnica permite reconstruir X_{max} con una precisión de unos 45 g/cm^2 para lluvias de 10^{19} eV. Aunque no puede determinar todo el perfil de la lluvia, es más preciso que métodos anteriores que solo usaban el tiempo de subida de las señales, ya que considera toda la

distribución temporal de las partículas.

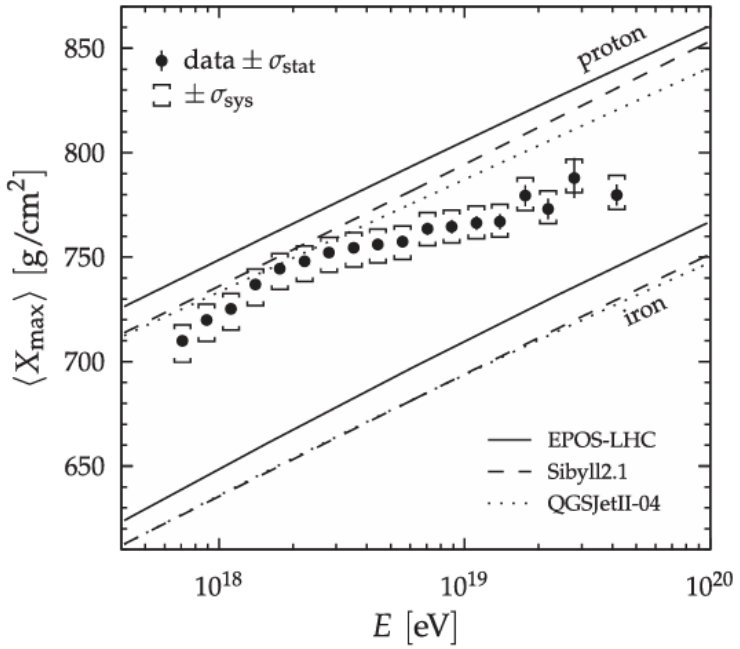


Figura 4.1: Evolución energética de la profundidad media máxima de la lluvia $\langle X_{max} \rangle$ predicha.[1]

4.3 Usando Aprendizaje Profundo

En uno de los estudios más recientes, Abdul Halim et al. en "Medición de la profundidad máxima de perfiles de lluvias de aire con energías entre $10^{18,5}$ y 10^{20} eV utilizando el detector de superficie del Observatorio Pierre Auger y aprendizaje profundo"[2] se investiga por primera vez la composición en masa de los rayos cósmicos de ultra alta energía utilizando el primer y segundo momento de las distribuciones de X_{max} en un rango de 3 a 100 EeV, empleando un nuevo método de reconstrucción basado en aprendizaje profundo. Se propuso un modelo de red neuronal profunda (DNN), donde se extrae características capa por capa y combina características de bajo nivel para formar características de alto nivel, que pueden encontrar una expresión distribuida de datos.[37] Dicho modelo fue entrenado con datos híbridos y aplicado a datos de SD para X_{max}. Su arquitectura combinó capas recurrentes (LSTM) para procesar la información temporal de las señales en los detectores de agua Cherenkov y capas convolucionales (CNN) para extraer patrones espaciales en la distribución de partículas. Los gráficos de interés son:

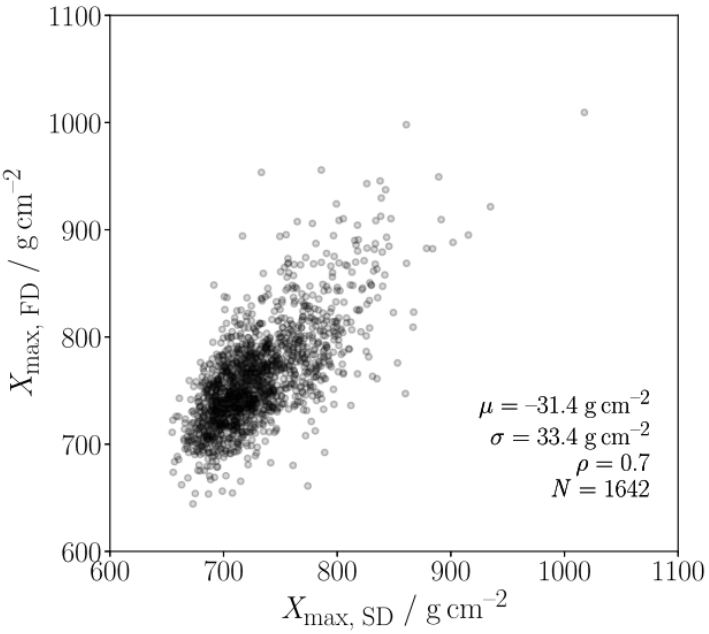


Figura 4.2: Correlación entre las observaciones de FD y las predicciones de DNN utilizando datos de SD.[2]

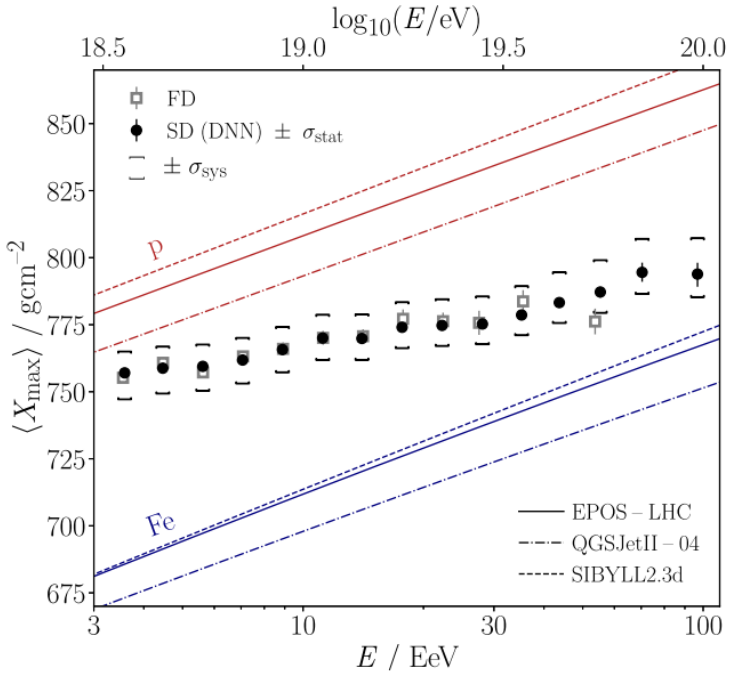


Figura 4.3: Evolución energética de la profundidad media máxima de la lluvia $\langle X_{max} \rangle$ predicha.[2]

Métricas de evaluación y análisis estadístico en Modelos de Regresión

De acuerdo con Chicco et al.[11], un un modelo de regresión, X_i representa la predicción asociada a la i-ésima observación, mientras que Y_i denota su valor real en los datos de entrenamiento. El algoritmo estima cada X_i en función de Y_i, perteneciente al conjunto de verdades fundamentales en un número total de muestras(n). Para las siguientes formulas, se va a considerar:

- Media de los valores reales

$$\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i \tag{5.0.1}$$

- Media Cuadrática Total: Es una medida de la variabilidad total en un conjunto de datos para evaluar la dispersión de los datos alrededor de la media global.

$$MST = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y})^2 \tag{5.0.2}$$

5.1 Error cuadrático medio (MSE)

Es una métrica que cuantifica la diferencia entre los valores predichos por un modelo $\hat{X}_i$ y los valores reales Y_i.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{X}_i - Y_i)^2 \tag{5.1.1}$$

El MSE se puede utilizar para detectar si hay valores atípicos, ya que atribuye pesos mayores a dichos puntos. Un MSE = 0 implica predicciones perfectas, por lo que se sugiere tener valores cercanos a cero.

5.2 Raíz del Error Cuadrático Medio(RMSE)

Representa la desviación estándar de los errores de predicción (residuos), midiendo cuánto se desvían, en promedio, las predicciones $\hat{X}_i$ de los valores reales Y_i.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{X}_i - Y_i)^2} \tag{5.2.1}$$

De manera análoga, se pretende que el valor sea cercano a cero.

5.3 Coeficiente de determinación (R^2)

El Coeficiente de determinación (R^2) El coeficiente de determinación (Wright, 1921) puede interpretarse como la proporción de la varianza en la variable dependiente que es predecible a partir de las variables independientes y puede tomar valores en el rango de $(-\infty, 1]$.

$$R^2 = 1 - \frac{MSE}{MST} = 1 - \frac{\sum_{i=1}^{n}(X_i - Y_i)^2}{\sum_{i=1}^{n}(\overline{Y} - Y_i)^2} \tag{5.3.1}$$

donde Veamos los casos principales:

- $R^2 \geqslant 0$:
  Para modelos de regresión lineal sin restricciones, R^2 es siempre no negativo y para un ajuste completamente lineal el valor es de 1.

- $R^2 = 0$:
  Ocurre cuando el modelo de regresión horizontal, es decir que su pendiente sea cero. En el caso de dos variables numéricas, esto implica que son independientes y que no están correlacionadas.

- $R^2 < 0$:
  Se encuentra en casos concretos donde se presenta una regresión lineal con restricciones, que se presente una regresión no lineal o al forzar la regresión a pasar por el origen $(0, 0)$ sin intercepto puede generar R^2 negativos si los datos no están centrados en cero

5.4 Varianza(σ^2)

Para una población de tamaño N, σ^2 es una medida de dispersión de los valores o datos X_i con respecto a $\overline{X}$.[9]

$$\sigma^2 = \frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{N} \tag{5.4.1}$$

5.5 Desviación Estándar(σ)

A diferencia de (σ^2), σ está expresada en las mismas unidades que los datos originales, lo que la hace más intuitiva para interpretar.

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{N}} \tag{5.5.1}$$

5.6 Error estándar de la media(SEM)

El error estándar de la media refleja cuánto varían las medias muestrales respecto a la media poblacional, dependiendo de la variabilidad de la población y del tamaño de la muestra. El SEM indica qué tan precisa es la media muestral como estimación de la media poblacional, ya que si la población tiene alta dispersión, las medias muestrales diferirán más entre sí, aumentando el SEM; en cambio, al tomar más muestras, estas medias se acercarán a la media real, reduciendo el SEM. [24]

$$SEM = \frac{\sigma}{\sqrt{n}} \tag{5.6.1}$$

5.7 Coeficiente de Pearson (ρ)

El coeficiente de correlación poblacional de Pearson cuantifica la relación lineal entre dos variables X y Y, estandarizando su covarianza para evitar interpretaciones sesgadas por diferencias en unidades de medida. Su fórmula es

$$\rho_{XY} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{5.7.1}$$

donde σ_X y σ_Y son las desviaciones estándares de X y Y, respectivamente. E representa la esperanza matemática y μ_X es la media poblacional de X y μ_Y la de Y. ρ oscila entre -1 (correlación negativa perfecta) y 1 (correlación positiva perfecta), con 0 indicando ausencia de relación lineal.[19]

Implementación del modelo usando datos híbridos (Golden)

6.1 Datos y Preprocesamiento

Para la adquisición de datos, se extrajo información de la producción de datos del Observatorio Pierre Auger, específicamente del repositorio denominado "The Observer Realm"[32]. Con el fin de robustecer el proceso, se implementó la ejecución paralela de múltiples jobs, así se logró adquirir archivos Golden, que contiene información tanto del SD como del FD, las fechas de recolección abarcan desde el 01 enero de 2004 hasta el 31 de diciembre de 2022 al cual se le realizaron cortes específicos(Cuadro 6.1). Los archivos extraídos Golden son de tipo ROOT, ya que tienen una terminación `.root`. ROOT sirve para el análisis de datos a gran escala, está escrito en C++ y funciona como una eficiente base de datos jerárquica[7], le permite a los usuarios almacenar datos en forma de ramas, y se usa la librería `uproot` para leer los archivos en Python.[34](Ver código 6.1)

Cortes	Eventos	ε(%)
Número total	224,556	100.0 %
Iluminación	224,556	99.8 %
Nivel de reconstrucción mínima	224,107	100.0 %
Máximo ángulo cenital (60º)	224,107	96.3 %
T4Trigger	215,816	100.0 %
T5Trigger	215,816	83.4 %
T5TriggerUB	179,991	100.0 %
$log_{10}(E/EV)$ >17.0	176,752	98.2 %
Archivo de rechazo de períodos malos	172,510	97.5 %

Cuadro 6.1: Selecciones básicas y específicas del análisis para el conjunto de datos Golden.

```
import uproot
import pandas as pd
rdata = uproot.open("E:/Golden/todosGolden.root")
rdata.keys()
rdatas = rdata["recData"]

todas_ramas = rdata["recData"].keys()
print("Ramas:")
for rama in todas_ramas:
    print(rama)
```

Listing 6.1: Lectura de una rama recData"de un archivo .root usando uproot

Para un mejor manejo y organización de los datos, se segmentó la base de datos original, dividiéndola en dos subconjuntos: una base correspondiente a FD y otra a SD. Debido a que el tratamiento metodológico aplicado a ambas bases es análogo, en este contexto solo se detallará el procedimiento correspondiente a FD, entendiendo que el enfoque para SD sigue la misma lógica de procesamiento, filtrado y análisis, garantizando así consistencia en el manejo de ambos conjuntos de datos.

```
# Filtra las ramas que contienen "event.fFDEvents"
ramas_fFDEvents = [rama for rama in todas_ramas if "event.fFDEvents" in
    rama]
# Muestra las ramas divididas
print("Ramas que contienen 'event.fFDEvents':")
for rama in ramas_fFDEvents:
   print(rama)
```

Y para generar los DataFrames correspondientes, se usó:

```
print ("DataFrame de FD:")
for subrama in ramas_fFDEvents:
    nombre_variable = subrama.split('.')[-1]
    # Crear el DataFrame con los datos de la subrama y asignarlo a la
    variable
    datos[nombre_variable] = pd.DataFrame(rdatas[subrama].array(library=
    "pd"))
    print(f"FD_{nombre_variable} = pd.DataFrame(rdatas['{subrama}'].
    array(library='pd'))")
```

Listing 6.2: Código para generar lo que va dentro del DataFrame

```
#Creacion del DataFrame completo
dataframeFD = {
    "FD_fUniqueID": pd.DataFrame(rdatas["event./event.fFDEvents/event.
    fFDEvents.fFdRecShower.fUniqueID"].array(library="pd")),
    ...
    "FD_fTelescopeData":pd.DataFrame(rdatas["event./event.fFDEvents/
    event.fFDEvents.fTelescopeData"].array(library="pd")),
    }
# Funcion para eliminar los corchetes
def quitar_corchetes(dataframeFD):
    return dataframeFD.applymap(lambda x: x[0] if isinstance(x, list)
    and len(x) == 1 else x)
dfFD = pd.concat(dataframeFD, axis=1)

dfFD.to_csv('FD.csv', sep='\t', index=False)
```

Listing 6.3: Guardado del DataFrame en un archivo CSV

Tras la eliminación de datos nulos y la selección exclusiva de variables numéricas, el archivo original `FD.csv` fue procesado y almacenado como `FD_num.csv` con 111 features(características), bajo la misma lógica, el archivo `SD_num.csv` contiene 74 features, y ambos conjuntos de datos comparten 172,510 observaciones.

En cuanto a librerías empleadas, `pandas` y `numpy` se usan para la manipulación y análisis eficiente de estructuras de datos, mientras que `train_test_split` de `sklearn`

`.model_selection` permite dividir el conjunto de datos en entrenamiento y prueba. `StandardScaler` normaliza los datos, mejorando el rendimiento de los modelos. Las métricas `r2_score`, `mean_squared_error` y `mean_absolute_error` evalúan la precisión del modelo. `xgboost` proporciona una implementación optimizada del algoritmo XGBoost, ideal para tareas de regresión. `matplotlib.pyplot` permite generar visualizaciones para interpretar los resultados, y `joblib` facilita la serialización del modelo entrenado para su posterior uso. Finalmente, `time` se utiliza para medir el tiempo de ejecución del proceso, lo cual es útil para evaluar el rendimiento computacional.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
import xgboost as xgb
import matplotlib.pyplot as plt
import joblib
import time
```

Listing 6.4: Librerías utilizadas

Para la selección de variables relevantes en el análisis, se consideró como variable objetivo `FD_RS_fXmax`, que representa la profundidad máxima de la lluvia atmosférica, mientras que `SD_RS_fEnergy` se empleó como referencia energética de los eventos. En cuanto al conjunto de datos del detector de superficie (SD), se incluyeron las 74 variables disponibles como características predictoras. Durante el preprocesamiento, los valores faltantes fueron imputados utilizando la mediana de cada variable correspondiente.

```
#comenzar a tomar el tiempo
start_time = time.time()
# Cargar datos
dfFD = pd.read_csv("datosFD_num.csv", sep="\t")
dfSD = pd.read_csv("datosSD_num.csv", sep="\t")

# Combinar los datos
fd_columns = ["FD_RS_fXmax"]
sd_columns = dfSD.columns
combined_data = pd.merge(dfFD[fd_columns], dfSD[sd_columns], left_index=
    True, right_index=True)
combined_data.fillna(combined_data.median(), inplace=True)
```

Se aplicó un filtrado energético, conservando únicamente los eventos con energías comprendidas entre $10^{18.5}$ y $10^{20.0}$ eV. Adicionalmente, se creó una nueva característica transformada calculando el logaritmo en base 10 de la energía registrada (`log_Energy`). Para el entrenamiento del modelo, se eliminó tanto la variable objetivo como la energía medida por el FD, utilizando exclusivamente los datos del SD. Estos datos fueron normalizados mediante estandarización (media = 0, desviación estándar = 1).

```

# Filtro del rango de energia
data_filtered = combined_data[(combined_data["SD_RS_fEnergy"] >=
    10**18.5) & (combined_data["SD_RS_fEnergy"] <= 10**20.0)].copy()
```

```
data_filtered["log_Energy"] = np.log10(data_filtered["SD_RS_fEnergy"])

# Caracteristicas y objetivo
X = data_filtered.drop(columns=["FD_RS_fXmax"])
y = data_filtered["FD_RS_fXmax"]

# Normalizacion de los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Antes de entrenar el modelo, los datos escalados (`X_scaled`) y las etiquetas (`y`) se dividieron aleatoriamente en conjuntos de entrenamiento (80 %) y prueba (20 %), utilizando una semilla fija (`random_state=42`) para garantizar reproducibilidad en los experimentos. Con el objetivo de mejorar el rendimiento predictivo del modelo en el rango de altas energías, se asignó un peso tres veces mayor a los eventos con energías logarítmicas entre 19.5 y 20.0.

```
# Datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
    test_size=0.2, random_state=42)

# Asignacion de pesos a las muestras en el rango de interes
sample_weights_train = np.where((y_train.index.map(data_filtered["
    log_Energy"]) >= 19.7) & (y_train.index.map(data_filtered["log_Energy
    "]) <= 20.0), 3, 1)
sample_weights_test = np.where((y_test.index.map(data_filtered["
    log_Energy"]) >= 19.7) & (y_test.index.map(data_filtered["log_Energy"
    ]) <= 20.0), 3, 1)
```

El siguiente fragmento del código convierte los conjuntos de datos de entrenamiento (`X_train, y_train`) y prueba (`X_test, y_test`) al formato DMatrix, ya que almacena los datos de manera eficiente, acelerando el entrenamiento y reduciendo el uso de recursos.

```
# Convertir los datos a formato DMatrix para XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train, weight=sample_weights_train
    )
dtest = xgb.DMatrix(X_test, label=y_test, weight=sample_weights_test)
```

6.2 Parámetros del modelo XGBoost

La optimización de hiperparámetros se llevó a cabo mediante un proceso iterativo de búsqueda paramétrica, evaluando el desempeño del modelo con diferentes configuraciones y seleccionando finalmente aquella que mostró la mejor capacidad predictiva según las métricas establecidas (Tabla 6.2). El modelo se configuró para minimizar el error cuadrático medio (RMSE) empleando una tasa de aprendizaje de 0.05, una profundidad máxima de árbol de 6 para controlar el sobreajuste, y un mínimo de peso en nodos hijos de 10 para evitar divisiones poco significativas. Además, se implementó submuestreo del 80 % en datos y 70 % en características para introducir aleatoriedad. Para garantizar la generalización del modelo, se aplicaron técnicas de regularización como gamma (`gamma=1`) para

penalizar nodos innecesarios, regularización L1 (`reg_alpha=0.5`) para reducir el peso de características poco relevantes, y regularización L2 (`reg_lambda=1`) para distribuir uniformemente el peso entre las variables y evitar el dominio de pocas características, y una semilla fija (seed=42) para asegurar reproducibilidad.

Parámetro	Valor definido	Probado en valores
Objective	reg:squarederror	
eval_metric	rmse	
learning_rate	0.05	0.01, 0.05, 0.1
max_depth	6	6, 8, 10
subsample	0.8	0.7, 0.8, 0.9
colsample_bytree	0.7	0.7, 0.8, 0.9
reg_alpha	0.5	0, 0.1, 0.5, 1
reg_lambda	1	0, 0.1, 0.5, 1
seed	42	

Cuadro 6.2: Parámetros para el entrenamiento del modelo XGBoost

```
params = {
    "objective": "reg:squarederror",  # Error cuadratico medio
    "eval_metric": "rmse",            # Metrica para evaluacion
    "learning_rate": 0.05,            # Reducir la tasa de aprendizaje
    "max_depth": 6,                   # Reducir la profundidad del arbol
    "min_child_weight": 10,           # Minimo de peso para cada division
    "subsample": 0.8,                 # Submuestreo de datos (80%)
    "colsample_bytree": 0.7,          # Submuestreo de caracteristicas
    (70%)
    "gamma": 1,                       # Regularizacion de la poda
    "reg_alpha": 0.5,                 # Regularizacion L1
    "reg_lambda": 1,                  # Regularizacion L2
    "seed": 42                        # Reproducibilidad
}
```

Para proporcionar una estimación más realista del rendimiento del modelo más que la división de entrenamiento/prueba y evitar el sobreajuste, se utilizó validación cruzada con 5 folds, usando cada vez 4 folds para entrenamiento y 1 fold para validación, de tal forma que se promediaron los resultados de las 5 ejecuciones y se obtuvo la raíz del error cuadrático medio con mejor precisión. El modelo se entrenó con un máximo de 1200 iteraciones y durante el entrenamiento, se monitorizó el error de validación en cada iteración, si después de 100 iteraciones consecutivas el error no mejoraba, el entrenamiento se detenía. De tal forma que, el entrenamiento finalizó al llegar a 1012 iteraciones.

```
cv_results = xgb.cv(
    params=params,
    dtrain=dtrain,
    num_boost_round=1200,      # Numero maximo de iteraciones
    nfold=5,                   # Numero de folds en la validacion cruzada
    metrics="rmse",            # Metrica a evaluar
```

```
    early_stopping_rounds=100,# Detener si no mejora en 100 iteraciones
    seed=42,                  # Reproducibilidad
    as_pandas=True,           # Devolver resultados en un DataFrame
    verbose_eval=100          # Mostrar progreso cada 100 iteraciones
)
# Mostrar resultados de la validacion cruzada
print("Resultados de la validacion cruzada:")
print(cv_results)
```

6.3 Entrenamiento del modelo y obtención de predicciones

Se utilizó los parámetros predefinidos `params` y los datos de entrenamiento convertidos al formato `DMatrix` (`dtrain`), donde `num_boost_round` define el número de iteraciones de boosting, usando el valor óptimo obtenido previamente en una validación cruzada (`cv_results.shape[0]`), y monitorea el rendimiento durante el entrenamiento mediante los conjuntos de entrenamiento `dtrain` y prueba `dtest` con evaluaciones cada 100 iteraciones; finalmente, genera predicciones `y_pred` para el conjunto de prueba `dtest` aplicando el modelo entrenado.

```
xgb_model = xgb.train(
    params=params,
    dtrain=dtrain,
    num_boost_round=cv_results.shape[0],  # Usar el mejor numero de
    iteraciones
    evals=[(dtrain, "train"), (dtest, "test")],
    verbose_eval=100
)

# Predicciones
y_pred = xgb_model.predict(dtest)
```

6.4 Evaluación y visualización del modelo

Se calcularon las métricas de evaluación sobre el conjunto de prueba, obteniendo los siguientes resultados: coeficiente de determinación (R^2) = 0.7271, error cuadrático medio (MSE) = 721.2621 y la raíz del error cuadrático medio (RMSE) = 26.8563. El sesgo (Bias) fue de 0.5895, mostrando que el error medio es cercano a cero, lo que indica que, en promedio, el modelo no subestima ni sobrestima significativamente. El coeficiente de Pearson (ρ) = 0.8653 mostró una fuerte correlación lineal entre las predicciones y los valores observados. Y, la desviación estándar de los residuos (σ) fue de 26.8499, similar en magnitud al RMSE, lo que sugiere una dispersión considerable de los errores alrededor de la media.

```
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"Coeficiente de determinacion (R^2): {r2:.4f}")
```

```
print(f"Error cuadratico medio (MSE): {mse:.4f}")
print(f"Raiz del error cuadratico medio (RMSE): {rmse:.4f}")

residuos = y_test - y_pred

bias = np.mean(residuos) #media de los residuos
print(f"Bias: {bias:.4f}")

pearson_r = np.corrcoef(y_test, y_pred)[0, 1] #Coeficiente de Pearson
print(f"Coeficiente de Pearson (r): {pearson_r:.4f}")

sigma = np.std(residuos)
print(f"Desviacion estandar de residuos: {sigma:.4f}") #Desviacion
    estandar
```

Se generó un gráfico de dispersión comparando los valores reales contra las predicciones de X_{max}(ver Figura 6.1).

```
plt.figure(figsize=(8, 7))
plt.scatter(y_test, y_pred, color='navy', alpha=0.6, label="Predicciones
    ", s=1)
plt.xlabel("$X_{max,SD}/g cm^{-2}$", fontsize=16)
plt.ylabel("$X_{max,FD}/g cm^{-2}$", fontsize=16)
plt.text(1027, 708, f"2", fontsize=10, color="black")
plt.text(1032, 686, f"2", fontsize=10, color="black")
plt.text(920, 700, f"$\mu$: {bias:.4f} g/cm", fontsize=14, color="black"
    )
plt.text(920, 680, f"$\sigma$: {sigma:.4f} g/cm", fontsize=14, color="
    black")
plt.text(920, 660, f"$\rho$: {pearson_r:.4f}", fontsize=14, color="black
    ")
plt.text(920, 640, f"N: 5283", fontsize=14, color="black")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()
```

Por último, para evitar que se entrene múltiples veces la red, se guarda el modelo para utilizarlo en futuras predicciones y se finaliza el tiempo.

```
xgb_model.save_model("xgbmSD_6.model")
joblib.dump(scaler, 'scalerXGBSD_6.pkl')

end_time = time.time()
print(f"\nTiempo total de entrenamiento: {end_time - start_time:.2f}
    segundos")
```

6.4.1 Comparación durante el entrenamiento

Este trabajo difiere del enfoque de Abdul Halim et al. (Phys. Rev. D 111, 022003)[2] en arquitectura y alcance. Mientras su modelo utiliza redes neuronales profundas (DNN) para predecir X_{max} directamente desde señales crudas del Surface Detector (SD), nuestro método emplea XGBoost con características preextraídas del SD, sin modificar la representación

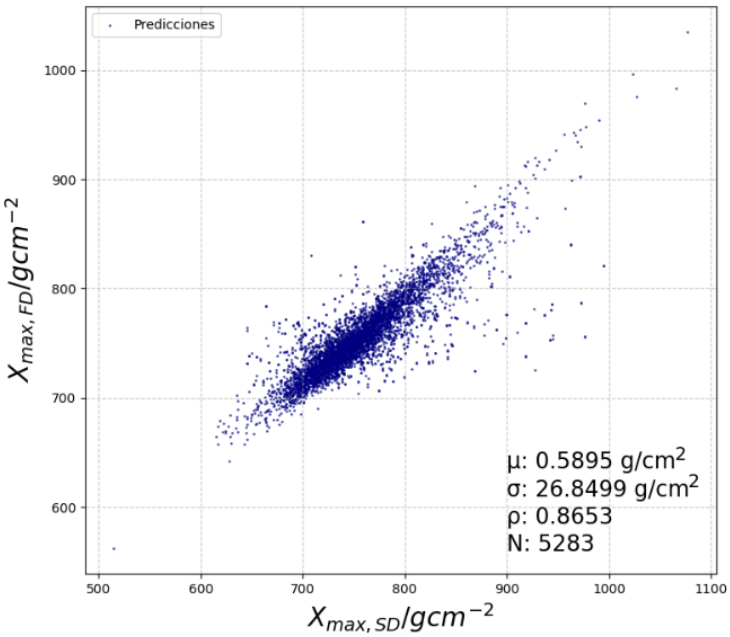


Figura 6.1: Aplicación de la estimación de X_{max} usando el modelo con XGBoost a los datos de Golden.

original de los datos. El modelo implementado por XGBoost presenta un sesgo sistemático significativamente menor ($\mu_{\text{XGBoost}} = 0.58\,\text{g/cm}^2$ frente a $\mu_{\text{DNN}} = -31.4\,\text{g/cm}^2$), junto con una mayor precisión ($\sigma_{\text{XGBoost}} = 26.8\,\text{g/cm}^2$ versus $\sigma_{\text{DNN}} = 33.4\,\text{g/cm}^2$) y una correlación mejorada ($\rho_{\text{XGBoost}} = 0.86$ comparado con $\rho_{\text{DNN}} = 0.70$), demostrando un rendimiento superior en todos los aspectos métricos clave. La gráfica que contiene esta información corresponde a la Figura 4.2, mientras que nuestros resultados se presentan en la Figura 6.1.

6.5 Tiempo de ejecución

El uso de la infraestructura del Laboratorio Nacional de Supercómputo del Sureste de México (LNS), en particular el clúster Cuetlaxcoapan con nodos Intel Xeon E5-2680 v3 (24 núcleos físicos, 128 GB RAM) y GPUs NVIDIA Tesla P100 (16 GB VRAM)[13], permitió reducir el tiempo de ejecución de esta parte del código a **52.17 segundos**, frente a los ∽15 minutos requeridos en una computadora convencional (CPU i5, 16 GB RAM).

De igual forma, el LNS permitió optimizar significativamente el procesamiento de las bases de datos de gran volumen (del orden de gigabytes), las cuales no se podían manejar en mi equipo. Para ello, se estableció una conexión segura al servidor mediante SSH, se configuró un entorno de trabajo personalizado cargando el módulo Anaconda3 (tras modificar el archivo .bashrc para habilitar la ruta de módulos /opt/ohpc/pub/BUILDS/-modules/all/) y posterior a eso, la creación de un entorno virtual específico usando conda create –prefix. Luego, se implementó un tunelamiento SSH para acceder dinámicamente a Jupyter Lab desde el servidor, evitando la dependencia exclusiva de scripts. Esto no solo

aceleró la ejecución de archivos (con tiempos menores que en mi computadora local), sino que también permitió una identificación más rápida de errores gracias a la interactividad de Jupyter y la capacidad de procesamiento del LNS.

Predicción en datos únicamente de SD utilizando el modelo preentrenado

7.1 Datos y Preprocesamiento

Se realizó la descarga de datos que contiene información únicamente de SD correspondientes a las fechas desde el 01 de enero de 2004 al 13 de abril de 2024.

Como criterios de preselección, se requiere un ángulo cenital menor a 60° para considerar trayectorias relativamente verticales o poco inclinadas, al igual que al menos cinco de estos seis detectores de superficie estén rodeados completamente por otros tanques activos y registren simultáneamente señales significativas, además sólo se consideran eventos con $\log_{10}(\mathrm{E/eV}) > 17$ y se conservan únicamente los eventos cuando el SD está operando correctamente. Después de aplicar estos cortes mostrados en la Tabla 7.1, el número de eventos se redujo de 7,915,620 a 5,881,657 eventos.

Cortes	Eventos	$\varepsilon(\,\%)$
Número total	7,915,621	100.0 %
Iluminación	7,909,024	99.9 %
Nivel de reconstrucción mínima	7,909,024	100.0 %
Máximo ángulo cenital (60º)	7,566,009	95.7 %
T4Trigger	7,564,204	100.0 %
T5Trigger	6,184,885	81.8 %
T5TriggerUB	6,184,754	100.0 %
$log_{10}(E/EV)$ >17.0	6,034,695	97.6 %
Archivo de rechazo de períodos malos	5,881,657	97.5 %

Cuadro 7.1: Selecciones básicas y específicas del análisis para el conjunto de datos SD.

Siguiendo el mismo procedimiento aplicado en la sección anterior para la generación de archivos CSV, se creó una nueva base de datos llamada `SDnewversion.csv`. Este archivo conserva las mismas variables presentes en `SD_num.csv` pero con sus propios valores respectivos, conformando un conjunto que registra un total de 2,618,948 observaciones.

Para realizar predicciones en un nuevo conjunto de datos, se cargó el modelo `xgbmSD_6.model` y el escalador `scalerXGBSD_6.pkl`

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
import xgboost as xgb
import joblib

scaler_X = joblib.load('scalerXGBSD_6.pkl')
xgb_model = xgb.Booster()
xgb_model.load_model('xgbmSD_6.model')
```

Listing 7.1: Carga del modelo e importación de librerías necesarias

Se cargaron dos archivos CSV (`SD_num.csv` y `SDnewversion.csv`), se realizó un filtro a los datos del segundo archivo conservando solo los registros donde la energía `SD_RS_fEnergy` está en el rango entre $10^{18.5}$ y 10^{20}, luego crea un nuevo DataFrame `new_data` que replica la estructura de columnas del primer archivo (`SD_num.csv`) pero con los valores filtrados del segundo, y finalmente añade una nueva columna llamada `log_energy` que contiene el logaritmo base 10 de los valores de energía.

```
#datos de SD extraidos de Golden
dfSD = pd.read_csv("SD_num.csv", sep="\t")
sd_columns = dfSD.columns

#Cargar archivo que contiene unicamente datos SD
dfSD_nuevo = pd.read_csv("SDnewversion.csv")

# Datos de energia en la nueva base
dfSD_nuevo = dfSD_nuevo[(dfSD_nuevo["SD_RS_fEnergy"] >= 10**18.5) & (
    dfSD_nuevo["SD_RS_fEnergy"] <= 10**20)].copy()

# Crear DataFrame con las columnas utilizadas en el modelo
new_data = pd.DataFrame()
for col in sd_columns:
    new_data[col] = dfSD_nuevo[col]

new_data["log_Energy"] = np.log10(new_data["SD_RS_fEnergy"])
```

Se llevó a cabo la preparación de datos y predicciones usando el modelo XGBoost previamente entrenado. Primero, se seleccionó y ordenó las columnas de características eliminando la variable objetivo `FD_RS_fXmax` para garantizar que coincidan con el formato esperado por el modelo. Luego, se normalizaron los datos usando el scaler previamente ajustado `scaler_X` A continuación, se convirtieron los datos normalizados a un formato DMatrix y generó predicciones para la variable objetivo `FD_RS_fXmax`. Finalmente, se agregaron estas predicciones al DataFrame `new_data` en la columna `Pred_FD_RS_fXmax`.

```
feature_columns = data_filtered.drop(columns=["SD_RS_fXmax"]).columns
new_data = new_data[feature_columns]

# Normalizacion de los datos
X_SD_nuevo_scaled = scaler_X.transform(new_data)

dmatrix_nuevo = xgb.DMatrix(X_SD_nuevo_scaled)
#Predicciones
y_pred_SD_nuevo = xgb_model.predict(dmatrix_nuevo)
new_data["Pred_SD_RS_fXmax"] = y_pred_SD_nuevo
```

7.2 Intervalos de energía

Para garantizar la comparabilidad con estudios previos, se utilizó los mismos intervalos (bins) energéticos, permitiendo mantener una correspondencia directa. Específicamente, se consideraron los siguientes valores de energía (en escala logarítmica base 10): 18.55, 18.65, 18.75, 18.85, 18.95, 19.05, 19.15, 19.25, 19.35, 19.45, 19.55, 19.64, 19.74, 19.85 y 20.00.

```
# puntos de energia especificos
energy_points = np.array([18.55, 18.65, 18.75, 18.85, 18.95, 19.05,
                          19.15, 19.25, 19.35, 19.45, 19.55, 19.64,
                          19.74, 19.85, 20.00])
```

7.3 Predicciones para $\langle X_{\mathbf{max}} \rangle$ con su error

Se realizó un análisis estadístico de las predicciones de $\langle X_{\mathrm{max}} \rangle$ en función de la energía, organizando los datos en intervalos energéticos y calculando medidas de tendencia central y dispersión. Primero, se inicializó listas vacías para almacenar los valores medios de $\mathrm{X_{max}}$ y sus errores estándar de la media (SEM). Luego, para cada punto de energía definido en `energy_points`, se creó un filtro a los datos dentro de un rango de ± 0.05 unidades logarítmicas de energía. Si existen datos en el intervalo, se calcula la media y la desviación estándar de las predicciones de $\langle X_{\mathrm{max}} \rangle$(`Pred_FD_RS_fXmax`), y determina el SEM como medida de incertidumbre estadística. Si no hay datos en el intervalo, se asignan valores NaN para mantener la consistencia. Finalmente, se convirtieron los resultados a un array NumPy y se aplicó un ajuste sistemático sumando 6.32536 a todos los valores de $\langle X_{\mathrm{max}} \rangle$, lo que corresponde a un desplazamiento necesario para comparar con datos experimentales.

```
# listas para almacenar los valores de Xmax y SEM
mean_Xmax = []
sem_values = []

# indices de y_test en data_filtered
test_indices = y_test.index  # indices de y_test en data_filtered

# Calcular la media de Xmax y el SEM para cada intervalo de energia
for i in range(len(energy_points) - 1):
    lower_energy = energy_points[i]
    upper_energy = energy_points[i + 1]

    # Filtrar los datos en el intervalo de energia
    mask = (data_filtered.loc[test_indices, "log_CalEnergy"] >=
    lower_energy) & (data_filtered.loc[test_indices, "log_CalEnergy"] <
    upper_energy)

    if mask.sum() > 5:  # Asegurar que haya suficientes datos
        y_true_interval = y_test[mask]
        mean_Xmax.append(np.mean(y_true_interval))
        sem = np.std(y_true_interval) / np.sqrt(mask.sum())  # Calculo
    del SEM
        sem_values.append(sem)
    else:
```

```
        mean_Xmax.append(np.nan)
        sem_values.append(np.nan)

mask = (data_filtered.loc[test_indices, "log_CalEnergy"] >= 19.85) & \
       (data_filtered.loc[test_indices, "log_CalEnergy"] <= 20.00)
if mask.sum() > 5:
    y_true_interval = y_test[mask]
    mean_Xmax.append(np.mean(y_true_interval))
    sem = np.std(y_true_interval) / np.sqrt(mask.sum())
    sem_values.append(sem)
else:
    mean_Xmax.append(np.nan)
    sem_values.append(np.nan)
# Convertir listas a arrays numpy para evitar errores al graficar
mean_Xmax = np.array(mean_Xmax)
sem_values = np.array(sem_values)
```

7.4 Evolución de $\langle X_{\mathbf{max}} \rangle$ en función de la energía

Para la visualización de los resultados obtenidos, se creó el siguiente código

```
#Prediccion de Xmax generada por XGBoost
plt.figure(figsize=(10, 8))
plt.errorbar(energy_points, mean_Xmax, yerr=sem_values, fmt='v', color='red',
             capsize=5, label='$X_{max}$ (XGBoost Training)', zorder=4, markersize=8)

#-------Modelos teoricos-----------
def RailsFunc(lgE, p):
    return p[0] + p[1] * (lgE - 18.) + p[2] * (lgE - 18.) ** 2

def GetModelRails():
    return {
        'EPOS-LHC': {'H': {'mean': [748.78, 57.73, -0.85]}, 'Fe': {'mean': [648.60, 63.12, -1.97]}},
        'Sibyll2.3c': {'H': {'mean': [761.9, 57.4, -8.07e-10]}, 'Fe': {'mean': [656.6, 60.1, -0.00016]}},
        'QGSJetII-04': {'H': {'mean': [733.34, 54.05, -0.16]}, 'Fe': {'mean': [635.93, 59.70, -1.98]}}
    }
lgE = np.linspace(18.5, 20, 100)
models = GetModelRails()
styles = {
    'EPOS-LHC': {'H': {'color': 'darkgreen', 'linestyle': '-'},
                 'Fe': {'color': 'darkgreen', 'linestyle': '-'}},
    'Sibyll2.3c': {'H': {'color': 'darkgreen', 'linestyle': '--'},
                   'Fe': {'color': 'darkgreen', 'linestyle': '--'}},
    'QGSJetII-04': {'H': {'color': 'darkgreen', 'linestyle': '-.'},
                    'Fe': {'color': 'darkgreen', 'linestyle': '-.'}}
}

for model, particles in models.items():
```

```
    for particle, params in particles.items():
        if model == 'Sibyll2.3c' and particle == 'H':
            label = '_nolegend_'
        elif model == 'EPOS-LHC' and particle == 'H':
            label = '_nolegend_'
        elif model == 'QGSJetII-04' and particle == 'H':
            label = '_nolegend_'
        else:
            label = model  # Se usa el nombre del modelo para la leyenda
        plt.plot(
            lgE,
            RailsFunc(lgE, params['mean']),
            color=styles[model][particle]['color'],
            linestyle=styles[model][particle]['linestyle'],
            label=f"{label}",
            zorder=1
        )
plt.text(19.2, 840, "p", fontsize=14, color="darkgreen")
plt.text(19.4, 705, "Fe", fontsize=14, color="darkgreen")

# -------Puntos especificos de P.R. D 111---------
x_points = [18.55, 18.65, 18.75, 18.85, 18.95, 19.05, 19.15, 19.25,
    19.35, 19.45, 19.55, 19.64, 19.74, 19.85, 20.00]
y_points = [757.0, 758.8, 759.5, 761.8, 765.7, 770.0, 769.9, 774.0,
    774.7, 775.3, 778.6, 783.2, 787.2, 794.5, 793.9]

# Errores asimetricos
y_error_asymmetric = [
    [7.7, 9.6], [7.8, 9.2], [7.9, 8.8], [8.0, 8.5], [8.2, 8.2],
    [8.4, 7.9], [8.7, 7.7], [9.1, 7.5], [9.5, 7.4], [10.0, 7.3],
    [10.5, 7.3], [11.0, 7.3], [11.6, 7.5], [12.2, 7.8], [13.1, 8.3]
]

plt.errorbar(
    x_points, y_points,
    yerr=[np.array([lower for lower, upper in y_error_asymmetric]),
    np.array([upper for lower, upper in y_error_asymmetric])],
    fmt='D', color='#3357FF', ecolor='#3357FF', elinewidth=2, capsize=3,
    label="$\langle X_{max}\rangle$(PHYSICAL REVIEW D 111, 022003)",
    markersize=8, zorder=3
)

# -------Puntos especificos de P.R. D 90 ---------
x_point = [ 18.55, 18.65, 18.75, 18.85, 18.95, 19.05, 19.14,19.25
    ,19.34,19.45,19.62]
y_point = [ 754.5, 756.1, 757.4, 763.6, 764.6, 766.4, 767.0,
    779.5,773.1,787.9,779.8]

# Errores asimetricos
y_error_asymm = [ [7.5,8.5],
    [7.5,8.5], [7.5,8.5], [7.7,8.1], [7.8,7.8], [8.0,7.6],
    [8.2,7.4], [8.5,7.2], [8.7,7.1], [8.9,7.0], [9.4,6.9]]

plt.errorbar(
```

```
    x_point, y_point,
    yerr=[np.array([lower for lower, upper in y_error_asymm]),
          np.array([upper for lower, upper in y_error_asymm])],
    fmt='o', color='black', ecolor='black', elinewidth=2, capsize=3,
    label="$\langle X_{max}\rangle$(PHYSICAL REVIEW D 90, 122005)",
    markersize=8, zorder=2
)

# Configuracion general de la grafica
plt.xlabel('log_{10}(E/eV)')
plt.ylabel('$\langle X_{max}\rangle / (g/cm^2)$')
plt.legend()
plt.grid(True, linestyle="--", alpha=0.7)
plt.show()
```

En la Figura 7.1 se presenta la evolución del valor medio de las predicciones de X_{max} ($\langle X_{\max} \rangle$) en función de los intervalos de energía. Los resultados obtenidos en este trabajo se representan mediante símbolos de triángulos invertidos en color rojo, incluyendo las correspondientes barras de error correspondientes al error estándar de la media (SEM). Comparándolos con:

- Resultados reportados en PHYSICAL REVIEW D 111, 022003 (2025)[2] utilizando redes neuronales profundas (DNN)[2], representados por símbolos de diamante en color azul (Anexo 11.1)
- Datos experimentales publicados en PHYSICAL REVIEW D 90, 122005 (2014)[1], obtenidos con el detector de fluorescencia y mostrados mediante círculos negros (Anexo 11.3)
- Predicciones teóricas para núcleos de Protón y Hierro generadas por los modelos EPOS-LHC, Sybill 2.3c y QGSJetII-04[5], representadas mediante líneas en color verde (Anexo 11.2)

7.5 Número de eventos

El conteo de eventos en función de $\langle X_{\max} \rangle$ se generó mediante el siguiente código, y los resultados se compararon con los reportados en PHYSICAL REVIEW D 111 [2], tal como se muestra en la Tabla 7.2. Nuestro modelo entrenado con XGBoost presenta un mayor número de eventos de FD en comparación con la referencia, lo que sugiere una mejora en el ajuste estadístico debido a una muestra más amplia. Sin embargo, si bien una mayor cantidad de eventos puede incrementar la precisión estadística, no garantiza un mejor modelado físico. Adicionalmente, el conjunto de predicciones para X_{max} en nuestro análisis abarca un rango más extenso, lo que podría indicar una mayor capacidad para capturar variaciones en los datos.

```
# puntos de energia especificos
energy_points = np.array([18.5, 18.6, 18.7, 18.8, 18.9, 19.0,
                          19.1, 19.2, 19.3, 19.4, 19.5, 19.6,
                          19.7, 19.8, 19.9, 20.00, 20.1])
```

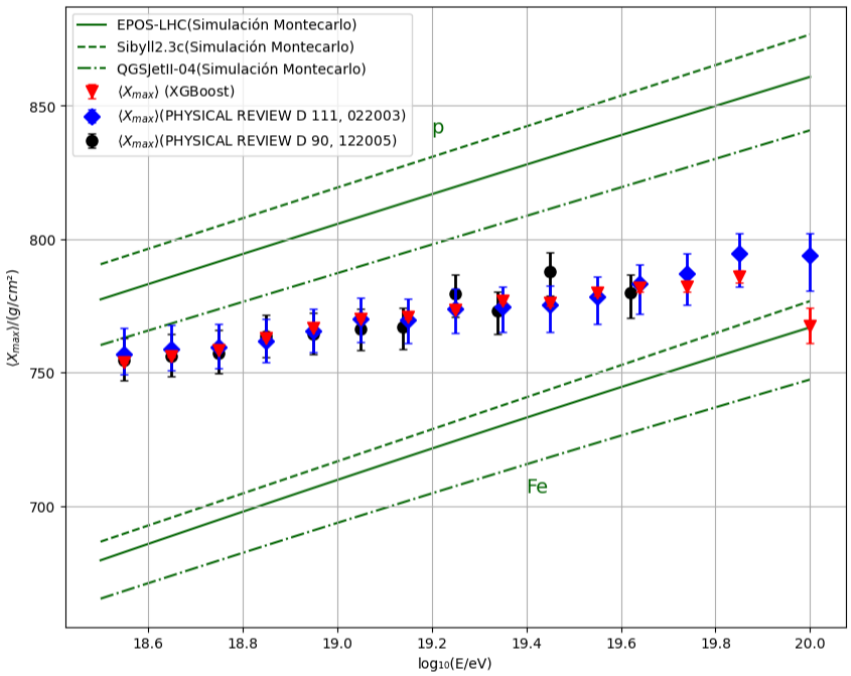


Figura 7.1: Evolución energética de la profundidad media máxima de la lluvia $\langle X_{max} \rangle$ predicha.

```

# Creacion de bins a partir de los puntos de energia
bins = pd.interval_range(start=energy_points.min(), end=energy_points.
    max(), freq=0.1, closed='right')

# Asignacion cada evento a un bin
data_filtered["Energy_Bin"] = pd.cut(data_filtered["log_CalEnergy"],
    bins=bins)

# Contar el numero de eventos en cada bin
event_counts = data_filtered["Energy_Bin"].value_counts().sort_index()

# Mostrar el numero de eventos en cada intervalo
print("Numero de eventos en cada intervalo de energia de FD:")
print(event_counts)

# Creacion de bins a partir de los puntos de energia
bins = pd.interval_range(start=energy_points.min(), end=energy_points.
    max(), freq=0.1, closed='right')

# Asignar cada evento a un bin
new_data["Energy_Bin"] = pd.cut(new_data["log_CalEnergy"], bins=bins)

# Contar el numero de eventos en cada bin
event_counts = new_data["Energy_Bin"].value_counts().sort_index()
print("Numero de eventos en cada intervalo de energia de Xmax:")
print(event_counts)
```

Bin $log_{10}E$/eV	$X_{max,FD}$(P.R. D 111)	$X_{max,FD}$	$X_{max,DNN}$(P.R. D 111)	$X_{max,XGBoost}$
18.5-18.6	1,347	6,981	8,739	24,347
18.6-18.7	1,007	4,967	9,360	14,835
18.7-18.8	707	3,516	7,725	9,329
18.8-18.9	560	2,788	6,506	6,361
18.9-19.0	417	2,014	5,228	4,442
19.0-19.1	312	1,529	3,863	3,099
19.1-19.2	253	1,112	2,781	2,073
19.2-19.3	159	702	1,791	1,379
19.3-19.4	122	543	1,205	898
19.4-19.5	80	349	701	514
19.5-19.6	50	207	455	346
19.6-19.7	35	74	277	180
19.7-19.8		52	113	69
19.8-19.9		20	54	50
>19.9		4	26	9

Cuadro 7.2: Número de eventos obtenidos de FD y la predicción usando XGBoost y comparándolo con el PHYSICAL REVIEW D 111, 022003 (2025)[2], X_{max} se expresa en g/cm^2

Alternativas descartadas

8.1 X_{max} usando el Algoritmo Forward-Forward

En la fase inicial del desarrollo, se planteó la implementación de una arquitectura de red neuronal capaz de integrar y procesar de manera conjunta los datos provenientes tanto del detector de superficie (SD) como del detector de fluorescencia (FD), con el objetivo de aprovechar la información complementaria proporcionada por ambos sistemas.

A diferencia de los métodos clásicos, la implementación del algoritmo Forward-Forward reemplaza los pasos forward y backward por dos pasos forward independientes: uno procesa datos reales (positivos) y otro datos sintéticos (negativos), donde cada capa optimiza una función objetivo local que maximiza la "bondad"(goodness) para los datos positivos y la minimiza para los negativos.[20] Se intentó adaptar este esquema para integrar información multivariada de los detectores de superficie (SD) y fluorescencia (FD), con el objetivo de capturar correlaciones no lineales entre sus señales y mejorar la precisión en la reconstrucción de X_{max}.

Al igual que en el modelo basado en XGBoost, se utilizaron los mismos conjuntos de datos, manteniendo constante el preprocesamiento, pero sustituyendo el algoritmo de aprendizaje automático y las variables de entrada.

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import time
import joblib
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
```

Listing 8.1: Librerías utilizadas

Primero, se cargaron dos conjuntos de datos (FD y SD) que son archivos CSV. Luego se combiní ambos datasets mediante una operación de merge. Para manejar valores faltantes, se implementó una imputación usando la mediana de cada columna. A continuación, se realizó un filtro a los eventos conservando únicamente aquellos con energías en el rango de 10^{15} a $10^{19.5}$ eV, añadiendo una nueva columna con el logaritmo de la energía para facilitar el análisis posterior. De igual forma, se implementó una técnica de aumento de datos específica para la región de energía entre $10^{16.5}$ y $10^{17.5}$ eV, donde genera copias sintéticas de los eventos originales añadiendo ruido gaussiano ($\mu = 0$, $s = 0{,}005$) y replicando este proceso 10 veces (factor=10).

```
#Cargar datos
dfFD = pd.read_csv("datosFD_num.csv", sep="\t")
```

```
dfSD = pd.read_csv("datosSD_nuam.csv", sep="\t")

# Seleccion de columnas
fd_columns = dfFD.columns()
sd_columns = dfSD.columns()

data = pd.merge(dfFD[fd_columns], dfSD[sd_columns], left_index=True, right_index=True)
data.fillna(data.median(), inplace=True)

# Filtrar datos en el rango de energia
data_filtered = data[(data["FD_RS_fCalEnergy"] >= 10**15) & (data["FD_RS_fCalEnergy"] <= 10**19.5)].copy()
data_filtered["log_CalEnergy"] = np.log10(data_filtered["FD_RS_fCalEnergy"])

# Funcion para aumentar datos en una region especifica
def augment_data_region(data, lower_bound, upper_bound, factor=5):
    subset = data[(data["FD_RS_fCalEnergy"] >= lower_bound) & (data["FD_RS_fCalEnergy"] <= upper_bound)]
    augmented_data = subset.copy()
    for _ in range(factor):
        noise = np.random.normal(0, 0.005, subset.shape)
        augmented_data = pd.concat([augmented_data, subset + noise], axis=0)
    return pd.concat([data, augmented_data], axis=0)

data_filtered = augment_data_region(data_filtered, 10**16.5, 10**17.5, factor=10)
```

Se realizó la selección de características y la variable objetivo para el modelo, utilizando como variables predictoras la energía reconstruida `FD_RS_fCalEnergy` y el error en azimut `FD_RS_fAzimuthError`, mientras que como variable objetivo se seleccionó la profundidad atmosférica del máximo desarrollo de la lluvia `FD_RS_fXmax`. Posteriormente, se aplicó un proceso de normalización estandarizada a ambos conjuntos de datos mediante el método StandardScaler, que transformó las características para tener media cero y desviación estándar unitaria. Para las variables predictoras (X), se ajustó el escalador a los datos y se aplicó la transformación correspondiente, mientras que para la variable objetivo (y) se realizó el mismo proceso, previo reajuste dimensional del array. Finalmente, con el objetivo de garantizar la consistencia en futuras aplicaciones del modelo, se almacenaron ambos escaladores en archivos binarios (`scalerFF1_X.pkl` y `scalerFF1_y.pkl`) utilizando la biblioteca joblib, permitiendo así su posterior carga y uso para la validación del modelo. Este procedimiento aseguró que los datos mantuvieran la misma escala tanto en el entrenamiento como en la evaluación del algoritmo.

```
X = data_filtered[["FD_RS_fCalEnergy", "FD_RS_fAzimuthError"]]
y = data_filtered["FD_RS_fXmax"]

# Normalizacion de caracteristicas y objetivo
scaler_X = StandardScaler()
scaler_y = StandardScaler()

```

```
X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()

# Guardar los escaladores
joblib.dump(scaler_X, 'scalerFF1_X.pkl')
joblib.dump(scaler_y, 'scalerFF1_y.pkl')
```

Se implementó una arquitectura de red neuronal fully-connected mediante el módulo `nn.Sequential` de PyTorch, diseñada para procesar datos con dos características de entrada. La red se estructuró en cuatro capas lineales con dimensiones decrecientes (512, 256, 128 y 1 neurona respectivamente), donde cada capa lineal fue seguida por una capa de normalización `BatchNorm1d` (con parámetros $e = 1e - 05$ y momentum=0.1) y una función de activación `LeakyReLU` (con pendiente negativa de 0.1), excepto en la capa final de salida. Para regularizar el modelo y prevenir el overfitting, se incorporaron capas de `Dropout` con probabilidad de 0.3 después de las dos primeras capas ocultas. Todos los tensores fueron direccionados al dispositivo de cómputo especificado (GPU/CPU) mediante el método .to(device).

```
def create_model(input_dim):
    model = nn.Sequential(
        nn.Linear(in_features=2, out_features=512, bias=True),
        nn.BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True),
        nn.LeakyReLU(negative_slope=0.1),
        nn.Dropout(p=0.3, inplace=False),
        nn.Linear(in_features=512, out_features=256, bias=True),
        nn.BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True),
        nn.LeakyReLU(negative_slope=0.1),
        nn.Dropout(0.3),
        nn.Linear(256, 128),
        nn.BatchNorm1d(128),
        nn.LeakyReLU(negative_slope=0.1),
        nn.Linear(128, 1)
    ).to(device)
    return model
```

Posteriormente, se implementó el procedimiento de entrenamiento Forward-Forward mediante una función personalizada que optimizaba los parámetros del modelo mediante descenso de gradiente adaptativo. La configuración utilizó el optimizador Adam con tasa de aprendizaje inicial de 0.0001 y decaimiento de pesos (L2 regularization) de 1e-5, junto con un scheduler `ReduceLROnPlateau` que reducía la tasa de aprendizaje a la mitad cuando la pérdida de validación no mejoraba durante 50 épocas consecutivas. El ciclo de entrenamiento iteró durante un máximo de 4000 épocas, calculando en cada iteración: (1) la pérdida personalizada `custom_loss` sobre los datos de entrenamiento, (2) su gradiente mediante `backpropagation`, y (3) la actualización de parámetros. También, se monitoreó el desempeño en el conjunto de validación sin afectar los gradientes `torch.no_grad()`. Se incorporó un mecanismo de early stopping que interrumpía el entrenamiento si no se observaba mejora en la pérdida de validación durante 150 épocas consecutivas para permitir suficiente exploración del espacio de parámetros. La función retornó el modelo entrenado junto con el mejor valor de pérdida de validación alcanzado,

garantizando así la selección del estado óptimo durante el proceso de optimización.

```
def forward_forward(model, X_train, y_train, X_val, y_val, epochs=4000, lr=0.0001, patience=150):
    optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=1e-5)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=50, verbose=True)
    best_loss = float('inf')
    patience_counter = 0

    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        output = model(X_train)
        loss = custom_loss(output, y_train, X_train)
        loss.backward()
        optimizer.step()

        # Evaluacion en conjunto de validacion
        model.eval()
        with torch.no_grad():
            val_loss = custom_loss(model(X_val), y_val, X_val)

        # Scheduler basado en la validacion
        scheduler.step(val_loss)

        if val_loss < best_loss or epoch < 2000:
            best_loss = val_loss
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience and epoch >= 2000:
            print(f"Early stopping at epoch {epoch}")
            break

    return model, best_loss.item()
```

Se convirtieron los datos normalizados (`X_scaled`, `y_scaled`) en tensores de PyTorch y los envía al dispositivo de cómputo (GPU/CPU). Luego se creó y se entrenó el modelo mediante la función `forward_forward()`, guardando el modelo resultante en `'modeloFF1.pth'`. Posteriormente, se generó las predicciones en modo de evaluación, desnormalizando tanto las predicciones como los valores reales usando el scaler previamente ajustado, se calculó el coeficiente de determinación R^2 para cuantificar el rendimiento y finalmente, se graficó los resultados mediante un scatter plot que compara valores reales vs predichos de X_{max}.(ver Figura 8.1)

```
X_tensor_full = torch.tensor(X_scaled, dtype=torch.float32).to(device)
y_tensor_full = torch.tensor(y_scaled.reshape(-1, 1), dtype=torch.float32).to(device)
model_final = create_model(input_dim=X_scaled.shape[1])
model_final = forward_forward(model_final, X_tensor_full, y_tensor_full, X_tensor_full, y_tensor_full)[0]
```

```
5
6 # Guardar el modelo final
7 torch.save(model_final, 'modelo_nn1.pth')
8 print("Se guardo el modelo final")
9
10 # Generar predicciones finales con el modelo entrenado
11 model_final.eval()  # Asegurar modo de evaluacion
12 with torch.no_grad():
13     y_pred_scaled = model_final(X_tensor_full)  # Predicciones escaladas
14
15 # Desnormalizar las predicciones y los valores reales
16 y_pred = scaler_y.inverse_transform(y_pred_scaled.cpu().numpy())
17 y_real = scaler_y.inverse_transform(y_scaled.reshape(-1, 1))  # Valores
       reales originales
18
19 # Calcular R^2
20 r2 = r2_score(y_real, y_pred)
21 print(f"Coeficiente de determinacion (R^2): {r2:.4f}")
22
23 # Crear la grafica
24 plt.figure(figsize=(10, 6))
25 plt.scatter(y_real, y_pred, alpha=0.5, label="Predicciones vs Reales")
26 plt.plot(
27     [y_real.min(), y_real.max()],
28     [y_real.min(), y_real.max()],
29     'k--',  # Linea diagonal de referencia
30     lw=2,
31     label="Linea de perfeccion"
32 )
33 plt.xlabel("Valores reales de Xmax (g/cm^2)", fontsize=12)
34 plt.ylabel("Predicciones de Xmax (g/cm^2)", fontsize=12)
35 plt.title(f"Comparacion entre valores reales y predicciones\n(R^2 = {r2
       :.4f})", fontsize=14)
36 plt.legend()
37 plt.grid(True, linestyle="--", alpha=0.7)
38 plt.show()
```

Para aplicar el modelo entrenado a un nuevo dataset, se cargó `'modelo_nn1.pth'` y se configuró en modo evaluación mediante `model_nn.eval()`. Posteriormente, se recuperaron los escaladores utilizados durante el entrenamiento (`scaler_X` y `scaler_y`), asegurando la consistencia en el preprocesamiento de los datos. Los nuevos datos (que contienen únicamente información de SD) procedentes del archivo `SD_newversion.csv` se importaron como un DataFrame de pandas. Para garantizar la compatibilidad con las variables del modelo, se renombraron las columnas `SD_RS_fAzimuthError` y `SD_RS_fEnergy` a `FD_RS_fAzimuthError` y `FD_RS_fCalEnergy`, respectivamente. Es decir, que se le asignó un valor a cada variable del FD una variable de SD. Se aplicó un filtrado basado en el rango de energía (entre $10^{18.5}$ y $10^{20.3}$ eV), y se añadió una nueva columna con el logaritmo en base 10 de la energía calibrada. Las características relevantes para el modelo (`FD_RS_fAzimuthError` y `FD_RS_fCalEnergy`) se extrajeron y normalizaron utilizando el escalador previamente ajustado (`scaler_X`).

```
1 # Cargar el modelo entrenado desde archivo
2 modelo_path = 'modelo_nn1.pth'
```

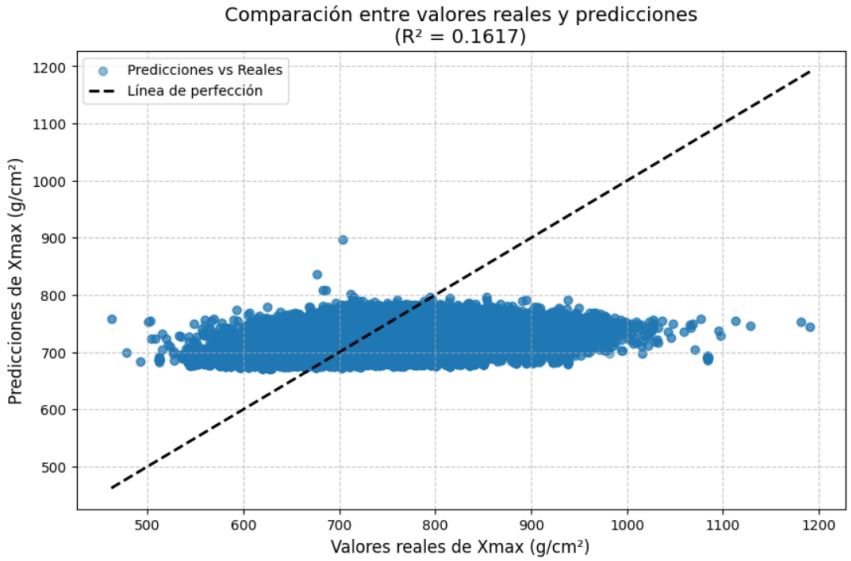


Figura 8.1: Aplicación de la estimación de X_{max} usando el modelo de Forward-Forward a los datos de Golden.

```
model_nn = torch.load(modelo_path)
model_nn.eval()  # Poner el modelo en modo evaluacion
model_nn.to(device)

# Cargar los escaladores usados en el entrenamiento
scaler_X = joblib.load('scaler_X.pkl')
scaler_y = joblib.load('scaler_y.pkl')

# Carga de los nuevos datos de SD
dfSD_nuevo = pd.read_csv("SD_newversion.csv", sep="\t")

# Renombrar columnas para coincidir con los datos usados en el modelo
dfSD_nuevo.rename(columns={"SD_RS_fAzimuthError": "FD_RS_fAzimuthError",
    "SD_RS_fEnergy": "FD_RS_fCalEnergy"}, inplace=True)
# Aplicar recorte de energia
dfSD_nuevo = dfSD_nuevo[(dfSD_nuevo["FD_RS_fCalEnergy"] >= 10**18.5) & (
   dfSD_nuevo["FD_RS_fCalEnergy"] <= 10**20.3)].copy()

# Convertir la energia a escala logaritmica
dfSD_nuevo["log_CalEnergy"] = np.log10(dfSD_nuevo["FD_RS_fCalEnergy"])

# Seleccion de caracteristicas para el modelo (manteniendo la energia en
    formato original)
X_SD_nuevo = dfSD_nuevo[["FD_RS_fCalEnergy", "FD_RS_fAzimuthError"]]

# Normalizacion utilizando el escalador previamente ajustado
X_SD_nuevo_scaled = scaler_X.transform(X_SD_nuevo)
```

Para evitar problemas de memoria durante la inferencia, se implementó una estrategia de predicción por lotes, donde el conjunto de datos normalizados `X_SD_nuevo_scaled` se procesó en segmentos de tamaño `batch_size = 1000`. En cada iteración, un lo-

te de datos se convirtió a un tensor de PyTorch con tipo `torch.float32`- Utilizando el contexto `torch.no_grad()` para desactivar el cálculo de gradientes y optimizar el uso de memoria, las predicciones del modelo (`model_nn`) se generaron y almacenaron en una lista (`y_pred_SD_nuevo`), tras convertir los resultados a un array de NumPy en la CPU. Posteriormente, las predicciones se desnormalizaron aplicando la transformación inversa del escalador de salida (`scaler_y.inverse_transform`), recuperando así las unidades originales. Finalmente, los valores predichos para la variable objetivo (`Pred_FD_RS_fXmax`) se incorporaron al DataFrame original (`dfSD_nuevo`).

```
# Predicci n por lotes para evitar problemas de memoria
y_pred_SD_nuevo = []
batch_size = 1000
for i in range(0, len(X_SD_nuevo_scaled), batch_size):
    X_batch = torch.tensor(X_SD_nuevo_scaled[i:i+batch_size], dtype=
    torch.float32).to(device)
    with torch.no_grad():
        y_pred_SD_nuevo.extend(model_nn(X_batch).cpu().numpy())

# Desnormalizaci n de las predicciones
y_pred_SD_nuevo_inv = scaler_y.inverse_transform(np.array(
    y_pred_SD_nuevo))

dfSD_nuevo["Pred_FD_RS_fXmax"] = y_pred_SD_nuevo_inv.flatten()
```

Se aplicó un filtrado inicial al DataFrame `dfSD_nuevo`, seleccionando únicamente las entradas donde `Pred_FD_RS_fXmax` era menor a 1000 y eliminando valores nulos en la columna `log_CalEnergy`. Posteriormente, se verificó que los valores de `log_CalEnergy` fueran finitos y positivos, garantizando la validez de los datos para el análisis. En caso de que el filtrado dejara menos de dos registros válidos, se generó un error (`ValueError`) para evitar procesamiento incorrecto. Se definieron los límites de energía (`min_val` y `max_val`) y, si estos coincidían, se lanzó otro `ValueError` debido a la imposibilidad de crear intervalos (`bins`) con un único valor.

```
df_filtered = dfSD_nuevo[dfSD_nuevo["Pred_FD_RS_fXmax"] < 1000].dropna(
    subset=["log_CalEnergy"])

# Asegurar que los valores de log_CalEnergy son finitos y positivos
df_filtered = df_filtered[np.isfinite(df_filtered["log_CalEnergy"]) & (
    df_filtered["log_CalEnergy"] > 0)]

# Verificar si hay suficientes datos para crear bins
if len(df_filtered) < 2:
    raise ValueError("No hay suficientes datos validos despues del
    filtrado.")

# Crear bins asegurando que sean validos y monotonicamente crecientes
min_val = df_filtered["log_CalEnergy"].min()
max_val = df_filtered["log_CalEnergy"].max()

# Evitar rangos invalidos
if min_val == max_val:
    raise ValueError("Los valores de energia son identicos, no se pueden
    crear bins.")
```

Se generaron 20 intervalos (bins) en el rango de energía logarítmica utilizando `np.linspace`, y se asignaron a una nueva columna (`Energy_Bins`) mediante `pd.cut`. Luego, se calculó el promedio de `Pred_FD_RS_fXmax` en cada bin (`avg_filtered`) y se extrajeron los puntos medios de los intervalos (`energy_avg_filtered`) para su representación gráfica.

```
bins = np.linspace(min_val, max_val, 20)

# Crear bins en la columna
df_filtered["Energy_Bins"] = pd.cut(df_filtered["log_CalEnergy"], bins,
    include_lowest=True)

# Calcular el promedio de Xmax en cada bin
avg_filtered = df_filtered.groupby("Energy_Bins")["Pred_FD_RS_fXmax"].
    mean()

# Obtener los valores medios de los bins para graficar
energy_avg_filtered = [interval.mid for interval in avg_filtered.index.
    categories]
```

Para generar predicciones del modelo en un rango de energía controlado, se creó un array logarítmico (`energy_range`) entre $10^{17.5}$ y 10^{20} eV, y se construyó un DataFrame (`df_pred`) con estos valores y la mediana de `FD_RS_fAzimuthError`. Tras normalizar los datos con `scaler_X`, se realizaron predicciones por lotes (`batch_size = 20`) para evitar sobrecarga de memoria, desnormalizando luego los resultados con `scaler_y`.

```
batch_size = 20
energy_range = np.logspace(17.5, 20, 300)
df_pred = pd.DataFrame({"FD_RS_fCalEnergy": energy_range, "
    FD_RS_fAzimuthError": data_filtered["FD_RS_fAzimuthError"].median()})
X_pred_scaled = scaler_X.transform(df_pred)
X_pred_tensor = torch.tensor(X_pred_scaled, dtype=torch.float32).to(
    device)
predictions = []
for i in range(0, len(X_pred_tensor), batch_size):
    batch = X_pred_tensor[i:i+batch_size]
    with torch.no_grad():
        pred = model_nn(batch).cpu().numpy()
    predictions.extend(pred)

y_pred_inv = scaler_y.inverse_transform(np.array(predictions))
```

Luego, se calcularon barras de error del 0.5 % sobre las predicciones (`y_err`) y se muestrearon cada 10 puntos para visualización. Paralelamente, se agruparon los datos reales (`data_filtered`) en bins similares, obteniendo sus promedios (`real_avg`) y los centros de energía correspondientes (`energy_avg`) para comparación con las predicciones.

```
# Calcular barras de error
y_err = y_pred_inv.flatten() * 0.005
error_indices = np.arange(0, len(energy_range), 10)

# Calcular promedios de datos reales
```

```
bins = np.linspace(data_filtered["log_CalEnergy"].min(), data_filtered["
    log_CalEnergy"].max(), 20)
data_filtered["Energy_Bins"] = pd.cut(data_filtered["log_CalEnergy"],
    bins, include_lowest=True)
real_avg = data_filtered.groupby("Energy_Bins", observed=True).mean()
energy_avg = [interval.mid for interval in real_avg.index]
```

Finalmente, al momento de realizar la gráfica, añadimos los mismos puntos que 7.1 variando únicamente las predicciones del modelo (en lugar de colocar las de XGBoost, se agregaron las de Forward-Forward.

```
# Graficar predicciones con barras de error
plt.errorbar(energy_avg_filtered, avg_filtered.values, yerr=std_filtered
    .values,
             fmt='o', color="red", label="SD(Forward-Forward)", capsize
    =5, zorder=5)
```

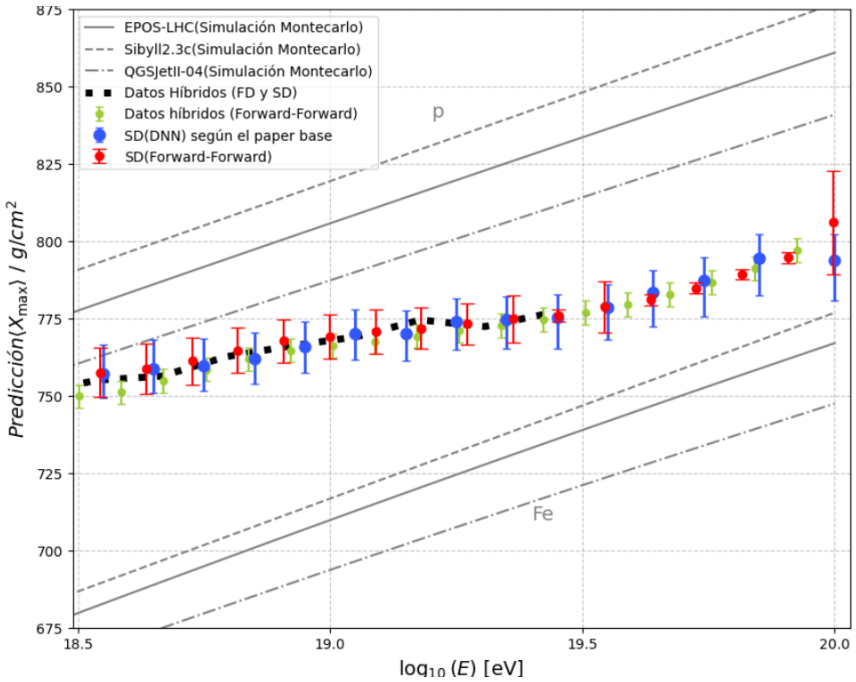


Figura 8.2: Evolución energética de la profundidad media máxima de la lluvia $\langle X_{max} \rangle$ predicha. De color rojo, es la predicción hecha por el algoritmo Forward-Forward

Como se observa en la Figura 8.1, el modelo que implementa Forward-Forward presentó un desempeño limitado, evidenciado por un coeficiente de determinación bajo ($R^2 = 0.1617$), lo que indicó una capacidad predictiva insuficiente para estimar X_{max}.

8.2 X_{max} usando XGBoost con diferentes variables de entrada

Antes de definir el conjunto final de variables predictoras, se realizó un análisis exploratorio preliminar en el que se evaluó la relación estadística entre distintas características y las

mediciones de X_{max} del detector de fluorescencia (FD). Para ello, se calculó la matriz de correlación y se priorizaron aquellas variables que exhibían una mayor dependencia lineal con `FD_RS_Xmax`, descartando predictores con asociación débil o redundante. Esto permitió identificar candidatos iniciales que posteriormente se refinaron mediante técnicas de selección de características, asegurando que las variables finalmente incorporadas al modelo maximizaran su capacidad predictiva manteniendo una interpretabilidad física coherente.

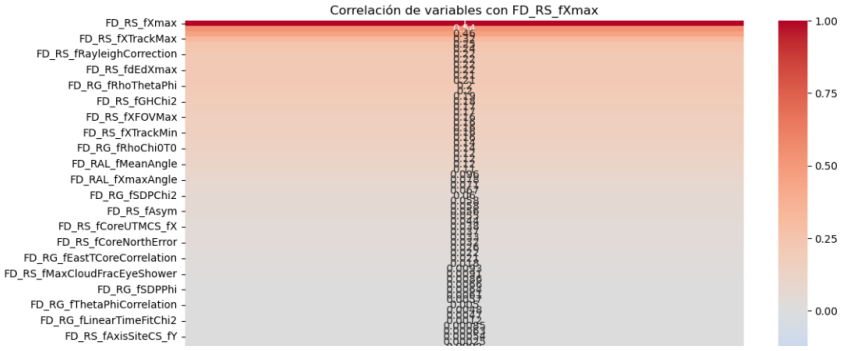


Figura 8.3: Variables del SD basada en su correlación lineal con X_{max} medido en el FD.

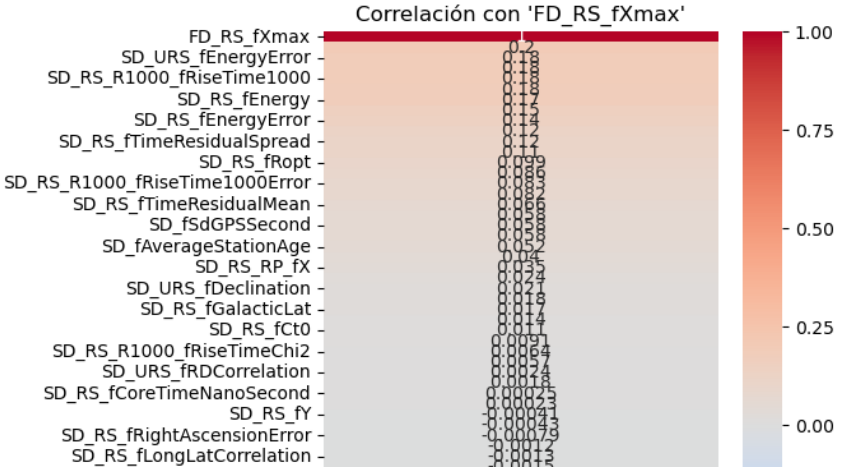
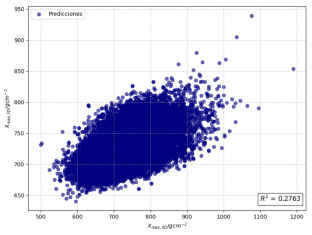
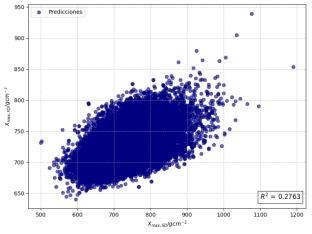
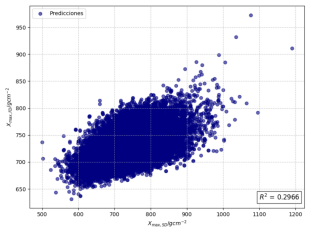
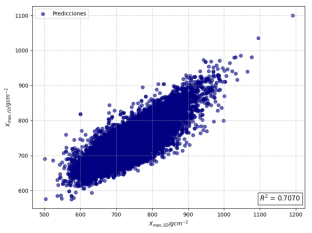


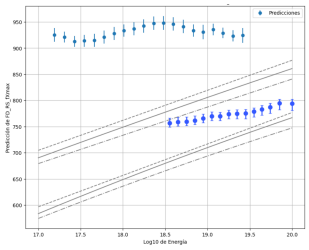
Figura 8.4: Variables del FD basada en su correlación lineal con X_{max} medido en el FD.

Con el objetivo de mejorar la precisión del modelo, se implementó un proceso de optimización mediante la evaluación de distintas variables predictoras. Esta selección se basó en un análisis de correlación entre las mediciones del SD y el FD con respecto a X_{max}, priorizando aquellas características que exhibían una mayor asociación lineal con la variable objetivo.

En la última fila de la Tabla 8.1 se observó que el coeficiente de correlación (R^2) mejoraba al entrenar el modelo con datos del FD. Por ello, se intentó implementar dicho modelo; sin embargo, surgió la necesidad de determinar qué variables del SD debían utilizarse. Para ello, se identificó la variable del SD con mayor correlación respecto a cada

Variables predictoras	R^2	**Tiempo**	**Gráfica**
"SD_RS_fAngleNdof" "SD_RS_fEnergyError" "SD_RS_fRopt" "SD_RS_fEnergy" "SD_RS_fZenithError" "FD_RS_fAzimuthError" "SD_RS_fZenithAzimuthCorrelation" "SD_URS_fXmax" "SD_URS_fXmaxError"	0.2966	597.12 seg.	
"SD_RS_fAngleNdof" "SD_RS_fEnergyError" "SD_RS_fRopt" "SD_RS_fEnergy" "SD_RS_fZenithError" "SD_RS_fAzimuthError" "SD_RS_fZenithAzimuthCorrelation" "SD_URS_fXmax" "SD_URS_fXmaxError"	0.3852	545.78 seg.	
"FD_RS_fXTrackMax" "FD_RS_fdEdXmax" "FD_RS_fGHChi2" "FD_RS_fXFOVMax" "FD_RS_fDeclination" "FD_RS_fRightAscension" "FD_RS_fCalEnergy"	0.4052	541.54 seg.	
"FD_RS_fXTrackMax" "FD_RS_fdEdXmax" "FD_RS_fGHChi2" "FD_RS_fXFOVMax" "FD_RS_fDeclination" "FD_RS_fRightAscension" "FD_RS_fCalEnergy" "FD_RS_fdEdXmaxError" "FD_RS_fEnergyError"	0.4817	351.40 seg.	

Cuadro 8.1: Predicciones de $\langle X_{max} \rangle$ usando variables de SD asociadas a variables de FD

Variables predictoras FD	Variables predictoras SD	Gráfica
"FD_RS_fXTrackMax"	"SD_RS_P_fZ"	
"FD_RS_fdEdXmax"	"SD_RS_R1000_fRiseTimeNDF"	
"FD_RS_fGHChi2"	"SD_RS_fAngleNdof"	
"FD_RS_fXFOVMax"	"SD_RS_RP_fZ"	
"FD_RS_fDeclination"	"SD_RS_fDeclination"	
"FD_RS_fRightAscension"	"SD_RS_fRightAscension"	
"FD_RS_fCalEnergy"	"SD_RS_fEnergy"	
"FD_RS_fdEdXmaxError"	"SD_RS_URS_fEnergy	
"FD_RS_fEnergyError"	"SD_RS_URS_fEnergyError	

Cuadro 8.2: Desempeño del modelo (R^2) con diferentes conjuntos de variables predictoras

variable del FD, asignándose las correspondencias mostradas en la Tabla 8.1. Los resultados se obtuvieron entrenando un modelo XGBoost con los siguientes parámetros: error cuadrático medio como función objetivo (`objective: reg:squarederror`), métrica RMSE (`eval_metric: rmse`), tasa de aprendizaje de 0.005 (`learning_rate`), profundidad máxima de 8 (`max_depth`), submuestreo de datos y características al 70 % (`subsample, colsample_bytree`), regularización L1/L2 (`reg_alpha: 0.1`, `reg_lambda: 1.0`), y semilla 42 para reproducibilidad. El entrenamiento utilizó 2500 iteraciones (`num_boost_round`) con parada temprana si no había mejora en 50 rondas (`early_stopping_rounds`), evaluándose en conjuntos de entrenamiento y prueba (`evals`). A pesar de eso, se identificó que las tendencias obtenidas no seguían un comportamiento adecuado o, al menos, no el esperado. Ya que las predicciones de $\langle X_{max} \rangle$ no se encontraban dentro del rango.

Tras evaluar el bajo desempeño predictivo de R^2 y ser insuficiente, se descartó el enfoque de entrenamiento híbrido (SD+FD) y se optó por utilizar exclusivamente datos del Detector de Superficie (SD) para el modelado.

La decisión de emplear exclusivamente datos del Detector de Superficie (SD) para el entrenamiento se fundamentó en que el modelo estaría destinado a aplicaciones donde únicamente se disponía de mediciones del SD. Así, cumple con:

1. Consistencia operacional: Eliminación de dependencias de variables del FD para garantizar su aplicación en condiciones reales que sólo usen SD.

2. Homogeneidad del entrenamiento: Evitar sesgos al correlacionar parámetros de FD que no estarían disponibles durante la implementación.

Conclusiones

De acuerdo con el modelo XGBoost empleado para datos del SD, los resultados obtenidos confirman que la composición en masa de los rayos cósmicos ultraenergéticos evoluciona de ligera a pesada. Esta transición se refleja en el comportamiento de $\langle \ln(E/A) \rangle$, determinado a partir del primer momento de las distribuciones de X_{max}, donde el modelo logra un rendimiento predictivo bueno ($R^2 = 0.72$) sin depender de arquitecturas complejas de Deep Learning. Las predicciones muestran una tendencia clara hacia valores compatibles con núcleos de hierro, indicando una composición más pesada, y se mantienen dentro de los límites establecidos por los modelos de interacción hadrónica EPOS-LHC y QGSJetII-04. Además, se observa un alto grado de consistencia entre las mediciones de $\langle X_{max} \rangle$ y los resultados previos basados en datos del detector de fluorescencia, lo que refuerza la validez de la transición observada y confirma la fiabilidad del modelo para el análisis de rayos cósmicos de ultra alta energía. La principal ventaja metodológica radica en la capacidad de predecir $\langle X_{max} \rangle$ directamente desde las señales del SD, eliminando la necesidad de pasos intermedios de reconstrucción de cascadas.

Apéndice

```
SD_fSdGPSSecond
SD_fSdGPSNanoSecond
SD_fYYMMDD
SD_fHHMMSS
SD_fAverageStationAge
SD_RS_fEnergy
SD_RS_fCoreTimeSecond
SD_RS_fCoreTimeNanoSecond
SD_RS_fGalacticLong
SD_RS_fGalacticLat
SD_RS_fDeclination
SD_RS_fRightAscension
SD_RS_fEnergyError
SD_RS_fCoreNorthError
SD_RS_fCoreEastError
SD_RS_fCoreNorthEastCorrelation
SD_RS_fZenithError
SD_RS_fAzimuthError
SD_RS_fZenithAzimuthCorrelation
SD_RS_fRightAscensionError
SD_RS_fDeclinationError
SD_RS_fRDCorrelation
SD_RS_fGalacticLongError
SD_RS_fGalacticLatError
SD_RS_fLongLatCorrelation
SD_RS_fX
SD_RS_fY
SD_RS_fZ
SD_RS_P_fX
SD_RS_P_fY
SD_RS_P_fZ
SD_RS_fCurvature
SD_RS_fCurvatureError
SD_RS_fCt0
SD_RS_fCt0Error
SD_RS_RP_fX
SD_RS_RP_fY
SD_RS_RP_fZ
SD_RS_fRiseTime1000
SD_RS_fRiseTime1000Error
SD_RS_fAngleChi2
SD_RS_fAngleNdof
SD_RS_fTimeResidualMean
SD_RS_fTimeResidualSpread
SD_RS_fRopt
SD_RS_fEnergyLDFSys
SD_RS_R1000_fRiseTime1000
SD_RS_R1000_fRiseTime1000Error
SD_RS_R1000_fRiseTimeChi2
SD_RS_R1000_fRiseTimeNDF
SD_RS_R1000_fAlpha
SD_RS_R1000_fBeta
SD_URS_fEnergy
SD_URS_fCoreTimeSecond
SD_URS_fCoreTimeNanoSecond
SD_URS_fGalacticLong
SD_URS_fGalacticLat
SD_URS_fDeclination
SD_URS_fRightAscension
SD_URS_fEnergyError
SD_URS_fCoreNorthError
SD_URS_fCoreEastError
SD_URS_fZenithError
SD_URS_fAzimuthError
SD_URS_fRightAscensionError
SD_URS_fDeclinationError
SD_URS_fRDCorrelation
SD_URS_fGalacticLongError
SD_URS_fGalacticLatError
SD_URS_fLongLatCorrelation
SD_URS_fNmu
SD_URS_fNmuError
SD_URS_fXmax
SD_URS_fXmaxError
```

Listing 10.1: Variables del Detector de Superficie con la que se entreó el modelo de xGBoost

Anexo

$log_{10}E$ range	$\langle log_{10}E \rangle$	$\langle X_{max} \rangle$
$[18{,}5 - 18{,}6)$	18.65	$757{,}0 \pm 0{,}5^{+7,7}_{-9,6}$
$[18{,}6 - 18{,}7)$	18.65	$758{,}8 \pm 0{,}5^{+7,8}_{-9,2}$
$[18{,}7 - 18{,}8)$	18.75	$759{,}5 \pm 0{,}5^{+7,9}_{-8,8}$
$[18{,}8 - 18{,}9)$	18.85	$761{,}8 \pm 0{,}5^{+8,0}_{-8,5}$
$[18{,}9 - 19{,}0)$	18.95	$765{,}7 \pm 0{,}5^{+8,2}_{-8,2}$
$[19{,}0 - 19{,}1)$	19.05	$770{,}0 \pm 0{,}6^{+8,4}_{-7,9}$
$[19{,}1 - 19{,}2)$	19.15	$769{,}9 \pm 0{,}6^{+8,7}_{-7,7}$
$[19{,}2 - 19{,}3)$	19.25	$774{,}0 \pm 0{,}8^{+9,1}_{-7,5}$
$[19{,}3 - 19{,}4)$	19.35	$774{,}7 \pm 0{,}9^{+9,5}_{-7,4}$
$[19{,}4 - 19{,}5)$	19.45	$775{,}3 \pm 1{,}0^{+10,0}_{-7,3}$
$[19{,}5 - 19{,}6)$	19.55	$778{,}6 \pm 1{,}3^{+10,5}_{-7,3}$
$[19{,}6 - 19{,}7)$	19.64	$783{,}2 \pm 1{,}4^{+11,0}_{-7,3}$
$[19{,}7 - 19{,}8)$	19.74	$787{,}2 \pm 2{,}0^{+11,6}_{-7,5}$
$[19{,}8 - 19{,}9)$	19.85	$794{,}5 \pm 3{,}6^{+12,2}_{-7,8}$
$[19{,}9 - \infty)$	20.00	$793{,}9 \pm 4{,}5^{+13,1}_{-8,3}$

Cuadro 11.1: Primer momento de las distribuciones X_{max} en PHYS. REV. D 111, 022003 (2025). Las energías se expresan en [eV] y $\langle X_{max} \rangle$ está dada en $[g/cm^2]$, seguido de sus incertidumbres estadísticas y sistemáticas.

Modelo	Partícula	Parámetros (p_0, p_1, p_2)
EPOS-LHC	H	748.78, 57.73, -0.85
	Fe	648.60, 63.12, -1.97
Sibyll2.3c	H	761.9, 57.4, -8.07e-10
	Fe	656.6, 60.1, -0.00016
QGSJetII-04	H	733.34, 54.05, -0.16
	Fe	635.93, 59.70, -1.98

Cuadro 11.2: Parámetros de los modelos teóricos para las partículas H y Fe usados en $p_0 + p_1(log_{10}E - 18) + p_2(log_{10}E - 18)^2$.[5]

$log_{10}E$ range	$\langle log_{10}E \rangle$	$\langle X_{max} \rangle$
$[18{,}6 - 18{,}7)$	18.65	$756{,}1 \pm 2{,}7^{+7,4}_{-8,8}$
$[18{,}7 - 18{,}8)$	18.75	$757{,}4 \pm 2{,}8^{+7,5}_{-8,5}$
$[18{,}8 - 18{,}9)$	18.85	$763{,}6 \pm 2{,}9^{+7,7}_{-8,1}$
$[18{,}9 - 19{,}0)$	18.95	$764{,}6 \pm 3{,}2^{+7,8}_{-7,8}$
$[19{,}0 - 19{,}1)$	19.05	$766{,}4 \pm 3{,}3^{+8,0}_{-7,6}$
$[19{,}1 - 19{,}2)$	19.15	$767{,}0 \pm 3{,}6^{+8,2}_{-7,4}$
$[19{,}2 - 19{,}3)$	19.25	$779{,}5 \pm 5{,}1^{+8,5}_{-7,2}$
$[19{,}3 - 19{,}4)$	19.35	$773{,}1 \pm 5{,}0^{+8,7}_{-7,1}$
$[19{,}4 - 19{,}5)$	19.45	$787{,}9 \pm 9{,}6^{+8,9}_{-7,0}$
$[19{,}5 - \infty)$	19.62	$779{,}8 \pm 5{,}0^{+9,4}_{-6,9}$

Cuadro 11.3: Primer momento de las distribuciones X_{max} en PHYSICAL REVIEW D 90, 122005 (2014). Las energías se expresan en [eV] y $\langle X_{max} \rangle$ está dada en $[g/cm^2]$, seguido de sus incertidumbres estadísticas y sistemáticas.

Bin $log_{10}E$/eV	$X_{max,FD}$(PR.111)	$X_{max,FD}$	$X_{max,DNN}$(PR.111)	$X_{max,XGBoost}$
18.5-18.6	1,347	6,981	8,739	24,347
18.6-18.7	1,007	4,967	9,360	14,835
18.7-18.8	707	3,516	7,725	9,329
18.8-18.9	560	2,788	6,506	6,361
18.9-19.0	417	2,014	5,228	4,442
19.0-19.1	312	1,529	3,863	3,099
19.1-19.2	253	1,112	2,781	2,073
19.2-19.3	159	702	1,791	1,379
19.3-19.4	122	543	1,205	898
19.4-19.5	80	349	701	514
19.5-19.6	50	207	455	346
19.6-19.7	35	74	277	180
19.7-19.8		52	113	69
19.8-19.9		20	54	50
>19.9		4	26	9

Cuadro 11.4: Número de eventos obtenidos de FD y la predicción usando XGBoost y comparándolo con el PHYSICAL REVIEW D 111, 022003 (2025)[2], X_{max} se expresa en g/cm^2

Bibliografía

[1] A. et al. Aab. Depth of maximum of air-shower profiles at the pierre auger observatory. i. measurements at energies above $10^{17,8}$ eV. *Phys. Rev. D*, 90:122005, Dec 2014. URL: https://link.aps.org/doi/10.1103/PhysRevD.90.122005, doi:10.1103/PhysRevD.90.122005. 15, 34

[2] A. et al. Abdul Halim. Measurement of the depth of maximum of air-shower profiles with energies between $10^{18,5}$ and 10^{20} eV using the surface detector of the pierre auger observatory and deep learning. *Phys. Rev. D*, 111:022003, Jan 2025. URL: https://link.aps.org/doi/10.1103/PhysRevD.111.022003, doi:10.1103/PhysRevD.111.022003. vi, 15, 16, 26, 34, 36, 52

[3] J. Abraham, P. Abreu, M. Aglietta, C. Aguirre, E. Ahn, D. Allard, I. Allekotte, J. Allen, P. Allison, J. Alvarez-Muñiz, M. Ambrosio, L. Anchordoqui, S. Andringa, A. Anzalone, C. Aramo, E. Arganda, S. Argirò, K. Arisaka, F. Arneodo, others, and M. Ziolkowski. The fluorescence detector of the pierre auger observatory. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 620(2-3):227–251, 2010. doi:10.1016/j.nima.2010.04.023. 4

[4] Ingomar Allekotte. Una mirada al observatorio pierre auger y a sus descubrimiento, 2022. URL: https://enula.org/2022/11/una-mirada-al-observatorio-pierre-auger-y-a-sus-descubrimientos/. 4

[5] Auger Open Data. Xmax analysis. https://www.kaggle.com/code/augeropendata/xmax-analysis, 2021. 3, 34, 51

[6] M. Ave, M. Roth, and A. Schulz. A generalized description of the time dependent signals in extensive air shower detectors and its applications. *Astroparticle Physics*, 88:46–59, 2017. URL: https://www.sciencedirect.com/science/article/pii/S0927650517300105, doi:10.1016/j.astropartphys.2017.01.003. 14

[7] Rene Brun and Fons Rademakers. Root — an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1):81–86, 1997. New Computing Techniques in Physics Research V. URL: https://www.sciencedirect.com/science/article/pii/S016890029700048X, doi:10.1016/S0168-9002(97)00048-X. 20

[8] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. *Machine learning and the physical sciences*, volume 91. American Physical Society, 2019. doi:10.1103/RevModPhys.91.045002. 9

[9] P. Cervantes Hernández. Media, varianza y desviación estándar. *Ciencia y Mar*, 12(34):29–36, 2008. 18

[10] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. KDD '16, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2939672.2939785. 10

[11] D Chicco, M.J. Warrens, and G. Jurman. The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. *PeerJ Computer Science 7:e623*, 2021. doi:10.7717/peerj-cs.623. 17

[12] Pierre Auger Collaboration. The pierre auger cosmic ray observatory. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 798:172–213, 2015. URL: https://www.sciencedirect.com/science/article/pii/S0168900215008086, doi:10.1016/j.nima.2015.06.058. 4, 5

[13] Benemérita Universidad Autónoma de Puebla. Infraestructura-laboratorio nacional de supercómputo del sureste de méxico, 2021. URL: https://lns.buap.mx/q=infraestructura%20. 27

[14] XGBoost Developers. Xgboost documentation (release 3.0.0). https://xgboost.readthedocs.io/en/release_3.0.0/, 2024. 12

[15] P. García Abia. Rayos cósmicos. Presentación en CERN Academic Training Lectures, 2016. URL: https://indico.cern.ch/event/572737/contributions/2612066/attachments/1482662/2299945/PGA-Rayos-cosmicos.pdf. 1

[16] F. A. Gomez Albarracín. *Estudio de la composición de rayos cósmicos de ultra alta energía en el Observatorio Pierre Auger*. Tesis doctoral, Universidad Nacional de La Plata, 2011. 2, 3

[17] Yann Guardincerri, Matías Leoni Olivera, and Bonadeo. Detector de fluorescencia en el observatorio pierre auger. 2006. 5

[18] A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017. URL: http://cds.cern.ch/record/2699693. 7

[19] Juan Diego et al. Hernández Lalinde. Sobre el uso adecuado del coeficiente de correlación de pearson: definición, propiedades y suposiciones. 2018. 19

[20] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022. URL: https://arxiv.org/abs/2212.13345, arXiv:2212.13345. 37

[21] E. Hossain. *Machine Learning Crash Course for Engineers*. Springer eBooks. Springer, 2023. doi:10.1007/978-3-031-46990-9. 7

[22] Cendejas L. M. V. Ibargüen, H. S. Rayos cósmicos ultraenergéticos: el observatorio pierre auger., 2006. URL: https://amc.mx/revistaciencia/images/revista/57_1/rayos_cosmicos.pdf. 1

[23] Georgia Karagiorgi, Gregor Kasieczka, Samuel Kravitz, Benjamin Nachman, and David Shih. Machine learning in the search for new fundamental physics. *Nature Reviews Physics*, 4(6):399–412, 2022. doi:10.1038/s42254-022-00455-1. 9

[24] Dong Kyu Lee, Ji Won In, and Sangseok Lee. Standard deviation and standard error of the mean. *Korean Journal of Anesthesiology*, 68(3):220–223, 2015. doi:10.4097/kjae.2015.68.3.220. 19

[25] Andreas Mayr, Harald Binder, Olaf Gefeller, and Matthias Schmid. The evolution of boosting algorithms: From machine learning to statistical modelling. *Methods of Information in Medicine*, 53(06):419–427, 2014. URL: http://dx.doi.org/10.3414/ME13-01-0122, doi:10.3414/me13-01-0122. 10

[26] R Mitchell and E. Frank. Accelerating the xgboost algorithm using gpu computing. *PeerJ Computer Science*, page e127, 2021. URL: https://peerj.com/articles/cs-127.pdf, doi:10.7717/peerj-cs.127. 10, 13

[27] Andrew Ng and Tengyu Ma. Cs229 lecture notes. Technical report, Stanford University, June 2023. URL: https://cs229.stanford.edu/main_notes.pdf. 7

[28] Observatorio Pierre Auger. ¿qué son los rayos cósmicos? URL: https://visitantes.auger.org.ar/index.php/que-son-los-rayos-cosmicos/. 1

[29] Observatorio Pierre Auger. Detectores de fluorescencia – observatorio pierre auger. https://visitantes.auger.org.ar/index.php/detectores-de-fluorescencia-2/, s.f. 5

[30] Observatorio Pierre Auger. Detectores de superficie – observatorio pierre auger. https://visitantes.auger.org.ar/index.php/detectores-de-superficie-2/, s.f. 6

[31] G. Ortega and J. Revelo. Modelamiento para el número de muones en función de la energía y el ángulo de entrada del protón incidente. *Revista Sigma*, 14(1):49–57, 2018. URL: http://coes.udenar.edu.co/revistasigma/articulosXIV/1.pdf. 1

[32] Pierre Auger Observatory. The observer realm – pierre auger observatory data production, 2024. URL: https://web.iap.kit.edu/observer/. 20

[33] V. L. Poma Almanza. *Búsqueda de correlaciones entre eventos de rayos cósmicos ultraenergéticos con fuentes astrofísicas de rayos gamma*. PhD thesis, Pontificia Universidad Cat´olica del Perú, 2019. URL: https://tesis.pucp.edu.pe/server/api/core/bitstreams/21043242-e065-4f2c-9d40-0a7166412488/content. 1

[34] A. Roy and J. Pivarski. Using a dsl to read root ttrees faster in uproot. *arXiv preprint arXiv:2303.02202*, 2023. URL: https://arxiv.org/abs/2303.02202, arXiv:2303.02202. 20

[35] Shagan Sah. Machine learning: A review of learning types. 2020. doi:10.20944/preprints202007.0230.v1. 7, 8, 9

[36] I. D. Vergara Quispe. *Composición primaria de los rayos cósmicos de ultra alta energía y estudios de modelos hadrónicos en el desarrollo de cascadas en el Observatorio Pierre Auger*. Doctoral dissertation, Universidad Nacional de La Plata, 2021. vi, 2

[37] H. Yi, S. Shiyu, D. Xiusheng, and C. Zhigang. A study on deep neural networks framework. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 1519–1522. IEEE, October 2016. 15