Desempeño de diferentes modelos de aprendizaje máquina

Kevin Alejandro Ramírez Luna - A01711063@tec.mx

11 sept 2025

***Abstract*— The next document presents the development and implementation of two machine learning models to predict the hourly energy demand of a power plant. The first model was trained using the gradient descent (GD) algorithm; the second was the Random Forest (RF) algorithm using the sklearn framework. The process consisted of a thorough data cleaning and transformation phase, followed by an exploratory analysis to identify relevant characteristics in order to compare both models.**

Resumen— El siguiente trabajo presenta el desarrollo e implementación de dos modelos de aprendizaje automático (machine learning) para predecir la demanda de energía/hora de una planta energética. El primer modelo se entrenó utilizando el algoritmo de descenso de gradiente (GD); el segundo fue el algoritmo de Random Forest (RF) usando el framework de sklearn . El proceso consistió en una fase exhaustiva de limpieza y transformación de los datos, seguida de un análisis exploratorio para identificar características relevantes para poder hacer un análisis de ambos modelos.

I. INTRODUCCIÓN

Usando el software de Python queremos demostrar el funcionamiento de una regresión lineal haciendo uso de uno de los algoritmos base para el Machine Learning; el algoritmo de Gradiente Descendente. Un algoritmo muy utilizado para problemas de regresión lineal o logística. En nuestro caso: regresión lineal para la predicción de una variable dependiente (variable numérica).

Posteriormente a esto, queremos implementar un modelo de aprendizaje supervisado con un Framework de Python (SkLearn) para implementar el Random Forest. Con el objetivo de poder comparar resultados y determinar en qué caso un modelo puede ser más óptimo que otro.

Pero antes de lanzarnos a programar nuestro algoritmo, necesitamos recaudar los datos de entrada para el algoritmo. La página UC_IRVINE es un buen repositorio con varios datasets útiles en problemas de Machine Learning que están disponibles para

Buscando entre la gran diversidad de datasets que proponen, se seleccionó el siguiente:

Fig. 1. Dataset a analizar [1]

El conjunto de datos contiene 9568 instancias de datos recopilados en una central eléctrica de ciclo combinado durante 6 años (2006-2011), cuando la central funcionaba a plena carga.

Se seleccionó el dataset al ser un buen candidato para lo que queremos hacer: predecir datos a través de una regresión lineal. Además que cuenta con una buena cantidad de datos y llega a ser lo suficientemente desafiante para una primera demostración/acercamiento del algoritmo.

II.PREPARACIÓN DEL ENTORNO

Antes de empezar a programar necesitamos crear un entorno con todas las dependencias/librerías necesarias para la realización de esta tarea. Para ello, estaremos usando Python por su facilidad y gran catálogo de librerías de Machine Learning; tales como Numpy, Pandas, MatplotLib, Seaborn, etc.

Para no generar problemas con proyectos futuros/pasados, creamos un entorno de desarrollo Pip. Los comandos a ingresar en el shell son:

```shell
# Creación del enviroment
python -m venv IA_AVANZA

# Activación del enviroment (en linux)
source IA_AVANZA/bin/activate

# Activación del enviroment (en windows)
```

```
IA_AVANZA\Scripts\activate

# Instalación de las librerias a utilizar
pip install pandas scikit-learn numpy
openpyxl seaborn graphviz
```

Código 1. Comandos para el enviroment [2]

Cabe recalcar que lo anterior se hizo para trabajar en VisualStudio Code sin embargo, perfectamente funcionaria en otros entornos como Anaconda. En Google Collab no son necesarios los comandos anteriores al contar con estas dependencias ya instaladas.

Ahora, con todo listo para trabajar, comenzaremos a programar nuestros algoritmos. Vamos a comenzar con el GD.

III. ETL - EXTRACT

Una vez seleccionado el dataset, se descarga y se coloca en una carpeta local. Con esto preparado y leyendo la documentación del dataset podemos proseguir sabiendo que el dataset cuenta con las dimensiones de 9568 instancias y 4 features. Donde las features dadas tiene la siguiente nomenclatura y significado:

Variable X	Rango
Temperature (T)	1.81°C and 37.11°C
Ambient Pressure (AP)	992.89-1033.30 milibar.
Relative Humidity (RH)	25.56% to 100.16%
Exhaust Vacuum (V)	25.36-81.56 cm Hg
Net hourly electrical energy output (EP) - Valor a predecir	420.26-495.76 MW

Fig. 2. Features del dataset

Esto será importante a la hora de debuggear y selección de variables independientes para la elaboración de nuestro *Hypothesis Model*, es decir: la regresión lineal y el segundo algoritmo, el Random Forest.

Con lo anterior en mente, sabemos cuales son nuestras posibilidades candidatas para la regresión, así como nuestro target Y (EP)

Al final, la estructura del proyecto estará basada en un script de Python donde hagamos la implementación y ETL del dataset, así como el respectivo dataset en nuestra carpeta local.

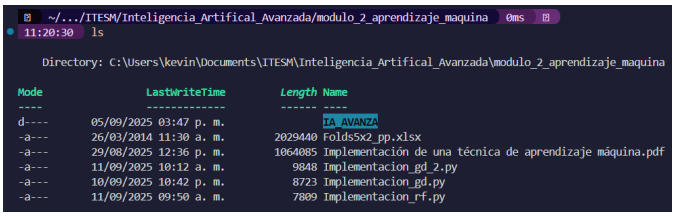


Fig. 3. Estructura del proyecto

IV. ETL - TRANSFORM - GD

Con los datos cargamos, procederemos a hacer transformaciones y limpieza de datos para poder trabajar en la regresión lineal. La limpieza y transformación de datos debe estar orientada en:

1. Limpieza de datos

Eliminación de valores nulos: Se aplicó df.dropna() para eliminar cualquier registro con valores faltantes en el dataset (si es que los hay)

2. Escalar valores a rangos similares

Escalar valores a rangos similares para no dar más peso uno al otro usando StandardScaler para estandarizar todas las variables a una escala similar (para no tener problemas con variables X con más rango, una que otras).

3. Selección de variables

Selección de variables independientes y dependientes para la elaboración del modelo. La elección de nuestras X´s y Y´s debe estar dada de una manera cualitativa que nos permita tener un buen acercamiento a lo que queremos modelar. En este caso, para elegir de una mejor manera estas variables, se implementó una matriz de correlación:

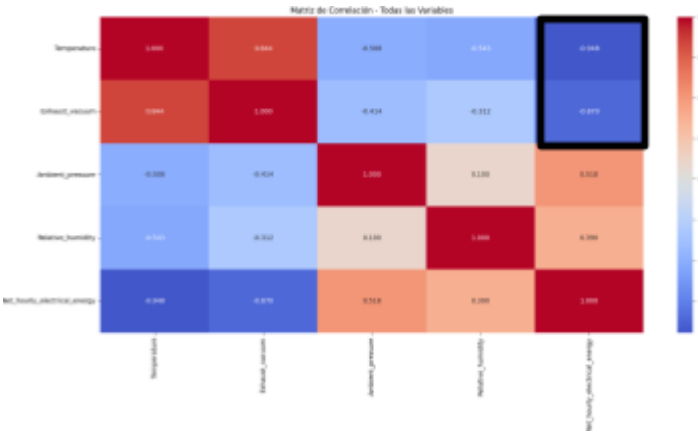


Fig. 4. Matriz de correlación

Nótese que los valores más altos en la matriz de correlación para nuestra salida llegan a ser negativas (T y AP) y son cercanas a -1.0. Esto quiere decir que hay una relación lineal negativa (hay una relación inversamente proporcional)

Conociendo esto y que estamos en un problema de regresión donde queremos conocer un valor energético, puede que estas dos variables signifiquen mucho por la mera naturaleza del problema; donde normalmente la temperatura influye mucho en la producción de energía. Esta será nuestra *hipótesis inicial* (Ho) para el modelo.

4. Redimensión de listas por matrices

Al seleccionar variables independientes y dependientes en un problema de regresión lineal es seguro que la columna Y (nuestro target) quede como una lista y no una matriz como podría ser con las filas seleccionadas en Y. Es ahí cuando usamos .reshape(-1, 1) para poder transformar una lista a una matriz (n,1).

5. División de sets de datos para entrenamiento y prueba de resultados

Dividimos nuestros datos en una proporción de 80/20: 7654 muestras para entrenamiento (80%) y 1914 para prueba (20%). Para controlar el split de nuestros datos usamos una semilla para tener una aleatoriedad controlada.

$$7654 + 1914 = 9568\ datos$$

Ecuación 1. Total de datos

IV.I MODELADO DE LA SOLUCIÓN - GD

Conociendo que la ecuación de la regresión lineal está definida como

$$y = \theta_1 x_1 + \theta_2 x_2 + \ldots \theta_n x_n + \beta$$

Ecuación 2. Regresión lineal

Donde:

$\theta = parámetro\ del\ modelo\ (peso\ sináptico)$

$x = valor\ de\ entrada\ (feature)$

$\beta = bias$

Para nuestro caso específico, con las dos variables seleccionadas (Temperature y Exhaust Vacuum), nuestro modelo toma la forma:

$$Energy\ output = \theta_1 Temperature + \theta_2 Ambient\ Pressure + \beta$$

Ecuación 3. Función de hipótesis desarrollada

El proceso de implementación siguió estos pasos:

- Inicialización de parámetros iniciales: comenzamos con valores iniciales de θ y β en cero.
- Implementamos un valor de Learning Rate lo suficientemente bajo para descender lo más controlado posible. Se eligió de forma arbitraria el valor de 0.1 ya que es un valor óptimo para que los gradientes convergen de manera lenta y precisa.
- Función de hipótesis: Implementamos la función que calcula las predicciones del modelo según la ecuación (3)
- Función de costo: Utilizamos el error cuadrático medio (MSE) para medir el desempeño del modelo.
- Implementación del algoritmo (SG):
    - Entrenamiento: Iteramos a través de los datos para ajustar los parámetros del modelo. Se hizo un total de 1000 épocas para ver la evolución del error en un número grande de iteraciones e intentando no caer en problemas de *Overfitting o Underfitting*
    - Acumulación del error: Al ir iterando, guardamos el error en cada época para graficarlo posteriormente y ver si es que el proceso de entrenamiento con las 1000 epochs es óptimo.
- Evaluación: Medimos el desempeño del modelo en datos de entrenamiento y prueba. Graficamos para la evaluación del modelo. Usamos métricas como el MSE y la R^2 para poder hacer diagnósticos y algunas conclusiones del modelo.

V.. RESULTADOS Y VALIDACIÓN - GD

Tras completar el proceso de entrenamiento con las 1000 épocas y una tasa de aprendizaje de 0.1, obtuvimos los siguientes resultados después del entrenamiento realizado.

Parámetros del modelo final

$$\theta_1(Temperature) = -12.8272$$

$$\theta_2(Exhaust\ Vacuum) = -4.0648$$

$$\beta = 454.3706$$

Quedando la ecuación final como:

$$Energy\ output = -12.8272 \cdot Temperature - 4.0648 \cdot Ambient\ Pressure + \beta$$

Ecuación 4. Ecuación característica para la salida

Nótese que los coeficientes negativos para *Temperature* y *Exhaust Vacuum* coinciden con lo observado en la matriz de correlación (Fig. 4). Esto quiere decir que nuestra hipótesis (Ho) se cumplió, ambas variables son inversamente proporcionales a la salida. Esto tiene sentido físico: a mayor temperatura, menor eficiencia en la conversión de energía eléctrica.

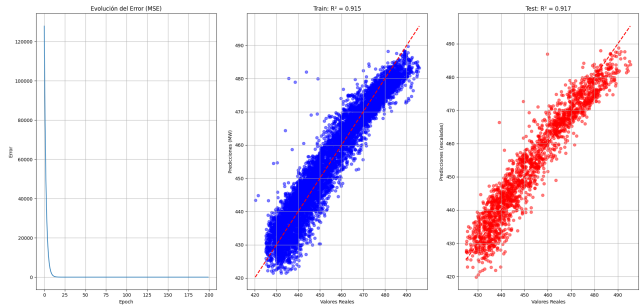


Fig. 5. Desempeño del modelo GD - 2 features

	MSE	R²
Entrenamiento	24.3859	0.9164
Prueba	24.0854	0.9170

Fig. 6. Métricas de evaluación

Recapitulando y rescatando lo mejor de lo anterior podemos observar que el valor de la R² tanto en entrenamiento como en prueba indica que el modelo explica aproximadamente el 91% de la variabilidad en la producción de energía. Lo que nos indica que se ajusta de una buena manera a los datos; las entradas si se están ajustando a la salida esperada.

Al comparar la similitud entre ambas métricas de entrenamiento y prueba sugiere que el modelo no tiene problemas de overfitting, se observa que generaliza adecuadamente a datos no vistos durante el entrenamiento. Aunado a esto, se puede ver que el MSE es relativamente bajo considerando el rango de nuestra salida EP (Fig 2.)

Pero, ¿Cómo cuantificamos esto? Sacamos nuestro RMSE (RMSE es una medida de cuál es el nivel de dispersión de estos valores residuales)[3] para ver la diferencia entre el valor predicho:

$$RMSE_{Test} = \sqrt{MSE}_{Prueba}$$

Sustituyendo con el dato que nos arrojó el código de Python:

$$RMSE_{Test} = \sqrt{24.0683}$$

$$RMSE_{Test} = \pm 4.9054$$

Ecuación 5. RMSE de prueba GD

Nuestra predicción se desvía por un valor de aproximada de 4.9054MW, un valor muy bajo

Sin embargo, aunque nuestro p_{value} se moldea a los valores en un 91% puede que estemos cayendo en problemas de Underfitting. En un problema de Machine Learning este valor sigue siendo bajo, no tenemos la suficiente certeza de que hará buenas predicciones futuras. Es ahí cuando debemos hacer mejoras al modelo para que haga mejores predicciones.

VI SET DE VALIDACIÓN Y MEJORAS - GD

Como posibles mejoras detectadas a la hora de el desarrollo de nuestro modelo de Machine Lerning se detectó que al añadir más *features* para tener una mayor cantidad de datos llega a ser favorable. Al añadir las otras dos features RH,V que también muestran una varianza moderada positiva (Fig 2.) se obtienen estos resultados:

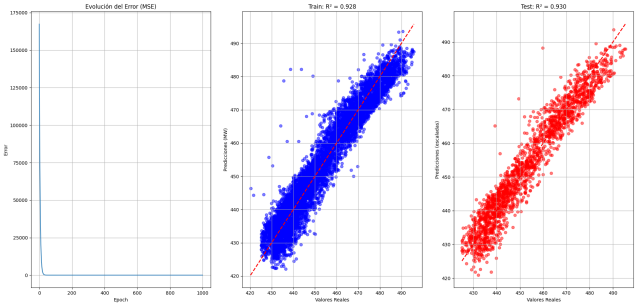


Fig. 6. Desempeño del modelo GD - Todas las features

Observando que nuestro modelo tiene un ajuste del aprox 92.5% (R²) a los datos de prueba. Dando a entender que estas variables que tenían una correlación media/baja en la salida generan un mejor desempeño en el modelo.

Cabe recalcar que esto último no en todos los casos se ha de cumplir, y que la selección de nuevas *features* para el modelo debe estar respaldada por alguna medida estadística

o por la prueba experimental para ver si es que el modelo si es mejor o es peor.

Para corroborar este último punto procedemos a hacer un slit en nuestro set de validación para tener un set de test. Un set que nos permita corroborar con nuevos datos a los no usados en entrenamiento-validación. Graficamos y obtenemos de igual manera su MSE y R². Al final, obteniendo las tres siguientes gráficas:

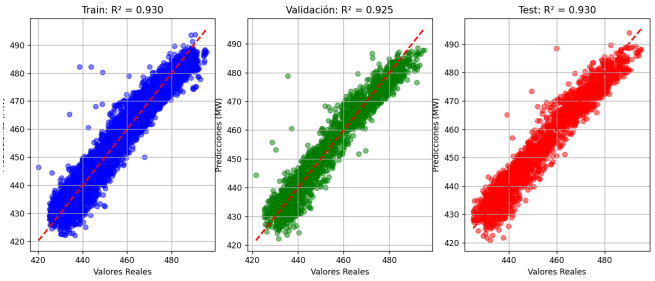


Fig. 6. Desempeño y validación del modelo GD - Todas las features

La grafica nueva de color rojo vista en la Fig.6 nos da a entender que a nuevos datos de entrada el modelo se ajusto bien, logra hacer predicciónes con un 93% de precisión en la parte de test. Siendo un resultado increíble que también, al estar probando con valores nuevos: no estamos cayendo en problemas de *Overfitting* porque nuestras predicciones responden bien a nuevas entradas.

Y al correr el código, obtenemos un MSE del 20.3016 que reinterpretando en el RMSE con la ecuación (5) tenemos que nuestras predicciones se desvían por $\pm\ 4.5057MW$, una predicción mucho mejor a la anterior y que si denota un cambio grande en la predicción inicial que hicimos solamente con dos features.

Por último, algo de lo que no nos habíamos percatado pero quería dejar para el final es la gráfica de la evolución del error. Esta última nos compara el error acumulado (MSE) en una serie de tiempo con los epochs. Al inicio dijimos que 1000 épocas sería lo suficientemente buenas para entrenar el modelo, pero ¿Qué tal si no? Bajemos este número a 200 épocas para nuestro modelo con *5 features*. Obteniendo los siguientes resultados:

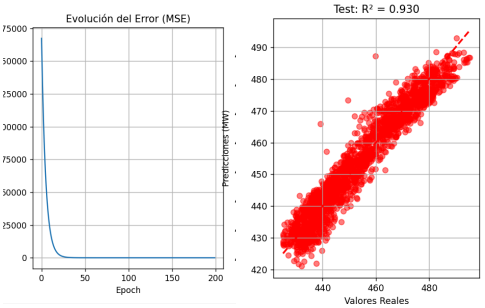


Fig. 7. Error y R2 del modelo con 200 epocas

Al observar los resultados con 200 épocas obtenemos una R² de igual, del 93%. Y un MSE en test del 20.3617. El cambio entre 1000 y 200 épocas muestra un cambio muy minúsculo y arroja el mismo desempeño en la R². Aunado a esto, el modelo presenta un bias/sesgo muy bajo. En aproximadamente ~25 épocas el MSE baja rápidamente a 0, mostrando que el modelo aprende muy rápido y llega a converger a un error bajo.

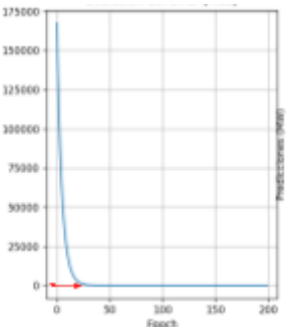


Fig. 8. Error del modelo y Validation Loss

Todo esto lo menciono porque creo importante dar la mención de que el proceso de entrenamiento debe considerarse en función también del gasto computacional. Si tengo una mejora diminuta, ¿Realmente vale la pena intentar converger en más tiempo a esa mejora? No lo creo.

Investigando parece haber métodos como el *EarlyStopping* [4] que me permite determinar el mejor número de épocas para un entrenamiento si es que no hay un cambio significativo en el modelo. Esta implementación se dejará como una posible mejora que se le puede hacer a nuestro GD para maximizar esta parte de consumo-beneficio

VII. SÍNTESIS FINAL DEL GD

A manera de síntesis el trabajo demostró exitosamente la implementación del modelo de regresión lineal utilizando el algoritmo seleccionado. Mostró que se ajusta bien a los datos dados dando un buen resultado tomando en cuenta solo dos features. Sin embargo, se concluye que también considerar otras variables , por más mínimas que puedan ser, pueden favorecer en mejores/peores resultados, ya es cuestión de analizar algunas otras métricas estadísticas que nos permitan tomar estas decisiones. Aunado a esto, la importancia de la selección de un algoritmo como SGD en comparativa a GD puede ser favorable para hacer que no se necesiten tantos epoch ni procesamiento de datos más lento; o la implementación de otras medidas como el *EarlyStopping*. Estas cuestiones dependen del tamaño de nuestros datos así como de los resultados a los que queramos llegar para poder mejorar la tasa de producción energética a cambio de un mayor gasto computacional por todo el procesamiento que hay detrás de nuestro algoritmo.

VIII. ETL- TRANSFORM - RF

Con la implementación y conclusión del Gradient Descende queremos comprobar si algún otro algoritmo de Machine Lerning puede generar mejores resultados para nuestro problema, ya que, como mencionamos anteriormente en un problema de regresión lineal 91-93% de acurrency se puede considerar Underfitting, más para nuestro problema de producción energética. En este punto, y con la ayuda de un framework para facilitar la implementación, buscamos usar otro algoritmo para la predicción de valores. Es ahí cuando se decanto por usar el Random Forest.

El Random Forests son colecciones de pequeños árboles de decisión que trabajan juntos mediante un método de ensamble (ensemble method).

Donde, posteriormente con un metodo de bagging, se inyectan sus entradas con datos aleatorios para poder hacer predicciones.

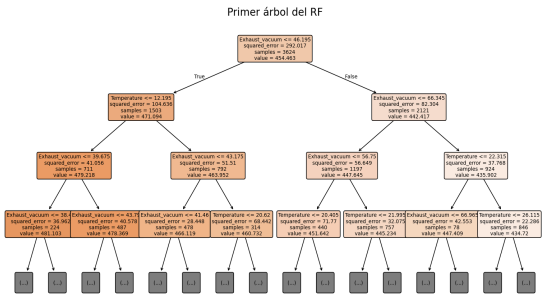


Fig. 9. Primer árbol del RF

Usando el framework de SkLearn podemos hacer una implementación rápida.

Pero antes de implementarlo, tenemos que hacer unas transformaciones en los datos, ya que el Random Forest no trabaja de la misma manera que el GD; algunas si se mantienen pero otras sí serán un poco diferentes. A continuación se enumeran:

1. Limpieza de datos

Eliminación de valores nulos: Se aplicó df.dropna() para eliminar cualquier registro con valores faltantes en el dataset (si es que los hay)
Selección de variables

Se seleccionaron todas las features de la Fig 2. sabiendo todo el trasfondo de la elección y para comparar en igualdad de condiciones la implementación final de GD después de todos los puntos que tocamos.

2. Renombre de instancias

Cambiamos los nombres de las instancias a las que hay en la documentación para tener una mejor claridad con nombres y concordancia con lo establecido

3. División de sets de datos para entrenamiento, prueba y test de resultados

Se realizó una división de los conjuntos de entrenamiento (60%), validación (20%) y prueba (20%). El set de test será crucial para ajustar hiperparámetros y detectar overfitting en el modelo.

Cómo se logra observar, prácticamente el proceso ETL de limpieza y transformación para el RF es lo mismo que el del GD. Solo que en esta implementación no es necesario redimensionar matrices o escalar valores, ya que el algoritmo a implementar, y su construcción en el framework, no lo requieren.

IX MODELADO DE LA SOLUCIÓN - RF

La implementación del algoritmo en sí es algo sencilla, no ocupamos programar alguna función adicional ya que librerías como SkLearn nos dan un método dentro de la librería que ya cuenta con el. Solo es cuestión de antes, pasar una serie de hiperparametros, declarar nuestras X´s e Y.

El entrenamiento se realizó utilizando la clase RandomForestRegressor del framework SkLearn.

```python
# Hiperparámetros

hyperparams = {

      'n_estimators': 200, #   Número de árboles

       'max_depth': 10, #   Profundidad máxima

      'min_samples_split': 5, #   Mínimo muestras para dividir

     'min_samples_leaf': 2, #  Mínimo de hojas (muestras)

            'max_features':   'sqrt',   # Características por división

              'random_state':    42,    # Reproducibilidad

     'n_jobs': -1                       #
```

```
Usar todos los núcleos de la compu para el entrenamiento

}

print("\n=== Entrenamiento del modelo ===")

rf_model = RandomForestRegressor(**hyperparams)

rf_model.fit(X_train, y_train)
```

Código 2. Implementación del Random Forest

La elección de estos parámetros se declararon de forma arbitraria teniendo en cuenta un ejemplo proporcionado por el Dr.Benjamín Valdez (ver código)

Para controlar la complejidad, regularizar el modelo, y evitar *Overfitting* se añadieron unos hiperparametros adicionales en la configuración del RF. A continuación están listados:

- min_samples_split: 5 (mínimo de muestras para dividir un nodo).
- min_samples_leaf: 2 (mínimo de muestras en un nodo hoja).

X.I RESULTADOS Y VALIDACIÓN - RF

Tras el entrenamiento, y desde un inicio contando con los sets de entrenamiento, test y validación el modelo de Random Forest mostró un desempeño excelente, superando significativamente al modelo de Gradient Descent.

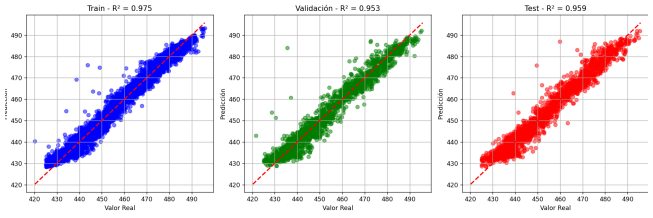


Fig. 10. Desempeño y validación del modelo RF

Obteniendo una R^2 de 0.97% en train y una R^2 de 0.59% en test. Lo que nos indica que nuestro modelo no cae en Overfitting, en general se está teniendo una muy buena respuesta a las entradas y la recta que modela a nuestros valores predichos es muy buena.

Con R² alrededor de 0.95-0.97, el modelo explica casi toda la variabilidad de los datos, lo que representa un excelente poder predictivo. Al correr el código obtenemos que:

```
=== Evaluación del modelo ===
Entrenamiento - MSE: 7.167933, R²: 0.9754
Validación    - MSE: 13.784907, R²: 0.9526
Test          - MSE: 11.929685, R²: 0.9589
```

Fig. 9. Desempeño y validación del modelo RF

Sacando el RMSE:

$$RMSE_{Test} = \sqrt{MSE} = \sqrt{11.9296} = \pm 3.4539\ MW$$

Ecuación 6. RMSE de prueba RF

Nuestra nueva predicción se desvía por apenas ~3.5 unidades. El modelo generaliza muy bien para sus salidas, de una mejor manera al GD.

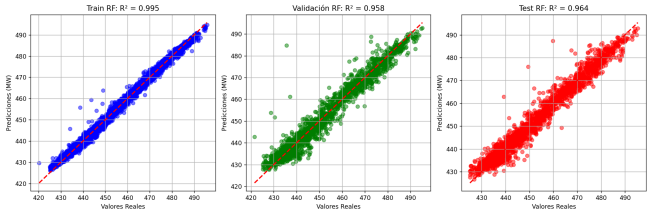
XI. MODELADO DE LA SOLUCIÓN - RF CON CV

Los hiper parámetros utilizados en la sección IX se seleccionaron de manera arbitraria basándose en el ejemplo de referencia. Si bien los resultados fueron excelentes, es muy probable que no representen la configuración óptima para este conjunto de datos específico. Para encontrar la mejor combinación de hiperparámetros de manera sistemática y científica, se empleó la técnica de Grid Search con Validación Cruzada (GridSearchCV) proporcionada por SkLearn

```
Python
grid_search.best_estimator_
```

Código 3. Función Grid_Search

Una función que automatiza la búsqueda exhaustiva sobre un rango predefinido de hiperparámetros. Al definir estos rangos y pasar los hiperparametros a sus respectivas entradas X´s obtenemos los siguientes resultados:




```
Entrenamiento - MSE: 1.573879, R²: 0.9946
Validación    - MSE: 12.359388, R²: 0.9575
Test          - MSE: 10.303490, R²: 0.9645
```

Fig. 11. Desempeño y validación del modelo RF

Por la mera naturaleza de esa R^2 perfecta resulta sospechoso que nuestro modelo no se esté sobre ajustando a los datos,

sin embargo, al ver cómo es que los datos de Train se ajustan a nuevos datos resulta fascinante cómo es que esta función optimiza de una manera perfecta todos nuestros valores, ¡Es casi como magia!. Como única desventaja podría decir que el tiempo de cómputo y recursos necesarios son absurdamente sorprendentes. Es un algoritmo bastante rudo de correr

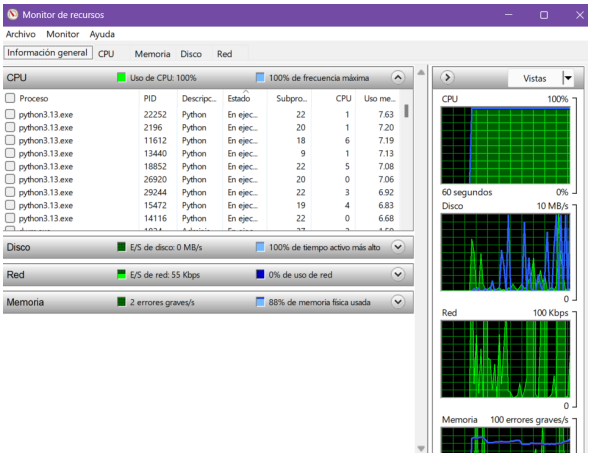


Fig. 12. Monitor de recursos (RIP PC)

XII. CONCLUSIÓN GENERAL

A lo largo del documento pudimos ver y demostrar la implementación y comparación de dos paradigmas distintos de Machine Learning para un mismo problema de regresión.

El modelo de Gradient Descent, implementado desde cero, sirvió como una excelente base pedagógica para comprender los fundamentos de la optimización y el ajuste de modelos lineales. A través de un proceso iterativo de mejora, se logró un modelo con un desempeño notable (R^2 ~0.93, RMSE ~4.5 MW), validando la hipótesis inicial sobre la relación inversa entre la temperatura y la producción de energía. Sin embargo, su rendimiento, aunque bueno, mostró un límite inherente a la capacidad de representación de un modelo lineal, incluso con pocas múltiples variables.

Por otro lado, el modelo de Random Forest (RF), aprovechando el poder de los métodos de ensamble superó significativamente al modelo lineal. Al alcanzar una R^2 prácticamente perfecta del 0.99% en su train. La optimización de hiperparámetros mediante GridSearchCV fue crucial para refinar el modelo y mostrar un mejor desempeño que las primeras implementaciones que hicimos.

Con todo lo anterior me gustaría concluir que, en problemas de Machine Lerning, donde además de variables numéricas tengamos categóricas, la elección de un RF sobre un GD nunca debe de hacerse de manera extremista. Este problema de energía, aunque desafiante en su implementación, no supone que todos los problemas puedan tener esta facilidad de poder seleccionar todas las *features* ni de que las variables se encuentren limpias en su totalidad. Es ahí cuando otras herramientas estadísticas nos dictaminaron que bueno o malo es nuestro proceso de selección a lo que queramos lograr/predecir/clasificar. De igual forma, es importante considerar que los problemas de Machine Lerning son bastante robustos y a veces un MSE, acurrency o R^2 un 2% más bajo no lo vale. La elección de diferentes técnicas no debe estar sesgada, se debe estar abierto a la experimentación y a un crudo análisis para que así: podamos ser videntes de lo que queramos c:

VIII. REFERENCIAS

[1] Combined Cycle Power plant. (2014, 25 marzo). UCI Machine Learning Repository. Recuperado 27 de agosto de 2025, de https://archive.ics.uci.edu/dataset/294/combined+cycle+power+plant

[2] Entornos virtuales y paquetes. (s. f.). Python Documentation. Recuperado 29 de agosto de 2025, de https://docs.python.org/es/3/tutorial/venv.html

[3] RMSE. (2025, 21 julio). ORACLE Help Center. Recuperado 11 de septiembre de 2025, de https://docs.oracle.com/cloud/help/es/pbcs_common/PFUSU/insights_metrics_RMSE.htm#PFUSU-GUID-FD9381A1-81E1-4F6D-8EC4-82A6CE2A6E74

[4] Team, K. (s. f.). Keras documentation: EarlyStopping. KERAS. Recuperado 11 de septiembre de 2025, de https://keras.io/api/callbacks/early_stopping/