# Evidencia 1: Implementación de técnicas de aprendizaje máquina. (Portafolio Análisis)

Guadalupe Paulina López Cuevas A01701095@tec.mx

***Resumen—***

***En este proyecto se usó el dataset Parkinson's Telemonitoring, que contiene mediciones biomédicas de la voz de pacientes con Parkinson en etapas tempranas, para predecir sus puntuaciones clínicas (motor_UPDRS y total_UPDRS). Se realizaron diferentes etapas de ETL como el preprocesamiento, incluyendo la limpieza de datos, reducción de features y análisis de correlación, para tener el dataset en las mejores condiciones para el entrenamiento del modelo. Inicialmente se implementó una regresión lineal simple, donde los resultados no fueron satisfactorios debido a la baja correlación lineal entre las variables. Posteriormente se aplicó una transformación polinomial de 2do grado que mejoró de forma considerable el rendimiento del modelo, reduciendo el error y aumentando la capacidad de predicción. Los resultados muestran que, aunque la predicción no es perfecta, el modelo puede capturar parte de la relación entre las variables. Posteriormente se implemento el modelo de random forest para conseguir un mejor resultado que permitiera predecir mejor, pero sin aprenderse los valores de entrenamientos.***

## I. Introducción

El estudio de la enfermedad de Parkinson a través de señales de voz representa una herramienta importante para el diagnóstico y seguimiento de los pacientes, ya que la voz refleja directamente los cambios motores característicos de esta condición. El dataset utilizado en este proyecto recopila más de cinco mil grabaciones de voz obtenidas con telemonitoreo, permitiendo un registro continuo y no invasivo del progreso de la enfermedad. Con estas grabaciones se obtienen diversos features como jitter, shimmer, NHR/HNR y métricas no lineales que miden irregularidades en la voz.

El objetivo principal fue analizar el dataset, realizar el proceso de limpieza y selección de variables relevantes, y finalmente implementar un modelo de regresión que permita predecir la severidad de los síntomas del Parkinson. Se evaluaron tanto la regresión lineal simple como una regresión polinomial, con el objetivo de determinar qué tan bien estos enfoques pueden ajustarse a la complejidad de los datos.

## II. Información sobre dataset

El dataset "Parkingsons telemonitoring" es un data set proporcionado por Oxford en el que la información corresponde a un estudio que se hizo sobre la progresión de la enfermedad de Parkinson en etapas tempranas. Se recopilaron un total de 5,875 grabaciones de voz de 42 pacientes, a los cuales se les realizo un telemonitoreo desde sus casas por seis meses. El objetivo principal de todos estos datos fue predecir las puntuaciones clínicas de la enfermedad (motor_UPDRS y total_UPDRS) basándose en las medidas biomédicas extraídas de la voz.

El dataset cuenta con variables relacionadas a características acústicas como variaciones de frecuencia, amplitud, complejidad no lineal y la relación entre componentes tonales y de ruido. Algunos conceptos importantes de saber sobre los atributos que se midieron:

- **Jitter (Estabilidad del tono):** analizan cambios en la frecuencia fundamental (pitch) de la voz. Si hay mucha variación, la voz suena temblorosa o inestable.
- **Shimmer (Estabilidad del volumen):** mide que tan estable es el volumen de la voz. Una persona con Parkinson suena "temblorosa" en intensidad de la voz.
- **NHR/HNR:** Proporción de ruido vs claridad en la voz.
- **RPDE, DFA, PPE**: métricas más avanzadas que miden la complejidad e irregularidad de la voz.

Este dataset cuenta con los siguientes 22 atributos:

| | | |
|---|---|---|
| subject# (sin unidad) | Integer | ID único de cada paciente |
| age (medida en años) | Integer | Edad del paciente |
| sex (sin unidades) | Binary | Genero (0 = masculino, 1= femenino) |
| test_time (medida en días) | Continuous | Tiempo transcurrido desde el inicio de la prueba (en días). |
| Jitter(%) (voces normales < 1%) | Continuous | Variación % de la frecuencia de la voz. (mide que tan estable es la voz). |
| Jitter(Abs) (segundos(s) o milisegundos (ms), valores normales ~10-100 ms) | Continuous | Variación absoluta (en segundos) de la frecuencia de la voz. |
| Jitter:RAP (%) (valores altos = voz inestable) | Continuous | Mide la irregularidad de la voz promediando las variaciones entre ciclos consecutivos. |
| Jitter:PPQ5 (%) (valores altos = voz inestable) | Continuous | Mide variaciones de tono, pero considerando |

| | | |
|---|---|---|
| | | promedios en ciclos de 5 vibraciones. |
| Jitter:DDP (%) (valores altos = voz inestable) | Continuous | 3 veces el valor de RAP, otra forma de medir la irregularidad del tono. |
| Shimmer (%) (valores normales< 3-4%) | Continuous | Variación relativa en la intensidad de la voz. Mide que tan uniforme es la voz. |
| Shimmer(dB) | Continuous | Shimmer expresado en decibeles. |
| Shimmer:APQ3 (%) (valores normales< 3-4%) | Continuous | Variación de amplitud promedio en 3 ciclos consecutivos. |
| Shimmer:APQ5 (%) (valores normales< 3-4%) | Continuous | Variación de amplitud promedio en 5 ciclos consecutivos. |
| Shimmer:APQ11 (%) (valores normales< 3-4%) | Continuous | Variación de amplitud promedio en 11 ciclos consecutivos. |
| Shimmer:DDA (%) (valores normales< 3-4%) | Continuous | Similar a APQ3, pero calculado de otra manera. |
| NHR (Noise-to-Harmonics Ratio) (Unidad: proporción (adimensional), valores normales > 0.2) | Continuous | Relación entre el ruido y los componentes tonales de la voz. Un valor alto significa mas ruido en la voz. |
| HNR (Harmonics-to-Noise Ratio) (Unidad: decibeles, valores normales > 20dB) | Continuous | Relación entre los componentes tonales y el ruido. Es inverso a NHR: valores altos significa una voz más clara. |
| RPDE (Recurrence Period Density Entropy) (Unidad: adimensional (0-1)) | Continuous | Mide la complejidad de la señal de voz. Detecta irregularidades en los patrones de vibración de las cuerdas vocales. |
| DFA (Detrended Fluctuation Analysis) (Unidad: adimensional, valores típicos 0.5 y 1.5) | Continuous | Exponente que mide el componente fractal de la voz (que tan caótica o regular es). |
| PPE (Pitch Period Entropy) (Unidad: adimensional, valores altos = mayor irregularidad) | Continuous | Mide la variabilidad no lineal del tono. Cuanto más alto, mas irregular es la frecuencia. |
| motor_UPDRS (Escala numérica clínica: Indice) | Continuous (Target) | Puntuación clínica que mide la severidad de los síntomas motores del Parkinson (rigidez, temblores, lentitud de movimiento) |
| total_UPDRS (Escala numérica clínica: menos revero 0 – 108 más severo) | Continuous (Target) | Puntuación clínica total que evalúa tanto síntomas motores como no motores. |

## III. Entorno para trabajar

Lo primero de queremos hacer es crear un entorno en donde podamos trabajar todo nuestro proyecto.

```
# Crear environment para el proyecto
python -m venv yenv

# Activar environment (Windows)
yenv\Scripts\activate
```

*Código. 1. Crear un entorno de trabajo*

De igual forma debemos instalar (de ser necesario) e importar las librerías necesarias para poder trabajar con nuestro dataset, operaciones, transformaciones, funciones, graficas, etc:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import math
```

*Código. 2. Librerías para el proyecto*

En este caso se usara Visual Code como herramienta de trabajo donde tendremos la terminal tanto para correr el programa como para tener el proyecto en GitHub.

## IV. Extracción

Para poder utilizar un dataset que nos funcione para un modelo de técnicas de aprendizaje de maquina tenemos que hacer varios pasos antes de poner empezar a entrenar nuestro modelo, en este caso es la extracción y limpieza de dichos datos. Al extraer los datos, ya sea de un Excel, .csv, .data o cualquier tipo de archivo, los ponemos en nuestro espacio de trabajo para poder visualizarlos y poder manipularlos para nuestra limpieza del dataset. De igual forma una vez extraídos es importante conseguir la información del dataset ya que esto nos da todos los atributos dentro del dataset, la cantidad de datos que hay (que nos servirá saberlo posteriormente), si hay valores nulos, y tipo de dato.

## V. Limpieza de datos

Para la limpieza del dataset se hicieron varias transformaciones para asegurar que los datos que hay, features, son los mejores.

### *V.I Null count*

Primero se revisa si el dataset tiene algún valor nulo que al momento de manipular los datos provoque problemas.

### *V.II Remove null*

En caso de que haya valores nulos, estos se deben remover. En caso de este dataset no existieron valores nulos, pero de igual forma se eliminó cualquier valor que pudiera ser nulo.

### *V.III Remove duplicate*

De igual forma se tiene que revisar si el dataset cuenta con valores duplicados que puedan afectar al momento de hacer el entrenamiento del modelo.

### *V.IV Remove columns (subject#, test_time)*

Una vez que se limpia el dataset, se deben quitar las columnas (features) que no serán relevantes para el entrenamiento ni para las predicciones. En este caso las columnas que fueron removidas fueron las siguientes:

- subject#: Se remueve porque es el ID del paciente (valor unico).
- test_time: Se remueve porque es el tiempo transcurrido desde que inicio la prueba (valor no relevante en la prueba).

### *V.V Quitar outliers*

Muchas veces en los datsets existen valores que son comportamientos fuera de los patrones o errores de medición, etc. Por estos valores que se encuentran muy lejos de demás dataset es necesario quitarlos. La manera en la que se quitaron los outliers(valores fuera del patrón normal), por medio de calcular los percentiles entre el 25% y el 75%, los demás fueron despreciados, ya sea que se encontraran en el extremo izquierdo (valores muy negativos) o en el extremo derecho (valores muy positivos).

### *V.VI Hacer shuffle de datos*

Finalmente se realiza un shuffle al dataset ya que en este caso este contiene varias pruebas sobre las mismas 42 personas y es necesario mezclar para que el modelo no entrene a partir de algunos pacientes y al hacer la pruebas sea con un paciente diferente, sino que el modelo pueda ver a todos los pacientes, pero no todas sus pruebas.

Una ves finalizada la limpieza de los datos, en este caso el dataset termina con las siguientes dimensiones:

4511 rows x 20 columns

*Ecuación. 1. Dimensiones del dataset después de limpieza*

## VI. Correlación entre features y targets

Para este dataset la distribución es de la siguiente manera:

- Features: ['age', 'sex', 'Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'JitteR:PPQ5', 'Jitter:DDP', 'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5', 'Shimmer:APQ11', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE']
- Targets:[ 'motor_UPDRS', 'total_UPDRS',]

El siguiente paso, después de la limpieza de datos, es buscar cual es la correlación entre los features(X) y el target(Y). Para este dataset la matriz de correlación se vio de la siguiente manera:

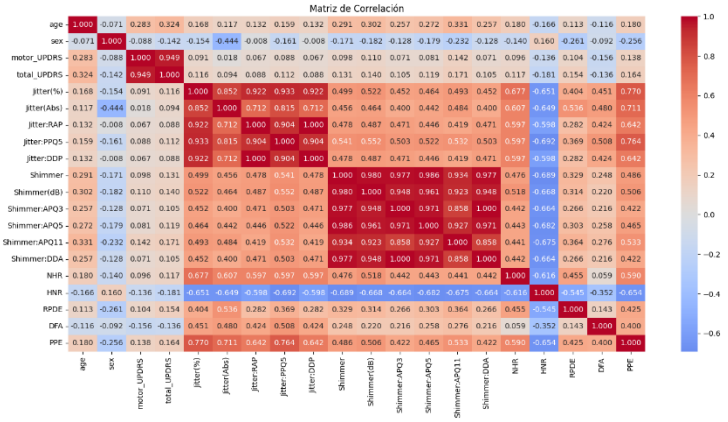


*Fig. 1. Matriz de correlación (18 features, 2 targets)*

Como se puede observar en la Fig.1, en la tercera y cuarta columna y fila se encuentran nuestros targets. La manera de interpretar esta matriz es observando los recuadros de estas columnas y filas y recorrerlos completos para ver cuales features están con un color muy oscuro (rojo o azul), esto nos dice que tanta correlación existe. Para este dataset se pude ver que si hay correlación entre los features y los targets, pero esta relación es muy baja lo cual puede perjudicar el modelo de entrenamiento.

Existe otra grafica que puede ayudar a visualizar de mejor manera si los datos de los features tienen una correlación lineal con los targets. Para este dataset el plot de pares se vio de la siguiente manera:

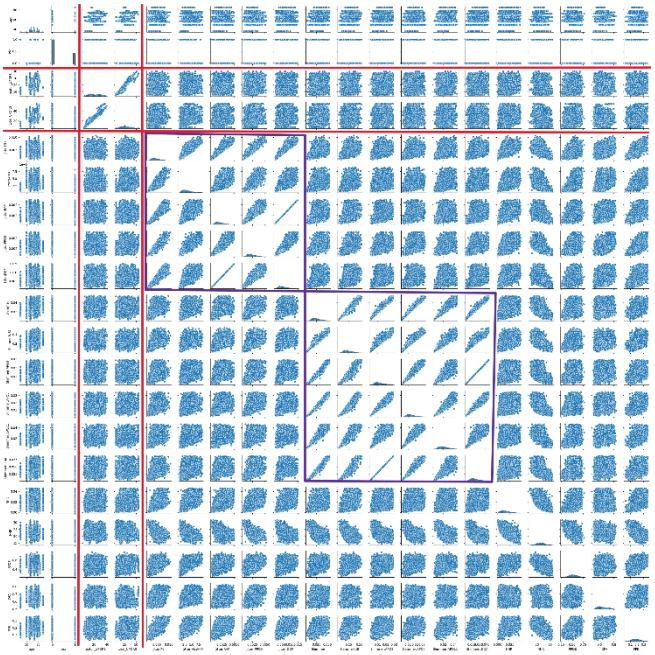

*Fig. 2. Gráfica de pares (18 features, 2 targets)*

Dentro de la Fig.2 se puede ver entre las líneas rojas la correlación que tienen cada uno de los features con los targets. Se puede ver que el comportamiento de cada feature con respecto a cada target NO es lineal, esto puede ser debido a que los valores tienen mucha variabilidad entre ellos, es por eso que en la matriz de correlación los valores son muy pequeños.

La manera ideal como se debería de ver la correlación entre los features y los targets es como se puede ver en los recuadros morados, esas graficas donde los valores "siguen" una diagonal significa que tienen una correlación lineal.

En este dataset, no existe una relación lineal entre features y targets, pero si entre algunos features entre sí. Para el entrenamiento de un modelo, que los features tengan una correlación lineal tan grande mientras que no exista mucha correlación lineal entre features y targets se vuelve un problema llamado multicolinealidad, así que lo que se procederá a hacer, que podría ser una recomendación, quitar algunos de los features que tengan relación entre si para disminuir esta correlación y que no perjudique el modelo.

## VII. REDUCCIÓN DE FEATURES

Para la selección de los features hay que basarse en la matriz de correlación para seleccionar las que tienen una correlación mas alta con los targets. No es necesario que todas estén fuertemente correlacionadas, pero aquellas con mayor correlación positiva o negativa suelen ser las mas relevantes para el modelo. Hacemos un nuevo dataset con los features de mayor correlación y volvemos a sacar la matriz de correlación como se muestra a continuación:

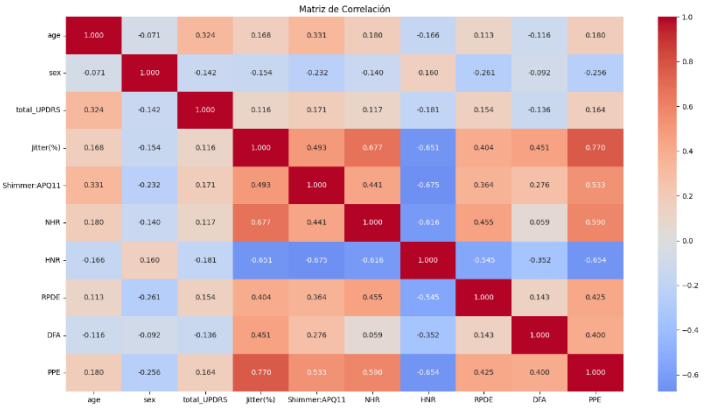


*Fig. 3. Matriz de correlación (9 features,1 target)*

Como se observa en la Fig.3, se redujeron considerablemente los features y targets (en este caso solo se usará un target). Sigue sin haber una correlación lineal muy fuerte entre estos features pero son los que tienen la mayor correlación. El dataset quedo reestructurado de la siguiente manera:

- Features: ['age', 'sex', 'Jitter(%)', 'Shimmer:APQ11', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE']
- Target: ['total_UPDRS']

4511 rows x 10 columns

*Ecuación. 2. Dimensiones del dataset reducido*

De igual forma se usará otra grafica que ayudará a visualizar de mejor manera la correlación lineal entre los features reducidos y el target. Para este dataset reducido el plot de pares se vio de la siguiente manera:

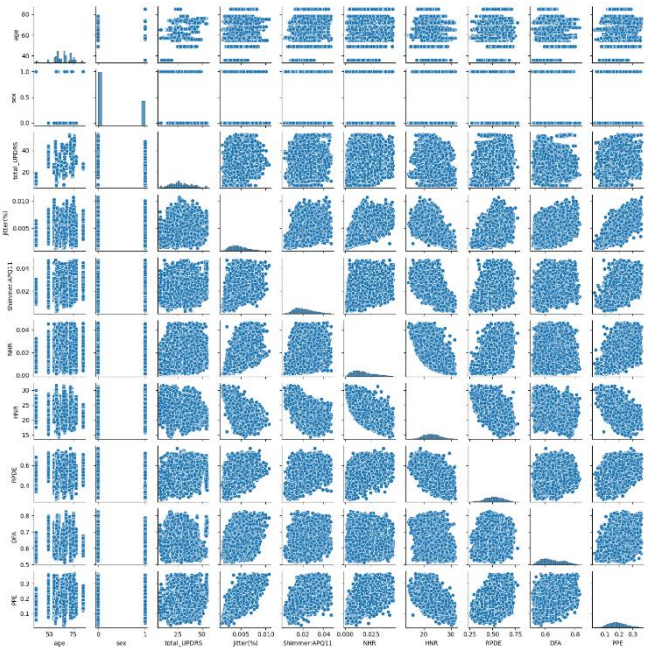

*Fig. 4. Gráfica de pares (9 features,1 target)*

En la Fig.4 se ve la correlación lineal que anteriormente se vio en el dataset completo, solo que ahora se puede ver que en

general no existe más esa correlación lineal tan fuerte entre los features. Teniendo el dataset listo se puede empezar a usar para el aprendizaje del modelo que se implementara.

## VIII. MODELO DE ENTRENAMIENTO

Antes de poner el dataset dentro del modelo se debe separar el dataset en train, val y test. Esto debido a que para poder hacer un buen entrenamiento del modelo es necesario que al entrenarlo no se utilicen todos los datos del dataset para que el modelo no "aprenda" los valores y que al momento de hacer el val y el test salga que tuvo un buen rendimiento cuando simplemente se terminó ajustando a los valores de entrenamiento.

Para este dataset se hizo una división en una proporción de 60/20/20 donde habrá 2705 muestras para entrenamiento (60%), 903 muestras para validación (20%) y 903 muestras para prueba (20%), para que nuestro split siempre sea el mismo utilizamos una semilla de "random_state = 42".

$$2705 + 903 + 903 = 4511$$

*Ecuación. 3. Tamaño del dataset a trabajar*

Una vez teniendo los datasets de entrenamiento, validación y prueba, el siguiente paso es hacer una última transformación a los datos para poder meterlos al modelo.

### *VIII.I Escalamiento*

Para los features, los valores independientes, es necesario escalarlos debido a que muchas veces los features cuentan con escalas diferentes, pueden ser valores muy grandes o pequeños, y de esta manera ningún feature va a tener mayor o menor peso que influya en el target. En este caso para el escalamiento se uso la siguiente función:

```
scaler = StandardScaler()
df_x_train = scaler.fit_transform(x_train)
df_x_test = scaler.transform(x_test)
```

*Código. 3. Función para escalamiento*

Esta función hace una transformación matemática, que tiene su media = 0, una desviación estándar = 1 y hace que los datos sigan una distribución normal estándar, donde cada columna se transforma usando la siguiente ecuación:

$$z = (x - \mu) / \sigma$$

**x**: Valor original del dato
**μ**: Media (promedio) de la característica
**σ**: Desviación estándar de la característica

*Ecuación. 4. Ecuación de escalamiento*

### *VIII.II Cambio de dimensiones*

Para el target, valor dependiente, es necesario cambiarle las dimensiones debido a que como esta en una lista y no en una matriz, debemos convertir una serie de pandas a un array puro, cambiando las dimensiones de la siguiente manera:

$$(n, 1) => (n,)$$

*Ecuación. 5. Cambio de 2D a 1D*

Ahora si podemos empezar con el modelo de entrenamiento. Este dataset es un problema de regresión por lo que para poder resolverlo es necesario implementar una regresión lineal que es conocida por la siguiente formula:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n + b$$

**$\hat{y}$** = Predicciones (vector de shape (m,))
**X** = Matrix de features (shape (m, n))
**θ** = Parámetros del modelo (vector de weights, shape (n,))
**b** = Término de bias (escalar)

*Ecuación. 6. Función de regresión lineal*

Teniendo nuestra función de hipótesis podemos crear nuestro modelo de entrenamiento con las siguientes funciones:

### *VIII.IV Función de Hipótesis*

hyp_theta(): Calcula la hipótesis de la regresión lineal, genera la predicción lineal para cada instancia de datos del modelo, haciendo el producto punto entre los features y los parámetros (θ o w) más el término bias (b), durante el proceso de entrenamiento.

$$hyp\_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

*Ecuación. 7. Función de hipótesis*

### *VIII.V Función de Mean Square Error (MSE)*

MSE(): Calcula el Error Cuadrático Medio que cuantifica el costo (error) del modelo. Mide la diferencia al cuadrado entre la predicción (ŷ) y los valores reales (y).

$$J(\theta) = (1/2m) * \Sigma(hyp\_\theta(x^{(i)}) - y^{(i)})^2$$

*Ecuación. 8. Función de mean square error*

### *VIII.VI Función de Gradient Descent (GD)*

update(): Implementa el algoritmo de Gradiente Descendiente. Actualiza de manera iterativa los parámetros theta (θ) y bias (b) para ajustar sus valores durante el entrenamiento.

$$\theta_j := \theta_j - \alpha * (1/m) * \Sigma(hyp\_\theta(x^{(i)}) - y^{(i)}) * x_j^{(i)}$$
$$b := b - \alpha * (1/m) * \Sigma(hyp\_\theta(x^{(i)}) - y^{(i)})$$

*Ecuación. 9. Funciones de gradient descent para* theta (θ) y bias (b).

### *VIII.VII Función de $R^2$ Score*

Calcula el coeficiente de determinación $R^2$ que mide la proporción de la varianza en los features. Indica que porcentaje de la variabilidad de los datos es aprendida por el modelo.

Ya teniendo las funciones del modelo, es momento de definir los parámetros iniciales para el entrenamiento:

```
theta_W = np.zeros(len(df_x_train.shape[1]))
error = []      # Error
b = 100         # Bias
alfa = 0.01     # Learning rate
```

```
epoch = 100     # Epocas
```

*Código. 4. Inicialización de parámetros*

Como se ve en el Código.4, tanto los parámetros de theta_w como b no importa cuál sea su valor inicial debido a que durante el entrenamiento estos se irán ajustando. El arreglo vacío de los errores para irlo llenando conforme se vaya entrenando el modelo. Para el learning rate, alfa = 0.01, se puso ese valor debido a que la ser un hiperparametro, este controla el tamaño de los pasos que da el algoritmo durante el entrenamiento. Las épocas van a empezar en 100 para ver el comportamiento del modelo, posteriormente de ser necesario se aumentará.

## IX. Entrenamiento del Modelo

Después del entrenamiento del modelo estos fueron los resultados:

```
Epoch 100: MSE = 399.034822, R-squared = -5.688124

Resultados finales después de 100 épocas:
Error final: 399.034822
Theta final: [ 1.86225932 -0.64207698  0.16735363
  0.50818107 -1.23674468  0.5286094 ]
b final: 54.79
```

*Fig. 5. Resultados de entrenamiento*

En la Fig.5 se puede ver claramente que el modelo no está entrenando bien y se encuentra demasiado alejado de los valores a predecir, además de que la $R^2$ es muy negativa indicando que el modelo no saber predecir los valores porque no existe una correlación lineal. Esta prueba de igual manera se intentó con epoch = 400, y el resultado no bajo de manera significativa.

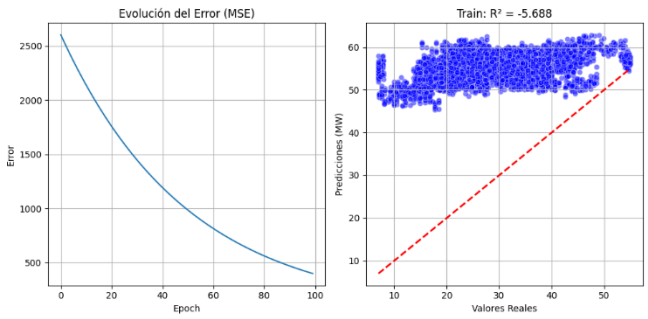


*Fig. 6. Gráficas del error y Predicción vs Valor Real en entrenamiento.*

En la Fig.6 se puede observar que, aunque el error está disminuyendo, el modelo no se acerca para nada a los valores. Este es uno de los riesgos al saber que los features no cuentan con una correlación lineal fuerte con respecto al target. De igual forma se puede observar cómo es que el error en el entrenamiento tiene un sesgo grande debido a que la curva del error pasa muy lejos del origen, lo que significa que tarda mucho en aprender para reducir el error. A pesar de que la gráfica de error no presenta oscilaciones y de una manera suave reduce significativamente el error, la curva se encuentra muy lejos de un error cercano a cero lo que significa que este entrenamiento no es muy preciso en predicciones del entrenamiento. De igual forma en la gráfica no se aprecia un overfitting del modelo, sin embargo, esto no es significativo ya que el error es de 399.03. De esta forma debemos hacer modificaciones para que el error se reduzca y ya después observar si existe underfitting, fit u overfitting.

## X. Transformación a Términos Polinomiales

Debido a que, como se mencionó anteriormente, la correlación lineal entre los features y el target no es lineal una solución o propuesta que se puede hacer es hacer nuevamente una transformación a los features pero en este caso convertirlos a términos polinomiales para intentar ver si así el modelo puede acercarse de manera más precisa a los valores reales del target.

Lo que se procederá hacer es usar el dataset reducido que hicimos anteriormente y crear un nuevo dataframe con los polimonios (2do grado) también, solo los features deben ser transformados y posteriormente se vuelve a agregar dentro del dataframe el target, teniendo como resultado un dataset con las siguientes dimensiones:

4511 rows x 55 columns

*Ecuación. 10. Dimensiones del dataset después de limpieza*

La razón por la que en este caso solo se transforman a un polinomio de 2do orden es porque tampoco se quiere un overfitting por parte del modelo y que se sobreajuste a los datos.

Nuevamente volvemos hacer una división en una proporción de 60/20/20 donde habrá 2705 muestras para entrenamiento (60%), 903 muestras para validación (20%) y 903 muestras para prueba (20%), para que nuestro split siempre sea el mismo utilizamos una semilla de “random_state = 42”.

Hacemos nuevamente el escalamiento (features) y el cambio de dimensión (target), como se hizo previamente en el primer dataset reducido, para ajustar los valores y las dimensiones necesarias para reentrenar el modelo.

Volvemos a definir los parámetros para el entrenamiento:

```
theta_W = np.zeros(len(df_x_train.shape[1]))
error = []      # Error
b = 100         # Bias
alfa = 0.01     # Learning rate
epoch = 3000    # Epocas
```

*Código. 5. Inicialización de parámetros nuevos*

En este caso el único parámetro que se cambio fue las épocas que se llevaran a cabo. Para las pruebas, además de cambiar el número de épocas, también se cambió el learning rate a valores como alfa = 0.1, alfa = 0.001 pero en el modelo el MSE se disparaba y tenía un error gigantesco o los valores de predicción

se volvían demasiado grandes, por lo que al final alfa = 0.01 se quedó como el hiperparametro.

## XI. Resultado de entrenamiento

Después retransformar los datos, ahora si se ponen a reentrenar de nuevo.

```
Resultados finales después de 3000 épocas:
Error final: 44.525225
Theta final: [ 0.          1.49412797 -0.48514373 -0.28718689 -1.1940406  -0.41181206
  -0.07405148 -0.3555951  -0.50770325 -1.08879838 -0.36315473  0.02096855
   0.10974389  0.84565088 -0.91764186  0.82162337 -0.7448633   0.67202244
   2.7668688  -0.48514373  3.57277776  0.08055622  0.20320201 -0.87374554
  -0.65508334  0.10467448 -2.38333137 -1.489874    1.24293998  1.37388363
   0.40942457 -0.2309396  -0.8911447   0.11336381  2.91524431 -2.24292529
  -1.48569076 -3.03231617 -0.66838463  3.15217238  0.00932539  2.90022461
  -1.72423285 -1.03511447 -0.43076504 -1.99123232  1.67568178 -1.72963428
  -1.44568759  0.13005538  1.95529377 -1.22857873 -1.28700894 -0.45676852
   2.451483  ]
b final: 28.75
```

*Fig. 7. Resultados de reentrenamiento*

Como se puede ver en la Fig.7 el MSE bajo de manera considerable de 399.03 a 44.52, donde la puntuación clínica del parkinson tiene un rango de 0-180 pero los valores rondan ~28-37, lo que significa que el modelo mejoro muchísimo tanto con los valores polinomiales como con la cantidad de épocas a ejecutar. De igual forma se puede ver que la $R^2$ ahora si muestra que el modelo si está haciendo predicciones y que si puede existir una correlación lineal baja, pero si existe.

```
Epochs =  3000
Epoch 100: MSE = 395.415959, R-squared = -5.672828
Epoch 200: MSE = 94.456331, R-squared = -0.593994
Epoch 300: MSE = 53.920133, R-squared = 0.090073
Epoch 400: MSE = 48.289550, R-squared = 0.185091
Epoch 500: MSE = 47.352843, R-squared = 0.200899
```

*Fig. 8. Entrenamientos primeros 500 Epochs*

En la Fig.8 se observan los resultados de los primeros 500 epochs donde se puede ver como después las 200 epochs el modelo mejora considerablemente y disminuye mucho el MSE a 53.92 y el $R^2$ se vuelve positivo en 0.090, lo que significa que el modelo a partir de este punto está aprendiendo a predecir los valores.

```
Epoch 2500: MSE = 44.848145, R-squared = 0.243167
Epoch 2600: MSE = 44.778449, R-squared = 0.244343
Epoch 2700: MSE = 44.711462, R-squared = 0.245473
Epoch 2800: MSE = 44.647026, R-squared = 0.246561
Epoch 2900: MSE = 44.584992, R-squared = 0.247608
Epoch 3000: MSE = 44.525225, R-squared = 0.248616
```

*Fig. 9. Entrenamientos últimos 500 Epochs*

En la Fig.9 se observan los resultados de los últimos 500 epochs de 3000 epochs, en este caso se puede ver como ya en estas últimas iteraciones el MSE y el $R^2$ no cambian mucho, siguen cambiando, dando a entender que el modelo sigue entrenando lo mejor que puede para ser lo más preciso, pero ya no hay mucho cambio. Al final del entrenamiento el MSE llego a 44.52 y el $R^2$ llego a 0.248, estos valores representan que aunque el error haya bajado a comparación de las primeras iteraciones y el porcentaje de predicción haya subido, el modelo no esta aprendiendo lo suficiente para que sea significativo el entrenamiento.

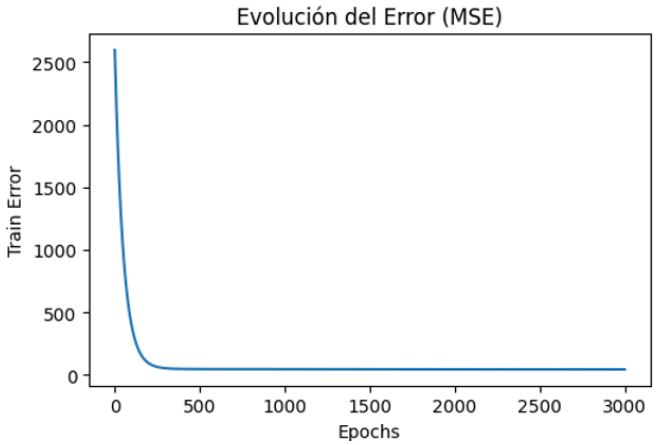


*Fig. 10. Evaluación del error en el entrenamiento*

Como se puede ver en la Fig.10 la gráfica del error está bajando de manera más considerable a comparación de la del primer entrenamiento. En este caso se puede ver que el sesgo bajo se ve más bajo a diferencia del primer entrenamiento, aunque hay que considerar que en este segundo entrenamiento el número de epochs es hasta 3000, es por esto que la gráfica parece que tiene un sesgo bajo, pero en realidad se puede ver que la gráfica va acercándose al origen casi hasta después de 250 epochs transcurridos. Como ya se vio anteriormente esto es debido a que el modelo no esta aprendiendo lo suficiente y tiene un error grande.

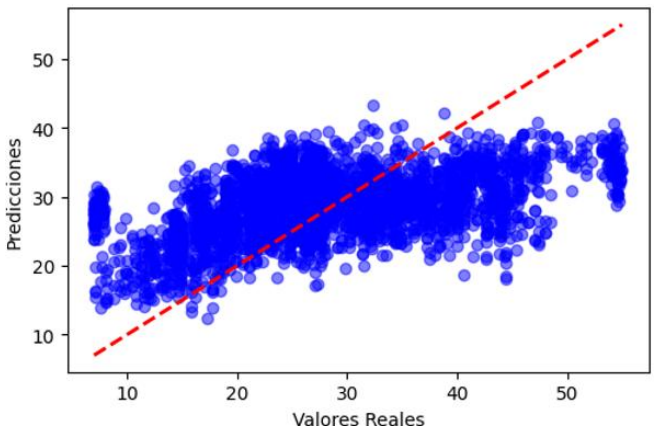


*Fig. 11. Gráfica Predicción vs Valor real en entrenamiento*

Como se puede ver en la Fig.11 la línea roja ahora ya se encuentra encima de los valores de nuestro dataset, a pesar de que no todos los valores se encuentran cerca de la línea, muchos de ellos si están cerca de ella. Esta gráfica muestra la visualización de que el modelo no se ajusta bien a los datos de

entrenamiento, se puede ver que hay mucha dispersión entre los datos y la recta que los modela.

## XII. PROBAR MODELO CON DATASET DE VAL Y TEST

Ahora es momento de hacer el val y el test con lo demás del dataset para saber cómo se comportará nuestro modelo ya entrenado.

```
Epoch 5: MSE = 73.149713, R-squared = 0.235173

Error Cuadrático Medio (MSE) en el conjunto de validación: 73.149713
```

*Fig. 12.1 Resultados de validación*

Como se ve en la Fig.12.1 los resultados de la validación son un poco distintos al entrenamiento, en este caso el error es mucho más grande que el error en el entrenamiento (44.52 donde la puntuación clínica del parkinson tiene un rango de 0-180 pero los valores rondan ~28-37), lo que significa que el modelo está presentando sobreajuste. A pesar de que no es un error muy pequeño o una $R^2$ sea cercana a 1, se mantienen dentro de los valores que se esperaban para un modelo entrenado para predecir features y tergets con baja correlación lineal.

```
Epoch 5: MSE = 47.335874, R-squared = 0.192790

Error Cuadrático Medio (MSE) en el conjunto de prueba: 47.335874
```

*Fig. 12.2 Resultados de prueba*

Como se ve en la Fig.12.2 los resultados de la prueba son bastante similares a los del entrenamiento, lo que significa que, a pesar de que la validación presenta sobreajuste, aquí no se presenta ese sobreajuste y predice igual que en el entrenamiento. A pesar de que no es un error muy pequeño o la $R^2$ sea cercana a 1, se mantienen dentro de los valores que se esperaban para un modelo entrenado para predecir features y tergets con baja correlación lineal.

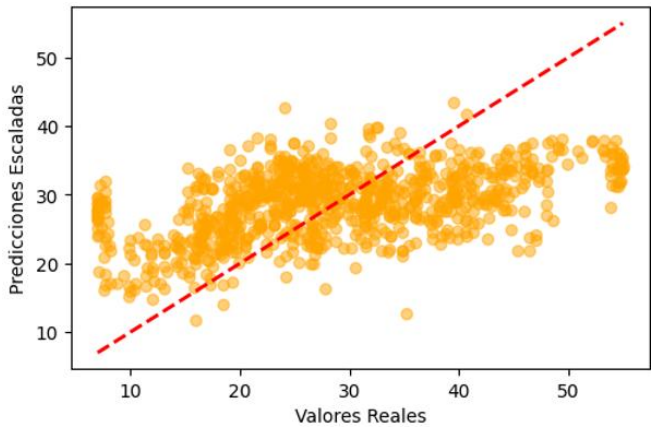


*Fig. 13.1 Grafica Predicción vs Valor real en validación*

Como se observa en la Fig.13.1 el resultado es muy similar al del entrenamiento, a pesar de que la línea roja no representa por completo el comportamiento de todo el dataset logra predecir algunos valores. De igual forma se observa que el modelo no se ajusta bien a los datos de entrenamiento, se puede ver que hay mucha dispersión entre los datos y la recta que los modela.

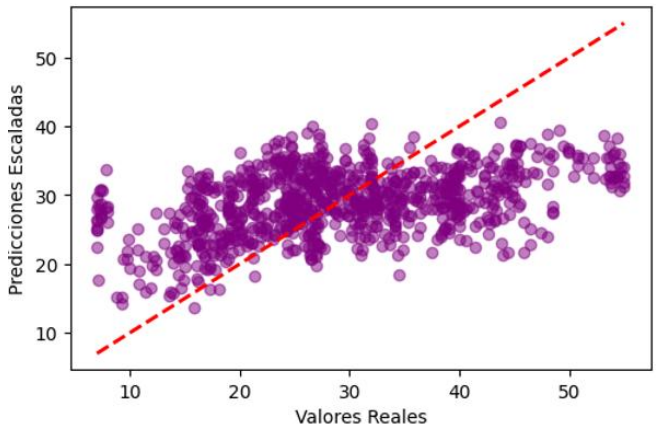


*Fig. 13.2 Grafica Predicción vs Valor real en prueba*

Como se observa en la Fig.13.2 de igual forma se observa que el modelo no se ajusta bien a los datos de entrenamiento, se puede ver que hay mucha dispersión entre los datos y la recta que los modela.

| | MSE | $R^2$ | bias o sesgo | varianza |
|---|---|---|---|---|
| Train | 44.5252 | 0.2486 | Alto | Bajo |
| Val | 73.1497 | 0.2351 | Alto | Bajo |
| Test | 47.3358 | 0.1927 | Alto | Bajo |

Como se observa en la tabla previa, estos fueron los resultados del entrenamiento del modelo, que funcionan para saber que para este dataset el modelo no esta funcionando para predecir. La manera en la que se puede saber si el modelo presenta sesgo es a través de la $R^2$, si este valor se encuentra por debajo de ~0.3-0.4 el modelo cuenta con un sesgo alto (underfitting) no se esta ajustando en lo absoluto al dataset, por ende la varianza es baja debido a que la varianza funciona para saber si el modelo se sobreajusta (overfitting) a los valores del train y este no es el caso.

## XIII. CONCLUSIONES: ALGORITMO DE REGRESIÓN LINEAL

Durante el desarrollo de este proyecto me permitió comprender de manera práctica las etapas necesarias para llevar a cabo el proceso de ETLs para el aprendizaje automático: desde la exploración y limpieza del dataset hasta la implementación y evaluación de modelos predictivos. Los resultados obtenidos para este dataset muestran que la regresión lineal simple no es suficiente para capturar la relación entre las variables acústicas y las puntuaciones clínicas del Parkinson, ya que la correlación lineal es baja entre los features y el target. Sin embargo, al transformar los datos en términos polinomiales, el modelo logró

una mejora considerable en el error y en la capacidad de hacer predicciones, lo cual confirma que para la resolución del problema se requieren si o si métodos que consideren relaciones no lineales.
Sabiendo que el modelo polinomial aún presenta limitaciones, con este trabajo puedo darme cuenta de que efectivamente en problemas reales de análisis y aprendizaje de datasets complejos, relaciones no lineales, son necesarias las redes neuronales. Aunque en este trabajo no se han implementado, se puede ver el potencial de la precisión de este tipo de datasets con redes neuronales y de igual manera entrenamientos multihilos y por minibatches para hacerlos más rápidas, eficientes y precisas las predicciones. Con esto se puede ver la importancia del preprocesamiento de datos y la selección adecuada del modelo para abordar problemas reales de predicción de datos complejos.

## SEGUNDA PARTE: IMPLEMENTACIÓN FRAMEWORK

Para esta segunda parte, el objetivo es la implementación de un modelo a través de un framework que permita que a través de un nuevo entrenamiento se pueda predecir con mayor precisión y menor error los valores de prueba.

Sabiendo con anterioridad que el dataset tiene características específicas que se lograron identificar previamente cuando se entrenó el primer modelo a través del algoritmo de regresión lineal, la poca correlación lineal entre features y targets y las multicolinealidad entre los features, que dificultaron el procesamiento del dataset y fue necesario hacer una seria de pasos para la limpieza y ajustes adecuados de los datos para que se pudieran implementar en el entrenamiento del modelo, lo ideal es buscar un modelo al que estas características no le perjudiquen el aprendizaje y aun así pueda predecir los valores con un error bajo pero sin llegar al overfitting, debido al aprendizaje de los valores. Tomando en cuenta estas consideraciones un modelo que puede abordar estas limitaciones es el modelo Random Forest debido a las características con las que cuenta este algoritmo. Para poder entender esto es necesario saber cómo funciona este algoritmo.

## XIV. RANDOM FOREST

La manera en la que funciona el Random Forest es en la construcción de muchos árboles de decisión que se entrenan sobre diferentes subconjuntos de datos y variables, del dataset original, y combina los resultados haciendo un promedio.

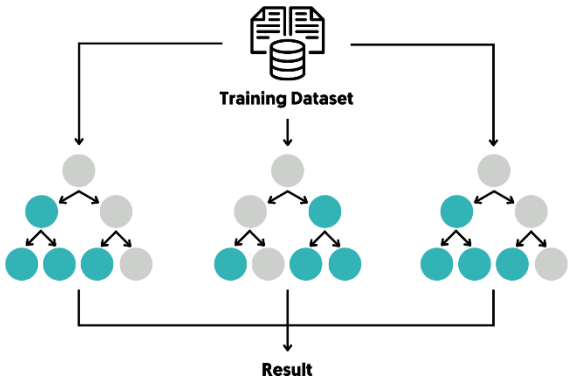


*Fig. 14. Random Forest algorithm*

Esta metodología de funcionamiento permite que sea muy robusto para problemas de dataset que puedan tener multicolinealidad o datos con relaciones no lineales porque no depende de la existencia de correlaciones lineales entre los features y el target para poder hacer predicciones. Otra de las características del Random Forest es que no es sensible al ruido presente en el dataset ya que cuando promedia los resultados de muchos árboles, esto reduce la varianza y se vuelve mas estable el modelo. También el Random Forest es bien conocido por evitar el overfitting en un modelo ya que tiene un buen equilibrio entre sesgo y varianza. Estas características son lo que hacen que este modelo sea una opción adecuada para poder procesar el dataset que tenemos y logremos un entrenamiento con mejor capacidad de predicción ante nuevos datos.

## XV. LIMPIEZA DE DATOS

Como se menciono anteriormente para este tipo de algoritmo no son necesarias tantas transformaciones como se hizo en la parte anterior, sin embargo, es necesario hacer nuestra limpieza del dataset. Para la limpieza del dataset se hicieron varias transformaciones similares a las que se hicieron en la parte anterior.

### *XV.I Null count*

Primero se revisa si el dataset tiene algún valor nulo que al momento de manipular los datos provoque problemas.

### *XV.II Remove null*

En caso de que haya valores nulos, estos se deben remover. En caso de este dataset no existieron valores nulos, pero de igual forma se eliminó cualquier valor que pudiera ser nulo.

### *XV.III Remove duplicate*

De igual forma se tiene que revisar si el dataset cuenta con valores duplicados que puedan afectar al momento de hacer el entrenamiento del modelo.

### *XV.IV Remove columns (subject#, test_time, motor_UPDRS)*

Una vez que se limpia el dataset, se deben quitar las columnas (features) que no serán relevantes para el entrenamiento ni para las predicciones. En este caso las columnas que fueron removidas fueron las siguientes:

- subject#: Se remueve porque es el ID del paciente (valor unico).
- test_time: Se remueve porque es el tiempo transcurrido desde que inicio la prueba (valor no relevante en la prueba).
- Motor_UPDRS: Se remueve debido a que para este entrenamiento solo se tomara en cuenta un target a predecir.

### *XV.VI Hacer shuffle de datos*

Finalmente se realiza un shuffle al dataset ya que en este caso este contiene varias pruebas sobre las mismas 42 personas y es necesario mezclar para que el modelo no entrene a partir de algunos pacientes y al hacer la pruebas sea con un paciente diferente, sino que el modelo pueda ver a todos los pacientes, pero no todas sus pruebas.

Una vez realizada la limpieza de los datos podemos proceder al siguiente paso.

## XVI. Implementación de modelo con framework

Antes de poner el dataset dentro del modelo se debe separar el dataset en train, validation y test. Esto debido a que para poder hacer un buen entrenamiento del modelo es necesario que al entrenarlo no se utilicen todos los datos del dataset para que el modelo no "aprenda" los valores y que al momento de hacer el validation y el test salga que tuvo un buen rendimiento cuando simplemente se terminó ajustando a los valores de entrenamiento.

Para este dataset se hizo una división en una proporción de 60/20/20 donde habrá 3525 muestras para entrenamiento (60%), 1175 muestras para validación (20%) y 1175 muestras para prueba (20%), para que nuestro split siempre sea el mismo utilizamos una semilla de "random_state = 42".

$$3525 + 1175 + 1175 = 5875$$

*Ecuación. 11. Tamaño del dataset a trabajar*

Una vez teniendo los datasets de entrenamiento, validación y prueba, el siguiente paso es hacer una última transformación a los datos para poder meterlos al modelo.

### *XVI.I Escalamiento*

Para los features, los valores independientes, aunque no es necesario escalarlos, esto puede beneficiar a las predicciones del modelo. En este caso para el escalamiento se uso la siguiente función:

```
scaler = StandardScaler()
df_x_train = scaler.fit_transform(x_train)
df_x_test = scaler.transform(x_test)
```

*Código. 6. Función para escalamiento*

Esta función hace una transformación matemática, que tiene su media = 0, una desviación estándar = 1 y hace que los datos sigan una distribución normal estándar, donde cada columna se transforma usando la siguiente ecuación:

$$z = (x - \mu) / \sigma$$

**x**: Valor original del dato
**μ**: Media (promedio) de la característica
**σ**: Desviación estándar de la característica

*Ecuación. 12. Ecuación de escalamiento*

Una vez teniendo las ultimas transformaciones de los datasets ya se puede empezar a entrenar el modelo. Debido a que se quiere comparar el MSE y la accuracy del entrenamiento y de la validación debemos crear arreglos que nos permitan guardar los valores calculados para sacar la curva de aprendizaje del modelo al predecir los valores del train vs val. Para esto lo que se hace es evaluar el rendimiento del modelo con subconjuntos de datos de distintos tamaños con los conjuntos de train y val para observar si el modelo tiene algún beneficio al tener más datos cada vez.

## XVII. Comparación train vs validation

Para el entrenamiento de este modelo estos fueron los hiperparametros que se tuvieron configurados:

```
model_rf    =    RandomForestRegressor(n_estimators=400,
max_leaf_nodes=10, n_jobs=-1, random_state=42)
```

*Código. 7. Hiperparametros para Random Forest*

Estos parámetros se escogieron para tener un número alto de árboles (más posibles soluciones), limitar la complejidad del árbol (para que no pueda extenderse tanto) y controlar el número de nodos hoja (para evitar overfitting). Valores proporcionados en una actividad de Random Forest que se tomaron para prueba del modelo.

Para saber cuál fue el comportamiento del modelo y que nos dicen los datos es necesario saber los tipos de resultados que podemos obtener a partir de los resultados numéricos y gráficos del modelo y que nivel de ajuste tiene el modelo.

Existen tres tipos de nivel de ajuste para los modelos que explican el comportamiento de las predicciones y cómo se comportan frente a datos nuevos.

### *XVII.I Underfitting (Subajuste)*

Se le llama underfitting cuando el modelo es demasiado sencillo para capturar los patrones del dataset, su desempeño es bajo tanto en train como en val y test.

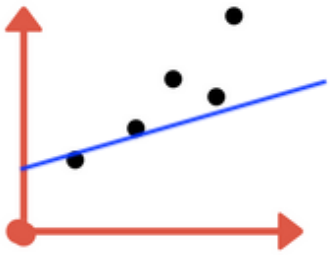

*Fig. 15. Underfitting*

### *XVII.II Fit (Buen ajuste)*

Se le llama fit cuando el modelo logra un equilibrio y aprende los patrones principales del dataset del train logrando generalizar para datos nuevos (val y test).

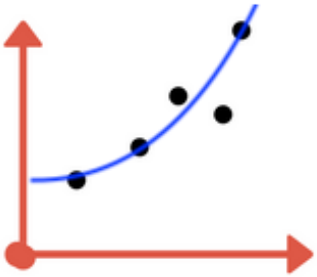

*Fig. 16. Fit*

### *XVII.III Overfitting (Sobreajuste)*

Se le llama overfitting cuando el modelo es demasiado complejo y aprende no solo los patrones sino hasta a veces el ruido del dataset, teniendo un buen desempeño en el train pero uno malo en el val y test.

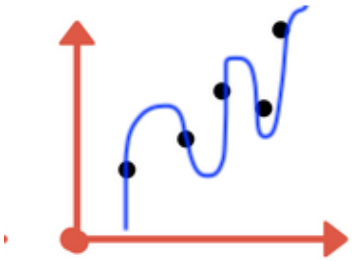

*Fig. 17. Underfitting*

### *XVII.IV Bias o sesgo*

El bias o sesgo se refieren a los errores que comete el modelo al simplificar demasiado el problema, esto puede pasar cuando el modelo ignora relaciones importantes entre features y target teniendo predicciones alejadas de las deseadas. Si se tiene un sesgo alto la tendencia es caer en underfitting.

### *XVII.V Varianza*

La varianza se refiere a la sensibilidad del modelo a los datos de train, esto aumenta cuando se ajusta demasiado a valores específicos y al cambiar los valores no los pueda predecir. Si se tiene una varianza alta la tendencia es caer en overfitting.

Después del entrenamiento del modelo estos fueron los resultados de train y val:

```
MSE Train: 48.1024
MSE Val: 50.5037
R² Train: 0.5810
R² Val: 0.5523
```

*Fig. 18. Resultados de entrenamiento*

En la Fig.18 se puede ver claramente que el modelo no está entrenando bien debido a que el valor del error (donde la puntuación clínica del parkinson tiene un rango de 0-180 pero los valores rondan ~28-37) en el entrenamiento es muy grande al igual que en la validación, a pesar de ser grandes ambos valores son similares lo que significa que por el momento no existe un sobreajuste del modelo (overfitting). Las $R^2$ a pesar de ser positivas, indican que el modelo solo sabe predecir casi el 50% de las veces.

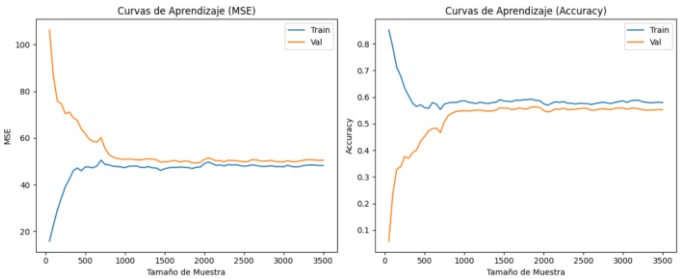


*Fig. 19. Gráficas del MSE y Accuracy (Train vs Val).*

En la Fig.19 se puede observar que, al principio por tener un tamaño pequeño de dataset al momento de entrenar el modelo, este sufre de underfitting (subajuste) debido a que se muestra que el train MSE es muy bajo y la $R^2$ es alta mientras que val MSE es muy alto y la $R^2$ es baja, no se ajusta a los patrones del dataset. Posteriormente ambas curvas de aprendizaje mejoran al ir aumentando el tamaño del dataset, sin embargo, en cierto punto se queda estancados en un valor lo que hace que su desempeño se vuelva limitado (MSE ≈ 50, $R^2$ ≈ 0.55), a pesar de que no llega a un error muy bajo o un accurracy muy alto, ambos valores de train y val terminan teniendo una brecha muy pequeña entre ambos (varianza muy baja) lo que significa que el modelo no está sufriendo de un overfitting (sobreajuste) fuerte, no está memorizando los valores del train. Esto se puede deber a que la complejidad de los árboles se está restringiendo (max_leaf_nodes=10) y el modelo no puede explicar de mejor manera el comportamiento porque está limitado. En este modelo

podemos percibir que tiene un sesgo alto, lo que significa que el modelo en un principio cae en underfitting (subajuste) que no permite que se ajuste a los valores, en este caso la varianza es baja ya que tanto train como val terminan en valores muy cercanos lo que significa que no tenemos overfitting (sobreajuste) no se aprende los valores de train.

Una de las primeras cosas que se debe de hacer cuando el modelo no está prediciendo de manera adecuada es configurar o ajustar los hiperparemetros para que estos ayuden al modelo a adaptarse mejor y predecir mejor sin hacer overfitting.

## XVIII. AJUSTE DE PARAMETROS

Para el ajuste de los parámetros debemos de saber primero para que sirve cada uno de los que tenemos a disposición:

| Hiperparametro | Función |
|---|---|
| n_estimators | Número de árboles en la configuración. Tener más árboles mejora la estabilidad del modelo, pero aumenta tiempo de entrenamiento. |
| max_depth | Profundidad máxima de cada árbol, Un valor muy pequeño puede llevar al underfitting, y un valor muy alto puede llevar al overfitting |
| max_features | Número de features que se consideran en cada división. Controla diversidad de los árboles: valores bajos aumenta variabilidad y reduce correlación entre ellos. |
| min_samples_split | Número mínimo de muestras necesarias para dividir un nodo. Evita que el modelo cree divisiones copo útiles con pocos datos. |
| min_samples_leaf | Número mínimo de muestras que debe haber en cada hoja final. Si es > 1, obliga a que cada predicción se base en más datos, evita overfitting. |
| max_leaf_nodes | Número máximo de hojas por árbol. Limita directamente la complejidad de los árboles. |
| bootstap | Indica si los árboles se entrenan con muestras aleatorias con reemplazo del conjunto de entrenamiento. |
| n_jobs | Número de procesadores a usar en paralelo. Si es -1, utilizas todos los disponibles. |

En este caso se quieren utilizar los parámetros que permitan una mejor predicción, más rápida y que no llegue al overfitting. Para esta mejora del modelo se usarán los siguientes hiperparametros:

```
model_rf_v2 = RandomForestRegressor(n_estimators=1000,
max_depth=12, min_samples_leaf= 3, min_samples_split= 5,
random_state=42)
```

*Código. 8. Hiperparametros para Random Forest versión 2*

El ajuste de los hiperparametros funciona como una forma de regularización con el objetivo de penalizar o limitar la complejidad del modelo para que generalice mejor y cada una de las predicciones no se basen en aprenderse los valores del entrenamiento.

- ***n_estimators=1000:*** Tener un número alto de árboles permite tener estabilidad y reducir la varianza del modelo, con esto se pueden tener predicciones robustas.
- ***max_depth=12:*** Con esto se limito la profundidad de cada árbol para evitar que el modelo memorice los valores de dataset de entrenamiento (overfitting) pero también mantener suficiente complejidad para aprender bien.
- ***min_samples_leaf=3:*** Con esto se establece tener un mínimo de 3 muestras por hoja para reducir la probabilidad del sobreajuste (overfitting) en los nodos terminales.
- ***min_samples_split=5:*** Con esto se establece un mínimo de 5 muestras para dividir un nodo, obligando al modelo a tener divisiones mejor fundamentadas y que no genere ramas muy especificas que afecten para generalizar las predicciones.
- ***random_state=42:*** Asegurar que a pesar de ser aleatorio los resultados sean los mismos y con esto asegurar que los resultados tienen sentido.

## XIX. IMPLEMENTACIÓN DEL MODELO AJUSTADO

De igual forma para poner el dataset dentro del modelo mejorado se debe separar el dataset en train, validation y test. Se van a volver a usar los dataset que se hicieron previamente con una división en una proporción de 60/20/20 donde habrá 3525 muestras para entrenamiento (60%), 1175 muestras para validación (20%) y 1175 muestras para prueba (20%), para que nuestro split siempre sea el mismo utilizamos una semilla de "random_state = 42".

$$3525 + 1175 + 1175 = 5875$$

*Ecuación. 13. Tamaño del dataset a trabajar versión 2*

Al igual que en el modelo pasado, se escalaron los features para un mejor desempeño del modelo.

Una vez teniendo las ultimas transformaciones de los datasets ya se puede empezar a entrenar el modelo. Volveremos a evaluar el rendimiento del modelo con subconjuntos de datos de distintos tamaños con los conjuntos de train, val y test para observar si el modelo tiene algún beneficio al ajustar los hiperparametros.

## XX. COMPARACIÓN TRAIN VS VALIDATION VS TEST

Después del entrenamiento del modelo estos fueron los resultados de train y val:

```
MSE Train: 3.3856364675535806
MSE Val: 9.53394807361668
MSE Test: 10.707635032097759
R² Train: 0.9705085036582103
R² Val: 0.9154927543664407
R² Test: 0.9068207688732209
```

*Fig. 20. Resultados de entrenamiento mejorado*

En la Fig.20 se puede ver claramente que el modelo mejoro significativamente, el train MSE bajo de 48.10 a 3.33 (donde la puntuación clínica del parkinson tiene un rango de 0-180 pero los valores rondan ~28-37) y su $R^2$ subió de 0.58 a 0.97, esto significa que para el dataset de train el modelo fue muy bien entrenado. Por otro lado, tenemos los valores de val MSE (9.53) y test MSE (10.70) que al ser considerablemente diferentes al error en el train se puede inferir que el modelo ahora tiene un poco de overfitting (sobreajuste), es decir, aunque logra predecir muy bien los valores de val y test teniendo un error bastante pequeño el modelo aprendió algunos valores de train y por eso train tiene un error mucho más bajo. Sin embargo, para val $R^2$ (0.91) y test $R^2$ (0.90) son valores bastante significativos ya que a pesar de que en el error se ve que existe un poco de overfitting, las predicciones llegan a ser bastante acertadas y comprobando que el modelo está funcionando mucho mejor que cualquiera de los modelos previamente implementados en este dataset.

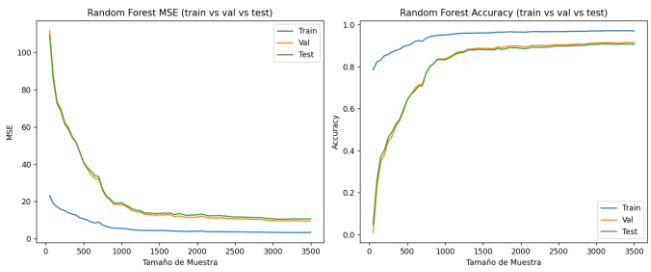


*Fig. 21. Gráficas del MSE y Accuracy (Train vs Val).*

En la Fig.21 se puede observar que como, al igual que el modelo anterior, estas predicciones comienzan con underfitting (subajuste) debido a que al usar pocas muestras para el entrenamiento el modelo no logra predecir adecuadamente. Una vez teniendo suficientes muestras para el entrenamiento del modelo, se puede ver como en train el error baja muy bien, mientras que, para val y test, teniendo un comportamiento muy parecido, logran tener un error bajo, pero justamente al final de la gráfica se puede ver que efectivamente este modelo presenta overfitting (sobreajuste) que se visualiza con la separación que se encuentra, a partir del tamaño de muestras 1500, entre el error de train y los errores de val y test. Para el accuracy, se puede ver que de igual manera los resultados tardan en tener un buen desempeño, pero a partir del tamaño de muestra de 1000, los resultados son muy buenos ya que superan el 90% de predicciones acertadas en cualquiera de los tres datasets, pero de igual forma existe la separación entre el accuracy de train y val y test que significa el overfitting (sobreajuste) del modelo.

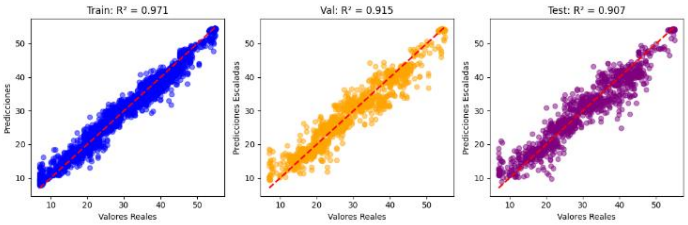


*Fig. 22. Gráficas del MSE y Accuracy (Train vs Val).*

En la Fig.22 se puede observar que aun cuando el modelo muestra un comportamiento de overfitting(sobreajuste) a los valores de entrenamiento ahora si se puede ver que en cada uno de los distintos datasets el modelo predice de manera significativa los valores:

- Train: el modelo logra representar el 97.1% de la varianza de este dataset, lo que significa que los puntos están muy alineados a la línea roja que ayuda a visualizar la precisión.
- Val: el modelo logra representar el 91.5% de la varianza de este dataset, lo que significa que los puntos tienen un poco más de dispersión, pero aun así tiene un buen nivel de generalización para valores no vistos en el entrenamiento.
- Test: el modelo logra representar el 90.7% de la varianza de este dataset, muy similar al de validación, lo que significa que en realidad el modelo no se encuentra sobreajustado ya que es capaz de predecir bien a datos nuevos y no antes vistos.

| | MSE | $R^2$ | bias o sesgo | varianza |
|---|---|---|---|---|
| Train | 3.3856 | 0.9705 | Bajo | Bajo |
| Val | 9.5339 | 0.9154 | Bajo | Bajo |
| Test | 10.7076 | 0.9068 | Bajo | Bajo |

Como se observa en la tabla, estos fueron los resultados del entrenamiento del modelo mejorado, funcionan para saber que para este dataset el modelo está funcionando para predecir. Como se mencionó en la primera implementación la manera en la que se puede saber si el modelo presenta sesgo es a través de la $R^2$, en los resultados se observa que los tres datasets diferentes presentan un valor superior a 0.9 lo que significa que el sesgo es bajo y el modelo si esta entrenando bien y prediciendo de manera adecuada los valores, por ende la varianza es baja debido a que la diferencia del porcentaje de la

$R^2$ es menor al 0.1 lo que significa que el modelo se entreno bien y no esta memorizando los datos del entrenamiento, no hay overfitting. Después de este análisis se puede concluir que este modelo tiene un nivel de ajuste del fit.

## XXI. Conclusión

Después de las implementaciones de los distintos modelos en el mismo dataset, las comparaciones permitieron observar que aunque el modelo de regresión lineal proporciona aproximaciones útiles, la falta de correlaciones lineales fuertes limita significativamente su precisión. Si bien la regresión con términos polinomiales mostró mejoras respecto al modelo lineal al reducir el error y aumentar el coeficiente $R^2$, los resultados demostraron que no son suficientes para modelar la complejidad de los datos biomédicos del Parkinson, debido a la baja correlación entre features y target, lo que resultó en un modelo con un sesgo alto (underfitting) y un desempeño limitado ($R^2 \approx 0.25$, MSE ≈ 44-73).

Por otro lado, el análisis de los resultados de la segunda implementación con Random Forest demostró que es posible superar estas limitaciones mediante enfoques más avanzados. La aplicación de técnicas de regularización mediante ajuste de hiperparámetros demostró resultados significativamente superiores, permitiendo al modelo alcanzar un equilibrio óptimo entre sesgo y varianza. El modelo optimizado evitó tanto el underfitting (subajuste) como el overfitting (sobreajuste), obteniendo un ajuste adecuado (fit) con alta capacidad de generalización. Los valores de $R^2 > 0.90$ y MSE ≈ 10 en validation y test corroboran la efectividad del buen desempeño del modelo para capturar patrones complejos en conjuntos de datos con comportamiento anormal y alta dispersión, confirmando que los modelos basados en árboles pueden capturar de manera más efectiva la complejidad inherente a los datos biomédicos del Parkinson.

## XXII. Referencias


- Tsanas, A., & Little, M. A. (2009). *UCI Machine Learning Repository: Parkinson's Telemonitoring Data Set*. University of California, Irvine. Recuperado de https://archive.ics.uci.edu/ml/datasets/parkinsons+telemonitoring
- Ruder, S. (2017). *An overview of gradient descent optimization algorithms*. arXiv:1609.04747. https://arxiv.org/abs/1609.04747
- Brownlee, J. (2021). *Linear Regression for Machine Learning*. Machine Learning Mastery. https://machinelearningmastery.com/linear-regression-for-machine-learning/
- Scikit-learn developers. (2023). StandardScaler documentation. *Scikit-learn: Machine Learning in Python*. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- Frost, J. (2023). *How to Interpret R-squared in Regression Analysis*. Statistics By Jim. https://statisticsbyjim.com/regression/interpret-r-squared-regression/
- Scikit-learn. (s.f.). *RandomForestRegressor — scikit-learn 1.5.0 documentation*. Recuperado en septiembre de 2025 de https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
- Bagnato, J. I. (2017, 12 de diciembre; actualizado 22 de marzo de 2021). *Qué es overfitting y underfitting y cómo solucionarlo*. Aprende Machine Learning. Recuperado de https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/
- Brownlee, J. (2019, 17 de julio). *A Gentle Introduction to the Bias-Variance Trade-Off in Machine Learning*. Machine Learning Mastery. Recuperado de https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/