



# Tecnológico de Monterrey

**Inteligencia Artificial Avanzada para la Ciencia de Datos**

Raúl Emiliano Guzmán Acevedo A01754602

ITESM CEM

Grupo: 101

Instituto Tecnológico y de Estudios Superiores de Monterrey

## Introducción

El código presentado realiza un análisis de regresión lineal para predecir el tiempo de vuelta de carros de carreras basado en su peso, y posteriormente clasifica si un vehículo es un carro de carreras o no. La regresión lineal se implementa utilizando un algoritmo de descenso de gradiente, y se evalúan los resultados mediante una matriz de confusión.

## Importación de Bibliotecas y Carga de Datos

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, precision_score, recall_score, f1_score, accuracy_score

# Importar el set de datos para su correcto manejo
df = pd.read_excel('/Users/raulguzman/Documents/Escuela/Septimo/proyectos/IA-Avanzada-Modulo-2/IA-Avanzada-Modulo-2/carros.xlsx')
df
```

	Peso (kg)	Tiempo de vuelta (s)	Carro de carreras
0	1099.632095	63.771502	No
1	1560.571445	136.369249	No
2	1385.595153	97.722718	Sí
3	1278.926787	121.028483	Sí
4	924.814912	168.907977	No
...	...	...	...
95	1195.036477	101.905149	Sí

- **NumPy y Pandas:** Se utilizan para la manipulación y análisis de datos.
- **Matplotlib:** Se usa para la visualización de resultados.
- **scikit-learn:** Provee herramientas para la partición de datos y la evaluación del modelo mediante métricas de rendimiento y la matriz de confusión.

## Preprocesamiento de Datos

```
"""
Se dividiendo los datos en dos conjuntos, uno de entrenamiento y otro de prueba.
Se selecciona el peso del carro como variable independiente y el tiempo de vuelta como variable dependiente.

Se separan los datos en 70% para entrenamiento y 30% para prueba.
"""
x = np.array(df['Peso (kg)'])
y = np.array(df['Tiempo de vuelta (s)'])
carro = np.array(df['Carro de carreras'])

# Separar los datos en entrenamiento y prueba.
x_train, x_test, y_train, y_test, carro_train, carro_test = train_test_split(x, y, carro, test_size=0.3, random_state=42)
```

Las características (x), el objetivo (y) y las etiquetas (carro) se extraen del DataFrame. Posteriormente, los datos se dividen en conjuntos de entrenamiento y prueba, reservando el 30% de los datos para la prueba.

## Implementación del Algoritmo de Descenso de Gradiente

```
"""
Definimos los parámetros iniciales para el modelo de regresión.
```

```
Parametros:
```

- m: La pendiente del modelo.
- b: La línea y que intersecta con la regresión lineal.
- epochs: Iteraciones que se hacen.
- alpha: El learning rate para el descenso de gradiente.
- tolerance: La tolerancia para la convergencia del modelo.
- n\_t: Número de datos de entrenamiento.

```
"""
```

```
#Parametros iniciales para el modelo de regresion
```

```
m = 0
```

```
b = 0
```

```
epochs = 10000
```

```
alpha = 0.0000001
```

```
tolerance = 1e-6
```

```
n_t = len(x_train)
```

Python

El algoritmo de descenso de gradiente se utiliza para encontrar la pendiente (m) y la intersección (b) de la recta que mejor ajusta los datos de entrenamiento

```
"""
```

```
Algoritmo de descenso de gradiente
```

```
Este código implementa el algoritmo de descenso de gradiente para ajustar una línea recta a un conjunto de datos.
```

```
Utiliza la fórmula  $y = mx + b$  para calcular la predicción y el error.
```

```
Luego, calcula el gradiente de los parámetros m y b y los actualiza utilizando la tasa de aprendizaje alpha.
```

```
El algoritmo se repite durante un número determinado de épocas previamente establecidas hasta que se alcanza la convergencia o se supera la tolerancia.
```

```
Durante la ejecución, se imprimen los valores de los parámetros cada 100 iteraciones.
```

```
"""
```

```
for epoch in range(epochs):
```

```
    # Calculo de la predicción y el error
```

```
    y_pred = m * x_train + b
```

```
    error = y_pred - y_train
```

```
    # Calculo del gradiente
```

```
    grad_m = (1/n_t) * sum(error * x_train)
```

```
    grad_b = (1/n_t) * sum(error)
```

```
    # Actualización de los parámetros
```

```
    m = m - alpha * grad_m
```

```
    b = b - alpha * grad_b
```

```
    # Verificar la convergencia
```

```
    if abs(alpha*grad_m) < tolerance and abs(alpha * grad_b) < tolerance:
```

```
        print('Converge en la iteración:', epoch)
```

```
        break
```

```
    # Imprimir valores de los parámetros cada 100 iteraciones
```

```
    if epoch % 100 == 0:
```

```
        print(f'Iteración: {epoch+1}, m = {m}, b = {b}')
```

El algoritmo itera hasta 1000 veces o hasta que la actualización de los parámetros sea menor que la tolerancia definida, al finalizar, se imprimen los valores finales de m y b

## Clasificación y Evaluación

```
# Clasificador
pred_train = m * x_train + b # Predicciones para los datos de entrenamiento
clasificador_carr_t = np.where(y_train > pred_train, "Sí", "No")
matriz = confusion_matrix(carro_train, clasificador_carr_t, labels=["Sí", "No"])
disp = ConfusionMatrixDisplay(confusion_matrix=matriz, display_labels=["Carro de carreras", "No es carro de carreras"])
```

Python

```
disp = disp.plot(cmap=plt.cm.Blues)
plt.title("Carros")
plt.show()
```

Python

```
# Precision, recall y f1-score
precision = precision_score(carro_train, clasificador_carr_t, pos_label="Sí")
recall = recall_score(carro_train, clasificador_carr_t, pos_label="Sí")
f1 = f1_score(carro_train, clasificador_carr_t, pos_label="Sí")
```

```
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-score: {f1}')
```

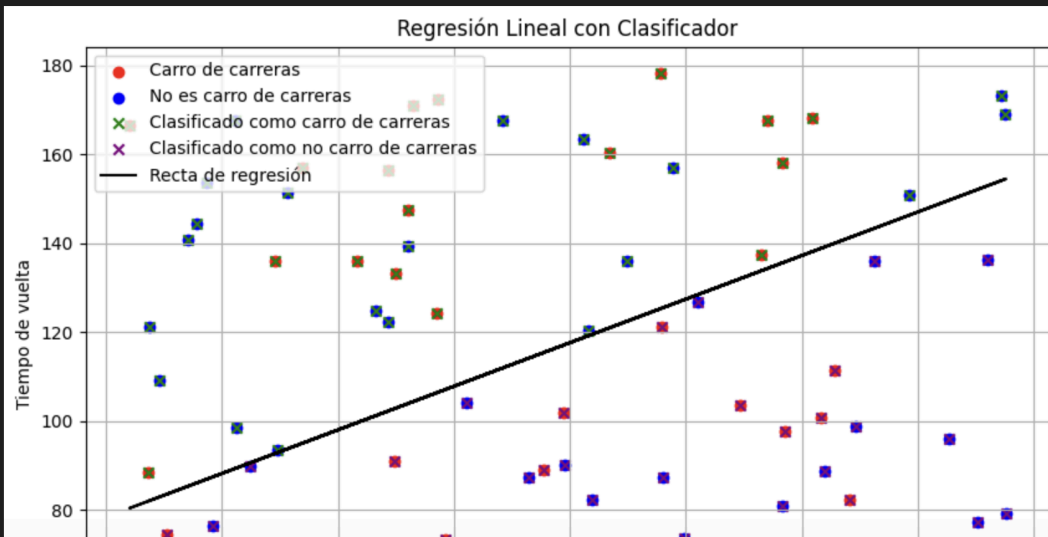
Se calcula la predicción y se evalúa el modelo.

## Gráfica

```
plt.scatter(x_train[carro_train == 'Sí'], y_train[carro_train == 'Sí'], color='red', label='Carro de carreras')
plt.scatter(x_train[carro_train == 'No'], y_train[carro_train == 'No'], color='blue', label='No es carro de carreras')
plt.scatter(x_train[clasificador_carr_t == 'Sí'], y_train[clasificador_carr_t == 'Sí'], color='green', marker='x', label='Clasifica')
plt.scatter(x_train[clasificador_carr_t == 'No'], y_train[clasificador_carr_t == 'No'], color='purple', marker='x', label='Clasifica')
plt.plot(x_train, pred_train, color='black', label='Recta de regresión')

plt.xlabel('Peso')
plt.ylabel('Tiempo de vuelta')
plt.title('Regresión Lineal con Clasificador')
plt.legend()
plt.grid(True)
plt.show()
```

Python



Se repite el proceso con el set de prueba de los datos.

## Análisis del primer modelo.

Como parte del proceso de la creación de estos modelos se nos solicitó un análisis más detallado sobre los resultados obtenidos.

Para empezar dentro del código se puede notar la separación de los datos en lo siguiente:

```
# Separar los datos en entrenamiento (70%), validación (15%) y prueba (15%)
x_train, x_temp, y_train, y_temp, carro_train, carro_temp = train_test_split(x, y, carro, test_size=0.3, r
x_val, x_test, y_val, y_test, carro_val, carro_test = train_test_split(x_temp, y_temp, carro_temp, test_si

# Normalización de los datos
scaler_x = StandardScaler()
scaler_y = StandardScaler()

x_train_scaled = scaler_x.fit_transform(x_train.reshape(-1, 1)).flatten()
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1)).flatten()

x_val_scaled = scaler_x.transform(x_val.reshape(-1, 1)).flatten()
y_val_scaled = scaler_y.transform(y_val.reshape(-1, 1)).flatten()

x_test_scaled = scaler_x.transform(x_test.reshape(-1, 1)).flatten()
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1)).flatten()
```

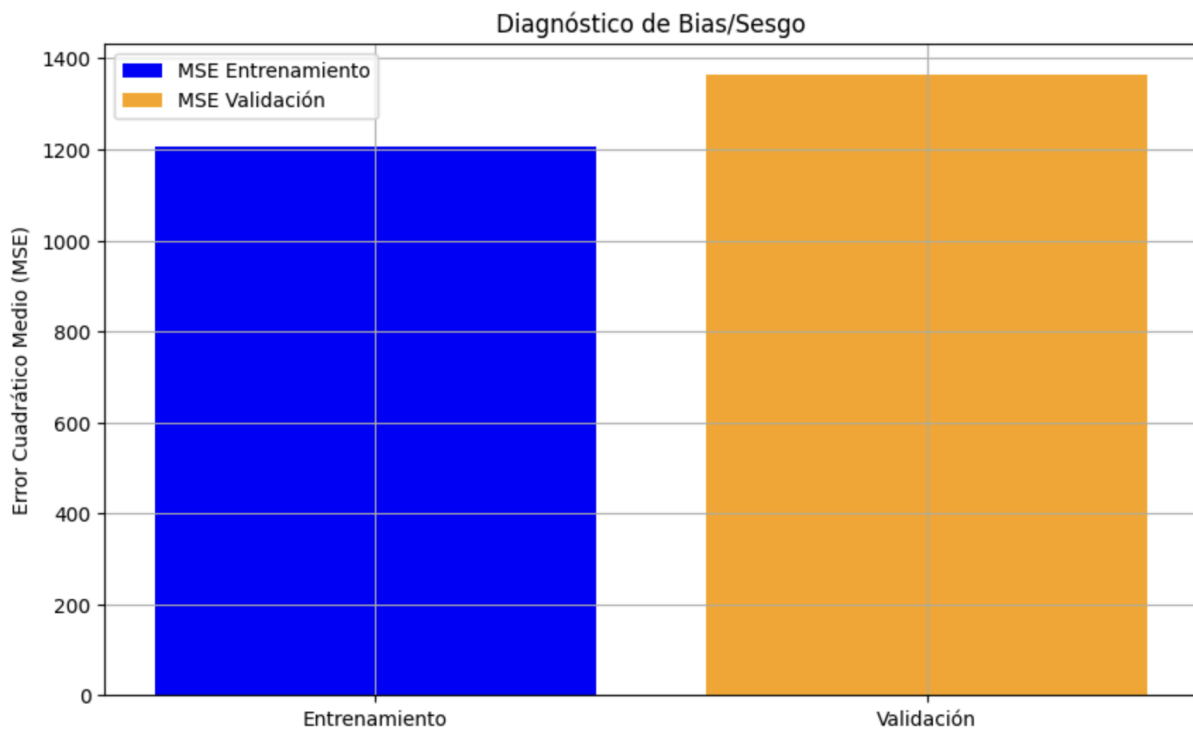
Se observa que los datos se dividen en entrenamiento, validación y prueba, en una proporción de 70-15-15.

Posterior a esto se tiene la evaluación del modelo:

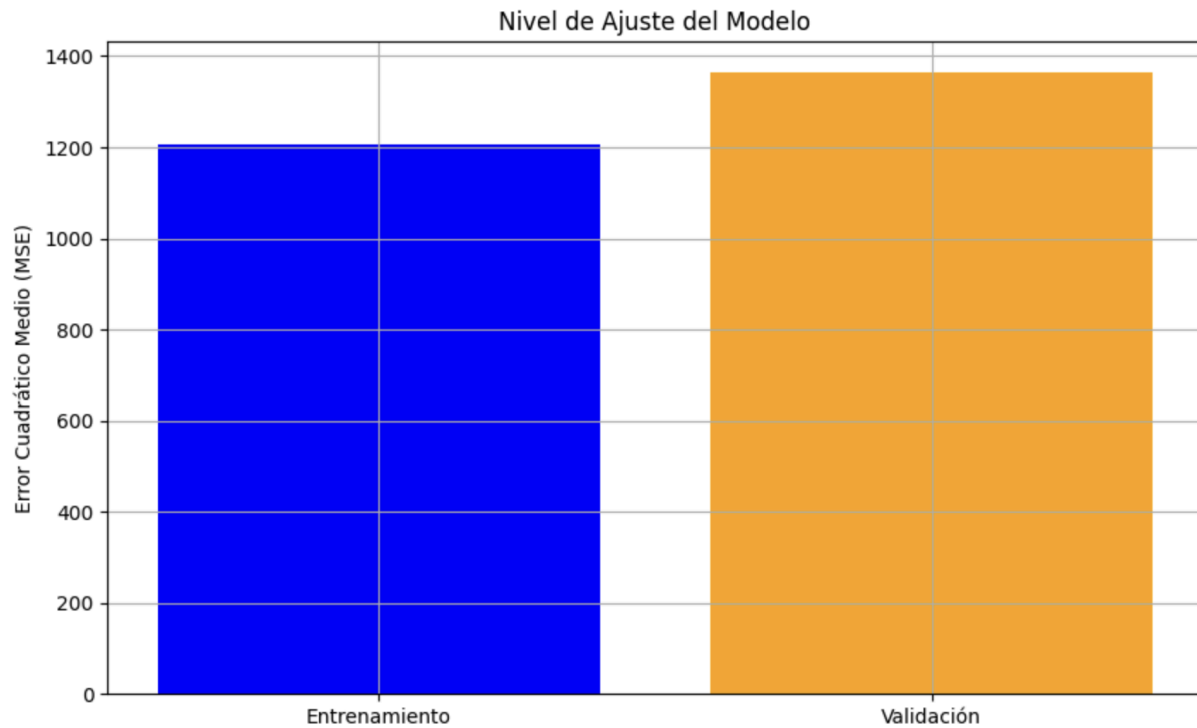
```
Con Regularización Ridge - Entrenamiento:  
MSE: 1204.9875346292004, R²: 0.0003500233999288449  
Con Regularización Ridge - Validación:  
MSE: 1363.1857420896924, R²: -0.01872769479654446
```

Se cambió un poco el enfoque del primer código, en este se le aplicó una regularización, utilizando Ridge se reduce la varianza de los datos y mejora el modelo.

Posteriormente se realizó un análisis del Bias/Sesgo del modelo



Aquí se puede observar que se obtiene un Bias bajo ya que la gráfica del conjunto de entrenamiento no supera al de validación. Por lo tanto el modelo si ajusta bien los datos.



Pasando al nivel de ajuste del modelo, no hay evidencia de un underfitting o de overfitting.

Para resumir, el conjunto de datos que se le asignó a este modelo no es lo suficientemente grande, aunque con los datos que cuenta si se puede hacer un análisis bastante certero de lo que está ocurriendo dentro de este, a diferencia del modelo anterior, si se nota una mejora significativa gracias a la regularización de los datos y escalamiento de estos, las siguientes acciones a tomar serían agregar más datos a la base de datos, observar el comportamiento del modelo y ajustar en caso de que exista algún caso de overfitting o de underfitting.