Adrian Pineda Sanchez

Fecha de entrega: 30/08/2024

Momento de Retroalimentación: Módulo 2 Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución. (Portafolio Implementación)

Requerimientos

- Define una semilla que corresponda con los últimos cuatro dígitos de tu matrícula
- Carga el set de datos de Valhalla y divide el set de datos en entrenamiento (40%), validación (40%), y prueba (20%), utilizando el método train_test_split. y la semilla definida arriba
- Entrena un modelo base de tipo SGDRegressor que utilice una tasa de aprendizaje de 1E-4, un máximo de iteraciones de un millón, y que utilice la semilla definida arriba
- Calcula el error cuadrático medio para este modelo, sobre los datos de entrenamiento, validación, y prueba. Estos datos servirán como línea base.
- Realiza una gráfica donde muestres cada subconjunto de datos (entrenamiento, validación, prueba) y el modelo de regresión obtenido (como una recta)
- Crea una lista que contenga 20 elementos (enteros) entre 2 y 39 (sin repetición, y que incluyan el número 2). Estos valores representarán la cantidad de instancias que se usarán para el análisis
- Para cada uno de los tamaños del punto anterior, entrena 100 modelos usando un subconjunto aleatorio del set de entrenamiento que contenga esa cantidad de muestras. Por ejemplo, para el tamaño de 2 muestras, se deben entrenar 100 modelos utilizando 2 muestras seleccionadas aleatoriamente de las 40 muestras disponibles en el set de entrenamiento
- Para cada uno de los modelos del punto anterior, calcula el error cuadrático medio en el subconjunto de entrenamiento (el que tiene un número cambiante de muestras), y en el subconjunto de validación.
- Calcula el promedio de las 100 repeticiones para cada uno de los modelos y sus errores. Esto debería generar dos listas de 20 valores cada uno, donde cada elemento representa el error promedio de las 100 repeticiones que se hicieron para cada subconjunto de entrenamiento.
- Agrega a las listas anteriores los errores de entrenamiento y validación de la línea base.
- Haz una gráfica donde se muestre la evolución del error promedio de entrenamiento y validación, para cada uno de los diferentes tamaños de entrenamiento
- Con base en la grafica anterior, explica el tipo de ajuste obtenido para el primer modelo (el entrenado sobre 2 muestras) y para el modelo final (el entrenado sobre 40 muestras). También explica como cambia el tipo de ajuste a medida que se incrementa el número de muestras del entrenamiento. Incluye también en tu análisis el grado de sesgo y de varianza para los diferentes modelos.
- Con base en la gráfica y los datos, identifica la cantidad de muestras más adecuada para realizar el entrenamiento. Justifica tu selección.
- Entrena un nuevo modelo utilizando esa cantidad de muestras, y calcula su error cuadrático medio sobre el subconjunto de entrenamiento (el de la cantidad de muestras seleccionadas), el de validación, y el de prueba.
- Para facilitar la revisión, entrega dos archivos. El primero debe ser un Jupyter Notebook con todo el desarrollo (código comentado). El segundo debe ser un PDF del Jupyter Notebook. Revisa las instrucciones del primer entregable para ver cómo exportar el archivo HTML y posteriormente pasarlo a PDF.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

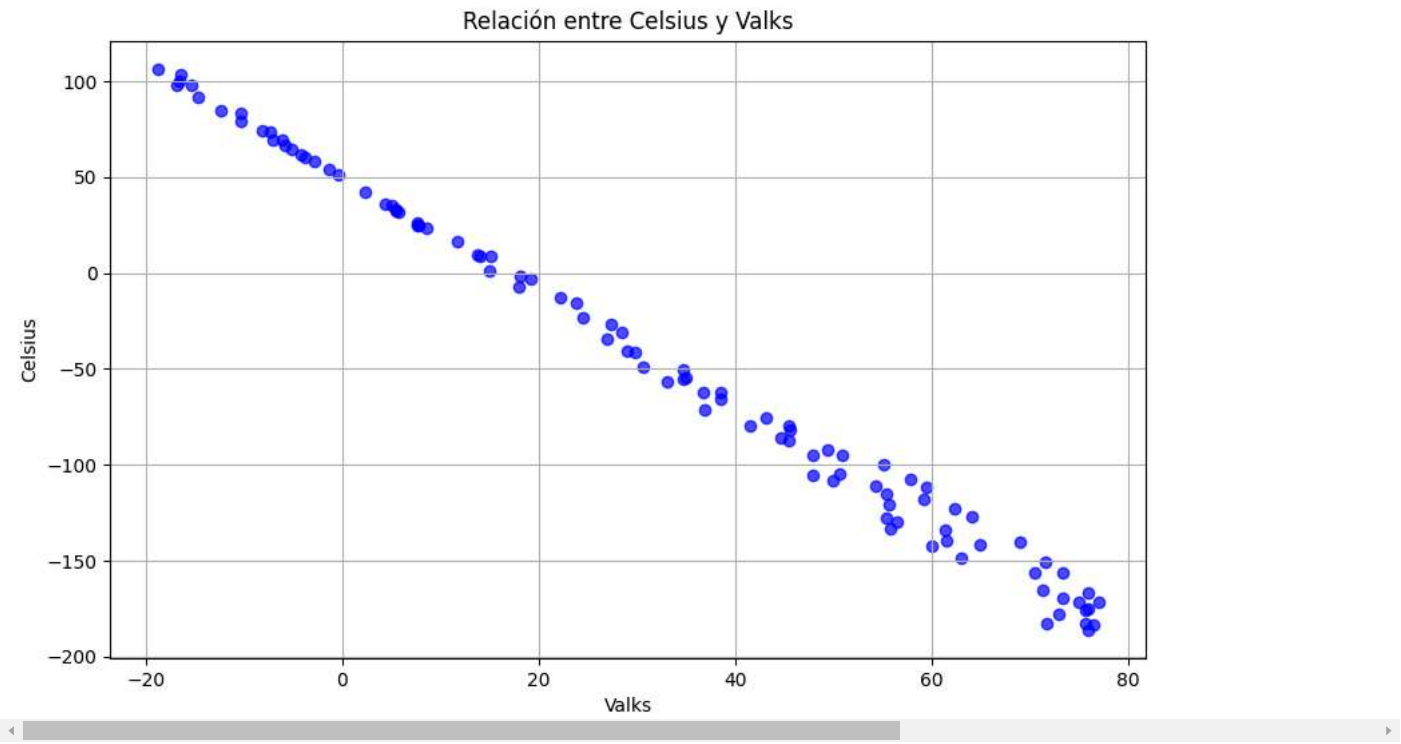
1. Importacion Dataframe y transformacion de datos

```
df = pd.read_csv("https://raw.githubusercontent.com/AdrianPinedaSanchez/RetoIAAvanzada/Adrian/AdrianPineda_Portafolio%20An%C3%A1lisis/M%C3%B
df
```

	Celsius	Valks
0	61.4720	-139.7400
1	70.5790	-156.6000
2	-7.3013	73.2690
3	71.3380	-165.4200
4	43.2360	-75.8350
...	...	...
95	-7.0094	69.6320
96	36.8820	-71.2400
97	26.9390	-34.2550
98	-18.8100	106.4300
99	13.7120	9.1011

100 rows × 2 columns

```
# Convertir las columnas a arrays numpy
X = df[['Celsius']].to_numpy()
y = df[['Valks']].to_numpy()


# Crear el scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', alpha=0.7)

# Configuración del gráfico
plt.title('Relación entre Celsius y Valks')
plt.xlabel('Valks')
plt.ylabel('Celsius')
plt.grid(True)
plt.show()
```



2. Division datos en entrenamiento (40%), validación (40%), y prueba (20%), utilizando el método train_test_split.

Semilla con matricula A00834710

```
import numpy as np

seed = 4710
np.random.seed(seed)

# Escalar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir los datos en entrenamiento, validación y prueba
X_train_val, X_test, y_train_val, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=seed)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_val, y_train_val, test_size=0.5, random_state=seed)

# Verificar las proporciones
train_size = X_train.shape[0] / X.shape[0]
valid_size = X_valid.shape[0] / X.shape[0]
test_size = X_test.shape[0] / X.shape[0]

print(f'Proporción de entrenamiento: {train_size:.2f}')
print(f'Proporción de validación: {valid_size:.2f}')
print(f'Proporción de prueba: {test_size:.2f}')
```

```
Proporción de entrenamiento: 0.40
Proporción de validación: 0.40
Proporción de prueba: 0.20
```

Entrenamiento modelo base de tipo SGDRegressor

- tasa de aprendizaje de 1E-4
- máximo de iteraciones de un millón
- Utilizar la semilla

```
from sklearn.linear_model import SGDRegressor

# Entrenar el modelo con los datos escalados
model = SGDRegressor(learning_rate='constant', eta0=1e-4, max_iter=1000000, random_state=seed)
model.fit(X_train, y_train.ravel())
```

```
SGDRegressor(eta0=0.0001, learning_rate='constant', max_iter=1000000,
             random_state=4710)
```

Error Cuadratico MSE Y R^2

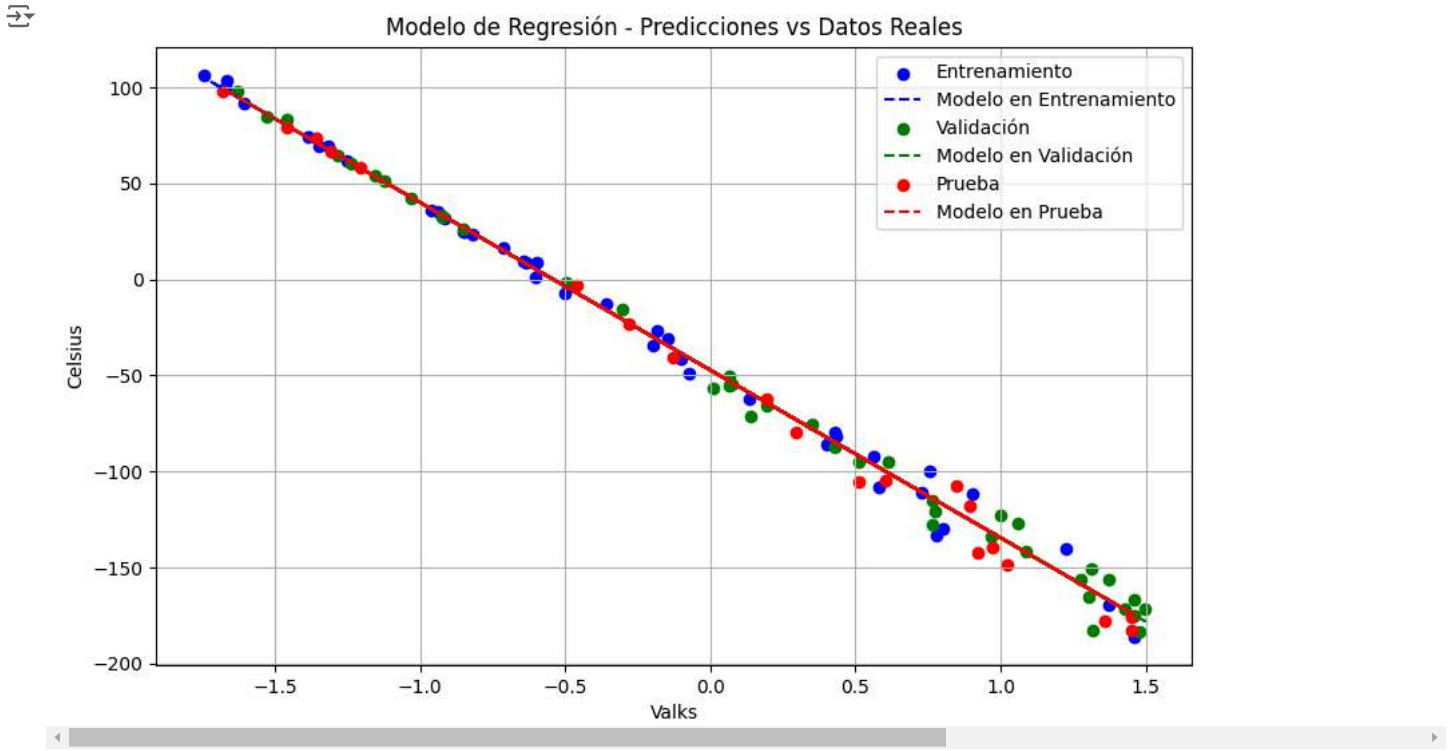
```
from sklearn.metrics import mean_squared_error

# Calcular el MSE y el R^2
mse_train = mean_squared_error(y_train, model.predict(X_train))
mse_valid = mean_squared_error(y_valid, model.predict(X_valid))
mse_test = mean_squared_error(y_test, model.predict(X_test))

r2_train = r2_score(y_train, model.predict(X_train))
r2_valid = r2_score(y_valid, model.predict(X_valid))
r2_test = r2_score(y_test, model.predict(X_test))

# Calcular el MSE y R^2 para todo el conjunto de datos combinado
y_pred_combined = np.concatenate([
    model.predict(X_train),
    model.predict(X_valid),
    model.predict(X_test)
])

y_combined = np.concatenate([y_train, y_valid, y_test])

mse_combined = mean_squared_error(y_combined, y_pred_combined)
r2_combined = r2_score(y_combined, y_pred_combined)

# Mostrar los resultados
print(f'MSE Entrenamiento: {mse_train:.4f}')
print(f'R^2 Entrenamiento: {r2_train:.4f}')
print(f'MSE Validación: {mse_valid:.4f}')
print(f'R^2 Validación: {r2_valid:.4f}')
print(f'MSE Prueba: {mse_test:.4f}')
print(f'R^2 Prueba: {r2_test:.4f}')
print(f'MSE Total: {mse_combined:.4f}')
print(f'R^2 Total: {r2_combined:.4f}')
```

```
MSE Entrenamiento: 40.5095
R^2 Entrenamiento: 0.9935
MSE Validación: 42.8875
R^2 Validación: 0.9945
MSE Prueba: 59.9740
R^2 Prueba: 0.9930
MSE Total: 45.3536
R^2 Total: 0.9942
```

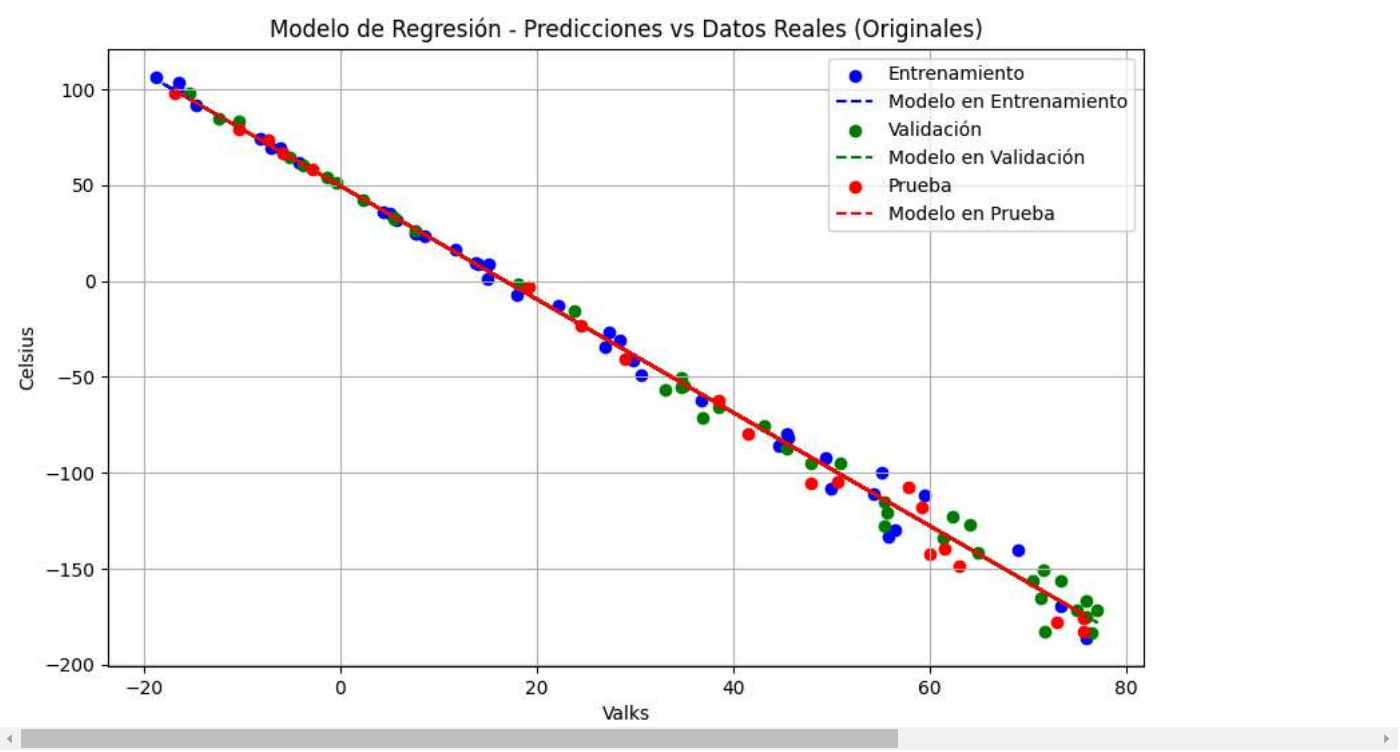
Gráfica subconjunto de datos (entrenamiento, validación, prueba) y el modelo de regresión obtenido (como una recta)

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Graficar los datos de entrenamiento
plt.scatter(X_train, y_train, color='blue', label='Entrenamiento')
plt.plot(X_train, model.predict(X_train), color='blue', linestyle='--', label='Modelo en Entrenamiento')

# Graficar los datos de validación
plt.scatter(X_valid, y_valid, color='green', label='Validación')
plt.plot(X_valid, model.predict(X_valid), color='green', linestyle='--', label='Modelo en Validación')

# Graficar los datos de prueba
```

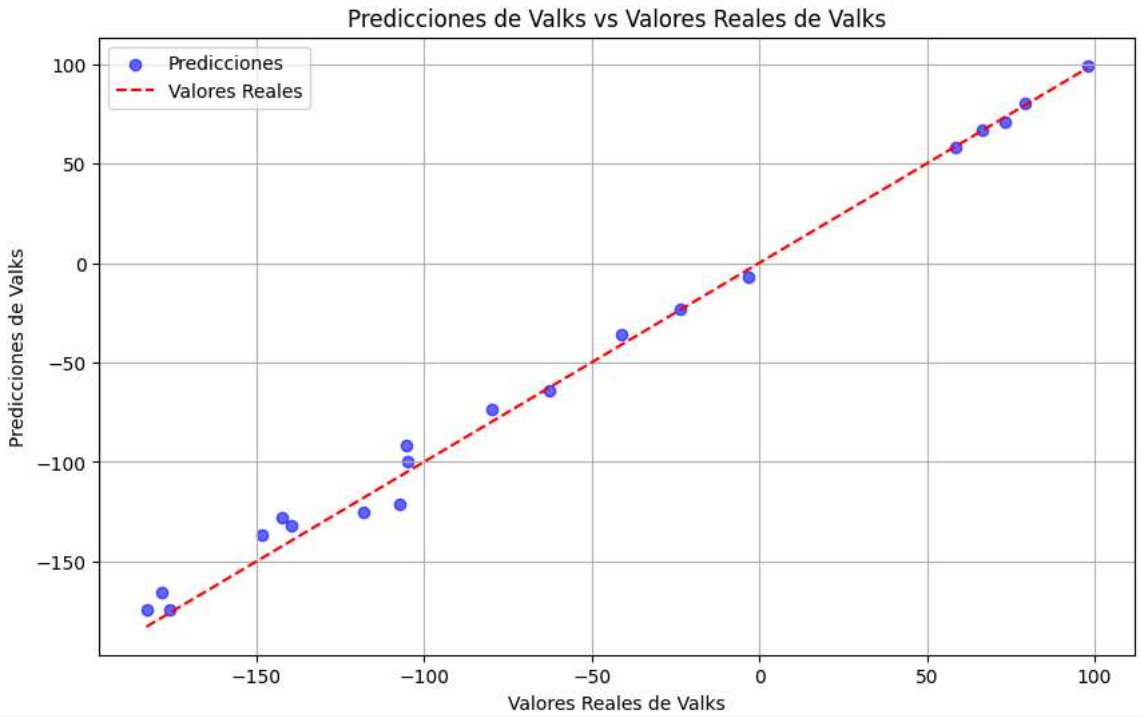
```
plt.scatter(X_test, y_test, color='red', label='Prueba')
plt.plot(X_test, model.predict(X_test), color='red', linestyle='--', label='Modelo en Prueba')

plt.xlabel('Valks')
plt.ylabel('Celsius')
plt.title('Modelo de Regresión - Predicciones vs Datos Reales')
plt.legend()
plt.grid(True)
plt.show()
```



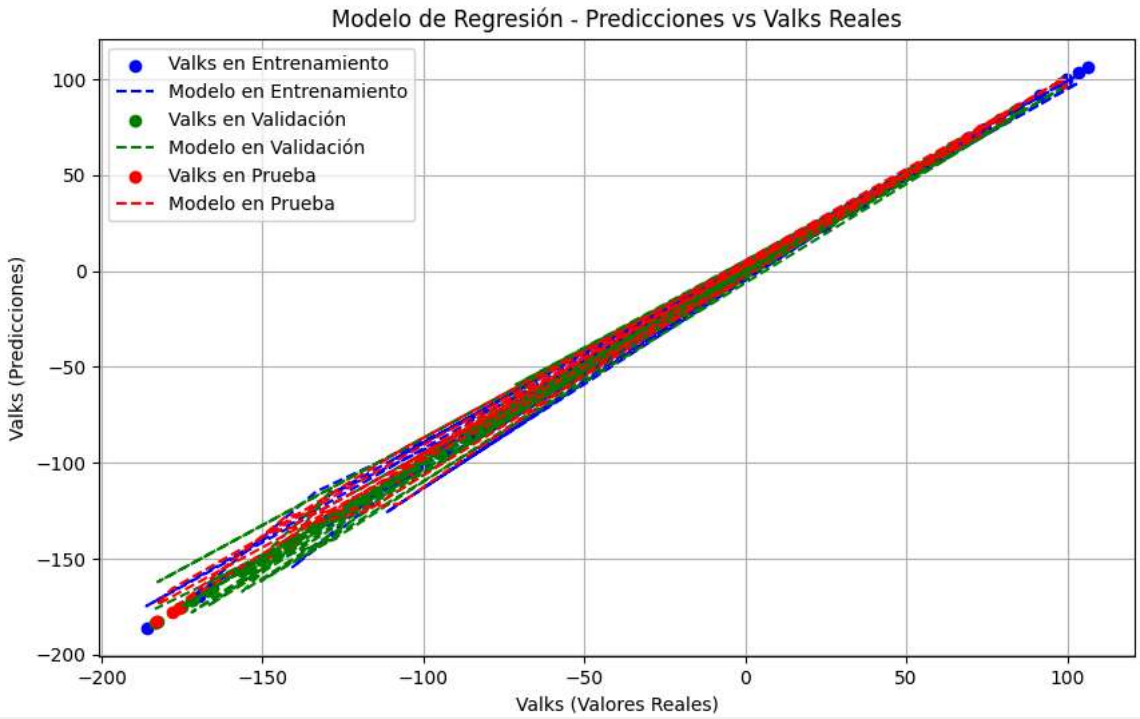

```
# Desescalar las predicciones y los valores reales
X_train_orig = scaler.inverse_transform(X_train)
X_valid_orig = scaler.inverse_transform(X_valid)
X_test_orig = scaler.inverse_transform(X_test)

y_train_pred_orig = model.predict(X_train)
y_valid_pred_orig = model.predict(X_valid)
y_test_pred_orig = model.predict(X_test)



plt.figure(figsize=(10, 6))

# Graficar los datos de entrenamiento
plt.scatter(X_train_orig, y_train, color='blue', label='Entrenamiento')
plt.plot(X_train_orig, y_train_pred_orig, color='blue', linestyle='--', label='Modelo en Entrenamiento')

# Graficar los datos de validación
plt.scatter(X_valid_orig, y_valid, color='green', label='Validación')
plt.plot(X_valid_orig, y_valid_pred_orig, color='green', linestyle='--', label='Modelo en Validación')

# Graficar los datos de prueba
plt.scatter(X_test_orig, y_test, color='red', label='Prueba')
plt.plot(X_test_orig, y_test_pred_orig, color='red', linestyle='--', label='Modelo en Prueba')

plt.xlabel('Valks')
plt.ylabel('Celsius')
plt.title('Modelo de Regresión - Predicciones vs Datos Reales (Originales)')
plt.legend()
plt.grid(True)
plt.show()
```




```
import matplotlib.pyplot as plt

# Obtener las predicciones del modelo en el conjunto de prueba
y_pred = model.predict(X_test)

# Crear el scatter plot para comparar predicciones vs valores reales
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6, label='Predicciones')

# Línea de referencia (45 grados) para indicar las predicciones perfectas
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', label='Valores Reales')

# Configuración del gráfico
plt.title('Predicciones de Valks vs Valores Reales de Valks')
plt.xlabel('Valores Reales de Valks')
plt.ylabel('Predicciones de Valks')
plt.legend()
plt.grid(True)
plt.show()
```




```
plt.figure(figsize=(10, 6))

# Graficar los valores reales y predicciones para Valks
plt.scatter(y_train, y_train, color='blue', label='Valks en Entrenamiento')
plt.plot(y_train, y_train_pred_orig, color='blue', linestyle='--', label='Modelo en Entrenamiento')

plt.scatter(y_valid, y_valid, color='green', label='Valks en Validación')
plt.plot(y_valid, y_valid_pred_orig, color='green', linestyle='--', label='Modelo en Validación')

plt.scatter(y_test, y_test, color='red', label='Valks en Prueba')
plt.plot(y_test, y_test_pred_orig, color='red', linestyle='--', label='Modelo en Prueba')

plt.xlabel('Valks (Valores Reales)')
plt.ylabel('Valks (Predicciones)')
plt.title('Modelo de Regresión - Predicciones vs Valks Reales')
plt.legend()
plt.grid(True)
plt.show()
```



3.Entrenamiento y evaluacion diferentes modelos

```
import numpy as np

# Crear una lista de 20 enteros entre 2 y 39 sin repetición, incluyendo el número 2
sample_sizes = np.random.choice(np.arange(3, 40), size=19, replace=False).tolist()
sample_sizes.append(2)  # Aseguramos que el número 2 esté incluido
sample_sizes.sort()  # Ordenar la lista para mejor visualización

print("Tamaños de muestra seleccionados:", sample_sizes)
```

```
Tamaños de muestra seleccionados: [2, 4, 5, 7, 11, 13, 14, 15, 17, 18, 19, 20, 23, 26, 27, 28, 34, 37, 38, 39]
```

Entrenamiento de 100 modelos de un subconjunto aleatorio de entrenamiento junto con su MSE en entrenamiento y validacion

```
from sklearn.metrics import mean_squared_error

# Listas para almacenar los errores promedio
errors_train = []
errors_val = []

# Iterar sobre cada tamaño de muestra
for size in sample_sizes:
    temp_train_errors = []
    temp_val_errors = []

    for _ in range(100):
        # Seleccionar aleatoriamente 'size' muestras del conjunto de entrenamiento
        X_sample, _, y_sample, _ = train_test_split(X_train, y_train, train_size=size/len(X_train), random_state=None)

        # Entrenar el modelo con el subconjunto seleccionado
        model = SGDRegressor(learning_rate='constant', eta0=1e-4, max_iter=1000000, random_state=seed)
        model.fit(X_sample, y_sample.ravel())

        # Calcular el error cuadrático medio en el subconjunto de entrenamiento y validación
        temp_train_errors.append(mean_squared_error(y_sample, model.predict(X_sample)))
        temp_val_errors.append(mean_squared_error(y_valid, model.predict(X_valid)))

    # Calcular el promedio de los errores para cada tamaño de muestra
```

```
    errors_train.append(np.mean(temp_train_errors))
    errors_val.append(np.mean(temp_val_errors))

print('Errores promedio de entrenamiento:', errors_train)
print('Errores promedio de validación:', errors_val)
```

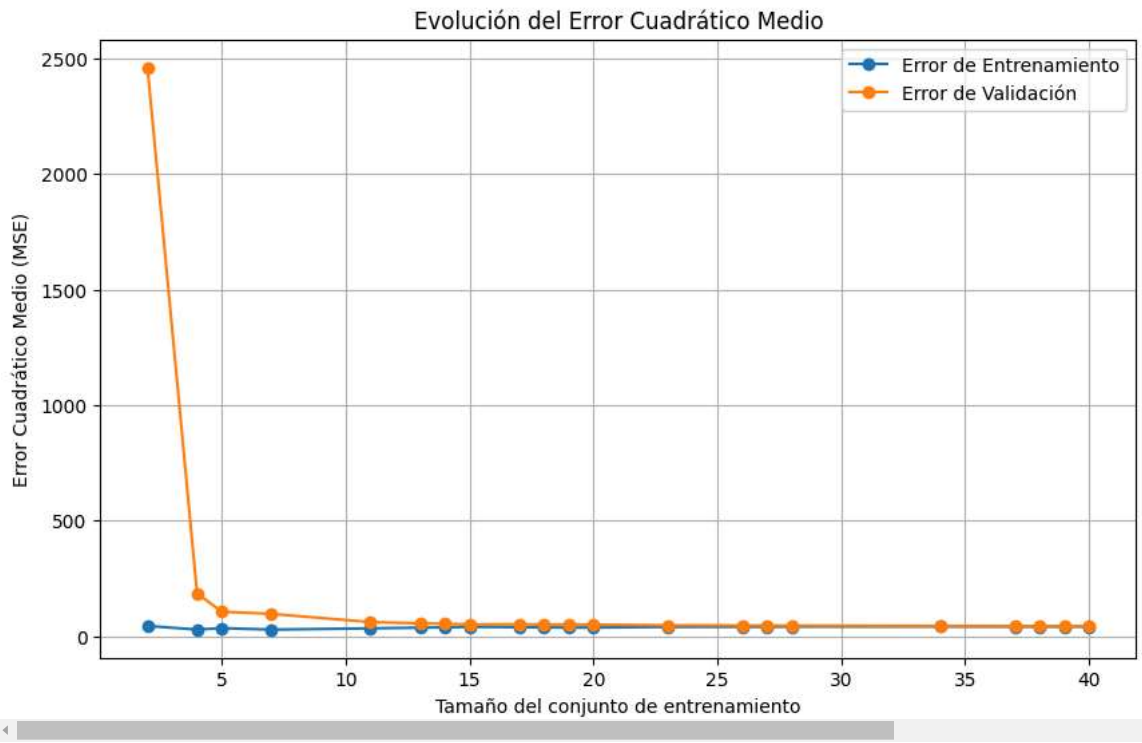
```
Errores promedio de entrenamiento: [43.66945870830213, 28.027262136155276, 34.00272595614245, 27.802362425261773, 33.208593834630875, 36
Errores promedio de validación: [2461.5294865133264, 184.3795193247983, 105.3337720285573, 95.91726998091025, 60.36051415901267, 54.9145
```

Agregar los errores a MSE de train y validacion

```
# Agregar los errores de la línea base a las listas
errors_train.append(mse_train)
errors_val.append(mse_valid)
```

Evolucion del MSE

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Graficar los errores de entrenamiento
plt.plot(sample_sizes + [len(X_train)], errors_train, marker='o', label='Error de Entrenamiento')

# Graficar los errores de validación
plt.plot(sample_sizes + [len(X_train)], errors_val, marker='o', label='Error de Validación')

plt.xlabel('Tamaño del conjunto de entrenamiento')
plt.ylabel('Error Cuadrático Medio (MSE)')
plt.title('Evolución del Error Cuadrático Medio')
plt.legend()
plt.grid(True)
plt.show()
```



Evolucion del tipo de ajuste a medida que se incrementa el número de muestras del entrenamiento. Incluye también en tu análisis el grado de sesgo y de varianza para los

diferentes modelos

1. Modelos con Pocas Muestras (2-5 muestras)

- **Tipo de Ajuste**:
  - Con un conjunto de entrenamiento tan pequeño, el modelo tiende a **sobreajustarse** (overfitting). Esto significa que se adapta en exceso a las pocas muestras disponibles, lo que resulta en un error muy bajo en el conjunto de entrenamiento, pero un error mucho más alto en el conjunto de validación. Este comportamiento sugiere que el modelo está memorizando los pocos datos disponibles en lugar de aprender patrones generales.
- **Sesgo**:
  - **Sesgo Alto**. El modelo no tiene suficiente información para aprender adecuadamente las relaciones subyacentes en los datos, por lo que no puede generalizar bien. El error en el conjunto de validación es alto, lo que indica que no está capturando correctamente las tendencias generales.
- **Varianza**:
  - **Varianza Alta**. Debido al tamaño extremadamente pequeño de la muestra, cualquier pequeño cambio en los datos de entrenamiento provoca grandes diferencias en el modelo. Esto significa que el modelo es muy sensible a las variaciones en los datos y no generaliza bien para nuevos datos.

2. Modelos con Tamaños Intermedios de Muestras (5-15 muestras)

- **Tipo de Ajuste**:
  - Al aumentar el tamaño de muestra a un rango intermedio, el modelo empieza a encontrar un **mejor equilibrio** entre sesgo y varianza. Ya no está sobreajustándose tanto como cuando había muy pocas muestras, pero aún no tiene suficientes datos para capturar todas las complejidades del problema. Sin embargo, a medida que se incrementa el tamaño de muestra, el error en el conjunto de validación disminuye considerablemente.
- **Sesgo**:
  - **Sesgo Moderado**. El modelo ahora tiene más datos para aprender y, por lo tanto, es capaz de captar las tendencias generales de manera más eficiente. El sesgo aún puede estar presente, pero se ha reducido considerablemente en comparación con los tamaños de muestra pequeños.
- **Varianza**:
  - **Varianza Moderada**. La varianza también ha disminuido porque el modelo es menos sensible a pequeños cambios en los datos de entrenamiento. A medida que se le dan más muestras, el modelo es menos propenso a ajustarse demasiado a casos específicos en los datos.

3. Modelos con Muchas Muestras (15-40 muestras)

- **Tipo de Ajuste**:
  - Con más de 15 muestras, el modelo alcanza un **ajuste equilibrado**. El error tanto en el conjunto de entrenamiento como en el de validación se estabiliza, lo que sugiere que el modelo está generalizando correctamente y ha aprendido bien las relaciones subyacentes en los datos. Aunque agregar más muestras puede continuar reduciendo el error, los beneficios marginales son menores.
- **Sesgo**:
  - **Sesgo Bajo**. Con suficientes datos, el modelo ha aprendido a captar correctamente las tendencias subyacentes, lo que minimiza el sesgo. El error de validación es bajo, lo que indica que el modelo está generalizando bien a datos nuevos.
- **Varianza**:
  - **Varianza Baja**. Al contar con una cantidad considerable de muestras, el modelo es estable y menos sensible a cambios en los datos de entrenamiento. La varianza es baja, lo que significa que el modelo no se ajusta en exceso a las peculiaridades de los datos de entrenamiento, y puede generalizar correctamente para otros datos.

Conclusion

Por los comportamientos mostrados conclui que el modelo que parece estar cercano a un equilibrio entre el equilibrio de ambos errores tanto el de entrenamiento asi como el de validacion parece ser el cercano a los 15 muestras

4. Nuevo Modelo utilizando las muestras

```
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

sample_sizes = [5, 10, 15]
models = []
mse_train = []
mse_valid = []
mse_test = []

for size in sample_sizes:
    # Seleccionar un subconjunto del tamaño de muestra actual
    X_sample, _, y_sample, _ = train_test_split(X_train, y_train, train_size=size/len(X_train), random_state=seed)

    # Entrenar el modelo
    model = SGDRegressor(learning_rate='constant', eta0=1e-4, max_iter=1000000, random_state=seed)
    model.fit(X_sample, y_sample.ravel())
    models.append(model)

    # Calcular los errores
    mse_train.append(mean_squared_error(y_sample, model.predict(X_sample)))
    mse_valid.append(mean_squared_error(y_valid, model.predict(X_valid)))
    mse_test.append(mean_squared_error(y_test, model.predict(X_test)))

    print(f'\nMSE Entrenamiento ({size} muestras): {mse_train[-1]:.4f}')
    print(f'MSE Validación ({size} muestras): {mse_valid[-1]:.4f}')
    print(f'MSE Prueba ({size} muestras): {mse_test[-1]:.4f}')
```

```
MSE Entrenamiento (5 muestras): 22.3382
MSE Validación (5 muestras): 44.1680
MSE Prueba (5 muestras): 69.3116

MSE Entrenamiento (10 muestras): 14.9061
MSE Validación (10 muestras): 43.5190
MSE Prueba (10 muestras): 64.2164

MSE Entrenamiento (15 muestras): 30.3757
MSE Validación (15 muestras): 46.3651
MSE Prueba (15 muestras): 49.6263
```

El mas balanceado parece ser en 15 muestras

```
# Seleccionar un subconjunto de 15 muestras del conjunto de entrenamiento
best_sample_size = 15
X_best_sample, _, y_best_sample, _ = train_test_split(X_train, y_train, train_size=best_sample_size/len(X_train), random_state=seed)

# Entrenar el modelo
best_model = SGDRegressor(learning_rate='constant', eta0=1e-4, max_iter=1000000, random_state=seed)
best_model.fit(X_best_sample, y_best_sample.ravel())

# Calcular los errores en entrenamiento, validación y prueba
mse_train_best = mean_squared_error(y_best_sample, best_model.predict(X_best_sample))
mse_valid_best = mean_squared_error(y_valid, best_model.predict(X_valid))
mse_test_best = mean_squared_error(y_test, best_model.predict(X_test))


mse_train_base = 40.5095
mse_valid_base = 42.8875
mse_test_base = 59.9740

# Imprimir los valores
print(f'MSE Entrenamiento Línea Base: {mse_train_base:.4f}')
print(f'MSE Validación Línea Base: {mse_valid_base:.4f}')
print(f'MSE Prueba Línea Base: {mse_test_base:.4f}')
print("\n")

print(f'MSE Entrenamiento (15 muestras): {mse_train_best:.4f}')
print(f'MSE Validación (15 muestras): {mse_valid_best:.4f}')
print(f'MSE Prueba (15 muestras): {mse_test_best:.4f}')
```

```
MSE Entrenamiento Línea Base: 40.5095
MSE Validación Línea Base: 42.8875
MSE Prueba Línea Base: 59.9740


MSE Entrenamiento (15 muestras): 30.3757
MSE Validación (15 muestras): 46.3651
MSE Prueba (15 muestras): 49.6263
```

```
# Comparar con los valores de la línea base
print(f'R^2 Entrenamiento Línea Base: {r2_train:.4f}')
print(f'R^2 Validación Línea Base: {r2_valid:.4f}')
print(f'R^2 Prueba Línea Base: {r2_test:.4f}')
print("\n")

# Calcular el R^2 para el modelo con 15 muestras
r2_train_best = r2_score(y_best_sample, best_model.predict(X_best_sample))
r2_valid_best = r2_score(y_valid, best_model.predict(X_valid))
r2_test_best = r2_score(y_test, best_model.predict(X_test))

# Imprimir los R^2 del nuevo modelo
print(f'R^2 Entrenamiento (15 muestras): {r2_train_best:.4f}')
print(f'R^2 Validación (15 muestras): {r2_valid_best:.4f}')
print(f'R^2 Prueba (15 muestras): {r2_test_best:.4f}')
```

```
R^2 Entrenamiento Línea Base: 0.9935
R^2 Validación Línea Base: 0.9945
R^2 Prueba Línea Base: 0.9930


R^2 Entrenamiento (15 muestras): 0.9942
R^2 Validación (15 muestras): 0.9941
R^2 Prueba (15 muestras): 0.9942
```

Conclusiones del mejor Modelo

El R^2 asi como el MSE en entrenamiento, validacion (aunque bajo un poco en este), y prueba la disminucion del del modelo entrenado con 15 muestras es ligeramente superior al del modelo base, lo cual indica que este modelo con 15 muestras logra capturar la variabilidad de los datos de entrenamiento un poco mejor que el modelo entrenado con todo el conjunto de datos.

Mejor Configuración: En general, el modelo con 15 muestras parece funcionar mejor, especialmente en el conjunto de prueba, donde tiene un menor MSE y un mayor R^2 en comparación con la línea base. Esto sugiere que el modelo con 15 muestras no solo se ajusta bien a los datos de entrenamiento, sino que también generaliza mejor a datos no vistos.

Argumento: Aunque el MSE de validación del modelo con 15 muestras es ligeramente superior, los beneficios en términos de generalización (MSE más bajo y R^2 más alto en el conjunto de prueba) hacen que esta configuración sea preferible. El modelo con 15 muestras es más eficiente en términos de entrenamiento, y ofrece una generalización superior, lo que lo convierte en una opción robusta para este problema.

Aunque el modelo con 15 muestras parece ofrecer un mejor rendimiento en algunos aspectos, la mejora no es tan significativa como para ser determinante. Ambos modelos tienen un desempeño muy similar, y la elección entre ellos podría depender más de factores como la eficiencia en el tiempo de entrenamiento o la simplicidad del modelo

```
!jupyter nbconvert --to html /content/drive/MyDrive/Colab_Notebooks/Valhalla_AdrianPineda.ipynb
```