Reconocimiento de Patrones (ML) - T01

ALEJANDRO ZARATE MACIAS

02 de Febrero 2026

Abstract

En este trabajo se abordan distintos problemas de regresión lineal y polinomial aplicados al reconocimiento de patrones. Se utilizan tanto conjuntos de datos reales como funciones analíticas para analizar el comportamiento de los modelos de regresión bajo diferentes configuraciones. Se estudia el efecto del grado del polinomio, el número de iteraciones y la inclusión de regularización L1 y L2. Finalmente, se emplean curvas de aprendizaje para evaluar el desempeño de los modelos y analizar fenómenos de underfitting y overfitting.

1 Problema 1

1.1 Enunciado

Considere el conjunto de datos de [1]. Elabore un script en Python para resolver el problema de regresión asociado con la predicción del consumo de combustible en ciudad y en carretera utilizando sklearn. Puede usar dos modelos lineales separados, uno para cada categoría de consumo de combustible. Escriba todas las suposiciones y las operaciones de preprocesamiento de datos que realice.

1.2 Metodología

Se utilizó el conjunto de datos correspondiente a los consumos de combustible de vehículos modelo 2022, el cual contiene 946 observaciones y 15 variables. No se encontraron valores faltantes, por lo que no fue necesario realizar un tratamiento de eliminación de registros con valores nulos.

La carga del conjunto de datos se realizó directamente desde Kaggle utilizando la librería `kagglehub`, lo que garantiza reproducibilidad del experimento:

```
def get_dataset() -> pd.DataFrame:
    file_path = "MY2022 Fuel Consumption Ratings.csv"
    df = kagglehub.dataset_load(
        KaggleDatasetAdapter.PANDAS,
        "rinichristy/2022-fuel-consumption-ratings",
        file_path
    )
```

```
    return df
```

Antes de definir el modelo, se analizó la cardinalidad de las variables categóricas con el objetivo de evitar un incremento excesivo en la dimensionalidad. En particular, se descartaron las variables *Make* y *Model* debido a su alto número de valores distintos.

```
for column in df.select_dtypes(include=['object']).columns:
    print(column, df[column].nunique())
```

Resultados de valores distintos por columna:

- Make: 39
- Model: 715
- Vehicle Class: 14
- Transmission: 23
- Fuel Type: 4

Con base en este análisis, se seleccionaron las siguientes variables explicativas:

- Variables numéricas: *Engine Size(L)*, *Cylinders*
- Variables categóricas: *Vehicle Class*, *Transmission*, *Fuel Type*

Las variables categóricas se transformaron mediante codificación *one-hot* utilizando `get_dummies` de la libreria pandas, eliminando una categoría por variable para evitar colinealidad.

```
X = df[['Engine Size(L)', 'Cylinders',
        'Vehicle Class', 'Transmission', 'Fuel Type']]

X = pd.get_dummies(
    X,
    columns=['Vehicle Class', 'Transmission', 'Fuel Type'],
    drop_first=True
)
```

Se definieron dos variables objetivo: consumo de combustible en ciudad y consumo en carretera. Para cada una se entrenó un modelo de regresión lineal independiente utilizando una partición 80/20 para entrenamiento y prueba.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y_city, test_size=0.2, random_state=14
)

model = LinearRegression()
model.fit(X_train, y_train)
```

El desempeño de los modelos se evaluó utilizando el error cuadrático medio (MSE) y el coeficiente de determinación R^2.

1.3 Resultados

Los resultados obtenidos para ambos modelos muestran un desempeño sólido y consistente entre entrenamiento y prueba.

Para el consumo de combustible en ciudad, se obtuvieron los siguientes valores:

- **Entrenamiento:** MSE = 1.6035, $R^2 = 0.8697$
- **Prueba:** MSE = 1.3690, $R^2 = 0.8663$

Para el consumo de combustible en carretera, los resultados fueron:

- **Entrenamiento:** MSE = 1.0452, $R^2 = 0.8084$
- **Prueba:** MSE = 0.9527, $R^2 = 0.7752$

En ambos casos, el desempeño en prueba es comparable al de entrenamiento, lo que sugiere una buena capacidad de generalización del modelo.

1.4 Discusión

Los resultados indican que un modelo lineal, utilizando un conjunto reducido de variables cualitativas y cuantitativas relacionadas al tipo de vehículo, es capaz de explicar generalizar de buena manera el consumo que estos podrian tener tanto en ciudad, como en carretera.

El modelo presenta un mejor desempeño en la predicción del consumo en ciudad que en carretera, lo cual es razonable debido a que el consumo urbano está más influenciado por características como el tamaño del motor, el número de cilindros y el tipo de transmisión. La ausencia de un incremento significativo del error en prueba sugiere que el modelo no presenta sobreajuste.

Sin embargo, el modelo asume una relación estrictamente lineal entre las variables, por lo que no captura posibles efectos no lineales ni interacciones más complejas entre las características.

1.5 Conclusión

Se desarrollaron dos modelos de regresión lineal para predecir el consumo de combustible en ciudad y en carretera a partir de características básicas del vehículo. Los resultados muestran que este enfoque proporciona predicciones razonablemente precisas y con buena generalización.

En conclusión, la regresión lineal resulta una herramienta adecuada como primera aproximación al problema.

2 Problema 2

2.1 Enunciado

Resuelva el problema 1 utilizando las ecuaciones normales de regresión lineal. Compare las soluciones de ambos problemas y anote sus conclusiones.

2.2 Metodología

En este problema se resolvió el mismo caso planteado en el Problema 1, utilizando exactamente el mismo conjunto de datos, las mismas variables explicativas y las mismas particiones de entrenamiento y prueba. La diferencia principal es que el ajuste del modelo se realizó mediante las ecuaciones normales de regresión lineal, en lugar de utilizar la implementación directa de `sklearn`.

Se seleccionaron las mismas variables explicativas: tamaño del motor, número de cilindros, clase del vehículo, tipo de transmisión y tipo de combustible. Las variables categóricas fueron codificadas mediante *one-hot encoding*, eliminando una categoría por variable para evitar colinealidad.

```
X = df[['Engine Size(L)', 'Cylinders',
        'Vehicle Class', 'Transmission', 'Fuel Type']]

X = pd.get_dummies(
    X,
    columns=['Vehicle Class', 'Transmission', 'Fuel Type'],
    drop_first=True
)
```

Para poder aplicar la formulación matricial de las ecuaciones normales, se añadió explícitamente una columna de unos a la matriz de características con el fin de modelar el intercepto. Posteriormente, los parámetros se estimaron utilizando la pseudoinversa de Moore–Penrose, lo cual evita problemas numéricos cuando la matriz no es invertible.

```
X_train_b = np.c_[np.ones(X_train_city.shape[0]),
    X_train_city]
X_test_b  = np.c_[np.ones(X_test_city.shape[0]), X_test_city
    ]

theta_city = np.linalg.pinv(
    X_train_b.T @ X_train_b
) @ X_train_b.T @ y_train_city
```

Se entrenaron dos modelos independientes: uno para el consumo de combustible en ciudad y otro para el consumo en carretera. El desempeño se evaluó mediante MSE y R^2 en entrenamiento y prueba.

2.3 Resultados

Los resultados obtenidos mediante ecuaciones normales son practicamente idénticos a los obtenidos en el Problema 1, tanto para consumo en ciudad como para consumo en carretera.

Para el consumo en ciudad:

- **Entrenamiento:** MSE $= 1.6035$, $R^2 = 0.8697$
- **Prueba:** MSE $= 1.3690$, $R^2 = 0.8663$

Para el consumo en carretera:

- **Entrenamiento:** MSE = 1.0452, $R^2 = 0.8084$
- **Prueba:** MSE = 0.9527, $R^2 = 0.7752$

La igualdad de las métricas confirma que ambos enfoques resuelven el mismo problema de mínimos cuadrados.

2.4 Discusión

Las ecuaciones normales y la implementación de `LinearRegression` de `sklearn` producen la misma solución, ya que ambos métodos estiman el modelo de mínimos cuadrados ordinarios. La principal diferencia entre ambos enfoques radica en la forma de resolver el sistema y no en el modelo obtenido.

Desde un punto de vista práctico, el uso explícito de las ecuaciones normales requiere mayor cuidado numérico, especialmente cuando se trabaja con un número elevado de variables o con variables altamente correlacionadas, como ocurre tras aplicar *one-hot encoding*. Por esta razón, se utilizó la pseudoinversa para garantizar estabilidad.

En contraste, `sklearn` abstrae estos detalles y emplea métodos numéricos robustos, lo que simplifica la implementación y reduce el riesgo de errores.

2.5 Conclusión

Resolver el problema mediante ecuaciones normales permite comprender la base matemática de la regresión lineal y verificar que la solución coincide con la obtenida usando `sklearn`. No obstante, para aplicaciones prácticas, el uso de implementaciones optimizadas resulta más conveniente y seguro.

En conclusión, las ecuaciones normales son útiles con fines didácticos y de validación, mientras que las librerías especializadas son preferibles para problemas reales con mayor complejidad.

3 Problema 3

3.1 Enunciado

Considere la siguiente función:

$$f(x) = 2^{\cos(x^2)}, \qquad x \in \mathcal{I} = [-\pi, \pi].$$

El objetivo es aproximar f mediante un modelo polinomial

$$h(x; \theta, n) = \sum_{j=0}^{n} \theta_j x^j, \qquad \theta = (\theta_0, \theta_1, \ldots, \theta_n)^T,$$

para un orden adecuado n. Elabore un script en Python para resolver el problema de regresión asociado. Observe que el conjunto de datos D consiste en un

muestreo de f en $\mathcal{I}$ de tamaño m. Escriba todas las suposiciones que realice. Además, escriba los hiperparámetros de optimización que elija y explique por qué los seleccionó de esa manera. Incluya una gráfica del error vs iteraciones y una gráfica de la solución. No olvide indicar qué valor de n elige y por qué.

3.2 Metodología

Se generó un conjunto de datos sintético muestreando la función $f(x) = 2^{\cos(x^2)}$ en el intervalo $[-\pi, \pi]$ con $m = 1000$ puntos equiespaciados. En este ejercicio se asumió que no existe ruido en las observaciones, por lo que los valores y corresponden directamente a la función.

```
m = 1000
x = np.linspace(-np.pi, np.pi, m)
y = 2 ** np.cos(x ** 2)
```

Para aproximar la función se utilizó un modelo polinomial de grado $n = 12$. La entrada escalar x se transformó a una representación polinomial hasta el grado seleccionado. El valor $n = 12$ se eligió como un compromiso entre flexibilidad y evitar un modelo excesivamente complejo.

```
n = 12

def create_poly_features(x, degree):
    return np.column_stack([x ** j for j in range(degree +
        1)])

X = create_poly_features(x, n)
```

Debido a que las potencias de x pueden tener escalas distintas (especialmente para grados altos), se normalizaron las columnas de la matriz de características mediante estandarización (media y desviación estándar). Esto se realizó con el objetivo de mejorar la estabilidad numérica y facilitar el entrenamiento con descenso por gradiente.

```
X_mean = X.mean(axis=0)
X_std = X.std(axis=0)
X_std[X_std == 0] = 1
X_norm = (X - X_mean) / X_std
```

Posteriormente, el conjunto de datos se dividió en entrenamiento y prueba (80/20). Esta partición se usó para medir desempeño fuera de muestra y comparar el error en ambos subconjuntos durante el entrenamiento.

```
X_train, X_test, y_train, y_test = train_test_split(
    X_norm, y, test_size=0.2, random_state=42
)
```

El ajuste de los parámetros del modelo se realizó mediante descenso por gradiente durante 1000 iteraciones. Se utilizó una tasa de aprendizaje $\eta = 0.1$,

elegida para lograr una disminución progresiva y rápida, aunque corriendo el riesgo de que el modelo no alcanzara su punto optimo al dar pasos no tan pequeños. En cada iteración se calculó la predicción, se actualizó el vector de parámetros y se registró el MSE tanto en entrenamiento como en prueba para construir la gráfica de error vs iteraciones.

```
learning_rate = 0.1
n_iterations = 1000
theta = np.zeros(n + 1)

errors_train, errors_test = [], []

for i in range(n_iterations):
    y_pred_train = X_train @ theta
    gradient = (1 / len(y_train)) * X_train.T @ (
        y_pred_train - y_train)
    theta = theta - learning_rate * gradient

    errors_train.append(mean_squared_error(y_train,
        y_pred_train))
    errors_test.append(mean_squared_error(y_test, X_test @
        theta))
```

Finalmente, se generaron dos visualizaciones: la curva de error (MSE) en función de las iteraciones para entrenamiento y prueba, y la comparación entre la función real y la aproximación polinomial obtenida en el intervalo $[-\pi, \pi]$.

3.3 Resultados

El entrenamiento del modelo polinomial se realizó con un grado $n = 12$, una tasa de aprendizaje de 0.1 y 1000 iteraciones. Durante el proceso se observó una disminución progresiva del error cuadrático medio tanto en el conjunto de entrenamiento como en el de prueba, lo cual indica que el algoritmo de descenso por gradiente converge de manera estable.

En la Figura 1 se muestran dos resultados clave:

- la evolución del error (MSE) en función de las iteraciones
- la comparación entre la función real y la aproximación polinomial obtenida

Aunque el error disminuye conforme avanzan las iteraciones, el ajuste final del modelo no logra reproducir adecuadamente la forma de la función original.

Las métricas finales obtenidas confirman este comportamiento, ya que los valores de R^2 resultaron negativos tanto en entrenamiento como en prueba, lo que indica un desempeño pobre del modelo.

3.4 Discusión

Aunque el descenso por gradiente converge, el modelo polinomial de grado 12 no es capaz de aproximar correctamente la función en el intervalo considerado. La

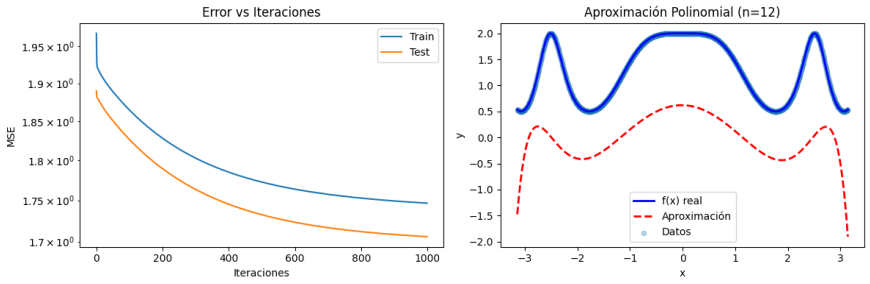

Figure 1: Izquierda: Error (MSE) en entrenamiento y prueba vs iteraciones. Derecha: comparación entre la función real $f(x)$ y la aproximación polinomial con $n = 12$.

gráfica de la solución muestra desviaciones importantes entre la aproximación y la función real, incluso produciendo valores fuera del rango esperado.

Este comportamiento sugiere que la representación polinomial en base monomial no es adecuada para este problema, aun cuando se aplicó normalización y se eligieron hiperparámetros razonables. El modelo converge, pero lo hace hacia una solución que no describe bien la función objetivo.

3.5 Conclusión

Se implementó una regresión polinomial entrenada mediante descenso por gradiente para aproximar la función $f(x) = 2^{\cos(x^2)}$. A pesar de que el error disminuyó durante el entrenamiento, la aproximación obtenida fue deficiente, como lo evidencian los valores negativos de R^2 y la comparación gráfica.

En conclusión, este experimento muestra que la convergencia del algoritmo de optimización no garantiza un buen ajuste del modelo. Para este tipo de funciones, es necesario considerar representaciones más adecuadas o métodos alternativos que mejoren la estabilidad y la calidad de la aproximación.

4 Problema 4

4.1 Enunciado

Resuelva el problema 3 utilizando el modelo de regresión polinomial de sklearn [2]. Compare las soluciones de ambos problemas y escriba sus conclusiones.

4.2 Metodología

Se reutilizó el mismo conjunto de datos del Problema 3 y el mismo grado polinomial ($n = 12$). La diferencia principal fue que aquí se usó la implementación de `sklearn` para generar automáticamente las características polinomiales y ajus-

tar el modelo lineal en un `Pipeline`. Esto evita la implementación manual del descenso por gradiente y delega la solución a un método numéricamente estable.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

n = 12
model = make_pipeline(
    PolynomialFeatures(degree=n),
    LinearRegression()
)
model.fit(x_train, y_train)
```

Finalmente, se generó la aproximación sobre una malla de valores para comparar visualmente la función real contra la predicción del modelo.

```
x_plot = np.linspace(-np.pi, np.pi, 200).reshape(-1, 1)
y_plot_pred = model.predict(x_plot)
```

4.3 Resultados

Los resultados obtenidos con la regresión polinomial de `sklearn` muestran un ajuste significativamente superior al observado en el Problema 3. Las métricas obtenidas fueron:

- **Entrenamiento:** MSE $= 0.006398$, $R^2 = 0.9808$
- **Prueba:** MSE $= 0.007259$, $R^2 = 0.9764$

En la Figura 2 se observa que la aproximación polinomial (línea roja punteada) sigue de manera muy cercana el comportamiento de la función real en todo el intervalo. A diferencia del Problema 3, no se presentan oscilaciones severas ni valores fuera del rango esperado.

4.4 Discusión

Los resultados evidencian que, utilizando el mismo grado polinomial, la implementación de `sklearn` produce un modelo con excelente capacidad de ajuste y generalización. A diferencia del enfoque manual con descenso por gradiente, el modelo converge a una solución adecuada y estable.

Esta mejora se debe principalmente a que `sklearn` emplea métodos numéricos robustos para resolver el problema de mínimos cuadrados, evitando los problemas de inestabilidad y mal condicionamiento observados en el Problema 3.

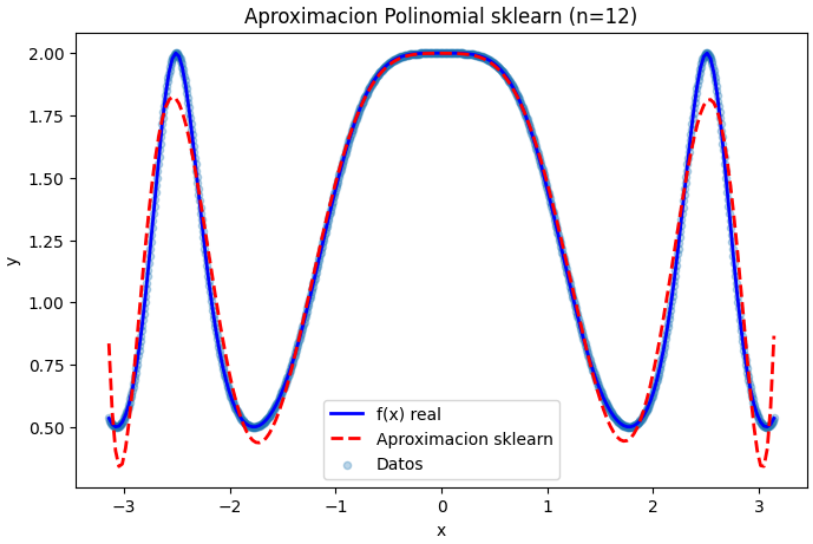

Figure 2: Comparación entre la función real $f(x)$ y la aproximación polinomial obtenida con `sklearn` para $n = 12$.

4.5 Conclusión

Al comparar ambos enfoques, se concluye que la regresión polinomial implementada con `sklearn` es claramente superior a la solución manual mediante descenso por gradiente para este problema. Aunque ambos utilizan el mismo grado polinomial, la diferencia en la metodología de optimización y estabilidad numérica impacta directamente en la calidad del ajuste.

5 Problema 5

5.1 Enunciado

Resuelve el problema 3 usando las ecuaciones normales de regresión polinómica. Compara las soluciones de ambos problemas y escribe tus conclusiones.

5.2 Metodología

Se utilizó el mismo planteamiento del Problema 3 en cuanto a la forma del modelo (polinomio de grado $n = 12$), pero en lugar de descenso por gradiente se estimaron los parámetros en una sola etapa usando ecuaciones normales. La diferencia principal respecto al Problema 4 es que aquí se construye explícitamente la matriz polinomial y se resuelve con pseudoinversa.

```
n = 12
```

```
X = np.column_stack([x ** j for j in range(n + 1)])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

theta = np.linalg.pinv(X_train.T @ X_train) @ X_train.T @
    y_train
```

Las predicciones se obtuvieron multiplicando directamente la matriz de características por el vector de parámetros.

```
y_test_pred = X_test @ theta
```

5.3 Resultados

El modelo entrenado mediante ecuaciones normales logró un ajuste adecuado en el conjunto de entrenamiento y un desempeño razonable en el conjunto de prueba. Las métricas obtenidas fueron:

- **Entrenamiento:** MSE $= 0.006276$, $R^2 = 0.9804$
- **Prueba:** MSE $= 0.027940$, $R^2 = 0.9223$

En la Figura 3 se observa que la aproximación polinomial sigue de manera cercana el comportamiento general de la función real. No obstante, se aprecian pequeñas desviaciones en las regiones cercanas a los extremos del intervalo.

5.4 Discusión

En comparación con el Problema 3, el uso de ecuaciones normales mejora de forma significativa el ajuste del modelo, eliminando los problemas de convergencia observados con descenso por gradiente. El modelo logra capturar correctamente la forma global de la función.

Sin embargo, al compararlo con el Problema 4, se observa que el desempeño en prueba es inferior, lo cual sugiere una mayor sensibilidad a la cantidad de datos y a la estabilidad numérica del método.

5.5 Conclusión

La regresión polinomial resuelta mediante ecuaciones normales permite obtener una buena aproximación de la función sin necesidad de un proceso iterativo de optimización. No obstante, su desempeño en generalización es inferior al obtenido con la implementación de `sklearn`.

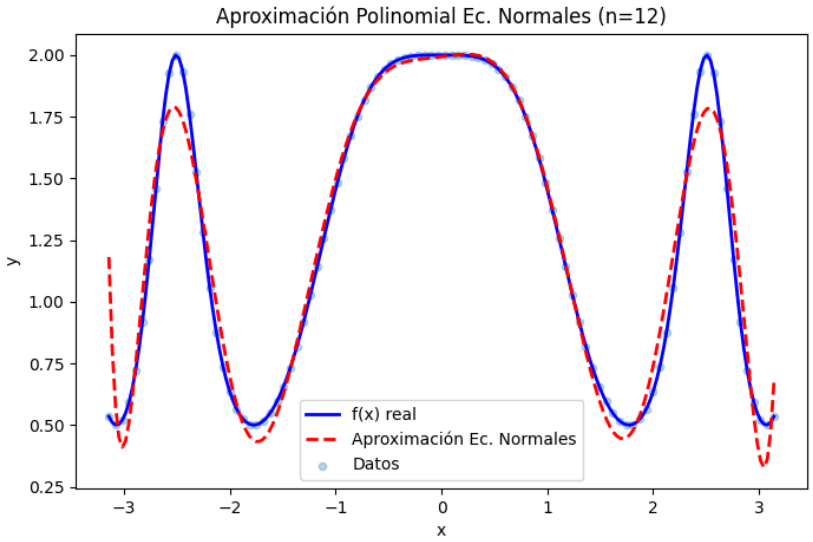

Figure 3: Comparación entre la función real $f(x)$ y la aproximación polinomial obtenida mediante ecuaciones normales para $n = 12$.

6 Problema 6

6.1 Enunciado

Considere el problema 3, pero esta vez utilizando un modelo lineal. Dibuje las curvas de aprendizaje asociadas. ¿Qué concluye?

6.2 Metodología

Se reutilizó el conjunto de datos del Problema 3, pero reemplazando el modelo polinomial por un modelo lineal (`LinearRegression`). La parte nueva de este problema fue el uso de `learning_curve` para graficar curvas de aprendizaje con validación cruzada ($cv = 5$), midiendo MSE.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import learning_curve

model_linear = LinearRegression()

train_sizes, train_scores, test_scores = learning_curve(
    model_linear, x, y, cv=5,
    train_sizes=np.linspace(0.1, 1.0, 10),
    scoring='neg_mean_squared_error'
)

train_mse = -train_scores.mean(axis=1)
```

```
test_mse  = -test_scores.mean(axis=1)
```

Adicionalmente, se entrenó un modelo con partición 80/20 para reportar métricas finales.

```
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42
)
model_linear.fit(x_train, y_train)
```

6.3 Resultados

Las métricas obtenidas para el modelo lineal fueron:

- **Entrenamiento:** MSE = 0.320026, $R^2 = 0.0002$
- **Prueba:** MSE = 0.374073, $R^2 = -0.0405$

En la Figura 4 se observan (izquierda) las curvas de aprendizaje y (derecha) la aproximación final del modelo lineal. La aproximación obtenida es prácticamente una recta casi constante y no captura la estructura no lineal de la función, lo cual coincide con los valores de R^2 cercanos a cero o negativos.

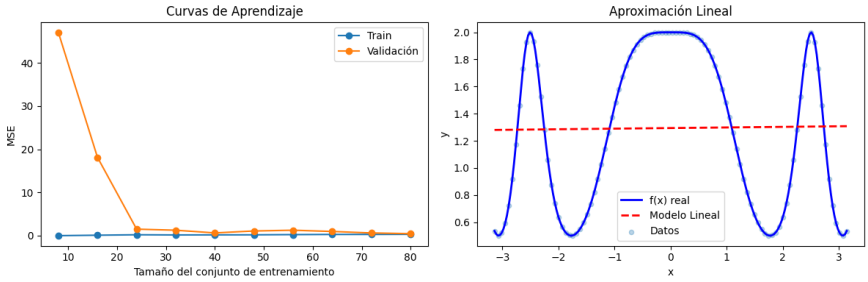

Figure 4: Izquierda: curvas de aprendizaje (MSE) del modelo lineal en entrenamiento y validación. Derecha: comparación entre $f(x)$ y la aproximación lineal.

6.4 Discusión

Las curvas de aprendizaje muestran que el error de entrenamiento se mantiene bajo y estable, mientras que el error de validación disminuye al aumentar el tamaño del conjunto de entrenamiento, pero se estanca en un valor relativamente alto. Este patrón indica underfitting: el modelo es demasiado simple para representar la relación entre x y y.

La gráfica de la aproximación confirma este comportamiento, el modelo lineal no puede reproducir las variaciones de la función y termina ajustando esencialmente un valor promedio. Por esta razón, agregar más datos no produce una

mejora sustancial, ya que la limitación principal es la capacidad del modelo y no la cantidad de información disponible.

6.5 Conclusión

El modelo lineal no es adecuado para aproximar $f(x) = 2^{\cos(x^2)}$ en $[-\pi, \pi]$, ya que presenta underfitting evidente. Las curvas de aprendizaje sugieren que el error de validación se estabiliza incluso al aumentar la cantidad de datos, por lo que la mejora requiere aumentar la capacidad del modelo, en lugar de únicamente incrementar el tamaño del conjunto de entrenamiento.

7 Problema 7

7.1 Enunciado

Considere la siguiente función:

$$f(x) = 2x^2 - 5, \qquad x \in \mathcal{I} = [-\pi, \pi].$$

Use regresión polinomial con $n = 20$. Grafique las curvas de aprendizaje asociadas. ¿Qué concluye?

7.2 Metodología

Se generó un conjunto de datos sintético para la función $f(x) = 2x^2 - 5$ con $m = 100$ puntos en $[-\pi, \pi]$. Para aproximar la función se usó regresión polinomial con $n = 20$. Como el grado es alto, se estandarizaron las características para estabilizar el entrenamiento.

La implementación utiliza `PolynomialFeatures` para expandir la variable x y `SGDRegressor` para entrenar iterativamente con un número fijo de iteraciones (1000), lo que permite observar el efecto del entrenamiento.

```
from sklearn.preprocessing import PolynomialFeatures,
    StandardScaler
from sklearn.linear_model import SGDRegressor

n = 20
poly = PolynomialFeatures(degree=n)
X_poly = poly.fit_transform(x)

scaler = StandardScaler()
X_poly_scaled = scaler.fit_transform(X_poly)

model = SGDRegressor(max_iter=1000, tol=1e-6, eta0=0.01,
    random_state=14)
```

Se calcularon curvas de aprendizaje con `learning_curve` (nueva parte relevante en esta sección) usando validación cruzada ($cv = 5$) y MSE.

```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(
    model, X_poly_scaled, y, cv=5,
    train_sizes=np.linspace(0.1, 1.0, 10),
    scoring='neg_mean_squared_error'
)
```

7.3 Resultados

El modelo polinomial de grado $n = 20$ logró un ajuste excelente tanto en entrenamiento como en prueba. Las métricas obtenidas fueron:

- **Entrenamiento:** MSE $= 0.010867$, $R^2 = 0.9997$
- **Prueba:** MSE $= 0.012839$, $R^2 = 0.9997$

En la Figura 5 se muestran (izquierda) las curvas de aprendizaje y (derecha) la aproximación obtenida. Se observa que la predicción del modelo coincide prácticamente con la función real en todo el intervalo, lo cual es esperado debido a que la función objetivo es cuadrática y, por tanto, pertenece a la familia de funciones que un polinomio puede representar exactamente.

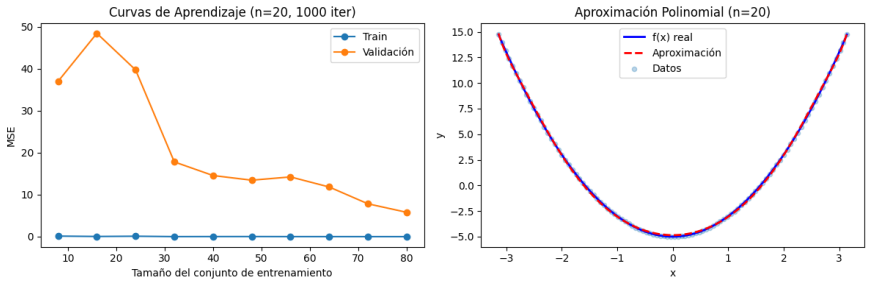

Figure 5: Izquierda: curvas de aprendizaje (MSE) para regresión polinomial con $n = 20$. Derecha: comparación entre $f(x) = 2x^2 - 5$ y la aproximación obtenida.

7.4 Discusión

Las curvas de aprendizaje muestran un error de entrenamiento muy bajo y un error de validación que disminuye conforme crece el tamaño del conjunto de entrenamiento. La separación entre ambas curvas es pequeña, lo cual indica buena generalización y ausencia de sobreajuste relevante, a pesar del alto grado polinomial ($n = 20$).

En este caso, el grado $n = 20$ es mayor al necesario, ya que un polinomio de grado 2 sería suficiente para modelar exactamente la función. Sin embargo, el buen desempeño se explica porque los datos siguen una relación polinomial simple y la estandarización ayuda a la estabilidad numérica durante el entrenamiento con sdg.

7.5 Conclusión

La regresión polinomial con $n = 20$ aproxima la función $f(x) = 2x^2 - 5$ con alta precisión, reflejada en MSE muy bajo y $R^2 \approx 1$ tanto en entrenamiento como en prueba. Las curvas de aprendizaje confirman que el modelo generaliza correctamente.

8 Problema 8

8.1 Enunciado

Considere el problema 7, pero esta vez use solo 5 iteraciones para el entrenamiento. Dibuje las curvas de aprendizaje asociadas. ¿Qué concluye?

8.2 Metodología

Se replicó exactamente el experimento del Problema 7 (misma función, mismas características polinomiales $n = 20$ y misma estandarización). La única modificación fue reducir el número de iteraciones de entrenamiento del optimizador a 5, con el fin de observar el efecto de un entrenamiento insuficiente.

```
model = SGDRegressor(max_iter=5, tol=1e-6, eta0=0.01,
    random_state=14)
```

8.3 Resultados

Al entrenar el modelo de regresión polinomial con $n = 20$ utilizando únicamente 5 iteraciones, el desempeño disminuye de manera considerable en comparación con el caso del Problema 7. Las métricas obtenidas fueron:

- **Entrenamiento:** MSE = 3.194353, $R^2 = 0.9097$
- **Prueba:** MSE = 2.698958, $R^2 = 0.9303$

En la Figura 6 se muestran las curvas de aprendizaje y la aproximación obtenida. Se observa que el error de validación permanece alto incluso al aumentar el tamaño del conjunto de entrenamiento. Asimismo, la aproximación polinomial no coincide completamente con la función real, especialmente en las regiones cercanas a los extremos del intervalo.

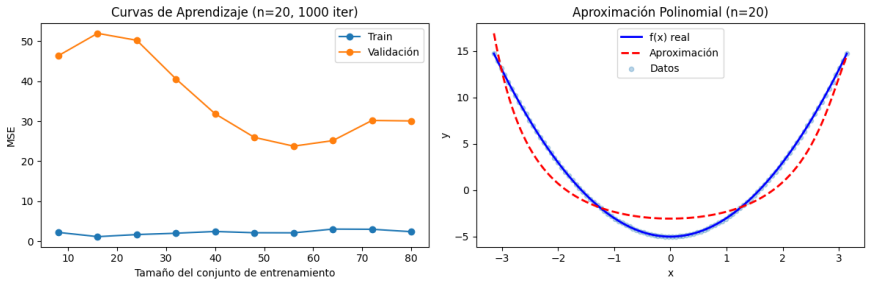

Figure 6: Izquierda: curvas de aprendizaje (MSE) para regresión polinomial con $n = 20$ entrenada con solo 5 iteraciones. Derecha: comparación entre $f(x) = 2x^2 - 5$ y la aproximación obtenida.

8.4 Discusión

Las curvas de aprendizaje indican que el modelo no logra reducir el error de validación de forma significativa, aun cuando se incrementa el número de muestras de entrenamiento. Esto sugiere que el problema no radica en la cantidad de datos, sino en un entrenamiento insuficiente.

La gráfica de la aproximación muestra que el modelo conserva la forma general de la función, pero con un desplazamiento y curvatura incorrectos. Este comportamiento es característico de un modelo con capacidad suficiente, pero cuyos parámetros no han convergido debido al número limitado de iteraciones.

8.5 Conclusión

Reducir el entrenamiento a solo 5 iteraciones impide que el modelo alcance una solución adecuada, aun cuando la función objetivo es simple y polinomial. En conclusión, este experimento demuestra que un número insuficiente de iteraciones provoca underfitting, y que permitir la convergencia del algoritmo de optimización es tan importante como la elección del modelo y sus características.

9 Problema 9

9.1 Enunciado

Considere el problema 7, pero esta vez, agregue la regularización L2 [3]. Anote sus resultados.

9.2 Metodología

Se reutilizó el pipeline del Problema 7 para construir características polinomiales ($n = 20$) y estandarizarlas. El cambio principal fue sustituir el modelo basado en SGD por regresión `Ridge`, incorporando regularización L2 con $\alpha = 1.0$.

```
from sklearn.linear_model import Ridge

model_ridge = Ridge(alpha=1.0)
model_ridge.fit(X_train, y_train)
```

También se calcularon curvas de aprendizaje usando el mismo procedimiento (validación cruzada $cv = 5$ y MSE).

```
train_sizes, train_scores, test_scores = learning_curve(
    model_ridge, X_poly_scaled, y, cv=5,
    train_sizes=np.linspace(0.1, 1.0, 10),
    scoring='neg_mean_squared_error'
)
```

9.3 Resultados

El modelo polinomial con regularización L2 presentó un desempeño muy alto tanto en entrenamiento como en prueba. Las métricas obtenidas fueron:

- **Entrenamiento:** MSE $= 0.036574$, $R^2 = 0.9990$
- **Prueba:** MSE $= 0.045137$, $R^2 = 0.9985$

En la Figura 7 se muestran las curvas de aprendizaje y la aproximación obtenida. Se observa que la curva de validación disminuye de forma estable conforme aumenta el tamaño del conjunto de entrenamiento, y que la aproximación sigue de manera muy cercana a la función real en todo el intervalo.

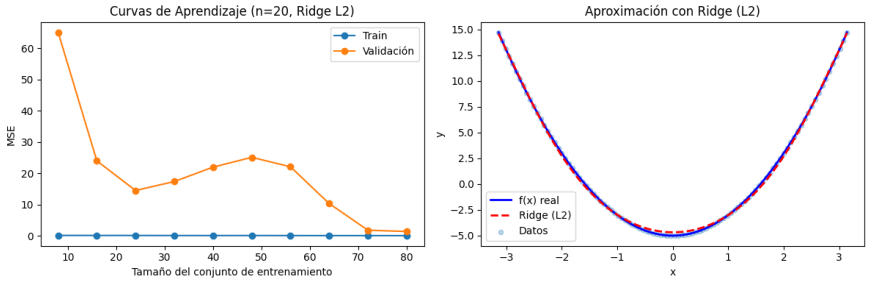

Figure 7: Izquierda: curvas de aprendizaje (MSE) para regresión polinomial con $n = 20$ y regularización Ridge (L2). Derecha: comparación entre $f(x) = 2x^2 - 5$ y la aproximación obtenida con regularización L2.

9.4 Discusión

La regularización L2 permite controlar la magnitud de los coeficientes del polinomio, lo que resulta en un modelo más estable y con mejor comportamiento en

generalización. En comparación con el Problema 7, el desempeño sigue siendo excelente, aunque con un ligero incremento en el error de entrenamiento, lo cual es esperado debido al sesgo introducido por la regularización.

A diferencia del Problema 8, donde el entrenamiento insuficiente limitaba el ajuste, aquí el modelo converge adecuadamente y mantiene una buena separación entre las curvas de entrenamiento y validación, evitando tanto el overfitting como inestabilidades numéricas.

9.5 Conclusión

La inclusión de regularización L2 en la regresión polinomial mejora la estabilidad del modelo y mantiene un excelente desempeño predictivo. Aunque introduce un pequeño aumento en el error de entrenamiento, el modelo generaliza de manera consistente y produce una aproximación suave y cercana a la función real.

10 Problema 10

10.1 Enunciado

Considere el problema 7, pero esta vez, añada la regularización L1 [4]. Anote sus resultados y compárelos con los del problema 9.

10.2 Metodología

Se mantuvo la misma configuración del Problema 9 (características polinomiales $n = 20$ y estandarización). El cambio principal fue sustituir `Ridge` (L2) por `Lasso` (L1), con $\alpha = 0.1$, para observar el efecto de una regularización que además tiende a hacer cero algunos coeficientes.

```
from sklearn.linear_model import Lasso

model_lasso = Lasso(alpha=0.1, max_iter=1000)
model_lasso.fit(X_train, y_train)
```

Las curvas de aprendizaje se calcularon igual que en el Problema 9 para mantener comparabilidad directa.

```
train_sizes, train_scores, test_scores = learning_curve(
    model_lasso, X_poly_scaled, y, cv=5,
    train_sizes=np.linspace(0.1, 1.0, 10),
    scoring='neg_mean_squared_error'
)
```

10.3 Resultados

El modelo con regularización L1 mostró un desempeño excelente tanto en entrenamiento como en prueba. Las métricas obtenidas fueron:

- **Entrenamiento:** MSE $= 0.009612$, $R^2 = 0.9997$
- **Prueba:** MSE $= 0.007749$, $R^2 = 0.9997$

En la Figura 8 se observan las curvas de aprendizaje y la aproximación obtenida. El error de validación disminuye rápidamente conforme aumenta el tamaño del conjunto de entrenamiento, y la aproximación coincide prácticamente con la función real en todo el intervalo.

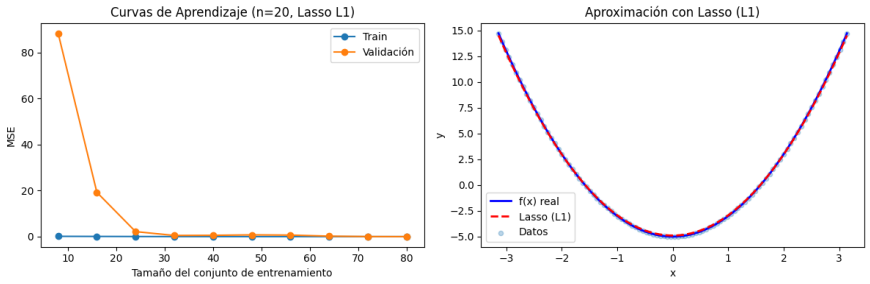

Figure 8: Izquierda: curvas de aprendizaje (MSE) para regresión polinomial con $n = 20$ y regularización Lasso (L1). Derecha: comparación entre $f(x) = 2x^2 - 5$ y la aproximación obtenida con Lasso.

10.4 Discusión

La regularización L1 permite obtener un modelo con excelente capacidad de generalización, manteniendo errores muy bajos en entrenamiento y prueba. En comparación con el Problema 9 (Ridge, L2), el desempeño es ligeramente mejor en términos de MSE, lo que sugiere que L1 es suficiente para controlar la complejidad del modelo en este caso.

10.5 Conclusión

La regresión polinomial con regularización L1 proporciona una aproximación muy precisa de la función $f(x) = 2x^2 - 5$, con excelente generalización y errores mínimos. En comparación con la regularización L2, L1 ofrece un ajuste ligeramente más eficiente al promover soluciones más simples.

References

[1] `https://www.kaggle.com/datasets/rinichristy/2022-fuel-consumption-ratings`.

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html`.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html.