Machine Learning y Data Analytics

HW 3 - Grupo 1

Ian Amighini
Julieta Brey
Lorenzo Nastri
Camila Sobrino
Matias Rodriguez Brun

Profesor Titular: Sergio Pernice

Consigna

a. Pídanle a su IA favorita que genere un programa Python que genere 3 vectores 3D: q, x1y x2 al azar (vectores cuyas coordenadas son números elegidos entre -1 y 1 al azar con una distribución constante).

   Pídanle que resuelva la ecuación que vimos en clase que encuentra el vector del span de x1 y x2 que llamamos v = lambda_1 x1 + lambda_2 x2 correspondiente al punto de span más cercano a q.

   Generen el vector r = q – v.

   Grafiquen en 3D los vectores x1, x2, q, v, y r. Además, grafiquen el span de x1 y x2. Comprueben gráficamente que r es ortogonal a dicho span.

b. Generen de algún problema que les interesen datos x e y, donde x es la "variable explicativa e y es la variable a explicar: y = a*1 + b*x + e, donde e es el vector error.

   a. Encuentren la regresión lineal en Excel.

   b. Resuelvan en Python las ecuaciones que vimos en clase que resuelve la regresión lineal y compare con la solución de Excel.

   c. Calculando los correspondientes productos escalares, verifique que el vector error (e = y – a*1 – b*x) es ortogonal a los vectores 1 y x.

a) Proyección de un Vector en un Subespacio de R3

Objetivo

Dado un vector q ∈ R3 y dos vectores x1,x2 ∈ R3, encontrar el punto v en el plano generado por x1 y x2 (es decir, el span de esos vectores) que se encuentra más cerca de q, y construir el vector ortogonal r=q−v.

Generación de Vectores

En este trabajo, se generaron tres vectores tridimensionales (x1, x2 y q) con coordenadas aleatorias en el intervalo [−1,1] usando np.random.uniform.

```
[1]  import numpy as np
     import plotly.graph_objects as go
```

```
[2]  # Generamos los vectores aleatorios en R^3 con coordenadas entre -1 y 1
     x1 = np.random.uniform(-1, 1, 3)
     x2 = np.random.uniform(-1, 1, 3)
     q  = np.random.uniform(-1, 1, 3)

     print("x1:", x1)
     print("x2:", x2)
     print("q: ", q)
```

```
x1: [0.07071169 0.30899651 0.22292364]
x2: [ 0.63012283 -0.75653408  0.38823219]
q:  [0.55088271 0.84782721 0.29522051]
```

Resolución del problema de proyección

Se usó la ecuación vista en clase para encontrar v, la proyección ortogonal de q sobre el subespacio generado por x1 y x2:

$$v = A(A^T A)^{-1} A^T q$$

- donde A=[x1 x2] es la matriz con x1 y x2 como columnas.

$$A = \begin{bmatrix} x_{1_1} & x_{2_1} \\ x_{1_2} & x_{2_2} \\ x_{1_3} & x_{2_3} \end{bmatrix}$$

- A^T A es una matriz cuadrada que se puede invertir si x1 y x2 son independientes.
- $(A^T A)^{-1} A^T$ Es la matriz de proyección ortogonal, que nos permite encontrar el punto más cercano a q en el subespacio.
- v es la proyección de q en el plano definido por x1 y x2.
- Esta matriz representa el subespacio bidimensional generado por x1 y x2.

Para verificar si los vectores forman un plano válido, se evalúa si son linealmente independientes. Si la matriz A^T A es singular (su determinante es 0), significa que los vectores están alineados y no generan un plano en 3D.

```
[ ]  # Creamos la matriz A con x1 y x2 como columnas
     A = np.column_stack((x1, x2))

     # Calculamos la proyección de q sobre el span de x1 y x2:
     # v = A (A^T A)^{-1} A^T q
     ATA = A.T @ A
     if np.linalg.det(ATA) != 0:
         lambdas = np.linalg.inv(ATA) @ (A.T @ q)
         v = A @ lambdas
     else:
         print("Los vectores x1 y x2 no son linealmente independientes.")
         v = np.array([0, 0, 0])

     print("v (proyección):", v)
```

```
v (proyección): [-0.22691927 -0.75027933  0.96786909]
```

Cálculo del Vector ortogonal r

Una vez obtenido v, se construyó r=q−v. Este vector es ortogonal al plano generado por x1 y x2, lo cual fue verificado gráficamente y numéricamente (producto escalar ≈ 0).

```python
# Calculamos el vector r = q - v
r = q - v
print("r = q - v:", r)
```

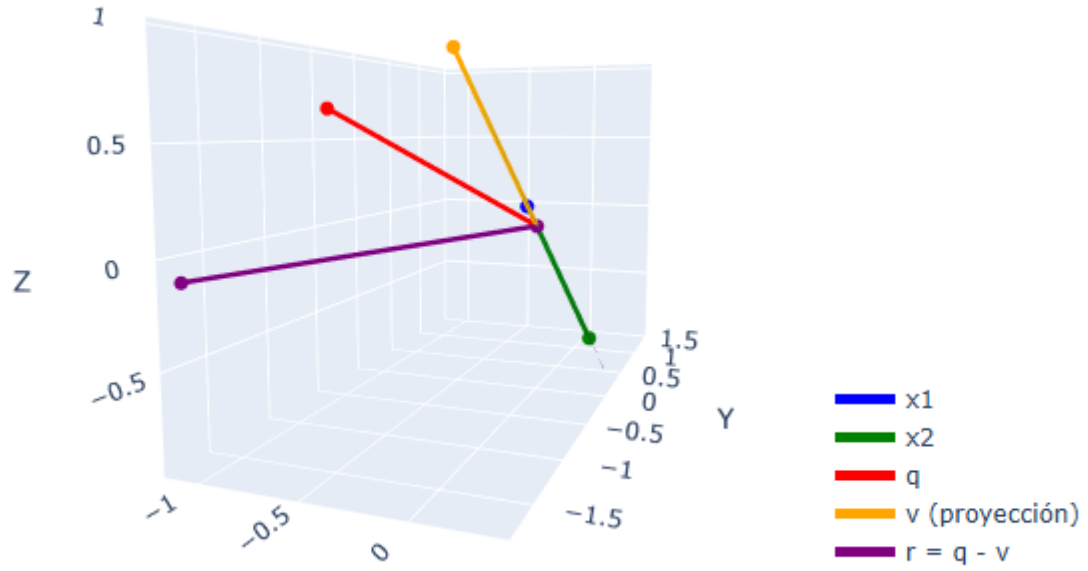
```
r = q - v: [-1.13935462 -1.71507289 -0.11253239]
```

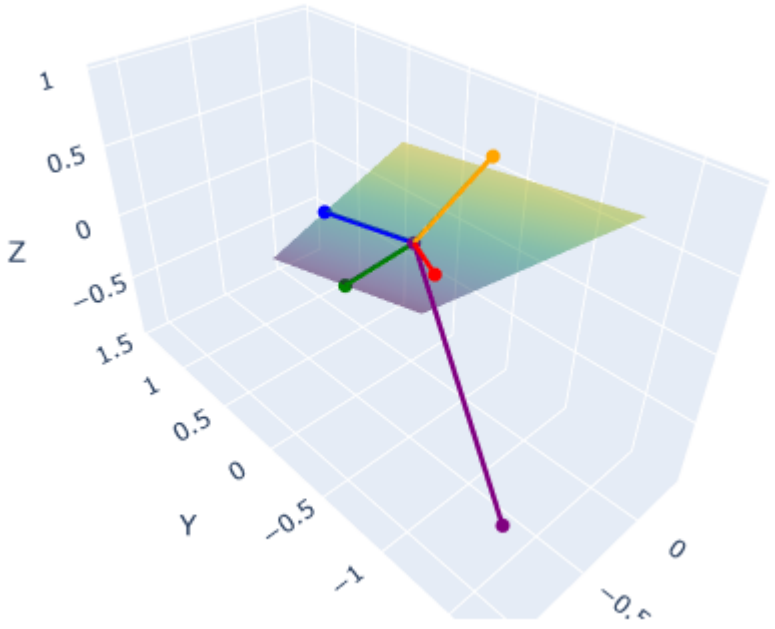
Visualización en 3D

Se utilizó plotly.graph_objects para graficar los vectores **x1,x2,q,v,r** y el plano generado por el span de x1 y x2, lo que permitió comprobar visualmente que r es perpendicular al plano.

```python
# Función para añadir un vector al gráfico
def add_vector(fig, vector, name, color):
    fig.add_trace(go.Scatter3d(
        x=[0, vector[0]],
        y=[0, vector[1]],
        z=[0, vector[2]],
        mode='lines+markers',
        marker=dict(size=4),
        line=dict(color=color, width=5),
        name=name
    ))

# Creamos la figura 3D
fig = go.Figure()

# Añadimos los vectores
add_vector(fig, x1, 'x1', 'blue')
add_vector(fig, x2, 'x2', 'green')
add_vector(fig, q, 'q', 'red')
add_vector(fig, v, 'v (proyección)', 'orange')
add_vector(fig, r, 'r = q - v', 'purple')
```

```python
# Creamos una malla para graficar el span de x1 y x2
# Usamos parámetros alpha y beta
alpha = np.linspace(-1, 1, 10)
beta  = np.linspace(-1, 1, 10)
alpha, beta = np.meshgrid(alpha, beta)

# Cada punto del plano es: P = alpha * x1 + beta * x2
X = alpha * x1[0] + beta * x2[0]
Y = alpha * x1[1] + beta * x2[1]
Z = alpha * x1[2] + beta * x2[2]

# Añadimos la superficie del span (con cierta transparencia)
fig.add_trace(go.Surface(x=X, y=Y, z=Z, opacity=0.5, colorscale='Viridis', showscale=False, name='Span(x1, x2)'))

# Configuración final del gráfico
fig.update_layout(
    title="Proyección de q sobre el span de x1 y x2 y vector ortogonal r",
    scene=dict(
        xaxis_title='X',
        yaxis_title='Y',
        zaxis_title='Z'
    )
)

fig.show()
```





b) Regresión Lineal y Verificación de Ortogonalidad

Objetivo: Ajustar un modelo de regresión lineal a datos sintéticos de la forma:

$$y = a + b \cdot x + e$$

y verificar que el vector de errores e=y−y^ es ortogonal tanto al vector constante como a la variable explicativa x, según lo demostrado en clase.

Generación de Datos Sintéticos

Se generan datos x e y con una relación lineal, pero con un pequeño error aleatorio:

$$y = a + b \cdot x + e$$

donde:

- a=3 es el intercepto.
- b=2 es la pendiente real.
- e~N(0,1)es un error aleatorio que simula ruido en los datos.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# 1. Generar datos sintéticos
np.random.seed(0)          # Para reproducibilidad
n = 50                     # Número de datos
x = np.linspace(0, 10, n)  # Valores de x en el intervalo [0, 10]

# Parámetros verdaderos
a_true = 3   # intercepto verdadero
b_true = 2   # pendiente verdadera

# Vector error: e ~ N(0,1)
error = np.random.normal(0, 1, n)

# Generamos la variable dependiente y
y = a_true + b_true * x + error
```

Cálculo de la Regresión Lineal usando la Ecuación Normal

La regresión se resuelve con la ecuación: $\hat{\beta} = (X^T X)^{-1} X^T y$

- donde X es la matriz de diseño con una columna de 1's y una columna con los valores de x.
- el vector β contiene los coeficientes óptimos.

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \quad \beta = \begin{bmatrix} a \\ b \end{bmatrix}$$

```python
# 2. Resolver la regresión lineal con las ecuaciones normales
# Construimos la matriz de diseño X (columna de 1's y columna de x)
X = np.column_stack((np.ones(n), x))

# Solución de las ecuaciones normales:
# beta_hat = (X^T X)^{-1} X^T y
beta_hat = np.linalg.inv(X.T @ X) @ (X.T @ y)
a_est, b_est = beta_hat  # Intercepto y pendiente estimados

print("Coeficientes estimados (Python):")
print("a (intercepto):", a_est)
print("b (pendiente):", b_est)
print("\nEstos resultados deberían coincidir con los obtenidos en Excel usando LINEST.")
```

```
Coeficientes estimados (Python):
a (intercepto): 3.8463705188082O5
b (pendiente): 1.8588377507009783

Estos resultados deberían coincidir con los obtenidos en Excel usando LINEST.
```

Cálculo del Vector de Error y Verificación de Ortogonalidad

Se estimó $\hat{y} = X\hat{\beta}$ y el error como: $e = y - \hat{y}$.

Para comprobar que el ajuste es correcto, se verifica que el error es ortogonal a las columnas de X

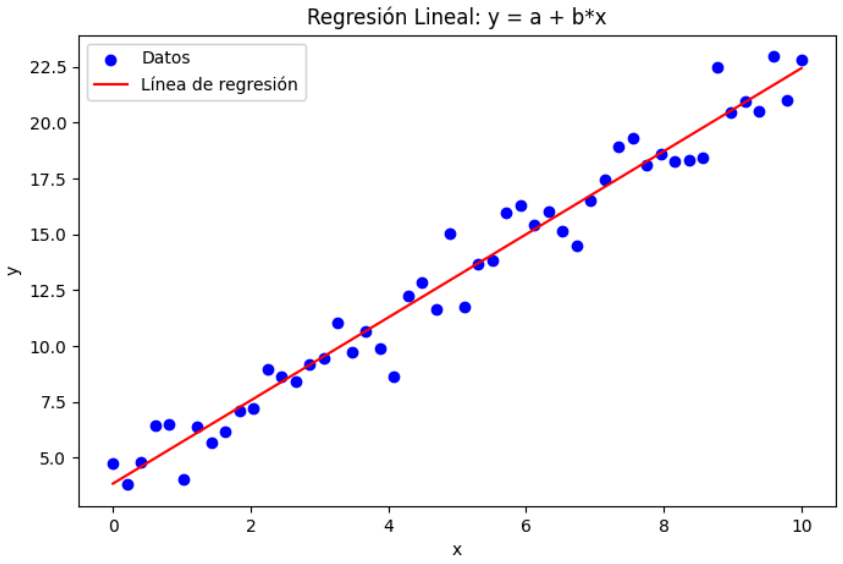
$$e \cdot X_1 = 0, \quad e \cdot X_2 = 0$$

```python
# 3. Calcular el vector de error y verificar ortogonalidad
# Cálculo de la recta de regresión estimada
y_est = X @ beta_hat

# Vector error: e = y - y_est
error_vector = y - y_est

# Verificar ortogonalidad:
# Producto escalar del error con el vector de unos (columna de 1's)
dot_with_ones = np.dot(error_vector, np.ones(n))
# Producto escalar del error con la variable x
dot_with_x = np.dot(error_vector, x)

print("\nVerificación de ortogonalidad del vector error:")
print("Producto escalar (error_vector · [1,1,...,1]):", dot_with_ones)
print("Producto escalar (error_vector · x):", dot_with_x)
print("\nLos productos escalares deben ser 0 (o muy cercanos a 0) para confirmar la ortogonalidad.")
```

```
Verificación de ortogonalidad del vector error:
Producto escalar (error_vector · [1,1,...,1]): 4.973799150320701e-14
Producto escalar (error_vector · x): 1.5543122344752192e-12

Los productos escalares deben ser 0 (o muy cercanos a 0) para confirmar la ortogonalidad.
```

Visualización de la Regresión

```python
# 4. (Opcional) Graficar datos y la línea de regresión
plt.figure(figsize=(8, 5))
plt.scatter(x, y, label='Datos', color='blue')
plt.plot(x, y_est, color='red', label='Línea de regresión')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Regresión Lineal: y = a + b*x')
plt.legend()
plt.show()
```



Plataforma de Python

hw3.ipynb