Universidad Rey Juan Carlos

Práctica 1. Regresión y Clasificación con Modelos Lineales y Logísticos (Python)

Gonzalo Vega Pérez

(g.vega.2020@alumnos.urjc.es)

Julia López Augusto

(j.lopeza.2020@alumnos.urjc.es)

09144517Q

03961534Z

Curso 2023/2024

ÍNDICE

Ejercicio 1. Regresión Lineal

En la siguiente tabla se muestran los datos de entrenamiento compuestos por dos características de entrada y una salida.

x_1	x_2	y
1	1	1.56
2	1	1.95
3	1	2.44
4	1	3.05
5	1	3.81
6	1	4.77
7	1	5.96
8	1	7.45
9	1	9.31
10	1	11.64

Tabla 1. Datos de entrenamiento para regresión lineal.

A continuación, resuelva los siguientes apartados:

1.1. Realice un script (o un Jupyter notebook) de Python en el que se obtengan los pesos ($\omega_0, \omega_1, \omega_2$) que forman la ecuación de la recta que se ajusta a la relación entre los datos de entrada y de salida. Para ello utilice funciones de alto nivel de la librería scikit-learn y adjunte en la memoria el valor de los pesos obtenidos.

Para resolver el ejercicio utilizaremos las funciones de la clase LinearRegression.

Primero, utilizando la biblioteca numpy creamos dos arrays que contengan los datos de entrada y salida. El array X contiene los datos de las dos características de entrada organizados en una matriz 10x2. Por otra parte, el array Y contiene los datos de salida.

A continuación, creamos un objeto de tipo LinearRegression y lo entrenamos utilizando la función fit(X,Y) junto con los dos arrays creados anteriormente.

Para obtener los pesos extraemos los atributos coef_ e intercept_ del modelo. Coef_ es un array de dos elementos que contiene ω_1 y ω_2. Mientras que intercept_ contiene el término independiente ω_0.

Obtenemos los siguientes resultados:

$$\omega_0 = -0.705$$

$$\omega_1 = 1.072$$

$$\omega_2 = 0$$

```python
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

if __name__ == '__main__':

    # Datos iniciales
    x1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    x2 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
    X = np.column_stack((x1, x2))
    Y = np.array([1.56, 1.95, 2.44, 3.05, 3.81, 4.77, 5.96, 7.45, 9.31, 11.64])

    # Crear modelo y entrenarlo
    mdl = LinearRegression().fit(X,Y)

    # Resultados (Pesos)
    coef = mdl.coef_
    intercept = mdl.intercept_

    print("Coef:" + str(coef))
    print("Intercept:" + str(intercept))
```

Fig. 1. Código completo del ejercicio 1.1.

```
Coef:[1.07260606 0.        ]
Intercept:-0.7053333333333338
```

Fig. 2. Resultados de ω_1, ω_2 y ω_0 respectivamente del ejercicio 1.1.

1.2. Represente en una figura en 3D el conjunto de datos de entrenamiento, así como la recta que mejor se ajusta a los datos de acuerdo con los valores de los pesos obtenidos en el apartado anterior. Se recomienda usar la librería Matplotlib.

Una vez entrenado el modelo utilizamos la función predict() para obtener los puntos que forman la recta que mejor se ajusta a los datos.

Haciendo uso de la biblioteca Matplotlib representamos los datos iniciales como puntos en el espacio 3D utilizando la función scatter3D(), y los puntos obtenidos por el modelo como una recta, utilizando la función plot3D().

```python
# Utilizar modelo entrenado para predecir resultados
Ye = mdl.predict(X)

# Representar graficamente
fig = plt.figure
ax = plt.axes(projection='3d')

# Mostrar datos iniciales
ax.scatter3D(x1, x2, Y, marker='x')
# Mostrar predicción
ax.plot3D(x1, x2, Ye, 'red')

# Nombrar Ejes
ax.set_xlabel('x1', labelpad=20)
ax.set_ylabel('x2', labelpad=20)
ax.set_zlabel('y', labelpad=20)

plt.show()
```

Fig. 3. Código completo del ejercicio 1.2.

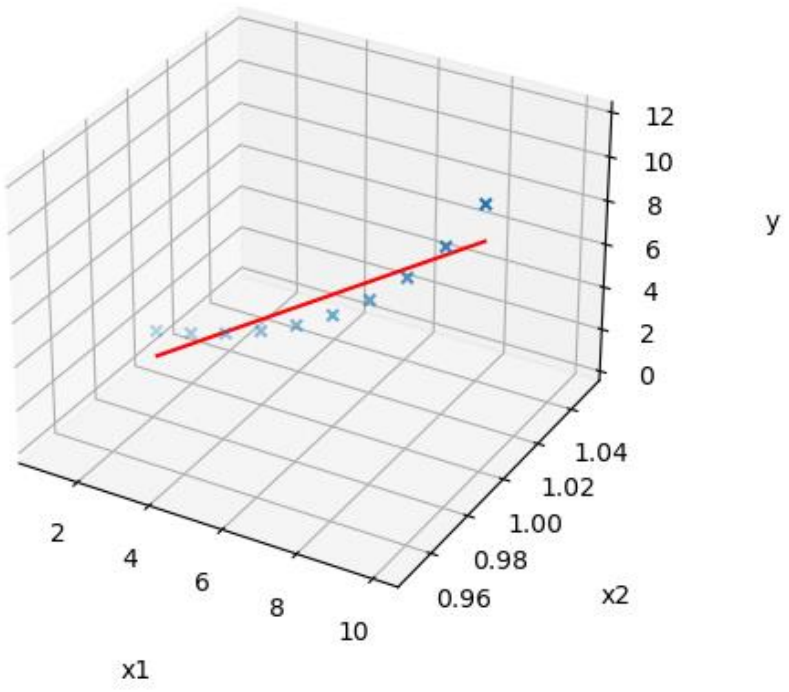

Fig. 4. Resultados visuales del ejercicio 1.2.

1.3. ¿Cree que una recta es la función que mejor se ajusta a la relación entre los datos de entrada y de salida? En caso contrario, indique cuál es la función que mejor se ajustaría.

Analizando la figura 3D del apartado anterior podemos observar que los datos iniciales se distribuyen siguiendo una curva. Es por ello que una función curva como las logarítmicas o las cuadráticas sería más apropiada que una recta.

Ejercicio 2. Función de coste

2.1. Realice un script (o un Jupyter notebook) de Python en el que se represente en 3D la función de coste a partir de logaritmo de la verosimilitud (asumiendo que el error sigue una distribución de tipo Gaussiana, de media cero y desviación típica la unidad) para ω_2 = 0, de tal forma que se aprecie el área donde dicha función presenta los valores máximos.

Para resolver el ejercicio se ha tenido que usar la fórmula definida en la diapositiva 20 del tema 2 de teoría; en concreto, sobre el logaritmo de la verosimilitud. Para ello, se ha utilizado los datos definidos en el ejercicio 1.

$$l(\omega) = n\log\frac{1}{\sqrt{2\pi\sigma}} - \frac{1}{\sigma^2}\cdot\frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - \sum_{j=0}^{d}\omega_j x_j^{(i)}\right)^2$$

En esta ocasión para poder graficar en 3D, en necesario que para poder hacerlo a ω0 y ω1 hay que pasarle como argumento 2 arrays (usando linspace) con muchos valores. Al ser ω0 = -0.705 y ω1 = 1.072 los arrays se comprenderás de 100 valores distribuidos uniformemente entre –10 y 10 para poder visualizar mejor el máximo.

Esos valores obtenidos de ω0 y ω1, es necesario usar np.meshgrid() que devuelve una lista de matrices de coordenadas de vectores.

Llegado a este punto, es necesario usar la fórmula definida anteriormente y se realiza con 2 bucles for que iteran sobre las variables definidas en la fórmula y así poder conseguir el logaritmo de la verosimilitud

Para poder graficar, todos los valores del logaritmo han sido almacenados en J_vals (array bidimensional con las dimensiones de np.meshgrid() de ω0.

Usamos figure(), proyectamos los ejes en 3d con plt.axes(), graficamos con ax.plot_surface de w0_mesh, w1_mesh y J_vals. Finalmente, al gráfico se le añade los nombres a cada eje xyz: w0, w1 y logW(w) respectivamente. El título del gráfico es: Función de coste y se muestra con plt.show()

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from math import log
from math import sqrt
from math import pi

# Datos
x1 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
x2 = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
X = np.column_stack((x1, x2))
Y = np.array([1.56, 1.95, 2.44, 3.05, 3.81, 4.77, 5.96, 7.45, 9.31, 11.64])
# w0 = -0.705, w1 = 1.072 y w2 = 0
#W = np.array([-0.705, 1.072])

# Para poder representarlo en 3D hay que dar un rango para expresar w0 y w1
w0_vals = np.linspace(-10, 10, 100)
w1_vals = np.linspace(-10, 10, 100)
w0_mesh, w1_mesh = np.meshgrid(w0_vals, w1_vals)

# Calcular la función de coste para cada combinación de w0 y w1
J_vals = np.zeros_like(w0_mesh)
for i in range(len(w0_vals)):
    for j in range(len(w1_vals)):
        J_vals[i, j] = len(Y) * log(1 / sqrt(2 * pi)) - (1 / (2 * len(Y))) * np.sum((Y - (w0_vals[i]* X[:, 0]+ w1_vals[i] *  X[:, 1]))**2)

# Graficar en 3D
fig = plt.figure()
ax = plt.axes(projection='3d')

ax.plot_surface(w0_mesh, w1_mesh, J_vals, cmap='viridis')
ax.set_xlabel('w0', labelpad=20)
ax.set_ylabel('w1', labelpad=20)
ax.set_zlabel('log V(w)', labelpad=20)
ax.set_title('Función de Coste')
plt.show()
```

Fig. 5. Código completo del ejercicio 2.

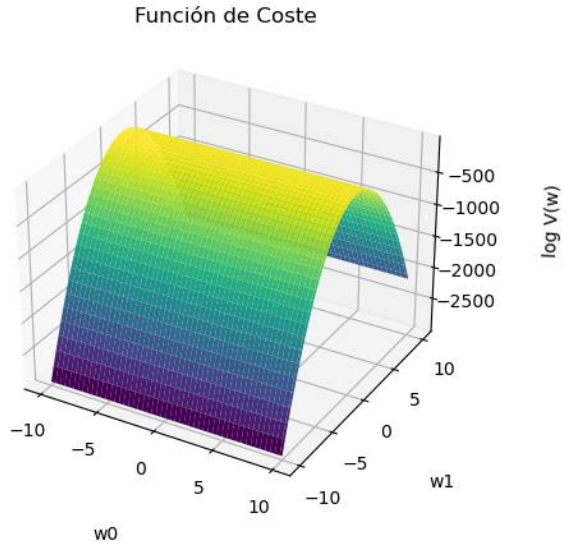

Fig. 6. Resultados visuales del ejercicio 2.

Ejercicio 3. Regresión Logística
Se desea construir un modelo basado en regresión logística binaria que sea capaz de clasificar los datos de la tabla adjunta.

x_1	x_2	x_3	y
0.89	0.41	0.69	+
0.41	0.39	0.82	+
0.04	0.61	0.83	0
0.75	0.17	0.29	+
0.15	0.19	0.31	0
0.14	0.09	0.52	+
0.61	0.32	0.33	+
0.25	0.77	0.83	+
0.32	0.23	0.81	+
0.40	0.74	0.56	+
1.26	1.53	1.21	0
1.68	1.05	1.22	0
1.23	1.76	1.33	0
1.46	1.60	1.10	0
1.38	1.86	1.75	+
1.54	1.99	1.75	0
1.99	1.93	1.54	+
1.76	1.41	1.34	0
1.98	1.00	1.83	0
1.23	1.54	1.55	0

Tabla 2. Datos de entrenamiento para regresión logística.

3.1. Realice un script (o un Jupyter notebook) de Python que construya el modelo de clasificador basado en regresión logística, utilizando funciones de alto nivel de scikit-learn. Adjunte en la memoria los pesos del modelo generado.

Para resolver este ejercicio seguimos los mismos pasos que en el ejercicio 1.1.

Primero creamos un array que contenga los datos de entrada, 'X', que será una matriz 20x3. Y un array que contenga los datos de salida, 'Y'. Las cruces tendrán asignado el valor 1 y los círculos en valor 0.

Después creamos y entrenamos el modelo utilizando las funciones LogisticRegression() y fit().

Una vez obtenido el modelo podemos observar el valor de los atributos coef_ e intercept_ del objeto para obtener los pesos.

Obtenemos los siguientes resultados:

$$\omega_0 = 1.192$$

$$\omega_1 = -0.275$$

$$\omega_2 = -0.734$$

$$\omega_3 = -0.904$$

```python
import numpy as np
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

if __name__ == '__main__':

    # Datos Iniciales
    x1 = [0.89, 0.41, 0.04, 0.75, 0.15, 0.14, 0.61, 0.25, 0.32, 0.40, 1.26,
          1.68, 1.23, 1.46, 1.38, 1.54, 1.99, 1.76, 1.98, 1.23]
    x2 = [0.41, 0.39, 0.61, 0.17, 0.19, 0.09, 0.32, 0.77, 0.23, 0.74, 1.53,
          1.05, 1.76, 1.60, 1.86, 1.99, 1.93, 1.41, 1.00, 1.54]
    x3 = [0.69, 0.82, 0.83, 0.29, 0.31, 0.52, 0.33, 0.83, 0.81, 0.56, 1.21,
          1.22, 1.33, 1.10, 1.75, 1.75, 1.54, 1.34, 1.83, 1.55]
    X = np.column_stack((x1, x2, x3))
    # '+' = 1
    #  0 = 0
    Y = np.array([1,1,0,1,0,1,1,1,1,1,0,0,0,0,1,0,1,0,0,0])


    # Crear Modelo y entrenarlo
    mdl = LogisticRegression(penalty=None).fit(X,Y)

    # Resultados (Pesos)
    coef = mdl.coef_
    intercept = mdl.intercept_

    print("Coef:" + str(coef))
    print("Intercept:" + str(intercept))
```

Fig. 7. Código completo del ejercicio 3.1.

```
Coef:[[-0.27473348 -0.73403497 -0.90436503]]
Intercept:[1.91474791]
```

Fig. 8. Resultados de ω_1, ω_2, ω_3 y ω_0 respectivamente del ejercicio 3.1.

3.2. Represente en una figura las entradas y la clase a la que pertenece cada uno de los datos de entrenamiento, así como la predicción de cada una de las entradas utilizando el modelo creado en el apartado anterior. Nota: utilice distintos marcadores y colores de forma que se puedan distinguir las clases y el tipo de salida (de los datos de entrenamiento o de la predicción a partir del modelo). Se recomienda usar la librería Matplotlib.

Utilizando la función predict() podemos obtener los resultados de la predicción. Además, podemos utilizar predict_proba() para conocer el porcentaje de seguridad de cada uno de los resultados.

Para representar gráficamente los resultados utilizamos un bucle for para recorrer cada una de las predicciones. Si el valor de salida real es 1, se pintará una cruz. Si es 0, se pintará un círculo. Además, si el valor de salida real coincide con la predicción, se pintará azul. En caso contrario se pintará de color rojo.

```python
# Utilizar modelo entrenado para predecir resultados
Ye = mdl.predict(X)
```

```python
# Representar graficamente
fig = plt.figure
ax = plt.axes(projection='3d')

data_num = 0
for data in X:
    # Si el dato es una cruz
    if Y[data_num] == 1:
        #print("plot cross " + str(data_num))
        # Si la predicción es cruz (acierto): pintar la cruz azul
        if Ye[data_num] == 1:
            ax.scatter3D(X[data_num][0], X[data_num][1], X[data_num][2], marker="X", color="blue")
        # Si la predicción es círculo (error): pintar la cruz roja
        else:
            ax.scatter3D(X[data_num][0], X[data_num][1], X[data_num][2], marker="X", color="red")
    # Si el dato es un círculo
    else:
        #print("plot circle " + str(data_num))
        # Si la predicción es círculo (acierto): pintar el círculo azul
        if Ye[data_num] == 0:
            ax.scatter3D(X[data_num][0], X[data_num][1], X[data_num][2], color="blue")
        # Si la predicción es cruz (error): pintar el círculo rojo
        else:
            ax.scatter3D(X[data_num][0], X[data_num][1], X[data_num][2], color="red")

    data_num += 1

# Nombrar Ejes
ax.set_xlabel('x1', labelpad=20)
ax.set_ylabel('x2', labelpad=20)
ax.set_zlabel('x3', labelpad=20)

plt.show()
```

Fig. 9. Código completo del ejercicio 3.2.

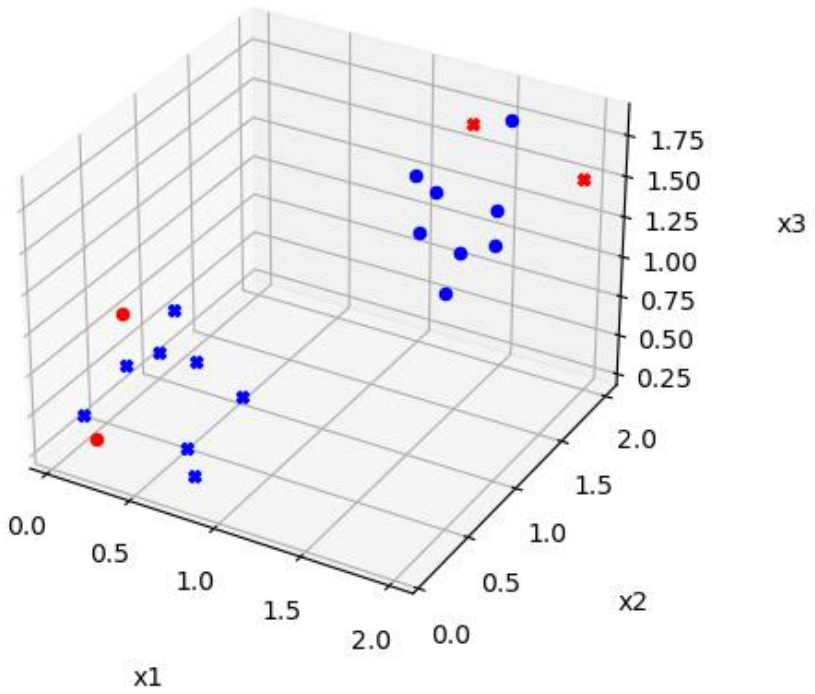

Fig. 10. Resultados visuales del ejercicio 3.2.

3.3. ¿Cuál es el error en porcentaje que presenta el modelo sobre los datos de entrenamiento? ¿A qué puede ser debido?

Usando accuracy_score() se puede obtener la exactitud del modelo y por eso es necesario hacer una resta respecto a 1 y multiplicarlo por 100 para obtener el porcentaje.

El error en porcentaje obtenido es de 19,99% y es debido a la simplicidad del diseño y que la cantidad de muestras en pequeña y por eso no podemos ver un resultado más exacto y porque la salida sólo toma valores de 0 o 1. El resultado obtenido concuerda con la realidad porque de las 20 muestras, hay 4 errores.

```
# Calcular el error en porcentaje
error_percentage = 100 * (1 - accuracy_score(Y, Ye))
print("Error en porcentaje sobre los datos de entrenamiento:", error_percentage, "%")
```

Fig. 11. Código completo del ejercicio 3.3.