Primer parcial Inteligencia Artificial Aplicada

Tema: Ajuste y predicción del numero de pasos del motor de un ventilador pulmonar implementando un modelo de regresión lineal en función de los datos obtenidos de los sensores.

Paso a paso: El código se divide en 6 opciones o ″partes″ y varias funciones de forma de dividir el trabajo e ir paso a paso con la implementación del modelo y lo que eso implica.

Importamos las librerías a utilizar en este caso pandas para administrar el dataset, numpy para crear y trabajar con matrices y vectores así también para utilizar las funciones matemáticas ya integradas en la biblioteca, de sklearn importamos una función que nos permite calcular un parámetro necesario mas adelante y matplotlib lo utilizaremos para representar gráficamente los resultados obtenidos.

```
1  import pandas as pd
2  import numpy as np
3  from sklearn.metrics import r2_score
4  import matplotlib.pyplot as plt
```

Luego recibimos la opción que el usuario ingresa por teclado, leemos y asignamos el nombre ″data″ al dataset suministrado en formato .csv .

```
49  opcion=int(input())
50  # Cargar los datos
51  data = pd.read_csv('Mediciones.csv')
```

Definimos la primera opción del código donde analizamos el dataset e identificamos las características a utilizar y el objetivo, las características serian las variables que utilizaremos en nuestro modelo y el objetivo lo que queremos predecir.

data.shape: nos devuelve el numero de filas y columnas de nuestro dataset ″data″
data.columns[ ]: nos devuelve el índice asignada a las columnas de interés.

```
54  if opcion==1:
55      #imprimir numero de filas y numero de columnas
56      print("Número de filas y columnas:", data.shape)
57
58      #seleccionar las caracteristicas(variables dependientes) y el objetivo
59      caracteristicas = data.columns[0:7 and 9] #[completar]
60      objetivo = data.columns[7] #
61
62      print(caracteristicas)
63      print(objetivo)
```

Definimos la segunda opción donde extraemos de nuestro dataset ″data″ la variable de interés ″VTI_F″ y los valores reales de lo que queremos predecir ″Pasos″ realizamos un .drop a la ultima fila de las columnas extraídas por que contienen un valores nulos ″NaN″ para luego llamar a la función ″regresion_manual( )″ que nos devuelve los coeficientes de una función que aproxima las imágenes de la misma función a los valores reales del ″objetivo″ en este caso ″Pasos″.

```
64  elif opcion==2:
65      # modelo completo solo con VTI_F, completar la funcion regresion manual
66
67      X = data['VTI_F']
68      y = data['Pasos']
69      X = X.drop(24)
70      y = y.drop(24)
71      coef = [regresion_manual(X, y)]# regresion_manual(X, y)
72      print(coef)
```

regresion_manual(X, y) : Es una función que recibe los dataframes ″X″ e ″y″ para devolver los coeficientes de la regresión lineal por el método de la pseudoinversa.

np.ones((X.shape[0], 1)): Una función de numpy que nos devuelve una matriz de unos de las dimensiones (m,n) en este caso m es igual al numero de columnas de X y n es uno.

np.c_[ ]: Esta función concatena las matrices que se agregan a su argumento en nuestro caso concatena la matriz de unos y la matriz X.

La operación que se realiza en la línea 12 sale de unos procesos matemáticos que obviamos en este resumen del mismo nos queda esto:

Donde:

$$\begin{aligned} y_1 &= w_0 + w_1 \cdot x_1 \\ y_2 &= w_0 + w_1 \cdot x_2 \\ y_n &= w_0 + w_1 \cdot x_n \end{aligned} \quad -> \quad \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \qquad A = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_n \end{bmatrix} \qquad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \qquad b = \begin{bmatrix} y_1 \\ y_2 \\ y_n \end{bmatrix}$$

Luego: $w = (A^T A)^{-1} A^T b$

```
7   def regresion_manual(X, y):
8       # Agregar una columna de unos para el término independiente
9       X = np.c_[np.ones((X.shape[0], 1)), X]
10
11      # Calcular los coeficientes utilizando la fórmula de la pseudo inversa
12      coeficientes = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
13
14      return coeficientes
```

Definimos la tercera opción donde realizamos operaciones ya anteriormente explicadas y definimos tres funciones nuevas ″predecir″, ″r2F″, ″rmse″.

```
74  elif opcion==3:
75      # modelo completo solo con VTI_F, completar las funciones que definen las métricas
76      X = data['VTI_F']
77      X = X.drop(24)
78      y = data['Pasos']
79      y = y.drop(24)
80      coef = regresion_manual(X, y)
81      print(coef)
82      y_pred = predecir(X,coef)
83      r2_ = r2F(y, y_pred)
84      rmse_val = rmse(y, y_pred)
85      # imprimir los primeros 2 elementos de y e y_pred
86      #  print(y[:3],  y_pred [COMPLETAR])
87      #print(y_pred)
88      print(y[:3],  y_pred [:3])
89      # imprimir r2 y rmse
90      print(r2_,  rmse_val )
```

La función ″predecir(X, coeficientes)″ recibe ″X″ y ″coeficientes″ y devuelve los ″Pasos″ predichos.

```
16  # Función para predecir los valores de y
17  def predecir(X, coeficientes):
18      Xm = np.c_[np.ones((X.shape[0], 1)), X]
19
20      return  Xm @ coeficientes
```

La función ″r2F(y_true, y_pred)″ recibe los ″Pasos″ dato, y los ″Pasos″ resultado de las predicciones y nos devuelve el coeficiente de determinación r2. El coeficiente determina la calidad del modelo para replicar los resultados, y la proporción de variación de los resultados que puede explicarse por el modelo. Se calculo mediante una función de sklearn y de forma manual dando el mismo resultado.

```
27  def r2F(y_true, y_pred):
28      # https://es.wikipedia.org/wiki/Coeficiente_de_determinaci%C3%B3n
29      # Indagando un poco existe una funcion en sklearn que devuelve el coeficiente de determinacion
30
31      numerador = ((y_true - y_pred) ** 2).sum()
32      denominador = ((y_true - y_true.mean()) ** 2).sum()
33      r2_1 = 1 - (numerador / denominador)
34      r2_2 = r2_score(y_true,y_pred)
35      #print(r2_1)
36      #print(r2_2)
37
38      return 1 - (numerador / denominador)
```

La función ″rmse(y_true, y_pred)″ recibe los ″Pasos″ dato, y los ″Pasos″ resultado de las predicciones y nos devuelve el root mean square error o el error cuadrático medio

```
23    def rmse(y_true, y_pred):
24        error = y_true - y_pred
25        return np.sqrt(np.mean((error) ** 2))
```

La opción cuatro junta las funciones de los pasos anteriores definiendo una nueva función ″ajustar_evaluar_modelo ( )″ para hacer el proceso mas ordenado y en menos líneas de código.

```
91  elif opcion==4:
92      # modelo completo solo con VTI_F, completar la función ajustar_evaluar_modelo
93      X_todo = data['VTI_F']  #data[completar]
94      y = data['Pasos'] # data[completar]
95      X_todo = X_todo.drop(24)
96      y = y.drop(24)
97      coeficientes_todo, y_pred_todo, r2_todo, rmse_todo = ajustar_evaluar_modelo(X_todo, y)
98      print(r2_todo, rmse_todo)
```

Definimos la función ″ajustar_evaluar_modelo(X_todo, y)″ que recibe los las variables necesarias para la regresión lineal y los ″Pasos″ devolviendo los valores que devolvían las funciones definidas anteriormente.

```
40    # Función para ajustar el modelo y evaluarlo
41    def ajustar_evaluar_modelo(X, y):
42        coeficientes = regresion_manual(X, y)
43        y_pred = predecir(X, coeficientes)
44        r2_ =[r2F(y,y_pred)]#completar
45        rmse_val = [rmse(y, y_pred)]#completar
46        return coeficientes, y_pred, r2_, rmse_val
```

En la opción 5 definimos un diccionario dando nombre a los modelos y asignando los índices del dataframe deseados a los nombres de los modelos del diccionario, para luego recorrer en un bucle for todos los modelos y utilizamos los funciones definidas anteriormente.

```
elif opcion==5:
    # Completar la combinaciones de características de los modelos solicitados
    models = {
        'Modelo_1': ['VTI_F'],
        'Modelo_2': ['VTI_F', 'BPM'],
        'Modelo_3': ['VTI_F','PEEP'],
        'Modelo_4': ['VTI_F','PEEP','BPM'],
        'Modelo_5': ['VTI_F','PEEP','BPM','VTE_F'],
      #COMPLETAR EL DICCIONARIO
    }
    for nombre_modelo, lista_caracteristicas in models.items():
        X = data[lista_caracteristicas]#data[completar]
        y = data['Pasos']
        X = X.drop(24)
        y = y.drop(24)
        coeficientes, y_pred, r2, rmse_val = ajustar_evaluar_modelo(X, y)
        print(nombre_modelo,r2, rmse_val)
```

Y la ultima opción, la opción seis utilizamos todas las funciones definidas anteriormente además de esto filtramos los datos de tal forma que lo separamos conforme a cada combinación entre ″PEEP″ y ″BPM″ distintos. Luego graficamos estas predicciones conforme al filtrado que le dimos a nuestro dataset y también graficamos los datos reales para compararlos con las predicciones.

data[ ].unique(): Nos devuelve los valores únicos de la columna o fila que seleccionemos en forma de array.

predicciones_totales = [ ]: Definimos una lista para ir cargando las predicciones.

En el ciclo for recorremos los vectores obtenidos anteriormente gracias a la función .unique() y luego asignamos la condición para la extracción de datos que en este caso es que el ″PEEP″ y el ″BPM″ sean iguales a los valores de los vectores ″valores_peep_unicos″ y ″valores_bpm_unicos″ filtrando así los datos.

predicciones_totales.append( ): Con esta funcion .append( ) vamos cargando la lista ″predicciones_totales″ con las predicciones correspondientes.

predicciones_concatenadas = np.concatenate(predicciones_totales): Al salir de los ciclos for concatenamos las ″predicciones_totales″ para luego aplicar las funciones ″r2F″ y ″rmse″ para comparar con los valores dato.

Para concluir graficamos utilizando la librería ″matplotlib″de las predicciones correspondientes para cada par de ″PEEP″ y ″BPM″ distintos, recorriendo las componentes de ″predicciones_concatenadas[ ]″

```python
elif opcion==6:
    # Modelos para cada combinación de PEEP y BPM
    data = data.drop(24)
    valores_peep_unicos = data['PEEP'].unique()#completar sugerencia, utilizar unique()
    valores_bpm_unicos = data['BPM'].unique() #completar
    type(valores_bpm_unicos)
    print(valores_peep_unicos)
    print(valores_bpm_unicos)
    predicciones_totales = []
    for peep in valores_peep_unicos:
        for bpm in valores_bpm_unicos:

            datos_subset = data[(data['PEEP'] == peep) & (data['BPM'] == bpm)] #completar el filtrado de datos, se deben filtrar los datos para cada para par de PEEP y BPM

            X_subset = datos_subset[['VTI_F']]
            y_subset = datos_subset['Pasos']
            coeficientes_subset, y_pred_subset, r2_subset, rmse_subset = ajustar_evaluar_modelo(X_subset, y_subset)
            print(peep, bpm, r2_subset, rmse_subset)
            predicciones_totales.append(y_pred_subset)
    predicciones_concatenadas = np.concatenate(predicciones_totales)
    y=data['Pasos']
    r2_global = r2F(y, predicciones_concatenadas)
    rmse_global = rmse(y, predicciones_concatenadas)
    print('Global', r2_global, rmse_global)
    #print(predicciones_concatenadas)
    #print(data['VTI_F'])
    # Graficar los resultados obtenidos
    plt.scatter(data['VTI_F'][:6],predicciones_concatenadas[0:6], s=20, c='red', alpha=1.0)
    plt.scatter(data['VTI_F'][6:12],predicciones_concatenadas[6:12], s=20, c='green', alpha=1.0)
    plt.scatter(data['VTI_F'][12:18],predicciones_concatenadas[12:18], s=20, c='black', alpha=1.0)
    plt.scatter(data['VTI_F'][18:24],predicciones_concatenadas[18:24], s=20, c='purple', alpha=1.0)
    plt.scatter(data['VTI_F'],data['Pasos'], s=10, c='orange', alpha=1.0)
    plt.text(200, 30000, 'PEEP=0 BPM=12 (Prediccion)', fontsize=10, color='red')
    plt.text(200, 29000, 'PEEP=0 BPM=20 (Prediccion)', fontsize=10, color='green')
    plt.text(200, 28000, 'PEEP=10 BPM=12 (Prediccion)', fontsize=10, color='black')
    plt.text(200, 27000, 'PEEP=10 BPM=20 (Prediccion)', fontsize=10, color='purple')
    plt.text(200, 26000, 'Valores Reales', fontsize=10, color='orange')
    plt.xlabel('VTI Fluke')  # Nombre del eje x
    plt.ylabel('Pasos')  # Nombre del eje y
    plt.grid(True)  # Mostrar cuadrícula
    plt.show()
```

Mas o menos a grandes rasgos este es el funcionamiento del código para entender mejor la sintaxis o algunas funciones del código es cuestión de revisar las paginas de las librerías donde te dan ejemplos de como usar y que realiza cada función de las librerías.

FabrizioChera