ANEXO

20 de septiembre de 2021

Practica 2: Predicción del costo de casa-habitación (Regresión Lineal)

Dataset disponible en: https://www.kaggle.com/fedesoriano/california-housing-prices-data-extra-features

1. Objetivo de la práctica.

Crear un modelo de regresión lineal que realice la predicción para el costo de las casas habitación en el este de California (década de 1990).

2. Conceptos.

Ya se habló anteriormente de Pandas así que toca hablar de las demás librerías que se usarán.
Scikitlearn es una librería también escrita en Python, enfocada al aprendizaje automático. Incluye diferentes algoritmos de clasificación, regresión y análisis de grupos.
Matplotlib es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python.
El dataset consiste en un archivo .csv que contiene datos de diferentes casas de California en los años 1990's y su respectivo precio.

3. Herramientas a usar.

Computadora con acceso a internet.
Los siguientes programas y librerías:

a. Python versión 3.8.8

b. Anaconda versión 4.10.1

c. Jupyter-lab 3.0.14

d. Pandas versión 1.2.4

e. Scikitlearn versión 0.24.1

f. Matplotlib versión 3.3.4

4. Desarrollo.

4.1. Entender el problema.

El problema consiste en predecir el valor de venta de casas en California en la década de 1990. El dataset incluye datos como el número total de habitaciones de las casas, la edad promedio de las casas, coordenadas, etc. Para este caso, se propone construir un modelo de regresión lineal que realice predicciones de los precios, de acuerdo a sus características.

4.2. Criterio de evaluación.

En este caso, se propone usar la raíz del error cuadrático medio (RMSE) y/o el error medio absoluto (MAE).

4.3. Preparar los datos.

a. Descargue los datos del link que se encuentra al inicio de esta práctica.

b. Extraer los datos.
Use el método mostrado en la práctica 0 sección 3 inciso h para extraerlos datos con Python OPCIONAL
Cree una carpeta para extraer ahí los archivos.

c. Cargar los datos.
Importe la librería pandas y cargue el archivo que en este caso se llama "California_Houses"

```
import pandas as pd
house = pd.DataFrame(pd.read_csv(Path+''California_Houses.csv"))
````

Código 1

d. Explorar los datos.
Use head() para ver el contenido de los datos. También use info() para ver si existen registros vacios.

```
house.head()
house.info()
```

house.head()

	Median_House_Value	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Population	Households	Latitude	Longitude	Distance_to_coast
0	452600.0	8.3252	41	880	129	322	126	37.88	-122.23	9263.040773
1	358500.0	8.3014	21	7099	1106	2401	1138	37.86	-122.22	10225.733072
2	352100.0	7.2574	52	1467	190	496	177	37.85	-122.24	8259.085109
3	341300.0	5.6431	52	1274	235	558	219	37.85	-122.25	7768.086571
4	342200.0	3.8462	52	1627	280	565	259	37.85	-122.25	7768.086571

(a) Función head()

house.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Median_House_Value       20640 non-null  float64
 1   Median_Income            20640 non-null  float64
 2   Median_Age               20640 non-null  int64
 3   Tot_Rooms                20640 non-null  int64
 4   Tot_Bedrooms             20640 non-null  int64
 5   Population               20640 non-null  int64
 6   Households               20640 non-null  int64
 7   Latitude                 20640 non-null  float64
 8   Longitude                20640 non-null  float64
 9   Distance_to_coast        20640 non-null  float64
 10  Distance_to_LA           20640 non-null  float64
 11  Distance_to_SanDiego     20640 non-null  float64
 12  Distance_to_SanJose      20640 non-null  float64
 13  Distance_to_SanFrancisco  20640 non-null  float64
dtypes: float64(9), int64(5)
memory usage: 2.2 MB
```

(b) Función info

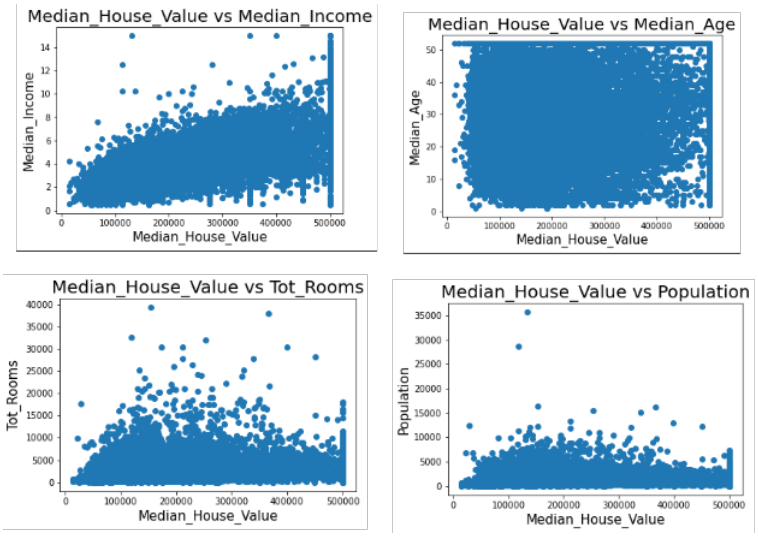
Como no hay datos nulos, ya se puede trabajar directamente sobre ellos.

e. Preprocesamiento de los datos.
Primero se realizan una serie de gráficas para darse una idea general de cómo se van determinando los precios de acuerdo a algunas características (Ingreso Promedio, Edad Promedio, Total de Habitaciones y Población).

```
def grafica(data, colx, coly):
    plt.plot(data[colx], data[coly], 'o')
    plt.xlabel(colx, size = 15)
    plt.ylabel(coly, size = 15)
    plt.title(colx+' vs '+ coly, size = 20)
    plt.show()

grafica(house, 'Median_House_Value', 'Median_Income')
grafica(house, 'Median_House_Value', 'Median_Age')
grafica(house, 'Median_House_Value', 'Tot_Rooms')
grafica(house, 'Median_House_Value', 'Population')
```

Código 2

El resultado de este código se puede ver en la figura c. Como se puede



(c) Valor promedio vs 4 características

apreciar, la mayoría de los precios se concentran en la primera mitad de las variables independientes(Ingreso Promedio, Edad Promedio, Total de Habitaciones y Población).

f. Dividir los datos en X y y.
Use el siguiente código para dividir los datos en X y y. Estos ayudarán a entrenar el modelo.

```
    y = house['Median_House_Value']
    X = house.iloc[:,1:]
    
```

Código 3

Compruebe que X y y tienen las columnas correctas usando head().[Ver figuras d y e]

```
X.head()
y.head()
```

g. Escalar los datos en X.
Hay dos formas de escalar los datos, usando StandardScaler y MinMaxScaler. Se usarán ambos para ver la diferencia entre ellos.[Ver las figuras f y g]
Importe las librerías necesarias y use StandardScaler.

X.head()

	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Population	Households	Latitude	Longitude	Distance_to_coast	Distance_to_LA	Dista
0	8.3252	41	880	129	322	126	37.88	-122.23	9263.040773	556529.158342	
1	8.3014	21	7099	1106	2401	1138	37.86	-122.22	10225.733072	554279.850069	
2	7.2574	52	1467	190	496	177	37.85	-122.24	8259.085109	554610.717069	
3	5.6431	52	1274	235	558	219	37.85	-122.25	7768.086571	555194.266086	
4	3.8462	52	1627	280	565	259	37.85	-122.25	7768.086571	555194.266086	

(d) Datos en X

```
y.head()

0    452600.0
1    358500.0
2    352100.0
3    341300.0
4    342200.0
Name: Median_House_Value, dtype: float64
```

(e) Datos en y

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc = StandardScaler()
Xsc = X.copy()
Xsc.iloc[:,:] = sc.fit_transform(X.iloc[:, :])
```

Código 4

Ahora use MinMaxScaler para hacer otro dataframe con los datos escalados.

```
scaler = MinMaxScaler()
Xnorm = X.copy()
scaled_values = scaler.fit_transform(X)
Xnorm.loc[:,:] = scaled_values
```

Código 5

```
Xsc.head()
```

	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Population	Households	Latitude	Longitude	Distance_to_coast	Distance_to_LA	Dista
0	2.344766	0.982143	-0.804819	-0.970706	-0.974429	-0.977033	1.052548	-1.327835	-0.635876	1.158969	
1	2.332238	-0.607019	2.045890	1.348649	0.861439	1.669961	1.043185	-1.322844	-0.616285	1.149889	
2	1.782699	1.856182	-0.535746	-0.825895	-0.820777	-0.843637	1.038503	-1.332827	-0.656307	1.151224	
3	0.932968	1.856182	-0.624215	-0.719067	-0.766028	-0.733781	1.038503	-1.337818	-0.666299	1.153580	
4	-0.012881	1.856182	-0.462404	-0.612239	-0.759847	-0.629157	1.038503	-1.337818	-0.666299	1.153580	

(f) X StandardScaler

```
Xnorm.head()
```

	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Population	Households	Latitude	Longitude	Distance_to_coast	Distance_to_LA	Dista
0	0.539668	0.784314	0.022331	0.019863	0.008941	0.020556	0.567481	0.211155	0.027398	0.546362	
1	0.538027	0.392157	0.180503	0.171477	0.067210	0.186976	0.565356	0.212151	0.030283	0.544152	
2	0.466028	1.000000	0.037260	0.029330	0.013818	0.028943	0.564293	0.210159	0.024390	0.544477	
3	0.354699	1.000000	0.032352	0.036313	0.015555	0.035849	0.564293	0.209163	0.022918	0.545050	
4	0.230776	1.000000	0.041330	0.043296	0.015752	0.042427	0.564293	0.209163	0.022918	0.545050	

(g) X MinMaxScaler

4.4. Construir el modelo.

Como se harán las mismas pruebas con los 2 datasets escalados, el división de datos, la construcción del modelo y la predicción se realizarán en una misma función.

Inserte el siguiente código para realizar el modelo, entrenamiento y predicción.

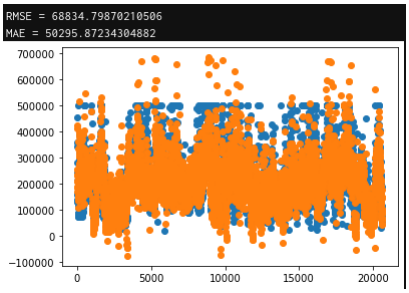
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error

def regression(X, y):
    #Dividimos los datos
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.3, shuffle=True)

    #Construir el modelo
    model = LinearRegression()
    model.fit(X_train, y_train)

    #Predicciones
    y_pred = model.predict(X_test)

    print(''RMSE =", mean_squared_error(y_test, y_pred, squared=False))
    print(''MAE =", mean_absolute_error(y_test, y_pred))
```

```
    plt.plot(y_test.index, y_test, 'o')
    plt.plot(y_test.index, y_pred, 'o')
return y_test, y_pred
```

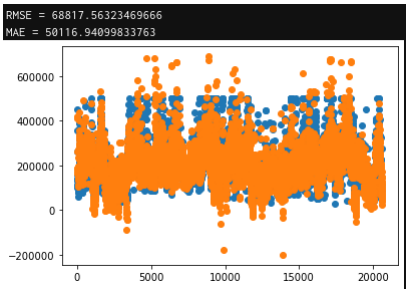
Código 6

Aplique la función usando los datasets hechos en el apartado 4.3 g).[Vea las figuras h y i]

```
y_test_sc, y_sc = regression(Xsc, y)
y_test_norm, y_norm = regression(Xnorm, y)
```



(h) Predicción con StandardScaler



(i) Predicción con MinMaxScaler

En este caso, el resultado del RMSE y MAE están expresados en dólares, es decir, el error de predicción está entre los 50000 y 70000 dólares, pero si se considera que los valores de las propiedades ronda en una escala de cientos de miles de dólares, este error es aceptable.

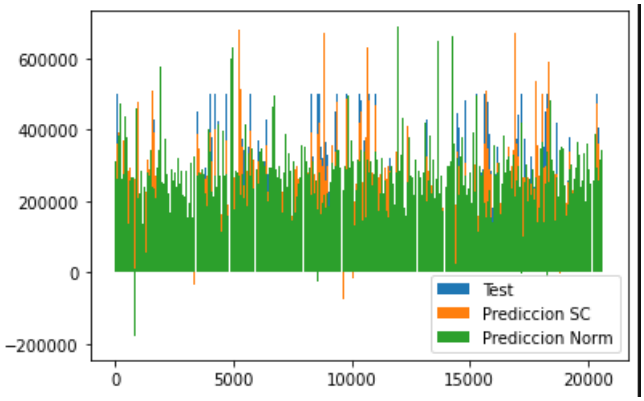
4.5. Análisis de errores.

Al usar MinMaxScaler se tienen mejores resultados que al usar StandardScaler. Así que se van a comparar los resultados de ambos métodos de escalamiento de datos.
Use el siguiente código para ver los resultados.

```
x_axis = y_test_sc.index
plt.bar(x_axis, y_test_sc, width = 10, label = 'Test')
plt.bar(x_axis, y_sc, width = 10, label = 'Prediccion SC')
plt.bar(x_axis, y_norm, width = 10, label = 'Prediccion Norm')
plt.legend()
plt.show()
```

Código 7

El resultado es el siguiente.



(j) Comparación de las predicciones

Como se puede apreciar, la mayoría de las predicciones se encuentran cerca de los precios reales. Los picos naranjas y verdes son los que se muestran con el RMSE y MAE.

4.6. Implementación.

Ver el Notebook de Jupyter llamado *P2-Regresion-lineal.ipynb*