UNIVERSIDAD DE PIURA Curso Python Nivel 2

Trabajo Final

Alcalde Paredes, Carla Valeria
Calopiña Arbulú, Bruno Milton
Fernández Aquino, Henry Ismael Junior
Luna Otero, Vincenzo Antonio
Oquelis Gonzales, Alessandra
Oquelis Guerrero, María José

6 de febrero de 2022

Índice

1. Introducción

En el presente trabajo se han aplicado los métodos, algoritmos, modelos, etc., de Machine Learning aprendidos en el curso de Python 2, con la finalidad que los alumnos los practiquen adecuadamente.

El objetivo principal del trabajo es analizar un dataset obtenido de la página “Kaggle”, que pueda ser analizado y modelado para poder brindar un valor de predicción para cualquier dato. El dataset escogido se llama ‘Breast Cancer’, el cual brinda información acerca de las características de los tumores como: área, perímetro, textura, suavidad, compacidad, concavidad.

Se han escogido las características y features más importantes en base al target que se ha elegido.

Además, se han utilizado distintas librerías para realizar análisis estadísticos y gráficas, junto con un entrenamiento del modelo que permita estudiar los resultados.

2. Análisis de problema

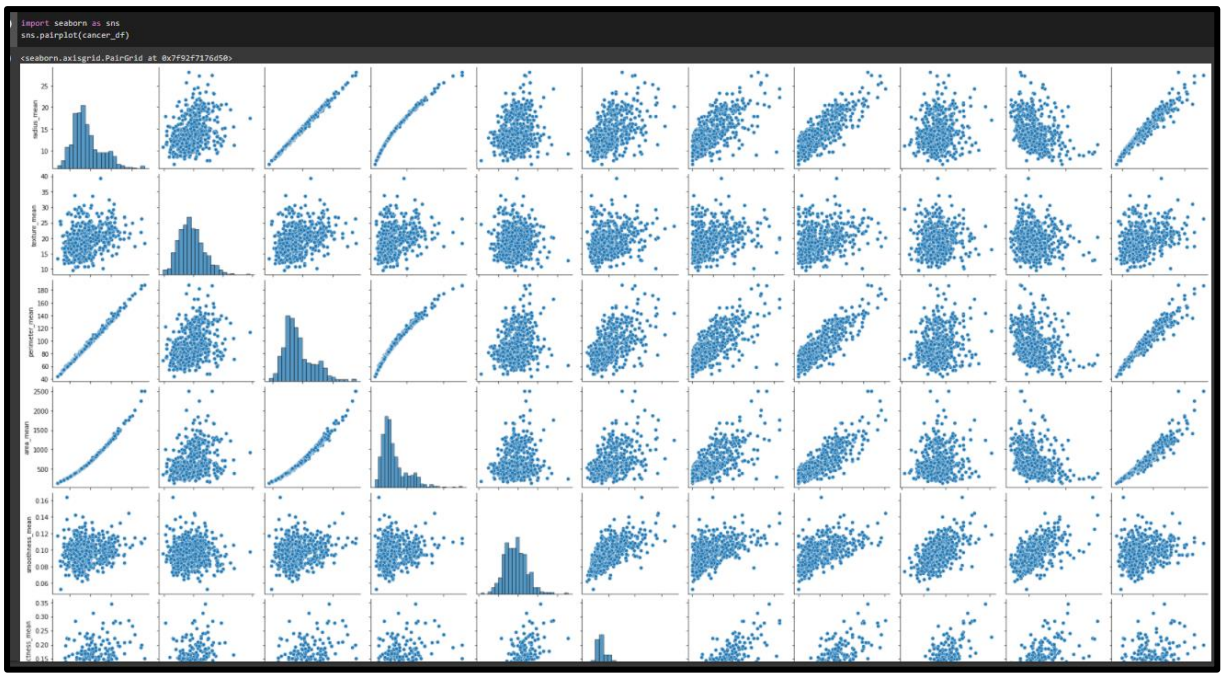
Con la data breast-cancer se generarán 4 modelos de Machine learning para poder predecir el valor del “radius_worst” de la persona.

Primero se importa la data y se eliminan algunas columnas ya que de lo contrario toma mucho tiempo en procesar la información.

```
[ ] import pandas as pd
    import numpy as np
    cancer = pd.read_csv('https://raw.githubusercontent.com/pedrorotta/PythonIntermedio2022/main/clase5/breast-cancer.csv')
    cancer_df = cancer.drop(cancer.columns[[0,1,12,13,14,15,16,17,18,19,20,21,23,24,25,26,27,28,29,30,31]], axis = 'columns')
    display(cancer_df)
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	radius_worst
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	25.380
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	24.990
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	23.570
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	14.910
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	22.540
...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	25.450
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	23.690
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	18.980
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	25.740
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	9.456

Luego se hace un análisis estadístico con seaborn. Se ve que muchas graficas se aproximan a una distribución normal y que ciertos features tienen una relación lineal.



Después se crea una función para medir la colinealidad entre features, VIF, este debe tener un valor menor a 5 y si no, se elimina.

```python
from sklearn.linear_model import LinearRegression
def calcular_vif (df, características):
    vif, tolerancia = {}, {}
    # todas las características que desea examinar
    for característica in características:
        # extraiga todas las demás características con las que retrocederá
        X = [f for f in características if f != característica]
        X, y = df[X], df[característica]
        # extraer r-cuadrado del ajuste
        r2 = LinearRegression().fit(X, y).score(X, y)

        # calcular tolerancia
        tolerancia[característica] = 1 - r2
        # calcular VIF
        vif[característica] = 1/(tolerancia[característica])
    # devuelve VIF DataFrame
    return pd.DataFrame({'VIF': vif, 'Tolerancia': tolerancia})
```

```python
calcular_vif(df= cancer_df,características = (#'radius_mean',
                                              'texture_mean',
                                              #'perimeter_mean',
                                              'area_mean',
                                              'smoothness_mean',
                                              #'compactness_mean',
                                              'concavity_mean',
                                              #'concave points_mean',
                                              'symmetry_mean',
                                              'fractal_dimension_mean'))
```

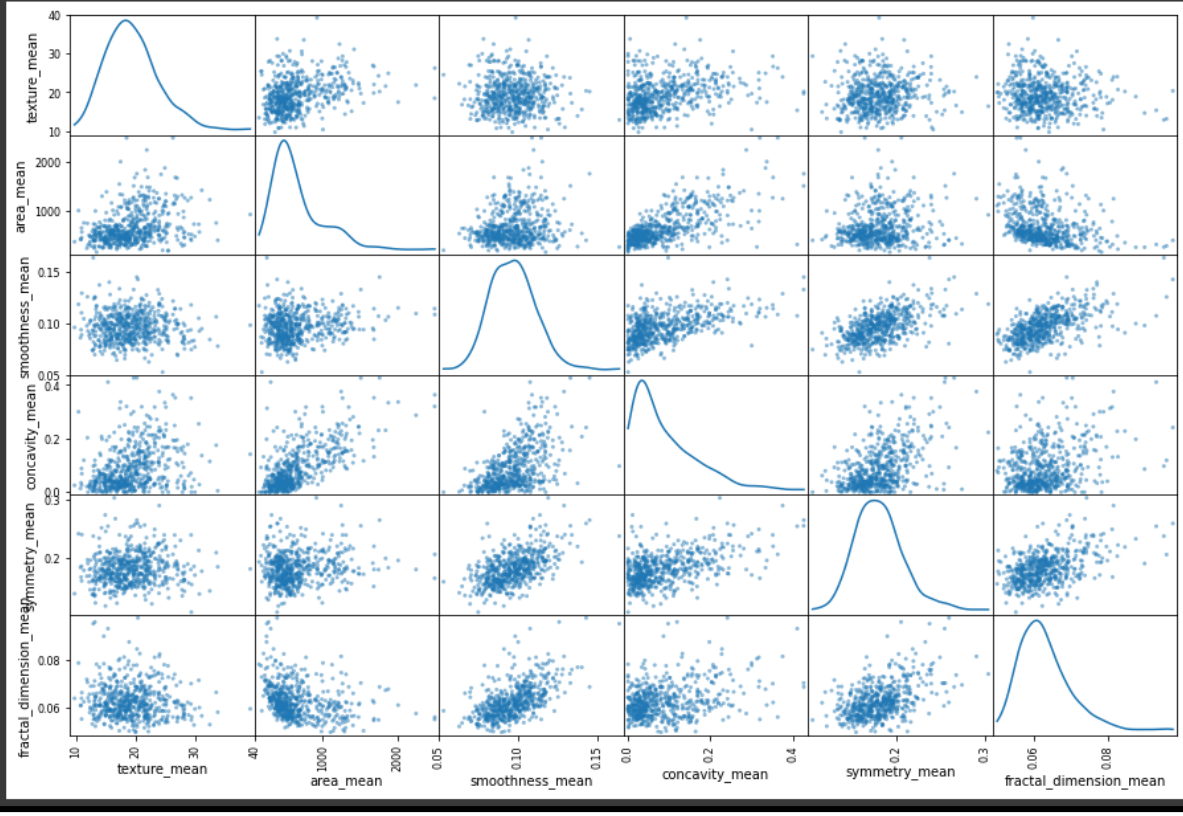
	VIF	Tolerancia
texture_mean	1.173247	0.852336
area_mean	4.471599	0.223634
smoothness_mean	2.098786	0.476466
concavity_mean	4.921308	0.203198
symmetry_mean	1.686023	0.593112
fractal_dimension_mean	3.311538	0.301974

```python
X = cancer_df.drop(columns= ['radius_mean','perimeter_mean','compactness_mean','concave points_mean','radius_worst'])
y = cancer_df['radius_worst']
display(X.head())
```

	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	fractal_dimension_mean
0	10.38	1001.0	0.11840	0.3001	0.2419	0.07871
1	17.77	1326.0	0.08474	0.0869	0.1812	0.05667
2	21.25	1203.0	0.10960	0.1974	0.2069	0.05999
3	20.38	386.1	0.14250	0.2414	0.2597	0.09744
4	14.34	1297.0	0.10030	0.1980	0.1809	0.05883

Luego se generan gráficas estadísticas con matplotlib para analizar la disposición de los datos y se ve que siguen una distribución normal.

```
import matplotlib.pyplot as plt
pd.plotting.scatter_matrix(X,figsize = (15,10) , diagonal = 'kde')
plt.show()
```



Preprocesamiento: Los features se estandarizan, escalan, teniendo en cuenta un rango entre 0-1, porque hay valores muy distintos y se añaden a un nuevo dataframe.

Estandarizar

```
from sklearn.preprocessing import StandardScaler as SS

X_array = np.array(X)
x = X_array[:,1]
x1 = X_array[:,3]
x = np.reshape(x,(len(x),1))
x1 = np.reshape(x1,(len(x1),1))


X_stand = []
for i in [x,x1]:
  standard = SS().fit(i)
  X_1 = standard.transform(i)
  X_stand.append(X_1)

print(X_stand[0])
```

Escalar

```
[ ] from sklearn.preprocessing import MinMaxScaler as MMS

x_stand = X_stand[0]
x_stand_1 = X_stand[1]
x2 = x = X_array[:,0]
x2 = np.reshape(x2,(len(x2),1))

X_scaler = []
for j in [x_stand,x_stand_1,x2]:
  scaler = MMS(feature_range = (0,1))
  x_scaler = scaler.fit_transform(j)
  X_scaler.append(x_scaler)

print(X_scaler[0])
```

Nuevo dataframe

```
[ ] X_final = X.drop(columns = ['texture_mean','area_mean','concavity_mean'])
    X_final['texture_mean']= X_scaler[2]
    X_final['area_mean']= X_scaler[0]
    X_final['concavity_mean']= X_scaler[1]
    display(X_final)
```

	smoothness_mean	symmetry_mean	fractal_dimension_mean	texture_mean	area_mean	concavity_mean
0	0.11840	0.2419	0.07871	0.022658	0.363733	0.703140
1	0.08474	0.1812	0.05667	0.272574	0.501591	0.203608
2	0.10960	0.2069	0.05999	0.390260	0.449417	0.462512
3	0.14250	0.2597	0.09744	0.360839	0.102906	0.565604
4	0.10030	0.1809	0.05883	0.156578	0.489290	0.463918
...	...	...	...	...	...	...
564	0.11100	0.1726	0.05623	0.428813	0.566490	0.571462
565	0.09780	0.1752	0.05533	0.626987	0.474019	0.337395
566	0.08455	0.1590	0.05648	0.621238	0.303118	0.216753
567	0.11780	0.2397	0.07016	0.663510	0.475716	0.823336
568	0.05263	0.1587	0.05884	0.501522	0.015907	0.000000

569 rows × 6 columns

Los 4 modelos de regresión son Random Forest Regressor, Random Forest Regressor con PCA, Gradient Boosting Regressor y Gradient Boosting Regressor con PCA.

Se verifica la forma de los vectores y se divide la data en 20% test y 80% train.

```
[ ] X_2 = np.array(X_final)
    y_2 = np.array(y)
    y_2 = np.reshape(y_2,(len(y_2),1))
    print(np.shape(y_2))
```

```
(569, 1)
```

```
[ ] X_2train,X_2test,y_2train,y_2test = tts(X_2,y_2,test_size = 0.2, random_state = 42)
```

Se definen los modelos, se los entrena y se hallan las métricas: r2 score train, r2 score test y %gen.

$$\%gen = \frac{r2\ score\ train - r2\ score\ test}{r2\ score\ train} \times 100$$

Sin PCA:

```
[ ]  modeloRFR = RFR(random_state = 42)
     modeloSGDR = SGDR(random_state = 42)
```

```
[ ]  r2_score_train = []
     r2_score_test = []
     gen_percent = []
     for k in [modeloRFR,modeloSGDR]:
       k.fit(X_2train,y_2train)
       ypredtrain = k.predict(X_2train)
       ypredtest = k.predict(X_2test)

       r2scoretrain = r2_score(y_2train,ypredtrain)
       r2_score_train.append(r2scoretrain)

       r2scoretest= r2_score(y_2test,ypredtest)
       r2_score_test.append(r2scoretest)

       gen = ((r2scoretrain-r2scoretest)/r2scoretrain)*100
       gen_percent.append(gen)

     dicc = {'Modelo':['Random Forest Regressor','Gradient Boosting Regressor']}
     dicc_df = pd.DataFrame(dicc)
     dicc_df['r2 score train'] = r2_score_train
     dicc_df['r2 score test'] = r2_score_test
     dicc_df['%gen'] = gen_percent
     display(dicc_df)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DataConversic
  """
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:494: DataConve
  y = column_or_1d(y, warn=True)
```

	Modelo	r2 score train	r2 score test	%gen
0	Random Forest Regressor	0.991558	0.953269	3.861442
1	Gradient Boosting Regressor	0.987127	0.962436	2.501295

Con PCA: El análisis con componentes principales es un método que se usa para reducir la dimensionalidad de la data dadas variables correlacionadas, donde n_components en la sintaxis es en número de componentes a mantener.

Modelo de Regresión con PCA

```
from sklearn.decomposition import PCA

r2_score_train_pca = []
r2_score_test_pca = []
gen_percent_pca= []

pca1 = PCA(n_components = 3)
pca1.fit(X_2)
Xpca = pca1.transform(X_2)

Xpcatrain,Xpcatest,y_2train,y_2test = tts(Xpca,y_2,test_size = 0.2,random_state = 42)

for m in [modeloRFR,modeloSGDR]:
  m.fit(Xpcatrain,y_2train)
  ypredtrain_pca = m.predict(Xpcatrain)
  ypredtest_pca = m.predict(Xpcatest)

  r2scoretrainpca = r2_score(y_2train,ypredtrain_pca)
  r2_score_train_pca.append(r2scoretrainpca)

  r2scoretestpca= r2_score(y_2test,ypredtest_pca)
  r2_score_test_pca.append(r2scoretestpca)

  gen_pca = ((r2scoretrainpca-r2scoretestpca)/r2scoretrainpca)*100
  gen_percent_pca.append(gen_pca)

dicc_1 = {'Modelo':['Random Forest Regressor PCA','Gradient Boosting Regressor PCA']}
dicc_1_df = pd.DataFrame(dicc_1)
dicc_1_df['r2 score train'] = r2_score_train_pca
dicc_1_df['r2 score test'] = r2_score_test_pca
dicc_1_df['%gen'] = gen_percent_pca
display(dicc_1_df)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: DataConversionWarning:

/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:494: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

	Modelo	r2 score train	r2 score test	%gen
0	Random Forest Regressor PCA	0.988396	0.934851	5.417343
1	Gradient Boosting Regressor PCA	0.980078	0.939035	4.187722

3. Análisis de resultados

	Modelo	r2 score train	r2 score test	%gen
0	Random Forest Regressor	0.991558	0.953269	3.861442
1	Gradient Boosting Regressor	0.987127	0.962436	2.501295

	Modelo	r2 score train	r2 score test	%gen
0	Random Forest Regressor PCA	0.988396	0.934851	5.417343
1	Gradient Boosting Regressor PCA	0.980078	0.939035	4.187722

Con los resultados obtenidos se puede afirmar que los cuatro modelos generalizan bien la data siendo los mejores los modelos sin PCA.

Finalmente, la función predicción() permitirá predecir el valor de Radius worst al ingresar los valores de los features que se piden. En el ejemplo se ha hecho uso de la fila 542, la cual tiene como Radius worst real 16.510

```
[138] 1 radio_list = []
      2 def predicción():
      3   texture_mean = float(input('Ingresa el valor del texture mean: '))
      4   smoothness_mean = float(input('Ingresa el valor del smoothness mean: '))
      5   area_mean = float(input('Ingresa el valor del area mean: '))
      6   concavity_mean = float(input('Ingresa el valor del concavity mean: '))
      7   symetry_mean = float(input('Ingresa el valor del symetry mean: '))
      8   fractal_dimension_mean = float(input('Ingresa el valor del fractal dimension mean: '))
      9
     10   for j in [modeloRFR,modeloSGDR]:
     11     global radio
     12     radio = j.predict([[texture_mean,area_mean,smoothness_mean,concavity_mean,symetry_mean,fractal_dimension_mean]])
     13     radio_list.append(radio)
     14
     15   predicciones_dicc = {'Modelos':['Random Forest Regressor','Gradient Boosting Regressor'],'Radius worst':[radio_list[0][0],radio_list[1][0]]}
     16   predicciones_df = pd.DataFrame(predicciones_dicc)
     17   return(predicciones_df)

     1 predicción()

Ingresa el valor del texture mean: 25.42
Ingresa el valor del smoothness mean: 0.08275
Ingresa el valor del area mean: 668.6
Ingresa el valor del concavity mean: 0.041050
Ingresa el valor del symetry mean: 0.1840
Ingresa el valor del fractal dimension mean: 0.05680
```

	Modelos	Radius worst
0	Random Forest Regressor	15.329600
1	Gradient Boosting Regressor	16.310017

4. Conclusiones

- Al estudiar ambos cursos de Python, hemos podido aprender la automatización de procesos con la intención de mejorarlos con innovación y eficiencia.

- Nos ha permitido poder aproximar los datos en una simplificación de procesos para evitar complicaciones en su estudio.

- Analizar distintas datasest con sus gráficos para poder observar su variación de datos con la finalidad de evaluar que tan correctas son, entre ellos analizar su normalidad.

- Al evaluar nuestra dataset, podemos darnos cuenta que muchas gráficas se aproximan a una distribución normal y que ciertos features tienen una relación lineal.

- Debido a que los datos presentan una gran varianza, es de esperar que al aplicar PCA los resultados no presenten una mejora.

- Gracias a estos conocimientos se ha podido analizar de una manera adecuada una determinada dataset, con la finalidad de predecir valores que nos permiten evaluar determinadas variables.