Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución

September 9, 2022

Francisco Leonid Galvez Flores

A01174385

Enlace al repositorio en GitHub

1 Librerias:

[701]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

Importamos las librerias necesarias para algunos procesos a realizar antes de la implementacion del modelo.

2 Importacion de la base de datos:

[702]:
```
url = "https://raw.githubusercontent.com/G4LF0/MLAlgorithmFramework/main/
↪breast-cancer.csv"
```

[703]:
```
df = pd.read_csv(url)
df.head(2)
```

[703]:
```
       id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0  842302         M        17.99         10.38           122.8     1001.0
1  842517         M        20.57         17.77           132.9     1326.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017

   ...  radius_worst  texture_worst  perimeter_worst  area_worst  \
0  ...         25.38          17.33            184.6      2019.0
1  ...         24.99          23.41            158.8      1956.0

   smoothness_worst  compactness_worst  concavity_worst  concave points_worst  \
```

```
0            0.1622            0.6656           0.7119                0.2654
1            0.1238            0.1866           0.2416                0.1860

   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
1          0.2750                  0.08902

[2 rows x 32 columns]
```

Declaramos el dataframe con ayuda de la libreria de pandas, la base de datos se encuentra alojada en nuestro repositorio en GitHub.

3 Analisis exploratorio de los datos:

[704]:
```
df.isnull().sum()
```

[704]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
```

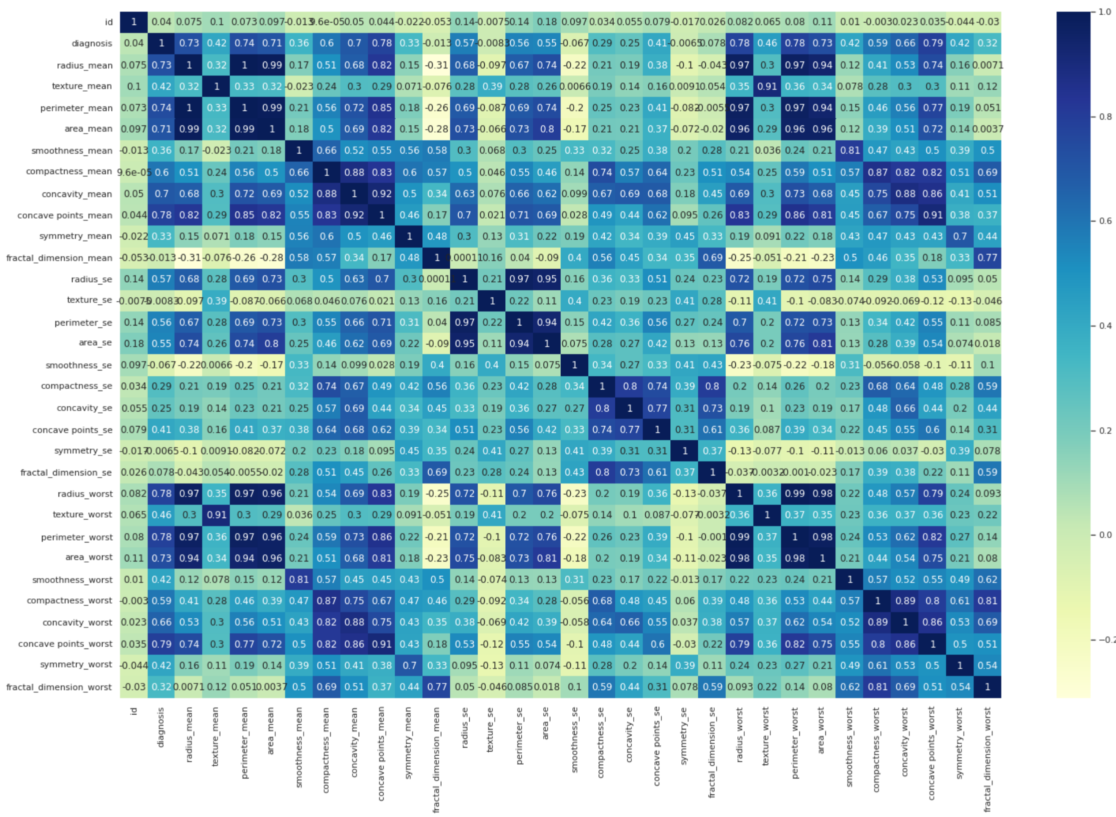
```
fractal_dimension_worst    0
dtype: int64
```

[705]:
```python
df.diagnosis = df.diagnosis.map({"M":1, "B":0})
df.shape
```

[705]:
```
(569, 32)
```

[706]:
```python
sns.set(rc = {'figure.figsize':(25,16)})
sns.heatmap(df.corr(), annot=True, cmap= 'YlGnBu')
```

[706]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7ebbc6e150>
```



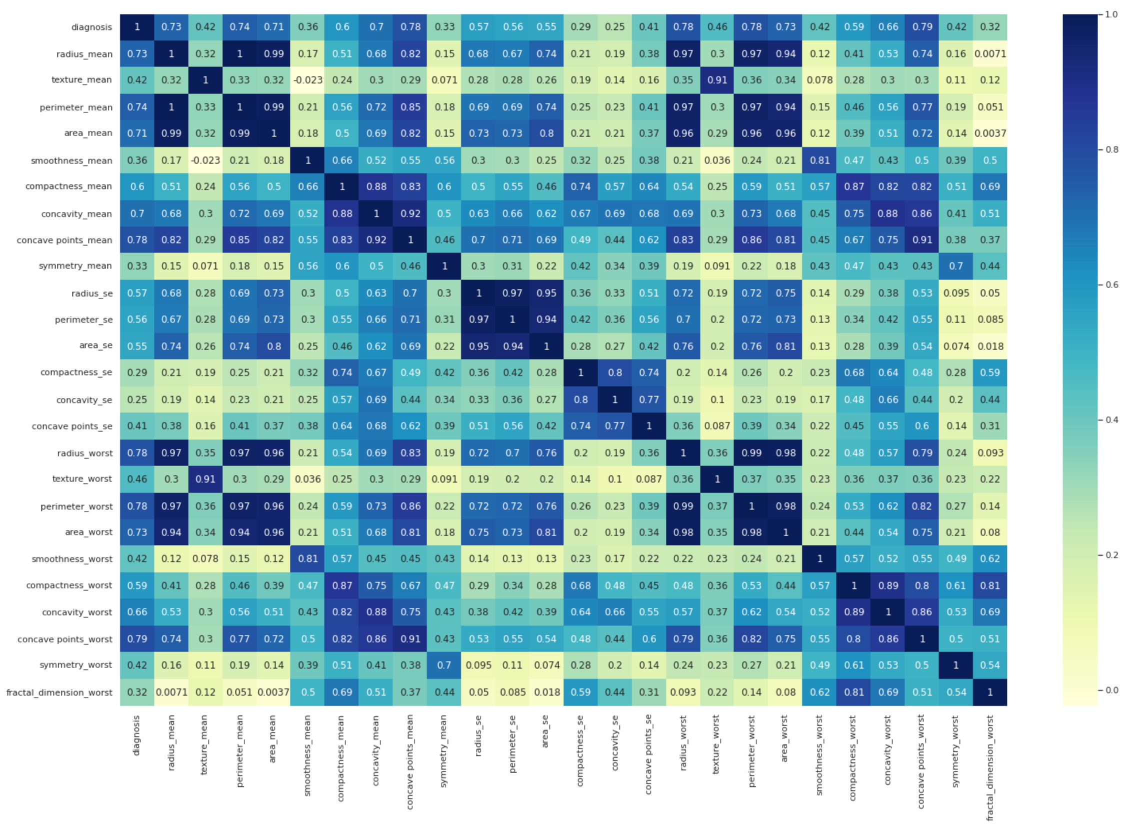
[707]:
```python
df = df.drop(["id",
 ↪"fractal_dimension_se","symmetry_se","smoothness_se","texture_se",
 ↪"fractal_dimension_mean"], axis = 1)
df.head(1)
```

[707]:
```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38           122.8     1001.0
```

```
   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0           0.1184            0.2776          0.3001               0.1471

   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  ...         25.38          17.33            184.6

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                   0.1189

[1 rows x 26 columns]
```

[708]:
```python
sns.set(rc = {'figure.figsize':(25,16)})
sns.heatmap(df.corr(), annot=True, cmap= 'YlGnBu')
```

[708]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7eb6f81110>



En esta parte lo que hicimos fue analizar columnas que no tuvieran relacion con la variable a predecir, asi como tambien cambiamos el valor de algunos variables para convertirlas en binarias.

4 Division de los datos en trainig, validating and testing:

[709]:
```
x = df.drop(["diagnosis"], axis = 1)
y = df.diagnosis
```

[710]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x.values, y.values,␣
 ↪train_size=0.9, random_state=16)
```

[711]:
```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, train_size=0.
 ↪7, random_state=16)
```

[712]:
```
datos_n = [x_train.shape[0], x_val.shape[0], x_test.shape[0]]
```

[713]:
```
df_mcd = pd.DataFrame({"Datos: ":["Training", "Validating", "Testing"],
                       "Registros": datos_n})
df_mcd
```

[713]:

	Datos:	Registros
0	Training	358
1	Validating	154
2	Testing	57

En esta parte lo que hacemos es divir el data set, al principio lo dividimos en un 90% para training y un 10% para test, despues lo que hacemos es que del 30% del 90% de training lo convertimos en validating.

5 Implementacion del modelo:

[714]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logreg = LogisticRegression(max_iter= 10000)
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_train)
```

Implementamos el algoritmo de regresion logistica con un numero de iteraciones maxima de 10,000.

Accuracy for training:

[715]:
```
score =accuracy_score(y_train,y_pred)
accuracy_training = score
print("Accuracy for the training set is: ", round(score*100, 4))
```

```
Accuracy for the training set is:  95.8101
```

Accuracy for validating:

[716]:
```python
y_pred = logreg.predict(x_val)
score =accuracy_score(y_val,y_pred)
accuracy_validating = score
print("Accuracy for the training set is: ", round(score*100, 4))
```

```
Accuracy for the training set is:  96.1039
```

Accuracy for testing

[717]:
```python
y_pred = logreg.predict(x_test)
score =accuracy_score(y_test,y_pred)
accuracy_testing = score
print("Accuracy for the training set is: ", round(score*100, 4))
```

```
Accuracy for the training set is:  94.7368
```

6 Metricas:

Metricas del training:

[718]:
```python
from sklearn import metrics

y_pred = logreg.predict(x_train)
confusion_matrix = metrics.confusion_matrix(y_train, y_pred)
confusion_matrix
```

[718]:
```
array([[220,   6],
       [  9, 123]])
```

[719]:
```python
sns.heatmap(confusion_matrix, annot=True)
```

[719]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7eb6a08350>
```

[720]:
```
from sklearn.metrics import classification_report

target_names = ['Sin tumor', 'Con tumor']
print(classification_report(y_train, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

   Sin tumor       0.96      0.97      0.97       226
   Con tumor       0.95      0.93      0.94       132

    accuracy                           0.96       358
   macro avg       0.96      0.95      0.95       358
weighted avg       0.96      0.96      0.96       358
```

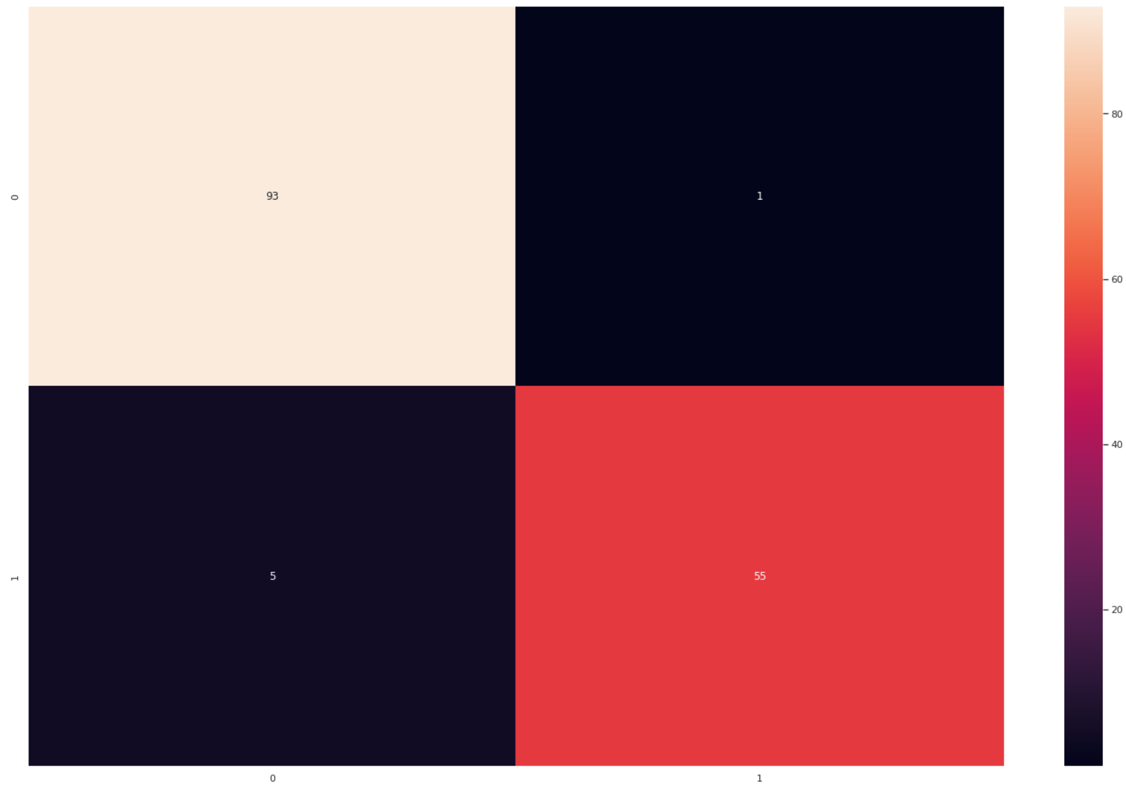
Metricas del validating:

[721]:
```
from sklearn import metrics

y_pred = logreg.predict(x_val)
confusion_matrix = metrics.confusion_matrix(y_val, y_pred)
confusion_matrix
```

[721]:
```
array([[93,  1],
       [ 5, 55]])
```

[722]:
```
sns.heatmap(confusion_matrix, annot=True)
```

[722]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7eb69f40d0>
```



[723]:
```
from sklearn.metrics import classification_report

target_names = ['Sin tumor', 'Con tumor']
print(classification_report(y_val, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

   Sin tumor       0.95      0.99      0.97        94
   Con tumor       0.98      0.92      0.95        60

    accuracy                           0.96       154
   macro avg       0.97      0.95      0.96       154
weighted avg       0.96      0.96      0.96       154
```

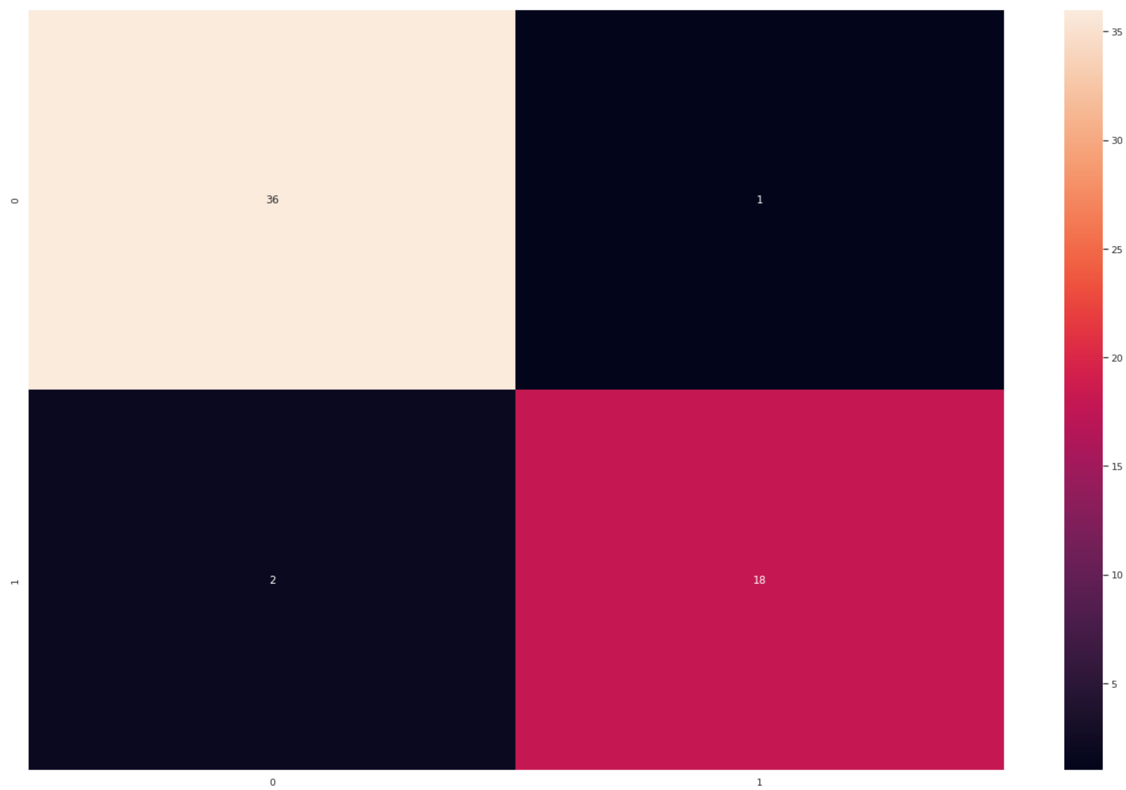
Metricas del testing:

[724]:
```python
from sklearn import metrics

y_pred = logreg.predict(x_test)
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
confusion_matrix
```

[724]:
```
array([[36,  1],
       [ 2, 18]])
```

[725]:
```python
sns.heatmap(confusion_matrix, annot=True)
```

[725]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7eb68cd190>
```



[726]:
```python
from sklearn.metrics import classification_report

target_names = ['Sin tumor', 'Con tumor']
print(classification_report(y_test, y_pred, target_names=target_names))
```

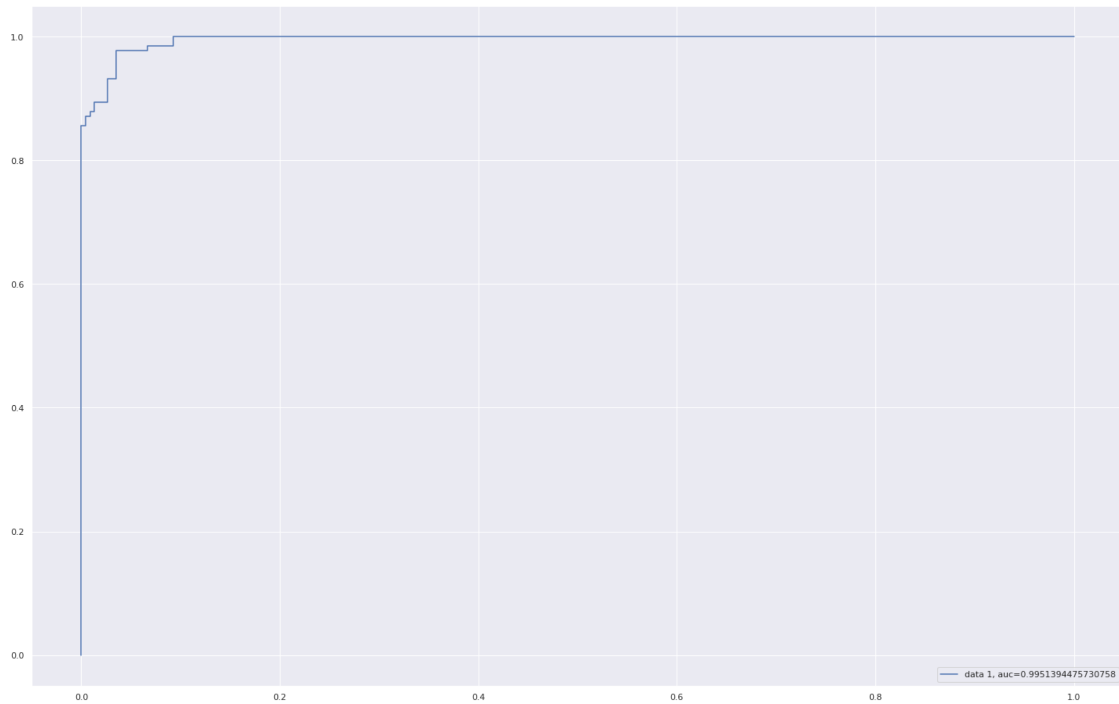
```
              precision    recall  f1-score   support

   Sin tumor       0.95      0.97      0.96        37
   Con tumor       0.95      0.90      0.92        20
```

```
    accuracy                           0.95        57
   macro avg       0.95      0.94      0.94        57
weighted avg       0.95      0.95      0.95        57
```

7 Curva ROC

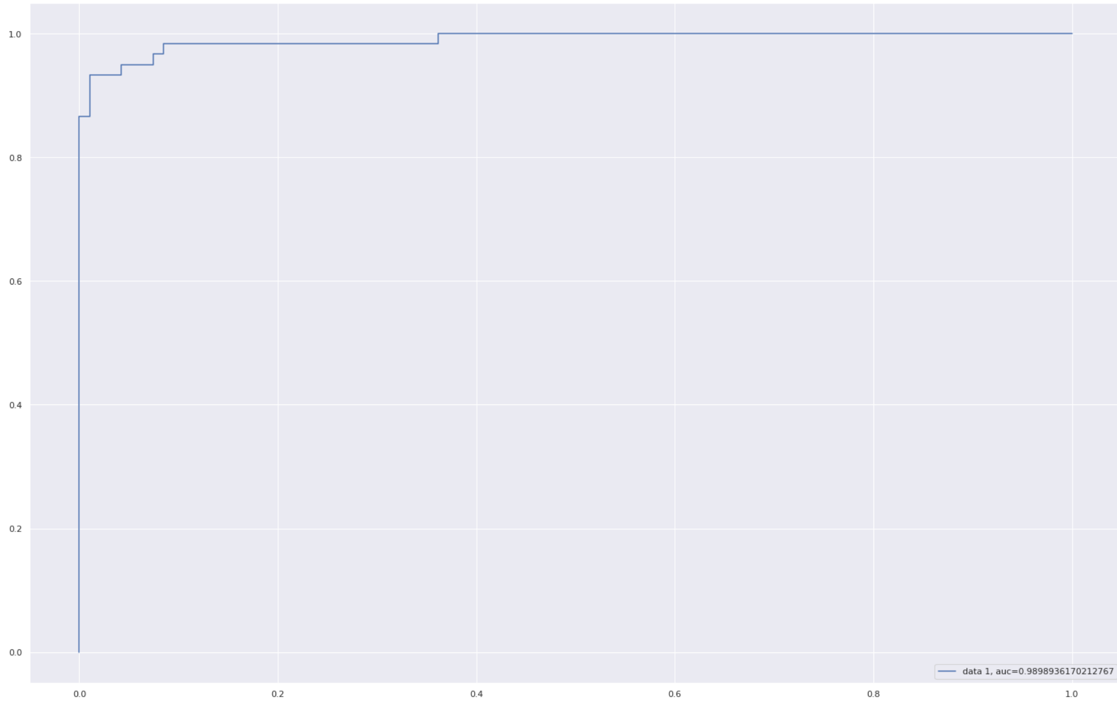
Training:

[727]:
```
y_pred_proba = logreg.predict_proba(x_train)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_train,  y_pred_proba)
auc = metrics.roc_auc_score(y_train, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



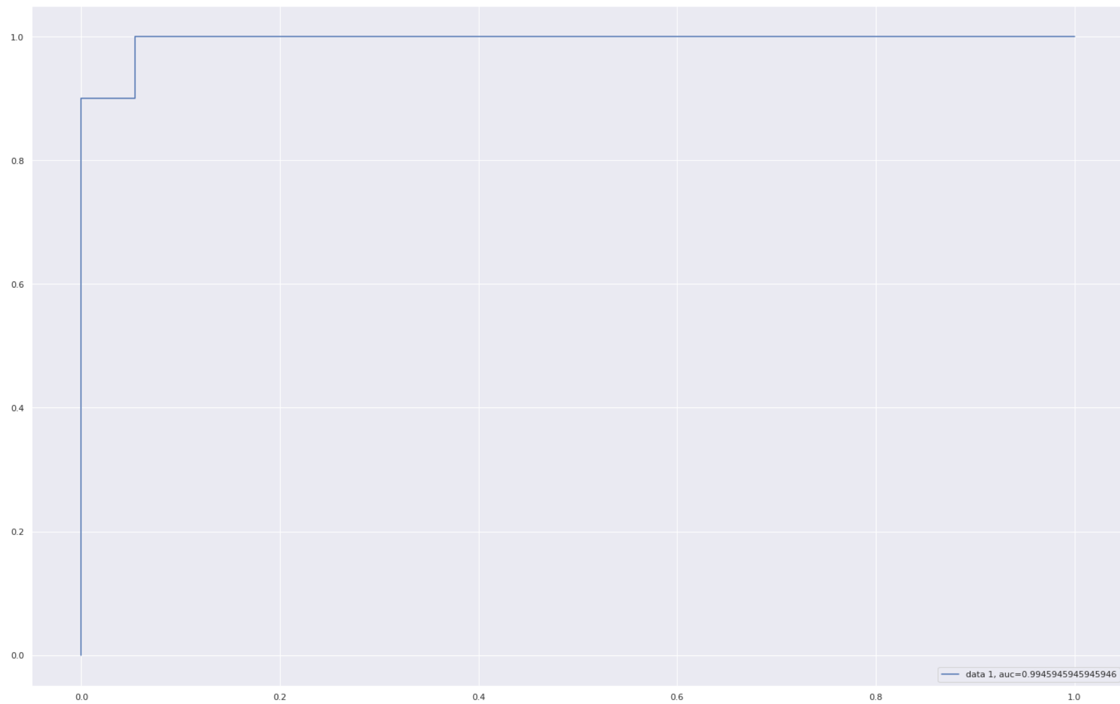
Validating:

[728]:
```
y_pred_proba = logreg.predict_proba(x_val)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_val,  y_pred_proba)
auc = metrics.roc_auc_score(y_val, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Testing:

[729]:
```
y_pred_proba = logreg.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Tenemos que decir que la curva ROC es un grafico en el que se ponen los valores positivos y falsos en cada eje, y como podemos ver dieron un muy buen accuracy en los tres datasets.

8 Tabla de accuracys:

[730]:
```
df_accuracys = pd.DataFrame({"Datos: ":["Training", "Validating", "Testing"],
                             "Registros": [accuracy_training,␣
 ↪accuracy_validating, accuracy_testing]})
df_accuracys
```

[730]:

	Datos:	Registros
0	Training	0.958101
1	Validating	0.961039
2	Testing	0.947368

Como podemos ver nuestro modelo dispone de muy poco error en el dataset de training y mucho menos error en el dataset de validacion, por lo que decimos que se encuentra en **balance**.

9 Predicciones:

[731]:
```
df.head(2)
```

[731]:
```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38           122.8     1001.0
1          1        20.57         17.77           132.9     1326.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017

   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  ...         25.38          17.33            184.6
1         0.1812  ...         24.99          23.41            158.8

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902

[2 rows x 26 columns]
```

[732]:
```
df.tail(2)
```

[732]:
```
     diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
567          1        20.60         29.33          140.10     1265.0
568          0         7.76         24.54           47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
567          0.11780           0.27700          0.3514                0.152
568          0.05263           0.04362          0.0000                0.000

     symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
567         0.2397  ...        25.740          39.42           184.60
568         0.1587  ...         9.456          30.37            59.16

     area_worst  smoothness_worst  compactness_worst  concavity_worst  \
567      1821.0           0.16500            0.86810           0.9387
568       268.6           0.08996            0.06444           0.0000

     concave points_worst  symmetry_worst  fractal_dimension_worst
567                 0.265          0.4087                  0.12400
568                 0.000          0.2871                  0.07039

[2 rows x 26 columns]
```

Necesito retro de esto, entiendo que es generando un dataframe de prueba, pero como le especifico a la funcion el rango de valor minimo o maximo que puede tomar cada feature????

[734]:
```
from sklearn.datasets import make_blobs
x_pred, _ = make_blobs(n_samples=5, centers=2, n_features=25, random_state=1)
y_pred = logreg.predict(x_pred)
for i in range(len(x_pred)):
  print("X=%s, Prediccion=%s" % (x_pred[i], y_pred[i]))
```

```
X=[-0.72845782  4.69207719 -9.11257134 -4.70774649 -5.81201403 -7.64029828
 -6.57288861 -2.60026731 -2.14022223  1.90796407 -0.0962929   5.88996541
 -7.30745134  6.11823492 -9.956714    3.56938727 -0.77773503  1.48943152
 -9.21446244 -6.34417423  6.84286602  9.59532625 -2.96950526  3.62412417
  7.32702498], Prediccion=0
X=[ 6.66207513e+00 -7.74857828e+00 -8.42609747e+00 -7.22692234e+00
  8.08342641e+00 -9.17740471e+00 -7.75986468e-01  9.20435790e+00
  4.76735928e-01  3.73579641e+00 -2.82080122e+00  4.48043019e+00
  7.22197876e+00 -9.49653324e+00  5.08070743e+00  1.03956020e+01
  5.19580765e+00 -3.70856875e+00  5.47546980e+00 -1.03703176e+01
 -3.30487462e-03  1.03588897e+01 -3.68635259e+00 -4.34464846e+00
 -7.53587330e+00], Prediccion=0
X=[-1.47299851  4.81654152 -9.79941278 -3.8343399  -7.73554447 -7.77566432
 -6.1529745  -1.95930155 -0.86573264  0.9614911  -1.99139466  3.0656596
 -5.48746065  7.6396888  -9.79610181  3.45294706 -2.2739048   1.8718286
 -7.63938979 -4.81346251  6.41838302  9.95881004 -4.82642828  4.01583475
  8.2683395 ], Prediccion=0
X=[ -2.34673261   3.56128423 -10.66895863  -3.96601315  -8.18219253
  -7.91881241  -4.6149936   -2.3467413   -2.25648607  -0.11129428
  -2.36326801   5.39684461  -5.86014725   6.92535308  -9.26133265
   5.50960534  -1.533745     1.79099968  -6.89209091  -6.39022006
   4.87237318   9.01588879  -3.94041067   4.4330755    8.36676646], Prediccion=0
X=[ 6.93843267 -8.56533428 -9.18628979 -7.97650893  7.87800946 -7.18690268
 -2.43736344  9.50833658 -0.64897771  3.79884677 -5.30545973  4.85143626
  7.10141398 -9.65885141  4.22772468 11.05097771  6.93041484 -6.24910202
  7.0217506  -6.30782912 -0.70411778  6.97264203 -3.26437171 -4.42541353
 -8.00334919], Prediccion=0
```

[735]:
```
 l = np.random.randint(low=0, high=52)
 x_pred = x_test[l:l+5]
```

[736]:
```
y_pred = logreg.predict(x_pred)
```

[737]:
```
y_pred
```

[737]:
```
array([1, 0, 0, 0, 1])
```

[739]:
```
pd.DataFrame({"Numero de prediccion":["1", "2", "3", "4", "5"],
              "Prediccion":y_pred})
```

[739]:

	Numero de prediccion	Prediccion
0	1	1

1	2	0
2	3	0
3	4	0
4	5	1

10 Conclusion:

Si bien el dataset que le pasamos al algoritmo implementado era un dataset muy limpio al cual no le tuvimos que hacer una gran cantidad de modificaciones o transformacion, al implementar el algoritmo mediante un framework nos dimos cuenta que esto es mas facil que hacerlo a mano, de hecho, en la entrega anterior yo implemente este algoritmo a mano, enlace. En esta entrega el hecho de implementarlo con un framewor facilita el estarle jugando a algunos parametros, por lo que hace esto mucho mas facil.