Universidad Autónoma de Baja California
Facultad de Ciencias Químicas e Ingeniería.

Meta 5.4

Clasificación

ALUMNO:
Urias Vega Juan Daniel
Zavala Roman Irvin Eduardo
Huertas Villegas Cesar

ASIGNATURA:
Inteligencia Artificial

DOCENTE:
Castro Rodríguez Juan Ramón

GRUPO:
561

Tijuana, B.C. México

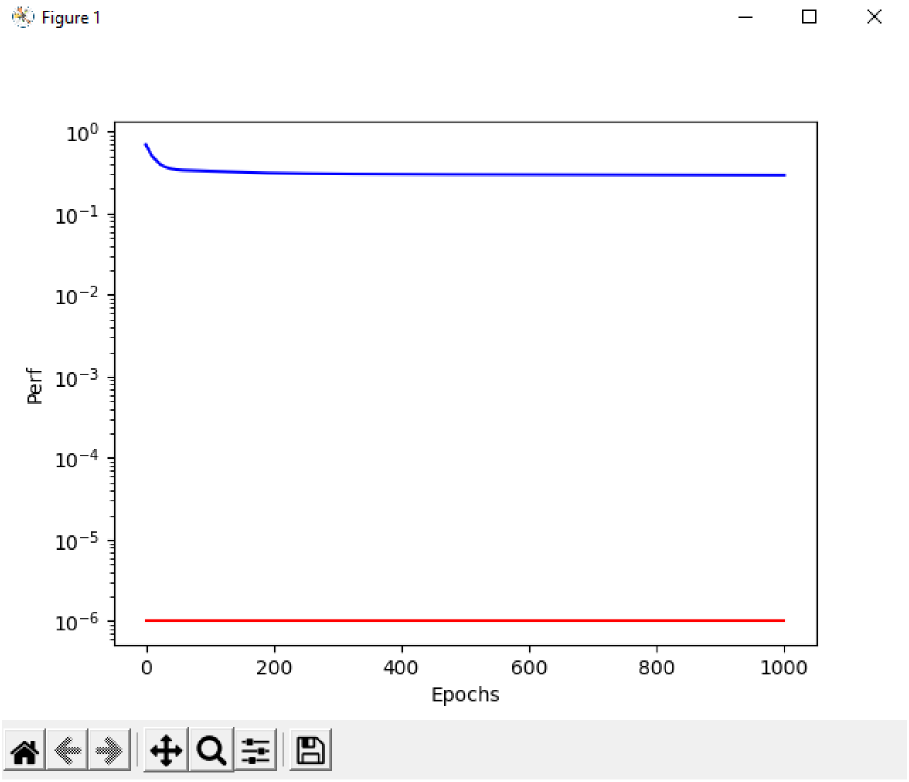
01 de junio del 2022

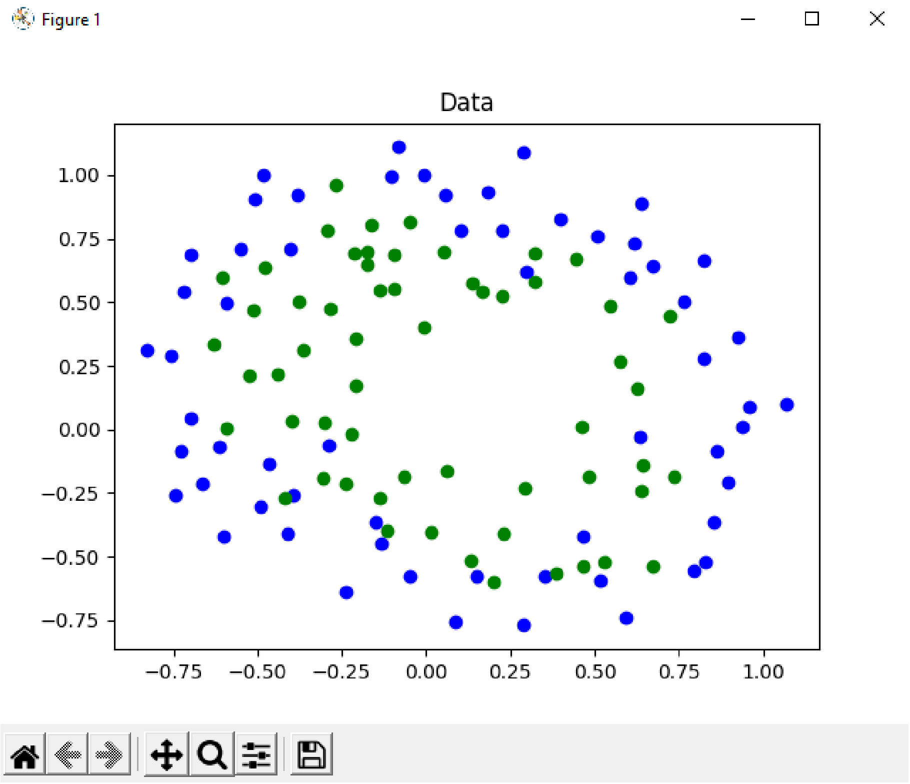
Desarrollo

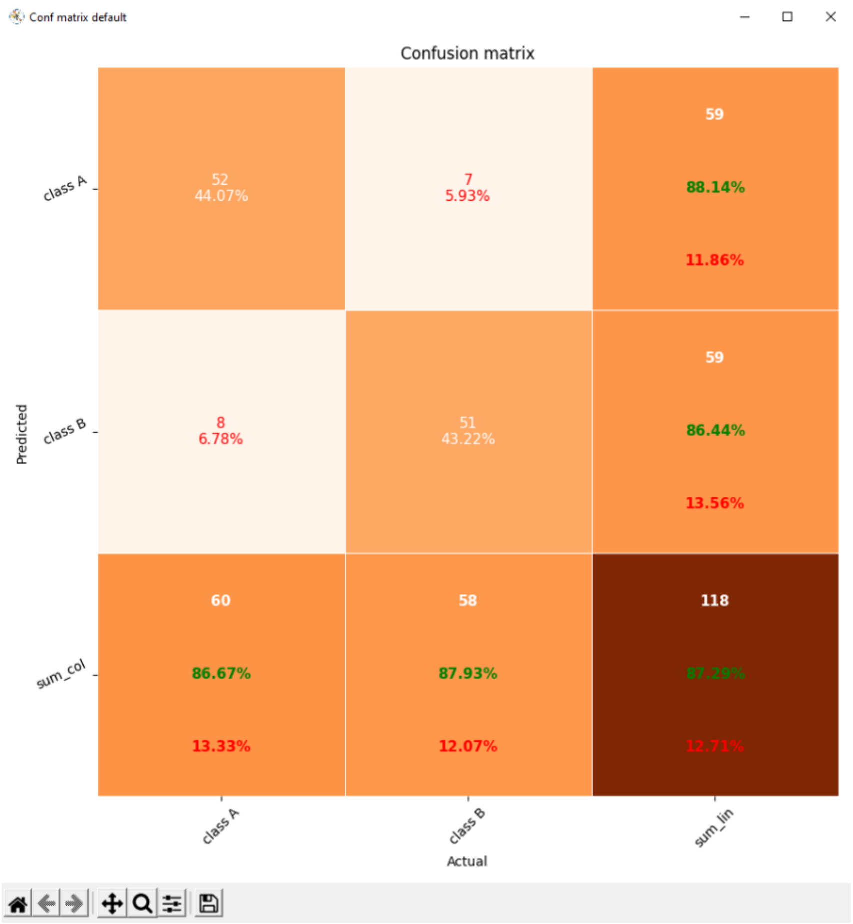
El objetivo de esa meta es el de diseñar e implementar un modelo de regresión logística de alto orden polinomio como método de aprendizaje supervisado con métricas de desempeño plotconfusion y plotroc.

Para el desarrollo de esta meta se utilizaron archivos dos archivos: syntethicclas_data1.dat y syntheticclass_data2.dat, ambos otorgados por el profesor. También se utilizaron librerías tales como Numpy para el manejo de arreglos, Scipy para el manejo de valores y Matplotlib para el manejo de interfaces gráficas para mostrar los resultados. Además, se hizo uso de un elemento denominado Matriz de diseño la cual fue elaborada en la meta anterior.

Opción 0:







Confusion matrix

	class A	class B	sum_lin
class A	52 44.07%	7 5.93%	59 88.14% 11.86%
class B	8 6.78%	51 43.22%	59 86.44% 13.56%
sum_col	60 86.67% 13.33%	58 87.93% 12.07%	118 87.29% 12.71%

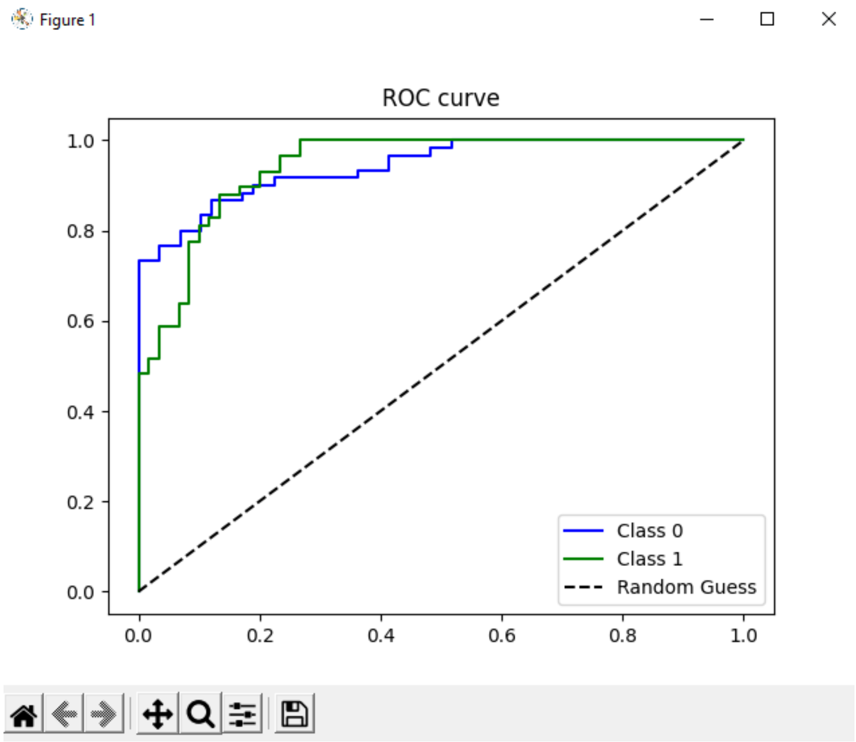


Figure 1
ROC curve
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Class 0
Class 1
Random Guess

Opción 1:

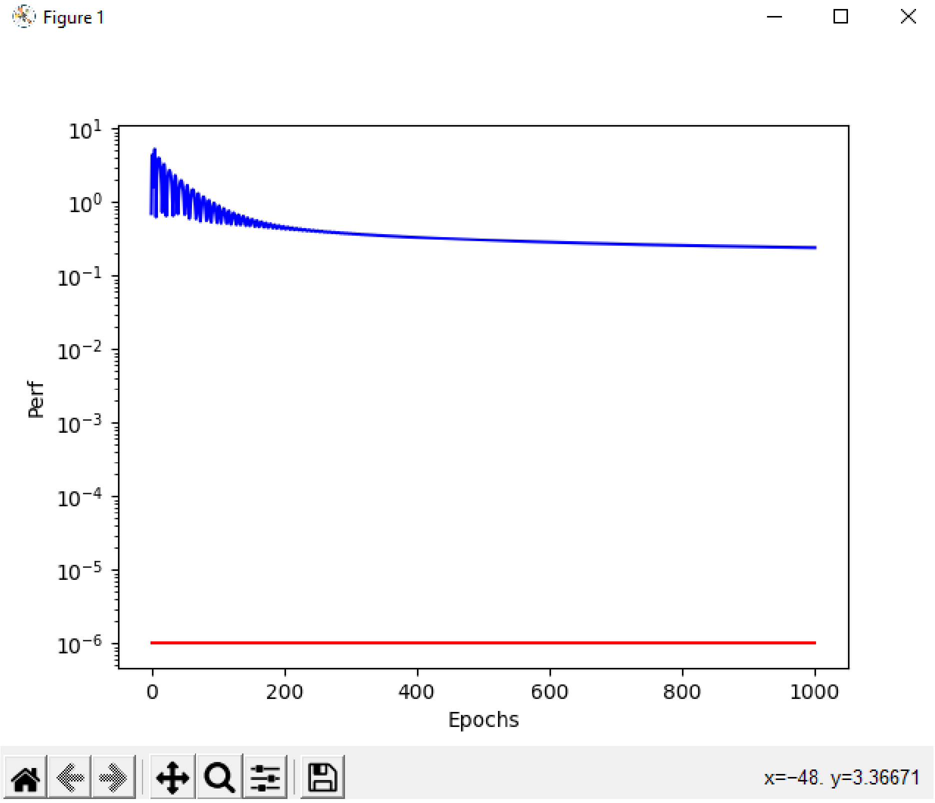
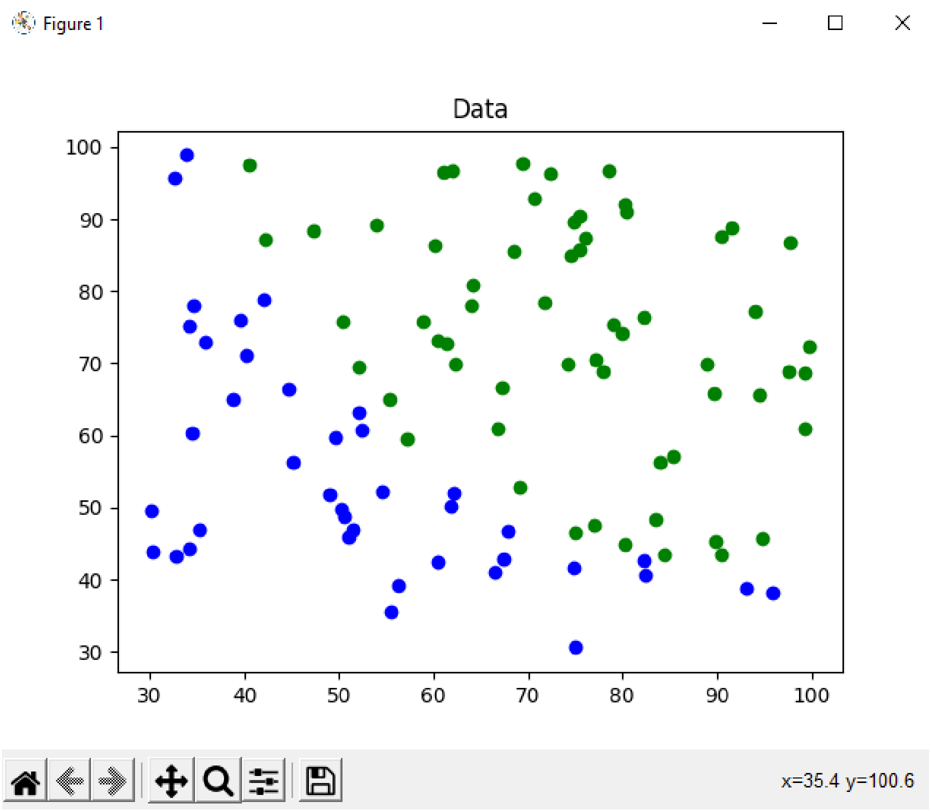
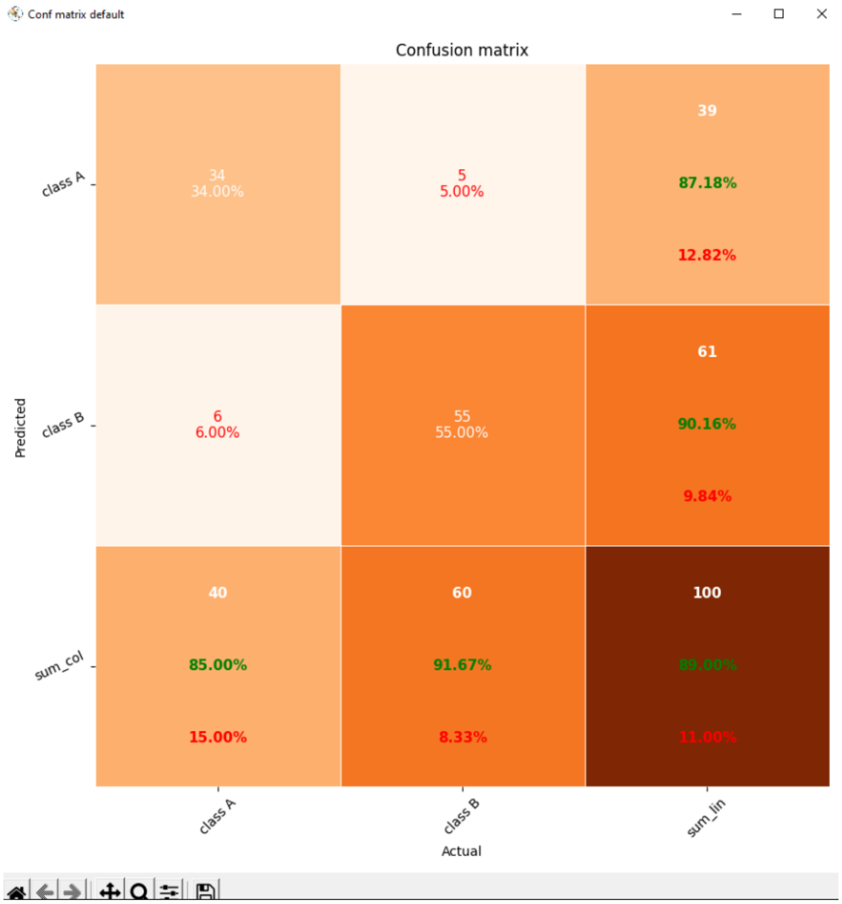


Figure 1
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
Perf
0
200
400
600
800
1000
Epochs
x=−48. y=3.36671





	class A	class B	sum_lin
class A	34 34.00%	5 5.00%	39 87.18% 12.82%
class B	6 6.00%	55 55.00%	61 90.16% 9.84%
sum_col	40 85.00% 15.00%	60 91.67% 8.33%	100 89.00% 11.00%

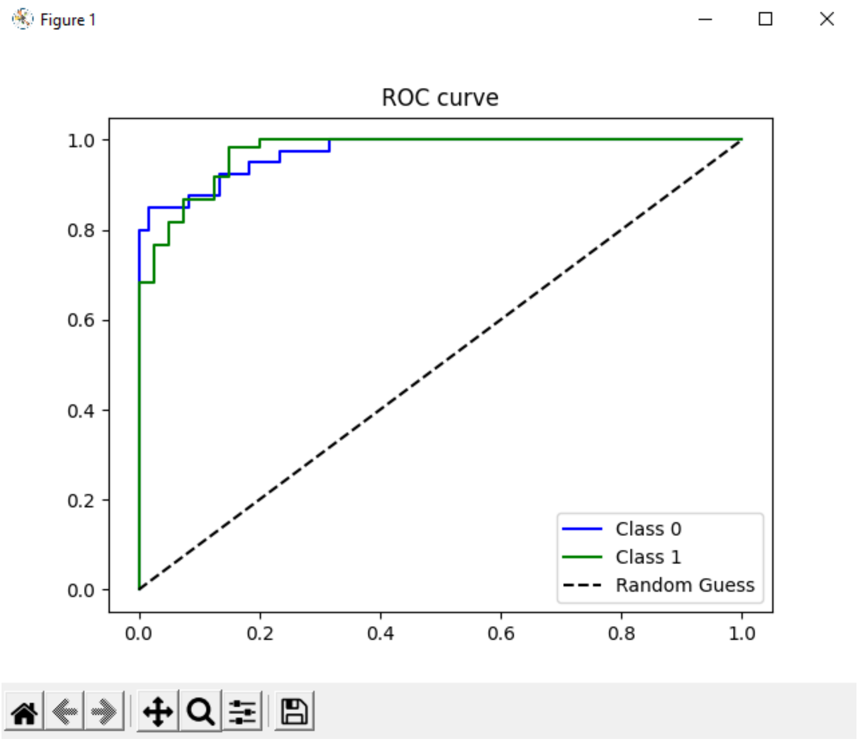
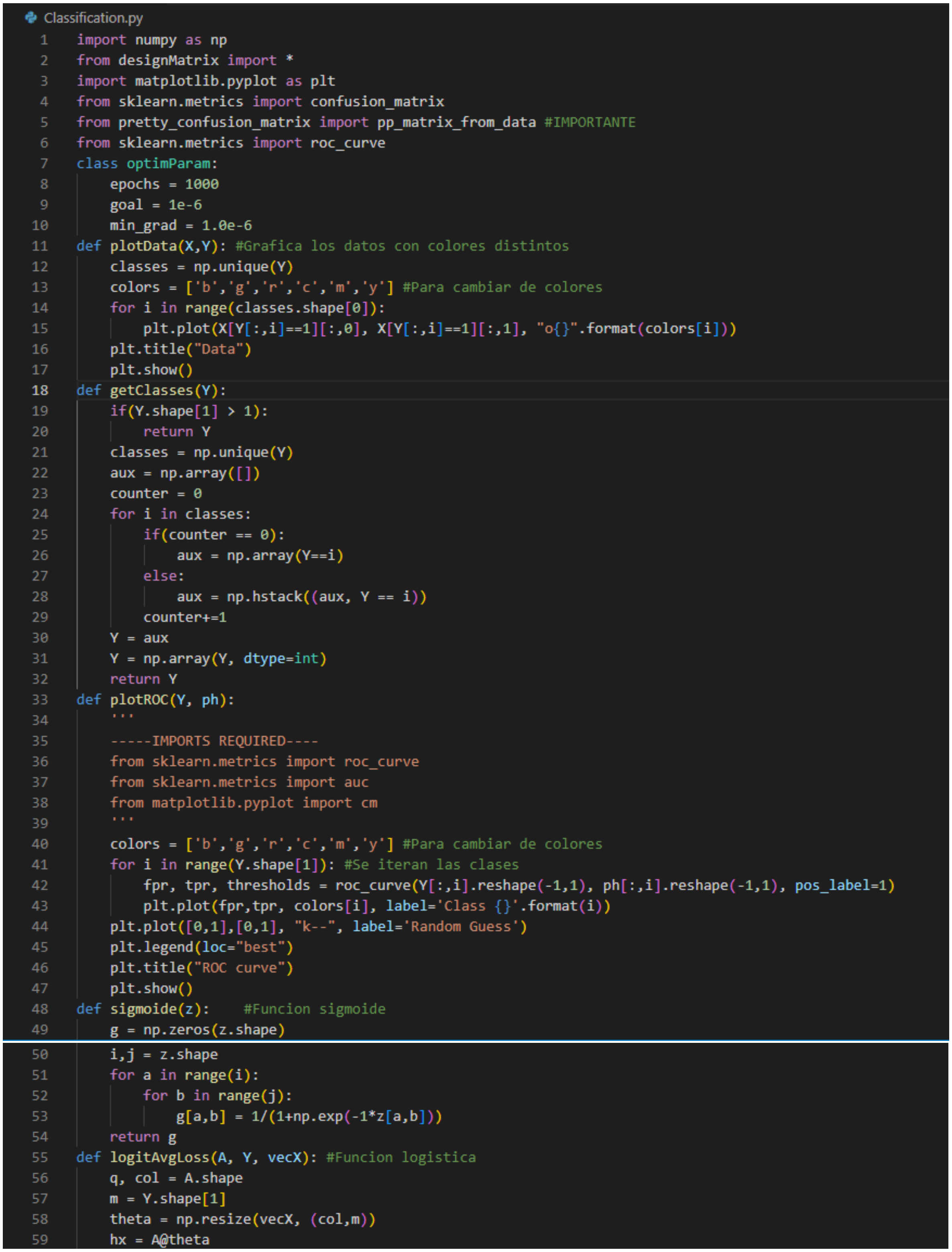


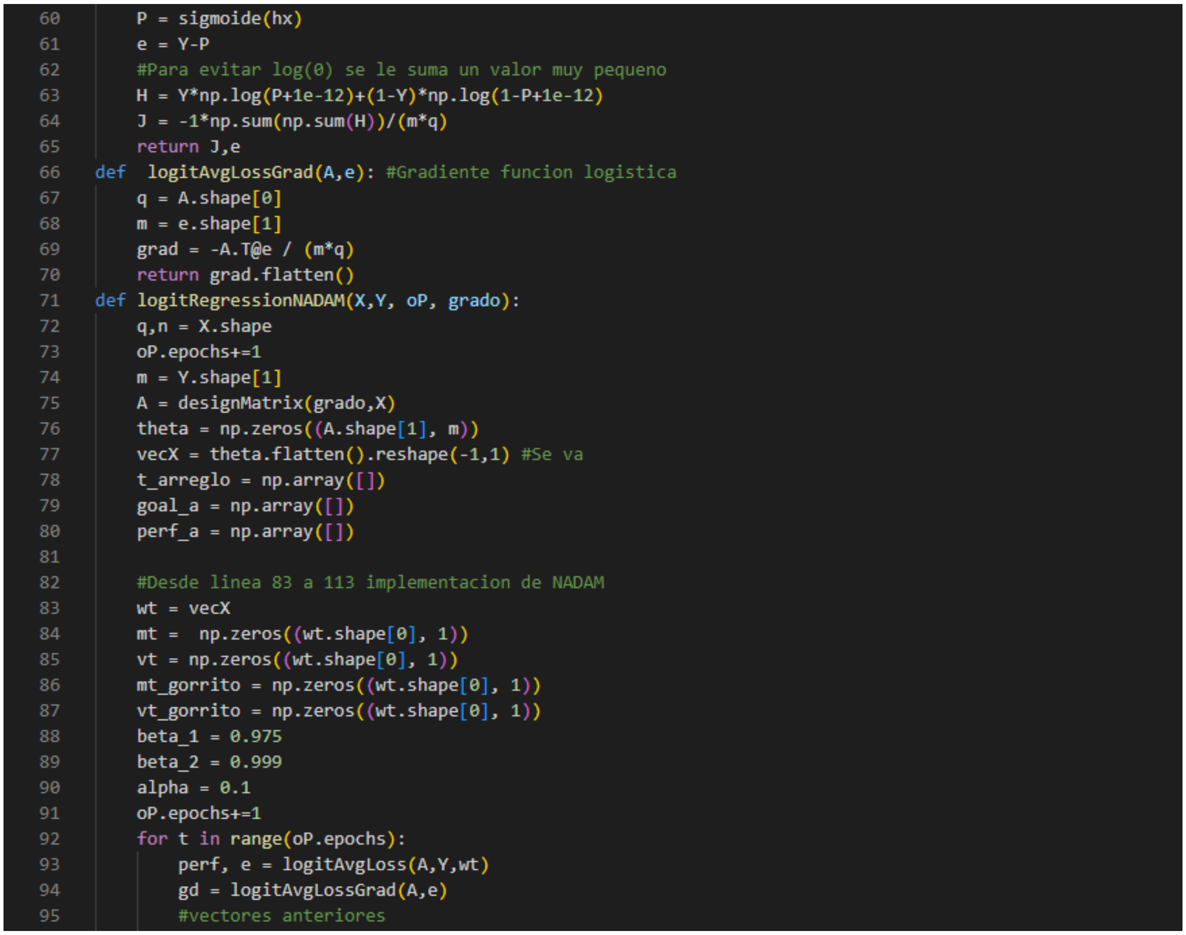
Figure 1
ROC curve
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Class 0
Class 1
Random Guess

Archivo: Classification.py

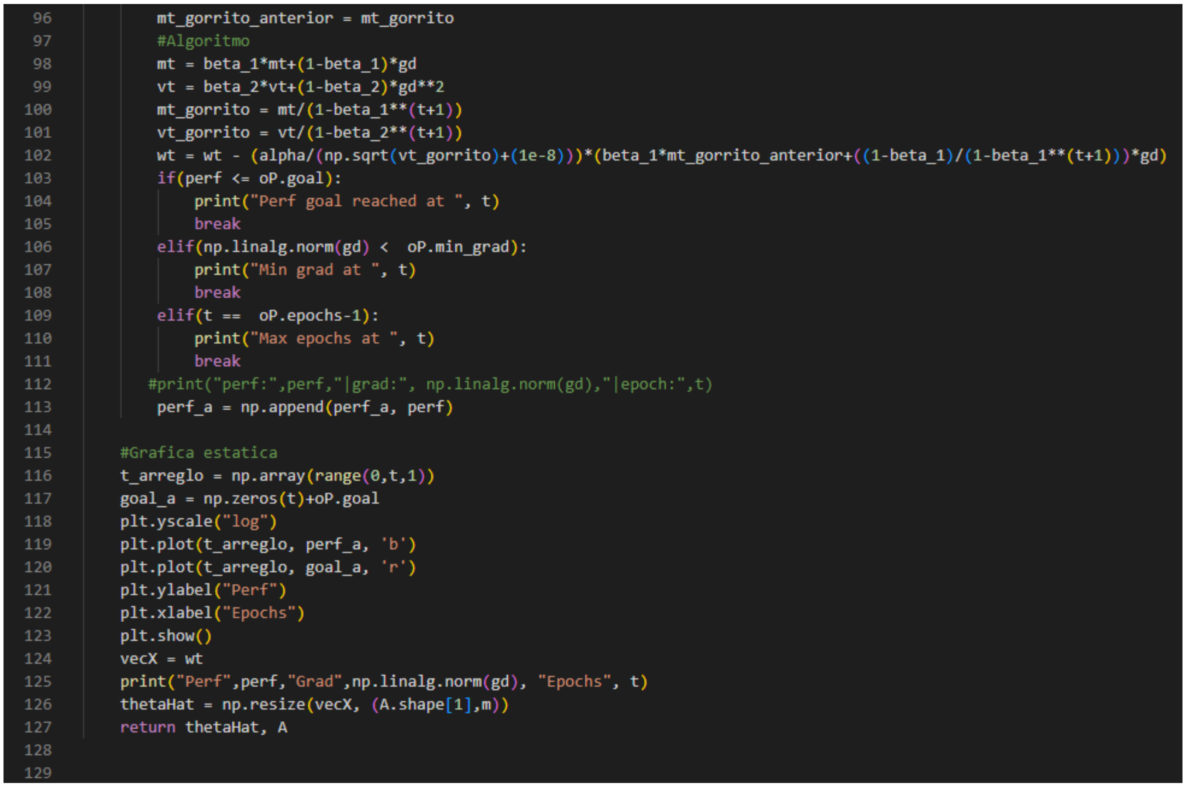


Classification.py

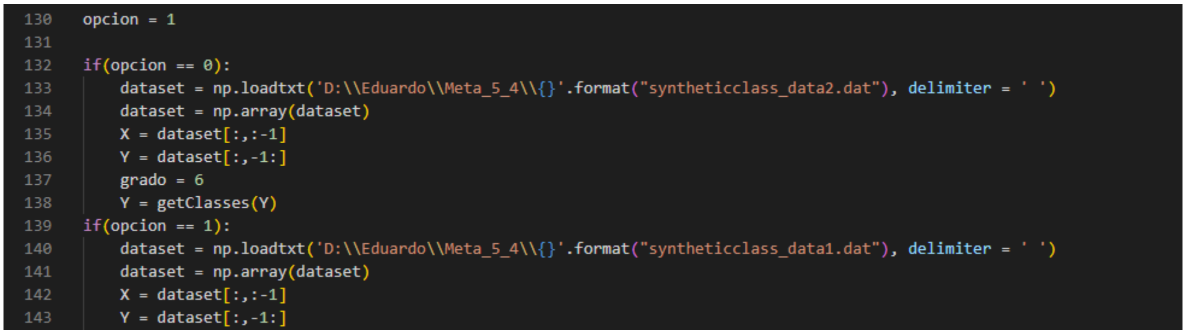
```python
import numpy as np
from designMatrix import *
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from pretty_confusion_matrix import pp_matrix_from_data #IMPORTANTE
from sklearn.metrics import roc_curve
class optimParam:
    epochs = 1000
    goal = 1e-6
    min_grad = 1.0e-6
def plotData(X,Y): #Grafica los datos con colores distintos
    classes = np.unique(Y)
    colors = ['b','g','r','c','m','y'] #Para cambiar de colores
    for i in range(classes.shape[0]):
        plt.plot(X[Y[:,i]==1][:,0], X[Y[:,i]==1][:,1], "o{}".format(colors[i]))
    plt.title("Data")
    plt.show()
def getClasses(Y):
    if(Y.shape[1] > 1):
        return Y
    classes = np.unique(Y)
    aux = np.array([])
    counter = 0
    for i in classes:
        if(counter == 0):
            aux = np.array(Y==i)
        else:
            aux = np.hstack((aux, Y == i))
        counter+=1
    Y = aux
    Y = np.array(Y, dtype=int)
    return Y
def plotROC(Y, ph):
    '''
    -----IMPORTS REQUIRED----
    from sklearn.metrics import roc_curve
    from sklearn.metrics import auc
    from matplotlib.pyplot import cm
    '''
    colors = ['b','g','r','c','m','y'] #Para cambiar de colores
    for i in range(Y.shape[1]): #Se iteran las clases
        fpr, tpr, thresholds = roc_curve(Y[:,i].reshape(-1,1), ph[:,i].reshape(-1,1), pos_label=1)
        plt.plot(fpr,tpr, colors[i], label='Class {}'.format(i))
    plt.plot([0,1],[0,1], "k--", label='Random Guess')
    plt.legend(loc="best")
    plt.title("ROC curve")
    plt.show()
def sigmoide(z):    #Funcion sigmoide
    g = np.zeros(z.shape)
    i,j = z.shape
    for a in range(i):
        for b in range(j):
            g[a,b] = 1/(1+np.exp(-1*z[a,b]))
    return g
def logitAvgLoss(A, Y, vecX): #Funcion logistica
    q, col = A.shape
    m = Y.shape[1]
    theta = np.resize(vecX, (col,m))
    hx = A@theta
```



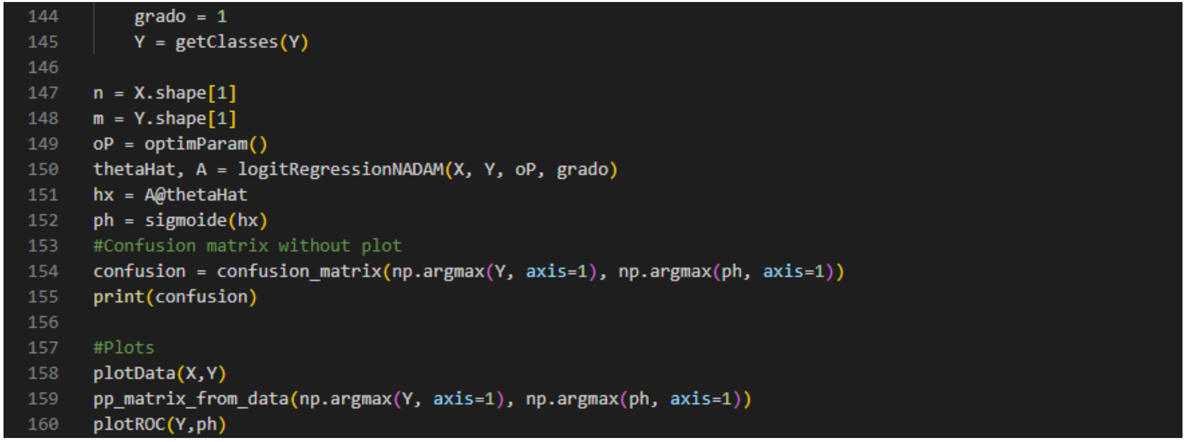
```
60      P = sigmoide(hx)
61      e = Y-P
62      #Para evitar log(0) se le suma un valor muy pequeno
63      H = Y*np.log(P+1e-12)+(1-Y)*np.log(1-P+1e-12)
64      J = -1*np.sum(np.sum(H))/(m*q)
65      return J,e
66  def  logitAvgLossGrad(A,e): #Gradiente funcion logistica
67      q = A.shape[0]
68      m = e.shape[1]
69      grad = -A.T@e / (m*q)
70      return grad.flatten()
71  def logitRegressionNADAM(X,Y, oP, grado):
72      q,n = X.shape
73      oP.epochs+=1
74      m = Y.shape[1]
75      A = designMatrix(grado,X)
76      theta = np.zeros((A.shape[1], m))
77      vecX = theta.flatten().reshape(-1,1) #Se va
78      t_arreglo = np.array([])
79      goal_a = np.array([])
80      perf_a = np.array([])
81
82      #Desde linea 83 a 113 implementacion de NADAM
83      wt = vecX
84      mt =  np.zeros((wt.shape[0], 1))
85      vt = np.zeros((wt.shape[0], 1))
86      mt_gorrito = np.zeros((wt.shape[0], 1))
87      vt_gorrito = np.zeros((wt.shape[0], 1))
88      beta_1 = 0.975
89      beta_2 = 0.999
90      alpha = 0.1
91      oP.epochs+=1
92      for t in range(oP.epochs):
93          perf, e = logitAvgLoss(A,Y,wt)
94          gd = logitAvgLossGrad(A,e)
95          #vectores anteriores
```



```
96          mt_gorrito_anterior = mt_gorrito
97          #Algoritmo
98          mt = beta_1*mt+(1-beta_1)*gd
99          vt = beta_2*vt+(1-beta_2)*gd**2
100         mt_gorrito = mt/(1-beta_1**(t+1))
101         vt_gorrito = vt/(1-beta_2**(t+1))
102         wt = wt - (alpha/(np.sqrt(vt_gorrito)+(1e-8)))*(beta_1*mt_gorrito_anterior+((1-beta_1)/(1-beta_1**(t+1)))*gd)
103         if(perf <= oP.goal):
104             print("Perf goal reached at ", t)
105             break
106         elif(np.linalg.norm(gd) <  oP.min_grad):
107             print("Min grad at ", t)
108             break
109         elif(t ==  oP.epochs-1):
110             print("Max epochs at ", t)
111             break
112        #print("perf:",perf,"|grad:", np.linalg.norm(gd),"|epoch:",t)
113         perf_a = np.append(perf_a, perf)
114
115     #Grafica estatica
116     t_arreglo = np.array(range(0,t,1))
117     goal_a = np.zeros(t)+oP.goal
118     plt.yscale("log")
119     plt.plot(t_arreglo, perf_a, 'b')
120     plt.plot(t_arreglo, goal_a, 'r')
121     plt.ylabel("Perf")
122     plt.xlabel("Epochs")
123     plt.show()
124     vecX = wt
125     print("Perf",perf,"Grad",np.linalg.norm(gd), "Epochs", t)
126     thetaHat = np.resize(vecX, (A.shape[1],m))
127     return thetaHat, A
128
129
```

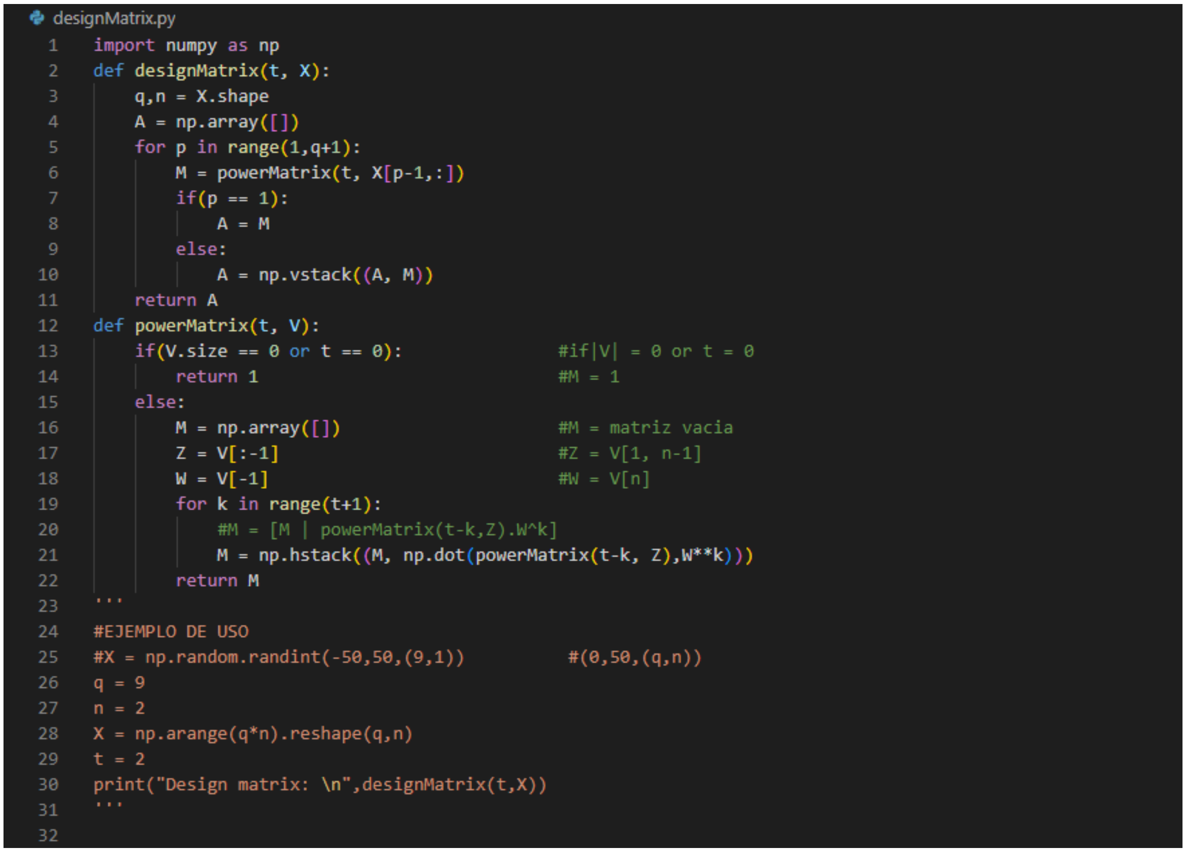


```python
opcion = 1

if(opcion == 0):
    dataset = np.loadtxt('D:\\Eduardo\\Meta_5_4\\{}'.format("syntheticclass_data2.dat"), delimiter = ' ')
    dataset = np.array(dataset)
    X = dataset[:,:-1]
    Y = dataset[:,-1:]
    grado = 6
    Y = getClasses(Y)
if(opcion == 1):
    dataset = np.loadtxt('D:\\Eduardo\\Meta_5_4\\{}'.format("syntheticclass_data1.dat"), delimiter = ' ')
    dataset = np.array(dataset)
    X = dataset[:,:-1]
    Y = dataset[:,-1:]
```



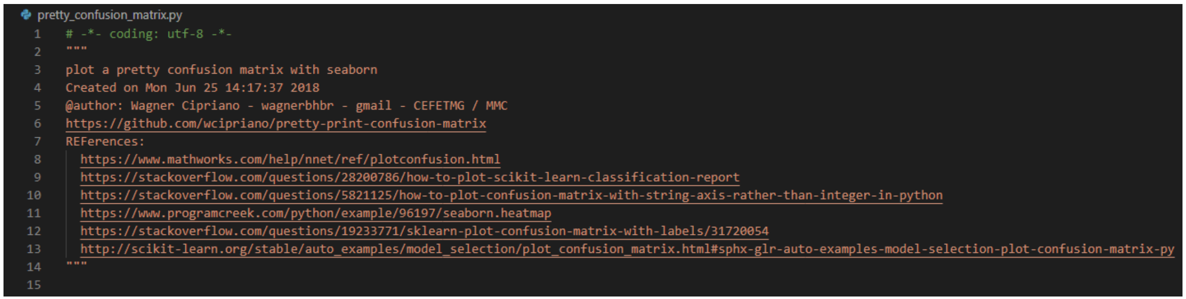
```python
    grado = 1
    Y = getClasses(Y)

n = X.shape[1]
m = Y.shape[1]
oP = optimParam()
thetaHat, A = logitRegressionNADAM(X, Y, oP, grado)
hx = A@thetaHat
ph = sigmoide(hx)
#Confusion matrix without plot
confusion = confusion_matrix(np.argmax(Y, axis=1), np.argmax(ph, axis=1))
print(confusion)

#Plots
plotData(X,Y)
pp_matrix_from_data(np.argmax(Y, axis=1), np.argmax(ph, axis=1))
plotROC(Y,ph)
```

Archivo: designMatrix.py

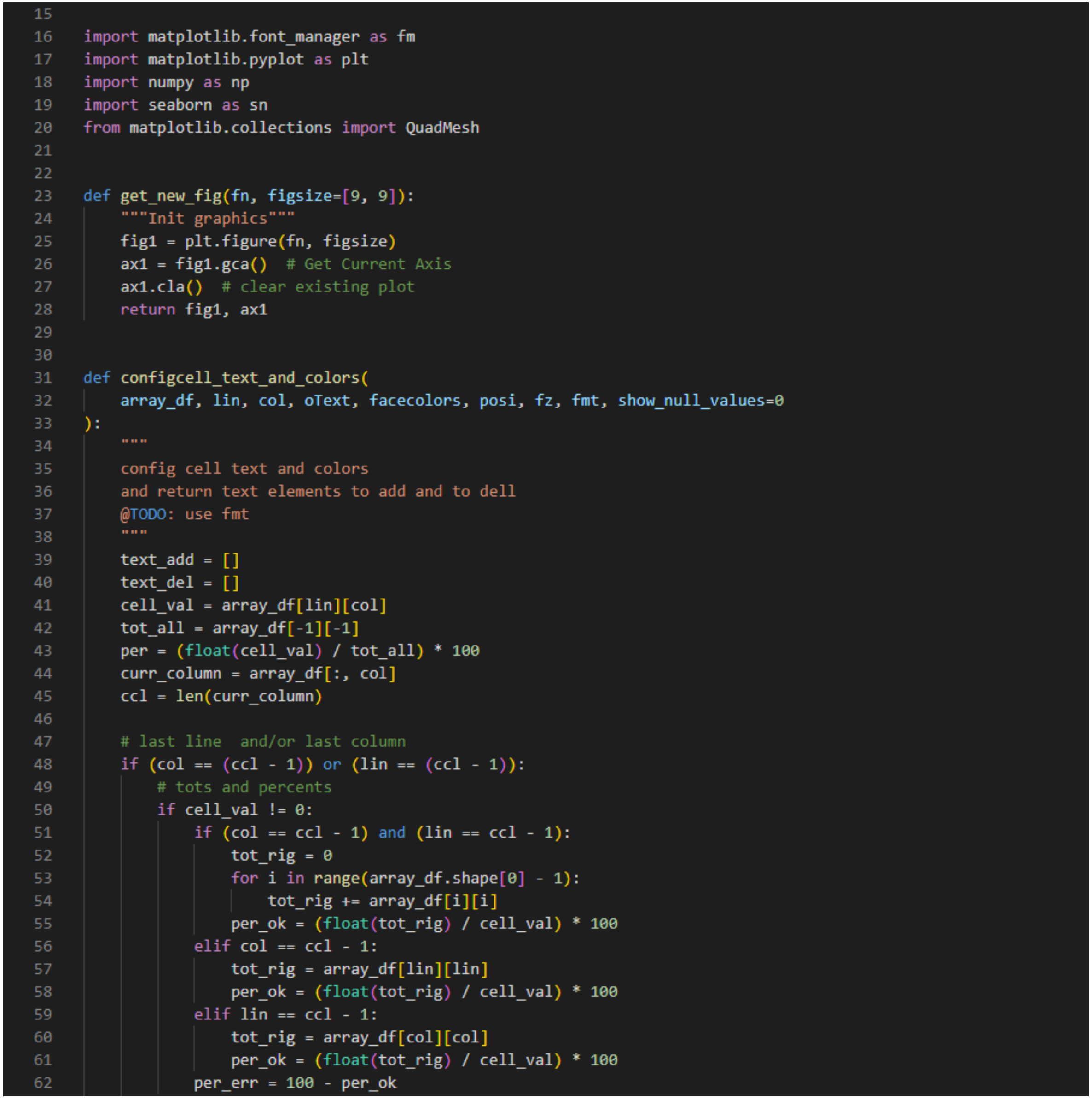


```python
import numpy as np
def designMatrix(t, X):
    q,n = X.shape
    A = np.array([])
    for p in range(1,q+1):
        M = powerMatrix(t, X[p-1,:])
        if(p == 1):
            A = M
        else:
            A = np.vstack((A, M))
    return A
def powerMatrix(t, V):
    if(V.size == 0 or t == 0):               #if|V| = 0 or t = 0
        return 1                             #M = 1
    else:
        M = np.array([])                     #M = matriz vacia
        Z = V[:-1]                           #Z = V[1, n-1]
        W = V[-1]                            #W = V[n]
        for k in range(t+1):
            #M = [M | powerMatrix(t-k,Z).W^k]
            M = np.hstack((M, np.dot(powerMatrix(t-k, Z),W**k)))
        return M
'''
#EJEMPLO DE USO
#X = np.random.randint(-50,50,(9,1))          #(0,50,(q,n))
q = 9
n = 2
X = np.arange(q*n).reshape(q,n)
t = 2
print("Design matrix: \n",designMatrix(t,X))
'''

```

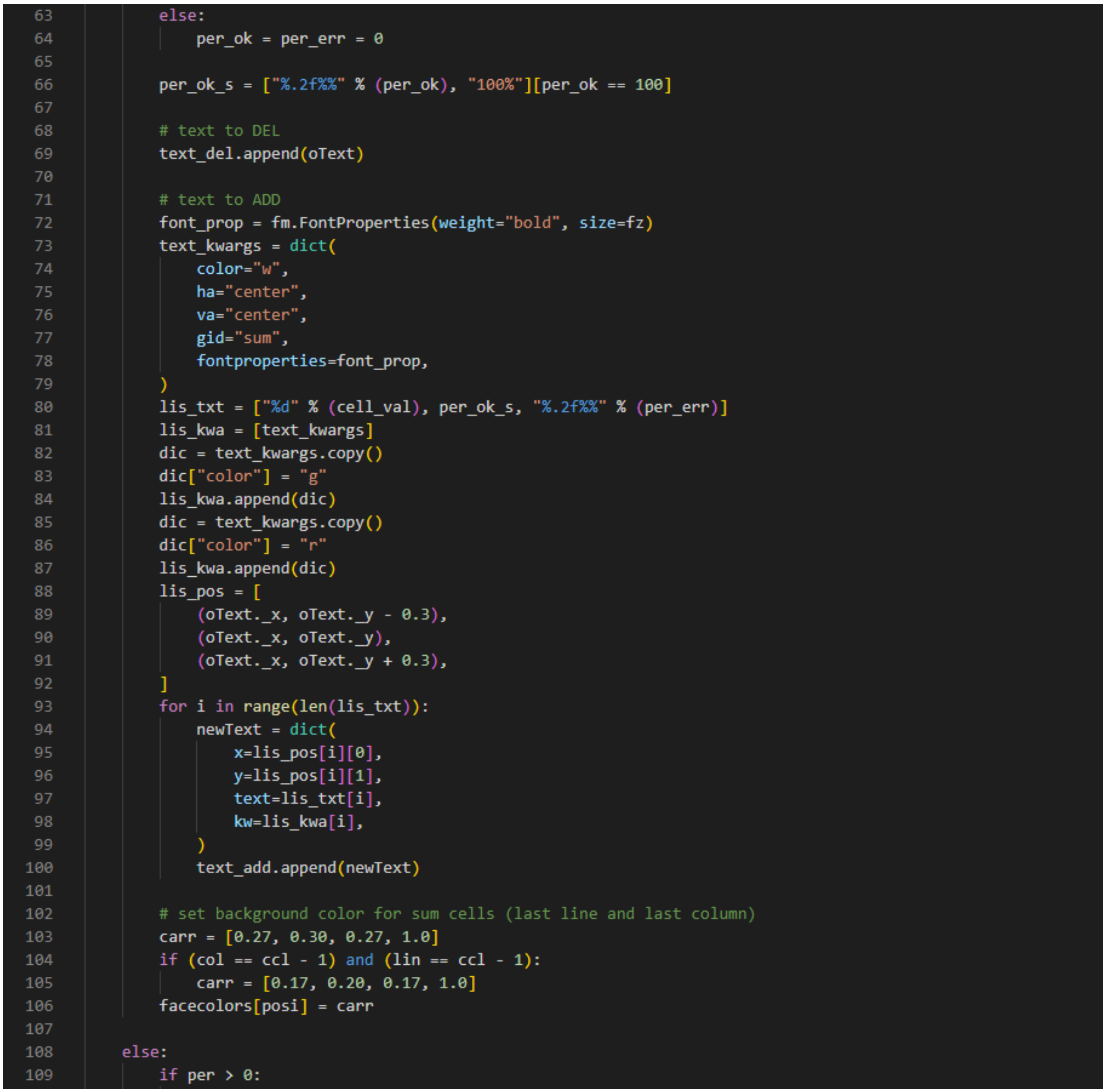
Archivo: Pretty_confusion_matrix.py



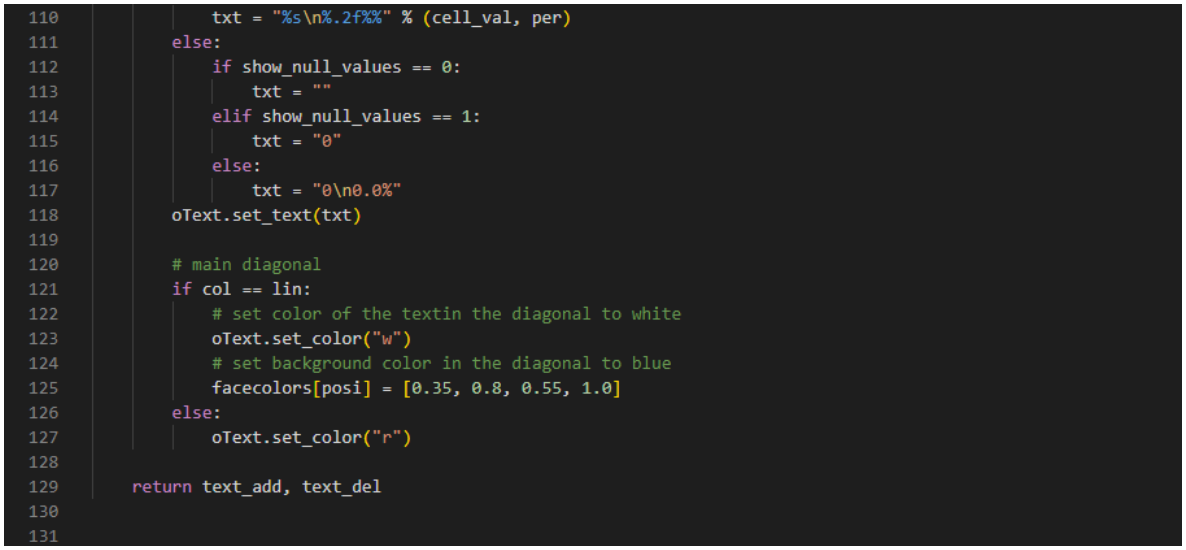
```python
# -*- coding: utf-8 -*-
"""
plot a pretty confusion matrix with seaborn
Created on Mon Jun 25 14:17:37 2018
@author: Wagner Cipriano - wagnerbhbr - gmail - CEFETMG / MMC
https://github.com/wcipriano/pretty-print-confusion-matrix
REFerences:
  https://www.mathworks.com/help/nnet/ref/plotconfusion.html
  https://stackoverflow.com/questions/28200786/how-to-plot-scikit-learn-classification-report
  https://stackoverflow.com/questions/5821125/how-to-plot-confusion-matrix-with-string-axis-rather-than-integer-in-python
  https://www.programcreek.com/python/example/96197/seaborn.heatmap
  https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels/31720054
  http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py
"""

```



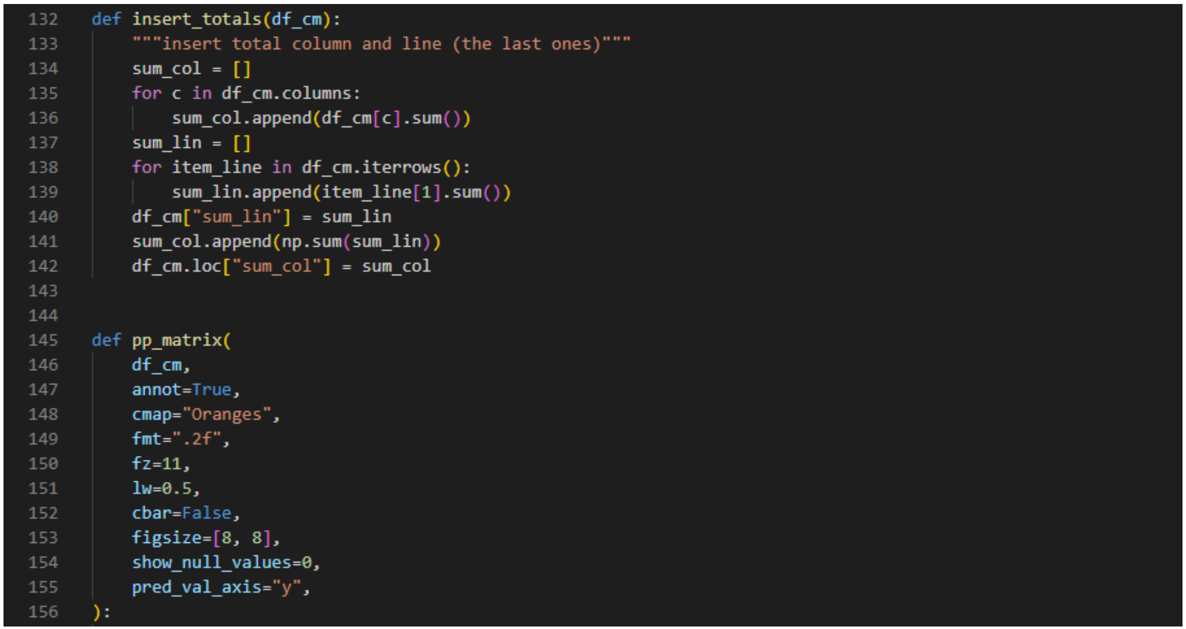
```python

import matplotlib.font_manager as fm
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sn
from matplotlib.collections import QuadMesh


def get_new_fig(fn, figsize=[9, 9]):
    """Init graphics"""
    fig1 = plt.figure(fn, figsize)
    ax1 = fig1.gca()  # Get Current Axis
    ax1.cla()  # clear existing plot
    return fig1, ax1


def configcell_text_and_colors(
    array_df, lin, col, oText, facecolors, posi, fz, fmt, show_null_values=0
):
    """
    config cell text and colors
    and return text elements to add and to dell
    @TODO: use fmt
    """
    text_add = []
    text_del = []
    cell_val = array_df[lin][col]
    tot_all = array_df[-1][-1]
    per = (float(cell_val) / tot_all) * 100
    curr_column = array_df[:, col]
    ccl = len(curr_column)

    # last line  and/or last column
    if (col == (ccl - 1)) or (lin == (ccl - 1)):
        # tots and percents
        if cell_val != 0:
            if (col == ccl - 1) and (lin == ccl - 1):
                tot_rig = 0
                for i in range(array_df.shape[0] - 1):
                    tot_rig += array_df[i][i]
                per_ok = (float(tot_rig) / cell_val) * 100
            elif col == ccl - 1:
                tot_rig = array_df[lin][lin]
                per_ok = (float(tot_rig) / cell_val) * 100
            elif lin == ccl - 1:
                tot_rig = array_df[col][col]
                per_ok = (float(tot_rig) / cell_val) * 100
            per_err = 100 - per_ok
```



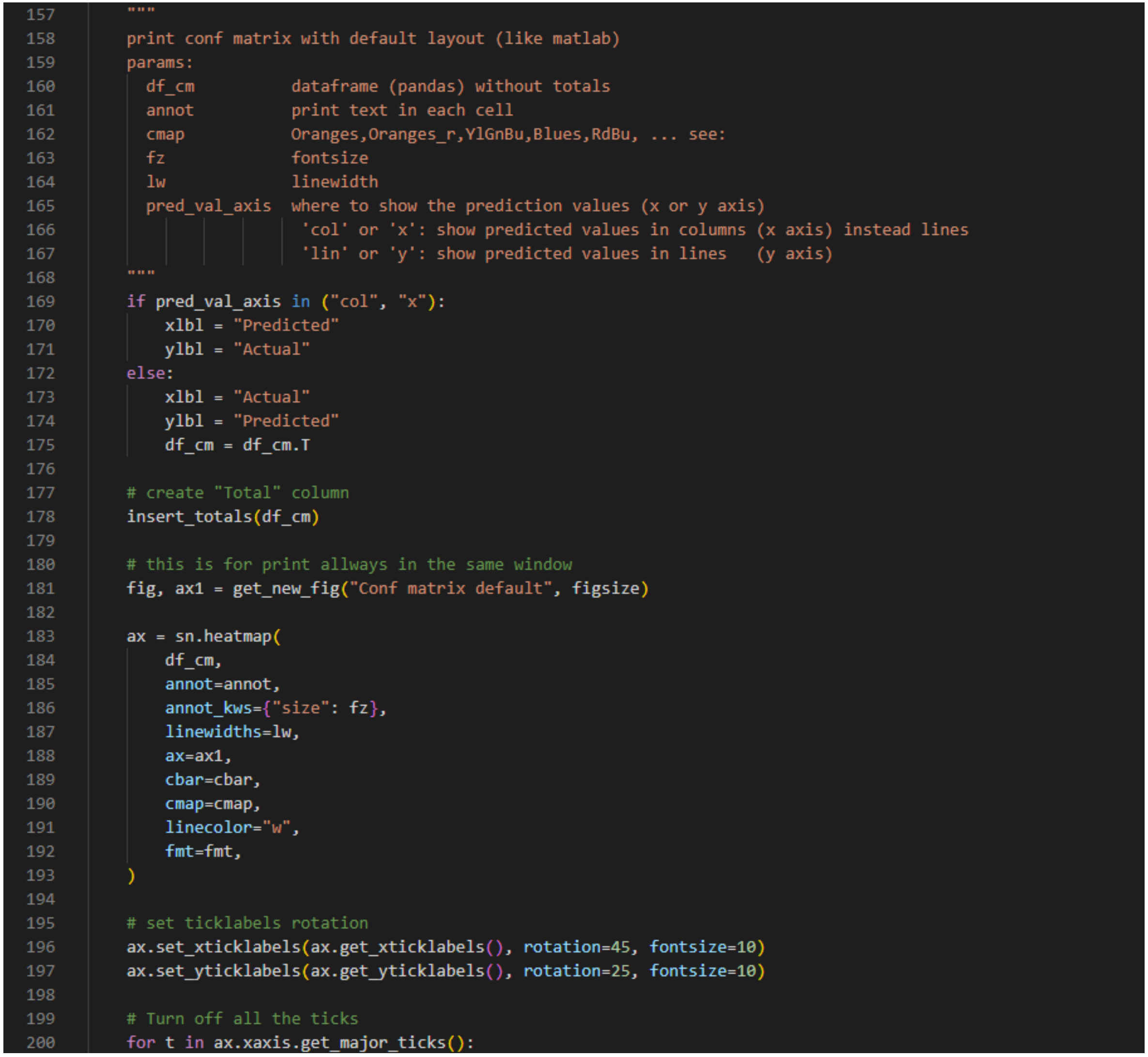
```python
        else:
            per_ok = per_err = 0

        per_ok_s = ["%.2f%%" % (per_ok), "100%"][per_ok == 100]

        # text to DEL
        text_del.append(oText)

        # text to ADD
        font_prop = fm.FontProperties(weight="bold", size=fz)
        text_kwargs = dict(
            color="w",
            ha="center",
            va="center",
            gid="sum",
            fontproperties=font_prop,
        )
        lis_txt = ["%d" % (cell_val), per_ok_s, "%.2f%%" % (per_err)]
        lis_kwa = [text_kwargs]
        dic = text_kwargs.copy()
        dic["color"] = "g"
        lis_kwa.append(dic)
        dic = text_kwargs.copy()
        dic["color"] = "r"
        lis_kwa.append(dic)
        lis_pos = [
            (oText._x, oText._y - 0.3),
            (oText._x, oText._y),
            (oText._x, oText._y + 0.3),
        ]
        for i in range(len(lis_txt)):
            newText = dict(
                x=lis_pos[i][0],
                y=lis_pos[i][1],
                text=lis_txt[i],
                kw=lis_kwa[i],
            )
            text_add.append(newText)

        # set background color for sum cells (last line and last column)
        carr = [0.27, 0.30, 0.27, 1.0]
        if (col == ccl - 1) and (lin == ccl - 1):
            carr = [0.17, 0.20, 0.17, 1.0]
        facecolors[posi] = carr

    else:
        if per > 0:
```



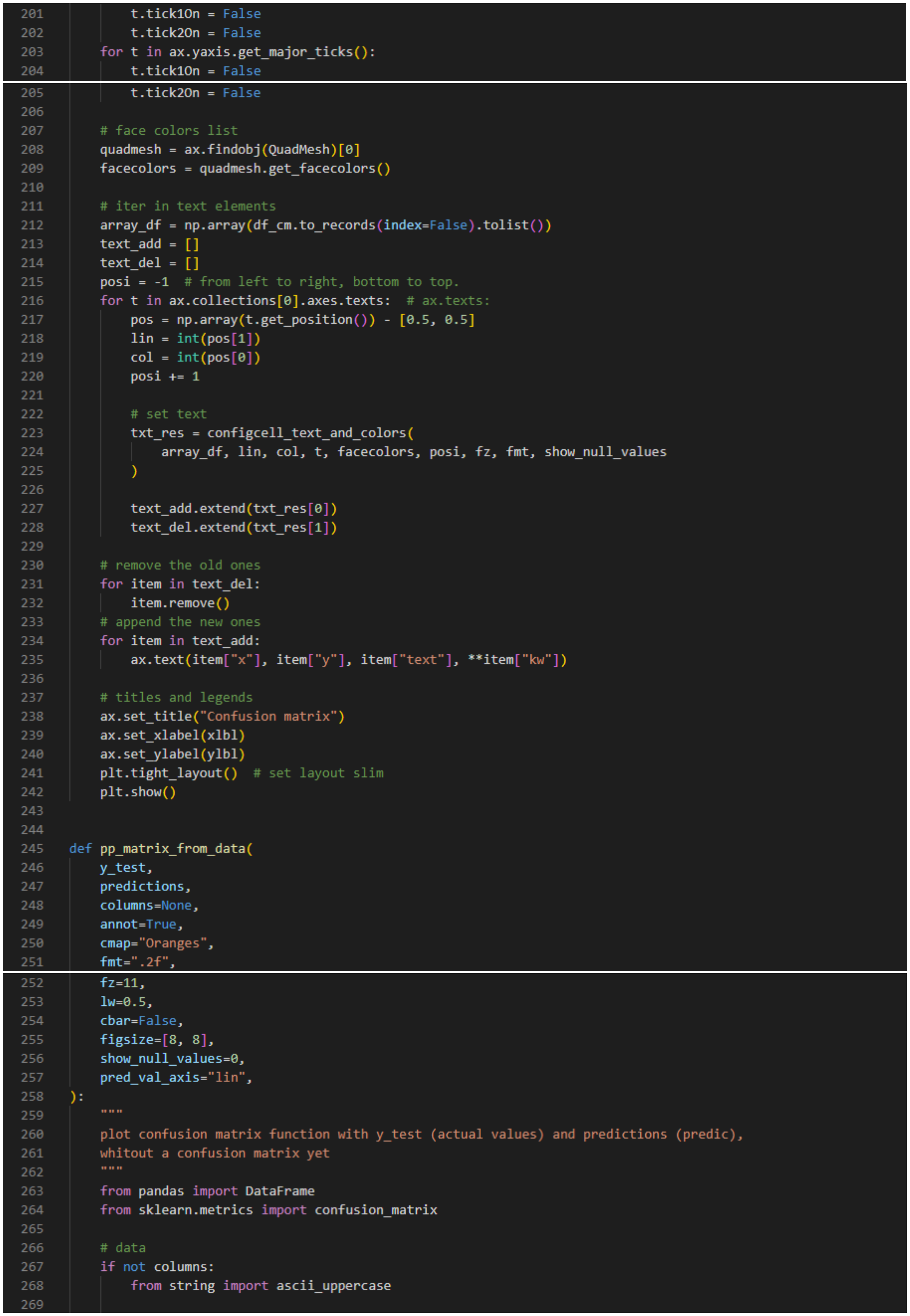
```python
            txt = "%s\n%.2f%%" % (cell_val, per)
        else:
            if show_null_values == 0:
                txt = ""
            elif show_null_values == 1:
                txt = "0"
            else:
                txt = "0\n0.0%"
        oText.set_text(txt)

        # main diagonal
        if col == lin:
            # set color of the textin the diagonal to white
            oText.set_color("w")
            # set background color in the diagonal to blue
            facecolors[posi] = [0.35, 0.8, 0.55, 1.0]
        else:
            oText.set_color("r")

    return text_add, text_del
```



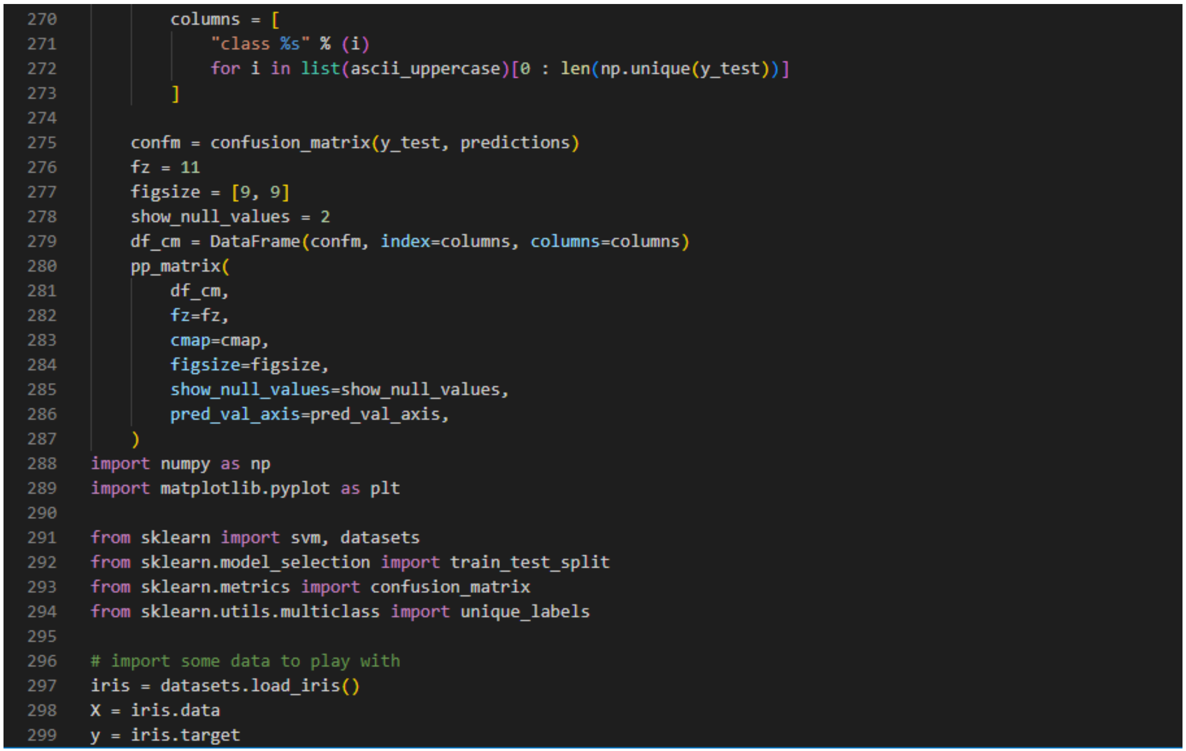
```python
def insert_totals(df_cm):
    """insert total column and line (the last ones)"""
    sum_col = []
    for c in df_cm.columns:
        sum_col.append(df_cm[c].sum())
    sum_lin = []
    for item_line in df_cm.iterrows():
        sum_lin.append(item_line[1].sum())
    df_cm["sum_lin"] = sum_lin
    sum_col.append(np.sum(sum_lin))
    df_cm.loc["sum_col"] = sum_col


def pp_matrix(
    df_cm,
    annot=True,
    cmap="Oranges",
    fmt=".2f",
    fz=11,
    lw=0.5,
    cbar=False,
    figsize=[8, 8],
    show_null_values=0,
    pred_val_axis="y",
):
```



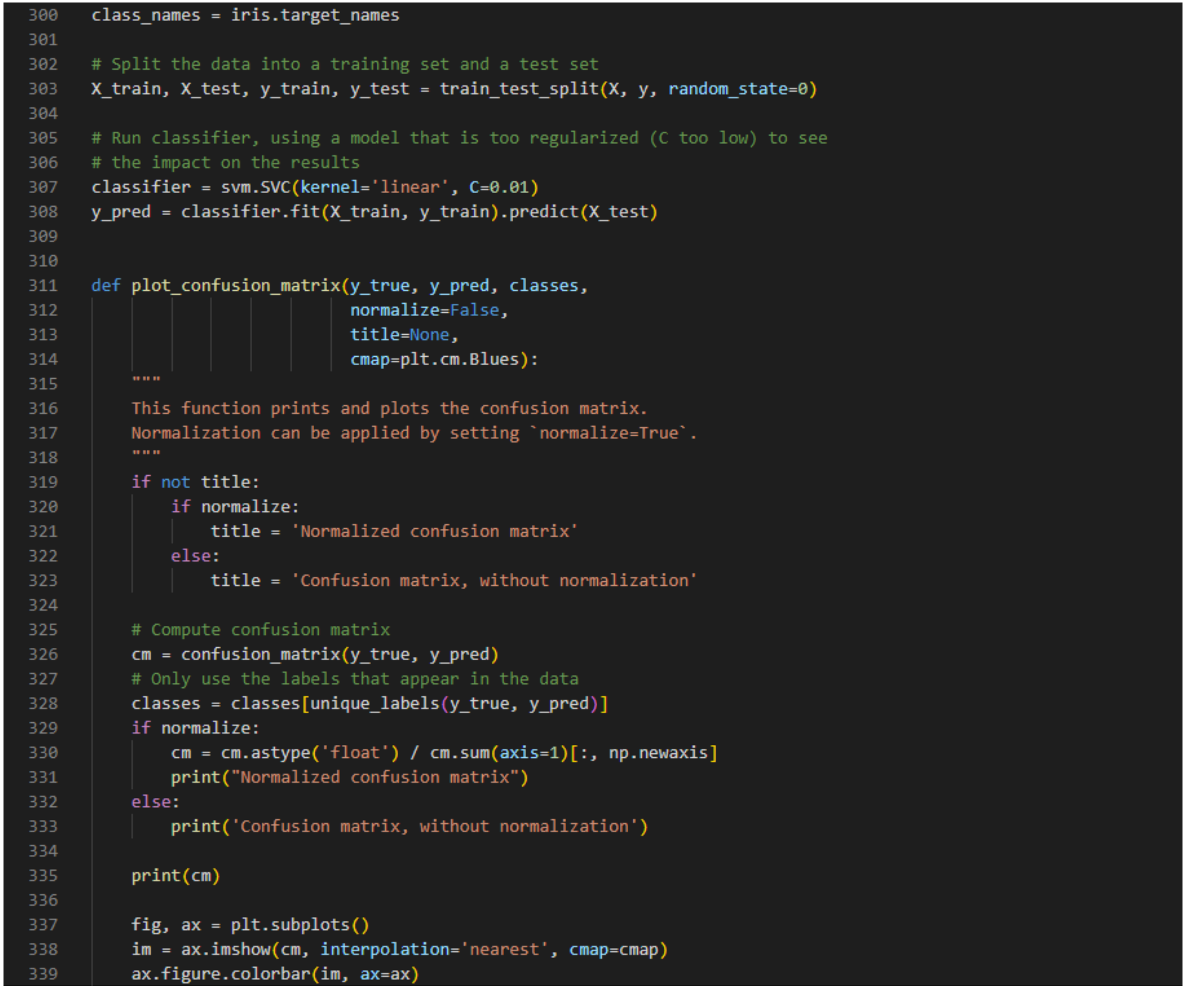
```python
    """
    print conf matrix with default layout (like matlab)
    params:
      df_cm          dataframe (pandas) without totals
      annot          print text in each cell
      cmap           Oranges,Oranges_r,YlGnBu,Blues,RdBu, ... see:
      fz             fontsize
      lw             linewidth
      pred_val_axis  where to show the prediction values (x or y axis)
                      'col' or 'x': show predicted values in columns (x axis) instead lines
                      'lin' or 'y': show predicted values in lines   (y axis)
    """
    if pred_val_axis in ("col", "x"):
        xlbl = "Predicted"
        ylbl = "Actual"
    else:
        xlbl = "Actual"
        ylbl = "Predicted"
        df_cm = df_cm.T

    # create "Total" column
    insert_totals(df_cm)

    # this is for print allways in the same window
    fig, ax1 = get_new_fig("Conf matrix default", figsize)

    ax = sn.heatmap(
        df_cm,
        annot=annot,
        annot_kws={"size": fz},
        linewidths=lw,
        ax=ax1,
        cbar=cbar,
        cmap=cmap,
        linecolor="w",
        fmt=fmt,
    )

    # set ticklabels rotation
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45, fontsize=10)
    ax.set_yticklabels(ax.get_yticklabels(), rotation=25, fontsize=10)

    # Turn off all the ticks
    for t in ax.xaxis.get_major_ticks():
```



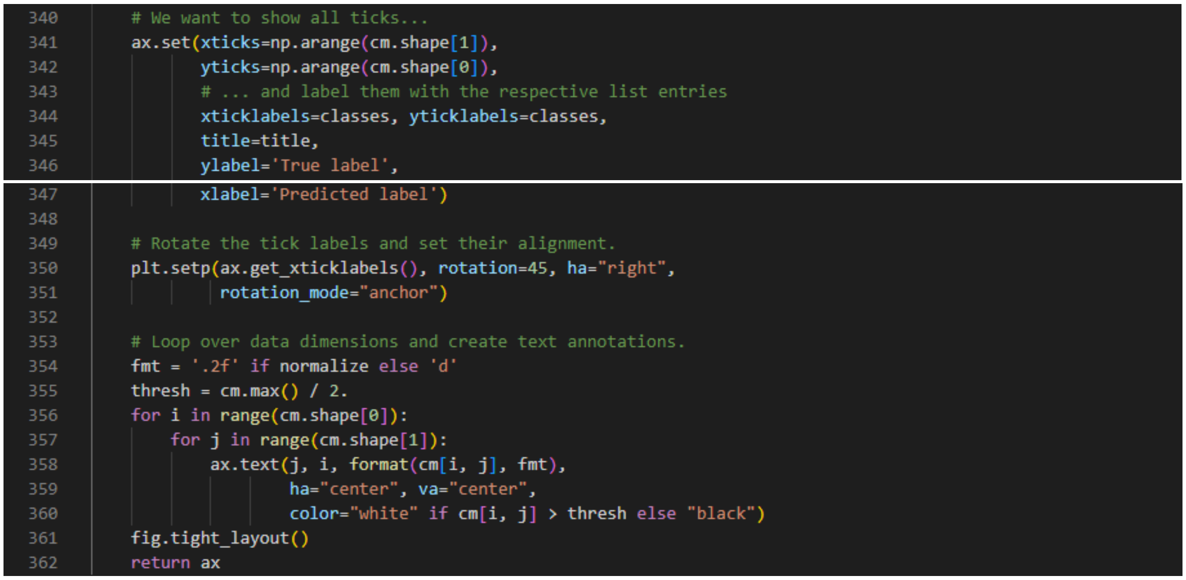
```python
        t.tick1On = False
        t.tick2On = False
    for t in ax.yaxis.get_major_ticks():
        t.tick1On = False
        t.tick2On = False

    # face colors list
    quadmesh = ax.findobj(QuadMesh)[0]
    facecolors = quadmesh.get_facecolors()

    # iter in text elements
    array_df = np.array(df_cm.to_records(index=False).tolist())
    text_add = []
    text_del = []
    posi = -1  # from left to right, bottom to top.
    for t in ax.collections[0].axes.texts:  # ax.texts:
        pos = np.array(t.get_position()) - [0.5, 0.5]
        lin = int(pos[1])
        col = int(pos[0])
        posi += 1

        # set text
        txt_res = configcell_text_and_colors(
            array_df, lin, col, t, facecolors, posi, fz, fmt, show_null_values
        )

        text_add.extend(txt_res[0])
        text_del.extend(txt_res[1])

    # remove the old ones
    for item in text_del:
        item.remove()
    # append the new ones
    for item in text_add:
        ax.text(item["x"], item["y"], item["text"], **item["kw"])

    # titles and legends
    ax.set_title("Confusion matrix")
    ax.set_xlabel(xlbl)
    ax.set_ylabel(ylbl)
    plt.tight_layout()  # set layout slim
    plt.show()


def pp_matrix_from_data(
    y_test,
    predictions,
    columns=None,
    annot=True,
    cmap="Oranges",
    fmt=".2f",
    fz=11,
    lw=0.5,
    cbar=False,
    figsize=[8, 8],
    show_null_values=0,
    pred_val_axis="lin",
):
    """
    plot confusion matrix function with y_test (actual values) and predictions (predic),
    whitout a confusion matrix yet
    """
    from pandas import DataFrame
    from sklearn.metrics import confusion_matrix

    # data
    if not columns:
        from string import ascii_uppercase

```



```python
        columns = [
            "class %s" % (i)
            for i in list(ascii_uppercase)[0 : len(np.unique(y_test))]
        ]

    confm = confusion_matrix(y_test, predictions)
    fz = 11
    figsize = [9, 9]
    show_null_values = 2
    df_cm = DataFrame(confm, index=columns, columns=columns)
    pp_matrix(
        df_cm,
        fz=fz,
        cmap=cmap,
        figsize=figsize,
        show_null_values=show_null_values,
        pred_val_axis=pred_val_axis,
    )
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels

# import some data to play with
iris = datasets.load_iris()
X = iris.data
y = iris.target
```



```python
class_names = iris.target_names

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Run classifier, using a model that is too regularized (C too low) to see
# the impact on the results
classifier = svm.SVC(kernel='linear', C=0.01)
y_pred = classifier.fit(X_train, y_train).predict(X_test)


def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
```



```python
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```