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

Meta 5.5

Aplicaciones: Inteligencia Artificial

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

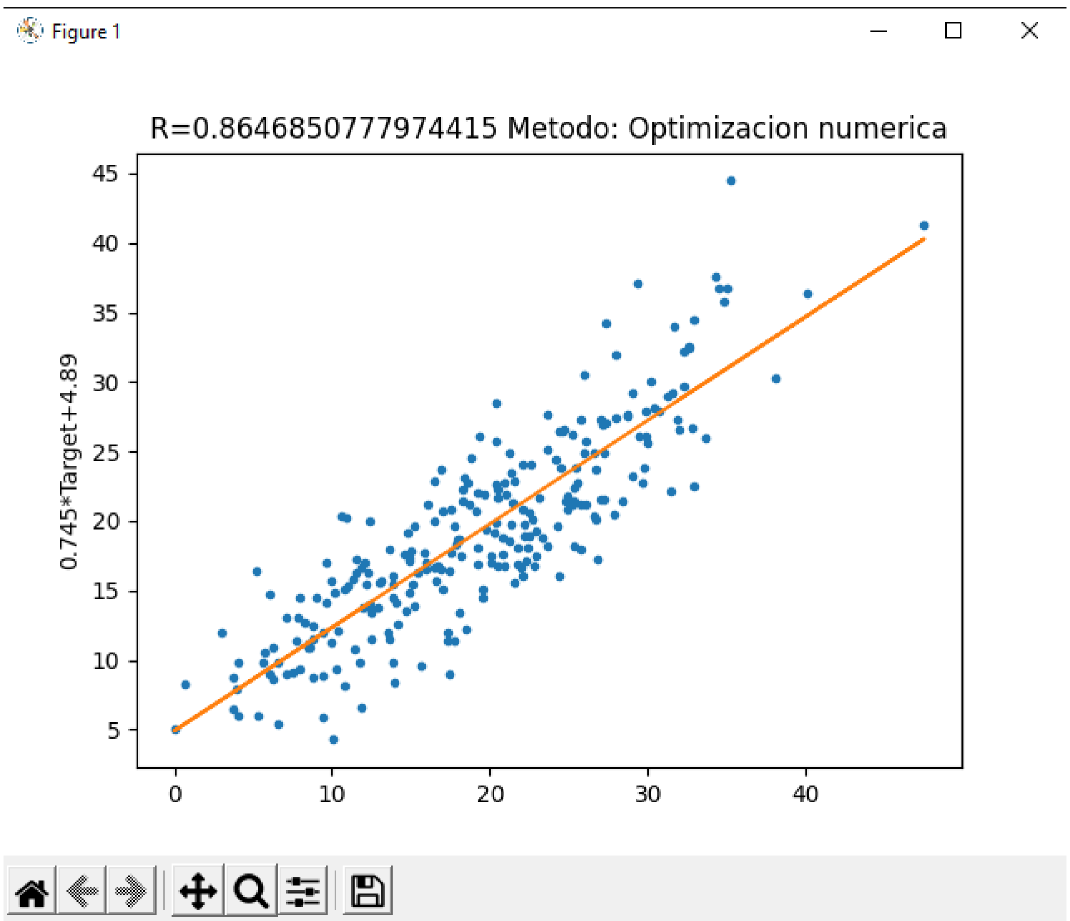
01 de junio del 2022

Desarrollo

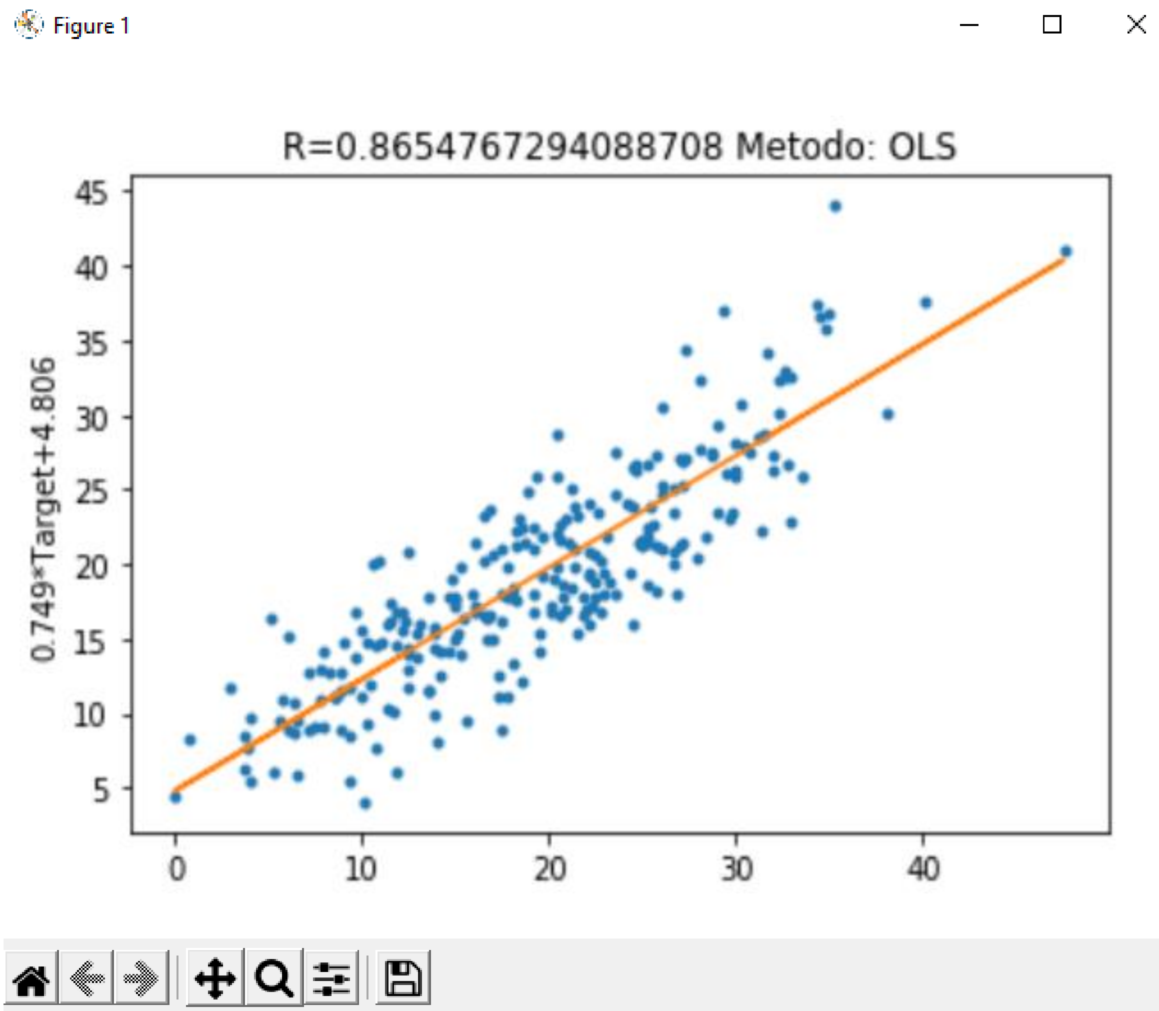
Para el desarrollo de esta meta se utilizaron elementos que se han hecho previamente, entre ellos se encuentra la matriz de diseño y la de confusión. Además de ello se utilizaron librerías como numpy para el manejo de arreglos, Scipy para el manejo de valores y Matplotlib para el manejo de interfaces gráficas para mostrar los resultados.

Resultados

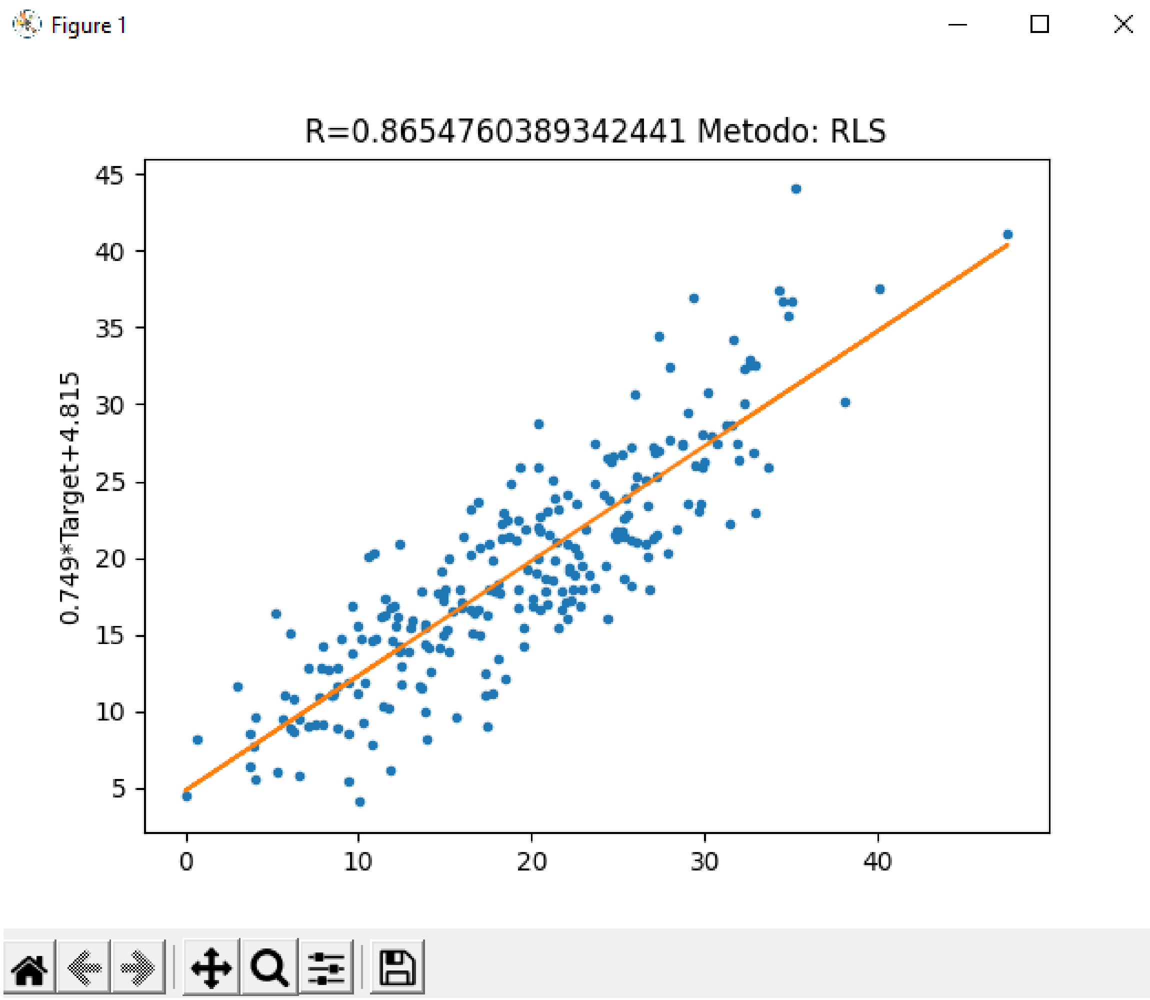
Inciso 1:




```
Perf 17.60205751469151 Grad 0.008948648911678832 Epochs 10001
RMSE:  3.0854222982270465
SSE:  [[4435.711]]
 MSE:  [[17.602]]
 RMSE:  [[4.195]]
 r = :  0.8646850777974415
 INDEX:  [[0.049]]
```

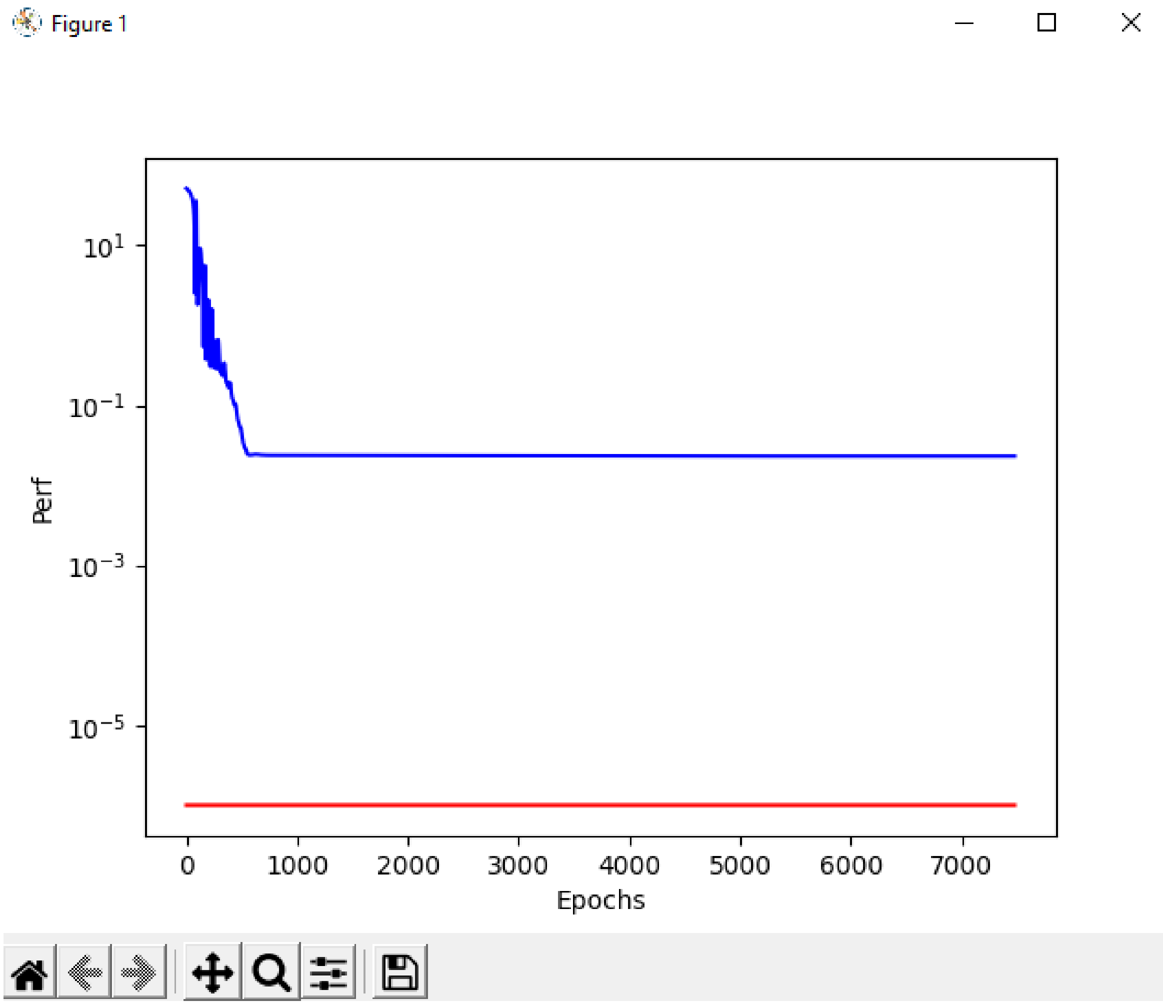



```
SSE:   [[4411.448]]
MSE:   [[17.506]]
RMSE:   [[4.184]]
r = :   0.8654767294088708
INDEX:   [[0.049]]
```

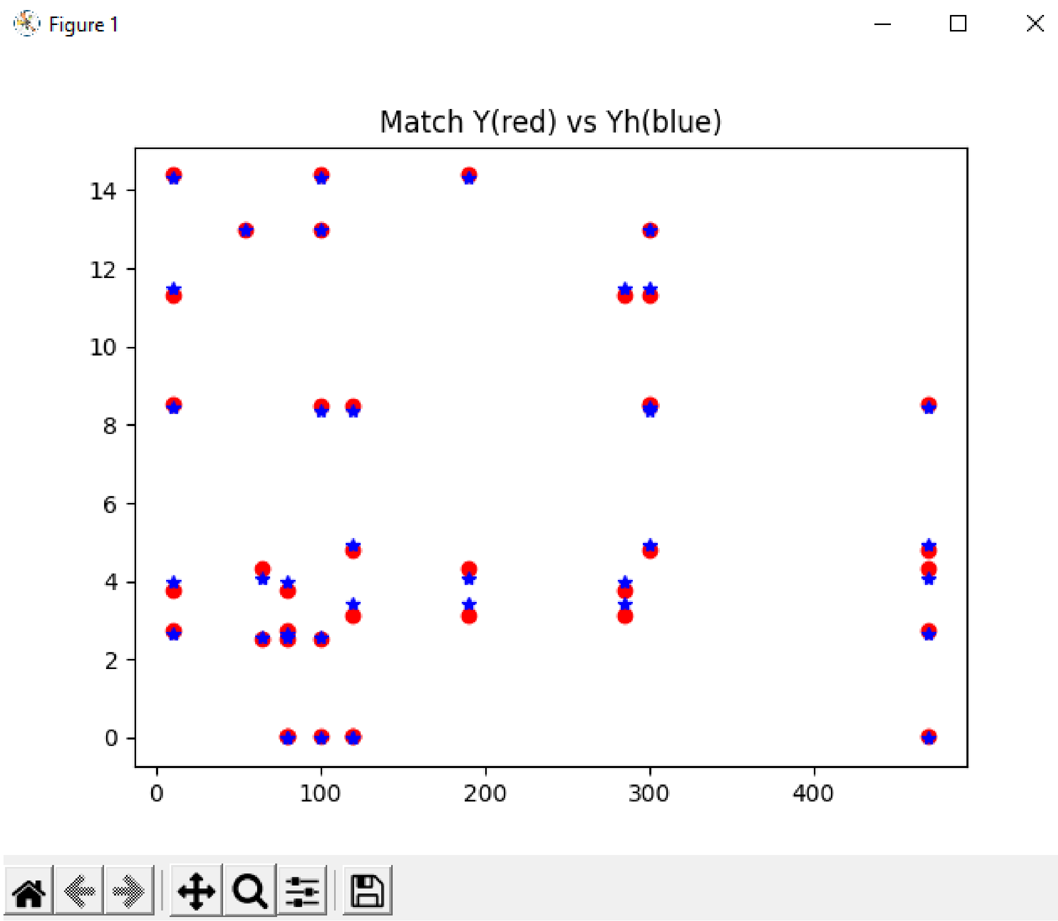



```
SSE:  [[4411.474]]
 MSE:  [[17.506]]
 RMSE:  [[4.184]]
 r = :  0.8654760389342441
 INDEX:  [[0.049]]
```

Inciso 2:




```
En la iteracion  7479  se alcanzo el gradiente minimo de  1e-06
gd: 9.859378890424805e-07
perf: [[0.023]]
```




```
[[1.236]
 [0.061]
 [0.04 ]
 [0.111]
 [1.208]]
    Y            Yh
 [[ 8.550e+00  8.444e+00]
 [ 3.790e+00  3.973e+00]
 [ 4.820e+00  4.923e+00]
 [ 2.000e-02 -1.017e-02]
 [ 2.750e+00  2.651e+00]
 [ 1.439e+01  1.432e+01]
 [ 2.540e+00  2.569e+00]
 [ 4.350e+00  4.054e+00]
 [ 1.300e+01  1.300e+01]
 [ 8.500e+00  8.377e+00]
 [ 5.000e-02 -1.994e-02]
 [ 1.132e+01  1.149e+01]
 [ 3.130e+00  3.439e+00]]
Y-Yh:
 [[ 0.106]
 [-0.183]
 [-0.103]
 [ 0.03 ]
 [ 0.099]
 [ 0.066]
 [-0.029]
 [ 0.296]
 [-0.003]
 [ 0.123]
 [ 0.07 ]
 [-0.166]
 [-0.309]]
```

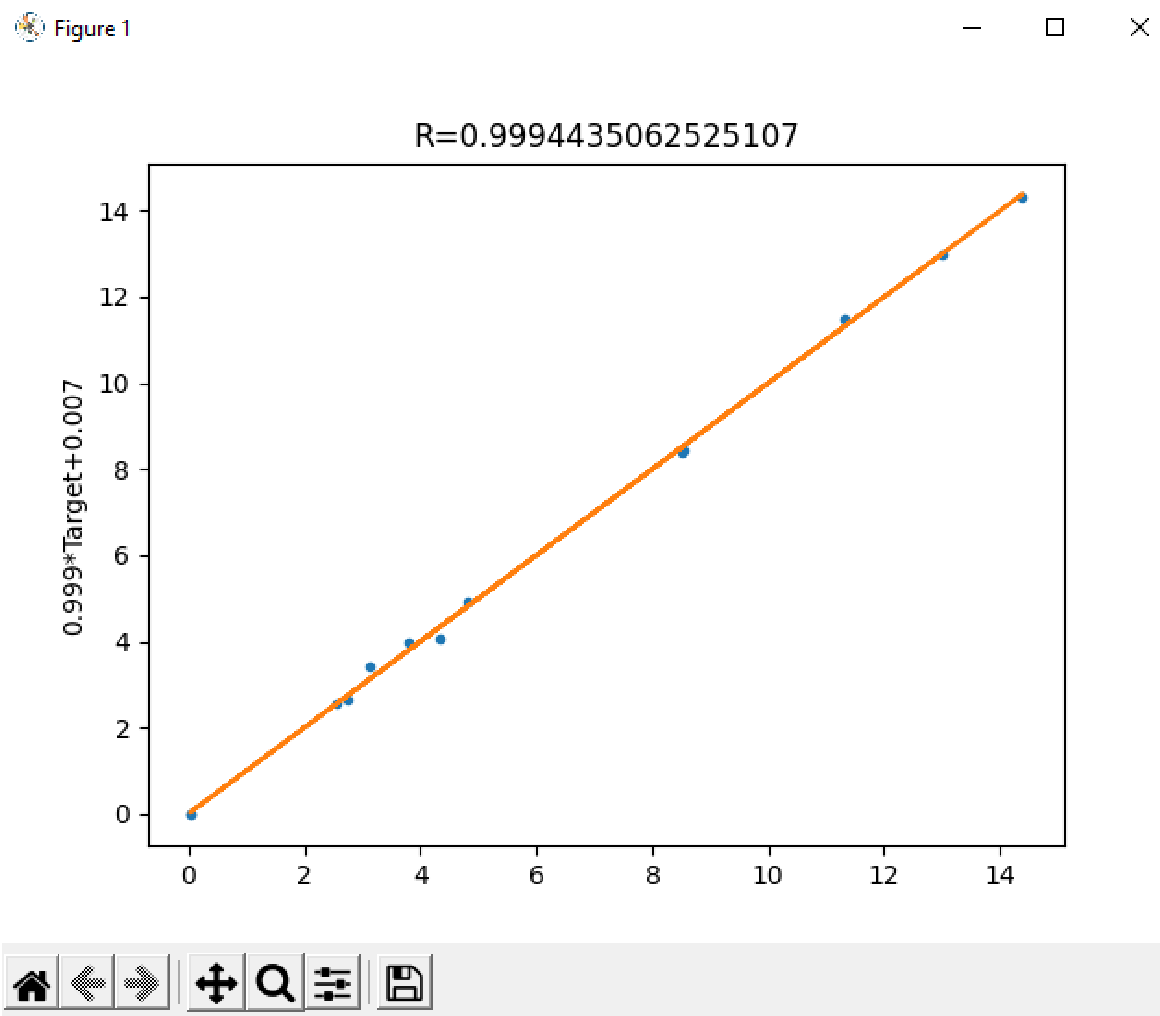


Figure 1
R=0.9994435062525107
0.999*Target+0.007
0
2
4
6
8
10
12
14

Inciso 3:

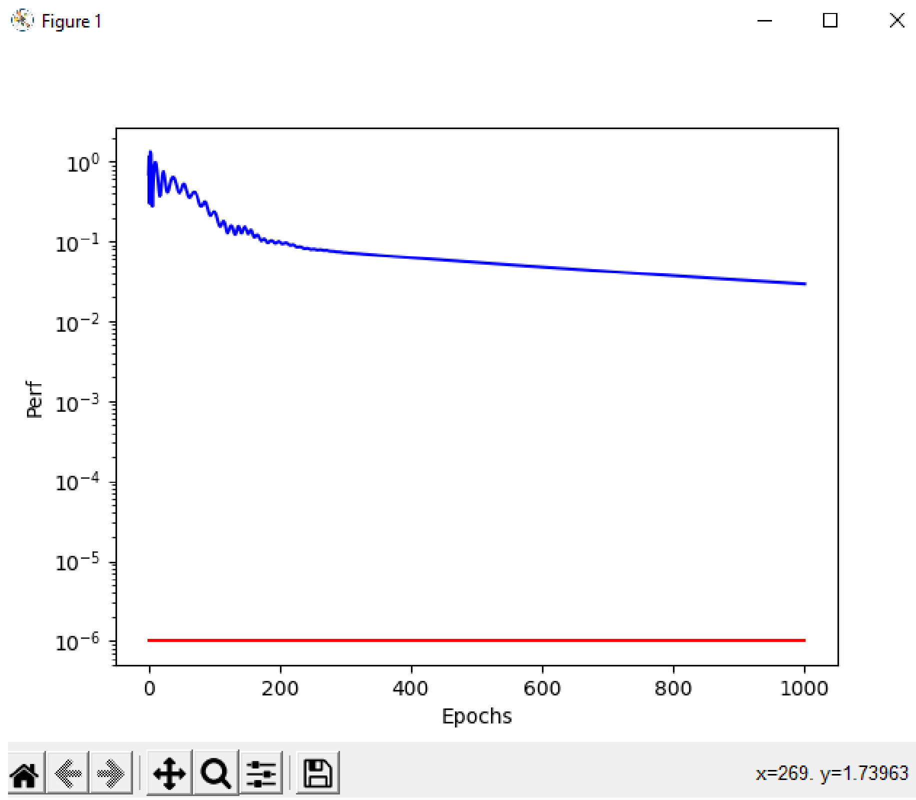


Figure 1
Perf
Epochs
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
200
400
600
800
1000
x=269. y=1.73963

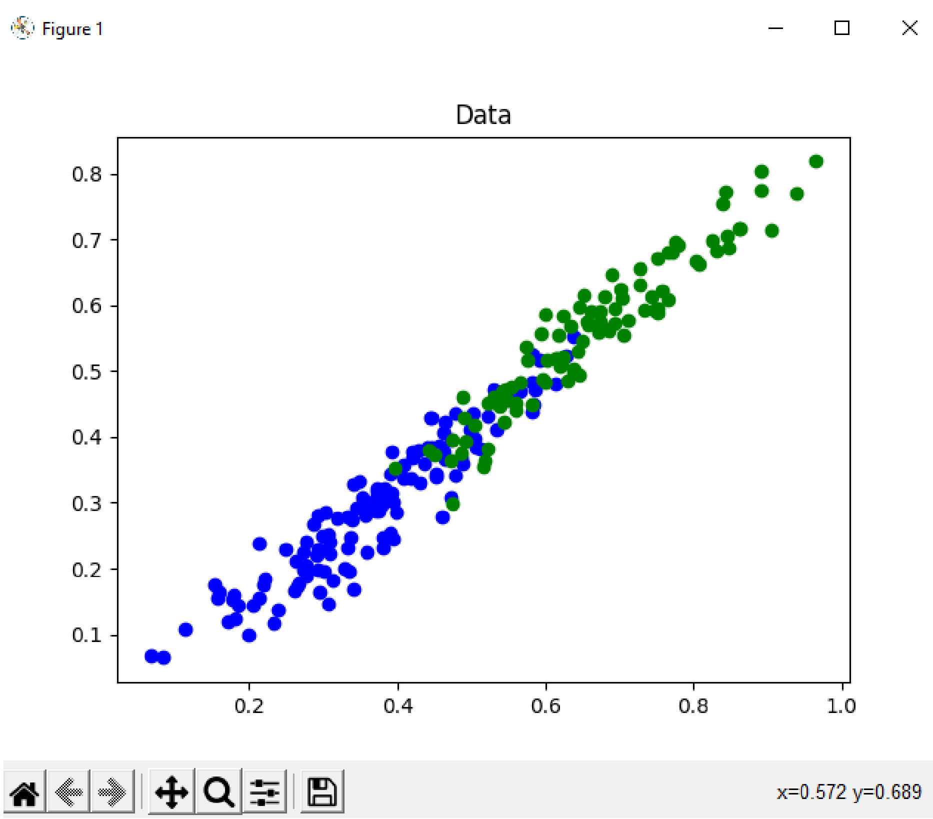
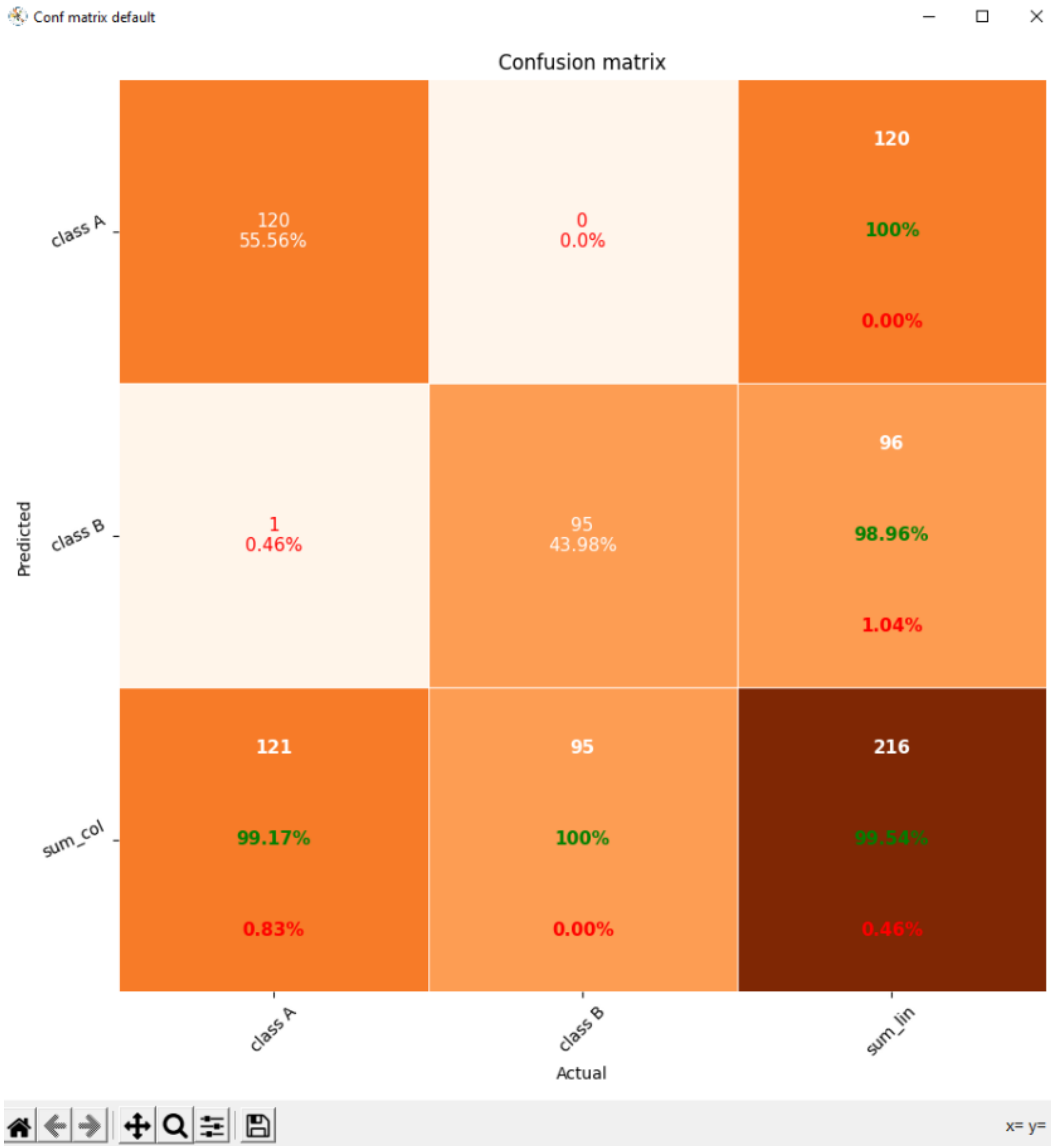
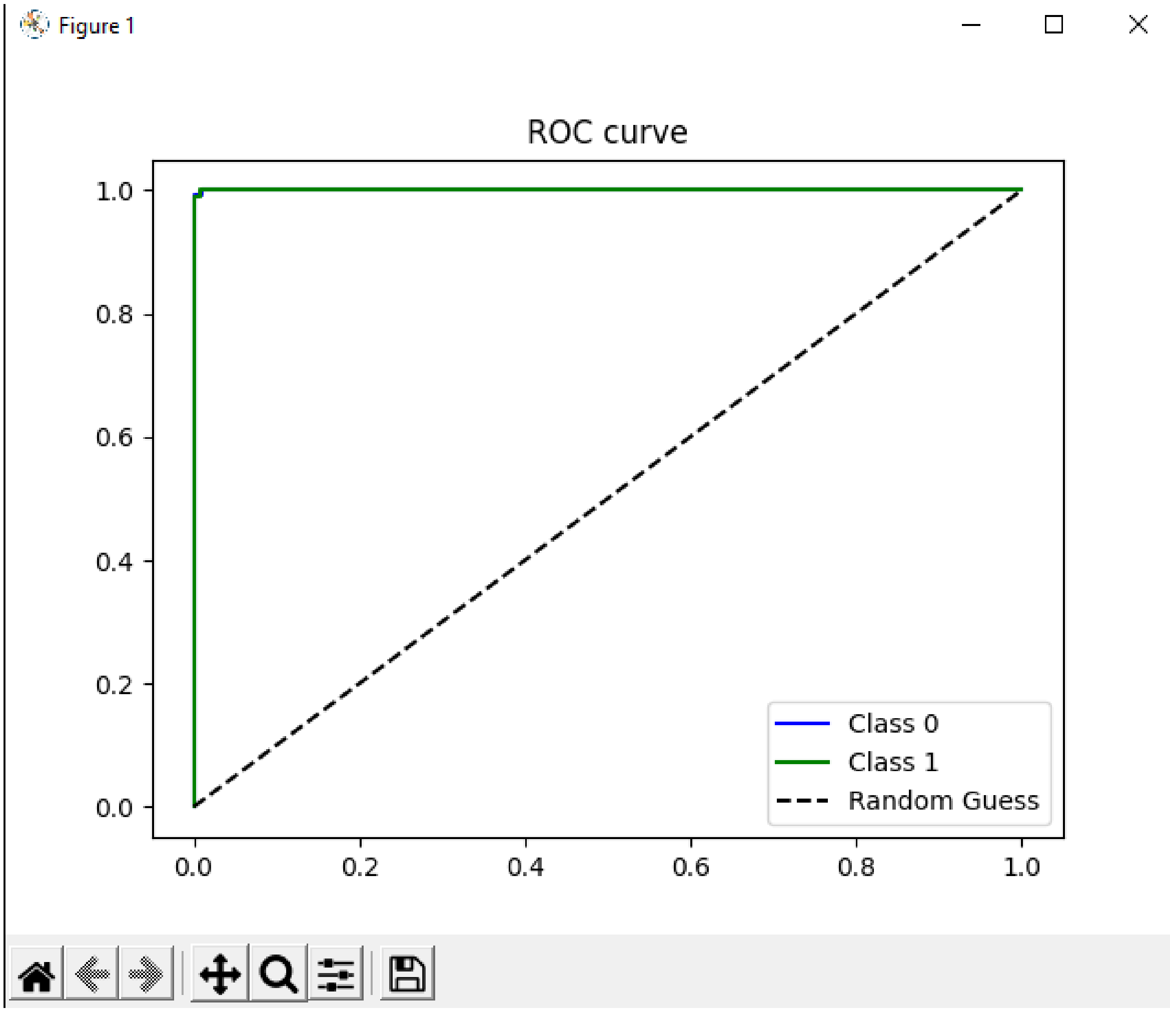


Figure 1
Data
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.2
0.4
0.6
0.8
1.0
x=0.572 y=0.689



Conf matrix default
Confusion matrix
120
55.56%
0
0.0%
120
100%
0.00%
1
0.46%
95
43.98%
96
98.96%
1.04%
121
99.17%
0.83%
95
100%
0.00%
216
99.54%
0.46%
class A
class B
sum_col
Predicted
class A
class B
sum_lin
Actual
x= y=




```
Max epochs at  1001
Perf 0.029428336751322228 Grad 0.0015684682420399954 Epochs 1001
[[120   1]
 [  0  95]]
```

Inciso 4:

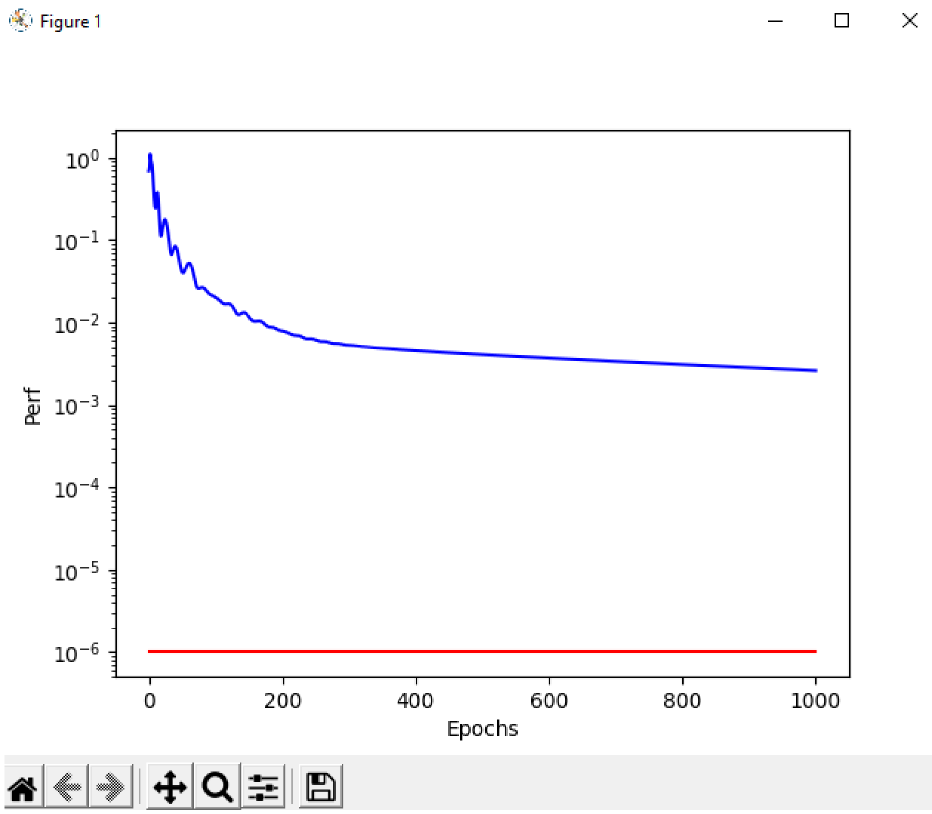


Figure 1
Perf
Epochs
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
200
400
600
800
1000

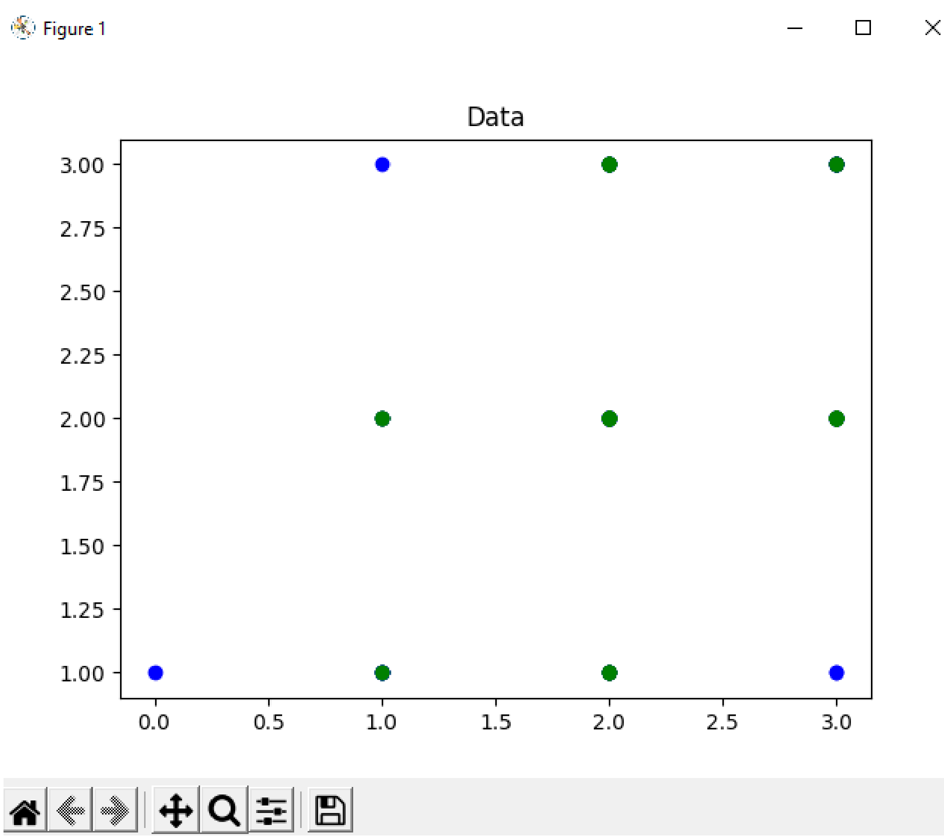
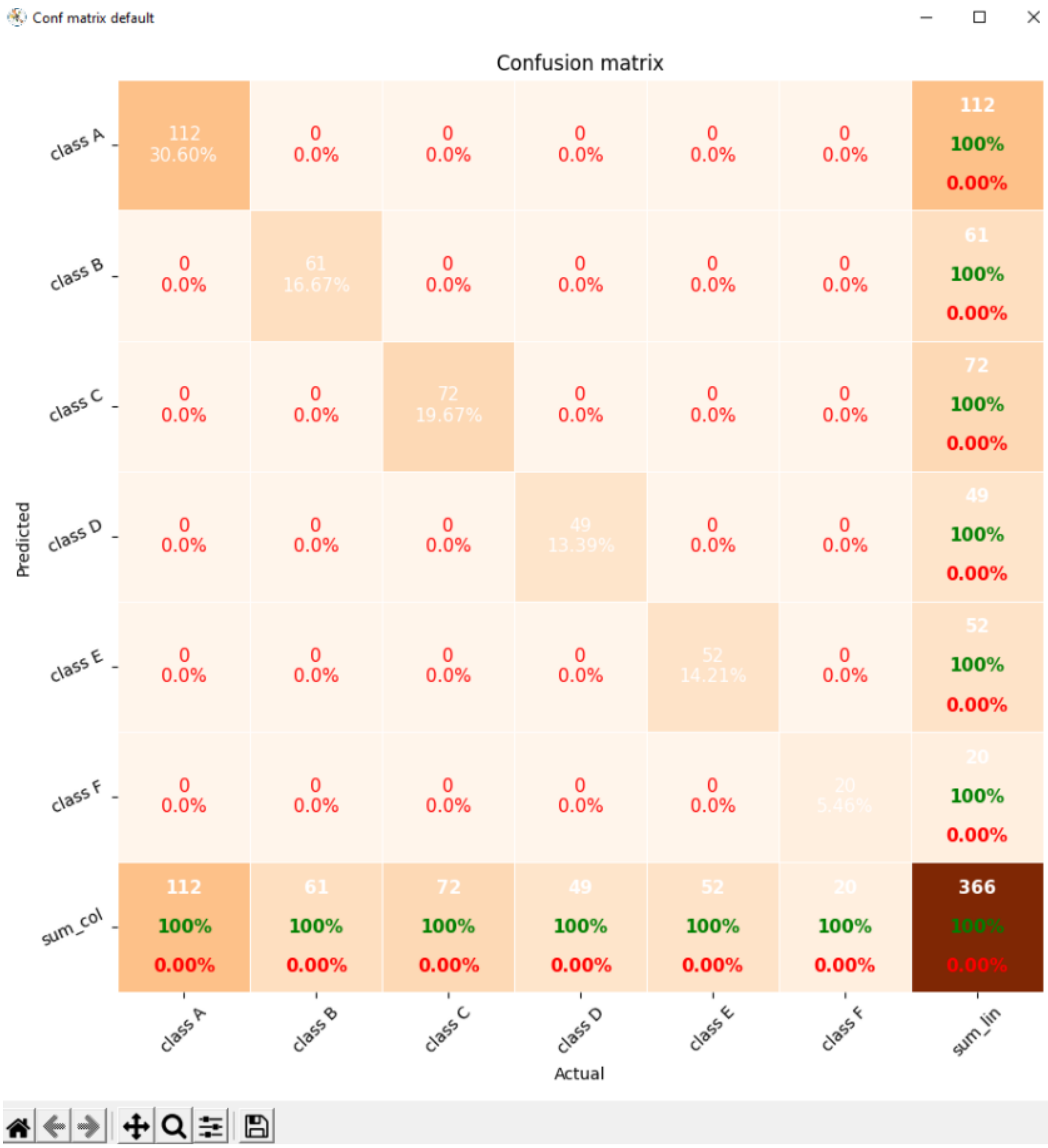
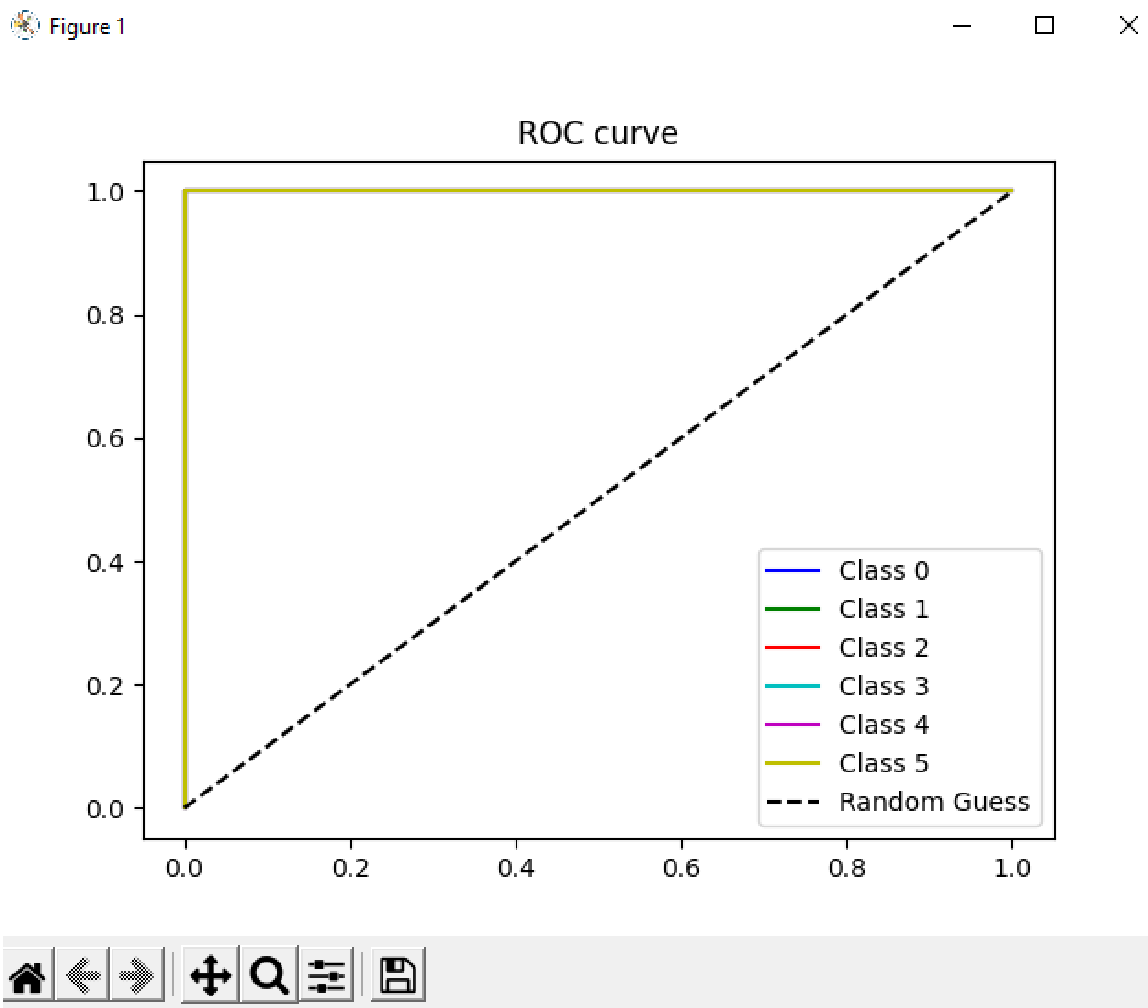


Figure 1
Data
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Conf matrix default
Confusion matrix
Predicted
Actual
class A
class B
class C
class D
class E
class F
sum_col
sum_lin
112 30.60%
61 16.67%
72 19.67%
49 13.39%
52 14.21%
20 5.46%
0 0.0%
112 100% 0.00%
61 100% 0.00%
72 100% 0.00%
49 100% 0.00%
52 100% 0.00%
20 100% 0.00%
366 100% 0.00%




```
Max epochs at  1001
Perf 0.0026166666902358747 Grad 0.0002464472786509669 Epochs 1001
[[112   0   0   0   0   0]
 [  0  61   0   0   0   0]
 [  0   0  72   0   0   0]
 [  0   0   0  49   0   0]
 [  0   0   0   0  52   0]
 [  0   0   0   0   0  20]]
```

Archivo: INCISO_1.py

```python
1 '''
2 1. Diseñar modelos de regresión a partir de los datos
3 (bodyfat_dataset.dat) que establecen la relación entre
4 la masa corporal (targets, última columna de la tabla)
5 de 252 pacientes a partir de 13 medidas predictoras
6 (inputs, primeras 13 columnas de la tabla) : Edad (años),
7 Peso (libras), Altura (pulgadas),Circunferencia del cuello (cm),
8 Circunferencia del pecho (cm), Circunferencia del abdomen 2 (cm),
9 Circunferencia de la cadera (cm), Circunferencia del muslo (cm),
10 Circunferencia de la rodilla (cm),Circunferencia del tobillo (cm),
11 Circunferencia de bíceps (extendida) (cm), Circunferencia del
12 antebrazo (cm) y Circunferencia de la muñeca (cm).
13 El objetivo del análisis de regresión es una forma de técnica
14 de modelado predictivo que investiga la relación entre una
15 variable dependiente (objetivo) y una variable independiente
16 (predictor). Esta técnica se utiliza para pronosticar y
17 encontrar la relación del efecto causal entre las variables.
18 EQUIPO 5:
19     HUERTAS VILLEGAS CESAR
20     URIAS VEGA JUAN DANIEL
21     ZAVALA ROMAN IRVIN EDUARDO
22 '''
23 import math
24 import numpy as np
25 import designMatrix as dm
26 import matplotlib.pyplot as plt
27 import scipy.stats as stats
28 np.set_printoptions(precision=3) #Para limitar decimales en numpy
29 #DE LINEAS 30 A 100 ES REGRESION CON OPTIMIZACION NUMERICA
30 class optimParam:
31     epochs = 0
32     goal = 0
33     min_grad = 0
34     show = 0
35 def RegressionGradMSE(A,e):
36     q = A.shape[0]
37     m = e.shape[1]
38     gradMse = -2*A.T@e / (m*q)
39     return gradMse
40 def RegressionMSE(A,Y,vecX):
41     col = A.shape[1]
42     m = Y.shape[1]
43     Theta = np.resize(vecX, (col,m))
44     Yh = A@Theta
45     e = Y-Yh
46     MSE = (np.square(e)).mean()
47     return MSE, e
48 def RegressionOptimgd(X,Y,grado,oP):
49     q,n = X.shape
50     oP.epochs+=1
51     m = Y.shape[1]
52     A = dm.designMatrix(grado,X)
53     vecX = np.random.rand(A.shape[1], m)
54     #Para graficar
55     t_arreglo = np.array([])
56     goal_a = np.array([])
57     perf_a = np.array([])
58 
59 
60     wt = vecX
61 
62     mt =  np.zeros((wt.shape[0], 1))
63     vt = np.zeros((wt.shape[0], 1))
64     mt_gorrito = np.zeros((wt.shape[0], 1))
```

```
65      vt_gorrito = np.zeros((wt.shape[0], 1))
66      beta_1 = 0.975
67      beta_2 = 0.999
68      alpha = 0.01
69      oP.epochs+=1
70      for t in range(oP.epochs):
71          perf, e = RegressionMSE(A,Y,wt)
72          gd = RegressionGradMSE(A,e)
73          #vectores anteriores
74          mt_gorrito_anterior = mt_gorrito
75          #Algoritmo
76          mt = beta_1*mt+(1-beta_1)*gd
77          vt = beta_2*vt+(1-beta_2)*gd**2
78          mt_gorrito = mt/(1-beta_1**(t+1))
79          vt_gorrito = vt/(1-beta_2**(t+1))
80          wt = wt - (alpha/(np.sqrt(vt_gorrito)+1e-8))*(beta_1*mt_gorrito_anterior+((1-beta_1)/(1-beta_1**(t+1)))*gd)
81          if(perf <= oP.goal):
82              break
83          elif(np.linalg.norm(gd) <  oP.min_grad):
84              break
85          elif(t ==  oP.epochs-1):
86              break
87          perf_a = np.append(perf_a, perf)
88      vecX = wt
89      '''t_arreglo = np.arange(t)
90      goal_a = np.zeros(t)+oP.goal
91      plt.yscale("log")
92      plt.plot(t_arreglo, perf_a, 'b')
93      plt.plot(t_arreglo, goal_a, 'r')
94      plt.ylabel("Perf")
95      plt.xlabel("Epochs")
96      plt.show()'''
97
98      print("Perf",perf,"Grad",gd[0][0], "Epochs", t)
99      Theta = np.resize(vecX, (A.shape[1],m))
100     return Theta, A
101
102 #DE LINEAS 103 A 115 ES OLS
103 def RegressionOLSMoorePenrose(X,Y):
104     q,n = X.shape
105     #A = np.ones((q,1)) #esto sirve si es regresion lineal
106     A = dm.designMatrix(2, X)
107     A = np.hstack((A, X))
108     ATA = np.matmul(A.T,A)
109     b = A.T@Y
110     THETA = np.linalg.pinv(ATA)@b
111     Yh = A@THETA
112     e = Y-Yh
113     RMSE = math.sqrt((np.square(e)).mean())
114     print("RMSE: ", RMSE)
115     return THETA, RMSE, Yh, e
116
117 #DE LINEAS 118 A 144 ES RLS
118 def P(x):
119     P.p = x
120 def Theta(x):
121     Theta.t = x
122 def RegressionRLS(X,Y):
123     q,n = X.shape
124     A = dm.designMatrix(1,X)
125     P.p = np.zeros((X.shape[0], Y.shape[1]))
126     Theta.t = np.zeros((X.shape[1], Y.shape[1]))
127     for p in range(q):
128         xIn = A[p,:]
129         xOut = Y[p,:]
130         THETA = rls(p,xIn,xOut)
131     return THETA, A
132 def rls(k,a,y):
133     npk = a.size
```

```python
    m = y.size
    if(k == 0):
        Theta(np.zeros((npk,m)))
        P(1e10*np.eye(npk, npk))
    tmp1 = (P.p@a.reshape(-1,1))*(a@P.p)
    tmp2 = 1+(a@P.p)@a.T
    P(P.p-tmp1/tmp2)
    diff = y-a@Theta.t
    tmp3 = P.p@a.reshape(-1,1)
    Theta(Theta.t + tmp3*diff)
    return Theta.t

def desempeno(X, Y, Yh, string):
    r, _= stats.pearsonr (Y.T[0], Yh.T[0])
    q,n = X.shape
    e = Y-Yh
    SSE = e.T@e
    MSE = SSE/q
    RMSE = np.sqrt(MSE)
    INDEX = r/MSE
    print("SSE: ", SSE, "\n","MSE: ", MSE, "\n", "RMSE: ", RMSE, "\n","r = : ", r, "\n","INDEX: ", INDEX, "\n",)

    plt.plot(Y.T[0], Yh.T[0], '.')
    m, b = np.polyfit(Y.T[0], Yh.T[0], 1)
    plt.plot(Y.T[0], m*Y.T[0] + b)
    plt.title("R={} Metodo: {}".format(r, string))
    str = "{}*{}+{}".format(round(m,3),"Target",round(b,3))
    plt.ylabel(str)
    plt.show()
dataset = np.loadtxt('D:\\Eduardo\\Meta_5_5\\{}'.format("bodyfat_dataset.dat"), delimiter = '\t')
dataset = np.array(dataset)
X = dataset[:,:-1]
Y = dataset[:,-1:]
oP = optimParam()
oP.epochs = 10000
oP.goal = 1e-8
oP.min_grad = 1e-10
oP.show = 20

n = X.shape[1]
m = Y.shape[1]

grado = 1
thetaHat,A = RegressionOptimgd(X,Y,grado,oP)
Yh1 = A@thetaHat
theta, rmse, Yh2, e = RegressionOLSMoorePenrose(X,Y)
thetaHat, A = RegressionRLS(X,Y)
Yh3 = A@thetaHat
desempeno(X, Y,Yh1, "Optimizacion numerica")
desempeno(X,Y,Yh2, "OLS")
desempeno(X,Y,Yh3, "RLS")

'''
SE PUEDE CONCLUIR QUE EL METODO OLS DA LA MEJOR CORRELACION
AUNQUE CON GRADO 2, LOS DEMAS CON GRADOS MAYORES A 1 BAJAN
EL RENDIMIENTO
'''
```

Archivo: INCISO_2.py

INCISO_2.py

```python
'''
2. Diseñar modelos de regresión a partir de los datos
(reaction_dataset.dat) que establecen la relación
entre la velocidad de reacción (targets, última columna
de la tabla) y concentraciones de tres reactivos
como medidas predictoras (inputs, primeras tres columnas
de la tabla): x1(hidrógeno), x2(n-pentano),
x3(isopentano). La velocidad de reacción (y^) para la
cinética de reacción es el modelo de Hougen-
Watson:

    ^         01x2 - x3/05
    y =   _______________________
          1 + 02x1 + 03x2 + 04x3

donde Y^ es la velocidad de reacción estimada,
x1,x2,x3 son las concentraciones de hidrógeno, n-
pentano e isopentano, respectivamente, y
01,02, ... , 05 son parámetros del modelo.

EQUIPO 5:
    HUERTAS VILLEGAS CESAR
    URIAS VEGA JUAN DANIEL
    ZAVALA ROMAN IRVIN EDUARDO
'''
import numpy as np
import designMatrix as dm
import matplotlib.pyplot as plt
import scipy.stats as stats
np.set_printoptions(precision=3)
def funcionMSE(X,Y,theta):
    yh = funcion(X,theta)
    e = Y - yh
    SSE = e.T@e
    MSE = SSE/len(Y)
    return MSE, e
def funcion(x, theta): #Modelo de Hougen-Watson
    x1 = x[:,0].reshape(-1,1)
    x2 = x[:,1].reshape(-1,1)
    x3 = x[:,2].reshape(-1,1)
    yh = (theta[0]*x2 - (x3/theta[4]))/(1 + theta[1]*x1 + theta[2]*x2 + theta[3]*x3)
    return yh
def gradiente(x, y,  theta):#Gradiente del modelo de Hougen-Watson
    x1 = x[:,0].reshape(-1,1)
    x2 = x[:,1].reshape(-1,1)
    x3 = x[:,2].reshape(-1,1)
    dyh_d01 = x2/(1 + theta[1]*x1 + theta[2]*x2 + theta[3]*x3)
    dyh_d02 = -1*((theta[0]*theta[4]*x2-x3)*x1)/(theta[4]*(1 + theta[1]*x1 + theta[2]*x2 + theta[3]*x3))
    dyh_d03 = -1*((theta[0]*theta[4]*x2-x3)*x2)/(theta[4]*(1 + theta[1]*x1 + theta[2]*x2 + theta[3]*x3))
    dyh_d04 = -1*((theta[0]*theta[4]*x2-x3)*x3)/(theta[4]*(1 + theta[1]*x1 + theta[2]*x2 + theta[3]*x3))
    dyh_d05 = x3/((theta[4]**2)*(1 + theta[1]*x1 + theta[2]*x2 + theta[3]*x3))
    J = -1*np.hstack((dyh_d01,dyh_d02,dyh_d03,dyh_d04,dyh_d05))
    perf, e = funcionMSE(x, y, theta)
    gd = (2.0*J.T)@e
    return gd
def nadam(X, Y, theta):
    wt = theta #Se van a optimizar los parametros theta
    n = len(theta)
    mt = np.zeros((n,1))
    vt = np.zeros((n,1))
    mt_gorrito = np.zeros((n,1))
    vt_gorrito = np.zeros((n,1))

    #Para graficar
    t_arreglo = np.array([])
```

```python
    goal_a = np.array([])
    perf_a = np.array([])
    beta_1 = 0.975
    beta_2 = 0.999
    alpha = 0.01
    tmax = 20000
    goal = 1e-6
    mingrad = 1e-6

    for t in range(tmax):
        perf, e = funcionMSE(X,Y,wt)
        gd =  gradiente(X,Y,wt)
        #vectores anteriores
        mt_gorrito_anterior = mt_gorrito
        #Algoritmo
        mt = beta_1*mt+(1-beta_1)*gd
        vt = beta_2*vt+(1-beta_2)*gd**2
        mt_gorrito = mt/(1-beta_1**(t+1))
        vt_gorrito = vt/(1-beta_2**(t+1))
        wt = wt - (alpha/(np.sqrt(vt_gorrito)+1e-8))*(beta_1*mt_gorrito_anterior+((1-beta_1)/(1-beta_1**(t+1)))*gd)

        if(perf <= goal):
            print("En la iteracion ",t," se alcanzo la precision de ",goal)

            break
        elif(np.linalg.norm(gd) < mingrad):
            print("En la iteracion ",t," se alcanzo el gradiente minimo de ",mingrad)

            break
        elif(t == tmax-1):
            print("Se alcanzo la maxima cantidad de iteraciones, la meta no se consiguio :(")
            break
        perf_a = np.append(perf_a, perf)
    print("gd:",np.linalg.norm(gd), "\nperf:",perf)
    t_arreglo = np.arange(t)
    goal_a = np.zeros(t)+goal
    plt.yscale("log")
    plt.plot(t_arreglo, perf_a, 'b')
    plt.plot(t_arreglo, goal_a, 'r')
    plt.ylabel("Perf")
    plt.xlabel("Epochs")
    plt.show()
    return wt


dataset = np.loadtxt('D:\\Eduardo\\Meta_5_5\\{}'.format("reaction_dataset.dat"), delimiter = ',')
dataset = np.array(dataset)
theta = np.array([[1], #Theta de inicio ya que no se tiene idea del valor
                  [1],
                  [1],
                  [1],
                  [1]])
X = dataset[:,:-1]
Y = dataset[:,-1:]
theta = nadam(X,Y,theta)
Yh = funcion(X, theta)
print(theta,"\n\tY\t\t\tYh\n",np.hstack((Y,Yh)))
plt.plot(X,Y, 'or')
plt.plot(X,Yh, '*b')
plt.title("Match Y(red) vs Yh(blue)")
plt.show()
print("Y-Yh:\n",Y-Yh)
r, _= stats.pearsonr(Y.T[0], Yh.T[0])
plt.plot(Y.T[0], Yh.T[0], '.')
m, b = np.polyfit(Y.T[0], Yh.T[0], 1)
plt.plot(Y.T[0], m*Y.T[0] + b)
plt.title("R={}".format(r))
str = "{}*{}+{}".format(round(m,3),"Target",round(b,3))
```

```python
plt.ylabel(str)
plt.show()

#Distintas pruebas
prueba = np.array([[470,300,10]])
yh_prueba = funcion(prueba, theta)
print(yh_prueba) #Deberia aproximar 8.55

prueba = np.array([[285,80,10]])
yh_prueba = funcion(prueba, theta)
print(yh_prueba) #Deberia aproximar 3.79

prueba = np.array([[285,190,120]])
yh_prueba = funcion(prueba, theta)
print(yh_prueba) #Deberia aproximar 3.13

```

Archivo: INCISO_3.py

INCISO_3.py

```python
'''
3. Diseñar modelos de clasificación logística para detectar cáncer
a partir de datos de espectrometría de masas en perfiles de
proteínas (ovarian_dataset.dat) que establecen la relación para
distinguir entre pacientes con cáncer (targets, últimas dos
columnas de la tabla) y medidas de intensidad de iones
(inputs, primeras 100 columnas de la tabla).
La metodología es seleccionar un conjunto reducido de medidas
o "características" que pueden usarse para distinguir entre
pacientes con cáncer y de control mediante un clasificador. Estas
características serán niveles de intensidad de iones a valores
específicos de masa / carga. El objetivo del modelo de
clasificación es distinguir entre el cáncer y el control de
los pacientes a partir de los datos de espectrometría de masas.

EQUIPO 5:
    HUERTAS VILLEGAS CESAR
    URIAS VEGA JUAN DANIEL
    ZAVALA ROMAN IRVIN EDUARDO
'''
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
from sklearn.metrics import roc_curve
class optimParam:
    epochs = 1000
    goal = 1e-6
    lr = 0.001
    lr_dec = 0.7
    lr_inc = 1.05
    max_perf_inc = 1.04
    mc = 0.9
    min_grad = 1.0e-6
    show = 20
def plotData(X,Y):
    classes = np.unique(Y)
    colors = ['b','g','r','c','m','y']
    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']
    for i in range(Y.shape[1]):
        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):
    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):
    q, col = A.shape
    m = Y.shape[1]
    theta = np.resize(vecX, (col,m))
    hx = A@theta
    P = sigmoide(hx)
    e = Y-P
    #Para evitar log(0) se le suma un valor muy pequeño
    H = Y*np.log(P+1e-12)+(1-Y)*np.log(1-P+1e-12)
    J = -1*np.sum(np.sum(H))/(m*q)
    return J,e
def  logitAvgLossGrad(A,e):
    q = A.shape[0]
    m = e.shape[1]
    grad = -A.T@e / (m*q)
    return grad.flatten()
def logitRegressionNADAM(X,Y, oP, grado):
    q,n = X.shape
    oP.epochs+=1
    m = Y.shape[1]
    A = designMatrix(grado,X)
    theta = np.zeros((A.shape[1], m))
    vecX = theta.flatten().reshape(-1,1)
    t_arreglo = np.array([])
    goal_a = np.array([])
    perf_a = np.array([])


    wt = vecX
```

```python
    mt =  np.zeros((wt.shape[0], 1))
    vt = np.zeros((wt.shape[0], 1))
    mt_gorrito = np.zeros((wt.shape[0], 1))
    vt_gorrito = np.zeros((wt.shape[0], 1))
    beta_1 = 0.975
    beta_2 = 0.999
    alpha = 0.1
    oP.epochs+=1
    for t in range(oP.epochs):
        perf, e = logitAvgLoss(A,Y,wt)
        gd = logitAvgLossGrad(A,e)
        #vectores anteriores
        mt_gorrito_anterior = mt_gorrito
        #Algoritmo
        mt = beta_1*mt+(1-beta_1)*gd
        vt = beta_2*vt+(1-beta_2)*gd**2
        mt_gorrito = mt/(1-beta_1**(t+1))
        vt_gorrito = vt/(1-beta_2**(t+1))
        wt = wt - (alpha/(np.sqrt(vt_gorrito)+(1e-8)))*(beta_1*mt_gorrito_anterior+((1-beta_1)/(1-beta_1**(t+1)))*gd)
        if(perf <= oP.goal):
            print("Perf goal reached at ", t)
            break
        elif(np.linalg.norm(gd) <  oP.min_grad):
            print("Min grad at ", t)
            break
        elif(t ==  oP.epochs-1):
            print("Max epochs at ", t)
            break
       #print("perf:",perf,"|grad:", np.linalg.norm(gd),"|epoch:",t)
        perf_a = np.append(perf_a, perf)
    #Grafica estatica
    t_arreglo = np.array(range(0,t,1))
    goal_a = np.zeros(t)+oP.goal
    plt.yscale("log")
    plt.plot(t_arreglo, perf_a, 'b')
    plt.plot(t_arreglo, goal_a, 'r')
    plt.ylabel("Perf")
    plt.xlabel("Epochs")
    plt.show()
    vecX = wt
    print("Perf",perf,"Grad",np.linalg.norm(gd), "Epochs", t)
    thetaHat = np.resize(vecX, (A.shape[1],m))
    return thetaHat, A


#Este codigo es igual a la meta 5.4 pero cargando el dataset del cancer de ovario
dataset = np.loadtxt('D:\\Eduardo\\Meta_5_5\\{}'.format("ovarian_dataset.dat"), delimiter = '\t')
dataset = np.array(dataset)
X = dataset[:,:-2] #inputs primeras 100 columnas
Y = dataset[:,-2:] #targets ultimas 2 columnas
grado = 1

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: INCISO_4.py

```
INCISO_4.py
'''
4. Diseñar modelos de clasificación logística para detectar
enfermedades de la piel (dermatology_dataset.dat) de 366
pacientes, que establecen la relación para clasificar las
enfermedades de psoriasis, dermatitis seborreica, liquen plano,
pitiriasis rosada, dermatitis crónica y la pitiriasis rubra
pilaris (targets, última columna de la tabla) y medidas con
12 características clínicas mas 22 características
histopatológicas (inputs, primeras 34 columnas de la tabla).

EQUIPO 5:
    HUERTAS VILLEGAS CESAR
    URIAS VEGA JUAN DANIEL
    ZAVALA ROMAN IRVIN EDUARDO
'''
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
from sklearn.metrics import roc_curve
class optimParam:
    epochs = 1000
    goal = 1e-6
    lr = 0.001
    lr_dec = 0.7
    lr_inc = 1.05
    max_perf_inc = 1.04
    mc = 0.9
    min_grad = 1.0e-6
    show = 20
def plotData(X,Y):
    classes = np.unique(Y)
    colors = ['b','g','r','c','m','y']
    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']
    for i in range(Y.shape[1]):
        fpr, tpr, thresholds = roc_curve(Y[:,i].reshape(-1,1), ph[:,i].reshape(-1,1), pos_label=1)
```

```python
        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):
    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):
    q, col = A.shape
    m = Y.shape[1]
    theta = np.resize(vecX, (col,m))
    hx = A@theta
    P = sigmoide(hx)
    e = Y-P
    #Para evitar log(0) se le suma un valor muy pequeno
    H = Y*np.log(P+1e-12)+(1-Y)*np.log(1-P+1e-12)
    J = -1*np.sum(np.sum(H))/(m*q)
    return J,e
def  logitAvgLossGrad(A,e):
    q = A.shape[0]
    m = e.shape[1]
    grad = -A.T@e / (m*q)
    return grad.flatten()
def logitRegressionNADAM(X,Y, oP, grado):
    q,n = X.shape
    oP.epochs+=1
    m = Y.shape[1]
    A = designMatrix(grado,X)
    theta = np.zeros((A.shape[1], m))
    vecX = theta.flatten().reshape(-1,1)
    t_arreglo = np.array([])
    goal_a = np.array([])
    perf_a = np.array([])


    wt = vecX
    mt =  np.zeros((wt.shape[0], 1))
    vt = np.zeros((wt.shape[0], 1))
    mt_gorrito = np.zeros((wt.shape[0], 1))
    vt_gorrito = np.zeros((wt.shape[0], 1))
    beta_1 = 0.975
    beta_2 = 0.999
    alpha = 0.1
    oP.epochs+=1
    for t in range(oP.epochs):
        perf, e = logitAvgLoss(A,Y,wt)
        gd = logitAvgLossGrad(A,e)
        #vectores anteriores
        mt_gorrito_anterior = mt_gorrito
        #Algoritmo
        mt = beta_1*mt+(1-beta_1)*gd
        vt = beta_2*vt+(1-beta_2)*gd**2
        mt_gorrito = mt/(1-beta_1**(t+1))
        vt_gorrito = vt/(1-beta_2**(t+1))
        wt = wt - (alpha/(np.sqrt(vt_gorrito)+(1e-8)))*(beta_1*mt_gorrito_anterior+((1-beta_1)/(1-beta_1**(t+1)))*gd)
        if(perf <= oP.goal):
            print("Perf goal reached at ", t)
            break
        elif(np.linalg.norm(gd) <  oP.min_grad):
            print("Min grad at ", t)
            break
        elif(t ==  oP.epochs-1):
            print("Max epochs at ", t)
            break
```

```python
            #print("perf:",perf,"|grad:", np.linalg.norm(gd),"|epoch:",t)
            perf_a = np.append(perf_a, perf)
        #Grafica estatica
        t_arreglo = np.array(range(0,t,1))
        goal_a = np.zeros(t)+oP.goal
        plt.yscale("log")
        plt.plot(t_arreglo, perf_a, 'b')
        plt.plot(t_arreglo, goal_a, 'r')
        plt.ylabel("Perf")
        plt.xlabel("Epochs")
        plt.show()
```

```python
    vecX = wt
    print("Perf",perf,"Grad",np.linalg.norm(gd), "Epochs", t)
    thetaHat = np.resize(vecX, (A.shape[1],m))
    return thetaHat, A


#Este codigo es igual a la meta 5.4 pero cargando el dataset del dermatologia
dataset = np.loadtxt('D:\\Eduardo\\Meta_5_5\\{}'.format("dermatology.dat"), delimiter = ' ')
dataset = np.array(dataset)
X = dataset[:,:-1] #inputs primeras 34 columnas
Y = dataset[:,-1:] #targets ultima columna
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

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
pretty_confusion_matrix.py
# -*- 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
```

```
            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:
                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",
):
    """
    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
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)
```

```
    # 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
```