Practica 3

Regresión Logística

Para esta practica hemos implementado la regresión logística y lineal de dos ejemplos. Uno de ellos (parte A ) representa las notas de estudiantes para entrar a la universidad. Mientras que los otros (parte B) representan si unos microchips pasan el test de calidad establecido.

Ambas funciones deben sobrepasar una precisión de predicción del 92% y 80% respectivamente.

**CÓDIGO FUENTE DE LA PARTE A:**

def our\_test\_A():

    #read data

    X , Y = readData("ex2data1.txt")

    #initial values

    b\_init = -8

    w\_init = np.array([0.0, 0.0])

    iterations = 1000

    alpha = 0.001

    #TRAINING

    w , b, history = lr.gradient\_descent(X, Y, w\_init, b\_init,lr.compute\_cost, lr.compute\_gradient, alpha , iterations)

    #Predict Values

    return w, b, X, Y

**CÓDIGO FUENTE DE LA PARTE B:**

def our\_test\_B():

    #read data

    X , Y = readData("ex2data2.txt")

    #initial values

    b\_init = 1

    iterations = 10000

    alpha = 0.01

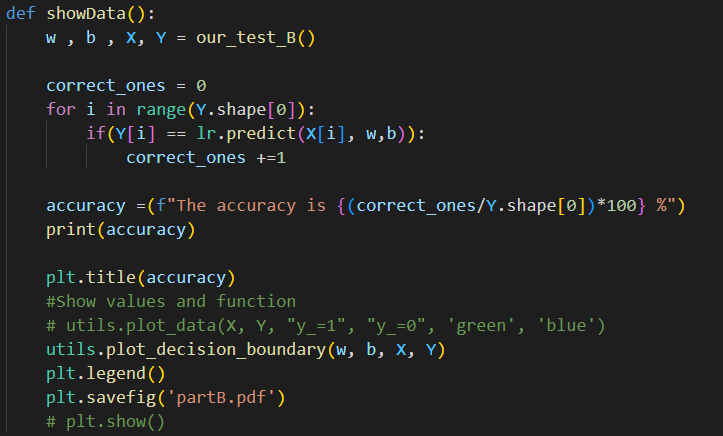
    X\_stack = utils.map\_feature(X[:,0],X[:,1])

    w\_init = np.zeros(X\_stack.shape[1])

    w , b, history = lr.gradient\_descent(X\_stack, Y, w\_init, b\_init,lr.compute\_cost\_reg, lr.compute\_gradient\_reg, alpha , iterations)

    return w, b, X\_stack, Y

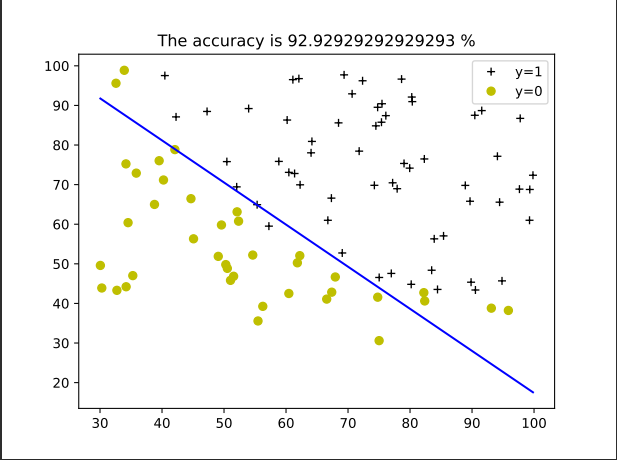
Ambas funciones son llamadas por:



La cual obtiene los datos y los representa en pantalla, a la vez que calcula la precisión de Theta y la variable independiente que predice los datos.

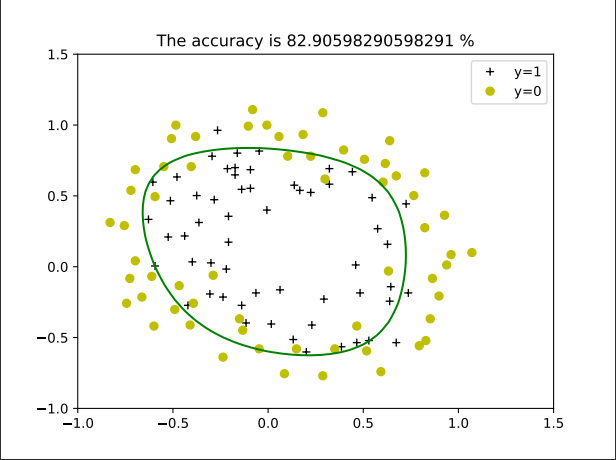
Ejecutando los datos almacenados en el archivo “ex2data1.txt”, obtenemos la gráfica de las notas con la recta de regresión:

**GRÁFICA A:**

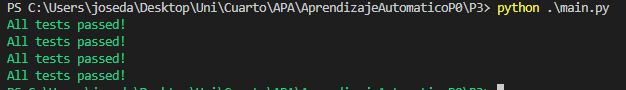
****

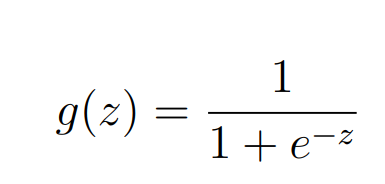
Ejecutando los datos de “ex2data2.txt”, obtenemos la siguiente gráfica:

**GRÁFICA B:**



Para comprobar que los métodos programados funcionan correctamente, pasamos estos por las funciones de testeo proporcionadas en el archivo public\_test.py.



**IMPLEMENTACIÓN**

La función sobre la cual realizamos el descenso es sigmoide (g).

def sigmoid(z):

    """

    Compute the sigmoid of z

    Args:

        z (ndarray): A scalar, numpy array of any size.

    Returns:

        g (ndarray): sigmoid(z), with the same shape as z

    """

    return  1/(1 + np.exp(-z))

Los métodos implementados para el Descenso de Gradiente de la recta de regresión son:

**Calcular el costo de la función**

def compute\_cost(X, y, w, b, lambda\_=None):

    total\_cost = 0

    m = y.shape[0]

    # fun\_val = fun\_wb(X,w, b)

    # total\_cost = np.sum((y + np.log(fun\_val)) - (1 - y) \* np.log(1 - fun\_val))

    for i in range(m):

      fun\_val = fun\_wb(X[i],w, b)

      log\_cost = np.log(fun\_val)

      log\_cost2 = np.log(1 - fun\_val)

      total\_cost += (-y[i] \* log\_cost) -  (1 - y [i]) \* log\_cost2

    return (total\_cost/m)

**Calcular el gradiente**

def compute\_gradient(X, y, w, b, lambda\_=None):

    dj\_dw = 0

    dj\_db = 0

    m = y.shape[0]

    #en teoria hay que hacer un doble for

    for i in range(m):

      dj\_dw += (fun\_wb(X[i],w,b) - y[i]) \* X[i]

      dj\_db += (fun\_wb(X[i],w,b) - y[i])

    return dj\_db / m, dj\_dw / m

Por otro lado, los métodos implementados para el la regresión logística son:

**Calcular el costo logístico de la función**

def compute\_cost\_reg(X, y, w, b, lambda\_=1):

    total\_cost = 0

    m = y.shape[0]

    for i in range(m):

      fun\_val = fun\_wb(X[i],w, b)

      log\_cost = np.log(fun\_val)

      log\_cost2 = np.log(1 - fun\_val)

      total\_cost += (-y[i] \* log\_cost) -  (1 - y [i]) \* log\_cost2

    total\_cost = total\_cost / m

    w\_cost = np.sum(w\*\*2)

    w\_cost = (w\_cost \* lambda\_) / (2 \* m)

    total\_cost += w\_cost

    return total\_cost

**Calcular el gradiente logístico**

def compute\_gradient\_reg(X, y, w, b, lambda\_=1):

    dj\_db = 0

    dj\_dw = 0

    m = y.shape[0]

    for i in range(m):

      dj\_db += fun\_wb(X[i], w , b) - y[i]

      dj\_dw += (fun\_wb(X[i], w, b) - y[i]) \* X[i]

    return dj\_db / m, (dj\_dw / m) + (np.dot(np.divide(lambda\_,m),w))

Como el descenso de gradiente es común para ambos ya que recibe las funciones a ejecutar, el código es el siguiente:

**Descenso de Gradiente**

def gradient\_descent(X, y, w\_in, b\_in, cost\_function, gradient\_function, alpha, num\_iters, lambda\_=None):

    J\_history = []

    w = copy.deepcopy(w\_in)

    b = b\_in

    for i in range(num\_iters):

      dj\_db, dj\_dw = gradient\_function(X, y, w, b)

      w -= alpha \* dj\_dw

      b -= alpha \* dj\_db

      if i < 100000:

        cost = cost\_function(X,y,w,b)

        J\_history.append(cost)

    return w, b, J\_history

Y por último, el método que nos permite predecir los valores, dándonos 1 si la función devuelve un valor mayor que 0.5, siendo 0.5 un umbral de prueba.

**Predicción**

def predict(X, w, b):

    """

    Predict whether the label is 0 or 1 using learned logistic

    regression parameters w and b

    Args:

    X : (ndarray Shape (m, n))

    w : (array\_like Shape (n,))      Parameters of the model

    b : (scalar, float)              Parameter of the model

    Returns:

    p: (ndarray (m,1))

        The predictions for X using a threshold at 0.5

    """

    if(fun\_wb(X, w, b)> 0.5):

      return 1

    return 0