Jose Daniel Rave Robayo
Ángel López Benítez

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])
```

Jose Daniel Rave Robayo
Ángel López Benítez

```
    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
```

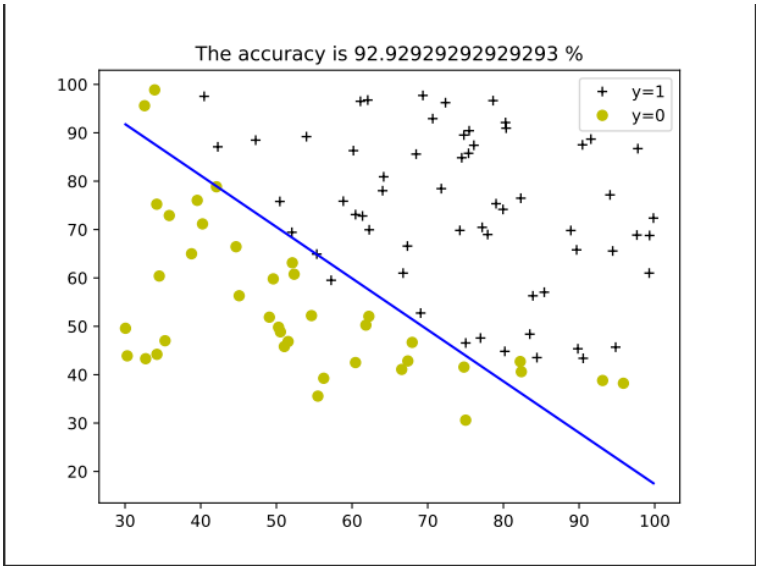
Ambas funciones son llamadas por:

```
def showData():
    w , b , X, Y = our_test_B()

    correct_ones = 0
    for i in range(Y.shape[0]):
        if(Y[i] == lr.predict(X[i], w,b)):
            correct_ones +=1

    accuracy =(f"The accuracy is {(correct_ones/Y.shape[0])*100} %")
    print(accuracy)

    plt.title(accuracy)
    #Show values and function
    # utils.plot_data(X, Y, "y_=1", "y_=0", 'green', 'blue')
    utils.plot_decision_boundary(w, b, X, Y)
    plt.legend()
    plt.savefig('partB.pdf')
    # plt.show()
```

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:

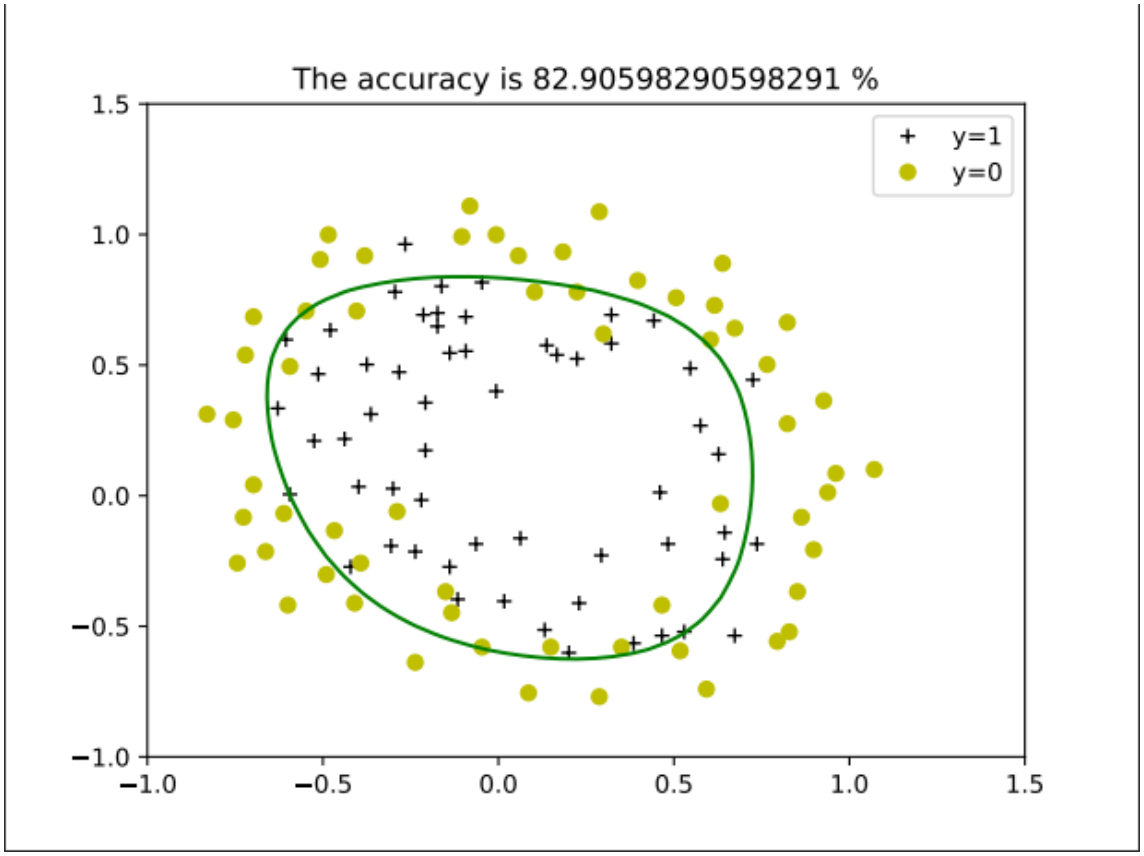
GRÁFICA A:



Jose Daniel Rave Robayo
Ángel López Benítez

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.

```
PS C:\Users\joseda\Desktop\Uni\Cuarto\APA\AprendizajeAutomaticoP0\P3> python .\main.py
All tests passed!
All tests passed!
All tests passed!
All tests passed!
```

IMPLEMENTACIÓN

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

$$g(z) = \frac{1}{1 + e^{-z}}$$

```python
def sigmoid(z):
    """
    Compute the sigmoid of z

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

    Returns:
```

Jose Daniel Rave Robayo
Ángel López Benítez

```
        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
```

Jose Daniel Rave Robayo
Ángel López Benítez

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))
```

Jose Daniel Rave Robayo
Ángel López Benítez

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
```