UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA

Informe Laboratorio 9

Regresión Logística

UNIVERSITARIA: Ortube Rengel Erika Mariana

CARRERA: Ingeniería de Sistemas

MATERIA: Inteligencia Artificial I (SIS420)

DOCENTE: Ing. Carlos W. Pacheco Lora

SUCRE – BOLIVIA

Regresión Logística

Se recopilaron datos de un *dataset* sobre sobre calificaciones dadas por dos jueces a participantes de salto en agua (clavado).

Lectura de datos

```
#Cargar y leer datos
#Las dos primeras columnas contienen la nota de dos examenes y la tercera columna
#contiene la etiqueta que indica si participante pasa a la siguiente ronda o no
data = np.loadtxt('/content/drive/MyDrive/regresion_logistica.txt', delimiter=',')
X, y = data[:, 0:2], data[:, 2]
print(X)
print(y)
```

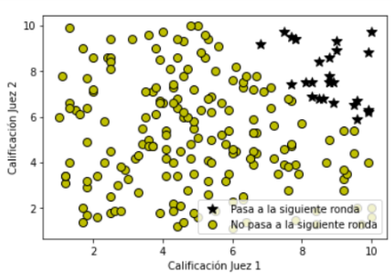
```
 [ 9.9  4. ]
 [ 2.5  8.1]
 [ 8.   5.7]
 [ 6.   1.1]
 [ 4.2  8.1]
 [ 1.8  3.2]
 [ 1.2  3.4]
 [ 7.1  6.4]
 [ 2.1  3.4]
 [ 4.6  6.2]
 [ 3.7  4.7]
 [ 5.8  3.2]
 [ 2.5  1.8]
 [ 4.1  5.2]
 [ 4.7  8.1]
 [ 4.8 10. ]
 [ 6.   2.3]
 [ 5.5  8. ]
 [ 5.6  9.1]
 [ 1.1  7.8]
 [ 6.6  5.9]
 [ 6.3  2.5]
 [ 7.2  4.2]
 [ 3.3  4.3]
```

```
[0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 1. 0.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Definición de función para graficar los datos

```
#definicion de funcion que grafica los puntos de datos X y y en una nueva figura
#grafica los puntos de datos con '*' para los positivos (pasa a la siguiente ronda) y 'o' para los negativos (no pasa a la siguiente ronda)
def plotData(X, y):
    # crea una nueva figura
    fig = pyplot.figure()

    #encuentra índices de ejemplos positivos y negativos
    pos = y == 1
    neg = y == 0

    #ejemplos de Plot
    pyplot.plot(X[pos, 0], X[pos, 1], 'k*', lw=2, ms=10)
    pyplot.plot(X[neg, 0], X[neg, 1], 'ko', mfc='y', ms=8, mec='k', mew=1)
```

```
plotData(X, y)
#adiciona etiquetas para los ejes
pyplot.xlabel('Calificación Juez 1')
pyplot.ylabel('Calificación Juez 2')
pyplot.legend(['Pasa a la siguiente ronda', 'No pasa a la siguiente ronda'])
pass
```



Definición de función para calcular la sigmoide

```
#defincion de funcion que calcular la sigmoide de una entrada z
def sigmoid(z):
    #convierte la intrada a un arreglo numpy
    z = np.array(z)

    g = np.zeros(z.shape)

    g = 1 / (1 + np.exp(-z))

    return g
```

```
#una prueba para la implementacion de la funcion sigmoid
z = 21
g = sigmoid(z)

print('g(', z, ') = ', g)
```

```
g( 21 ) =  0.9999999992417439
```

Siempre va a devolver 1 o 0, nunca mayor.

Agregando una columna de 1s en la misma cantidad de m (número de ejemplos) y los coloca delante de las x.

```
#configuracion de la matriz adecuadamente, y agregar una columna de unos que corresponde al termino de intercepción.
m, n = X.shape
# Agraga el termino de intercepción a A
X = np.concatenate([np.ones((m, 1)), X], axis=1)
```

Cálculo del costo

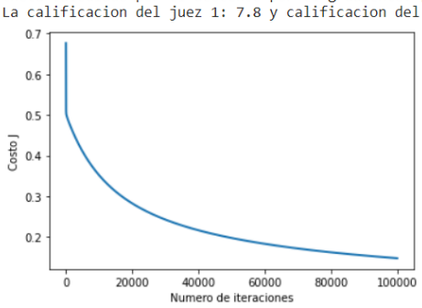
```
#definicion de funcion para calcular el costo
def calcularCosto(theta, X, y):
    # Inicializar algunos valores utiles
    m = y.size  # numero de ejemplos de entrenamiento
    J = 0
    h = sigmoid(X.dot(theta.T))
    J = (1 / m) * np.sum(-y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h)))
    return J
```

Calculo Descenso por el Gradiente

```
#defincion de funcion para calcular el descenso por el gradiente
def descensoGradiente(theta, X, y, alpha, num_iters):
    # Inicializa algunos valores
    m = y.shape[0] # numero de ejemplos de entrenamiento

    # realiza una copia de theta, el cual será acutalizada por el descenso por el gradiente
    theta = theta.copy()
    J_history = []

    for i in range(num_iters):
        h = sigmoid(X.dot(theta.T))
        theta = theta - (alpha / m) * (h - y).dot(X)

        J_history.append(calcularCosto(theta, X, y))
    return theta, J_history
```

```
# Elegir algun valor para alpha (probar varias alternativas)
alpha = 0.003
num_iters = 100000

# inicializa theta y ejecuta el descenso por el gradiente
theta = np.zeros(3)
theta, J_history = descensoGradiente(theta, X, y, alpha, num_iters)

# Grafica la convergencia del costo
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Numero de iteraciones')
pyplot.ylabel('Costo J')

# Muestra los resultados del descenso por el gradiente
print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta)))

# verificar si ingresa o no a la universidad
X_array = [1, 7.8, 6.1]
aprueba = sigmoid(np.dot(X_array, theta))   # Se debe cambiar esto

print('La calificacion del juez 1: 7.8 y calificacion del juez 2: 6.1 para un participante (usando el descenso por el gradiente) es: {:.0f}'.format(aprueba))
```

```
theta calculado por el descenso por el gradiente: [-8.94462762  0.66519498  0.44882404]
La calificacion del juez 1: 7.8 y calificacion del juez 2: 6.1 para un participante (usando el descenso por el gradiente) es: 0
```

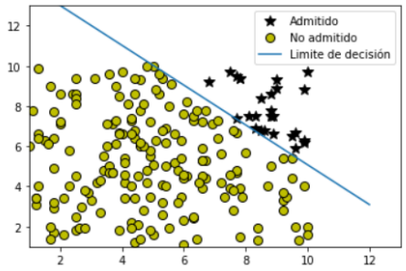


Se va haciendo una prueba con diferentes calificaciones de los jueces para un participante, y esto devuelve si con dichas calificaciones el participante pasa a la siguiente ronda (que es un 1) o si no pasa a la siguiente ronda (que es 0).

Gráfico de límites

```
#Graficar limites
plotDecisionBoundary(plotData, theta, X, y)
```

```
[-1. 12.]
[15.98622629  3.09251397]
```



Predicción

```
#Predice la probabilidad de:
#pasar a la siguiente ronda para un participante con una calificacion del juez 1 de 9.1
#pasar a la siguiente ronda para un participante con una calificacion del juez 2 de 8.8
prob = sigmoid(np.dot([1, 9.1, 8.8], theta))
print('Para un paritipante con calificaciones de: juez 1: 9.1 y juez 2: 8.8, se predice una probabilidad de pasar a la siguiente ronde de: {:.3f}%'.format(prob))
print('Valor esperado: 0.775 +/- 0.002\n')

# Compute accuracy on our training set
p = predict(theta, X)
print('Precisión de entrenamiento: {:.2f} %'.format(np.mean(p == y) * 100))
print('Precisión esperada (aproximadamente): 89.00 %')
```

```
Para un paritipante con calificaciones de: juez 1: 9.1 y juez 2: 8.8, se predice una probabilidad de pasar a la siguiente ronde de: 1.000%
Valor esperado: 0.775 +/- 0.002

Precisión de entrenamiento: 100.00 %
Precisión esperada (aproximadamente): 89.00 %
```

Se va haciendo una prueba con diferentes calificaciones de los jueces para un participante, y esto devuelve si con dichas calificaciones el participante tiene probabilidad de pasar a la siguiente ronda.