UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA

Informe Laboratorio 8

Regresión Polinomial

UNIVERSITARIA: Ortube Rengel Erika Mariana

CARRERA: Ingeniería de Sistemas

MATERIA: Inteligencia Artificial I (SIS420)

DOCENTE: Ing. Carlos W. Pacheco Lora

SUCRE – BOLIVIA

Regresión Polinomial

Se recopilaron datos de un *dataset* sobre sobre diferentes tipos de Bubble Tea, más específicamente de 56 tipos.

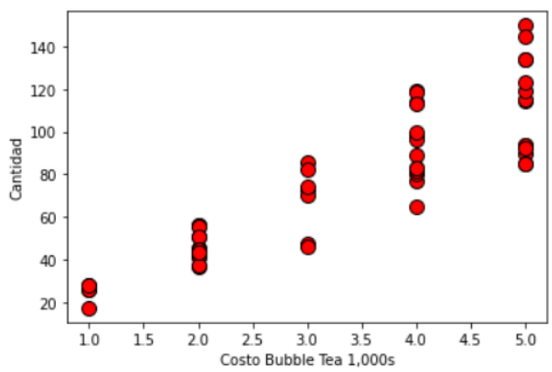
Lectura de datos

```
#Cargar y leer datos
data = np.loadtxt('/content/gdrive/MyDrive/regresion_polinomial.csv', delimiter=";",skiprows=1)
#print(data)
X = data[:, :1]
y = data[:, 1]
m = y.size

for i in range(26):
    print('{:8.0f}{:10.0f}'.format(X[i, 0], y[i]))
```

```
       2        36
       5       114
       4       119
       5        94
       2        43
       3        73
       1        17
       2        56
       5       115
       2        51
       4       114
       2        41
       5       150
       3        47
       2        42
       4        98
       4       118
       2        45
       2        45
       3        70
       2        56
       1        26
       5       119
       4        77
       5       134
       5       145
```

Definición de función para graficar los datos

```
#definicion de funcion que grafica los puntos x & y
def plotData(x, y):
    #abre una nueva figura
    fig = pyplot.figure()

    pyplot.plot(x, y, 'ro', ms=10, mec='k')
    pyplot.ylabel('Cantidad')
    pyplot.xlabel('Costo Bubble Tea 1,000s')
```

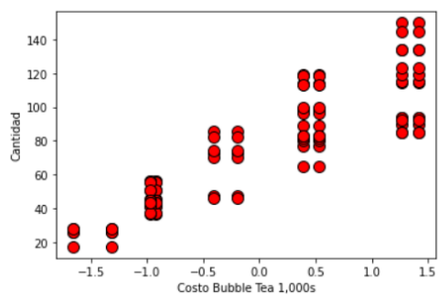


Definición de función para calcular la desviación estándar

```
#definicion de funcion que calcula la desviacion estandar
def  featureNormalize(X):
    X_norm = X.copy()
    mu = np.zeros(X.shape[1])
    sigma = np.zeros(X.shape[1])

    mu = np.mean(X, axis = 0)
    sigma = np.std(X, axis = 0)
    X_norm = (X - mu) / sigma

    return X_norm, mu, sigma
```

```
#llama featureNormalize con los datos cargados
X_norm, mu, sigma = featureNormalize(X)
```

```
plotData(X_norm, y)
```



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

```
#Añade el termino de interseccion a X
#(Columna de 1's para X0)
X = np.concatenate([np.ones((m, 1)), X_norm], axis=1)
print(X)
```

```
[[ 1.         -0.92599069 -0.9736597 ]
 [ 1.          1.26508587  1.41678666]
 [ 1.          0.53472702  0.39230965]
 [ 1.          1.26508587  1.41678666]
 [ 1.         -0.92599069 -0.9736597 ]
 [ 1.         -0.19563184 -0.4045058 ]
 [ 1.         -1.65634954 -1.31515203]
 [ 1.         -0.92599069 -0.9736597 ]
 [ 1.          1.26508587  1.41678666]
 [ 1.         -0.92599069 -0.9736597 ]
 [ 1.          0.53472702  0.39230965]
 [ 1.         -0.92599069 -0.9736597 ]
 [ 1.          1.26508587  1.41678666]
 [ 1.         -0.19563184 -0.4045058 ]
 [ 1.         -0.92599069 -0.9736597 ]
 [ 1.          0.53472702  0.39230965]
 [ 1.          0.53472702  0.39230965]
 [ 1.         -0.92599069 -0.9736597 ]
 [ 1.         -0.92599069 -0.9736597 ]
 [ 1.         -0.19563184 -0.4045058 ]
```

Cálculo del costo

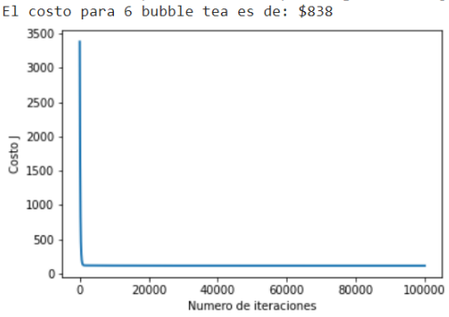
```
#definicion de funcion para calcular el costo
def computeCostMulti(X, y, theta):
    #Inicializa algunos valores que seran utiles
    m = y.shape[0] #numero de ejemplos de entrenamiento
    J = 0
    h = np.dot(X, theta)
    J = (1/(2 * m)) * np.sum(np.square(np.dot(X, theta) - y))
    return J
```

Calculo Descenso por el Gradiente para encontrar los valores de theta que permitan minimizar aún más el costo

```
#definicion de funcion para calcular el descenso por el gradiente
def gradientDescentMulti(X, y, theta, 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):
        theta = theta - (alpha / m) * (np.dot(X, theta) - y).dot(X)
        J_history.append(computeCostMulti(X, y, theta))
    return theta, J_history
```

```
[116] #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 = gradientDescentMulti(X, y, theta, 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)))

      #El costo para 6 bubble tea
      X_array = [6, 20.8, 136]
      X_array[1:3] = (X_array[1:3] - mu) / sigma
      price = np.dot(X_array, theta)   # Se debe cambiar esto

      print('El costo para 6 bubble tea es de: ${:.0f}'.format(price))
```

```
theta calculado por el descenso por el gradiente: [75.1525     33.23122738 -2.71545658]
El costo para 6 bubble tea es de: $838
```



Gráfico

```
plotData(X[:, 1], y)
pyplot.plot(X[:, 1], np.dot(X, theta), '-')
```

```
[<matplotlib.lines.Line2D at 0x7f0f0ac0e2d0>]
```