Exercício de Programação 1
Regressão Linear

Sheldon Goulart de Alcântara
alcantarash92@gmail.com
Vinícius Teodoro
vtcpires@gmail.com

Abstract

This report aims to explain the linear regression and the equations used to obtain the results proposed in the exercise.

1 Regressão Linear

Modelos de regressão são modelos matemáticos que relacionam o comportamento de uma variável Y com uma variável X.Quando a função **f** que relaciona duas variáveis é do tipo f(X) = a + bX temos o modelo de regressão simples. A variável X é a variável independente da equação, enquanto Y = f(X) é dependente das variações de X. O modelo de regressão é chamado de simples quando envolve uma relação casual entre duas variáveis. O modelo de regressão é multiplo quando envolve uma relação causal com mais de duas variáveis. Isto é, quando o comportamento de Y é explicado por mais de uma variável independe X1, X2,..., Xn.

Os modelos acima (simples ou multivariados) simulam relacionamentos entre as variáveis. Esse relacionamento poderá ser do tipo linear (equação da reta ou do plano) ou não linear (equação exponencial, geométrica, etc.). A análise de regressão compreende, portanto quatro tipos básicos de modelos:

- Linear simples.
- Linear multivariado.
- Não linear simples.
- Não linear multivariado.

1.1 Modelos de regrssão linear

Regressão é o processo matemático pelo qual derivamos os parâmetros 'a' e 'b' de uma função f(X). Estes parâmetros determinam as características da função que relaciona 'Y' com 'X'que no caso do modelo linear se representa por uma reta chamada de reta de regressão. Esta reta explica de forma geral e teoricamente a relação entre X e Y. Isto significa que os valores observados de X e Y nem sempre serão iguais aos valores de X'e Y' estimados pela reta de regressão. Haverá sempre alguma diferença, e essa diferença significa:

- As variações de Y não são perfeitamente explicadas pelas variações de X.
- Existem outras variáveis das quais Y depende.
- Os valores de X e Y são obtidos de uma amostra específica que apresenta distorções em relação a realidade.

Esta diferença em estatística é chamada de erro ou desvio.

O processo de regressão significa, portanto, traçar uma reta com menor distâncias entre os pontos plotados no gráfico,ou seja, diminuir a diferença entre Y e Y'.

$$y_0 = a_0 + b_0.x_0$$

equação da reta a partir dados coletados

$$y_1 = +b_1.x_1$$

equação da reta a partir das estimativas

2 Função de custo

A função de custo (ou função de erros) é a funcão que define a qualidade da linha de regressão. Essa função recebe um par ordenado (β_0, β_1) e retorna um valor de erro baseado no quão bem a reta se adequa aos dados a serem analisados. Ao utilizar o método da regressão linear temos por objetivo minimizar a função de custo:

$$J(\beta) = \frac{1}{2m}\sum_{i=1}^{n}(h_\beta(x_i - y_i))^2 \tag{1}$$

Onde o modelo hbeta é dado pela reta:

$$h_\beta(x) = \beta^T x = \beta_0 + \beta_1 x_1 \tag{2}$$

Na implementação em python utilizamos de operações matriciais da biblioteca NumPy para evitar a criação de loops desnecessários. A implementação da função de erros pode ser vista no Algoritmo 1 (compute cost):

```
def compute_cost(X, y, beta):
    lengthX = 2 * len(X)
    funct = np.power(((X * beta.T) - y), 2)
    return  np.sum(funct) / lengthX
```

Listing 1: Algoritmo 1 ComputeCost

3 Método Gradiente

O método do gradiente é usado em otimização. Para encontrar um mínimo (local) de uma função usa-se um esquema iterativo, onde em cada passo se toma a direção (negativa) do gradiente, que corresponde à direção de declive máximo.

3.1 Descrição

Começando com um vetor inicial x_0 visando alcançar um ponto minimo de **f**, consideramos a sucessão definida por:

$$x_0, x_1, x_2, ..., x_n$$

onde a pesquisa linear é dada pela direção de descida d_n

$$x_n + 1 = x_n + \omega_n.d_n$$

No caso do método do gradiente a condição de descida verifica-se tomando

$$d_n = -\nabla F(x_n)$$

ficando a iteração definida por

$$x_n + 1 = x_n - \omega\nabla F(x_n)$$

O algoritmo utilizado para a implementação da função Gradiente Descendente é relativamente simples. Dado um ponto inicial da superfície da função que se deseja minimizar ele encontra o sentido

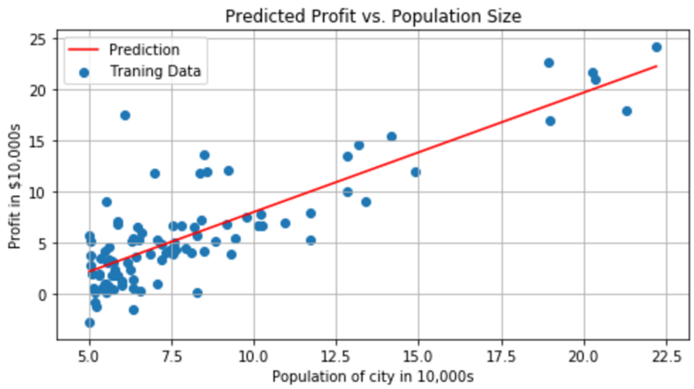

Figure 1: Função J(β)representada graficamente

de crescimento da função e se move em sua direção, segundo um dado parâmetro alfa (α), até atingir um ponto de convergência.

$$\beta_j = \beta_j - \alpha \frac{1}{n} \sum_{i=1}^{n} (h_\beta(x_i - y_i)) x_i j \tag{3}$$

```
def gradient_descent(X, y, theta, alpha, iters):
    temp        = np.matrix(np.zeros(theta.shape)) #np.zero: Return a new
    array of given shape and type, filled with zeros.
    parameters = int(theta.ravel().shape[1]) #Return a contiguous
    flattened array and return a tuple of array dimensions axis 1
    cost        = np.zeros(iters)
    n = len(X)
    for i in range(iters):
        for j in range(parameters):
            gradient = np.multiply((X.dot(theta.T) - y),X[:,j])
            temp[0,j] -= (alpha/(n)) * np.sum(gradient)

        theta = temp #update
        cost[i] = compute_cost(X, y, theta)
    return theta, cost
```

Listing 2: Algoritmo 2 GradientDescent

Pode-se observar no algoritmo 2 (GradientDescent), a implementação em python dessa funç ao. Note que as variáveis temp0 e temp1 são necessárias para que os parâmentros (β_0, β_1) sejam atualizados de forma simultânea. De modo contrário o cálculo dos parâmetros podem se influenciar mutuamente, gerando um resultado incorreto.

4 Regressão Linear Multipla

As funções desenvolvidas nesse trabalho podem ser usadas tanto para uma regresão linear simples (uma variável), como para regresões lineares múltiplas. Para demonstrar isso utilizaremos o DataSet sobre preços de imóveis como exemplo.

Inicialmente pode-se observar a relação entre o número de quartos e o tamanho do imóvel com seu preço plotando os dados do dataset em um gráfico de dispersão. Porém, antes de mais nada, note que como a dimensão das variáveis é bem diferente. Comparando os tamanhos das casas com seu número de quartos e o preço, pode-se ver como a discrepância entre esses números é grande. Para solucionar esse problema utilizou-se um método chamado '**feature normalization**'. Ele consiste em subtrair de cada atributo, o valor da média de sua coluna, e dividí-lo pelo seu desvio padrão. Desta forma podemos ajustar as dimensões utilizadas.

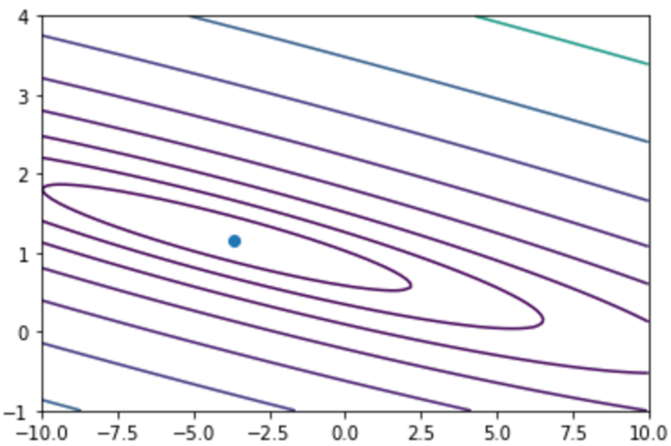

Figure 2: Gráfico do dataset com as dimensões devidamente ajustadas

Após isso podemos comparar o número de iterações de nosso algoritmo com os valores retornados de nossa função de custos

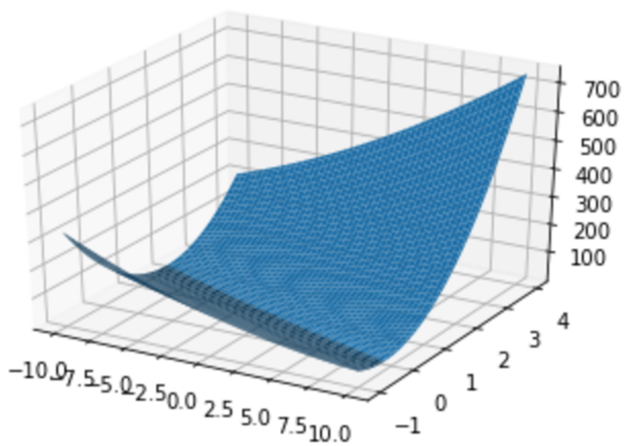

Figure 3: Número de iterações de nosso algoritmo com os valores retornados da função da custos

Pode-se visualizar todo o código de estudo utilizado em algoritmo 3 (Code Data2) e um exemplo do mundo real no algoritmo 4 (Code Groupon) :

```
# Load dataset
data2 = pd.read_csv('ex1data2.txt', header=None, names=['Size', 'Bedrooms', 'Price'])

# Executing the 'feature normalization'
data2 = (data2 - data2.mean()) / data2.std()
data2.head()

data2.insert(0, 'beta zero', 1)

# set X (training data) and y (target variable)
cols = data2.shape[1]
x = data2.iloc[:,0:cols]
```

```
y = data2.iloc[:,cols-1:cols]

x = np.matrix(x.values)
y = np.matrix(y.values)
beta2 = np.matrix(np.array([0,0,0]))
g2, cost2 = gradient_descent(x, y, beta2, alpha, iters)

x = np.linspace(data2.Size.min(), data2.Size.max(), 100)
f = g2[0, 0] + (g2[0, 1] * x) + (g2[0, 2] * x)

fig, ax = plt.subplots(figsize=(8,4))
ax.plot(x, f, 'r', label='Prediction')
ax.scatter(data2.Size, data2.Price, label='Traning Data2')
ax.legend(loc=2)
ax.set_xlabel('Size')
ax.set_ylabel('Price')
ax.set_title('Predicted Price vs. Size')
ax.grid(True)

fig, ax = plt.subplots(figsize=(12,8))
ax.plot(np.arange(iters), cost2, 'r')
ax.set_xlabel('Iterations')
ax.set_title('Error vs. Training Epoch')
ax.grid(True)


beta0_vals = np.linspace(-10, 10, 100)
beta1_vals = np.linspace(-1, 4, 100)

for i in range(len(beta0_vals)):
    for j in range(len(beta1_vals)):
        t = np.matrix(np.array([beta0_vals[i], beta1_vals[j]]))
        j_vals[i,j] = compute_cost(X, y, t)


plt.scatter(g2[0,0],g2[0,1],)
plt.contour(beta0_vals, beta1_vals, j_vals.T, np.logspace(-2, 3, 20));

beta0_mesh, beta1_mesh = np.meshgrid(beta0_vals, beta1_vals)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(beta0_mesh, beta1_mesh, j_vals.T);
```

Listing 3: Code Data2

```
# Load dataset
data3 = pd.read_csv('groupon-deals.csv', header=0)

# Make qualitative variables quantitative
data3['city'] = pd.Categorical(data3.city).codes
data3['category'] = pd.Categorical(data3.category).codes
data3['weekday_start'] = pd.Categorical(data3.weekday_start).codes


x = pd.concat([data3.iloc[:,1:3], data3.iloc[:,11:15]], axis=1)
x.insert(0, 'beta zero', 1)
y = data3.iloc[:,5:6]

x = np.matrix(x.values)
y = np.matrix(y.values)
theta = np.zeros(shape=(1,7))

# Plot cost function results
alpha = 1.00
iters = 100
```

```
theta, J_history = gradient_descent(x, y, theta, alpha, iters)
plt.plot(np.arange(iters), J_history)
plt.xlabel('Iterations')
plt.ylabel('Cost Function')
plt.show()
```

Listing 4: Code Groupon