Trabalho prático 1 - Aprendizado de Maquina

Nome: Leonardo Botrel Tobias **Data:** 24/09/2018

Task 0:

Funcionamento do algoritmo gradient descent

O gradient decent é um algoritmo muito usado em problemas de regressão, pois permite atualizar os parâmetros de uma função a fim de minimizar a diferença entre uma hipótese e um conjunto de treinamento.

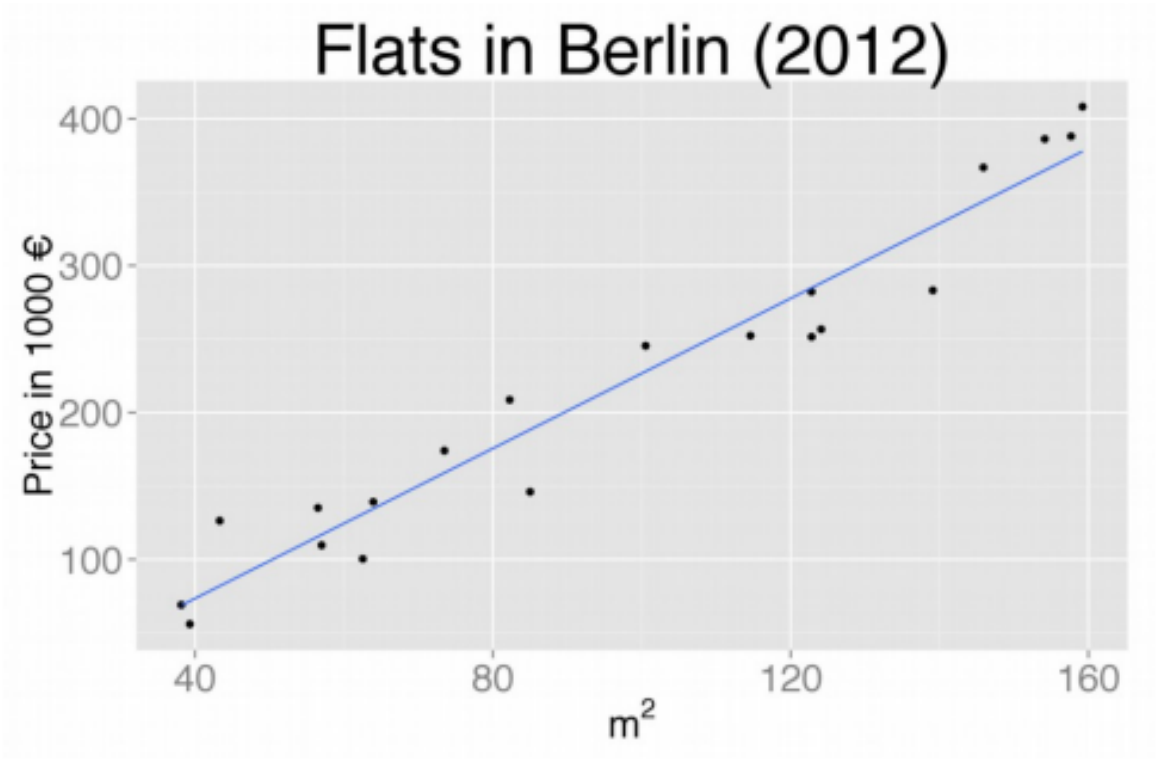
Um bom exemplo de uso desse algoritmo é o problema de prever o preço de uma casa com base na área dela. Supondo que queremos queiramos uma curva linear para prever o preço das casas, teremos uma hipótese do tipo $\boldsymbol{y} = b + \boldsymbol{x}w + \boldsymbol{e}$ onde w e b são os parâmetros que queremos prever e **x** é o conjunto de tamanho das casas. O objetivo é encontrar os valores de b e w que nos permita uma curva semelhante a da figura abaixo:

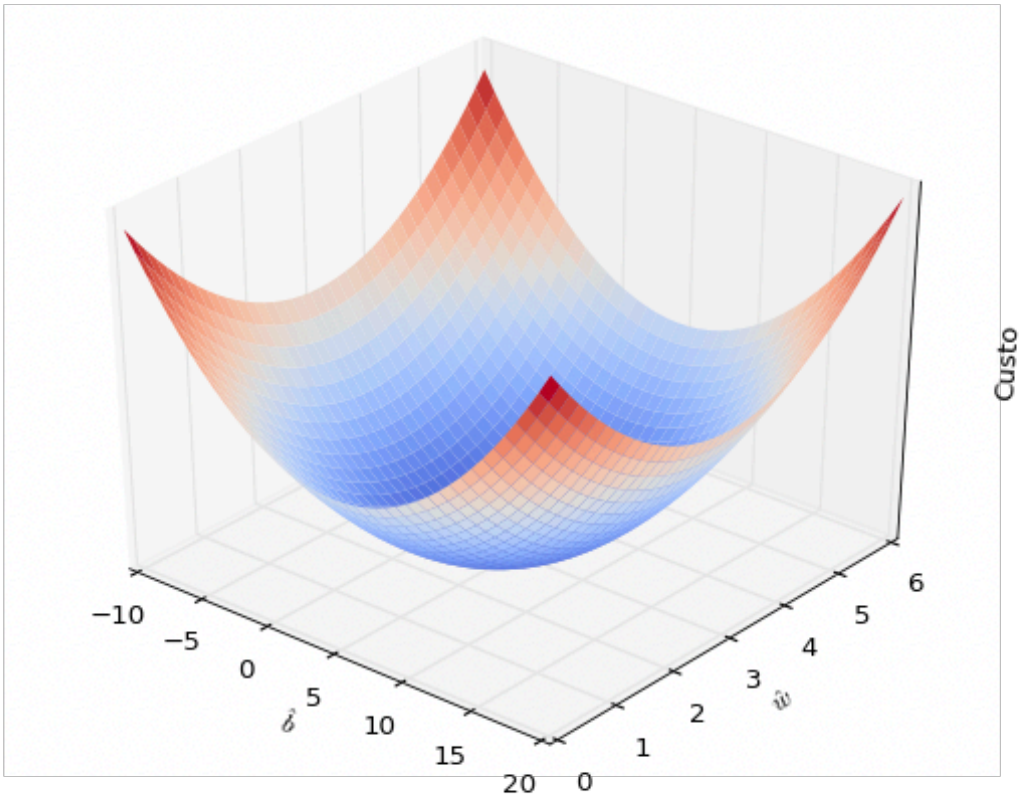
Queremos achar os valores de b e w que minimizam o quadrado da norma do vetor **e**, ou seja, minimiza a soma dos quadrados dos resíduos. Dessa forma garantimos que nossa função custo tenha apenas um mínimo e evitamos de cair em um mínimo local e não global.

Encontrar o menor valor dessa função significa encontrar valores de b e w que nos permita ter uma hipótese mais confiável. Vamos usar o gradient descent para fazer isso.

Entendendo e visualizando o gradient descent

Para melhor entendimento do algoritmo, é bom visualizar nossa função custo quando plotada nas duas dimensões dos parâmetros b e w que queremos aprender:





O gradiente dessa função é simplesmente um vetor de derivadas parciais, que dão a inclinação em cada ponto e em cada direção:

Se nos seguirmos na direção oposta do gradiente, então chegaremos ao ponto de mínimo.

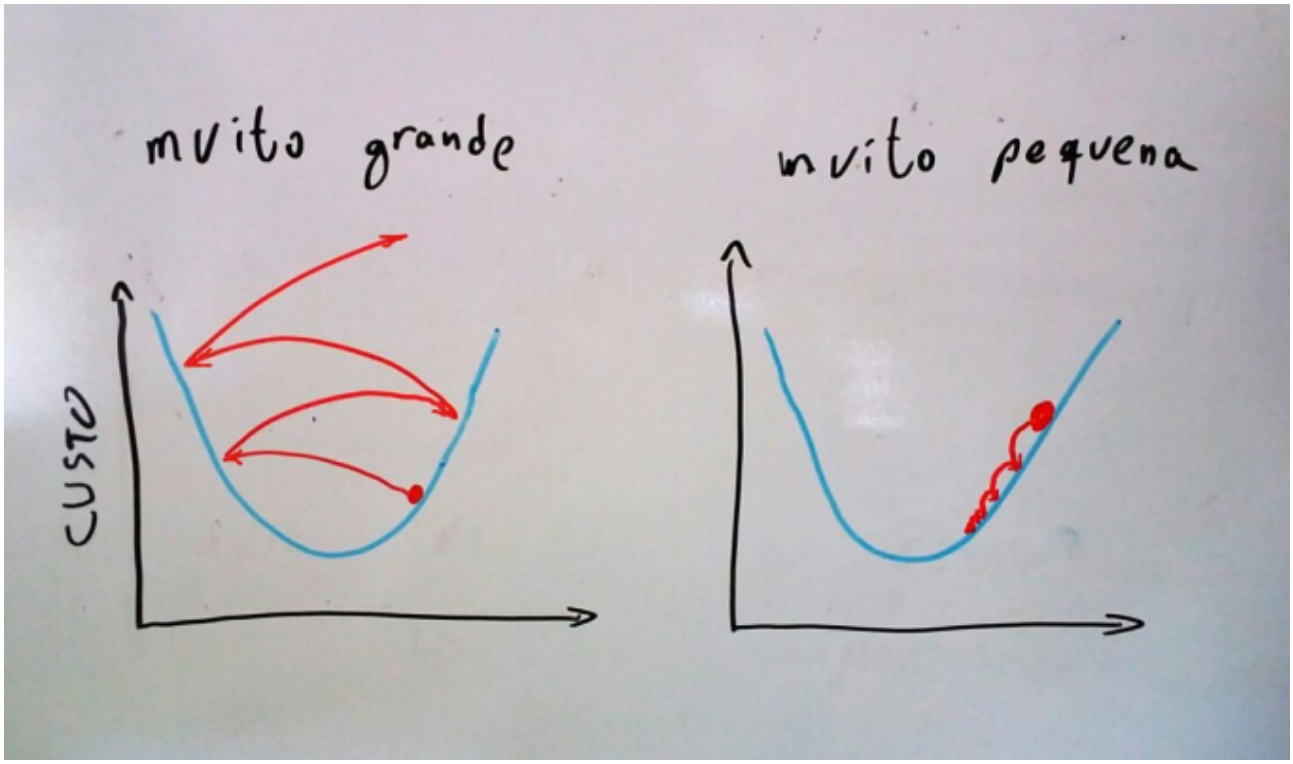
A cada iteração os parâmetros b e w dão um passo em direção ao mínimo, o tamanho desse passo será o valor do gradiente naquele ponto multiplicado pela constante α. Quando mais próximos estamos do ponto mínimo, menor é a inclinação do custo, menor é o gradiente e consequentemente menor é o passo em direção ao mínimo.

Hiper-parâmetros

Diferente dos parâmetros w, que são aprendidos durante o treinamento de uma regressão linear, os hiper-parâmetros não são aprendidos pela máquina durante o treinamento e devem ser ajustados manualmente. Temos três hiper-parâmetros: **A taxa de aprendizado, O número de iterações de treino e os valores iniciais de w.** Como a função custo é convexa, os valores iniciais de w não são muito importantes desde que os outros parâmetros estejam ajustados corretamente.

A **taxa de aprendizado** define o tamanho dos passos que daremos em direção ao mínimo em cada iteração. Se os passos forem pequenos, é muito provável que chegaremos bem próximo do mínimo, porém mais iterações serão necessárias e consequentemente o algoritmo ficará mais lento. Porém, por outro lado, se os passos forem muito grandes corre o risco de nos afastarmos do mínimo, já que do ponto inicial, podemos chegar em um ponto custo mais alto, nesse ponto o gradiente será ainda maior fazendo com que o passo seja ainda maior, dessa forma afastando mais ainda do mínimo.

$$\nabla(L) = \left[\frac{\partial L}{\partial b}, \frac{\partial L}{\partial w} \right]$$



Com uma boa taxa de aprendizado, é fácil selecionar o **número de iterações de treino.** Pode ocorrer de haver tantas iterações que o custo cai, como iterações que o custo sobe. Se a função custa flutua muito a cada iteração, recomenda-de diminuir a taxa de aprendizado. Se a função custo desce suavemente, mas muito devagar, recomenda-se aumentar a taxa de aprendizado.

Referências

• Slides da aula
• https://matheusfacure.github.io/2017/02/20/MQO-Gradiente-Descendente/
• https://oliveiratiano.wordpress.com/2016/07/31/aprendizado-de-maquina-na-pratica-1-

entendendo-o-algoritmo-mestre/

Task 1:

```
def compute_cost(X, y, beta):
    total_cost = 0
    m, _ = X.shape
    for i in range(m):
        x_i = X[i, 1]
        y_i = y[i]
        b_0 = beta[0, 0]
        b_1 = beta[0, 1]
        total_cost += (b_0 + b_1 * x_i - y_i) ** 2

    return total_cost / (2 * m)
```

Task 2:

```
def gradient_descent(X, y, theta, alpha, iters):
    '''
    alpha: learning rate
    iters: number of iterations
    OUTPUT:
    theta: learned parameters
    cost:  a vector with the cost at each training iteration
    '''
    temp       = np.matrix(np.zeros(theta.shape))
    parameters = int(theta.ravel().shape[1])
    cost       = np.zeros(iters)

    for i in range(iters):
        error = (X * theta.T) - y

        for j in range(parameters):
            term = np.multiply(error, X[:,j])
            temp[0,j] = theta[0,j] - ((alpha / len(X)) * np.sum(term))

        theta = temp
        cost[i] = compute_cost(X, y, theta)



    return theta, cost
```

Task 3:

gradient_descent(X, y, beta, alpha, iters):

matrix([[-1.10856950e-16, 8.84042349e-01, -5.24551809e-02]])

compute_cost(X, y, g):

matrix([[0.13204955]])

Task 4:

gradient_descent(X, y, beta, alpha, iters):

```
matrix([[ 1.32799776e+00,  5.86748183e-02,  1.06511876e+00,
          1.24084439e-02,  5.62026942e-01,  7.10297099e-02,
          2.78890936e-01,  8.48868572e-02,  1.78595436e-01,
          8.23904830e-02,  2.42925788e-01,  1.96349372e-01,
          2.63958885e-01,  1.82208931e-01,  6.43685123e-02,
         -4.01865369e-02, -5.19296870e-02, -9.26992876e-03,
          4.25910281e-01, -9.49981226e-02, -6.86428911e-02,
         -5.10782876e-04,  4.89746011e-02,  2.40221807e-02,
          1.13236752e-02, -4.32706903e-02,  1.72768716e-03,
          1.00308742e-03,  6.09790869e-01,  2.87416161e-01,
         -1.99395883e-02,  8.92170281e-02,  8.35592328e-02,
          1.05547253e-01,  3.78660785e-02,  3.08646635e-02,
          9.83717047e-02,  1.13151525e-01,  1.20023544e-01,
          1.57248874e-02,  7.62599472e-02, -2.69545371e-02,
          3.13359870e-02, -8.74911100e-02, -1.11654632e-01,
          1.54293922e-01,  8.61507145e-02,  9.00084919e-02,
          5.19671189e-02,  3.42949845e-02,  3.35460955e-01]])
```

compute_cost(X, y, g):

matrix([[9.31904825]])

Dificuldades encontradas

A primeira tarefa pedida foi explicar como o Gradient Descent funciona. Minha principal dificuldade nessa parte não foi entender o que o Gradiente Descent faz ou como ele funciona, mas entender os fundamentos matemáticos por trás desse algoritmo.

Nas tarefas seguintes, de implementar a função para calcular custo e implementar o Gradient Descent, novamente minha principal dificuldade foi implementar os cálculos realizados por essas funções, além de entender exatamente o papel de cada parâmetro

passados para as funções. Superando essas dificuldades, a tarefa de implementar uma regressão com varias variáveis foi bem simples.

Por fim tive muita dificuldade na regressão usando os dados do Groupon. A primeira dificuldade foi entender o que foi feito no artigo e quais os dados eles usaram. Depois de entender isso, tive que separar os dados certos para serem usados na regressão, depois transformar os dados categóricos que eram strings em dummy vectors. Essa parte foi um pouco difícil principalmente porque encontrei pouca informação na internet de como fazer isso. Outra dificuldade que encontrei nessa parte foi transformar alguns dos dados em logaritmo, como no artigo, pois existiam 0's na tabela, a solução para isso foi atribuir o valor 0 onde o log tinha dado o valor de -infinito. Depois de tudo isso feito, bastou apenas calcular os valores, o que levou um bom tempo na parte de calcular o gradiente.