Regressão Múltipla

Retomando o caso da rede de cientista de dados, considere que você também coletou dados adicionais: para cada um dos seus usuários, você sabe quantas horas trabalha todos os dias e se ele tem um PhD. Você gostaria de usar esses dados adicionais para melhorar seu modelo.

Assim, você hipotetiza um modelo linear com mais variáveis independentes:

$$\text{minutes} = \alpha + \beta_1 \text{friends} + \beta_2 \text{work hours} + \beta_3 \text{phd} + \varepsilon$$

Obviamente, se um usuário tem um PhD não é um número, mas podemos introduzir uma variável dummy (fictícia) que é igual a 1 para usuários com PhDs e 0 para usuários sem, após isso é tão numérico quanto os outros variáveis.

In [1]:
```
from collections import Counter
import matplotlib.pyplot as plt
import random
import math
import numpy as np
import gradient_descent as gd
```

In [2]:
```
num_friends = [10, 49, 41, 40, 25, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4
               15, 15, 15, 18, 20, 20]

daily_min = [18, 39, 37, 35, 28, 7, 9, 8, 7, 8, 10, 11, 12, 9, 13, 15, 14,
               25, 27, 29, 28, 30, 32]

work_hours = [8, 6, 6, 6, 6, 9, 10, 10, 12, 11, 10, 9, 9, 10, 9, 9, 8, 8, 8
               7, 6, 6, 6, 6, 6]

has_phd = [0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1
```

In [3]:

```
def mean(x):
    return sum(x) / len(x)

def de_mean(x):
    x_bar = mean(x)
    return [x_i - x_bar for x_i in x]

def dot(v, w):
    return sum(v_i * w_i
        for v_i, w_i in zip(v, w))

def sum_of_squares(x):
    return sum([x_i * x_i for x_i in x])

def variance(x):
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations) / (n - 1)

def standard_deviation(x):
    return math.sqrt(variance(x))

def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n - 1)

def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y) / stdev_x / stdev_y
    else:
        return
```

O modelo

Ajustamos o modelo para a seguinte forma:

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

Agora imagine que cada entrada *Xi* não é um único número, mas sim um vetor de números *Xi1,...,Xik*.
O modelo de regressão múltipla assume que:

$$y_i = \alpha + \beta_1 x_{i1} + \ldots + \beta_k x_{ik} + \varepsilon_i$$

Na regressão múltipla, o vetor de parâmetros é geralmente chamado **Beta**. Também queremos que incluir no vetor o termo constante alfa, o que podemos conseguir adicionando uma coluna de dados aos nossos dados:

In [ ]:

```
beta = [alpha, beta_1, ..., beta_k]
```

E ...

In [ ]:
```
x_i = [1, x_i1, ..., x_ik]
```

Então, nosso modelo é apenas:

In [5]:
```
def predict(x_i, beta):
    """assumes that the first element of each x_i is 1"""
    return dot(x_i, beta)
```

Neste caso particular, nossa variável independente x será uma lista de vetores, cada um deles com a seguinte aparência:

In [6]:
```
[1,  # constant term
 49, # number of friends
 4,  # work hours per day
 0]  # doesn't have PhD
```

Out[6]:
```
[1, 49, 4, 0]
```

In [7]:
```
x = [[1, x1, x2, x3] for x1, x2, x3 in
     zip(num_friends, work_hours, has_phd)]
```

In [9]:
```
x[:3]
```

Out[9]:
```
[[1, 10, 8, 0], [1, 49, 6, 0], [1, 41, 6, 1]]
```

Suposições Adicionais do Modelo dos Mínimos Quadrados

Existem algumas outras suposições que são necessárias para este modelo (e nossa solução) para fazer sentido.

A primeira é que as colunas de **X** são linearmente independentes, ou seja, não há correlação entre as variáveis de modo que alterar o valor de uma delas não afetará o valor da outra.

A segunda suposição importante é que as colunas de x são todas não correlacionadas com os erros. Se isso não ocorrer, nossas estimativas de beta serão sistematicamente erradas. O aumento no valor do erro não pode representar um aumento em qualquer uma das variáveis independentes.

Ajustando o modelo

Como fizemos no modelo linear simples, escolheremos **beta** para minimizar a soma dos erros quadrados. Encontrar uma solução exata não é simples de fazer à mão (testando repetidamente diferentes combinações de valores), o que significa que precisaremos usar gradiente descendente. Começaremos criando uma função de erro para minimizar. Para o gradiente descendente estocástico, queremos apenas que o erro ao quadrado corresponda a uma única previsão:

In [10]:
```
def error(x_i, y_i, beta):
    return y_i - predict(x_i, beta)

def squared_error(x_i, y_i, beta):
    return error(x_i, y_i, beta) ** 2

def squared_error_gradient(x_i, y_i, beta):
    """the gradient (with respect to beta) corresponding to the ith squared
    return [-2 * x_ij * error(x_i, y_i, beta) for x_ij in x_i]
```

Neste momento, estamos prontos para encontrar o beta ideal usando o gradiente descendente estocástico:

In [11]:
```
def estimate_beta(x, y):
    beta_initial = [random.random() for x_i in x[0]]
    return gd.minimize_stochastic(squared_error,
                                  squared_error_gradient,
                                  x, y,
                                  beta_initial,
                                  0.001)
```

In [12]:
```
random.seed(0)
beta = estimate_beta(x, daily_min) # [30.63, 0.972, -1.868, 0.911]
```

In [13]:
```
beta
```

Out[13]:
```
[37.666692222532504,
 0.3976700117030098,
 -2.8057270545719035,
 -0.6046244866262608]
```

Isso significa que nosso modelo se parece com:

$$\text{minutes} = 30.63 + 0.972\ \text{friends} - 1.868\ \text{work hours} + 0.911\ \text{phd}$$

Interpretando o modelo

Você deve pensar nos coeficientes do modelo como representando estimativas de todos os demais como sendo iguais para os impactos de cada fator. Sendo o restante igual, cada amigo adicional corresponde a um minuto extra gasto no site todos os dias. Sendo o restante igual, cada hora adicional na jornada de trabalho de um usuário corresponde a cerca de dois minutos a menos gastos no site todos os dias. Tudo o mais sendo igual, ter um PhD está associado a gastar um minuto extra no site todos os dias.

O que isso não nos diz (diretamente) é nada sobre as interações entre as variáveis. É possível que o efeito das horas de trabalho seja diferente para pessoas com muitos amigos do que para pessoas com poucos amigos. Este modelo não captura isso. Uma maneira de lidar com esse caso é introduzir uma nova variável que é o produto de "amigos" e "horas de trabalho". Isso efetivamente permite que o coeficiente de "horas de trabalho" aumente (ou diminua) à medida que o número de amigos aumenta.

Ou é possível que quanto mais amigos você tiver, mais tempo você gasta no site até certo ponto, após o qual mais amigos farão com que você gaste menos tempo no site (talvez com muitos amigos a experiência seja muito grande demais?) Poderíamos tentar capturar isso em nosso modelo adicionando outra variável que é o quadrado do número de amigos.

Uma vez que começamos a adicionar variáveis, precisamos nos preocupar se os seus coeficientes "são importantes". Não há limites para o número de produtos, logs, quadrados e poderes superiores que poderíamos adicionar.

Qualidade do ajuste

In [14]:
```
def total_sum_of_squares(y):
    """the total squared variation of y_i's from their mean"""
    return sum(v ** 2 for v in de_mean(y))


def multiple_r_squared(x, y, beta):
    sum_of_squared_errors = sum(error(x_i, y_i, beta) ** 2
                                for x_i, y_i in zip(x, y))
    return 1.0 - (sum_of_squared_errors / total_sum_of_squares(y))
```

In [15]:
```
print("R2 = ", multiple_r_squared(x, daily_min, beta))
```

```
R2 =  0.9519361468476204
```

Atividade 1

desenvolva um algoritmo baseado na técnica **Regressão Linear Múltipla** que seja capaz de fazer a **predição do valor de um imóvel** tendo como base uma de suas características.

Para isso leve em consideração a base de dados sobre preços de casas da **Kaggle**, cuja composição está descrita abaixo: **Columns**

- id
- date
- price (Valor do imóvel em dolares)
- bedrooms (quantidade de quartos)
- bathrooms (quantidade de banheiros)
- **sqft_living (área útil em pés quadrado)**
- sqft_lot (área do lote em pés quadrado)
- floors (pisos)
- waterfront
- view
- condition
- grade
- **sqft_above (área acima do porão em pés quadrado)**
- sqft_basement

- yr_built
- yr_renovated
- zipcode
- lat
- long
- sqft_living15
- sqft_lot15

Para mais detalhes sobre a base de dados acesse: https://www.kaggle.com/arejet/simple-linear-regression

Observações:

- Encontre o **alpha** e o **beta**
- Exiba o R2 da função
- Considere os atributos que tenha mais influenciam no preço, ou seja, aqueles de melhor correlação com o preço.
- Use o algorítmo de otimização do Gradiente Descendente

Resposta

Usando o SKLearn

Atividade 2

Repita a atividade 1 agora usando a biblioteca SKLearn e exiba apenas o Score

Resposta