Notes

Dr. Muhammad Usman Shahid Khan

November 20, 2019

Abstract

Question Answers

1 What is multiple linear regression? and how it is different from simple linear regression?

In simple linear regression a single independent variable is used to predict the value of a dependent variable. In multiple linear regression two or more independent variables are used to predict the value of a dependent variable. The difference between the two is the number of independent variables. In both cases there is only a single dependent variable.

Similar to Simple Linear Regression, we have input variable(X) and output variable(Y). But the input variable has nn features. Therefore, we can represent this linear model as follows;

$$Y = \beta_0 * x_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_n * x_n \tag{1}$$

where $x_0 = 1$.

This equation in matrix form can be presented as:

$$Y = \beta^T X \tag{2}$$

2 What is the cost function in multiple linear regression?

The cost function for multiple linear regression is:

$$J(\beta) = \frac{1}{m} \sum_{i=0}^{m} (Y'^i - Y^i)^2 \tag{3}$$

where m is the number of entries, Y'^i is the predicted Y for the ith entry. Y^i is the actual output.

3 What is the Gradient decent algorithm for multiple linear regression?

I Initialize values $\beta_0, \beta_1, \beta_n$ with some value. In this case we will initialize with 0.

II Iteratively update,

$$\beta_j = \beta_j - \alpha * \frac{\partial}{\partial \beta_j} J(\beta) \tag{4}$$

This operation $\frac{\partial}{\partial \beta_j} J(\beta)$ means we are finding partial derivative of cost with respect to each β_j. This is called Gradient The above equation can be written as:

$$\beta_j = \beta_j - \alpha * (\frac{2}{m}) \sum_{i=0}^{m} (y'^i - y^i)(x_j^i) \tag{5}$$

4 Python code for linear regression

The python code is shown below

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (20.0, 10.0)

data = pd.read_csv('student.csv')
print(data.shape)
data.head()

math = data['Math'].values
prog = data['Prog'].values
AI = data['AI'].values

m = len(math)
x0 = np.ones(m)
X = np.array([x0, math, prog]).T
# Initial Coefficients
B = np.array([0, 0, 0])
Y = np.array(AI)
alpha = 0.00001

def cost_function(X, Y, B):
        m = len(Y)
        J = np.sum((X.dot(B) - Y) ** 2)/(m)
        return J
```

```
inital_cost = cost_function(X, Y, B)
print(inital_cost)


m = len(Y)
epochs = 10000
for iteration in range(epochs):
        h = X.dot(B)

        loss = h - Y
        # Gradient Calculation
        gradient = X.T.dot(loss) *2 / m
        # Changing Values of B using Gradient
        B = B - alpha * gradient
        # New Cost Value
        cost = cost_function(X, Y, B)
        print (B,cost)

X1=np.array([1,45,51])
print X1.T.dot(B)
```