Tutorial 01

SCS3201 / IS3117 / CS3120 Machine Learning and Neural Computing

Linear Regression

In this tutorial, we will be discussing what linear regression is and how the gradient decent algorithm works. For that we are using a dataset consisting of **SAT scores** and **GPA**s of 84 students. SAT is a standardized test widely used for college admissions in the United States. Let's try to come with a linear relation between **SAT score** and the **GPA**.

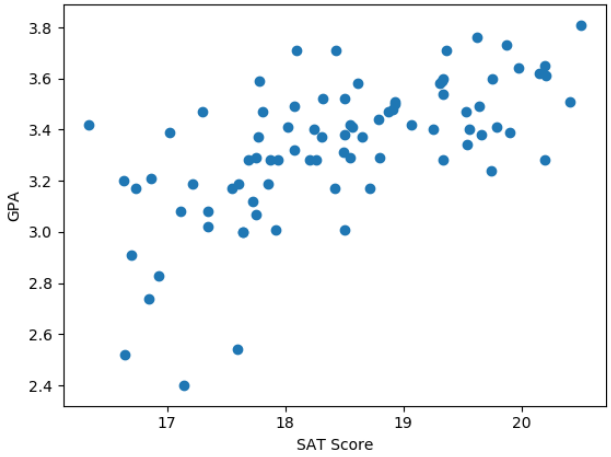
We'll be using python to code the model from scratch.

Download the dataset from here: https://bit.ly/rg-datset

Let's load the dataset as a numpy array. Then remove the header row. Next take SAT score as the X vector, and GPA as the Y vector. After that we'll visualize the data using matplotlib.

```
#import numpy and matplotlib
import numpy as np
import matplotlib.pyplot as plt

#load the dataset
train_data = np.genfromtxt('dataset.csv', delimiter=',')
#remove the header row
train_data = np.delete(train_data,0,0)

#take SAT Score as X vector
X = train_data[:,0]
#take GPA as Y vector
Y = train_data[:,1]

#visualize data using matplotlib
plt.scatter(X,Y)
plt.xlabel('SAT Score')
plt.ylabel('GPA')
plt.show()
```

Now you should be getting an output like this.



Our goal is to define a best fitting line to above data. Let's define the line as a linear relationship between these two variables.

$$\boldsymbol{Y = mX + c}$$

It's where if we fit a line to the data, **m** will be the slope(gradient) and **c** will be the intercept.

Loss Function

Error in our predicted value of **m** and **c** is the loss. We are trying to minimize this error to get the best fitting line.

Mean Squared Error function will be used here to calculate the loss. This function consits of three steps.

1. Calculate the difference between the predicted y value (y = mx+c) and the actual y, for a given x.
2. Square the difference.
3. Find the mean of the squares for every value in X.

This can be represented by the following equation.

$$Loss = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

Since $\bar{y}_i$ is the predicted value, we can substitute it with $mx_i + c$, which results in the following equation.

$$Loss = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

Let's minimalize the loss and find the best **m** and **c.**

The Gradient Decent Algorithm

Initially let's take **m = 0** and **c = 0.** And let **L** be the learning rate which controls how much the value of **m** and **c** changes with each step.

Then we'll take the partial derivatives of the loss function with respect to **m** and **c**. Let $\mathbf{D_m}$ and $\mathbf{D_c}$ be the partial derivatives of **m** and **c** respectively.

$$D_m = \frac{1}{n}\sum_{i=0}^{n}2(y_i - (mx_i + c))(-x_i) \quad = \frac{-2}{n}\sum_{i=0}^{n}x_i(y_i - \bar{y}_i)$$

$$D_c = \frac{1}{n}\sum_{i=0}^{n}2(y_i - (mx_i + c)) \quad = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)$$

Then in each step we update the value of **m** and **c** as follows.

$$m = m - L \times D_m \, , c = c - L \times D_c$$

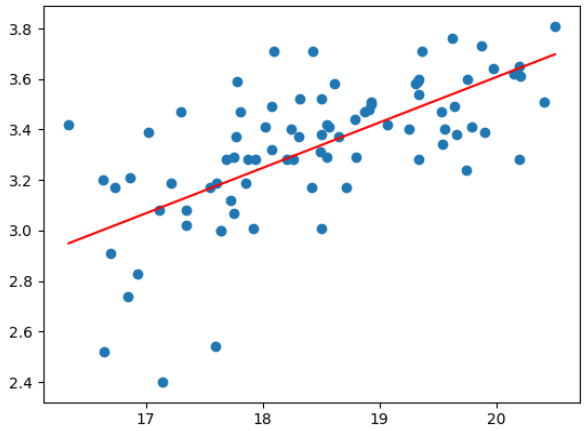
Then we'll repeat this process until loss is a very small value or ideally 0. The values of **m** and **c** will be left with the optimal values.

Let's get into business. Now we are going to code all these.

```
#initialize m and c
m = 0
c = 0

L = 0.0001  # The learning Rate
iterations = 1000  # The number of iterations to perform gradient descent

n = float(len(X)) # Number of elements in X

# Performing Gradient Descent
for i in range(iterations):
    Y_pred = m*X + c  # The current predicted value of Y
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative with respect to m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative with respect to c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c

#output m and c
print ("m = ",m)
print ("c = ",c)
```

Finally let's draw the best fitting line.

```
#Final predictions
Y_pred = m*X + c

#Draw the best fitting line
plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red')
plt.show()
```



What if there were more than one variable that decides the **GPA**? For an example, if **attendance** was also given in the dataset, you can create a linear relationship to **GPA** using both **SAT score** and **attendance**.

You can take SAT score as X_1 and attendance as X_2. So the linear relationship will look like this.

$$\boldsymbol{Y = m_1 X_1 + m_2 X_2 + c}$$

You can use the same gradient decent algorithm to find best fitting $\mathbf{m_1}$, $\mathbf{m_2}$ and **c.**

Use the following **Medical Cost Personal Dataset,** which is used for **Insurance Forecast** to try **Multivariate Linear Regression**.

Download the dataset from here: https://www.kaggle.com/mirichoi0218/insurance

****