 **GHARDA INSTITUTE OF TECHNOLOGY** 

***Department of Computer Engineering***

**Machine Learning Lab BE Computer (Semester-VII)**

**Experiment No.1 : Linear Regression**

**Aim**- To study, understand and implement a linear regression algorithm.

**Theory**-

Linear Regression comes under the category of supervised machine learning algorithms. In supervised learning when given a data-set, we already know what the correct output should look like, we already have an idea of the relationship between the input and the output. Supervised learning broadly covers two types of problems:

1. Regression problems
2. Classification problems

In simple words, regression problems try to predict results within a continuous output i.e they try to map input variables to some continuous function. The output here is a continuous set. It also helps to remember that when the target variable we are trying to predict is continuous (e.g. in mathematical sense [1,5] is a continuous set whereas {1,5} is discrete) then it is a regression problem.

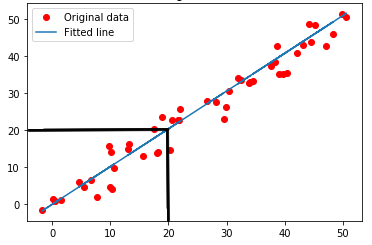


Fig.1:Linear Regression visual representation

In any supervised learning problem, our goal is simple:

*“Given a training set, we want to learn a function h: X →Y so that h(x) is a good prediction for the corresponding value of y“*

Here *h(x)* is called the hypothesis function and is basically what we are trying to predict through our learning algorithm i.e. Linear Regression.

For the case of univariate linear regression our hypothesis function is:

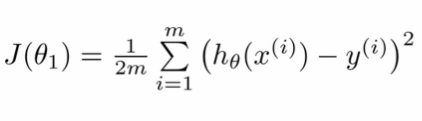
hypothesis equation

In the above equation, *θ0* and *θ1* are called the *parameters* of the hypothesis. It can be easily noticed that our equation for *h(x)* is actually just the mathematical equation for a line in a 2-dimensional plane and now we have seen that our hypothesis *h(x)* is in fact a line in the graphical sense as well hence the term *“linear”* regression.

**Cost Function**

It’s important to know how accurate our predictions are in order to know how well our model performs and if it needs further“training” or more “tuning” (which is basically adjustment of the parameters). This is where the cost function comes into the picture.

The cost function is an expression through which we evaluate the quality of our current hypothesis and proceed to make changes accordingly. In simpler words, our cost function decides the cost we want our model to incur depending on how far off our predictions are from the true value. It is only intuitive to think that the “cost” should in fact be the difference between our prediction and the true value i.e. *h(x)-y.* The cost function for linear regression is:



The main goal here is to minimize our cost function *J(θ)* so that we get *h(x)* as the function which passes through maximum points in the plot of X and Y or in other words we want to minimize the cost function so that the predictions of our model are as close as possible to the actual values.

The task is to find a line that fits best in the above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in a dataset). This line is called a regression line.

**Code -**

**\*\* Using Least Square Technique**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

n = np.size(x) # number of observations/points

m\_x = np.mean(x)

m\_y = np.mean(y) # mean of x and y vector

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x # calculating cross-deviation

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x # calculating deviation about x

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x # calculating regression coefficients

return (b\_0, b\_1)

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12]) # observations / data

b = estimate\_coef(x, y)

print("Estimated coefficients:”, b[0], b[1]))

y\_pred = b[0] + b[1]\*x # predicted response vector

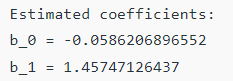
plt.scatter(x, y, color = "m")

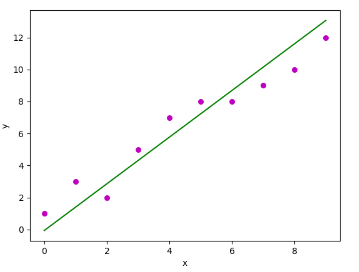
plt.plot(x, y\_pred, color = "g") # plotting the regression line

plot\_regression\_line(x, y, b)

plt.show() # function to show plot

**Output**

****

****

**\*\* Using Built-in functions and for standard dataset**

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model, metrics

# load the boston dataset

boston = datasets.load\_boston(return\_X\_y=False)

# defining feature matrix(X) and response vector(y)

X = boston.data

y = boston.target

# splitting X and y into training and testing sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4,

random\_state=1)

# create linear regression object

reg = linear\_model.LinearRegression()

# train the model using the training sets

reg.fit(X\_train, y\_train)

# regression coefficients

print('Coefficients: ', reg.coef\_)

# variance score: 1 means perfect prediction

print('Variance score: {}'.format(reg.score(X\_test, y\_test)))

# plot for residual error

## plotting residual errors in training data

plt.scatter(reg.predict(X\_train), reg.predict(X\_train) - y\_train,

color = "green", s = 10, label = 'Train data')

## plotting residual errors in test data

plt.scatter(reg.predict(X\_test), reg.predict(X\_test) - y\_test,

color = "blue", s = 10, label = 'Test data')

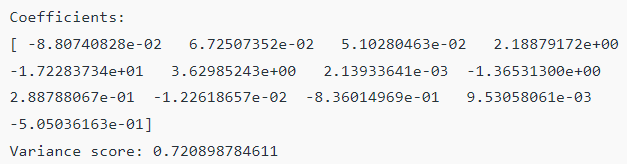
## plot title

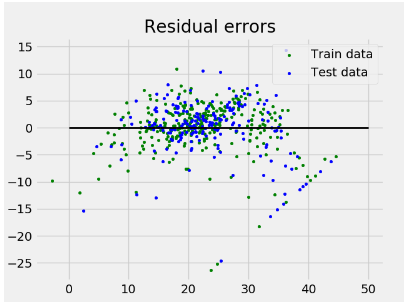
plt.title("Residual errors")

## method call for showing the plot

plt.show()

**Output**

****

****

**Results-**

**Assumptions-**

1. Linear Relationships
2. No multicolinearity
3. No autocorrelation
4. Homoscedasticity (same error for all values)

**Conclusion-**

The concept of linear regression is studied and implemented using least square method as well as python built-in functions with standard dataset.

**References-**

1. http://scikit-learn.org/stable/auto\_examples/linear\_model/plot\_ols.html
2. http://www.statisticssolutions.com/assumptions-of-linear-regression/