Chapter 2

**Simple Linear Regression**

Algorithms and codes for Simple Linear Regression

**2.1 What is Simple Linear Regression**

Simple Linear Regression is a type of Regression algorithms that models the *relationship between* a *dependent* *variable* and a single *independent* *variable*. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

= independent variable

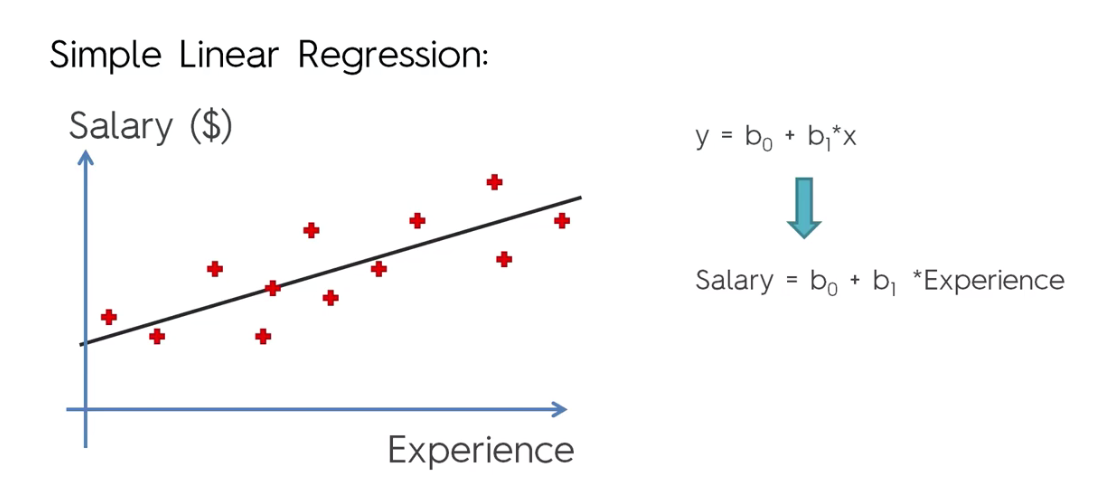
= dependent variable

= It is the intercept of the Regression line (can be obtained putting x=0) with y-axis. Eg: *Minimum salary for Fresher*.

= It is the slope of the regression line, which tells whether the line is increasing or decreasing.

= The error term. (For a good model it will be negligible)

* The key point in Simple Linear Regression is that the *dependent* *variable* must be a *continuous/real* value. However, the *independent* *variable* can be measured on *continuous* or *categorical* values.



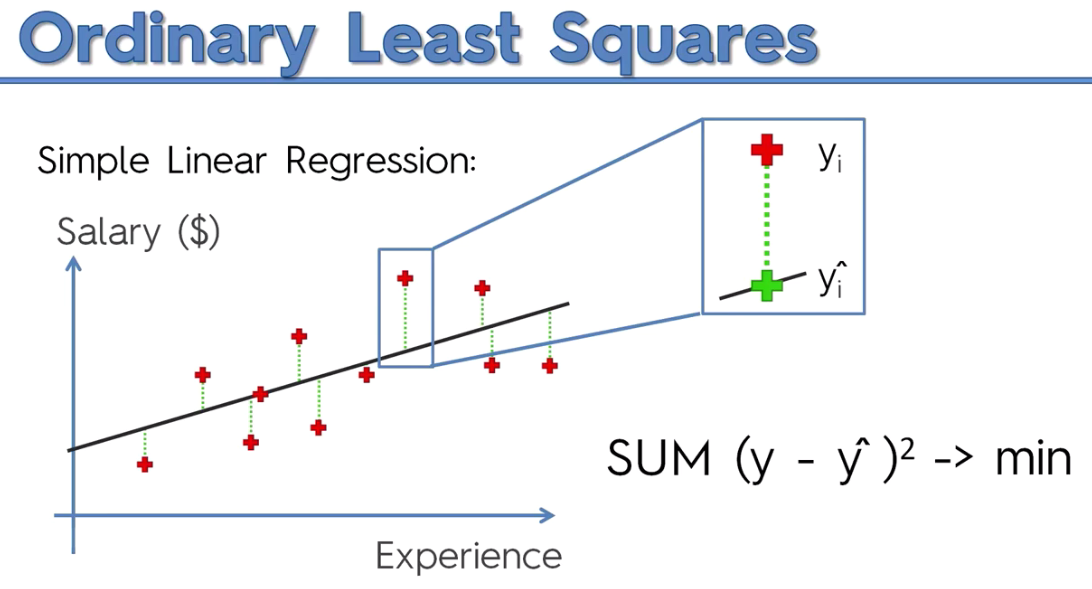
* Simple Linear regression algorithm has mainly two objectives:
* Model the relationship between the two variables: Such as the relationship between ***Income*** and ***expenditure***, ***experience*** and ***Salary***, etc.
* Forecasting new observations: Such as *Weather* *forecasting* according to temperature, *Revenue* of a company according to the investments in a year, etc.

**2.2 Finding the Best fitting line**

The algorithm finds a line for which

Where are the given points .

And are , where



**2.3 Implementation of Simple Linear Regression**

* Problem Statement example for Simple Linear Regression:
* Here we are taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable).
* The goals of this problem is:
* We want to find out if there is any *correlation* between these *two variables*
* We will find the *best fit line* for the dataset.
* How the *dependent variable* is changing by changing the *independent variable*.
* Create a Simple Linear Regression model: Now we will create a Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.
* To implement the Simple Linear regression model in machine learning using Python, we need to follow the below steps:

1. Step-1: Data Pre-processing

The first step for creating the Simple Linear Regression model is data pre-processing. We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

* First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

#*------------------- Importing the libraries -------------------*

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**import** pandas **as** pd

#*------------------- Importing the dataset -------------------*

dataset = pd**.read\_csv**('Salary\_Data.csv')

X = dataset**.**iloc[:, :-1]**.**values

y = dataset**.**iloc[:, 1]**.**values

#*-------------- Splitting the dataset into the Training set and Test set ----------------*

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

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size = 1/3, random\_state = 0)

* Next, we will load the dataset into our code:

dataset = pd**.read\_csv**('Salary\_Data.csv')

By executing the above line of code (ctrl+ENTER), we can read the dataset on our Spyder IDE screen by clicking on the variable explorer option.

* After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

X = dataset**.**iloc[:, :-1]**.**values

y = dataset**.**iloc[:, 1]**.**values

In the above lines of code, for **x** variable, we have taken **-1** value since we want to remove the last column from the dataset. For ***y*** variable, we have taken ***1*** value as a parameter, since we want to extract the second column and indexing starts from the zero.

* Next, we will split both variables into the test set and training set. We have ***30*** observations, so we will take ***20*** observations for the ***training*** set and ***10*** observations for the ***test*** set.
* We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

#*-------------- Splitting the dataset into the Training set and Test set ----------------*

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

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size = 1/3, random\_state = 0)

* Feature Scaling: For simple linear Regression, we will not use Feature Scaling. Because Python libraries take care of it for some cases, so we don't need to perform it here. Now, our dataset is well prepared to work on it and we are going to start building a Simple Linear Regression model for the given problem.

1. Step-2: Fitting the Simple Linear Regression to the Training Set:

* Now the second step is to fit our model to the training dataset. To do so, we will import the ***LinearRegression*** class of the ***linear\_model*** library from the scikit learn.
* After importing the class, we are going to create an object of the class named as a ***regressor***.
* And then we fit our data by using ***fit()*** method.

The code for this is given below:

#*---------  Fitting the Simple Linear Regression model to the Training dataset -----------*

**from** sklearn**.**linear\_model **import** LinearRegression   #*import library*

s\_l\_regressor = **LinearRegression**()                  #*regressor object*

s\_l\_regressor**.fit**(X\_train, y\_train)                 #*fit train data*

In the above code, we have used a ***fit()*** method to fit our Simple Linear Regression object to the training set. In the ***fit()*** function, we have passed the ***x\_train*** and ***y\_train***, which is our training dataset for the dependent and an independent variable. We have fitted our ***regressor*** object to the training set so that the model can easily learn the correlations between the ***predictor*** and ***target variables***. After executing the above lines of code, we will get the below output.

1. Step: 3. Prediction of test set result:

In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not. We will create a prediction vector y\_pred,

#*Predicting the test set results. Test set will be used*

y\_pred = s\_l\_regressor**.predict**(X\_test)

* You can check the variable by clicking on the variable explorer option in the IDE, and also compare the result by comparing values from ***y\_pred*** and ***y\_test***. By comparing these values, we can check how good our model is performing.
* We will create a vector of predictive values called ***y\_pred*** that will contain the predictions of the *test set salaries* and this is going to be the same for all machinery models that we will create. ***y\_pred*** is going to be the *vector* of *predictions* of the *dependent* *variable*.

1. Step: 4. visualizing the Training set results:

Now in this step, we will visualize the training set result. To do so, we will use the ***scatter()*** function of the ***pyplot*** library, which we have already imported in the pre-processing step. The ***scatter()*** function will create a scatter plot of observations.

#*visualising the Training-set result*

plt**.scatter**(X\_train, y\_train, color = "red")

plt**.plot**(X\_train, s\_l\_regressor**.predict**(X\_train), color = "blue")   #*notice the Train set is used*

plt**.title**("Salary vs Experience 'Trainig Set'")

plt**.xlabel**("Years of Experience")

plt**.ylabel**("Salary")

plt**.show**()

* The good fit of the line can be observed by calculating the ***difference between actual values and predicted values (difference of the corresponding y-values)***. But as we can see in the above plot, most of the observations are close to the regression line, hence our model is good for the training set.

1. Step: 5. visualizing the Test set results:

In the previous step, we have visualized the performance of our model on the training set. Now, we will do the same for the Test set. The complete code will remain the same as the above code, except in this, we will use ***x\_test***, and ***y\_test*** instead of ***x\_train*** and ***y\_train***.

* Here we are also changing the color of observations and regression line to differentiate between the two plots, but it is optional.

#*visualising the Test-set result*

plt**.scatter**(X\_test, y\_test, color = "red")

plt**.plot**(X\_train, s\_l\_regressor**.predict**(X\_train), color = "blue")   #*notice the Train set is used again*

plt**.title**("Salary vs Experience 'Trainig Set'")

plt**.xlabel**("Years of Experience")

plt**.ylabel**("Salary")

plt**.show**()

Notice:

* plt**.plot**(X\_train, s\_l\_regressor**.predict**(X\_train), color = "blue")remain same for both plot. Because it is the same regression-line.
* Here ***s\_l\_regressor.predict(x)*** is the equation (function) of regression line. As we can see, most of the observations are close to the regression line.

**2.4 Idea behind making a ML model**

Our model will *learn the correlations between* the *x\_train* and *y\_train* so that it can later predict the results of the *dependent variable* the *salaries* based on these information here.

* Once our model is trained, we will test its performance (its power of prediction) on the test-set.
* For example, here we will predict the corresponding salary and then we will compare the predicted salary to the real salary which are the salaries on the test-set.
* So here the machine is the *simple linear regression model* and learning is the fact that this *model learns on the training set* here composed of ***x\_train*** and ***y\_train***.
* We made this machine to learn on the *training set* to understand the *correlations* between the *experience* and the *salary* so that this machine based on its learning experience (on train-dataset) can then predict the *salary* with respect to the *experience* on *test-dataset*.

That's what machine learning is about.

**2.5 FAQ**

* Why do we take the squared differences and simply not the absolute differences?

Because the *squared differences* makes it easier to *derive a regression line*. Indeed, *to find that line we need to compute the* ***first******derivative*** *of the* ***loss******error******function***, and it is much harder to compute the derivative of absolute values than squared values.

* Why didn’t we apply Feature Scaling in our Simple Linear Regression model?

It’s simply because since ***y*** is a ***linear*** ***combination*** of the ***independent*** ***variables***, the coefficients can *adapt their scale* to put everything on the same scale. For example if you have two independent variables ***x1*** and ***x2*** and if ***y*** takes values between ***0*** and ***1***, ***x1*** takes values between ***1*** and ***10*** and ***x2*** takes values between ***10*** and ***100***, then ***b1*** can be multiplied by ***0.1*** and ***b2*** can be multiplied by ***0.01*** so that ***y***, ***b1x1*** and ***b2x2*** are all on the same scale.

* What does ***’regressor.fit(X\_train, y\_train)’*** do exactly?

The **fit** method will take the values of and and then will compute the coefficients and of the Simple Linear Regression equation (). That’s the whole purpose of this fit method here.