CS5542 Big Data Apps and Analytics

In Class Programming –9 Report
(Jongkook Son)

Project Overview:

Create a linear regression model in python using any dataset of your choice. For this model you can also create your own data. Find the best fit line in the data and calculate SSE (sum of square error) or MSE (Mean square error) , Y intercept, and Slope for the relationship in data. Explain your findings and understanding of these terms in detail in the report.

Requirements/Task(s):

1) Successfully executing the code with linear regression model and calculating following: (75 points)
a. SSE or MSE
b. Y intercept
c. Slope
2) Detail explanation of each in report (5 points)
3) overall code quality (10 points)
4) Pdf Report quality, video explanation (10 points)

What I learned in ICP:

I could have learned how to get the best fit linear line for the data set. Previously I simply used a scikit learn's predefined model So I don't need to think about the formula or principe to build this line. But by doing this ICP. I basically build this line from scratch. So that makes me think about the process to get the best fit line and the principle of linear regression model.

ICP description what was the task you were performing and Screen shots that shows the successful execution of each required step of your code

1. **Implement with python scratch**

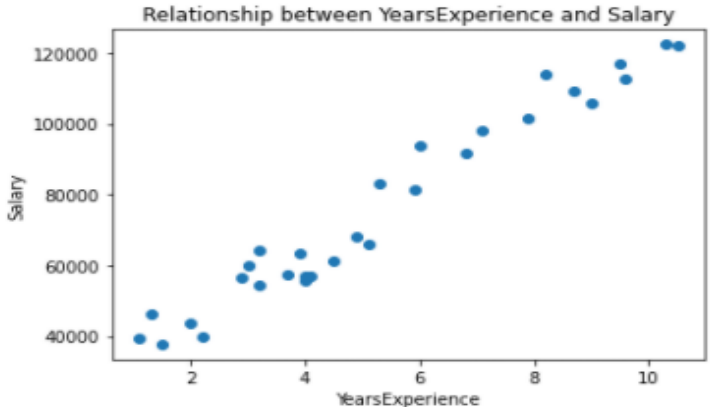
   Use a data set which contains salary and yearsExperience.

```
#Lets look into top few rows and columns in the dataset
df.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

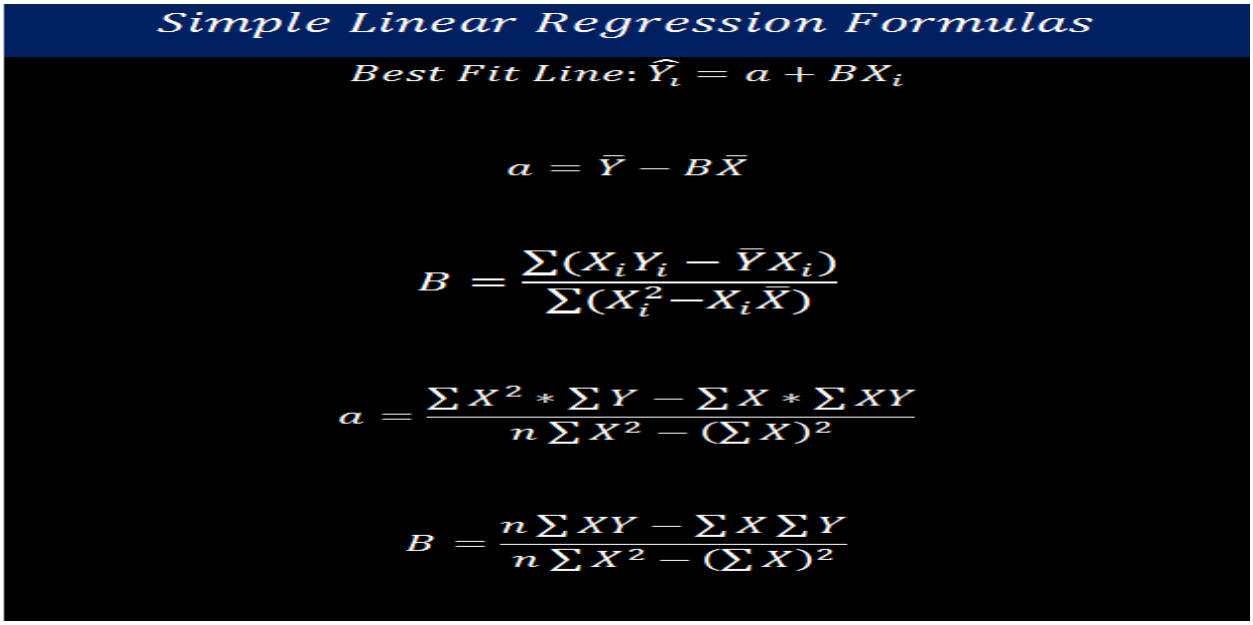
```
[5] # Scatter plot and show relationship between expereience and salary

    plt.scatter(df["YearsExperience"], df["Salary"])
    plt.title("Relationship between YearsExperience and Salary")
    plt.xlabel("YearsExperience")
    plt.ylabel("Salary")
    plt.show()
```



⇨ **Scatter plot and indicates relationship between years experience and salary.**

Find the best fit line in the data



Main function to calculate the coefficients of the linear best fit

```
# based on formula to get best fit line in linear regression

def simple_linear_regression(X, Y):
  # number of data
  n = len(X)

  # x bar
  X_mean = X.mean()

  # y bar
  Y_mean = Y.mean()

  # sum of x and y
  SUM_X = X.sum()
  SUM_Y = Y.sum()

  # sum of (X*Y) multiplied by n:
  SUM_XY= ((X*Y).sum())*n

  # sum of x * sum of y
  SUM_X_SUM_Y = SUM_X*SUM_Y

  # sum of (x*x) multiplied by n
  SUM_XX = ((X*X).sum())*n

  # square of sum of x
  SUM_X_Square = SUM_X*SUM_X

  # slope
  slope = (SUM_XY- SUM_X_SUM_Y)/(SUM_XX- SUM_X_Square)

  intercept = Y_mean - slope *X_mean

  return slope,intercept
```

⇨ **The picture shows a formula to find best fit line in linear regression model. So I implement it by defining a new function simple_linear_regression(X,Y). In this fuction, I first define x bar and y bar and sum of x and y. Then calculate Sum_xy, SUM_X_SUM_Y, SUM__XX, SUM_X_Square which is included in formula to get the slope. And after get the slope value, I input it to get y_intercept value.**

Find the slope value and y_intercept value for best fit line in the data

Calculate Slope and Y_intercept

```
# calcualte bestfit slope and y intercept
# training the model with train data
actual_slope, actual_intercept = simple_linear_regression(train["YearsExperience"], train["Salary"])
```

```
print("Slope: ", actual_slope)
print("Y_intercept: ", actual_intercept)
```

```
Slope:  10052.786167455975
Y_intercept:  23659.848673040913
```

Visualize the result

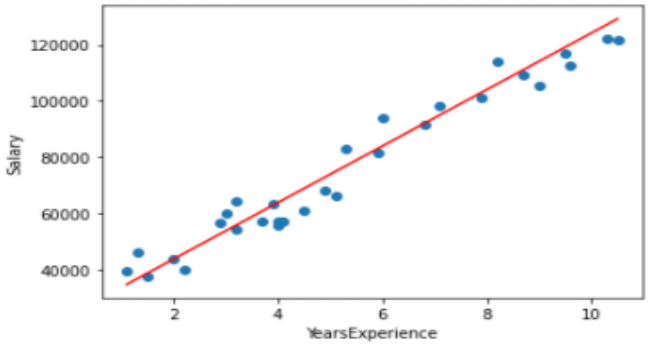
```
[10] # defining prediction function
     def get_regression_prediction(input, slope,intercept):
       pred_value = slope*input + intercept
       return pred_value
```

```
[11] # predicting values for whole dataset
     y_pred = get_regression_prediction(df["YearsExperience"], actual_slope, actual_intercept)
```

```
y_pred
```

```
[13] # plot the regression line with whole data

     plt.scatter(df["YearsExperience"], df["Salary"])
     plt.plot(df["YearsExperience"], y_pred, color="red")
     plt.xlabel("YearsExperience")
     plt.ylabel("Salary")
     plt.show()
```



⇨ **By using slope value and intercept value of best fit line, get the prediction value for yearsExperience. So we can draw linear regression line based on this. Below is the result of plotting regression line with whole data. And one can find out this line represents quite well for the data.**

▾ Calculating SSE and MSE

```
# error calculating using sum of squares error

def sum_of_squares_error(input, output, slope, intercept):
  predict = slope*input + intercept
  residual = (output - predict)

  SSE = (residual*residual).sum()

  return SSE
```

```
[15] SSE = sum_of_squares_error(df["YearsExperience"], df["Salary"],actual_slope, actual_intercept)
     print("Sum of squre error: ", SSE)

     Sum of squre error:  1057390424.8320996
```

```
[16] # calculating mean square error
     mse = mean_squared_error(df["Salary"], y_pred)

     print("Mean squre error: ", mse)

     Mean squre error:  35246347.49440332
```

⇨ **I calculated SSE by defining a function. Using slope and intercept of the best fit line for the data, which I calculated before, one can get predict value and deduct it from actual output. By doing this We can get residual value, then square residual and sum it. This way we can get the SSE value. For the mean squared error, I just used sklearn.metrics method. So I can easily get the value.**

2. Implement with scikit-learn

Using sckit learn to implement linear regression

▾ Simple Linear Regression Using Scikit-learn

```
[17] df = pd.read_csv("/content/Salary_Data.csv")
```

```
[18] # to using linear regression model
     X= df["YearsExperience"].values.reshape(-1, 1)
     Y= df["Salary"].values.reshape(-1, 1)
```

```
[19] # train test data split
     X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.3, random_state=30)
```

```
[20] regr = LinearRegression()
```

```
[21] model = regr.fit(X_train, y_train)
```

▾ Calcuate Slope and Intercept

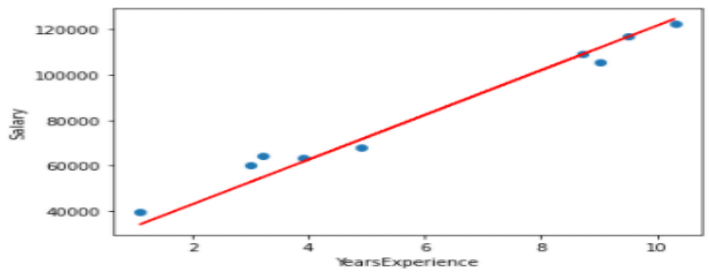
```
[22] # Slope
     print(regr.coef_[0])

     [9846.5759533]
```

```
[23] # Intercept
     print(regr.intercept_)

     [23278.82210193]
```

```
#Plot the regression line for testing data
plt.scatter(X_test, y_test)
plt.plot(X_test, predicted_data, color="red")
plt.xlabel("YearsExperience")
plt.ylabel("Salary")
plt.show()
```



▾ Calculating SSE and MSE

```
[27]
     print("Mean squared error: {}".format(mean_squared_error(y_test, predicted_data)))

     Mean squared error: 26123894.60572844
```

```
[28] res = predicted_data- y_test
     RSS =(res*res).sum()
```

```
[29] print("Residual sum of squares: ",  RSS)

     Residual sum of squares:  235115051.45155597
```

⇨ **Process is same but instead I used a scikit learn to make our lives easier.**

Conclusion:

There are two ways to implement ICP. First, implement with python scratch not using scikit-learn. So we can find best fit line by using formula defined for it. From this formula we can get the best fit line slope and y_intercept easily. And then we can build a linear line from these slope and intercept then could get the predict value of YearsExperince(X) by just input this x value to the linear line. Also we can get the SSE by defining function which get the residual value and then square it and sum it.

Secondly, We can get slope, y_intercept, SSE simply using scikit learn. By defining linear regression model predefined in scikit then fit it and we can get the slope and y_intercept value.

Challenges that I faced:

The most difficult challenge that I faced was to implement function to get the best fit line. I get the formula of the best fit line but it was hard to implement with python.

Video link

https://www.youtube.com/watch?v=dGEZTfI4lwQ