ML 19 - Simple Linear Regression By Virat Tiwari

December 5, 2023

1 ML 19 - Simple Linear Regression By Virat Tiwari

IN THIS ALGORITHM WE HAVE TO USE 9 STEPS FOR PERFORMING PREDICTION , THE SIMPLE LINER REGRESSION IN WHICH WE PREDICT THE OUTPUT FROM THE INPUT FEATURES AND IN THIS ALGORITHM WE USE ONLY ONE INDEPENDENT AND DEPENDENT FEATURE -

STEP 1 - READ THE DATASET

STEP 2 - EDA (EXPLORATORY DATA ANALYSIS)

STEP 3 - DIVIDE THE DATASET INTO THE DEPENDENT AND INDEPENDENT FEATURES

STEP 4 - DIVIDE DATASET INTO TRAIN AND TEST

STEP 5 - STANDARD SCALING OF TRAIN AND TEST - WE USE FOR NORMALISE THE FEATURES OF THE DATSET

STEP 6 - MODEL TRAINING

STEP 7 - FOR CHECKING THE ERROR WE HAVE TO USE PERFORMANCE METRICS (PERFORMANCE METRICS - MAE , MSE , RMSE , IT GIVES THE ERROR)

STEP 8 - ACCURACY OF THE MODEL -

STEP 9 - WE PICKLING OR SAVE THE MODELS INTO THE FORMAT OF FILE IN ANY SPECIF LOCATION IN HARD DRIVE SO THAT WE HAVE USED THESE FILES DURING THE DEPLOYMENT OF PRIOJECT

[1]:
```
# Thse are mandotory libraries that we have to import before performing any␣
↪alogorithm of ml or for the mathematical operations as well

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

STEP 1 - READ THE DATASET

[2]:
```
# pd.read_csv ( ) function is used for reading the datastet
```

```
df=pd.read_csv("height-weight.csv")
df.head()
```

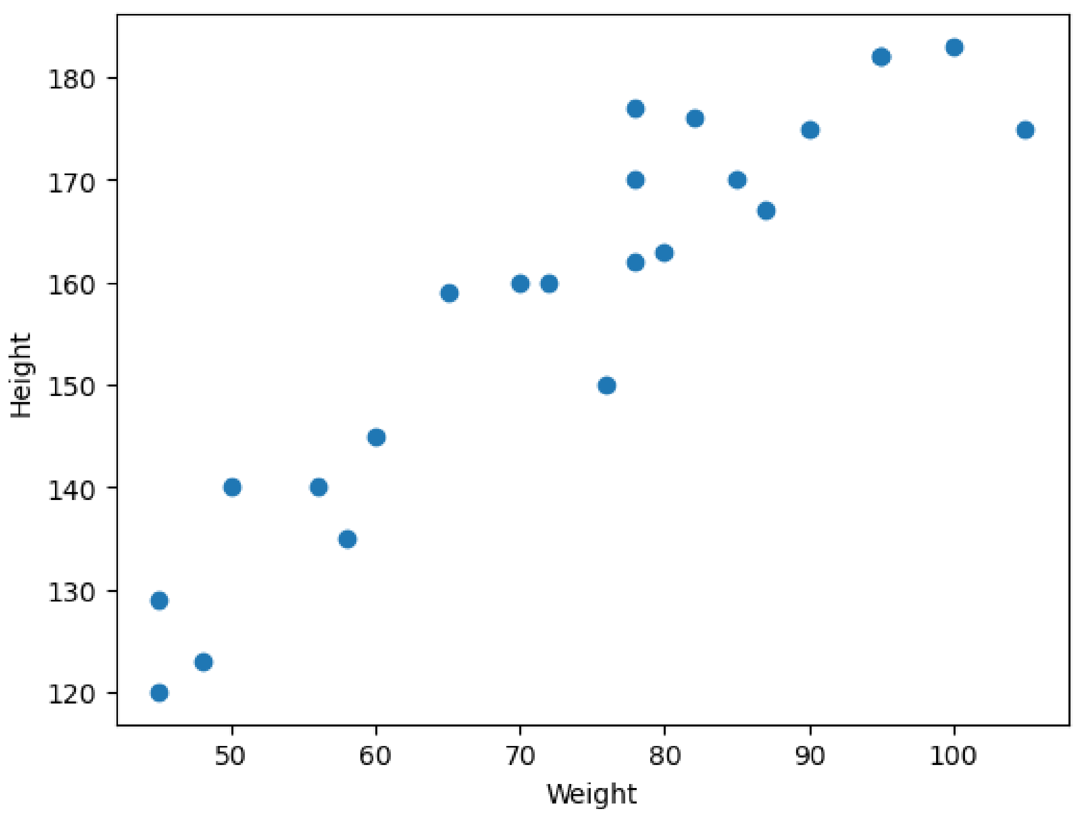
[2]:

	Weight	Height
0	45	120
1	58	135
2	48	123
3	60	145
4	70	160

[3]:

```
# .scatter( x, y ) function is used for intialising the datapoint accorinf=g to
↪the x and y axis of graogh in scatter plot
# Here we perform the scatter plot of "Weight" and "Height"

plt.scatter(df["Weight"],df["Height"])
plt.xlabel("Weight")
plt.ylabel("Height")
```

[3]: Text(0, 0.5, 'Height')



STEP 2 - EDA (EXPLORATORY DATA ANALYSIS)

[4]:
```
# . info (  ) function is used for getting the entire iformation of dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23 entries, 0 to 22
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Weight  23 non-null     int64
 1   Height  23 non-null     int64
dtypes: int64(2)
memory usage: 496.0 bytes
```

[5]:
```
# . describe ( )  function is used for getting the entire description like mean␣
 ↪median etc

df.describe()
```

[5]:

	Weight	Height
count	23.000000	23.000000
mean	73.826087	158.391304
std	17.872407	19.511626
min	45.000000	120.000000
25%	59.000000	142.500000
50%	78.000000	162.000000
75%	86.000000	175.000000
max	105.000000	183.000000

[6]:
```
# . isnull ( ) .sum ( )function is used for checking the null values in dataset

df.isnull().sum()
```

[6]:
```
Weight    0
Height    0
dtype: int64
```

STEP 3 - DIVIDE THE DATASET INTO THE DEPENDENT AND INDEPENDENT FEATURES

[7]:
```
# x=df [ [ ] ] function is used for getting independent features and we use two␣
 ↪times square brackets becouse we want it in array or input will be changes␣
 ↪it as we predict the new datapoint
# =df [ ]  function is used for  seperating the dependent feature w, in this we␣
 ↪just used one square due to single output column

x=df[["Weight"]] # Independent feature
```

```
y=df["Height"] # Depnedent feature
```

[8]:
```
# . shape function is used for getting ( datapoints and columns )

x.shape
```

[8]: (23, 1)

[9]:
```
# . shape function is used for getting ( datapoints and columns )

y.shape
```

[9]: (23,)

STEP 4 - DIVIDE DATASET INTO TRAIN AND TEST

[10]:
```
# we import train_test_split library for divide the dataset into train and test

from sklearn.model_selection import train_test_split
```

[11]:
```
# PROCESS OF SPLITING DATASET INTO THE FOR TRAIN AND TEST
# train_test_split(x,y,test_size=0.20 ("this is percentage of data that we have␣
↪used for test"),random_state=42) - We intialise the perimetre inside␣
↪train_test_split library
# We get x_train,x_test,y_train,y_test through this library

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
↪20,random_state=42)
```

[12]:
```
# Finally we split the dataset into train and test
# x_train = ((18, 1)
# x_test = (5, 1))

x_train.shape,x_test.shape
```

[12]: ((18, 1), (5, 1))

[13]:
```
# Finally we split the dataset into train and test
# y_train = ((18, 1)
# y_test = (5, 1))

y_train.shape,y_test.shape
```

[13]: ((18,), (5,))

STEP 5 - STANDARD SCALING OF TRAIN AND TEST - WE USE FOR NORMALISE THE FEATURES OF THE DATSET

[14]:
```python
# import StandardScaler library for normalise the features of the dataset

from sklearn.preprocessing import StandardScaler
```

[15]:
```python
# scaler variable store the values of StandardScaler

scaler=StandardScaler()
```

[16]:
```python
# scaler.fit_transform( ) function is used for foir the train dataset into the
 ↪model and transform it into the most suitable for the model in a single step

x_train=scaler.fit_transform(x_train)
```

Note - We always Tranform the test dataset not fit_tranform

[17]:
```python
# scaler.transform( ) is used for transform the test data

x_test=scaler.transform(x_test)
```

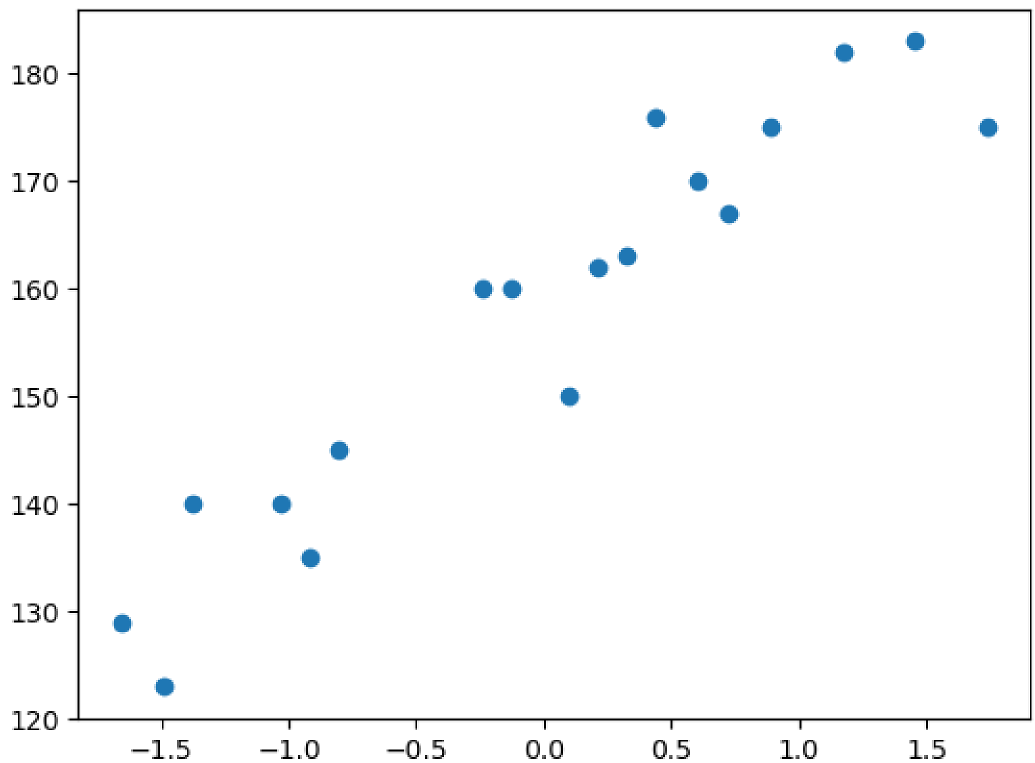
[18]:
```python
x_test
```

[18]:
```
array([[ 0.21043706],
       [ 0.21043706],
       [-1.6552288 ],
       [ 1.17153765],
       [-0.52452222]])
```

Note - We never perform StandardScaler on output feature

[19]:
```python
# In this plot we normalise the datapoints as we compare from the first plot

plt.scatter(x_train,y_train)
```

[19]:
```
<matplotlib.collections.PathCollection at 0x7fc25bb45630>
```



Note - Whenever we get new datapoint so we can convert that datapoint into standard scaler by using tranform ([[ ])]

[20]:
```
scaler.transform([[80]])
```

[20]:
```
array([[0.32350772]])
```

STEP 6 - MODEL TRAINING

[21]:
```
# import LinearRegression library is used for model training

from sklearn.linear_model import LinearRegression
```

[22]:
```
regressor=LinearRegression()
```

[23]:
```
# Training The Train Data
# Inside the fit ( ) function we have to give training dataset variables
# This step create the Best Fit Line

regressor.fit(x_train,y_train)
```

[23]:
```
LinearRegression()
```

[24]:
```python
# This is our intercept
# We get only one interv=cept due to one dependent or outputt feature

regressor.intercept_
```

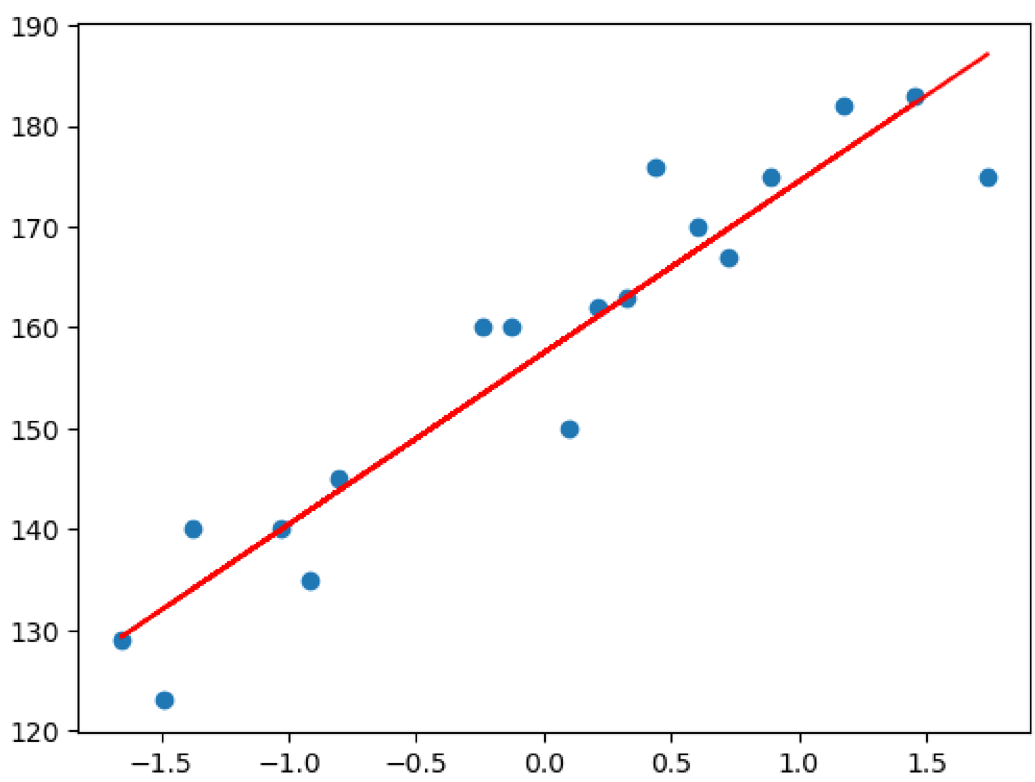
[24]: 157.5

[25]:
```python
# This is our coefficient
# We get only one coefficient due to one input or independent feature

regressor.coef_
```

[25]: array([17.03440872])

[26]:
```python
# In this plot we have seen red line that is "BEST FIT LINE"
# We predict that line from the train dataset

plt.scatter(x_train,y_train)
plt.plot(x_train,regressor.predict(x_train),"r")
```

[26]: [<matplotlib.lines.Line2D at 0x7fc25b82f1c0>]



PREDICTION OF TRAIN DATA -

1) Predicted Height output = intercept + coef_(Weight) 2) y_pred_train=157.5 + 17.03 (x_train)

PREDICTION OF TEST DATA -

1) Predicted Height output = intercept + coef_(Weight) 2) y_pred_test =157.5 + 17.03 (x_test)

[27]:
```
# Prediction for test data
#This is thenprediction of test data

y_pred_test=regressor.predict(x_test)
```

[28]:
```
# Predicted value

y_pred_test
```

[28]:
```
array([161.08467086, 161.08467086, 129.3041561 , 177.45645118,
       148.56507414])
```

[29]:
```
# Real value

y_test
```

[29]:
```
15    177
9     170
0     120
8     182
17    159
Name: Height, dtype: int64
```

STEP 7 - FOR CHECKING THE ERROR WE HAVE TO USE PERFORMANCE METRICS (PERFORMANCE METRICS - MAE , MSE , RMSE , IT GIVES THE ERROR)

[30]:
```
# Performance Matrics - MAE , MSE , RMSE

from sklearn.metrics import mean_squared_error,mean_absolute_error
```

[31]:
```
# Here we find differnt types of ERROR RATE

mse=mean_squared_error(y_test,y_pred_test)
mae=mean_absolute_error(y_test,y_pred_test)
rmse=np.sqrt(mse)
print(mse)
print(mae)
print(rmse)
```

```
109.77592599051664
9.822657814519232
10.477400726827081
```

Note - For checking the ACCURACY of the Model we have to use R Squared and Adjusted R squared Concept

STEP 8 - ACCURACY OF THE MODEL -

R SQUARE -

Formula - : R^2=1-SSR/SST

1) R^2 - Accuracy of the model

2) SSR - sum of square of residuals

3) SST - total sum of squares

[32]:
```
# import r2_score - This library is used for checking the accuracy of model R␣
↪squre concept

from sklearn.metrics import r2_score
```

[33]:
```
# Here we check the accuracy

score=r2_score(y_test,y_pred_test)
```

[34]:
```
# 0.7769... or 70 % is the Accuracy of the Model

score
```

[34]:
```
0.776986986042344
```

ADJUSTED R SQUARE -

Formula - : 1-[(1-R2)*(n-1)/(n-k-1)]

1) R2 - R2 or r Square of the model

2) n - No of observations

3) k - No of predictor variables

[35]:
```
# ADJUSTED R SQUARE - USED FOR CHECKING ACCURACY OF THE MODEL

1-(1-score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

[35]:
```
0.7026493147231252
```

STEP 9 - WE PICKLING OR SAVE THE MODELS INTO THE FORMAT OF FILE IN ANY SPECIF LOCATION IN HARD DRIVE SO THAT WE HAVE USED THESE FILES DURING THE DEPLOYMENT OF PRIOJECT

[36]:
```
# THIS IS FOR STANDARD SCALING

scaler
```

[36]: StandardScaler()

[37]:
```
# THIS IS FOR PREDICTION OF MODEL

regressor
```

[37]: LinearRegression()

THANK YOU SO MUCH !!

YOURS VIRAT TIWARI :)