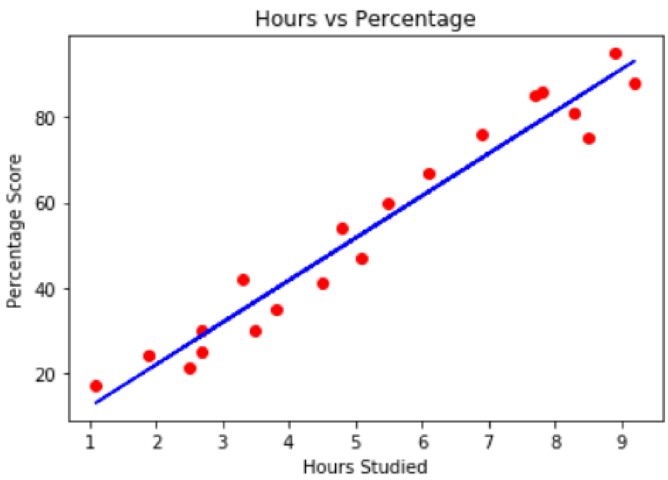
9.1. Data Science – Machine Learning – Linear Regression Example

Contents

9.1. Data Science – Machine Learning – Linear Regression Example

1. Scenario

- ✓ Let's find the relationship in between marks and number of study hours
- ✓ We want to find out the marks for given number of study hours to a student
- ✓ If we plot the independent variable (hours) on the x-axis and dependent variable (percentage) on the y-axis then linear regression gives us a straight line that best fits the data points.



2. Maths behind

- ✓ We know that the equation of a straight line is basically
  - **y = mx + b**

- ✓ Where **b** is the **intercept** and **m** is the **slope** of the line.
- ✓ So basically, the linear regression algorithm gives us the most optimal value for the **intercept** and the **slope**.
- ✓ The y and x variables remain the same
- ✓ There can be multiple straight lines depending upon the values of intercept and slope.
- ✓ Basically what the linear regression algorithm does is it fits multiple lines on the data points and returns the line that results in the least error.

3. Loading dataset

- ✓ Once we have dataset then we need to load by using pandas
- ✓ If we load dataset then it returns the DataFrame

Program Name: Loading student_scores dataset
demo1.py

```python
import pandas as pd

df = pd.read_csv('student_scores.csv')

print(df.head())
```

Output

```
   Hours  Scores
0    2.5      21
1    5.1      47
2    3.2      27
3    8.5      75
4    3.5      30
```

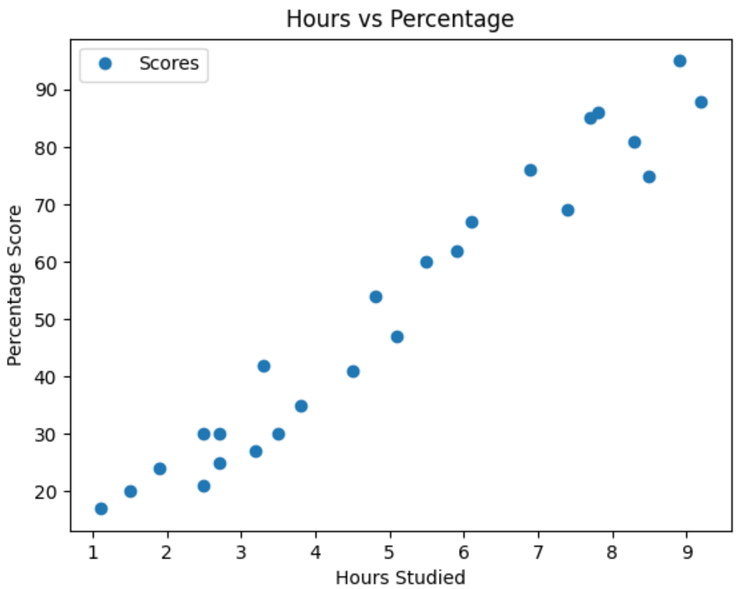
4. Plotting the data

✓ Let's plot our data points to see the relationship between the data.

Program Name: Plotting the dataset
demo2.py

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('student_scores.csv')

df.plot(x='Hours', y='Scores', style='o')

plt.title('Hours vs Percentage')

plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')

plt.show()
```

Output



Program Name

Preparing the data
demo3.py

```
import pandas as pd

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

print(X)
print(y)
```

Output

```
[[2.5]
 [5.1]
 [3.2]
 [8.5]
 [3.5]
 [1.5]
 [9.2]
 [5.5]
 [8.3]
 [2.7]
 [7.7]
 [5.9]
 [4.5]
 [3.3]
 [1.1]
 [8.9]
 [2.5]
 [1.9]
 [6.1]
 [7.4]
 [2.7]
 [4.8]
 [3.8]
 [6.9]
 [7.8]]
[21 47 27 75 30 20 88 60 81 25 85 62 41 42 17 95 30 24 67 69 30 54 35 76
 86]
```

Program Name	Splitting the data demo4.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 0)

print("X_train")
print(X_train)

print()
print("X_test")
print(X_test)

print()
print("Y_train")
print(y_train)

print()
print("y_test")
print(y_test)
```

Output

```
X_train
[[3.8]
 [1.9]
 [7.8]
 [6.9]
 [1.1]
 [5.1]
 [7.7]
 [3.3]
 [8.3]
 [9.2]
 [6.1]
 [3.5]
 [2.7]
 [5.5]
 [2.7]
 [8.5]
 [2.5]
 [4.8]
 [8.9]
 [4.5]]

X_test
[[1.5]
 [3.2]
 [7.4]
 [2.5]
 [5.9]]

Y_train
[35 24 86 76 17 47 85 42 81 88 67 30 25 60 30 75 21 54 95 41]

y_test
[20 27 69 30 62]
```

Program Name	Training the model demo5.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print("Model got trained with data")
```

Output

```
Model got trained with data
```

5. Intercept & coefficient

- ✓ In the theory section we said that linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data.
- ✓ We can get the values of the intercept and slop from linear regression

Program Name: Getting intercept from created model
demo6.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print(regressor.intercept_)
```

Output

```
2.018160041434669
```

Program Name	Getting coefficient from created model demo7.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print(regressor.coef_)
```

Output

```
[9.91065648]
```

6. Important info

- ✓ This means that for every one unit of change in hours studied, the change in the score is about 9.91%.
- ✓ In simpler words, if a student studies one hour more than they previously studied for an exam, they can expect to achieve an increase of 9.91% in the score achieved by the student previously.

7. Making Predictions

- ✓ Now successfully we have trained our algorithm.
- ✓ So, it's time to make some predictions.
- ✓ We need to use our test data and see how accurately our algorithm predicts the percentage score.

Program Name: Making predictions
demo8.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)


y_pred = regressor.predict(X_test)
print(y_pred)
```

Output

```
[16.88414476 33.73226078 75.357018   26.79480124
60.49103328]
```

Great

- ✓ The y_pred is a numpy array that contains all the predicted values for the input values in the X_test series.

Comparison

- ✓ To compare the actual output values for X_test with the predicted values.

Program Name	Comparing the predicted result demo9.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

d = {'Actual': y_test, 'Predicted': y_pred}

compare_df = pd.DataFrame(d)
print(compare_df)
```

Output

```
   Actual  Predicted
0      20  16.884145
1      27  33.732261
2      69  75.357018
3      30  26.794801
4      62  60.491033
```

- ✓ Though our model is not very precise, the predicted percentages are close to the actual ones.

8. Evaluating the Algorithm

- ✓ We need to evaluate the performance of algorithm.
- ✓ This step is particularly important to compare how well different algorithms perform on a particular dataset.
- ✓ For regression algorithms there are three evaluation metrics are commonly used

9. Evaluation metrics

- ✓ Mean Absolute Error (MAE)
- ✓ Mean Squared Error (MSE)
- ✓ Root Mean Squared Error (RMSE)

Mean Absolute Error (MAE)

- ✓ Mean Absolute Error (MAE) is the mean of the absolute value of the errors.

Mean Squared Error (MSE)

- ✓ Mean Squared Error (MSE) is the mean of the squared errors.

Root Mean Squared Error (RMSE)

- ✓ Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors

$$MAE = \overbrace{\boxed{\frac{1}{n}}}^{\text{Divide by the total number of data points}} \underbrace{\Sigma}_{\text{Sum of}} \underbrace{\left| \overbrace{\boxed{y}}^{\text{Actual output value}} - \overbrace{\boxed{\widehat{y}}}^{\text{Predicted output value}} \right|}_{\text{The absolute value of the residual}}$$

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)}_{\text{The square of the difference between actual and predicted}}^{2}$$

We no need to calculate manually

- ✓ We don't have to perform these calculations manually.
- ✓ The scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

Program Name: Loading student_scores dataset
demo10.py

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
```

Output

```
Mean Absolute Error: 4.183859899002982
Mean Squared Error: 21.598769307217413
Root Mean Squared Error: 4.647447612100368
```