Machine Learning Techniques

Regression

Types of Regression

Univariate/Simple Linear Regression

variables: independent =1 and dependent =1
Relationship Nature: Linear

Multiple Regression

variables: independent >1 and dependent =1
Relationship Nature: Linear/Non Linear

Multivariate Regression

variables: independent >1 and dependent >1
Relationship Nature: Linear/Non-Linear

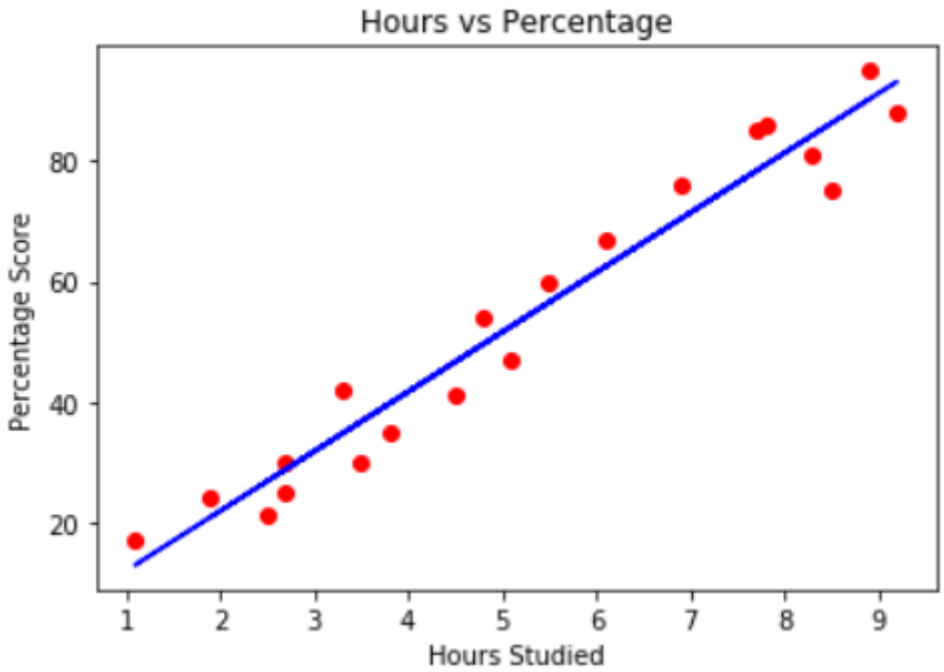
Polynomial Regression

the power of the independent variable is more than 1.
Relationship Nature: Non-Linear
Y = a + bX^2
The best fit line is not a straight line, rather a curve that fits into the data points.

Linear Regression

Example Problem: Determine the linear relationship between the numbers of hours a student studies and the percentage of marks that student scores in an exam?

We want to find out that given the number of hours a student prepares for a test, about how high of a score can the student achieve? If we plot the independent variable (hours) on the x-axis and dependent variable (percentage) on the y-axis, linear regression gives us a straight line that best fits the data points, as shown in the figure below.



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 (in two dimensions). The y and x variables remain the same, since they are the data features and cannot be changed. The values that we can control are the intercept and slope. 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.

This same concept can be extended to the cases where there are more than two variables. This is called multiple linear regression. For instance, consider a scenario where you have to predict the price of house based upon its area, number of bedrooms, average income of the people in the area, the age of the house, and so on. In this case the dependent variable is dependent upon several independent variables. A regression model involving multiple variables can be represented as:

y = b0 + m1b1 + m2b2 + m3b3 + ... mnbn

This is the equation of a hyper plane. Remember, a linear regression model in two dimensions is a straight line; in three dimensions it is a plane, and in more than three dimensions, a hyper plane.

Simple Linear Regression

In this regression task we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied. This is a simple linear regression task as it involves just two variables.

Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import sklearn
```

Reading Dataset

In [2]:

```
dataset = pd.read_csv('student_scores.csv')
dataset
```

Out[2]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

In [3]:

```
dataset.shape
```

Out[3]:

```
(25, 2)
```

In [4]:

```
dataset.head()
```

Out[4]:

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

In [5]:

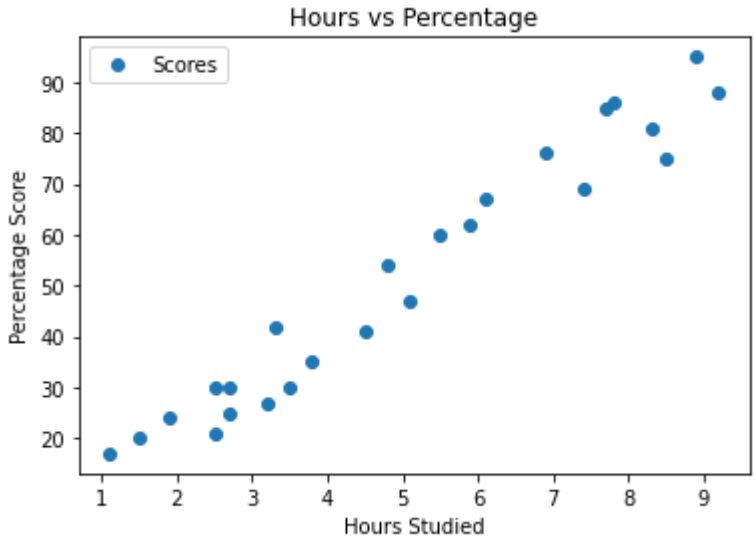
```
dataset.describe()
```

Out[5]:

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

In [6]:

```
dataset.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



Preparing the Data

In [7]:

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

In [8]:

```
!pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\users\khanw\.conda\envs\tf-gpu
\lib\site-packages (0.0)
Requirement already satisfied: scikit-learn in c:\users\khanw\.conda\envs\tf
-gpu\lib\site-packages (from sklearn) (1.0.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\khanw\.conda
\envs\tf-gpu\lib\site-packages (from scikit-learn->sklearn) (3.1.0)
Requirement already satisfied: scipy>=1.1.0 in c:\users\khanw\.conda\envs\tf
-gpu\lib\site-packages (from scikit-learn->sklearn) (1.7.1)
Requirement already satisfied: numpy>=1.14.6 in c:\users\khanw\.conda\envs\t
f-gpu\lib\site-packages (from scikit-learn->sklearn) (1.21.2)
Requirement already satisfied: joblib>=0.11 in c:\users\khanw\.conda\envs\tf
-gpu\lib\site-packages (from scikit-learn->sklearn) (1.1.0)
```

Splitting the Dataset

In [9]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Training the Algorithm

With Scikit-Learn it is extremely straight forward to implement linear regression models, as all you really need to do is import the LinearRegression class, instantiate it, and call the fit() method along with our training data.

In [10]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[10]:

```
LinearRegression()
```

Linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data.

In [11]:

```
print(regressor.intercept_)
print(regressor.coef_)
```

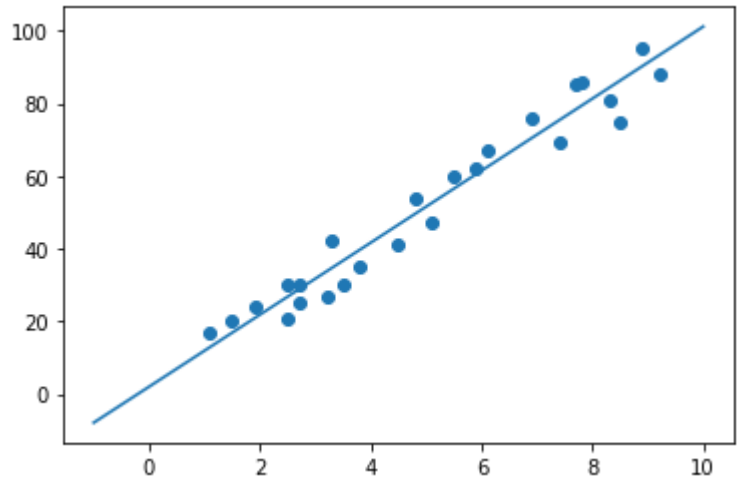
```
2.018160041434683
[9.91065648]
```

In [12]:

```
a = regressor.coef_
b = regressor.intercept_
```

In [13]:

```
p = np.linspace(-1,10,100)
q=a*p+b
plt.plot(p,q)
plt.scatter(X,y)
plt.show()
```



Making Predictions

Now that we have trained our algorithm, it's time to make some predictions. To do so, we will use our test data and see how accurately our algorithm predicts the percentage score.

In [14]:

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

Compare the actual output values for X_test with the predicted values as below,

In [15]:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[15]:

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

Evaluating the Algorithm

The final step is 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, three evaluation metrics are commonly used:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

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

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

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.

In [16]:

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 4.183859899002975
Mean Squared Error: 21.5987693072174
Root Mean Squared Error: 4.6474476121003665
```

References

1. Linear Regression in Python with Scikit-Learn https://stackabuse.com/linear-regression-in-python-with-scikit-learn/ (https://stackabuse.com/linear-regression-in-python-with-scikit-learn/)
2. A Beginner's Guide to Linear Regression in Python with Scikit-Learn https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html (https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html)
3. Linear Regression (Python Implementation) https://www.geeksforgeeks.org/linear-regression-python-implementation/ (https://www.geeksforgeeks.org/linear-regression-python-implementation/)
4. Machine Learning Repository - Datasets https://archive.ics.uci.edu/ml/datasets.php?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table (https://archive.ics.uci.edu/ml/datasets.php?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table)

Multiple Regression

We performed linear regression above involving two variables. Almost all real world problems that you are going to encounter will have more than two variables. Linear regression involving multiple variables is called "multiple linear regression". The steps to perform multiple linear regression are almost similar to that of simple linear regression. The difference lies in the evaluation. You can use it to find out which factor has the highest impact on the predicted output and how different variables relate to each other.

Here, we will use multiple linear regression to predict the gas consumptions (in millions of gallons) in 48 US states based upon gas taxes (in cents), per capita income (dollars), paved highways (in miles) and the proportion of population that has a drivers license.

Dataset: https://drive.google.com/open?id=1mVmGNx6cbfvRHC_DvF12ZL3wGLSHD9f (https://drive.google.com/open?id=1mVmGNx6cbfvRHC_DvF12ZL3wGLSHD9f)_ Details of Dataset: http://people.sc.fsu.edu/~jburkardt/datasets/regression/x16.txt (http://people.sc.fsu.edu/~jburkardt/datasets/regression/x16.txt)

Importing the Libraries

In [17]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [18]:

```
dataset = pd.read_csv('C:\\Users\\khanw\\Documents\\GitHub\\AI_Tutorials\\Datasets\\petrol_
```

In [19]:

```
dataset.head()
```

Out[19]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumpt
0	9.0	3571	1976	0.525	
1	9.0	4092	1250	0.572	
2	9.0	3865	1586	0.580	
3	7.5	4870	2351	0.529	
4	8.0	4399	431	0.544	

In [20]:

```
dataset.describe()
```

Out[20]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consu
count	48.000000	48.000000	48.000000	48.000000	48
mean	7.668333	4241.833333	5565.416667	0.570333	576
std	0.950770	573.623768	3491.507166	0.055470	111
min	5.000000	3063.000000	431.000000	0.451000	344
25%	7.000000	3739.000000	3110.250000	0.529750	509
50%	7.500000	4298.000000	4735.500000	0.564500	568
75%	8.125000	4578.750000	7156.000000	0.595250	632
max	10.000000	5342.000000	17782.000000	0.724000	968

Preparing the Data

The next step is to divide the data into attributes and labels as we did previously. However, unlike last time, this time around we are going to use column names for creating an attribute set and label.

In [21]:

```
X = dataset[['Petrol_tax', 'Average_income', 'Paved_Highways',
       'Population_Driver_licence(%)']]
y = dataset['Petrol_Consumption']
```

Dataset Splitting

Divide data into training and test sets:

In [22]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Training the Algorithm

And finally, to train the algorithm use the fit() method of the LinearRegression class:

In [23]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[23]:

```
LinearRegression()
```

As, in case of multivariable linear regression, the regression model has to find the most optimal coefficients for all the attributes. To see what coefficients our regression model has chosen, execute the following script:

In [24]:

```
coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

Out[24]:

	Coefficient
Petrol_tax	-40.016660
Average_income	-0.065413
Paved_Highways	-0.004741
Population_Driver_licence(%)	1341.862121

This means that for a unit increase in "petrol_tax", there is a decrease of 24.19 million gallons in gas consumption. Similarly, a unit increase in proportion of population with a drivers license results in an increase of 1.324 billion gallons of gas consumption. We can see that "Average_income" and "Paved_Highways" have a very little effect on the gas consumption.

Making Predictions

To make pre-dictions on the test data, use predict method.

In [25]:

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

To compare the actual output values for X_test with the predicted values, execute the following script:

In [26]:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[26]:

	Actual	Predicted
29	534	469.391989
4	410	545.645464
26	577	589.668394
30	571	569.730413
32	577	649.774809
37	704	646.631164
34	487	511.608148
40	587	672.475177
7	467	502.074782
10	580	501.270734

Evaluating the Algorithm

The final step is to evaluate the performance of algorithm. This is done by finding the values for MAE, MSE and RMSE.

In [27]:

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 56.822247478964684
Mean Squared Error: 4666.3447875883585
Root Mean Squared Error: 68.31064915215165
```

It can be seen that the value of root mean squared error is 60.07, which is slightly greater than 10% of the mean value of the gas consumption in all states. This means that our algorithm was not very accurate but can still make reasonably good predictions.

There are many factors that may have contributed to this inaccuracy, a few of which are listed here:

- **Need more data:** Only one year worth of data isn't that much, whereas having multiple years worth could have helped us improve the accuracy quite a bit.
- **Bad assumptions:** We made the assumption that this data has a linear relationship, but that might not be the case. Visualizing the data may help you determine that.
- **Poor features:** The features we used may not have had a high enough correlation to the values we were trying to predict.

References

1. Linear Regression in Python with Scikit-Learn https://stackabuse.com/linear-regression-in-python-with-scikit-learn/ (https://stackabuse.com/linear-regression-in-python-with-scikit-learn/)
2. A Beginner's Guide to Linear Regression in Python with Scikit-Learn https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html (https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html)
3. Linear Regression (Python Implementation) https://www.geeksforgeeks.org/linear-regression-python-implementation/ (https://www.geeksforgeeks.org/linear-regression-python-implementation/)
4. Machine Learning Repository - Datasets https://archive.ics.uci.edu/ml/datasets.php?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table (https://archive.ics.uci.edu/ml/datasets.php?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table)