Linear Regression for Machine Learning

Linear regression is used to model relationship between two variables by using a linear equation of the form:

$y = a + bx$ (1)

The variable y may be viewed as being related to variable x in some way.

The goal is to find that relationship from values of x, called the predictor variable, and the corresponding values of y, called the outcome variable, from a set of observed or measured data points for x and y. Once a satisfactory relationship has been found, we can use this relationship to predict values of y for a chosen value of x. Equation (1) models a straight-line relationship, where a is known as the intercept, the value of y when x is zero, and b represents the slope of the line. The line is called a regressions line because it is not likely to connect all data points but a closest approximation that can be found from the given data points. The process is called curve fitting because we try make the line pass through all data points evenly if not perfectly. In order to have confidence in the calculated relationship based on the observed sample data, the sample has to be representative of the population from which the sample is drawn.

Once a credible relationship has been found for the given data, it can then be used to make prediction for an outcome y based on values of predictor x.

Before carrying out regression, a scatter plot of observed or measured data should be made to assess whether a linear regression appears to be a reasonable choice.

Figure 1 shows a scatter diagram of automobile data consisting of 1267 data points for several automobile models and attributes. The scatter plot is for selected attributes of CO_2 emissions for engine sizes. The density of data in this plot has made the points appear as blobs as opposed to distinct points.

Here x is named as Engine Size and y is names as Co_2 emissions.

Figure 1: Sample Data

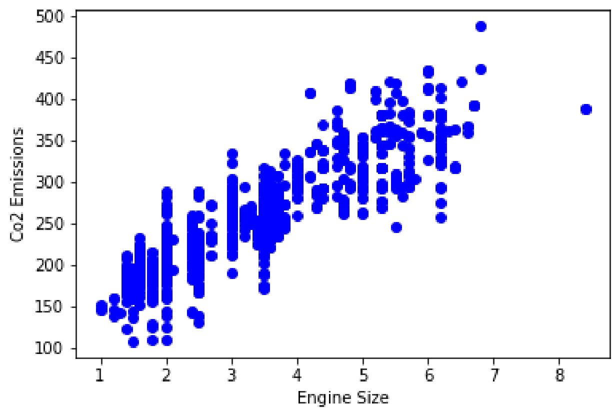
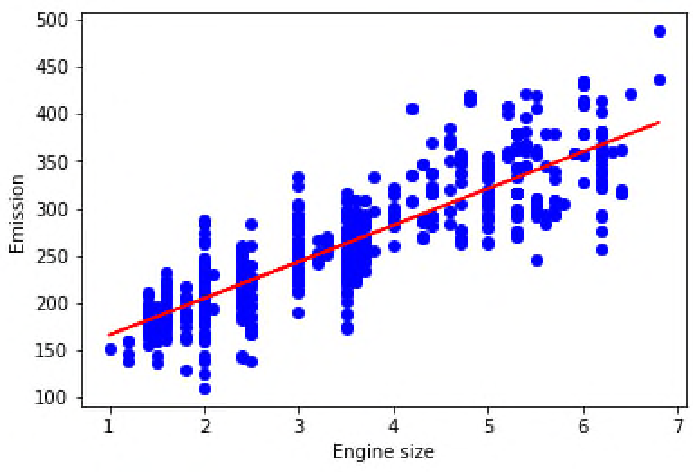


Figure 2: Graph Shows the Regression Line, in red, after Linear Regression



While the line passes through a few points, it can be viewed as a reasonable approximation of the situation. There can be multiple lines based on the intercept and slope. What the linear regression algorithm does is to try various lines through the data points and return the line that fits in terms of least deviation of points from the line. Statistical measures are used to determine how well a fit may serve in making predictions.

Some examples of business applications of linear regression are:

- Predict future prices or costs of items to be purchased by a business.
- Predict revenue from sales
- Compare performance of newly introduced products versus the existing ones.
- View the increase in emissions with the increase in power/size of automobile engines
- Predict rise and fall in disease based on collected on cases which have already occurred.

Regression methods falls in the category of supervised learning. Supervised learning involves inferring a function from labeled training data. It is called supervised because the modeler selects a model based on a view of data. Instead of a straight-line model, the modeler could have chosen a polynomial or an exponential curve. Once a model is chosen, it can go through a training using sample data. Training consists of trying out by making the line pass through the data in different positions and assessing whether a trial produces better or worse outcome. You may visualize it as sitting with a ruler and placing it in different positions through the data until you have found the best place for it. This is the learning process.

In performing regression using Python programming language, Scikit-Learn (sklearn) package is used to generate regression model and to train it with test data.

We will describe the various programming steps and show intermediate outputs until the process is completed. Details of mathematical calculations for regression analysis will not be shown. We will consider those details as hidden in Scikit-Learn package. The goal here is how to use Scikit-Learn in producing models for predictions without getting involved in mathematics of regression analysis. This is no different than using a square root function in a programming language to find square root of a number rather than writing code for calculating it from its mathematical formulation.

The progression through the solution will be demonstrated in terms of a number of small steps with corresponding outcomes and explanatory comments. The actual coding will appear in boxes to separate from explanations.

Step 1: Setting Up the Environment

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

Here pre-built code is brought in from various available packages called Python libraries. If you are using a Python distribution such as Anaconda, many such libraries are included already. You don’t have install them yourselves. You just have to tell what to import in your current environment. Here the four packages needed are pandas, numpy, matplotlib member pyplot, and sklearn mentioned earlier. Pd, np, plt, are nicknames for ease of use of these packages in writing program code.

Pandas is a high-level data manipulation tool set. Numpy is a math library, in particular for matrices. Matplotlib is a plotting package that provides 2D and 3D plotting. Sklearn is machine learning library with classification, regression, and clustering algorithms.

Step 2: Loading Data for Regression Analysis

The automobile data for this case study has been loaded from the following source and saved as an MS Excel file in your working directory as named below:

https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/FuelConsumptionCo2.csv

```
df=pd.read_excel(r'FuelConsumptionCo2.xlsx')
df.head()
```

Data into a data frame called df. Any name can be assigned for subsequent use in coding. The coding line df.head() provides a partial listing of data shown below. Of course you can print df itself.

```
   MODELYEAR   MAKE  ... FUELCONSUMPTION_COMB_MPG CO2EMISSIONS
0       2014  ACURA  ...                       33          196
1       2014  ACURA  ...                       29          221
2       2014  ACURA  ...                       48          136
3       2014  ACURA  ...                       25          255
4       2014  ACURA  ...                       27          244

[5 rows x 13 columns]
```

The data has 13 columns, only two of them are called Engine Size and CO_2 emissions are of interest to us in this case study. First 12 records belong to one automobile model are shown below:

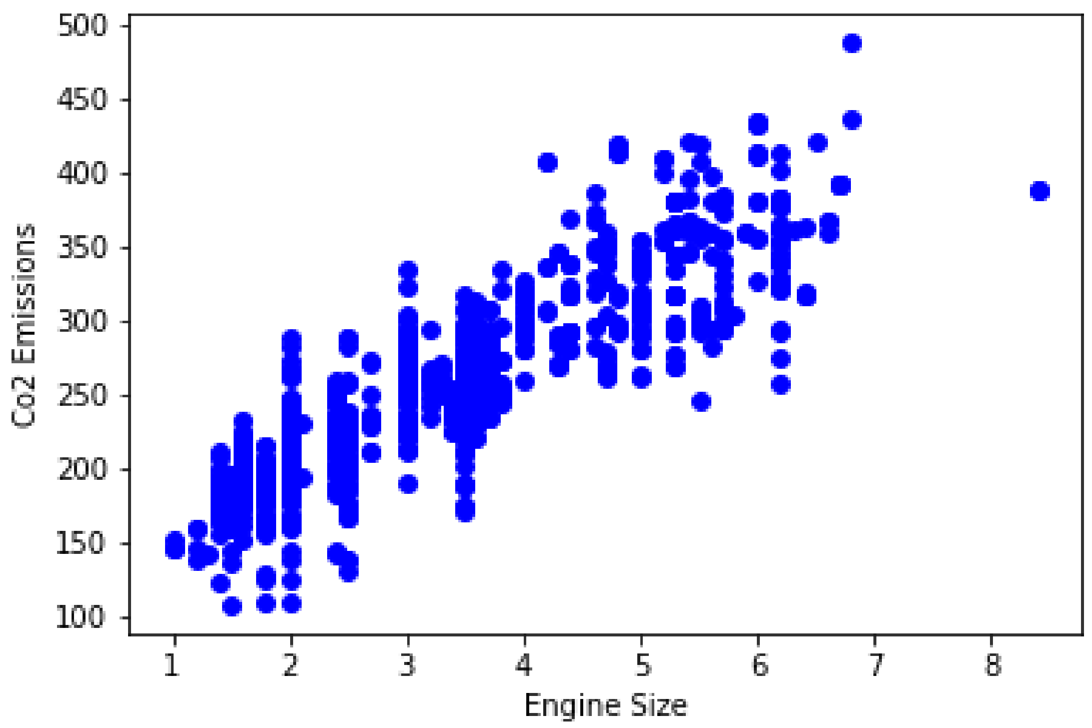
ENGINESIZE	CO2EMISSIONS
2	196
2.4	221
1.5	136
3.5	255
3.5	244
3.5	230
3.5	232
3.7	255
3.7	267
2.4	212
2.4	225
3.5	239

Step 3: Scatter Plot of Data and Select Features for Regression

```
plt.scatter(df["ENGINESIZE"], df["CO2EMISSIONS"], color="blue")
plt.xlabel("ENGINESIZE")
plt.ylabel("CO2EMISSIONS")
plt.show()
data = df[["ENGINESIZE","CO2EMISSIONS"]]    #selected features
```

The resulting plot of the entire data 1267 records for the Engine Size and Corresponding CO_2Emissions are shown in Figure 3.

Figure 3: Scatter Plot of Automobile Data



Although the data points are widely scattered, they do point out to the possibility of using a simple linear model.

Step 4: Selecting Training Data and Carrying Out Regression and Print Results

```
train= df[:(int((len(df)*.8)))]
test= df[(int((len(df)*.8))):]
regr= linear_model.LinearRegression()
train_x= np.array(train[["ENGINESIZE"]])
train_y= np.array(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)
print ("Coefficient : ",regr.coef_)
print ("Intercept : ",regr.intercept_)
```

Eighty percent has been selecting for training and test purposes. Training data to fit the model and testing data to test it. Sklearn will be used to make those choices.

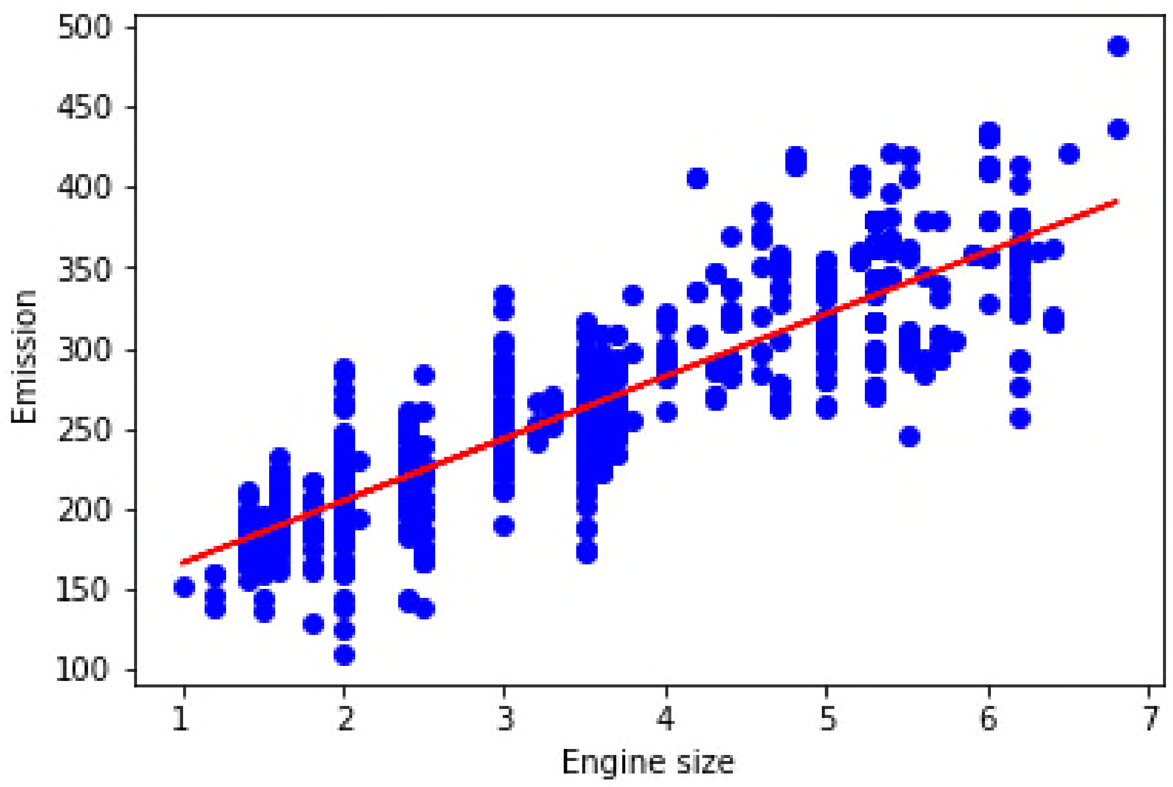
The printed results are:

```
Coefficient :  [[38.79512384]]
Intercept :  [127.16989951]
```

Given the general linear model equation (1), the corresponding regression line based on given data is:
$y = 127.17 + 38.8x$ (2)

Step 5: Plotting the Regression Line for Scatter Plot

```
plt.scatter(train["ENGINESIZE"], train["CO2EMISSIONS"], color='blue')
plt.plot(train_x, regr.coef_*train_x + regr.intercept_, '-r')
plt.xlabel("Engine size")
plt.ylabel("Emission")
```



Step 6: Making Predictions

```
def get_regression_predictions(input_features, intercept, slope):
  predicted_values= input_features*slope + intercept
  return predicted_values
```

The function get_regression_prediction simply returns the predicted value based on model shown in equation (2). Now this predictor can be called as often as needed by simply feeding a value of engine size.

One example of such a call is shown below.

```
def get_regression_predictions(input_features, intercept, slope):
  predicted_values= input_features*slope + intercept
  return predicted_values
```

The predicted value of emission, model based estimation, is:

```
Output: Estimated Emission:  262.9528329350173
```

Step 7: Checking the Goodness of Fit

```
from sklearn.metrics import r2_score
test_x = np.array(test[['ENGINESIZE']])
test_y = np.array(test[['CO2EMISSIONS']])
test_y_ = regr.predict(test_x)
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Mean sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y_ , test_y) )
```

Here the values of x and y are compared with the corresponding predictions.

The resulting output is:

```
Mean absolute error: 20.60
Mean sum of squares (MSE): 746.45
R2-score: 0.71
```

R^2 (R-Squared) is a goodness-of-fit measure for linear regression models. It is also known as the coefficient of determination.

In calculation, R^2 can be represented as:

R^2 = Variance_Explained_by_ the_Model/Total_Variance

This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively. R-squared measures the strength of the relationship between your model and the dependent variable on a convenient 0 – 100% scale. An R^2 of 1 indicates a perfect fit which is highly unlikely based on observed or measured data.
R-squared is always between 0 and 100%:
0% represents a model that does not explain any of the variation in the response variable around its mean. The mean of the dependent variable predicts the dependent variable as well as the regression model.
100% represents a model that explains all of the variation in the response variable around its mean.
Usually, the larger the R^2, the better the regression model fits your observations. However, this is not always true because some other factors not considered here.

Small R-squared values are not always a problem, and high R-squared values are not necessarily good!

Step 8: Putting it all Together

```
# Set up the environment by importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
#
# Read data from file FuelConsumptionCo2.xlsx inro a working data frame and plot it
df=pd.read_excel(r'FuelConsumptionCo2.xlsx')
#
# You may print the entire data frame but just a sample should do
df.head()
#
# Select some features for modeling, extract data for selected features. Plot them as well
data = df[["ENGINESIZE","CO2EMISSIONS"]]
plt.scatter(df["ENGINESIZE"] , df["CO2EMISSIONS"] , color="blue")
plt.xlabel("Engine Size")
plt.ylabel("Co2 Emissions")
plt.show()
#
data = df[["ENGINESIZE","CO2EMISSIONS"]]
# Generate training and testing data from the data frame
# using 80% data for training
train = data[:(int((len(data)*0.8)))]
test = data[(int((len(data)*0.8))):]
#
# Use sklearn package to model data
regr = linear_model.LinearRegression()
train_x = np.array(train[["ENGINESIZE"]])
train_y = np.array(train[["CO2EMISSIONS"]])
regr.fit(train_x,train_y)
#
# The coefficient or the slope of the regression line
print ("Coefficient : ",regr.coef_)
#
# The intercept of the regression line
print ("Intercept : ",regr.intercept_)
#
# Plot the regression line
plt.scatter(train["ENGINESIZE"], train["CO2EMISSIONS"], color='blue')
plt.plot(train_x, regr.coef_*train_x + regr.intercept_, '-r')
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

```
#
# Make prediction using the relationship of engine size to Co2 emissions by regression
# Create a function that can be called everytime a prediction is to be made
def get_regression_predictions(input_features,intercept,slope):
 predicted_values = input_features*slope + intercept
 return predicted_values
#
# Make a prediction for estimated emissions using a test engine size value
my_engine_size = 3.5
estimatd_emission =
get_regression_predictions(my_engine_size,regr.intercept_[0],regr.coef_[0][0])
print ("Estimated Emission :",estimatd_emission)
#
# Check the accuracy or reliability of prediction using statistical measures
from sklearn.metrics import r2_score
test_x = np.array(test[['ENGINESIZE']])
test_y = np.array(test[['CO2EMISSIONS']])
test_y_ = regr.predict(test_x)
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Mean sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y_ , test_y) )
```

You may copy and paste the entire coding shown above as a single run after you have save automobile data used in this case study as an MS Excel file in your working directory. This has been described in Step 2 above.

Valuable Resource:
https://medium.com/towards-artificial-intelligence/machine-learning-algorithms-for-beginners-with-python-code-examples-ml-19c6afd60daa