**Simple Linear Regression**

**About**

we learn how to use scikit-learn to implement simple linear regression. We download a dataset that is related to fuel consumption and Carbon dioxide emission of cars. Then, we split our data into training and test sets, create a model using training set, Evaluate your model using test set, and finally use model to predict unknown value.

**Understanding the Data**

### **FuelConsumption.csv:**

We have downloaded a fuel consumption dataset, **FuelConsumption.csv**, which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada.

* **MODELYEAR** e.g. 2014
* **MAKE** e.g. Acura
* **MODEL** e.g. ILX
* **VEHICLE CLASS** e.g. SUV
* **ENGINE SIZE** e.g. 4.7
* **CYLINDERS** e.g 6
* **TRANSMISSION** e.g. A6
* **FUEL CONSUMPTION in CITY(L/100 km)** e.g. 9.9
* **FUEL CONSUMPTION in HWY (L/100 km)** e.g. 8.9
* **FUEL CONSUMPTION COMB (L/100 km)** e.g. 9.2
* **CO2 EMISSIONS (g/km)** e.g. 182 --> low --> 0

#### **Creating train and test dataset**

Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the data. It is more realistic for real world problems.

This means that we know the outcome of each data point in this dataset, making it great to test with! And since this data has not been used to train the model, the model has no knowledge of the outcome of these data points. So, in essence, it is truly an out-of-sample testing.

msk = np.random.rand(len(df)) < 0.8

train = cdf[msk]

test = cdf[~msk]

### **Simple Regression Model**

Linear Regression fits a linear model with coefficients B = (B1, ..., Bn) to minimize the 'residual sum of squares' between the independent x in the dataset, and the dependent y by the linear approximation.

#### **Modeling**

Using sklearn package to model data.

from sklearn import linear\_model

regr = linear\_model.LinearRegression()

train\_x = np.asanyarray(train[['ENGINESIZE']])

train\_y = np.asanyarray(train[['CO2EMISSIONS']])

regr.fit (train\_x, train\_y)

# The coefficients

print ('Coefficients: ', regr.coef\_)

print ('Intercept: ',regr.intercept\_)

As mentioned before, **Coefficient** and **Intercept** in the simple linear regression, are the parameters of the fit line. Given that it is a simple linear regression, with only 2 parameters, and knowing that the parameters are the intercept and slope of the line, sklearn can estimate them directly from our data. Notice that all of the data must be available to traverse and calculate the parameters.

#### **Evaluation**

we compare the actual values and predicted values to calculate the accuracy of a regression model. Evaluation metrics provide a key role in the development of a model, as it provides insight to areas that require improvement.

There are different model evaluation metrics, lets use MSE here to calculate the accuracy of our model based on the test set:

- Mean absolute error: It is the mean of the absolute value of the errors. This is the easiest of the metrics to understand since it’s just average error.

- Mean Squared Error (MSE): Mean Squared Error (MSE) is the mean of the squared error. It’s more popular than Mean absolute error because the focus is geared more towards large errors. This is due to the squared term exponentially increasing larger errors in comparison to smaller ones.

- Root Mean Squared Error (RMSE).

- R-squared is not error, but is a popular metric for accuracy of your model. It represents how close the data are to the fitted regression line. The higher the R-squared, the better the model fits your data. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).