Machine Learning Using Python:

**Introduction to Scikit-Learn**

**Scikit-Learn** (sklearn) is a devoted machine learning platform introduced in Python. It provides simple and efficient tools for data mining and data analysis. It has been built upon Numpy, Scipy and Matplotlib modules which we have learnt so far. Moreover, sklearn is open source and commercially usable under BSD license.

Common prevalent techniques are pre-processing, classification, regression, clustering, dimensionality reduction, model selection etc. In our course, we will focus on the implementation of the following topics:

* Linear regression
* Logistic regression
* Decision trees
* Clustering

# Linear Regression

Linear Regression is a most commonly used machine learning technique to predict a target value based upon the relationship between the target value and multiple (or single) independent predictor features.

To understand the concept, we take Boston Housing dataset available within scikit-learn module. The objective is to predict the price of houses based on the given features. Let us store the dataset into a dataframe and list its features and the target value.

1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. import sklearn
5. from sklearn.datasets import load\_boston
6. boston = load\_boston()
7. print(boston.data.shape) *# (506, 13)*
8. print(boston.feature\_names) *# Column names*
9. print(boston.DESCR) *# brief desc.*
10. df = pd.DataFrame(boston.data)
11. print(df.head(5))

We observe top 5 samples of the dataset stored in dataframe df. Next, we assign the actual names of features to the dataframe columns.

1. df.columns = boston.feature\_names
2. print(df.head(5))

Moving ahead, we assign a new column to the dataframe with price value corresponding to each observation.

1. df['PRICE'] = boston.target
2. print(df.head(5))

We now have a complete dataframe with all the predictors and the target. Therefore, let us split the dataset into training and testing data where we consider 77% of the data under training.

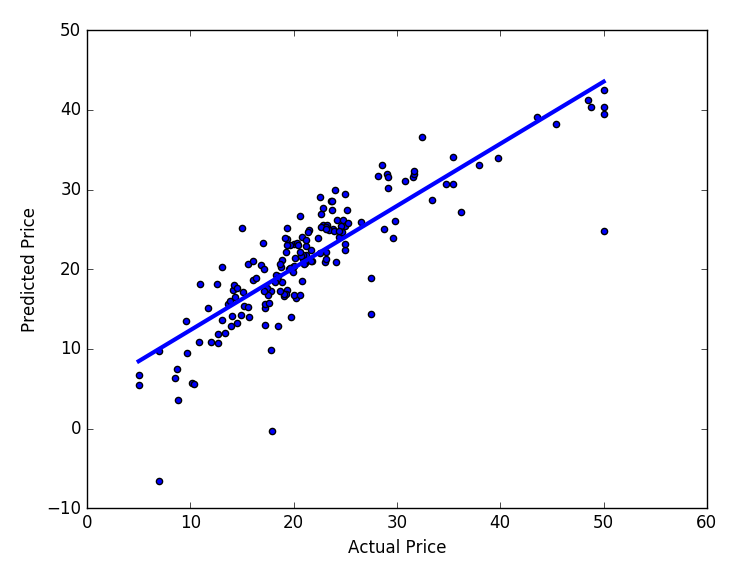
1. X = df.drop('PRICE', axis = 1)
3. *## (use sklearn.model\_selection in case sklearn.cross\_validation results into error)*
4. from sklearn.cross\_validation import train\_test\_split
5. X\_train, X\_test, y\_train, y\_test = train\_test\_split(
6. X, df.PRICE, test\_size = 0.33, *# Test data (33%)*
7. random\_state = 42) *# assign random\_state to any value, to get same samples on each fresh run*

With training and test data in hand, it's time to fit the Linear Regression model to make the necessary prediction of the data.

1. from sklearn.linear\_model import LinearRegression
2. lm = LinearRegression()
3. lm.fit(X\_train, y\_train)
4. pred\_test = lm.predict(X\_test)

We have the predicted data stored in pred\_test. Let us draw a scatter plot between predicted price and the actual price along with the best line of fit as shown.

1. plt.scatter(y\_test,pred\_test)
2. plt.plot(np.unique(y\_test),
3. np.poly1d(np.polyfit(y\_test, pred\_test, 1))(np.unique(y\_test)),
4. linewidth = 3)
5. plt.xlabel('Actual Price')
6. plt.ylabel('Predicted Price')



We can observe almost a significant linearity between both of the factors. The importance of each feature in predicting the price can be calculated using lm.coef\_. Also, analyzing the graph we still some variation in the data points near the best line of fit which can be mathematically found using Mean Square Error (MSE).

1. coeff\_df = pd.DataFrame(X\_train.columns, lm.coef\_)
2. print(coeff\_df)
3. mse = sklearn.metrics.mean\_squared\_error(y\_test, pred\_test)
4. print(mse)

# Logistic Regression

The Linear regression model is used when a given target attribute is quantitative or continuous in nature. However, to predict a categorical target attribute, we often rely upon the logistic regression which gives us a bounded probability within the range of [0,1].

To study the logistic regression, we take Iris dataset and determines the decision boundary using sepal width and height. To initiate, we load features into variable X and target species into variable Y.

2. import numpy as np
3. import matplotlib.pyplot as plt
4. from sklearn import linear\_model, datasets
5. iris = datasets.load\_iris()
6. X = iris.data[:, :2] *# Sepal Width and Sepal Height*
7. Y = iris.target *# Species -> Setosa, Versicolor, Virginica*

Next, we build the Logistic Regression model and set the inverse of regularization strength to 100,000 to avoid overfitting, since we have fewer data.

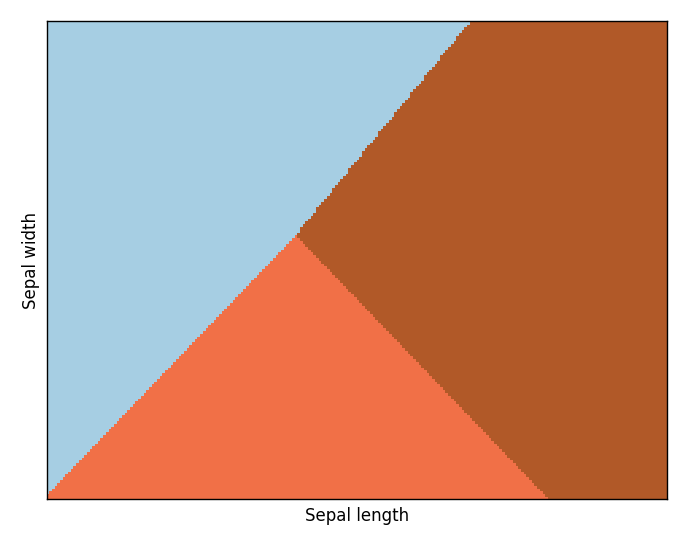
1. lm = linear\_model.LogisticRegression(C=1e5)
2. lm.fit(X, Y)

Once the model is fit, we're set to define the decision boundary for each species based on selected features. To do so, we create a mesh grid whose X and Y-axis range is equal to the minimum and maximum range of Sepal Height and Sepal Width.

1. *# Range of X and Y axis*
2. x\_min, x\_max = X[:, 0].min() - .5, X[:, 0].max() + .5
3. y\_min, y\_max = X[:, 1].min() - .5, X[:, 1].max() + .5
4. *# Step size in the mesh*
5. h = .02
6. *# Creating Mesh Grid*
7. xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

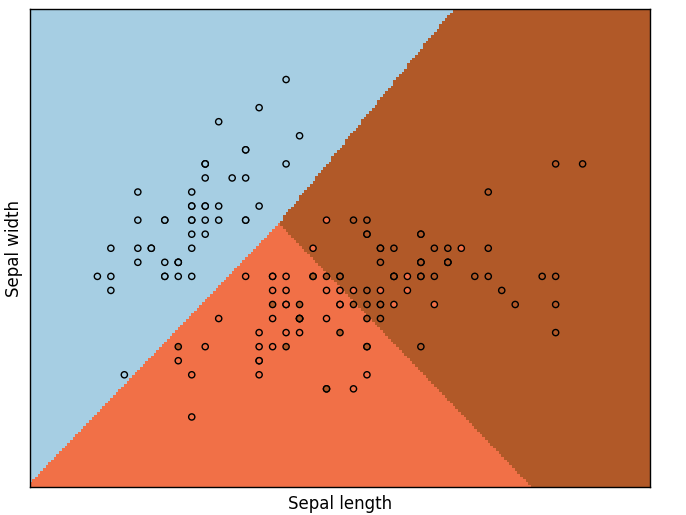
Next, we predict the decision boundary with the given data and visualize it using colored mesh as shown.

1. Z = lm.predict(np.c\_[xx.ravel(), yy.ravel()])
2. *# Put the result into a color plot*
3. Z = Z.reshape(xx.shape)
4. plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
5. *# Labeling*
6. plt.figure(1, figsize=(4, 3))
7. plt.xlabel('Sepal length')
8. plt.ylabel('Sepal width')
9. plt.xlim(xx.min(), xx.max())
10. plt.ylim(yy.min(), yy.max())
11. plt.xticks(());
12. plt.yticks(());
13. plt.show()



The colored boundaries classifies the species (in order: **Blue: Setosa, Orange: Versicolor** and **Brown: Virginica**) based on given sepal features. For more clarification, we can integrate data points on the plot as shown.

1. plt.scatter(X[:, 0], X[:, 1],c=Y, edgecolors='k', cmap=plt.cm.Paired)



# Decision Trees

Another form of supervised learning is decision tree. A decision tree is quite apt for classification and regression models. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

To demonstrate the working of a decision tree classifier, we take balance scale [dataset](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_0126429301951528961748/web-hosted/assets/data3.csv) ([information](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_0126429301951528961748/web-hosted/assets/res3.txt)) whose objective is to classify whether the scale has more load on right, left or centered. The attributes are the left weight, the left distance, the right weight, and the right distance.

Let us load the data into a dataframe and so forth get the training and the testing data. Assuming you have the dataset into an excel file format and stored in the same working directory.

2. import pandas as pd
3. from sklearn.cross\_validation import train\_test\_split
4. from sklearn.tree import DecisionTreeClassifier
5. from sklearn.metrics import accuracy\_score
6. df = pd.read\_excel('decision\_tree.xlsx')
7. X = df.values[:, 1:5] *# Other features*
8. Y = df.values[:,0] *# Target*
9. *# Splitting data*
10. X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, Y, test\_size = 0.3,
11. random\_state = 42)

A decision tree classifier can have two criterion i.e. Gini or Entropy. Let us build both of them to fit the model and achieve some predictions.

1. *# Gini Impurity*
2. lm\_gini = DecisionTreeClassifier(criterion = "gini", random\_state = 42,
3. max\_depth=3,
4. min\_samples\_leaf=5) *# min. samples req. at leaf node*
5. lm\_gini.fit(X\_train, y\_train)
6. y\_pred = lm\_gini.predict(X\_test)
7. *# Information Gain*
8. lm\_ig = DecisionTreeClassifier(criterion = "entropy", random\_state = 42,
9. max\_depth=3,
10. min\_samples\_leaf=5)
11. lm\_ig.fit(X\_train, y\_train)
12. y\_pred\_ig = lm\_ig.predict(X\_test)

The predictions in either of the cases have been stored in parameters y\_pred and y\_pred\_ig. Run the following snippet to get a certainty of the prediction.

2. print(y\_pred)
3. print(y\_pred\_ig)

We can now check the accuracy of each of the decision tree classifier criterion using the accuracy\_score.

1. print("Accuracy of Gini Impurity model: ",
2. accuracy\_score(y\_test, y\_pred) \* 100)
3. print("Accuracy of Information Gain model: ",
4. accuracy\_score(y\_test, y\_pred\_ig) \* 100)

Hence, we have achieved an equal accuracy score of 71.276%.

# KMeans Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). We discuss a general purpose clustering algorithm available under sklearn namely Kmeans. The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified.

To demonstrate the k-Means method, we take the [driver](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_0126429256644116481781/web-hosted/assets/data4.csv) local source dataset in which we estimate the clusters between Speed and Distance features. We load the dataset into a dataframe as shown, assuming you have saved the dataset into .xlsx/.csv format.

2. import pandas as pd
3. from sklearn.cluster import KMeans
4. import matplotlib.pyplot as plt
5. df = pd.read\_excel('kmeans.xlsx')
6. f1 = df['Distance\_Feature'].values*#[0:200]*
7. f2 = df['Speeding\_Feature'].values*#[0:200]*
8. X = pd.DataFrame({'dist': f1,
9. 'speed':f2})

Now, we build a KMeans model on the dataset and define the number of clusters, here 2.

1. kmeans = KMeans(n\_clusters = 2) *# don't exceed 4 else increase colormap*
2. k\_fit = kmeans.fit(X)
3. pred = kmeans.predict(X)

This fits the KMeans model on the given dataset and consists of the coordinates of the centroids for each of the two clusters which can be observed using cluster\_centers\_ attribute as shown:

1. centroids = kmeans.cluster\_centers\_
2. print(centroids)

Next, we can visualize the clusters using Matplotlib scatter plot, marking centroids with yellow color and clusters with distinct colors.

1. colmap = {1: 'r', 2: 'g', 3: 'b', 4: 'k'}
2. colors = list(map(lambda x: colmap[x+1], pred))
3. plt.scatter(f1, f2, color = colors,
4. alpha = 0.4, edgecolor = 'k')
5. plt.scatter(centroids[:, 0], centroids[:, 1], color = 'y')
6. plt.xlabel('Speed')
7. plt.ylabel('Distance')
8. plt.title('Number of Clusters: 2')
9. plt.show()

