Week 7 Laboratory Activity

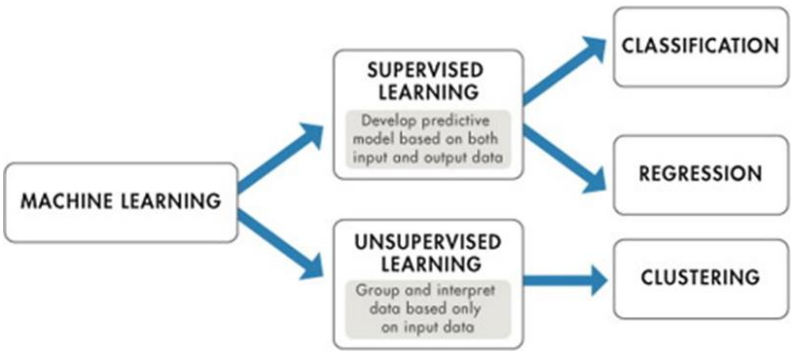
Many of the principles and theories of statistical machine learning are about the behaviour of different learning algorithms with varying amounts of data. This week, we will explore classification using the Decision Trees (DT) algorithm, clustering using the k-means algorithm (k-means), and be exposed to various cloud machine learning services.

Work through Week 7 Laboratory Activities to experiment with different aspects of Decision Trees (DT), Random Forest (RF) and K-means within the Jupyter Notebook environment. Share with other students and tutor your findings (during your tutorial session) from the activities. You are also encouraged to discussion among yourselves or with your tutors on your thoughts on these algorithms in the Moodle forums:

Activity 1: Classification and Decision trees and Random forest as two examples.
Activity 2: Clustering and K-means algorithm as an example.
Activity 3 (Optional): Using Cloud Services, such as GCP, AWS, Azure, Aliyun, DataRobots

Activity 1: Classification

We are going to start this week's practical with classification, which in machine learning, it part of the **supervised** learning style (as per Lectures in Week 5)



Brownlee, J. (2016). Supervised and Unsupervised Machine Learning Algorithms

One of the main things to note for supervised learning is that there is a **train and test datasets** (we won't look at validation datasets in this Unit).

For this activity, we will be attempting to predict whether a person has travelled abroad based on the person's age and income. Let's start with the usual importing of the necessary libraries.

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

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

Investigate the content of the dataset (as we would normally).

```
dataset.shape

dataset.head()
```

The dataset should have 400 rows and 3 columns. The columns should be 'Age', 'Income' and 'TravelAbroad'. This is a record of past incidences or survey data. As the intention is to build a model, we are going to use this as the training data.

Practice 1: If we use the whole dataset as the training data, how do we know how good our model is?

We proceed to split the dataset into input data and their corresponding labeled data. The first two columns are the input data, i.e. the 'Age' and the 'Income' of which we would like to predict if the person has 'TravelAbroad' before. From the past survey data collected, we have some samples and this 'TravelAbroad' is the respective labeled data. Let's split the data into X, the input data and y, the labeled data.

```
X = dataset.iloc[:, [0, 1]].values  # Input Data: Age and Income
y = dataset.iloc[:, 2].values       # Labeled Data: Travelled or not
```

You can review what X and y consists of. Since, we need to reserve a portion of the dataset for testing data, we can either simply take a portion of the data or a better way is take from a "random" sample of the data. Assuming that we want to keep 75% for the training data and 25% for the testing data. We can use the function `train_test_split()` from `sklearn.model_selection`.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.25, random_state = 0
)
```

Feature Scaling or Normalization

Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without feature scaling (a.k.a. normalization). For example, many classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed (heavily influenced) by this particular feature. Therefore, the range of all features should be normalized in order for each feature to contribute proportionately to the final distance. In addition, our motivation here is to visualise the feature space later on.

(The above paragraph may be difficult to understand, and it is for you to try to search online to figure this out and/or share what you know about it with others on the forum or informally on Slack).

Again, we use a built in function `StandardScaler()` from sklearn.preprocessing to do this. Note that we apply the `fit_transform()` to the training dataset and not the test dataset. The reason is that we want to scale it to have a mean of 0 and a certain standard deviation. The mean and standard deviation values will be stored in the `StandardScaler()` and applied to the testing dataset using the `transform()`.

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Train and Test using Decision Tree Algorithm

Now, our data is ready to be used to build the model. Note that, in reality, we will usually need to read and wrangle the data before we even start to normalize or standardize the data (although this stage does not necessarily occur in all cases). Let's use the Decision Tree algorithm, from `sklearn.tree`'s `DecisionTreeClassifier()` function.

```
# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(
    criterion = 'entropy', random_state = 0
)
classifier.fit(X_train, y_train)
```

Normally, when we want to build a model, most functions will have a method called `fit()` (note that this is not standard but many functions use this). At the end of running that code, we now have a model using the Decision Tree algorithm and the model is called `classifier`. Since we now have a model built, let's test how well our model is. We firstly use the built model with the testing input data (not labels as that's the output that we want to check against what we already know). We use the method `predict()` in the model. Note that this method's name is also quite common with other built-in functions for models, but many also do call this `pred()`.

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

We now have a set of labels, `y_pred`, that is the output from the prediction of using our testing data. We now need to compare this `y_pred` with the actual `y_test` (the true value) and determine the accuracy of our model's prediction. To do so, we want to view it using a confusion matrix (covered in your lectures). This is an important concept for classifier comparison.

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Your confusion matrix will be shown as a 2 by 2 matrix.

Practise 2: From the confusion matrix output, discuss what are the values of True Negative, True Positives, False Negative, and False Positive. Other than accuracy, would precision or recall be a better measure for this particular case?

Visualise (Optional)

We will take this opportunity to introduce a new visualisation, called the meshgrid. For this demonstration, we will look at the test results in the feature space. You do not need to know the syntax or how it works. This is just for illustration.

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(
    np.arange(
        start = X_set[:, 0].min() - 1,
        stop = X_set[:, 0].max() + 1,
        step = 0.01
    ),
    np.arange(
        start = X_set[:, 1].min() - 1,
        stop = X_set[:, 1].max() + 1,
        step = 0.01
    )
)

plt.contourf(
    X1,
    X2,
    classifier.predict(
        np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
    alpha = 0.75,
    cmap = ListedColormap(('red', 'green'))
)

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(
        X_set[y_set == j, 0],
        X_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i),
        label = j
    )

plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Income')
plt.legend()
plt.show()
```

Train and Test using Random Forest Algorithm

Another classifier is the random forests algorithm, which in essence consists of multiple trees, each based on a random sample of the training data. They usually perform rather well. Let's now fit a Random Forest Classification to the Training set. The concepts are similar to the above, just that we call a different model. All the reading of data, splitting of data and data normalization has been done above.

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(
    n_estimators = 20,
    criterion = 'entropy',
    random_state = 0
)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Practice 3: Discuss the results of the above confusion matrix. What is the prediction accuracy? Compare with the decision tree results.

In machine learning algorithms, one way to improve the test accuracy is to tune the parameters (e.g. number of trees in random forest algorithm). Tuning based on the parameters of the model building is called hyper-parameter tuning. Try the above using `n_estimators = 40`. Do you get better results?

Activity 2: Clustering

The next and last part of FIT1043 on machine learning will be about clustering. This is an **unsupervised** machine learning style, where there is no training and testing of the models but models are built by finding patterns within the dataset. We will use the k-means clustering algorithm for this activity.

Understanding Clustering (k-means)

We will illustrate a clustering task using K-means clustering in a 2-dimensional space. We will not look at the theory of k-means clustering; we will just use it as a black box in order to understand some fundamental concepts of segmentation and clustering. So this is a tutorial without mathematics or analysis. Now it's important to realise that in data science you will rarely use a simple 2-dimensional modelling. However, the simple nature of the material means we can carefully study the different aspects of learning. So this is excellent material for a tutorial. The material for this tutorial has been provided from this link.

k-means Clustering

The goal in K-means clustering is to find k subgroups in a given dataset. Please note the variable k represents the number of subgroups in the data. k-means assigns each data point in the dataset to one of the k subgroups iteratively based on the variables (features) in the dataset.

The output of the k-means clustering are:

1. k cluster centers. The centers can be used to find the relevant subgroup for a new data point.
2. Labels for all data points in the given dataset.

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

# Load the dataset from Moodle
df = pd.read_csv('Drivers.csv')
df.head()
```

As usual, we will look at the data and also the size of the dataset.

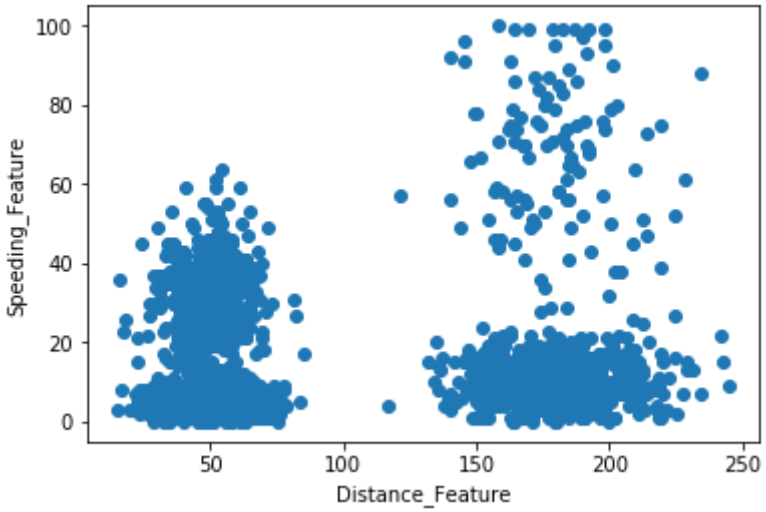
```
# Check the dataset dimension
df.shape
```

Visualise

Let's have a visual look at our data points.

```
#Visualise the dataset (Distance vs Speeding)
plt.scatter(x=df['Distance_Feature'],y=df['Speeding_Feature'])
plt.xlabel('Distance_Feature')
```

```
plt.ylabel('Speeding_Feature')
```



From a manual inspection of the data points, we probably can see 2 or 3 groups. We proceed to ask the machine to learn from the input data to determine the groupings.

k-Means

First thing to note is that for *k*-means, the machine learning algorithm does not know how many clusters it should determine and the default *k*-means algorithm expects the user to provide that information (there are advanced versions that tries to determine the most optimal number of clusters, but that's beyond this unit's scope).

```
# Run the K-means clustering over the dataset using only
# distance and speeding features. Set K=2: we only want
# to cluster the dataset into two subgroups
kmeans = KMeans(n_clusters=2).fit(
    df[['Distance_Feature','Speeding_Feature']]
)
```

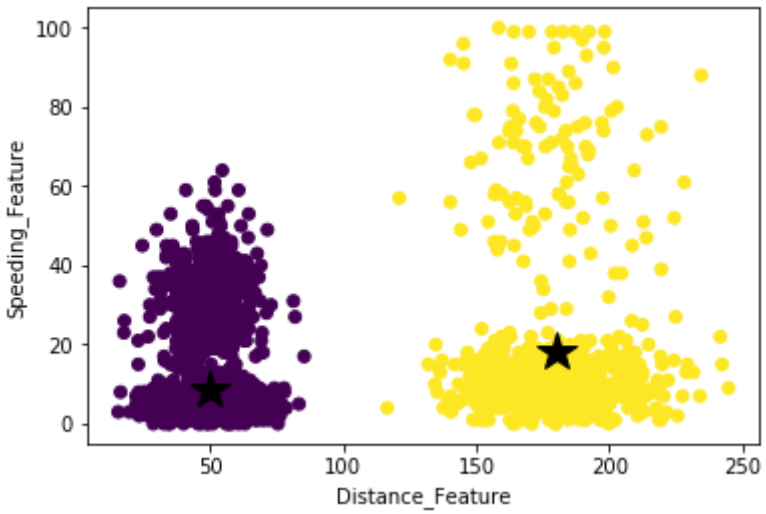
Similarly to how we do data auditing, prior to visualisation, we may want to have a look at some of the results of the clustering.

```
# Look at the outputs: Two cluster centers
kmeans.cluster_centers_

# Look at the outputs: Cluster labels
kmeans.labels_
```

Visualise

```
# Visualise the output labels
plt.scatter(
    x=df['Distance_Feature'],
```

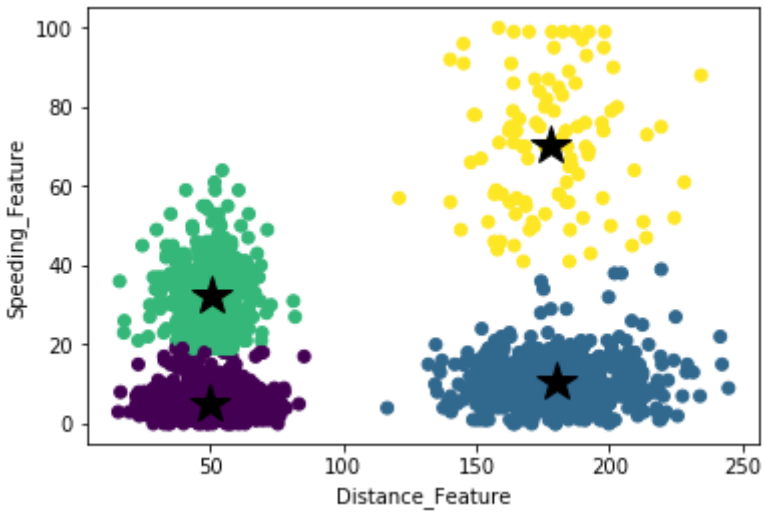
```
    y=df['Speeding_Feature'],
    c=kmeans.labels_)

# Visualise the cluster centers (black stars)
plt.plot(
    kmeans.cluster_centers_[:,0],
    kmeans.cluster_centers_[:,1],
    'k*',
    markersize=20
)
plt.xlabel('Distance_Feature')
plt.ylabel('Speeding_Feature')
plt.show()
```



Practise 4: Discuss what type of drivers do each group present?

Let's try to run *k*-means with different *k* values to get more clusters.

```
# Run K-means with another K value
# Set K=4: we want to cluster the dataset into four subgroups
kmeans2 = KMeans(n_clusters=4).fit(
    df[['Distance_Feature','Speeding_Feature']]
)

# Visualise the output labels
plt.scatter(
    x=df['Distance_Feature'],
    y=df['Speeding_Feature'],
    c=kmeans2.labels_
)

# Visualise the cluster centers (black stars)
plt.plot(
    kmeans2.cluster_centers_[:,0],
    kmeans2.cluster_centers_[:,1],
    'k*',
    markersize=20
)

plt.xlabel('Distance_Feature')
plt.ylabel('Speeding_Feature')
plt.show()
```



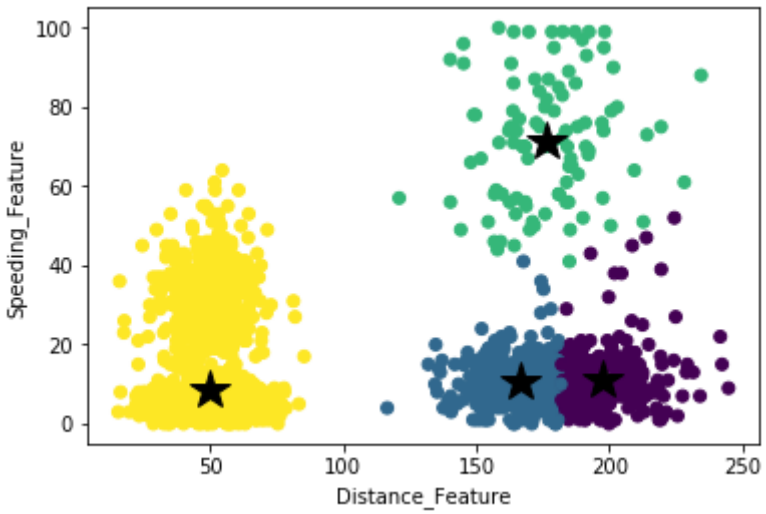
K-means initialization: try adding `init='random'` to the code like this:

```
kmeans2 = KMeans(n_clusters=4, init='random').fit(
    df[['Distance_Feature','Speeding_Feature']]
)
```

and run it multiple times (code below).

Practise 5: Does the clustering change? Why?

```
kmeans2 = KMeans(n_clusters=4, init='random').fit(
    df[['Distance_Feature','Speeding_Feature']]
)

plt.scatter(
    x=df['Distance_Feature'],
    y=df['Speeding_Feature'],
    c=kmeans2.labels_)

# Visualise the cluster centers (black stars)
plt.plot(
    kmeans2.cluster_centers_[:,0],
    kmeans2.cluster_centers_[:,1],
    'k*',
    markersize=20
)

plt.xlabel('Distance_Feature')
plt.ylabel('Speeding_Feature')
plt.show()
```



Activity 3: Exploring Cloud Machine Learning Tools

You have now had some exposure to the basics of machine learning, with some understanding of regression, classifiers such as decision trees and random forest, and clustering with k-means. These provide you with the basic machine learning foundations.

Other than the tools that are available in Python, many of the cloud platform providers have their own machine learning algorithms available. Note that like these Python libraries, the algorithms are standard implementation. As a data scientist, you should have the necessary background to augment these algorithms to maybe suit your needs better.

Nevertheless, this last activity for Week 7 is for you to explore:

- AWS Machine Learning
- Google Cloud AI and Machine Learning
- Microsoft Azure Machine Learning
- AliCloud Machine Learning
- IBM Watson Machine Learning

You should be able to obtain some trial credits for some of them.