National University of Singapore
School of Computing
Tutorial 4:
K-MEANS CLUSTERING

Introduction to the K-Means Clustering Technique

K-Means is a unsupervised clustering algorithm that takes a bunch of unlabelled points and tries to group them into 'k' number of clusters. It works in the following way:

- Step 1: Choose the number of clusters K (eg: 3 clusters)
- Step 2: Randomly assign the center of each cluster (mathematically, this is the **mean** of the points)
- Step 3: Assign each point to the cluster closest to it
- Step 4: Calculate cluster centroids again (this will **move** the mean)
- Step 5: Repeat steps 4 and 5 until we reach global optima where no improvements are possible and no switching of data points from one cluster to other.
- Step 6: Repeat the above steps for K values of 2 to *k*. The ideal number of clusters will be at the 'elbow' when you plot the scores of the different clusters.

1. Download the cars dataset from (https://www.kaggle.com/abineshkumark/carsdata) and import it into your notebook/IDE

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

df = pd.read_csv('cars.csv')
df.columns = ['mpg',
              'cylinders',
              'cubicinches',
              'hp',
              'weightlbs',
              'time-to-60',
              'year',
              'brand']
```

NOTE: IGNORE THE COLUMN CALLED 'BRAND' FOR THE MOMENT. Since we are currently implementing unsupervised learning, we are in a scenario where we don't actually know the brand. Instead, we can imagine that we are trying to find different categories of cars within the group of cars in our dataset.

Drop all the columns except 'cylinders and 'weight'. (Doing this exercise with only 2 features will allows us to plot and visualize what is going on with the K Means algorithm)

```
df_kmeans = df[['cylinders', 'weightlbs']].copy()
```

2. Normalize your data using SKLearn's standard scaler.

```
# Normalize your data
scaler = StandardScaler()

# Check for blank spaces and replace with NaNs
df_kmeans.replace(r'\s+|^$', np.nan, regex=True, inplace=True)
df_kmeans.isnull().sum()

# Drop NaN values
df_kmeans.dropna(inplace=True)
df_kmeans.isnull().sum()

df_kmeans_normalized = scaler.fit_transform(df_kmeans)
df_kmeans_normalized = pd.DataFrame(df_kmeans_normalized)
df_kmeans_normalized.info()
```

3. Fit the data onto the K-means algorithm using k=3. Print the cluster centers and the score.

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(df_kmeans_normalized)
print('\nCluster centres:')
print(kmeans.cluster_centers_)
```

4. Visualize where the clusters are using a matplotlib scatterplot (hint: use the k-means attrbute 'labels_' and the matplotib parameter 'c', which stands for labels')

```
labels = kmeans.labels_

plt.scatter(df_kmeans_normalized[0],df_kmeans_normalized[1],
        c=labels, cmap=plt.cm.rainbow)

plt.xlabel('cylinders')
plt.ylabel('weightlbs')
plt.show()
```

5. Repeat this for *k* of 2-10 using a for loop. Can you plot the scores using Matplotlib to visualize the elbow?

```
distance_to_cluster_centre = []

for k in range(2,10):
   kmeans = KMeans(n_clusters=k)
   kmeans.fit(df_kmeans_normalized)
```

```
        distance = np.min(kmeans.transform(df_kmeans_normalized),axis=1)
        average_distance = np.mean(distance)
        distance_to_cluster_centre.append(average_distance)

    clusters = np.arange(len(distance_to_cluster_centre))+1
    plt.plot(clusters, distance_to_cluster_centre)
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Average distance to closest cluster centroid')
    plt.ylim(0,1.5)
    plt.show()
```

1-4 John Ang,
5 Xiaofei Sun