

# CSE-3024 Web Mining

## Lab Assignment - 4

Alokam Nikhitha

19BCE2555

# K-Means

## Problem statement:

Illustrate the k-means clustering to cluster the data points for at least five epoch properly.

How to Implementing K-Means Clustering?

- Using the elbow method to determine the optimal number of clusters for kmeans clustering
- Visualising the clusters
- Plotting the centroids of the clusters

## Dataset used:

- Shopping-data
- <https://archive.ics.uci.edu/ml/machine-learning-databases/>

## Procedure:

- Import necessary libraries - sklearn, numpy, pandas, etc.
- Using pandas, we first import the dataset into our workspace.
- Select the number of clusters for the dataset ( K )
- Select K number of centroids
- By calculating the Euclidean distance or Manhattan distance assign the points to the nearest centroid, thus creating K groups

- Now find the original centroid in each group
- Again reassign the whole data point based on this new centroid, then repeat step 4 until the position of the centroid doesn't change.
- Using the elbow method to determine the optimal number of clusters for kmeans clustering
- Visualising the clusters and Plotting the centroids of the clusters

## Code:

```
#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#Importing the Datasets
dataset = pd.read_csv('shopping-data.csv')
X = dataset.iloc[:, 3:].values

#Elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300,
n_init=10)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
```

```
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```

```
#Applying Kmeans to the dataset
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300,
n_init=10);
y_kmeans = kmeans.fit_predict(X)
```

```
#Printing out the cluster each input belongs to
y_kmeans
```

```
# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c =
'red', label = 'Standard Customers')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c =
'blue', label = 'Careless Customers')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c =
'cyan', label = 'Target Customers')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c =
'magenta', label = 'Sensible Customers')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c =
'green', label = 'Careful Customers')
plt.scatter(kmeans.cluster_centers_[0, 0],
kmeans.cluster_centers_[0, 1], s = 300, c = 'yellow', label =
'Centroids')
plt.title ('Clusters of Clients')
plt.xlabel ('Annual Income (k$)')
plt.ylabel ('Spending Score (1-100)')
plt.legend()
plt.show()
```

## Code Snippets and Outputs:

```
In [1]: #importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
```

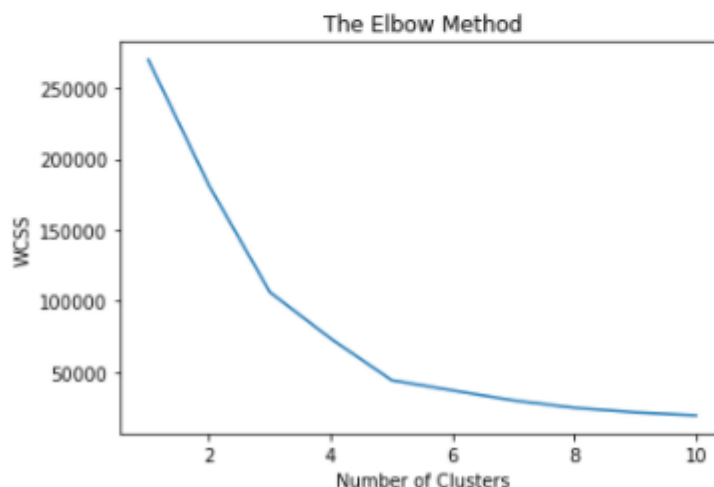
Import necessary libraries - sklearn, numpy, pandas, etc.

```
In [2]: dataset = pd.read_csv('shopping-data.csv')
X = dataset.iloc[:, 3:].values
```

Using pandas, we first import the dataset into our workspace and are assigning the income attribute along with shopping score as independent variables.

```
In [4]: #Elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



Here we are plotting a graph that marks Within Cluster Sum of Squares (WCSS) with the increase in number of clusters. We can see an elbow formation when the number of clusters is 5 and hence, we assume that optimal number of clusters in our dataset is 5

```
In [5]: #Applying Kmeans to the dataset
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10);
y_kmeans = kmeans.fit_predict(X)
```

Here we are training our k-means model with 5 clusters. We are also generating the y\_kmeans array that stores the cluster index of each input attribute from 0 to 4

```
In [9]: y_kmeans
Out[9]: array([3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0,
               3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0,
               3, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
               4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
               4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
               4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
               4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
               4, 2, 1, 2, 1, 2, 1, 2, 1, 2, 4, 2, 1, 2, 1, 2, 1, 2, 1, 2,
               1, 2, 1, 2, 1, 2, 1, 2, 4, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
               1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
               1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
               1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
               1, 2])
```

Here we are printing our y\_kmeans array and we can see that each input cell is assigned a value between 0 and 4, both inclusive. This corresponds to the cluster index of each input.

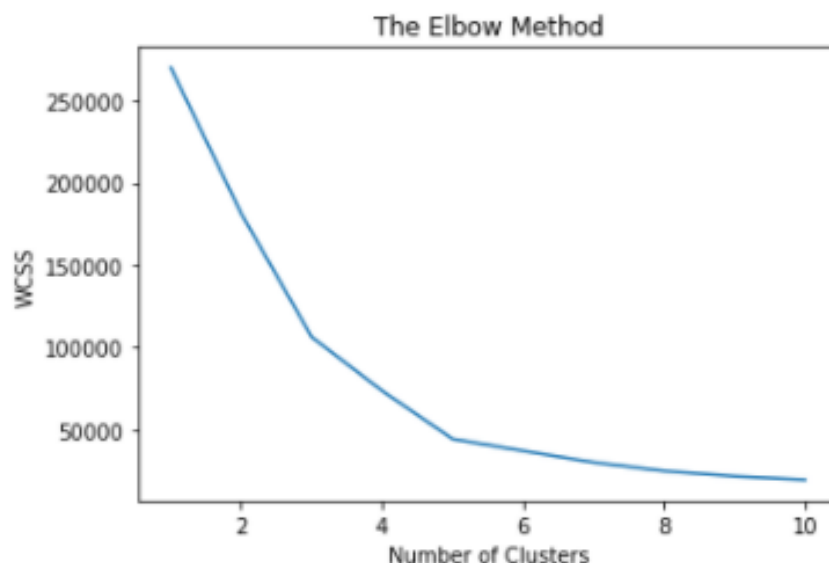
```
In [6]: # Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'yellow', label = 'Standard Customers')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'magenta', label = 'Careless Customers')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Target Customers')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'blue', label = 'Sensible Customers')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'cyan', label = 'Careful Customers')
plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1], s = 300, c = 'red', label = 'Centroids')
plt.title('Clusters of Clients')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



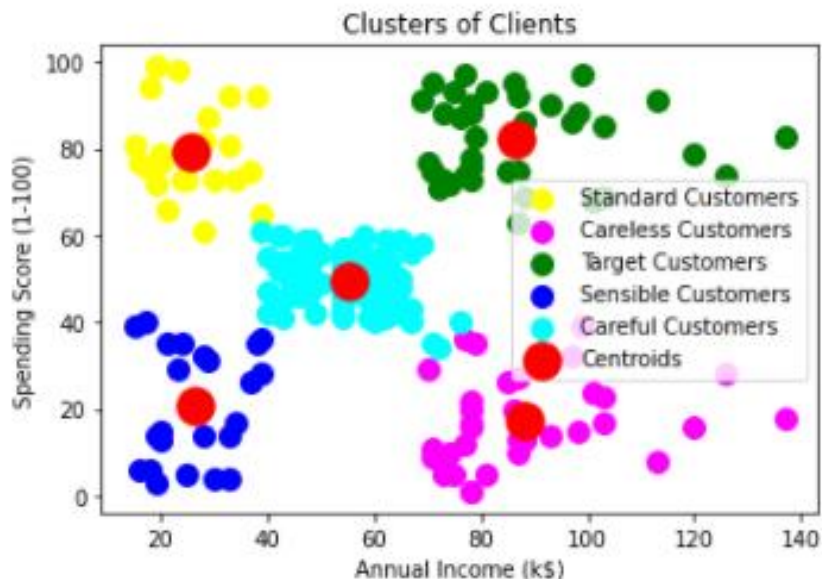
Here we have visualized our results. We have labelled different clusters as blue, green, pink, yellow and cyan. Each cluster correspond to different category of target audience. We have also marked centroids of each cluster which are red in colour.

## Results and Output

### Elbow Method Graph



## Clustering Graph:



Here different clusters are marked as blue, yellow, magenta, cyan and green. The red dot over each cluster represents its centroid.

We can categorise these clusters as: -

- Yellow Cluster corresponds to careless customers as they have low income but high spending.
- Blue Cluster as Sensible customers, becoz they have low income and low spending.
- Cyan Clusters are standard cluster that suggest they have median income and median spending.
- The pink coloured cluster correspond to Target Customers, as they have high income but low spending, the shopping company can give them offers and attractions as they are capable of spending more but they aren't doing it currently.
- Finally, the Green coloured clusters are Careful customers. They have high income and thus high spending as well.



# Random Forest

## Question:

The following are the basic steps involved in performing the random forest algorithm:

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
4. In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

## Dataset Used:

petrol\_consumption.csv, bill\_authentication.csv.

## Procedure:

- Using pandas, we first import the dataset into our workspace.
- Next we define the set of dependent and independent attributes.
- We then import the random forest regressor from sklearn `rn.ensemble` and train our model using the independent and dependent attributes.
- Next, we have printed the results of independent set as predicted by our regressor.
- Lastly, To check for the performance of our dataset, we have printed all the evaluation metrics

Since it has less Number of Rows we haven't split the dataset

## Petrol consumption dataset

### Code

```
#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#Importing the Dataset
dataset = pd.read_csv("petrol_consumption.csv")

#First few rows of our dataset
dataset.head(10)

#Checcking for null values
print(dataset.info())

X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, -1].values

#Training our Random Forest Regression Model
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=200, random_state=0)
regressor.fit(X, y)

#Predictions by Regressor
y_pred = regressor.predict(X)

#Printing Mean Absolute Error
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y, y_pred)

#Printing Mean Absolute Error
from sklearn.metrics import mean_squared_error
mean_squared_error(y, y_pred)

#Printing Root Mean Squared Error
np.sqrt(mean_squared_error(y, y_pred))

#Printing Root Mean Sqaured Log Error
```

```
np.log(np.sqrt(mean_squared_error(y, y_pred)))
```

```
#Printing R-square value
from sklearn.metrics import r2_score
r2_score(y, y_pred)
```

## Code Snippets and Explanation:

```
In [1]: #Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Here we are importing the required Libraries

```
In [2]: #Importing the Dataset
dataset = pd.read_csv("petrol_consumption.csv")
```

Using Pandas we are importing the data

```
In [3]: #First few rows of our dataset
dataset.head(10)
```

Out[3]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410
5	10.0	5342	1333	0.571	457
6	8.0	5319	11868	0.451	344
7	8.0	5126	2138	0.553	467
8	8.0	4447	8577	0.529	464
9	7.0	4512	8507	0.552	498

Printing the first few rows.

```
In [4]: #Checking for null values
print(dataset.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48 entries, 0 to 47
Data columns (total 5 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Petrol_tax                            48 non-null     float64
1   Average_income                       48 non-null     int64
2   Paved_Highways                       48 non-null     int64
3   Population_Driver_licence(%)         48 non-null     float64
4   Petrol_Consumption                   48 non-null     int64
dtypes: float64(2), int64(3)
memory usage: 2.0 KB
None
```

**Here we are checking for the null values.**

```
In [5]: #Set of independent and dependent attributes
X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, -1].values
```

```
In [6]: #Training our Random Forest Regression Model
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=200, random_state=0)
regressor.fit(X, y)
```

```
Out[6]: RandomForestRegressor(n_estimators=200, random_state=0)
```

**We have Defined set of Dependent and Independent attributes. The `n_estimators` here indicate the number of decision trees that we are using to train our random forest regressor. Hence we are using 200 decision trees for prediction. For final value we have used the average value of each decision tree to find the final consumption of petrol of a particular region.**

```
In [7]: #Predictions by Regressor  
y_pred = regressor.predict(X)
```

```
In [8]: #Printing Mean Absolute Error  
from sklearn.metrics import mean_absolute_error  
mean_absolute_error(y, y_pred)
```

Out[8]: 16.542083333333327

## Printing the Mean Absolute Error

```
In [9]: #Printing Mean Absolute Error  
from sklearn.metrics import mean_squared_error  
mean_squared_error(y, y_pred)
```

Out[9]: 676.4954427083334

## Printing the Mean Squared Error

```
In [10]: #Printing Root Mean Squared Error  
np.sqrt(mean_squared_error(y, y_pred))
```

Out[10]: 26.00952599930136

## Printing the Root Mean Squared Error

```
In [11]: #Printing Root Mean Squared Log Error  
np.log(np.sqrt(mean_squared_error(y, y_pred)))
```

Out[11]: 3.258462855507552

## Printing the Root Mean Squared Log Error

```
In [12]: #Printing R-square value  
from sklearn.metrics import r2_score  
r2_score(y, y_pred)
```

Out[12]: 0.9448102799874128

## Printing the R-square value

## Results and Conclusions:

Mean Absolute Error from cell8 is 16.542083333333327

Mean absolute error from cell 9 is 676.4954427083334

Root Mean Squared Error from cell10 is 26.00952599930136

Root Mean Squared Log Error from cell11 is 3.258462855507552

R-square value from cell12 is 0.9448102799874128

## Bill\_authentication dataset

### Code

```
#Importing Libraries  
import pandas as pd
```

```
#importing the bill_authentication dataset  
dataset = pd.read_csv('bill_authentication.csv')
```

```
#Displaying the first few rows of the dataset  
dataset.head()
```

```
X = dataset.iloc[:, 0:4].values  
y = dataset.iloc[:, 4].values
```

```
#Training our Random Forest Regression Model  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()
```

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
from sklearn.ensemble import RandomForestClassifier  
classifier= RandomForestClassifier(n_estimators=20, random_state=0)
```

```
classifier.fit(X_train, y_train)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix,  
accuracy_score  
print(confusion_matrix(y_test,y_pred))
```

```
#printing classification_report  
print(classification_report(y_test,y_pred))
```

```
#printing Accuracy  
print(accuracy_score(y_test, y_pred))
```

## Code Snippets and Explanation

```
In [1]: #Importing Libraries  
import pandas as pd
```

```
In [2]: #importing the bill_authentication dataset  
dataset = pd.read_csv('bill_authentication.csv')
```

Here we are importing the required Libraries. Using Pandas we are importing the data

```
In [3]: #Displaying the first few rows of the dataset  
dataset.head()
```

Out[3]:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	1

Printing the first few rows.

```
In [7]: X = dataset.iloc[:, 0:4].values
        y = dataset.iloc[:, 4].values
```

## Defining the Dependent and Independent variables

```
In [9]: #Training our Random Forest Regression Model
        from sklearn.preprocessing import StandardScaler
        sc = StandardScaler()
```

## Here we are training our Random forest Regression model

```
In [12]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
In [13]: from sklearn.ensemble import RandomForestClassifier
        classifier = RandomForestClassifier(n_estimators=20, random_state=0)
        classifier.fit(X_train, y_train)
```

```
Out[13]: RandomForestClassifier(n_estimators=20, random_state=0)
```

```
In [14]: y_pred = classifier.predict(X_test)
```

```
In [15]: from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
        print(confusion_matrix(y_test, y_pred))
```

```
[[147  6]
 [ 7 115]]
```

## Here we are printing the Confusion Matrix

```
In [16]: #printing classification_report
        print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.95	0.96	0.96	153
1	0.95	0.94	0.95	122
accuracy			0.95	275
macro avg	0.95	0.95	0.95	275
weighted avg	0.95	0.95	0.95	275

## Here we are printing the Classification Report



```
In [17]: #printing Accuracy
print(accuracy_score(y_test, y_pred))

0.9527272727272728
```

**The Accuracy of the model is 0.9527272727272728**

## Results and Conclusion

### Confusion Matrix

```
[[147  6]
 [ 7 115]]
```

### Classification Report

	precision	recall	f1-score	support
0	0.95	0.96	0.96	153
1	0.95	0.94	0.95	122
accuracy			0.95	275
macro avg	0.95	0.95	0.95	275
weighted avg	0.95	0.95	0.95	275

**Accuracy of the dataset is: 0.9527272727272728**