Exercise-7

AIM: Implementation of K-means Clustering algorithm.

DESCRIPTION:

K-Means Clustering Algorithm:

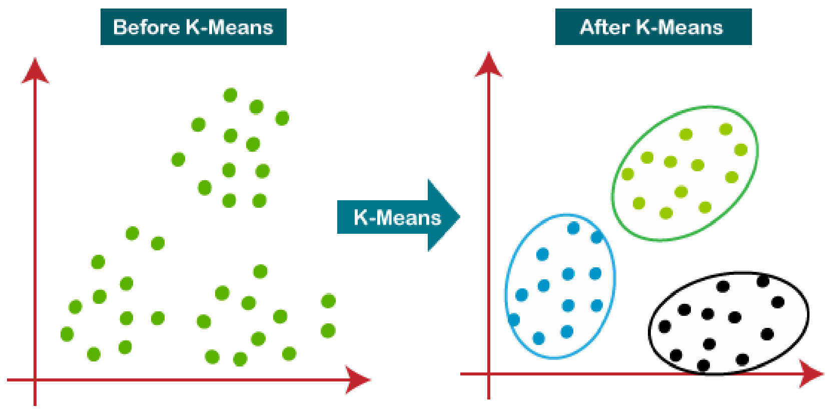
K-Means Clustering is an **unsupervised learning algorithm** that is used to solve the **clustering problems in machine learning**, which groups the **unlabelled dataset into different clusters.** Here **K** defines the number of **pre-defined clusters** that need to be created in the process, as if **K=2, there will be two clusters**, and for **K=3, there will be three clusters**, and so on.

It allows us to cluster the data into **different groups** and a convenient way to discover the categories of groups in the unlabelled dataset on its own without the need for any training. It is a **centroid-based algorithm**, where each cluster is associated with a centroid. The **main aim** of this algorithm is to **minimize the sum of distances between the data point** and their **corresponding clusters.**

The algorithm takes **the unlabelled dataset as input**, divides the dataset into **k-number of clusters,** and **repeats the process until** **it does not find the best clusters**. The value of k should be predetermined in this algorithm.

The **k-**means **clustering algorithm** mainly performs **two tasks:**

- Determines the **best value for K center points** or centroids by an iterative process.
- Assigns **each data point** to its **closest k-center.** Those data points which **are near to the particular k-center create a cluster.**



K-Means Algorithm:

The **working of the K-Means algorithm** is explained in the **below steps:**

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which mean reassign each data point to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Python Implementation of K-means Clustering Algorithm:

We have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there. In the given dataset, we have **Customer_Id, Gender, Age, Annual Income ($), and Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). **From this dataset**, we need to **calculate some patterns**, as it is an **unsupervised method**, so we don't know what to calculate exactly.

The **steps to be followed for the implementation** are given below:

- Data Pre-processing
- Finding the optimal number of clusters using the elbow method
- Training the K-means algorithm on the training dataset
- Visualizing the clusters

1. Data Pre-processing:

(a) Importing Libraries: firstly, we will import the libraries for our model, which is part of data pre-processing. The code is given below:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

(b) Importing the Dataset: Next, we will import the dataset that we need to use. So here, we are using the **Mall_Customer_data.csv** dataset. It can be imported using the below code:

# Importing the dataset

```
dataset = pd.read_csv('D:\AI TOOLS(PVP-19)\AI TOOLS LAB\WEEK-3(ML using
            USL)\Mall_Customers.csv');
```

(c) Extracting Independent Variables: Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

```
x = dataset. iloc [:, [3, 4]].values
```

Step-2: Finding the optimal number of clusters using the elbow method

In the second step, we will try to find the **optimal number of clusters** for our clustering problem. So, as discussed above, here we are going to **use the elbow method** for this purpose. The Elbow method is one of the most popular ways to **find the optimal number of clusters**. This method uses the concept of WCSS value.

As we know, the **elbow method** uses the **WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis**. An ideal way to figure out the right number of clusters would be to calculate the **Within-Cluster-Sum-of-Squares (WCSS).** WCSS is the sum of squares of the distances of each data point in all clusters to their respective centroids. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10.

$$WCSS= \sum_{P_i \text{ in Cluster1}} distance(P_i\ C_1)^2 +\sum_{P_i \text{ in Cluster2}} distance(P_i\ C_2)^2+\sum_{P_i \text{ in CLuster3}} distance(P_i\ C_3)^2$$

In the above formula of WCSS,

∑Pi in Cluster1 distance(Pi C1)2: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the **distance between data points and centr**oid, we can use any method such as **Euclidean distance or Manhattan distance.**

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.

  Plots a curve between calculated WCSS values and the number of clusters K.

#finding optimal number of clusters using the elbow method

```
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
   kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
           kmeans.fit(x)
wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```

As we can see in the above code, we have used **the KMeans** class of sklearn. cluster library to form the clusters. Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10th value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

Step- 3: Training the K-means algorithm on the training dataset:

As we have got the number of clusters, so we can **now train the model on the dataset.** To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using i, we will use 5, as we know there are 5 clusters that need to be formed. The code is given below:

```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

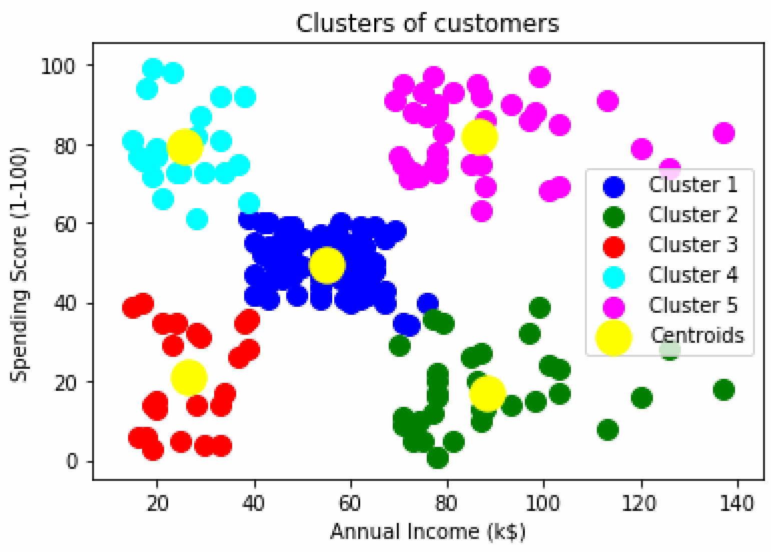
The first line is the same as above for creating the object of KMeans class.
In the second line of code, we have created the dependent variable **y_predict** to train the model.
By executing the above lines of code, we will get **the y_predict variable**. We can now compare the values of y_predict with our original dataset.

Step-4: Visualizing the Clusters:

The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one. To visualize the clusters will use scatter plot using mtp.scatter() function of matplotlib.

```
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the mtp.scatter, i.e., x[y_predict == 0, 0] containing the x value for the showing the matrix of features values, and the y_predict is ranging from 0 to 1.



The output image is clearly showing **the five different clusters** with **different colors**. The clusters are formed between **two parameters of the dataset**; **Annual income of customer** and **Spending**. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

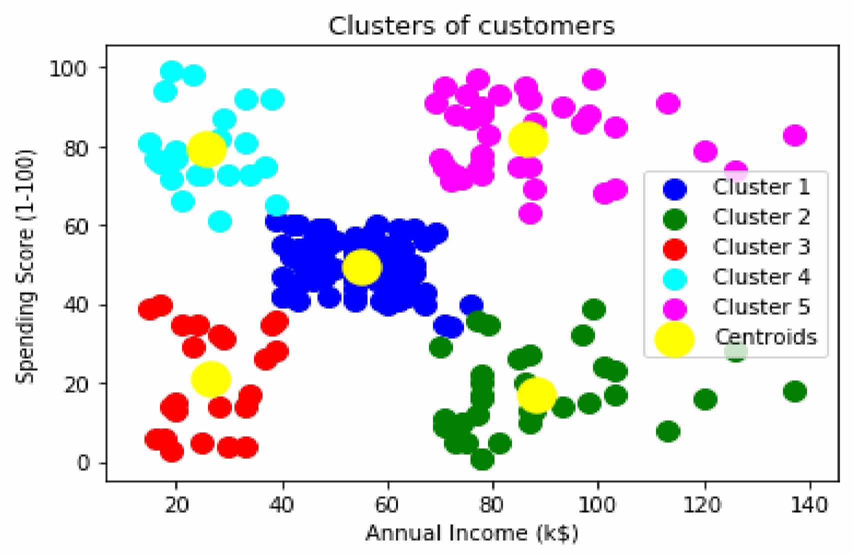
- **Cluster1 shows the customers with average salary** and **average spending** so we can categorize these customers as
- **Cluster2 shows the customer has a high income** but **low spending**, so we can categorize them as careful.
- **Cluster3** shows the **low income** and also **low spending** so they can be categorized as sensible.
- **Cluster4** shows the customers with **low income** with **very high spending** so they can be categorized as careless.
- **Cluster5** shows the customers with **high income and high spending** so they can be categorized as target, and these customers can be the most profitable customers for the mall owner.

PROGRAM:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('D:\AI TOOLS(PVP-19)\AI TOOLS LAB\WEEK-3( ML
        using USL)\Mall_Customers.csv')
print(dataset)
x = dataset.iloc[:, [3, 4]].values
print(x)
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS
#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
  kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
  kmeans.fit(x)
  wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
print(wcss_list)
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
print(kmeans)
print(y_predict)
```

```
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label =
'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label =
'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label =
'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label =
'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label =
'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c =
'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

<u>INPUT/OUTPUT:</u>



<u>CONCLUSION:</u> Program is executed successfully without any error.