Assignment DMW-2

Title: Clustering Techniques

Problem statement:

Consider a suitable dataset. For clustering of data instances in different graphs apply different clustering techniques. Visualize the clusters using suitable tools.

Objective:

- Understand the working of KMeans & DBSCAN clustering techniques.
- Implement clustering models using Python functions & libraries.

Outcomes:

Students will be able to:

- Understand the working ~~and~~ of KMeans and DBSCAN clustering tech.
- Implement clustering models using Python functions & libraries.

S/w & H/w Requirements:

Fedora 20 | windows 10, Jupyter Notebooks.

Theory:

- KMeans Clustering:

It is one of the most commonly used unsupervised Machine learning clustering techniques. It is a centroid based clustering techniques that needs you to decide the number of clusters (centroids) & randomly places the cluster centroids to begin the clustering process. The goal is to divide N observations into K clusters repeatedly until no more groups can be formed.

Advantages of k-means:

1. Easy to understand & implement.
2. Can handle large datasets well.

Disadvantages:

1. Sensitive to number of clusters / centroids chosen.
2. Does not work well with outliers.
3. Gets difficult in high dimensional spaces as distance between points increases & Euclidean distance diverges.
4. Gets slower as number of dimensions increases.

K means Algorithm:

1. Decide the number of clusters. This number is called k & number of clusters is equal to the number of centroids. Based on the value of k generate the coordinates for k random centroids.
2. For every point, calculate the Euclidean distance between the point & each of the centroids.
3. Assign the point to its nearest centroid. The points assigned to same centroid form a cluster.
4. Once clusters are formed, calculate new centroid for each cluster by taking the cluster mean. Cluster mean is the mean of the x & y coordinates of all pts belonging to the cluster.
5. Repeat step 2, 3 and 4 until centroids cannot move any further. Repeat these steps until convergence.

Elbow Method to find optimal number of clusters for k means:

1. For different values of k execute the following steps.
2. For each cluster calculate the sum squared distance of every pt.
3. Add the sum squared distances of each cluster to get total sum.
4. Keep adding the total sum for each k to a list.

5. Plot the sum of squared distances & k from the list.
6. Select the k at which a sharp change occurs (looks like an elbow).

- Density Based Spatial Clustering of Applications with Noise (DBSCAN)
  It is a density based clustering algorithm that forms clusters of dense regions of data points ignoring the low density areas.

Advantages of DBSCAN:
1. Works well for noisy datasets.
2. Can identify outliers easily.
3. clusters can take any irregular shape ~~unless~~ unlike kmeans where clusters are mostly spherical.

Disadvantages of DBSCAN:
1. Does not work very well for sparse datasets.
2. Sensitive to eps & minPts parameters.
3. Not partitionable for microprocessor system.

Important terms in DBSCAN:
1. Epsilon (eps): It is defined as the maximum distance between two points to be considered as neighbouring pts.
2. Minimum pts : This defines the ~~maximum~~ minimum no. of neighbouring pts that a given pt needs to be considered as a core data point. This includes the point itself.
   If the min pts meets the epsilon distance requirement then they are considered as a cluster.
3. Core point: A point is considered a core pt if it has minimum no. of pts at an epsilon distance from it, including the original point.

4. Border point: A data point that has less than minimum no. of data points needed but has at least one ~~core~~ pt in the neighbourhood.
5. Noise: A data point that is not a core point or a border point is considered ~~a~~ noise or an outlier.

DBSCAN Algorithm:

1. Decide the value of eps & minPts.
2. For each point:
   2.1 Calculate the distance from all pts. If the distance is less than or equal to eps then mark that pt as a neighbour of x.
   2.2 If the pt gets a neighbouring count greater than or equal to minPts, then mark it as core point or visited.
3. For each core point, if it is not already assigned to a cluster then create a new cluster. Recursively find all its neighbouring points & assign them the same cluster as the core point.
4. Continue these steps until all the unvisited pts are covered.

Conclusion:

We have successfully applied K means & DBSCAN clustering techniques & visualized the clusters.

In [6]:

```
import numpy as np

from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

In [7]:

```
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                            random_state=0)

X = StandardScaler().fit_transform(X)
```

In [45]:

```
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
```

In [46]:

```
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
```

In [47]:

```
print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Adjusted Rand Index: %0.3f"
      % metrics.adjusted_rand_score(labels_true, labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(labels_true, labels))
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X, labels))
```

```
Estimated number of clusters: 3
Estimated number of noise points: 18
Homogeneity: 0.953
Completeness: 0.883
V-measure: 0.917
Adjusted Rand Index: 0.952
Adjusted Mutual Information: 0.916
Silhouette Coefficient: 0.626
```

In [48]:

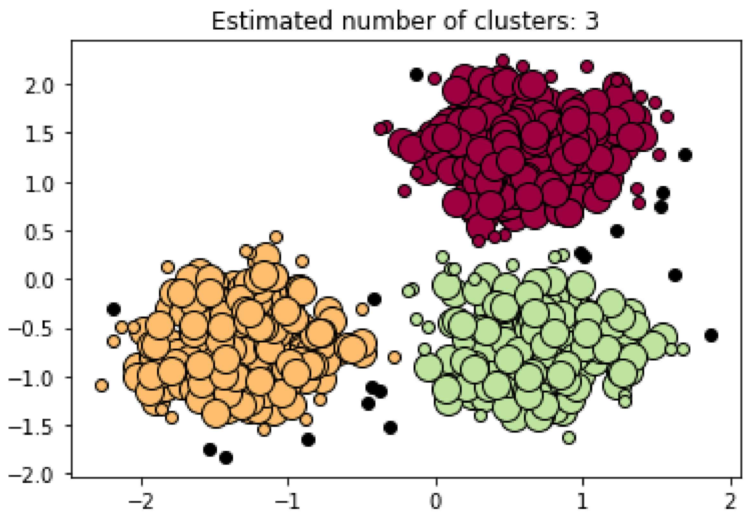
```
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```



In [ ]:

In [ ]:

In [16]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(context="notebook", palette="Spectral", style = 'darkgrid' ,font_scale = 1.5, colo
r_codes=True)
```

In [17]:

```
data=pd.read_csv('Mall_Customers.csv',index_col='CustomerID')
```

In [18]:

```
data.head
```

Out[18]:

```
<bound method NDFrame.head of              Genre  Age  Annual_Income_(k$)  Sp
ending_Score
CustomerID
1             Male   19                  15              39
2             Male   21                  15              81
3           Female   20                  16               6
4           Female   23                  16              77
5           Female   31                  17              40
...            ...  ...                 ...             ...
196         Female   35                 120              79
197         Female   45                 126              28
198           Male   32                 126              74
199           Male   32                 137              18
200           Male   30                 137              83

[200 rows x 4 columns]>
```

In [19]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200 entries, 1 to 200
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Genre               200 non-null    object
 1   Age                 200 non-null    int64
 2   Annual_Income_(k$)  200 non-null    int64
 3   Spending_Score      200 non-null    int64
dtypes: int64(3), object(1)
memory usage: 7.8+ KB
```

In [20]:

```
data.isnull().sum()
```

Out[20]:

```
Genre                  0
Age                    0
Annual_Income_(k$)     0
Spending_Score         0
dtype: int64
```

In [21]:

```
data.drop_duplicates(inplace=True)
```

In [22]:

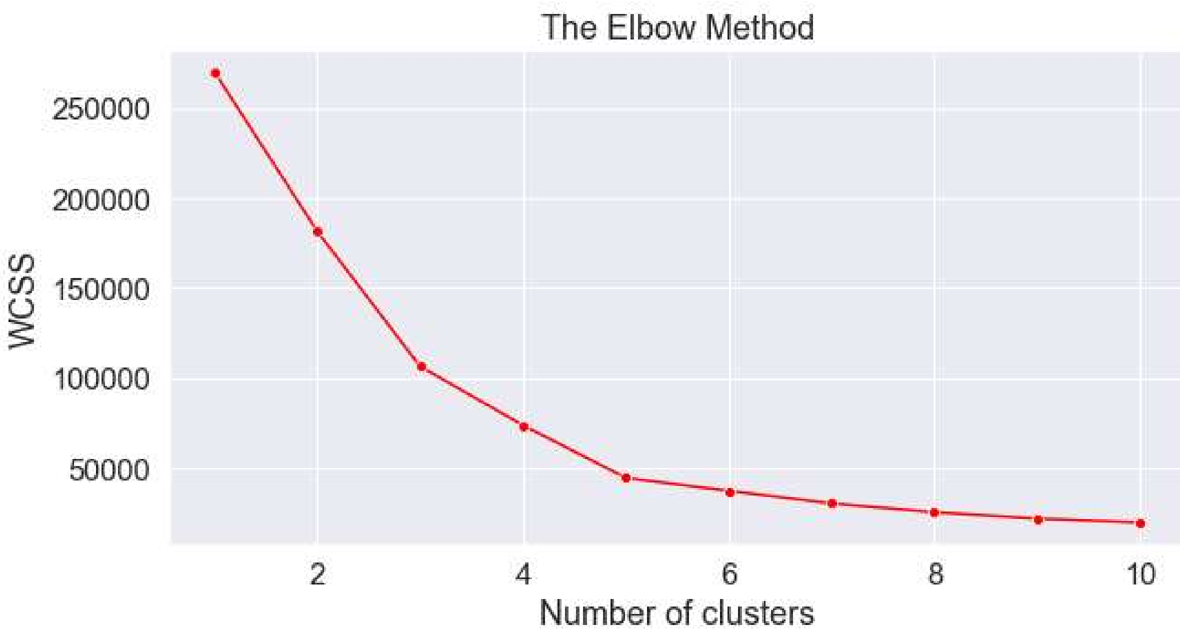
```
X = data.iloc[:, [2, 3]].values
```

In [23]:

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    # inertia method returns wcss for that model
    wcss.append(kmeans.inertia_)
```

In [24]:

```
plt.figure(figsize=(10,5))
sns.lineplot(range(1, 11), wcss,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

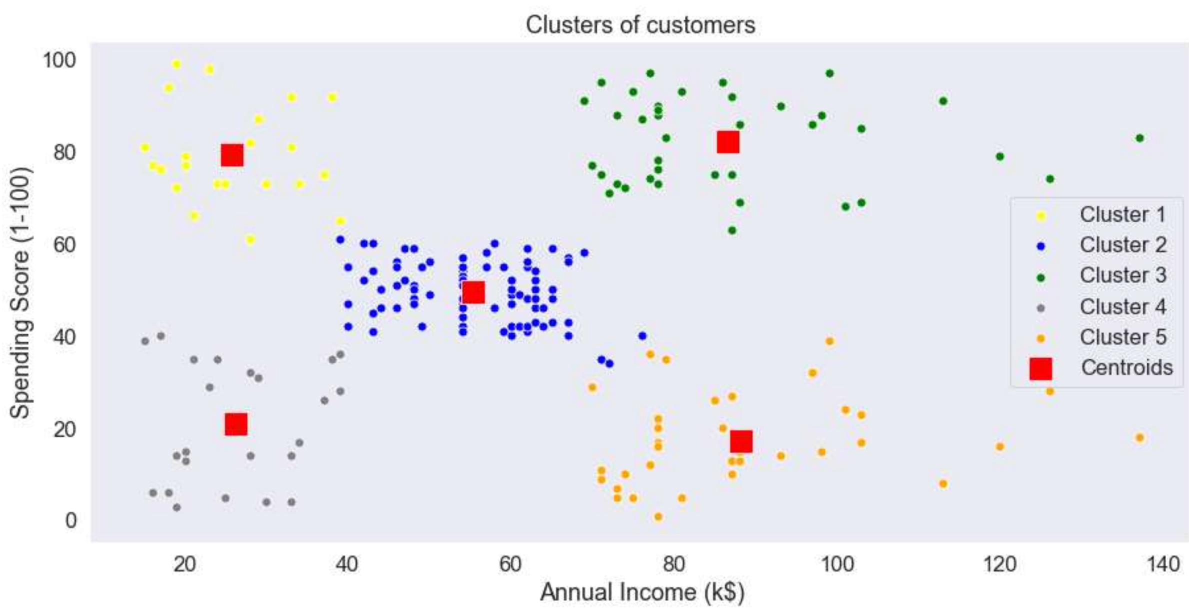


In [25]:

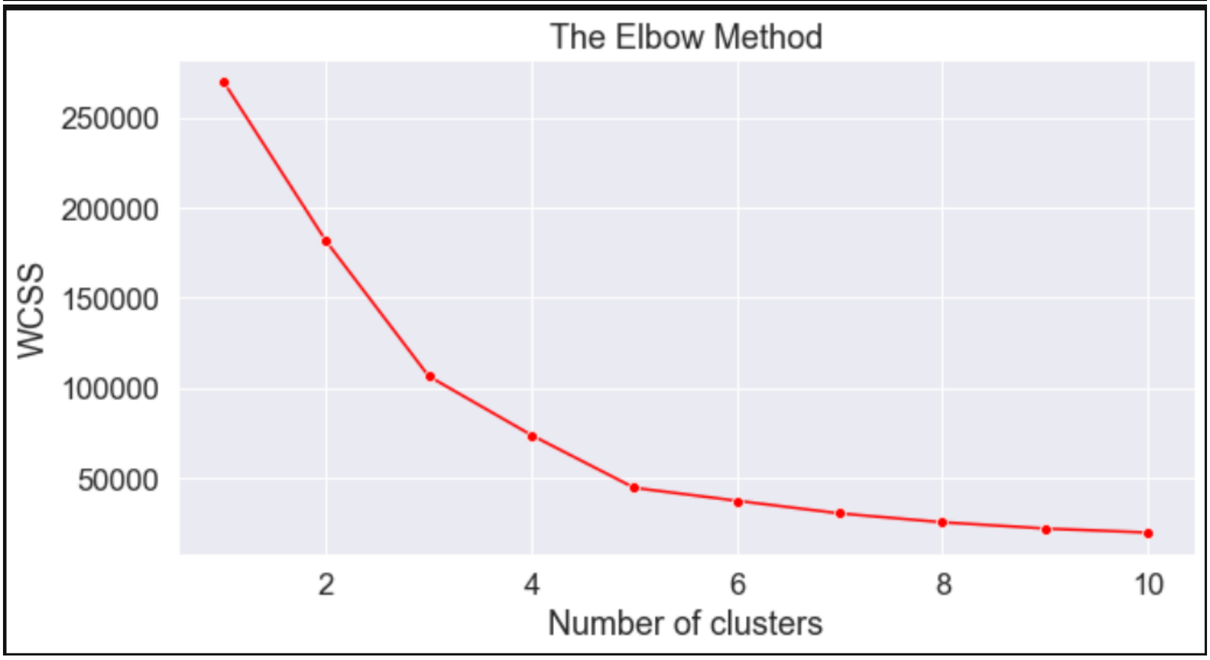
```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

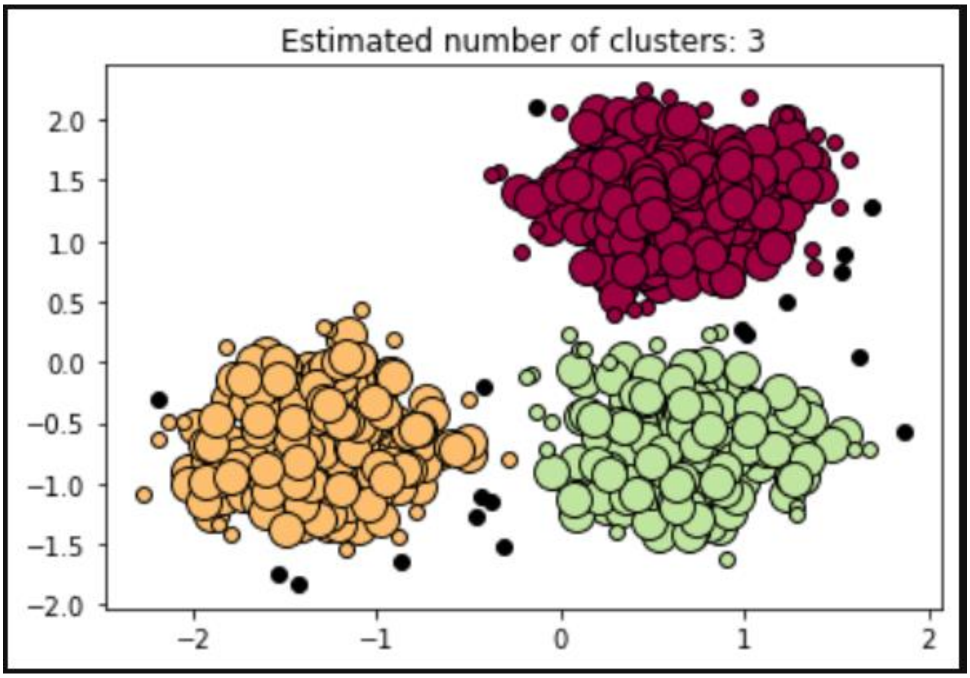
In [26]:

```
plt.figure(figsize=(15,7))
sns.scatterplot(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], color = 'yellow', label = 'Clust
er 1',s=50)
sns.scatterplot(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], color = 'blue', label = 'Cluster
2',s=50)
sns.scatterplot(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], color = 'green', label = 'Cluste
r 3',s=50)
sns.scatterplot(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], color = 'grey', label = 'Cluster
4',s=50)
sns.scatterplot(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], color = 'orange', label = 'Clust
er 5',s=50)
sns.scatterplot(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color = 're
d',
                label = 'Centroids',s=300,marker=',')
plt.grid(False)
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```




Clusters of customers
Spending Score (1-100)
Annual Income (k$)
100
80
60
40
20
0
20
40
60
80
100
120
140
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids


The Elbow Method
WCSS
250000
200000
150000
100000
50000
2
4
6
8
10
Number of clusters


Estimated number of clusters: 3
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2