Krish Sukhani

TE IT

Batch : D

2018140059

DWM - Exp 7B - Wine Quality Classification Daataset

```
#importing necessary libraries
import numpy as np
import pandas as pd
import warnings
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive
```

```
wine = pd.read_csv('/content/gdrive/My Drive/datasets/wine.csv',encoding= 'unicode_escape')
```

```
wine.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	qua
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	

```
wine.isnull().sum()
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

```
# plt.figure(figsize=(40,25))
```

```
# plt.subplots_adjust(left=0, bottom=0.5, right=0.9, top=0.9, wspace=0.5, hspace=0.8)
# plt.subplot(141)
# plt.title('Percentage of good and bad quality wine',fontsize = 20)
# wine['quality'].value_counts().plot.pie(autopct="%1.1f%%")


wine.drop(['quality'], axis=1, inplace=True)



wine.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8

```
X = wine.iloc[:,[8,10]]
# X=wine


X
```

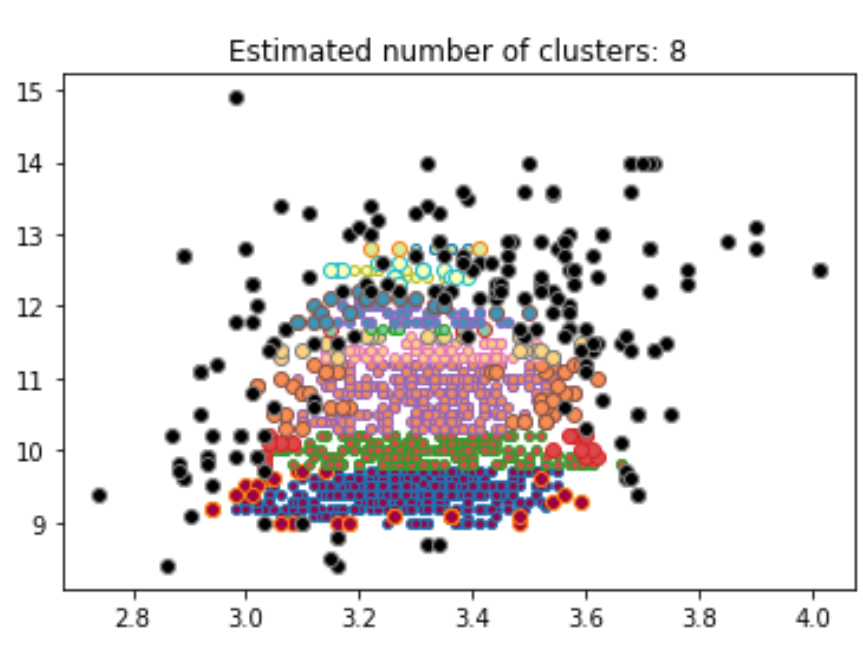
	pH	alcohol
0	3.51	9.4
1	3.20	9.8
2	3.26	9.8
3	3.16	9.8
4	3.51	9.4
...	...	...
1594	3.45	10.5
1595	3.52	11.2
1596	3.42	11.0
1597	3.57	10.2
1598	3.39	11.0

1599 rows × 2 columns

```
X = X.values


X
```

```
array([[ 3.51,  9.4 ],
       [ 3.2 ,  9.8 ],
       [ 3.26,  9.8 ],
       ...,
       [ 3.42, 11.  ],
       [ 3.57, 10.2 ],
       [ 3.39, 11.  ]])


from sklearn.cluster import DBSCAN
from sklearn import metrics
```

```
from sklearn.preprocessing import StandardScaler


db = DBSCAN(eps=0.1, 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_


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


print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X, labels))

     Estimated number of clusters: 8
     Estimated number of noise points: 179
     Silhouette Coefficient: 0.275


import matplotlib.pyplot as plt

# Black removed and is used for noise instead.
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),
             markersize=4)

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

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



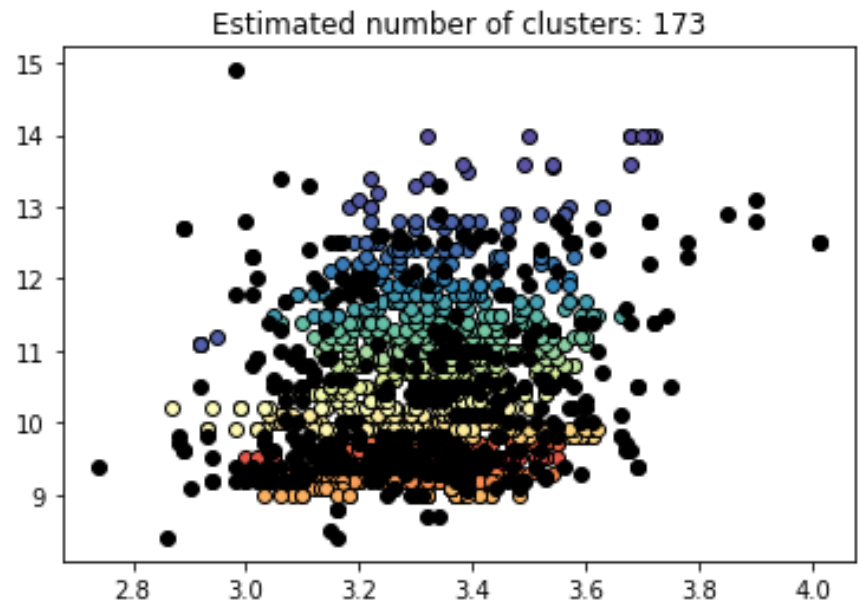
Conclusion : In this way, using DBScan the clustering was performed. It shows different results based on the value of eps i.e the distance between 2 samples. Also thee min_sample affects the clusters

```
from sklearn.cluster import OPTICS
from sklearn.preprocessing import StandardScaler
```

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


db = OPTICS(min_samples=4).fit(X_Optics)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
# core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X_Optics, labels))
```

```
Estimated number of clusters: 173
Estimated number of noise points: 494
Silhouette Coefficient: 0.214
/usr/local/lib/python3.7/dist-packages/sklearn/cluster/_optics.py:802: RuntimeWarning: divide by zero encou
  ratio = reachability_plot[:-1] / reachability_plot[1:]
```

```
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_Optics[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=4)

    xy = X_Optics[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()
```



Conclusion : In this way, using Optics the clustering was performed. It shows different results based on the value of eps i.e the distance between 2 samples. Also thee min_sample affects the clusters