Question 1: How many groups does this dataset have?

Answer :

2 groups. We simply run the code. Then we get a beautiful plot with one big circle contians one small circle.

If you look at the API of sklearn.datasets.make_circles, it said “Make a large circle containing a smaller circle in 2d.”

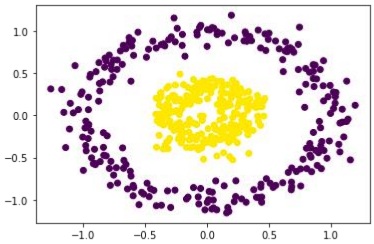

figure 1: sklearn.datasets.make_circles

Question 2: Perform a clustering of this dataset using k-means. What can we expect? What do you notice?

Answer:

When we use k-means in this make_circles dataset, we notice that the KMeans classify the dataset based on 2 different centel, instead of classify this dataset as 2 circles, the Kmeans divides the image into upper and lower part.

In the code, we generate the dataset first same as the question 1. Then we import KMeans and use KMeans to fit the data. One thing should notice is that we used both init=’k-means++’ and init=’random’, the results are the same.

```
# Question 2
import matplotlib.pyplot as plt
import numpy as np
import sklearn.datasets
from sklearn.utils import shuffle
# generate the dataset
data, labels = sklearn.datasets.make_circles(n_samples=500, noise=0.1, factor=0.3, random_state=0)
# Random permutation of the rows of the matrix (the observations are mixed)
data, labels = shuffle(data, labels)

# using kMeans to classify the data
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2, random_state=0,init='k-means++').fit(data)
labels = kmeans.labels_
# plot the image
plt.scatter(data[:,0], data[:,1], c=labels)
plt.show()
```

figure 2.1: code of KMeans in make_circles dataset

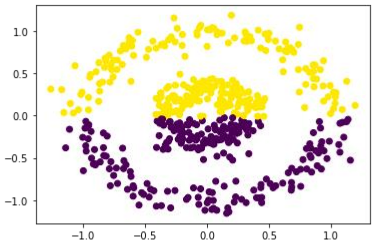

figure 2.2: KMeans method classifies the make_circles dataset

Question 3: What are the default values for important DBSCAN parameters in scikit-learn(ε and m)?

Answer:

According to the sklearn official documentation.

The default values for ***eps*** is ***0.5***.

The default values for ***min_samples*** is ***5***.

`sklearn.cluster`.DBSCAN

class `sklearn.cluster.`**`DBSCAN`**(*eps=0.5*, ***, *min_samples=5*, *metric='euclidean'*, *metric_params=None*, *algorithm='auto'*, *leaf_size=30*, *p=None*, *n_jobs=None*) [source]

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

Parameters:

eps : *float, default=0.5*

The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

min_samples : *int, default=5*

The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.

metric : *str, or callable, default='euclidean'*

The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by `sklearn.metrics.pairwise_distances` for its metric parameter. If metric is "precomputed", X is assumed to be a distance matrix and must be square. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors for DBSCAN.

figure 3: important DBSCAN parameters

Question 4: What do you notice? On what parameter is it probably necessary to play to improve this result?

Answer:

While applay an automatic classification by DBSCAN on make_circles dataset, we notice that there is only one cluster. In order to improve the result, we need to tune the eps parameter to a suitable distance. We tried min_samples also, but it doesn't change the rusult. So if we only change one parameter, it should be eps.

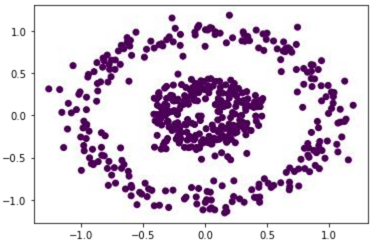

figure4.1: default DBSCAN

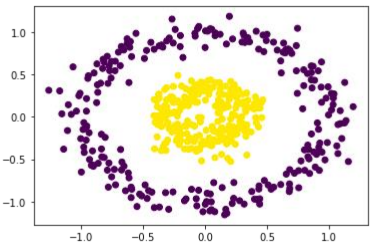

figure4.2: DBSCAN with eps=0.21

Question 5: Using the NearestNeighbours documentation in scikit-learn, explain what the code above does.

```
from sklearn.neighbors import NearestNeighbors

nn = NearestNeighbors(n_neighbors=4).fit(data)
distances, _ = nn.kneighbors(data)
```

figure 5: NearestNeighbours

Answer:

The code above training the NearestNeighbors estimator with the 'data' and find the nearest 4 neighbours of each data point in the 'data'.

The first line is import the NearestNeighbors from sklearn.

The second line is fitting the nearest neighbors estimator from the training dataset using 4 neighbors for kneighbor queries and assign it to nn.

The third line is find 4-nearest-neighbors of data points in the 'data', then sign the length array to distances and indice array to _.

The 'distances' is an array contains 500 queries. Each query has the lengths of 4 nearest neighbors. The first one of each row is the point itself.

```
print(distances.shape)
print(distances)
✓ 0.8s

(500, 4)
[[0.         0.04809514 0.0653732  0.0861943 ]
 [0.         0.0175769  0.03390971 0.03782292]
 [0.         0.03366237 0.07399293 0.09463227]
 ...
 [0.         0.07778193 0.0838797  0.10100307]
 [0.         0.05370197 0.08659094 0.08833146]
 [0.         0.00965661 0.0413765  0.04736844]]
```

figure 5.1: distance

The '_' is an array also contians 500 queries. Each query has the indices of 4 nearest neighbors in the population matrix. The first one of each row is the point itself,

```
print(_.shape)
print(_)
✓ 0.1s

(500, 4)
[[  0 211 328 335]
 [  1 347 224 134]
 [  2 338  96 361]
 ...
 [497 443  75 149]
 [498 187 340 221]
 [499 431  86 158]]
```

figure 5.2: _

Question 6: From the 4-distance graph, determine the appropriate eps value for this dataset using the current view heuristic. Reapply DBSCAN with these settings. Display the resulting point cloud.

Answer:

From the 4-distance graph we can see that the curve is very sharp from 0.25-0.15 base on the y-axis. And most of the 4-distance points are under 0.15. So we can determin our eps=0.15 and min_samples=4.

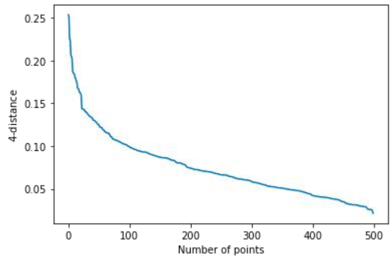

figure 6.1: 4-distance graph

The code and plot below you can see that, we divide the data into 2 groups and some purple dots seems like outliers ---- They are the dots belong to 4-distance dot which lengths large than 0.15 and smaller than 0.25.

```
# in the above cruve, we can see that most of the 4th points
# are under 0.15 and the maximun is 0.25
from sklearn.cluster import DBSCAN
# set the min_samples = 4 and eps=0.15
db = DBSCAN(eps=0.15, min_samples=4)
# db = DBSCAN(eps=0.25, min_samples=4)
# fit and predict
predictions = db.fit_predict(data)
# Display of the scatter plot colored by the predictions
plt.scatter(data[:,0], data[:,1], c=predictions)
plt.show()
```

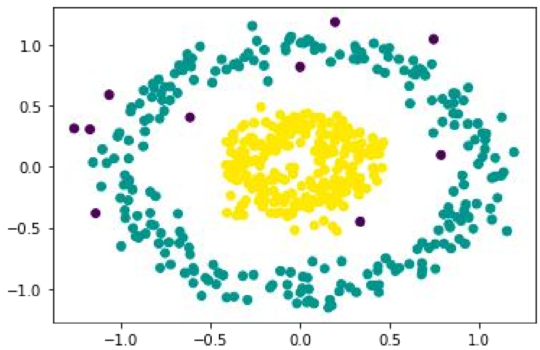

figure 6.2: DBSCAN with eps=0.15 and min_sample=4

Question 7: How many groups do you get? What are observations with label -1?

Answer:

Just like question 6. We get 3 groupss. 2 main group and 1 group contains outlier.

The label -1 is the outlier. To justify our answer, we code. As you can see below, the label =-1 is the 4-distance > eps=0.15. Which are the small amout, hence we can consider them as outlier.

```
distances[:,-1][predictions == -1]
✓ 0.1s

array([0.17617733, 0.25349463, 0.20459095, 0.25047591, 0.18552116,
       0.15885283, 0.2251783 , 0.18382363, 0.16621258, 0.17979533])
```

figure 7: label = -1

Question 8: How many observations does this dataset contain?

Answer:

The original Iris dataset import from sklearn contians 150 samples. And the noise we generate contains 20 samples. Totally we have 170 observations.

Question 9: Perform a principal component analysis and visualize the Iris dataset projected along its first two principal axes.

Answer:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
plt.scatter(X_pca[:,0], X_pca[:,1], c=y)
plt.show()
```

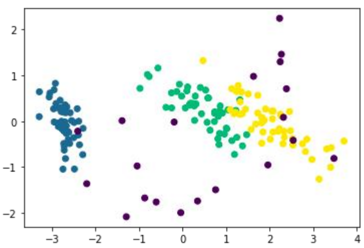

figure 9: visualize PCA in Iris dataset with noise

Question 10: Apply automatic classification using DBSCAN (work on 4-dimensional data, not projected data!). Visualize the groups obtained in the main plan. Compare this result to the partitioning obtained using a k-means.

Answer:

First we use the automatic classificaiton of DBSCAN. We can see in the below, while we using automatic classification of DBSCAN, DBSCAN classify 3 clusters. The outliers are mix with the three types of iris. We need to plot the image in order have a better look.

```
# import the data
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
min_, max_ = X.min(axis=0), X.max(axis=0)
noise = np.random.rand(20, 4) * (max_ - min_) + min_
X = np.concatenate((X, noise))
y = np.concatenate((y, -1 * np.ones(20, dtype=int)))
# using automatic DBSCAN
from sklearn.cluster import DBSCAN
db = DBSCAN()
predictions = db.fit_predict(X)
predictions
✓ 0.8s

array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,
       -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1,  1,
        1,  1,  1, -1, -1,  1, -1, -1,  1,  1,  1,  1,  1,  1,  1, -1, -1,
        1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1, -1, -1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1])
```

figure 10.1: Automatic DBSCAN

we use Kmeans on this dataset agian with parameter n_clusters = 4. Below is the result.

```
# import the data
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
min_, max_ = X.min(axis=0), X.max(axis=0)
noise = np.random.rand(20, 4) * (max_ - min_) + min_
X = np.concatenate((X, noise))
y = np.concatenate((y, 4 * np.ones(20, dtype=int)))
# uisng KMeans
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4).fit(X)
labels = kmeans.labels_
labels
✓ 0.3s

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 2, 2, 2, 3, 2, 3, 3, 3, 3, 2, 3, 2,
       2, 3, 2, 3, 2, 3, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 2, 2, 2, 2, 2,
       3, 3, 3, 2, 3, 3, 3, 3, 3, 2, 3, 3, 0, 2, 0, 0, 0, 0, 3, 0, 0, 0,
       2, 2, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 2, 0, 2, 2, 1, 3, 0,
       3, 2, 1, 3, 2, 0, 3, 3, 1, 3, 1, 3, 2, 1, 2, 2], dtype=int32)
```

figure 10.2: KMeans clusters iris dataset with noise

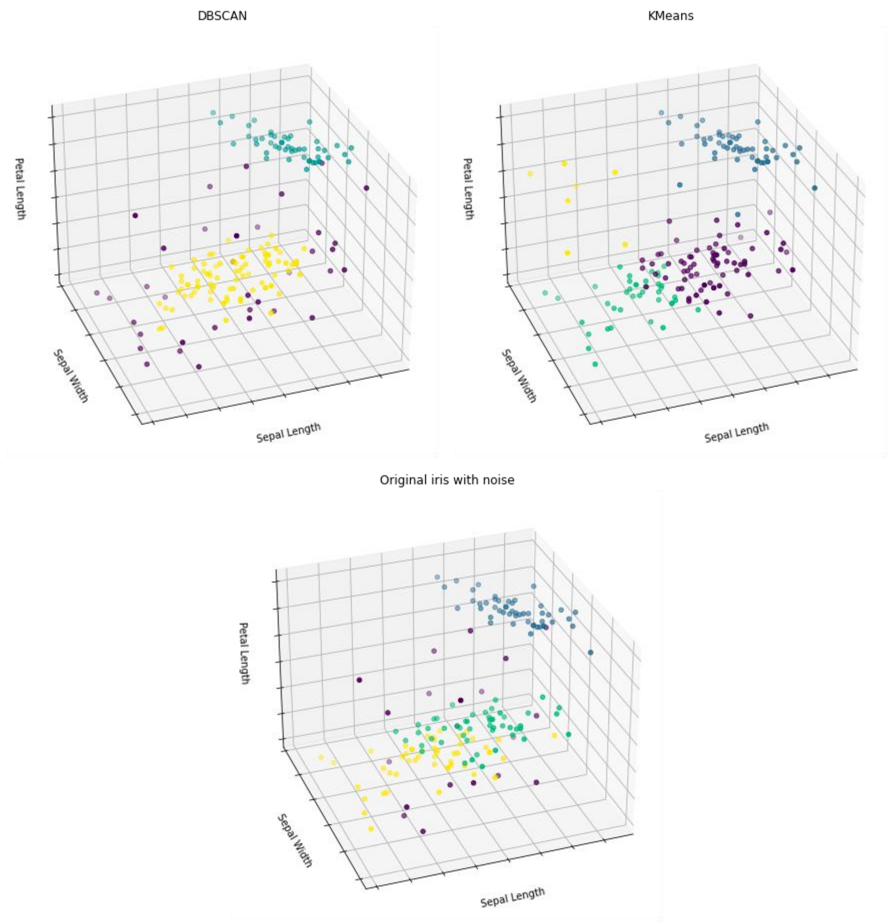

figure 10.3: 3D groups obtain Visualization

As you can see in the plots, The DBSCAN class one iris with large sepal length and petal length very well, combine the two types of iris with low petal length together cause they are very near. And some noises and iris far from these two sets. The DBSCAN is very good at detecting the outliers. The KMeans can calssify three types of iris better than DBSCAN, however, the KMeans can't recognize noise very well.

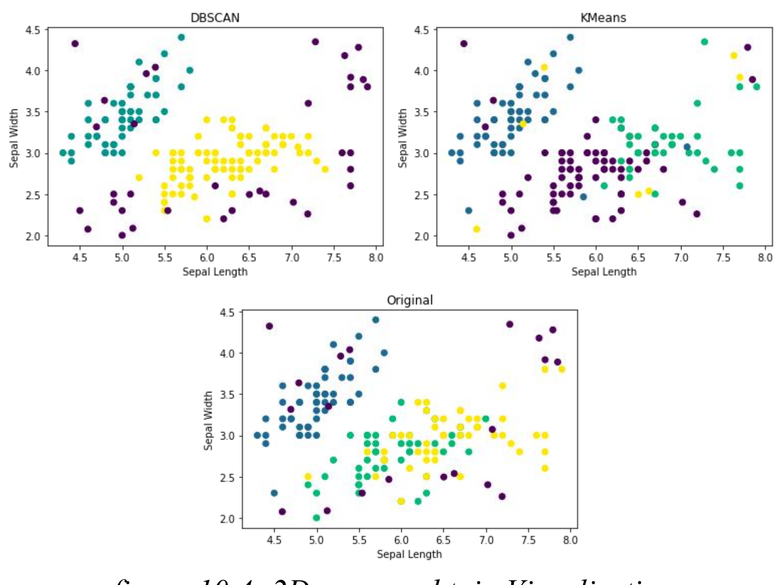

figure 10.4: 2D groups obtain Visualization

Question 11: Using the functions of sklearn.metrics, calculate the good detection rate of outliers. Up to what proportion of noisy data is the partitioning obtained by DBSCAN robust?

Answer:

First, we need to define what is good detection rate of outliers.

I believe the good detection rate is combine with two part:

1. What percentage of outliers can estimator detect?
2. What percentage of non-noise data does the estimator detect as outliers?

For the first part: the fomular should be a = (number of correct outliers estimator detect)/(total number of outliers)

For the second part, the fomular should be b = (number of incorrect ouliers estimator detect)/(total number of non-outliers)

The higher the a, higher the good detection rate of outliers. Higher the b, the lower the good dection rate of outliers.

we use accuracy_score in sklearn.metrics to compute the a.

```
from sklearn.metrics import accuracy_score
# calculate the a
a = accuracy_score(predictions[150:170],y[150:170])
print("a = " + str(a))
b = (predictions[0:150] == -1).sum() / 150
print("b = ", str(b))
```

✓ 0.5s

```
a = 0.85
b =  0.11333333333333333
```

figure 11: Good detection rate of outliers

As a conclusion, DBSCAN can detect 85% outliers in the iris dataset with noise. Also, DBSCAN misjudge around 11% of the real data as outliers.