HW3_SP21

April 19, 2021

1 HW3

1.1 GENERAL INSTRUCTIONS:

- **CLEARLY** mark where you are answering each question (all written questions must be answered in Markdown cells, NOT as comments in code cells)
- Show all code necessary for the analysis, but remove superfluous code

---

- a) (15 points) Using the simulated data found here, z score your variables, and use plotnine/ggplot to make a scatterplot of the data in a clear, and effective way.
- b) (30 points) In a MARKDOWN cell, discuss what patterns you see in the data, and **thouroughly discuss the pros/cons** of using *k-means*, *Gaussian Mixture Models* (with EM), *DBSCAN*, and *Hierarchical Clustering* with patterns like the ones you see.

(Please use "**" to make any mention of one of the algorithms **bold** in your discussion. For example "I think **DBSCAN** is the best algorithm ever!" will make the word "DBSCAN" bold in a Markdown cell).

- c) (17 points) choose which algorithm you think will perform the *best* on this data, and the algorithm you think will perform the *worst* on the data. **Briefly explain why you think each algo you chose is the best/worst** (there may be overlap from part b, but please answer this separately, despite repetition).
- d) (18 points) using the TWO algorithms you chose in part c, cluster the data and calculate the silhouette scores. If you had to choose any hyperparameter values (including but not limited to: k, distance metric, linkage critera, eps, min_samples…etc), **state how you chose them**. **Which algorithm did better according to the silhouette scores and plots from part e?**
- e) (20 points) using plotnine/ggplot, make one scatterplot for EACH of the two clustering algorithms that colors points by their cluster. **Are the clusters similar between the two algorithms? If yes, describe the shape/location of the clusters** (e.g. "cluster X is a small, dense, circlular cluster with high values for x and low values for y…"). **If no, describe the differnces in the shape/# of clusters.**

[1]:
```
# import necessary packages

import warnings
warnings.filterwarnings('ignore')
```

```
import pandas as pd
import numpy as np
from plotnine import *

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors

from sklearn.cluster import DBSCAN


from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette_score

from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch

%matplotlib inline
```

2 a)

[2]:
```
data = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/
↪CPSC392ParlettPelleriti/master/Data/HW3.csv")
data.head()
```

[2]:

	x	y
0	-0.386297	2.019190
1	0.122299	1.884267
2	0.109275	1.570052
3	0.123379	1.820644
4	0.360915	2.168495

[3]:
```
data.describe()
```

[3]:

	x	y
count	1175.000000	1175.000000
mean	0.745210	3.267538
std	1.501720	1.150104
min	-1.986871	0.502147
25%	-0.751886	2.822068
50%	1.274537	3.576190
75%	2.019319	4.039132
max	2.981937	5.434746

[4]:
```
data.isnull().sum(axis=0) # checked to make sure there is no missing data
```

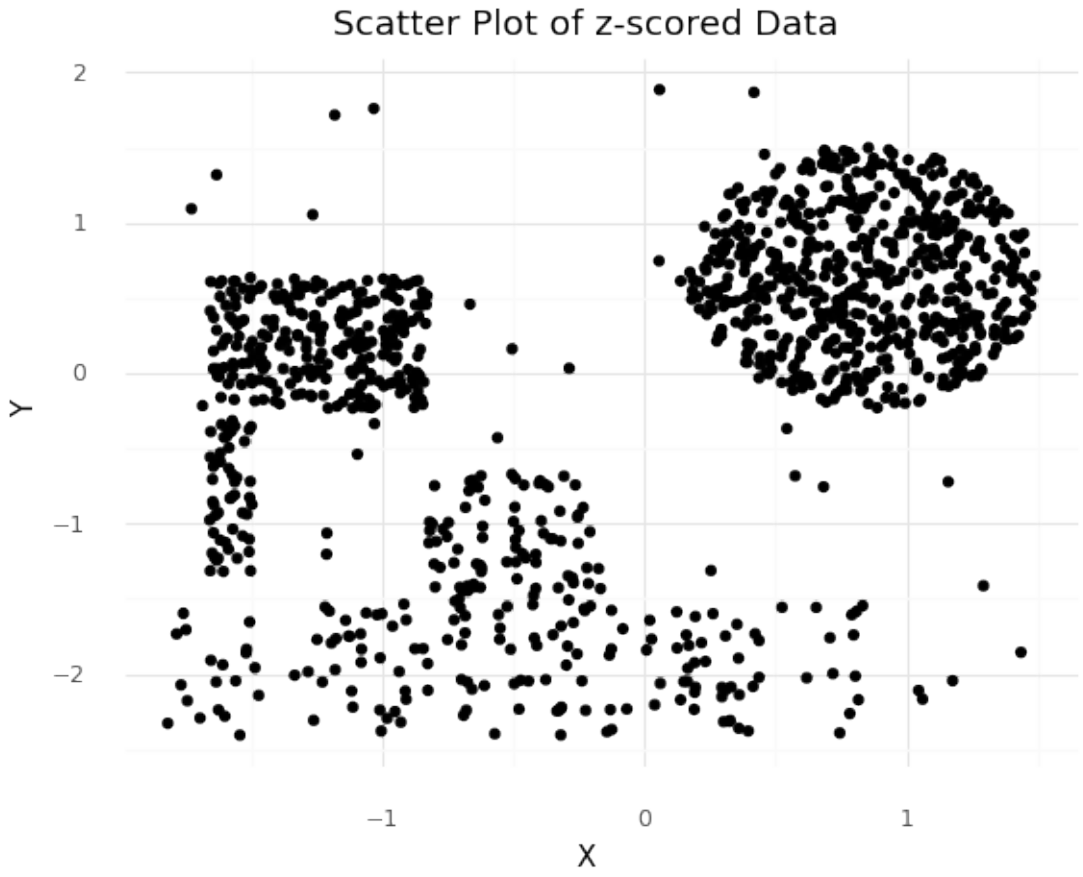
[4]:
```
x    0
y    0
dtype: int64
```

[5]:
```
features = ['x', 'y']
z = StandardScaler()
# z-socring the data
data[features] = z.fit_transform(data)
data.head()
```

[5]:

	x	y
0	-0.753795	-1.085884
1	-0.414975	-1.203248
2	-0.423652	-1.476569
3	-0.414256	-1.258590
4	-0.256012	-0.956009

[6]:
```
(ggplot(data, aes(x = "x", y = "y")) + geom_point() + theme_minimal() +␣
↪ggtitle("Scatter Plot of z-scored Data") + labs(x = "X", y = "Y"))
```



[6]: <ggplot: (324085595)>

3 b)

3.1 Question: Discuss what patterns you see in the data, and thouroughly discuss the pros/cons of using k-means, Gaussian Mixture Models (with EM), DBSCAN, and Hierarchical Clustering with patterns like the ones you see.

3.2 Answer:

- The distribution of the data points in the scatter plot above titled "ScatterPlot of X and Y" suggests that there are 2 or 3 cluster groups and several outliers. The first cluster group (most left group) appears to be approximately in the region of x-range: [-1.80, 0.20] and y-range: [-1.30, 1.60]. The second cluster group (middle group) appears to be approximately in the region of x-range: [-1.90, 0.80] and y-range: [-2.80, -0.65], however, this cluster is the least dense and may possibly just be noise. The third cluster group (most right group) appears to be approximately in the region of x-range: [0.20, 2.00] and the region of y-range: [-0.33, 1.60]. The points that are not in the x & y ranges of the clusters just stated are most likely outliers. There does not appear to be a hierarchical pattern observed in the graph because it looks like the clusters do not contain sub-clusters.
- K-Means
  - Pros: scales well with large data sets, flexible and adaptable, easy to implement and interpret results, not many parameters to optimize (only need to supply number of clusters)
  - Cons: generalizes clusters to be of a sphereical shape - bad for clusters that are non-convex shapes, variances are assumed to be the same across different clusters, outliers are not identified (hard assignment - either in a group or not and so noises are included in the cluster(s)), the number of clusters must be decided before analysis.
  - K-Means for this graph: K-Means would be a good choice for this dataset, but not the best because 2 of the 3 clusters appear to be of non-convex shapes. There would be overlap among the clusters if a convex shape were to be outlined around each of the clusters, which further argues against using K-Means. The graph appears to have several outliers, which is a big reason why K-Means is not ideal for this dataset. The K-Means alogirthm does not classify outliers and so the outliers will be grouped into the cluster(s). K-Means assumes that the clusters have the same variance, but this assumption is not good for our dataset because the middle cluster seems to have less cohesion amongst its data points compared to the left and right clusters. One could argue that the K-Means algorithm would be sufficient to use for this dataset because K-Means is relatively simple to implement/analyze and at least 1 of the 3 clusters (possibly 2 out of 3 because one could argue the middle cluster is convex shaped) of the dataset appears to be of convex shape. Although those are valid points, K-Means is not the most ideal for this dataset because the majority of the clusters seem to be non-convex shapes, the clusters do not seem to have the same variance, and there are a significant number of outliers (no concept of noise to account for outliers).

- Gaussian Mixture Models (GMM) with Expectation Maximization (EM)
  - Pros: Variance amongst the different clusters are not assumed to be the same, robust to outliers - (soft assignment - looks at the probability that a point is in a cluster), does not generalize clusters to be of only sphereical shapes - instead allows for different kinds of eliptical shapes, shines in situations when there is a lot of overlap of the clusters
  - Cons: Shapes of the clusters are assumed to be eliptical-like (less assumption of the shape than K-Means but still has a shape assumption), no concept of noise to properly classify outliers, algorithm is sometimes considered complex and/or slow - definitely more complex than K-Means
  - Gaussian-Mixture with EM for this graph: GMM w/ EM would be a good choice for this dataset, but not the best choice. Unlike K-Means, GM w/ EM does not assume the shapes of clusters to be of sphereical shapes only, which is good for the dataset provided because 2 of the 3 clusters shown appear to be non-convex shapes. GM w/ EM does not assume the clusters have equal variance which is good for this dataset since the clusters above seem to have different variances compared to each other. For example, the middle cluster seems to consist of data points that are more spread out compared to the right cluster which seems to be very dense. GM w/ EM is robust with outliers because it has soft assignment, meaning it does not assume a data point to be either in a cluster or not but instead calculates the probability of each data point being a cluster or not. A major downside of the GM w/ EM algorithm is that the algorithm does not have a concept of noise and so outliers are not properly classified as noise. Sometimes GA w/ EM is considered complex and/or slow to work with, however, it shines in situations when there is plenty of overlap among the clusters. But there is almost no overlap of clusters in the graph above and so GM w/ EM seems like not an ideal choice. K-Means is still a better choice for the data provided because GM w/EM is more complex and is often intended to use when there is a lot of overlap between the clusters.
- DBSCAN (Distributed Based Spatial Clustering of Applications with Noise)
  - Pros: The shape of the cluster is not assumed to be of a pre-defined shape, has noise concept - classifies outliers as noise, do not provide the number of clusters for the model (it figures it out on its own), few hyperparameters to optimize to make the model efficient (however it is sensitive to these parameters)
  - Cons: Less effective in high dimensional data (a lot of features), not great when there are clusters that are overlapping and/or touching, not optimal when the clusters consist of different densities, sensitive to hyperparameters, computationally expensive (but not as much as HCA)
  - DBSCAN for this graph: DBSCAN would be a great choice for the dataset provided for several reasons. First, DBSCAN does not assume the clusters to be of a pre-defined shape, which is good for this dataset because 2 of the 3 clusters do not appear to be of sphereical, convex, or eliptical shapes. Second, DBSCAN takes into consideration the idea of noise for classifying outliers. The graph of the data above contains a significant number of points that seem like outliers (do not belong to any single cluster). This is the most important reason why DBSCAN is ideal for the data provided. DBSCAN does have several disadvantages, but most of them do not apply to the dataset provided. For example, DBSCAN does not do well when the clusters are overlapping and/or touching. In the graph above, the clusters appear not to be touching and/or overlapping. DBSCAN also does not do well when there are a lot of features for the algorithm to consider, but our graph consists of only 2 features (X and Y). The main disadvantage to consider for using DBSCAN with the provided dataset is that the hyperparameters (eps and min

number of samples) should be optimized in order for DBSCAN to be most effective. Although this is a disadvantage, it is a problem that a data scientist can overcome with tuning of the parameters. Overall, DBSCAN is a great choice and the best algorithm to use for the data provided because of the reasons just stated.

- Hierarchical Clustering (HCA - Hierarchical Agglomerative Clustering)
  - Pros: Hierarchical relationship is established between clusters - good for hierarchical data, flexibility with clusters via dendrogram, flexibility with choosing linkage criteria and distance metric, number of clusters is arguably the only parameter needed for this model (one may or may not consider linkage and distance metrics as hyperparameters)
  - Cons: Computationally expensive (Big O of O(n^3)), cannot undo a merging of clusters, no concept of noise (does not classify outliers as noise like DBSCAN would)
  - Hierarchical Clustering for this graph: Hierarchical clustering would not be a good choice to use for the dataset provided. Usually HCA is used for data that is hierarchical by nature. For example, HCA would be good for a situation when an animal dataset is provided and a data scienctist wants to classify the animal data into their different kingdoms and groups. HCA would be good for that example because it would create a hierarchical relationship between the clusters and that visualization (dendrogram) can give readers better insight on the clusters of the data. In the dataset provided here, HCA would not be appropriate because a hierarchical pattern is not observed in the first graph. If clusters seem to exist within clusters, then that would suggest there is a hierarchical pattern/relationship in the data. But that is not the case with the data provided and so HCA would be unnecessary for this dataset especially considering how it is very computationally expensive. Also HCA would not be good for this dataset because it does not handle outliers like DBSCAN, meaning it does not have a concept of grouping the outliers into a noise group. Overall, HCA would not be a good choice to use with the data provided because it does not handle outliers, is very computationally expensive, and a there is no hierarchical pattern/relationship in the data being worked with.

4 c)

4.1 Question: Choose which algorithm you think will perform the best on this data, and the algorithm you think will perform the worst on the data. Briefly explain why you think each algo you chose is the best/worst (there may be overlap from part b, but please answer this separately, despite repetition).

4.1.1 Answer:

Best Performing Algorithm: DBSCAN

- DBSCAN would be the best algorithm out of the choices to use with the data of the dataset provided. DBSCAN is the best because it has a concept of noise (for outliers), does not limit a cluster to be of a pre-defined convex shape, the clusters do not seem to be overlapping in the graph of the data provided, there does not appear to be a hierarchical relationship in the graph of the data provided, and it is relatively simple to implement.

Good Choices: K-Means and GM w/ EM

- K-Means would be the next best algorithm to use out of the choices because it is simple to implement, not computationally expensive, and has very little parameters to tune. GM w/ EM is the next best choice after K-Means. It does not assume as much about the cluster shapes as K-Means however it is ususally intended to be used when working with data that has a lot of overlap. Like K-Means, GM w/ EM does not have a concept of noise which is not ideal because there appears to be a significant number of outliers in the graph above. K-Means would still be preferred however over GM w/ EM becasue K-Means is more simple to understand and work with.

Worst Choice: HCA

- HCA would be the worst choice algorithm to use out of the choices because it does not have a concept of noise, is very computationally expensive, and there does not appear to be a hierarchical relationship in the data provided. If the data provided were hierarchical by nature then it would be a good idea to use HCA to observe the hierchical relationship. Regardless of hierarchical patterns, HCA does not take into consideration the concept of noise for outliers which is very important for the data provided given their prevalence in the graph above.

5 d & e:

6 d)

6.1 Question: Using the TWO algorithms you chose in part c, cluster the data and calculate the silhouette scores. If you had to choose any hyperparameter values (including but not limited to: k, distance metric, linkage critera, eps, min_samples...etc), state how you chose them. Which algorithm did better according to the silhouette scores and plots from part e?

7 e)

7.1 Question: Using plotnine/ggplot, make one scatterplot for EACH of the two clustering algorithms that colors points by their cluster. Are the clusters similar between the two algorithms? If yes, describe the shape/location of the clusters (e.g. "cluster X is a small, dense, circlular cluster with high values for x and low values for y..."). If no, describe the differnces in the shape/# of clusters.

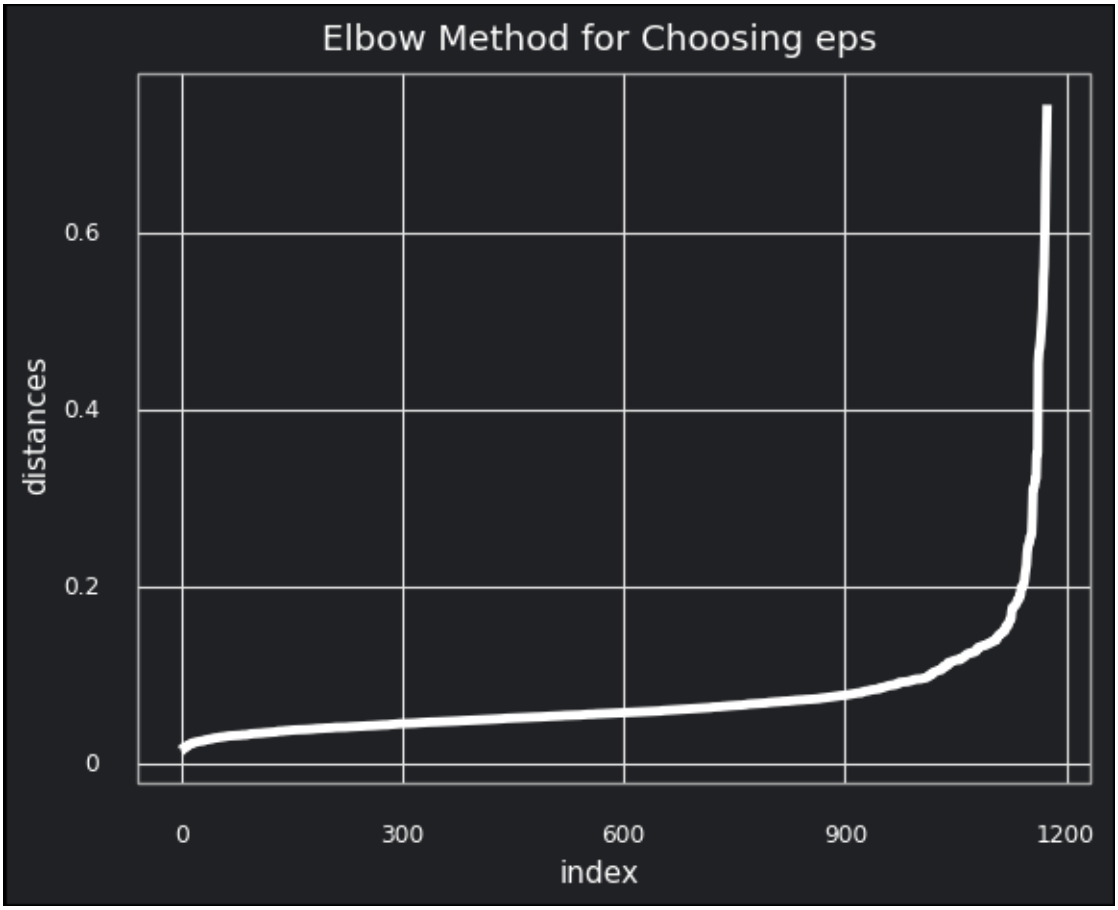
[45]:
```
# we ask for 4 nearest, because the data point it self (distance = 0) is␣
↪included
mins = 3
nn = NearestNeighbors(mins + 1)

nn.fit(data[features])

distances, neighbors = nn.kneighbors(data[features])
```

```
distances
```

[45]:
```
array([[0.        , 0.05255983, 0.05283091, 0.0775373 ],
       [0.        , 0.04519017, 0.05171335, 0.05534692],
       [0.        , 0.04721203, 0.06201024, 0.10691498],
       …,
       [0.        , 0.14000297, 0.29884062, 0.32151565],
       [0.        , 0.08908086, 0.0933355 , 0.11587237],
       [0.        , 0.06082158, 0.16839484, 0.24531375]])
```

[46]:
```
# sort the distances
distances = np.sort(distances[:, mins], axis = 0)
```

[74]:
```
#plot the distances
distances_df = pd.DataFrame({"distances": distances,
                             "index": list(range(0,len(distances)))})
plt = (ggplot(distances_df, aes(x = "index", y = "distances")) +
 geom_line(color = "white", size = 2) + theme_minimal() +
 labs(title = "Elbow Method for Choosing eps") +
 theme(panel_grid_minor = element_blank(),
      rect = element_rect(fill = "#202124ff"),
      axis_text = element_text(color = "white"),
      axis_title = element_text(color = "white"),
      plot_title = element_text(color = "white"),
      panel_border = element_line(color = "darkgray"),
      plot_background = element_rect(fill = "#202124ff")
      ))
ggsave(plot=plt, filename='elbow.png', dpi=300)

plt
```

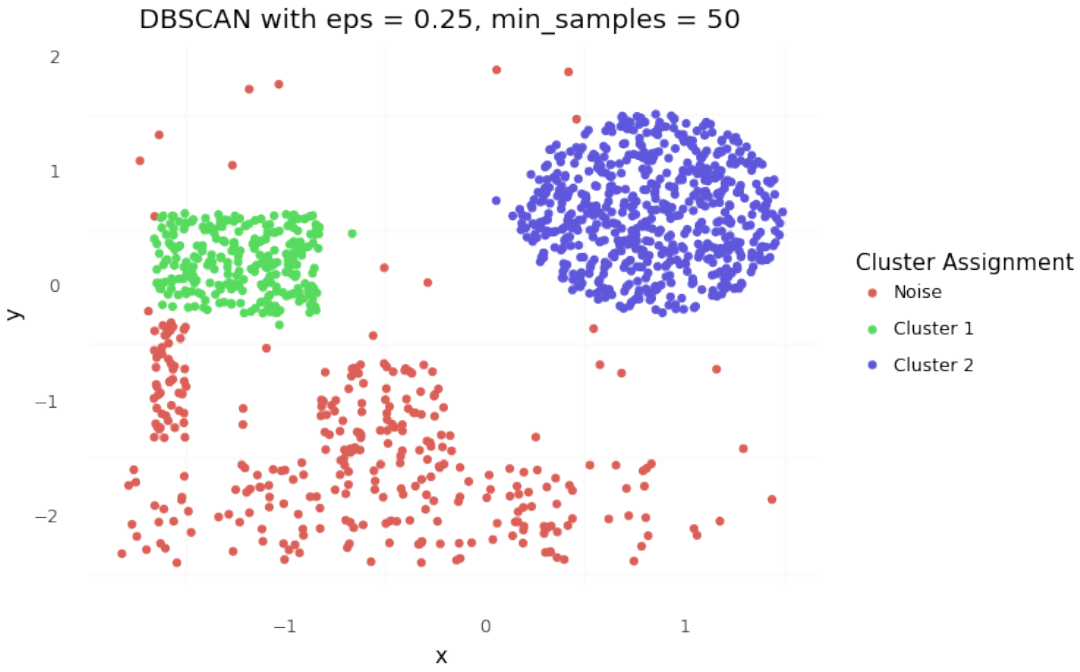


[74]:
```
<ggplot: (324362171)>
```

[207]:
```
# dbscan and plot
db1 = DBSCAN(eps = 0.25, min_samples = 50).fit(data)

labsList = ["Noise"]
labsList = labsList  + ["Cluster " + str(i) for i in range(1,len(set(db1.
 ↪labels_)))]

data["assignments"] = db1.labels_

# grab only clustered data points
d1_clustered = data.loc[(data.assignments >= 0)]
dbscan_silhouette_score = silhouette_score(d1_clustered[["x","y"]],␣
 ↪d1_clustered["assignments"])
print("DBSCAN Silhouette Score: " + str(dbscan_silhouette_score))

(ggplot(data, aes(x = "x", y = "y", color = "factor(assignments)")) +
 geom_point() +
```

```
 theme_minimal() +
 scale_color_discrete(name = "Cluster Assignment",
                      labels = labsList) +
theme(panel_grid_major = element_blank()) +
labs(title = "DBSCAN with eps = 0.25, min_samples = 50"))
```

```
DBSCAN Silhouette Score: 0.7036624118785121
```



[207]: <ggplot: (325500275)>

7.2 DBSCAN hyperparameters

eps

- The first parameter provided to DBSCAN is the eps variable which is the maximum distance between two samples for one to be considered as in the neighborhood of the other. In other words, this parameter helps configure how large a cluster can be. The larger this value is, the bigger a cluster can be in the model. To optimize this value, I employed the elbow method by using the minimum distances from neighbors with a NearestNeighbors model from sklearn. After creating the minimum distance dataframe I graphed the elbow plot (shown above). For the elbow method, one chooses an epsilon value where there is an inflection point on the elbow method graph. The inflection point appears to be about 0.18 and so I started running my model with this value. I then employed the method of Trial & Error to further tune my epilson value. I continued trying different eps values (and min_samples) until I got a good silhoutte score (0.70+) and a good segmentation of the clusters. My optimal epsilon ended up being 0.25 as shown above, which is close to 0.18 value from the elbow method.

min_samples

- The second parameter provided to DBSCAN is the min_samples which is the number of samples in a neighborhood for a point to be considered as a core point. I referred to the first graph of the z-scored data to optimize this parameter. I pinpointed where I thought the centroid of each of the clusters to be and then estimated how many points I think are needed to pool in the appropriate data points for each cluster. I used the centroid points to give me a baseline point to start with. For example, I looked at the right most cluster (oval-shaped one) and thought that there are at least 50 points within that cluster relative to its centroid. I tried counting the points, but it is difficult given that there is some overlap (intra-cluster overlapping). I used 50 as my baseline and then tried different values to see how the graph/clusters are changing. I then adjusted the min_samples value to help get a good silhoutte score (0.70+) and a good separation of clusters. I ended up having my best results when using esp = 0.25 and min_samples = 50 as shown above.

Trial & Error

- In addition to looking at the first graph to help me figure out the optimal parameters to use for the data provided, I also employed the scientific method of Trial & Error. Specifically, I tried changing the parameters to what I believe is better and then would look at the results. Each trial I ran gave me more insight on which directon I should go in terms of changing the parameters. Ultimately, I ran trials until I got the best silhouttee score I can get with a good segmentation of clusters.

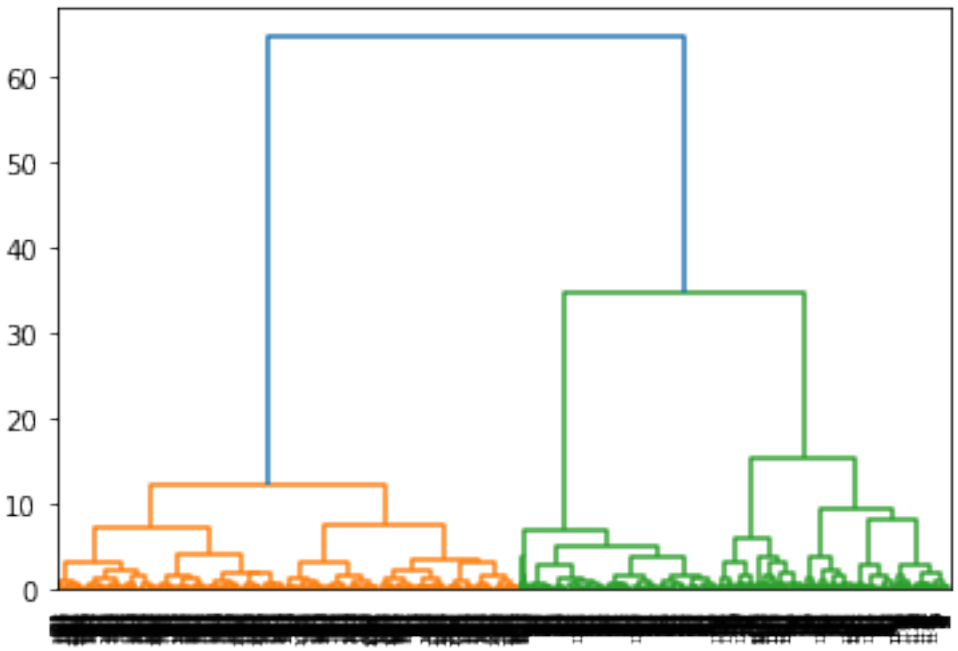
7.3 DBSCAN Graph Description

- The DBSCAN performed very well after tuning of the hyperparameters. After running several tests, I got the highest silhoutte score with eps equal to 0.25 and min_num_samples equal to 50. The silhoutte score tells us how well the data is segmented in terms of its clusters - cohesion and separation. Although a higher silhoutte score is usually considered better, a data scienctist cannot go solely off of the silhoutte score. I know from looking at the original graph, that the data should have either 2 or 3 clusters and the rest of the data should be classified as noise. With this intuition in mind, I continued tuning my DBSCAN hyperparameters to get the best silhoutte score I could get with a good segmentation of the clusters.

[178]:
```
# HCA - 3 clusters, affinity = euclidean, & linkage = ward
hca_ward = AgglomerativeClustering(n_clusters = 3, affinity = "euclidean",
                                   linkage = "ward")
# linkage - distance between the different clusters (method of calculating␣
 ↪these values ie average, furthest, nearest, etc)
#  affinity - intracluster distance calculating tuning

hca_ward.fit(data)
# dendro - for showing the branches
dendro = sch.dendrogram(sch.linkage(data, method='ward'))
# very computationally expensive

membership = hca_ward.labels_
HCA_ward_silhouette_score = silhouette_score(data, membership)
```

```
print("HCA Silhoutte Score (3 clusters, affinity - euclidean, & linkage - ward):
 ↪ " + str(HCA_ward_silhouette_score))
```

```
HCA Silhoutte Score (3 clusters, affinity - euclidean, & linkage - ward):
0.6478264291883887
```



7.4 HCA hyperparameters

n_clusters

- The first parameter provided to HCA is the number of clusters for it to find. I chose 3 clusters because it seems like there should be 3 clusters based on the first graph of the z-scored data.

affinity

- The second parameter provided to HCA is affinity which tells the model which metric to use to compute the linkage. The choices include the following: “euclidean”, “l1”, “l2”, “manhattan”, “cosine”, or “precomputed”. Euclidean distances are ideal when working with continuous data, Manhattan for working with high dimension (many features) data, hamming when working with categorical data, and consine distance used usually when dealing with word counting. I chose Euclidean because the data provided appears to be continuous.

linkage

- The third parameter provided to HCA is linkage which tells the model which linkage criteria to use. The choices include the following: “ward”, “average”, “complete”/“maximum”, or “single”. Single linkage looks at the distance between the 2 closest points in 2 clusters.

Average linkage takes the average distance between each point in one cluster and each point in the other cluster. Complete linkage looks at what is the distance between the 2 furthest data points in different clusters. Centroid linkage looks at the distance between the midpoint of each cluster from the midpoint of another cluster. Ward linkage merges 2 clusters and then assesses the average deviation that changed from the merging of clusters; ward wants to reduce this deviation as much as possible. I ran the model with each of the different linkage criteria and got a highest score with ward although the others' silhoutte scores were very close. Since ward yielded the highest silhoutte score, I stuck with it as shown above.

Trial & Error

- In addition to looking at the first graph to help me figure out the optimal parameters to use for the data provided, I also employed the scientific method of Trial & Error. Specifically, I tried changing the parameters to what I believe is better and then would look at the results. The parameters, affinity and linkage, do not affect an HCA model as much as how eps and min_samples affect DBSCAN, and so tuning of the parameters is not as involved.

8 e) Which Algorithm Did Better?

8.1 Answer:

- The DBSCAN did much better than the HCA model because it got a higher silhoutte score and better separated the data in clusters. As mentioned before, the silhoutte score tells us how well a model segmented data in terms of cluster separation and cohesion. A silhoutte score ranges from (-1, 1) ~ inclusive, with a score closer to +1 being better. The HCA algorithm received a silhoutte score of about 0.65 and the DBSCAN algorithm received a silhoutte score of about 0.70. This implies that the DBSCAN segmented the clusters better than the HCA model. The DBSCAN also did better according to the graph because it classified the loose points as noise and the percieved clusters as clusters. The HCA dendogram suggests there is a hierarchical relationship in the data provided because of the merging/branching observed in the HCA dendrogram. The more merging/branching that a dendrogram contains, the more the HCA algorithm is detecting hierarchical relationships which can be almost be thought of as sub-clusters. The original graph does not suggest there is hierarchical relationship in the data provided because there does not seem to be any sub-clusters within the 2 or 3 preceived clusters. The HCA algorithm however has a lot of branches that get merged which indicates it is detecting a lot of sub-clusters. The over-detection of sub-clusters makes the HCA a poorer choice of an algorithm for this dataset. The HCA algorithm is most likely detecting many sub-clusters because of the prevalence of noise data points in the data provided. The DBSCAN identifies that there are 2 clusters and the rest of the data is classified as noise. These results highlight how an HCA algorithm is not an ideal algorithm to use when there is a lot of noise in a dataset.

9 e) Are the clusters similar between the two algorithms?

- Although the graphs of the DBSCAN and HCA are different by appearance - scatterplot & dendrogram, their results are fairly similar but also have some differences. The main difference is in the number of clusters. DBSCAN has 2 clusters and HCA has 3 clusters (DBSCAN figured this out on its own and HCA took this as an argument). The right cluster that is of

an oval shape is the most dense and this is reflected both on the DBSCAN graph and the HCA dendrogram. In the DBSCAN graph, the very right cluster seems correctly segmented because it mostly consists of points within the spherical shape. The HCA dendrogram represents this same cluster with a green color. The HCA green colored cluster is very dense because there is a strong concentration of merging observed in the very bottom of the dendrogram. A strong concentration of merging at the bottom of a dendrogram indicates a cluster is dense. The similarities of the very left cluster can be also be observed in the DBSCAN graph and HCA dendrogram. The very left cluster is a dense cluster, which is shown in the DBSCAN graph (green cluster) and HCA dendrogram (orange cluster). The middle cluster is the least dense and once again this attribute is reflected on HCA dendrogram, which is a blue color. It is the least dense group and thus has least amount of merging and is the highest merged cluster. The DBSCAN classified this cluster as noise. Overall the results of the middle cluster (middle cluster for HCA and noise for DBSCAN) makes sense in both diagrams considering that the data in that region is dispersed.