ASSIGNMENT 2

K-Means-Hierarchical-Clustering

I Neha Moolchandani declare that I have completed this assignment completely and entirely on my own, without any consultation with others. I understand that any breach of the UAB Academic Honor Code may result in severe penalties.

Neha Moolchandani

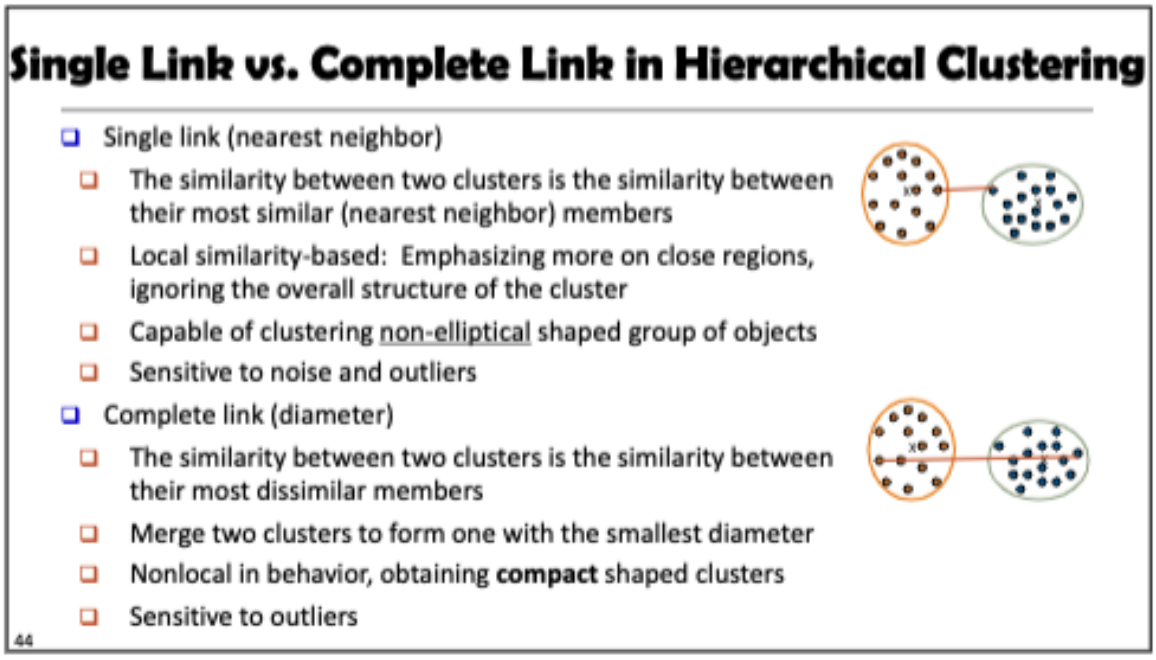
Course: Data Mining | Professor: Dr. Chengcui Zhang

1. **Different clustering algorithms behave different on the same dataset. Single Link and Complete Link are two agglomerative clustering algorithms. (20 pts)**

   Find a data file D such that these two algorithms behave differently on D and display their behavior. (A set of 6 to 10 points probably works well.)

Recourses: (You can use the following implementation of hierarchical clustering or your own implementation)

1. scikit learn: http://scikitlearn.org/stable/modules/clustering.html#clustering (check the Agglomerative Clustering)
2. Matlab: https://www.mathworks.com/help/stats/hierarchicalclustering.html
3. Plot Digitizer to generate 2D datasets: http://plotdigitizer.sourceforge.net/



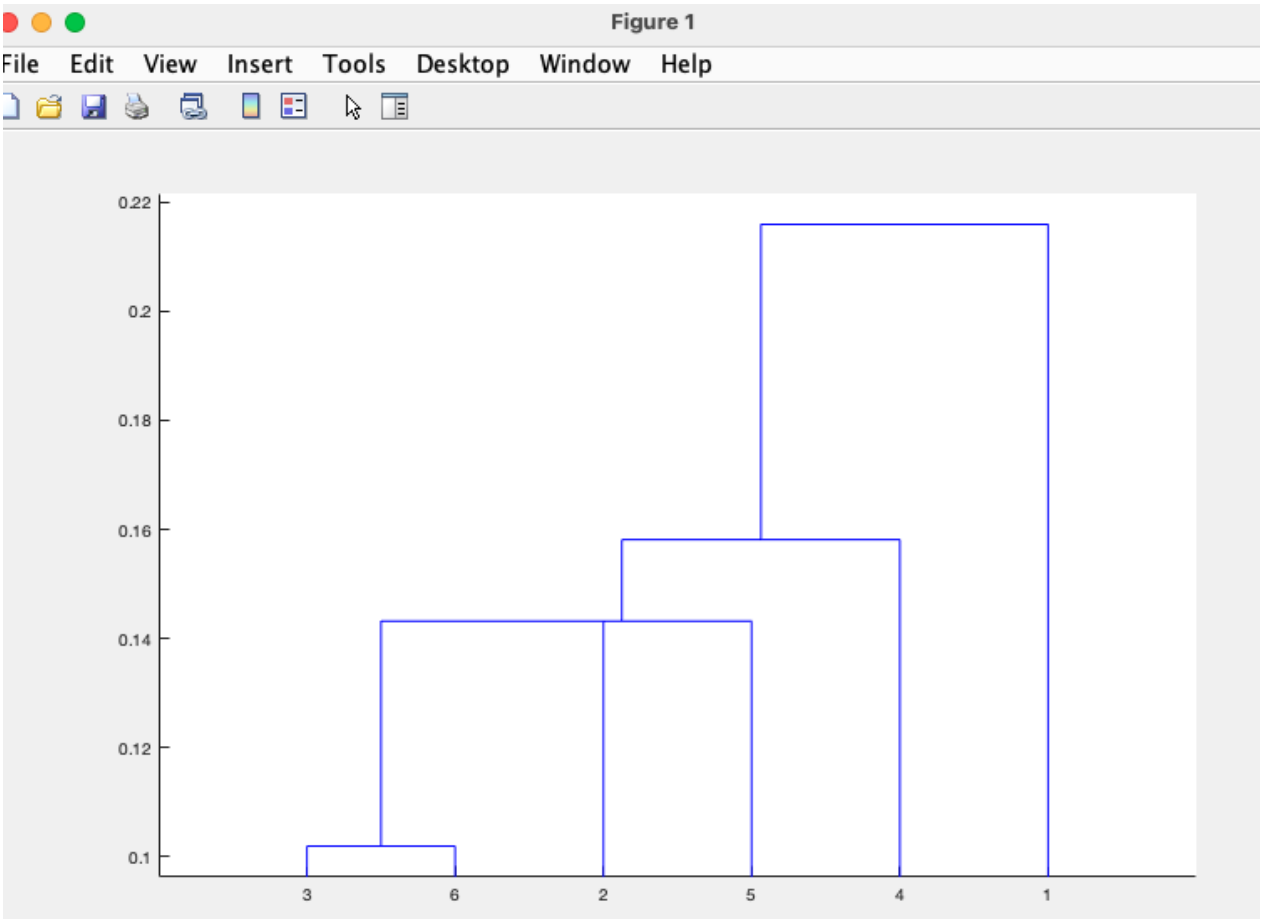


Figure 1
File Edit View Insert Tools Desktop Window Help
0.22
0.2
0.18
0.16
0.14
0.12
0.1
3
6
2
5
4
1

In [38]:
```
Z = single(y)
```

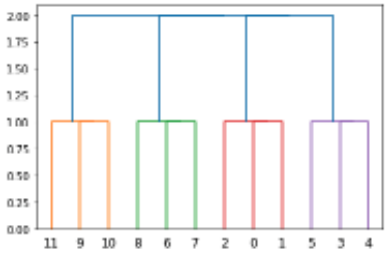
In [39]:
```
Z
```

Out[39]:
```
array([[ 0.,  1.,  1.,  2.],
       [ 2., 12.,  1.,  3.],
       [ 3.,  4.,  1.,  2.],
       [ 5., 14.,  1.,  3.],
       [ 6.,  7.,  1.,  2.],
       [ 8., 16.,  1.,  3.],
       [ 9., 10.,  1.,  2.],
       [11., 18.,  1.,  3.],
       [13., 15.,  2.,  6.],
       [17., 20.,  2.,  9.],
       [19., 21.,  2., 12.]])
```

In [40]:
```
fcluster(Z, 0.9, criterion='distance')
```

Out[40]:
```
array([ 7,  8,  9, 10, 11, 12,  4,  5,  6,  1,  2,  3], dtype=int32)
```

In [49]:
```
Z = linkage(X, 'single')
dn = dendrogram(Z)
plt.show()
```



In [42]:
```
# This above represents the dendrogram for single link
```

In [43]:
```
# Now to compute complete link we can use the same matrix X
```

In [44]:
```
y = pdist(X)
```

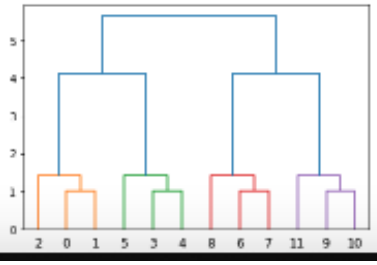
In [45]:
```
Z = complete(y)
```

In [46]:
```
Z
```

Out[46]:
```
array([[ 0.        ,  1.        ,  1.        ,  2.        ],
       [ 3.        ,  4.        ,  1.        ,  2.        ],
       [ 6.        ,  7.        ,  1.        ,  2.        ],
       [ 9.        , 10.        ,  1.        ,  2.        ],
       [ 2.        , 12.        ,  1.41421356,  3.        ],
       [ 5.        , 13.        ,  1.41421356,  3.        ],
       [ 8.        , 14.        ,  1.41421356,  3.        ],
       [11.        , 15.        ,  1.41421356,  3.        ],
       [16.        , 17.        ,  4.12310563,  6.        ],
       [18.        , 19.        ,  4.12310563,  6.        ],
       [20.        , 21.        ,  5.65685425, 12.        ]])
```

In [47]:
```
# This above represents the dendrogram for complete link
# Remeber in both matrixes the first two collumns are represented by the cluster
# the third collumn is the distance wihtin the clusters.
# The last collumn is the size of the new cluster
```

In [48]:
```
Z = linkage(X, 'complete')
dn = dendrogram(Z)
plt.show()
```



Using Jupyter and Scikit learn I graphed the dendrogram to show single and complete list. Here, single Link is shown here in the first graph as the "nearest neighbor" is shown in x cords: ''11 9'' "9 10" are closer than "11 10" or "9 6" or "9 0" or "9 4" we do this for all other colors and can see how each color represents the closeness compared to the other colors and the blue. We can see that complete Link in the second graph shown with the same array is also shown here since between "2 0" we also have a link to " 5 3 4" and between the big blue "0 3" we have a link to "1 11" this same pattern can be used for the other colors green red and purple to show that each link has another link to another color and therefore is complete as you can reach a point to any other point. This shows the diameter of the clusters in that how far they are for each other. Dendrograms show us the distance – the closer or the same color links are much closer than the other colors that they are linked to. The Jupyter notebook is attached in my submission for a closer view.

2. **Two data files (a and b) are given below. (2-dimensional data) They are to be clustered by k-means, using the centers of the three circular regions as initial seeds. 2A:**

797 Centroid 1= (2.8, 5) Centroid 2=(7.7, 1) Centroid 3=(7.8, 9)

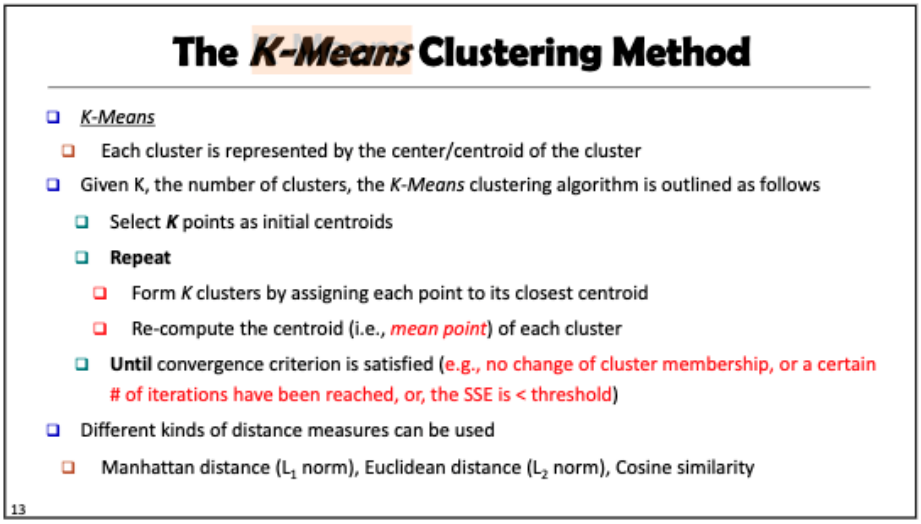
774 Centroid 1: (4, 5) Centroid 2: (7, 1.3) Centroid 3: (7.3, 8)

(Why did I say to use the center of each circle as initial seed, since k-means would not be so clever as to do so? I did because in part (b) the goal is to show that even if we tell k-means the "right" answer, k-means will insist on giving us back the "wrong" answer.)

In 2(b), it provides an example that even if we tell k-means the "right" initial centroids, k-means will insist on giving us back the "wrong" clustering result. You need to run the clustering, show this result, and also explain why this happens in words.

Use a k-means implementation (either an existing one or an implementation of your own) and perform it on the two datasets and show the final clustering results, using the provided "right" centroids (see the attached data files) as the initial seeds (k=3).

Briefly explain the result of (b). (40 pts)



Resources:

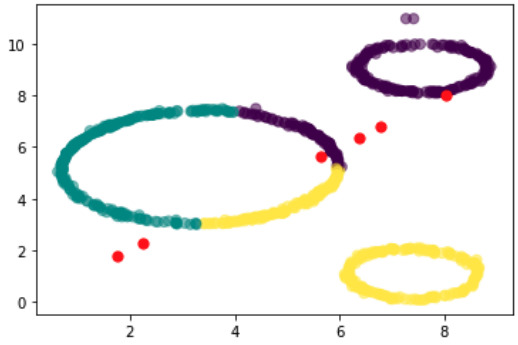
1. scikit Learn: http://scikitlearn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklea rn.cluster.KMeans

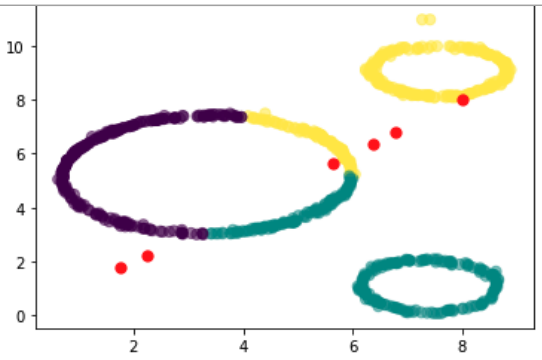
Wow! How cool! Ok so I used the code given with scitkitlearn and messed around with clusters and k-means algorithm. I did the code on both Data sets. We can see how I got two different clusters. This is because with each iteration from the start we end up with the solution that is nearest to best which is always not the perfect answer. Because K-means features we have to consider variance which will produce difference answers given the ranges or centroids we put in. No matter if we were to do k-means on many of the same set, the answer would change but would be nearby every time. You can even with both datasets, we changed the centroids but each cluster in each of the changes turned out to be very similar!

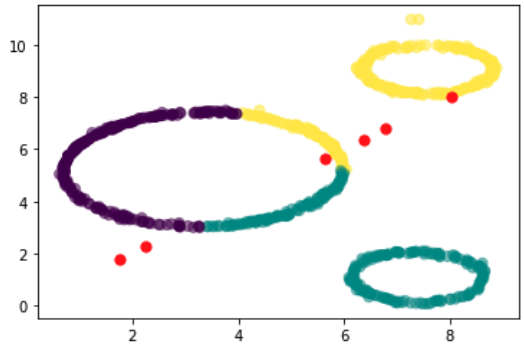
*Something I noticed was that the splicing was only taking integers and not floats so I had to truncate them in order for it to work effectively when splicing to compute the centroid centers. For both of the two out of the three centroids given, error was posed saying x and y had to be of same size. The rand points didn't show but the clusters did.

It was interesting to see how given the centroids, the algorithm iterated until if would find the optimal I believe this is based because of the math where we squared distance between the data points and centroid to find minimum.

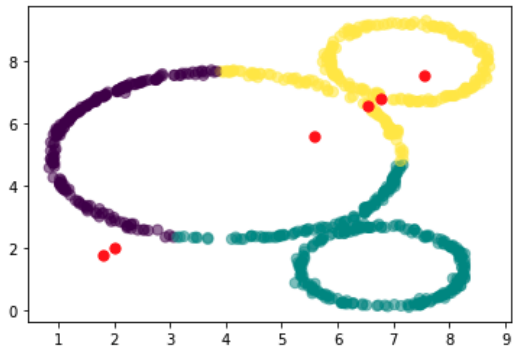
DATASET A:

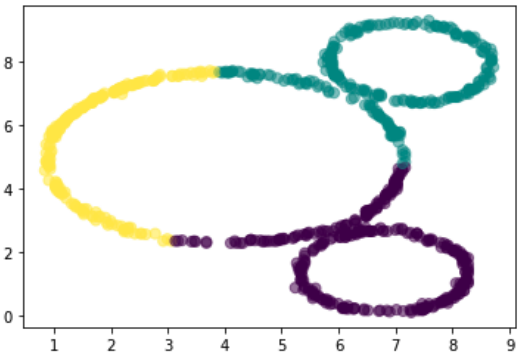


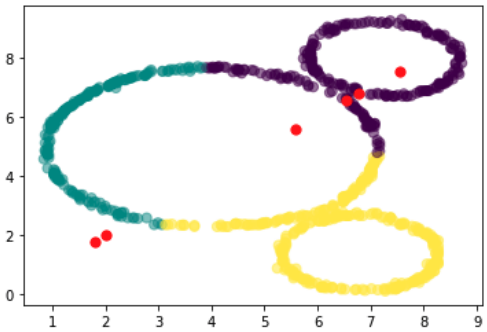


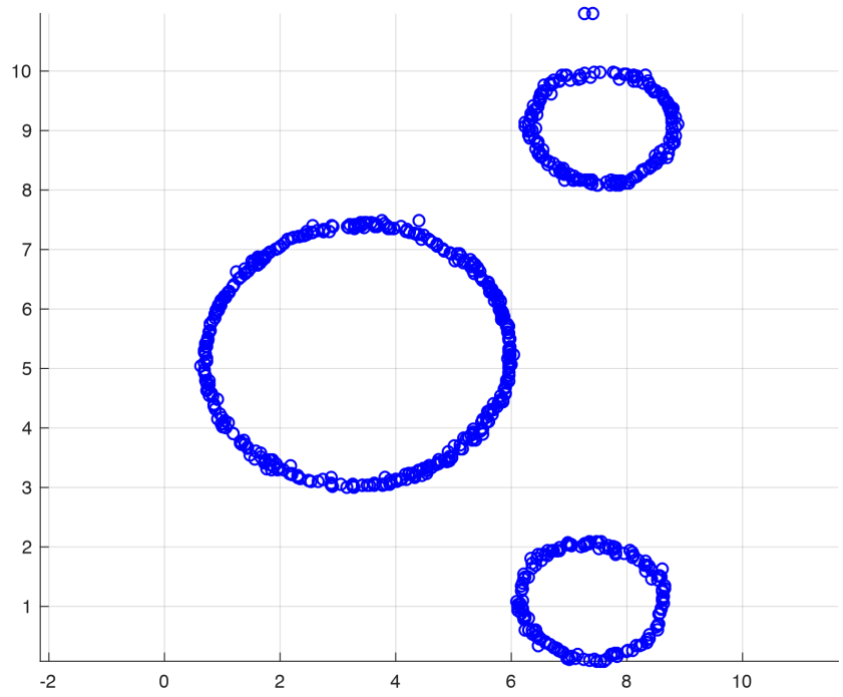


DATASET B:

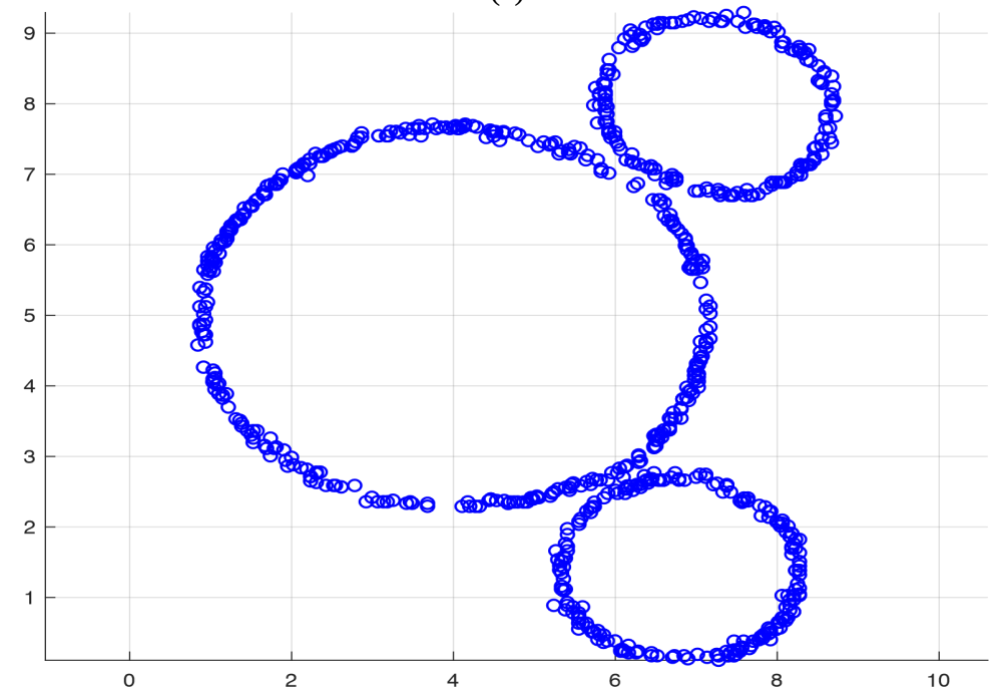








(a)



(b)

3. **Either: Prove that k-means will produce k clusters, all nonempty, or: Give an example of a set D of data points (with no repeated data point), a value for k (k<=n, where n is the number of data objects), and a set of k data points as initial seeds, such that some cluster becomes empty. (30 pts)**
A motivation for this problem: Many or most of you will become professional programmers. The programs you write are supposed to work all the time, not just 999 times out of 1000. The pseudocode for k-means in the textbook will fail if indeed one of the clusters becomes empty.

Let's choose we want approximately three clusters. Therefore, k is 3. If we assume we have points such as:

	x	y
A	1	3
B	2	5
C	2	6
D	2	7
E	3	7
F	5	7
G	6	6

The after computing Euclidean distance and k means many times we will see that after the first iteration A, C, B, and D will be in one cluster and E and F will be in another. G will be in the third cluster. We will notice that the distance between A and E is larger than E and F and therefore will join the second cluster. Again, we perform this step and we will see that the cluster will move closer to E as B C D are in one cluster and closest to E. Finally, E is out of cluster 2 and F will now go into cluster three with G therefore we no longer have a cluster two.
https://user.ceng.metu.edu.tr/~tcan/ceng465_f1314/Schedule/KMeansEmpty.html

(Prove it by code and get outputs that cluster is empty. Otherwise you can solve it using the euc distance after each distance and update the centroids… etc.)
In the end it will be empty. Take initial centroid 3 initial seeds. Pick 3-rand points.

Question: is it safe to write the code for k-means as the textbook, or will code written like that get you into some trouble with your manager (not initially, but after a while)? If you decide to try to find a malevolent data you can try the following. Try a data file of 6 to 12 data points in the plane so that when k-means is performed to k = 3 or k = 4 clusters, in iteration 2 (or iteration 3 or ...), one of the

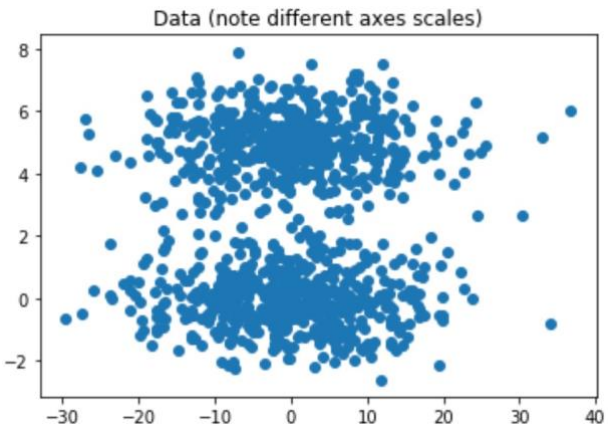
clusters becomes empty. The initial seeds must be data points; and your example should not rely on ties to accomplish its goals. (Comment: this effort might be easier if you let most of the points be collinear.) Show the step-by-step process of the clustering.

I don't think it is too safe to use the code from the textbook, if we apply that code to any given single dataset the k-means will produce different clusters every time. Given the definition of k-means we will have to keep running the Euclidean distance and finding the mean until the clusters reach the same position after multiple times. This is perhaps so we find the objective function to minimize the squared error function and therefore have the least error when performing k-means. We can use the above data set with X and Y:
See how we get a different cluster every time: You do this 100 times and

Label	Vector	Cluster id	Cluster centroid
A	1,3	0	1,3
B	2,5	2	2.25,6.25
C	2,6	2	2.25,6.25
D	2,7	2	2.25,6.25
E	3,7	2	2.25,6.25
F	5,7	1	5.5,6.5
G	6,6	1	5.5,6.5

Bonus (10 pts): Propose a solution in case of an empty cluster.
To avoid a empty cluster, you can code it in python in a try/catch statement or use a different modified k-means algorithm where the solution is o add the current cluster centers to to points as you find the new cluster centers for each next iteration.

4. Given 1000 points (visualized below and see the attached data file). Notice that the X dimension has a much larger value range than that of Y. Perform K-means (K=2) clustering on this dataset and visualize it. Then do zscore normalization on the dataset and perform K-means again, and visualize it. Finally, do Min-Max normalization (e.g., normalize(x) =(x-min)/(max-min)) on the dataset and run Kmeans again. Submit the plots as well as your code and highlight the normalization part. (20 pts)



Execution of the K-Means Clustering Algorithm

Select *K* points as initial centroids

Repeat

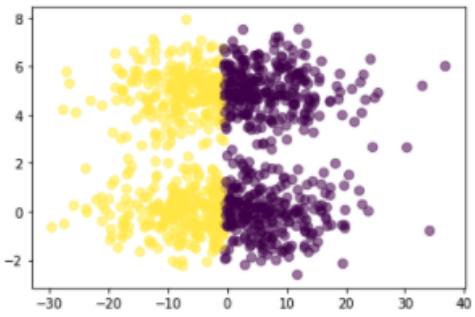
- Form *K* clusters by assigning each point to its closest centroid
- Re-compute the centroids (i.e., *mean point*) of each cluster

Until convergence criterion is satisfied

K-Means:

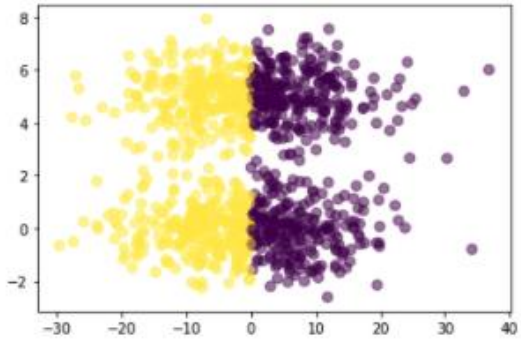
```
: # Assign2-Q4:
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

data = pd.read_excel (r'/Users/nehamoolchandani/Desktop/Coding/CS463- Data Mining/Q4-dataset.xl
df = pd.DataFrame(data, columns= ['X','Y'])
print (df)
kmeans = KMeans(n_clusters=2).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)

plt.scatter(df['X'], df['Y'], c= kmeans.labels_.astype(float), s=50, alpha=0.5)
# plt.scatter(centroids[:,4], centroids[:,5], c='red', s=50)
# plt.scatter(centroids[:,7], centroids[:,1.3], c='red', s=50)
# plt.scatter(centroids[:,7.3], centroids[:,8], c='red', s=50)
plt.show()
```

```
             X         Y
0     7.596496 -1.060931
1    -6.491310 -0.579551
2    -2.496095 -1.684283
3    -5.274757 -1.537358
4   -17.715394 -0.614091
..         ...       ...
995  21.055374  5.054196
996  -7.765841  3.194533
997  -9.804282  6.509640
998  11.875210  5.693303
999  17.406041  4.569326

[1000 rows x 2 columns]
[[ 7.29274158  2.61608473]
 [-8.34650003  2.3539061 ]]
```



Z-Score

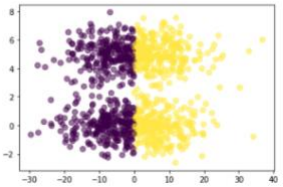
```
             X         Y
0     7.596496 -1.060931
1    -6.491310 -0.579551
2    -2.496095 -1.684283
3    -5.274757 -1.537358
4   -17.715394 -0.614091
..         ...       ...
995  21.055374  5.054196
996  -7.765841  3.194533
997  -9.804282  6.509640
998  11.875210  5.693303
999  17.406041  4.569326

[1000 rows x 2 columns]
[[ 7.70820373  2.59258686]
 [-7.91757774  2.39095613]]
```




```
[[ 0.79312944 -1.31975881]
 [-0.64317419 -1.14088491]
 [-0.23584729 -1.55138752]
 ...
 [-0.98094389  1.49335702]
 [ 1.22936011  1.19001783]
 [ 1.79324876  0.77236431]]

Z-score for Data-Q4 :
 [[ 1. -1.]
 [-1.  1.]
 [-1.  1.]
 ...
 [-1.  1.]
 [ 1. -1.]
 [ 1. -1.]]
```

K-Means again:

```
Z-score for Data-Q4 :
 [[ 1. -1.]
 [-1.  1.]
 [-1.  1.]
 ...
 [-1.  1.]
 [ 1. -1.]
 [ 1. -1.]]
              X         Y
0      7.596496 -1.060931
1     -6.491310 -0.579551
2     -2.496095 -1.684283
3     -5.274757 -1.537358
4    -17.715394 -0.614091
..          ...       ...
995   21.055374  5.054196
996   -7.765841  3.194533
997   -9.804282  6.509640
998   11.875210  5.693303
999   17.406041  4.569326

[1000 rows x 2 columns]
[[-7.91757774  2.39095613]
 [ 7.70820373  2.59258686]]
```

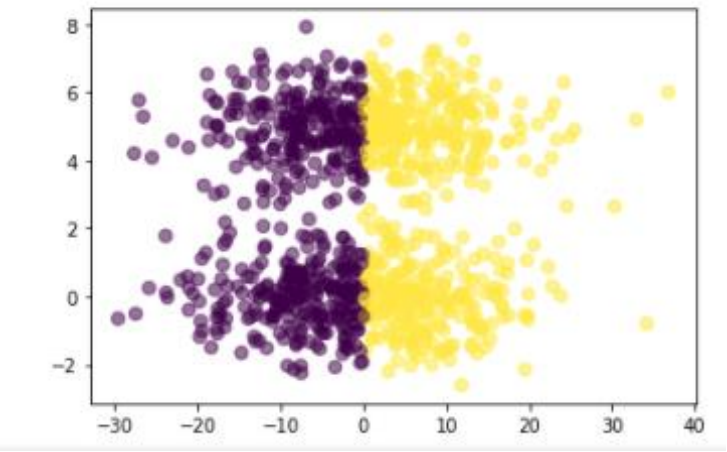


Min-Max:

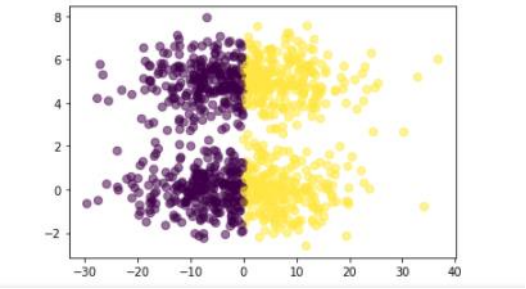
```
This is the data normalized:
 [[0.56053408 0.14690248]
 [0.34837049 0.19264466]
 [0.40853878 0.08766968]
 ...
 [0.29847683 0.86628106]
 [0.6249719  0.78871023]
 [0.70826671 0.6819066 ]]
              X         Y
0      7.596496 -1.060931
1     -6.491310 -0.579551
2     -2.496095 -1.684283
3     -5.274757 -1.537358
4    -17.715394 -0.614091
..          ...       ...
995   21.055374  5.054196
996   -7.765841  3.194533
997   -9.804282  6.509640
998   11.875210  5.693303
999   17.406041  4.569326
```

K-means again:

```
KMeans(n_clusters=2)
```




```
[1000 rows x 2 columns]
KMeans(n_clusters=2)
```



My k-means was wrong I didn't realize till after I was supposed to cal k-means after z and normalization. That's my bad.

5. **Bonus**: Kernel Kmeans demo with Gaussian Kernel (5 pts)
   Use the attached Matlab code and the data file to produce a clustering result. Note: to import the data into the matrix X, you need to replace the following codes in 'demo.m'

   Replace:

```
n = 500;
[X,label] = kmeansRnd(d,k,n);
```

   With:

```
n = 508; % the total # of objects
X=[copy and paste the 2D data from the excel file]; X=X';
```

   To run the demo, just type 'demo' after the Matlab prompt.

```
        clear; close all;
d = 2;
k = 2;

n = 508; % the total # of objects
X=[copy and paste the 2D data from the excel file]; X=X';

init = ceil(k*rand(1,n));
[y,model,mse] = knKmeans(X,init,@knGauss);
plotClass(X,y)

idx = 1:2:n;
Xt = X(:,idx);
t = knKmeansPred(model, Xt);
plotClass(Xt,t)
```

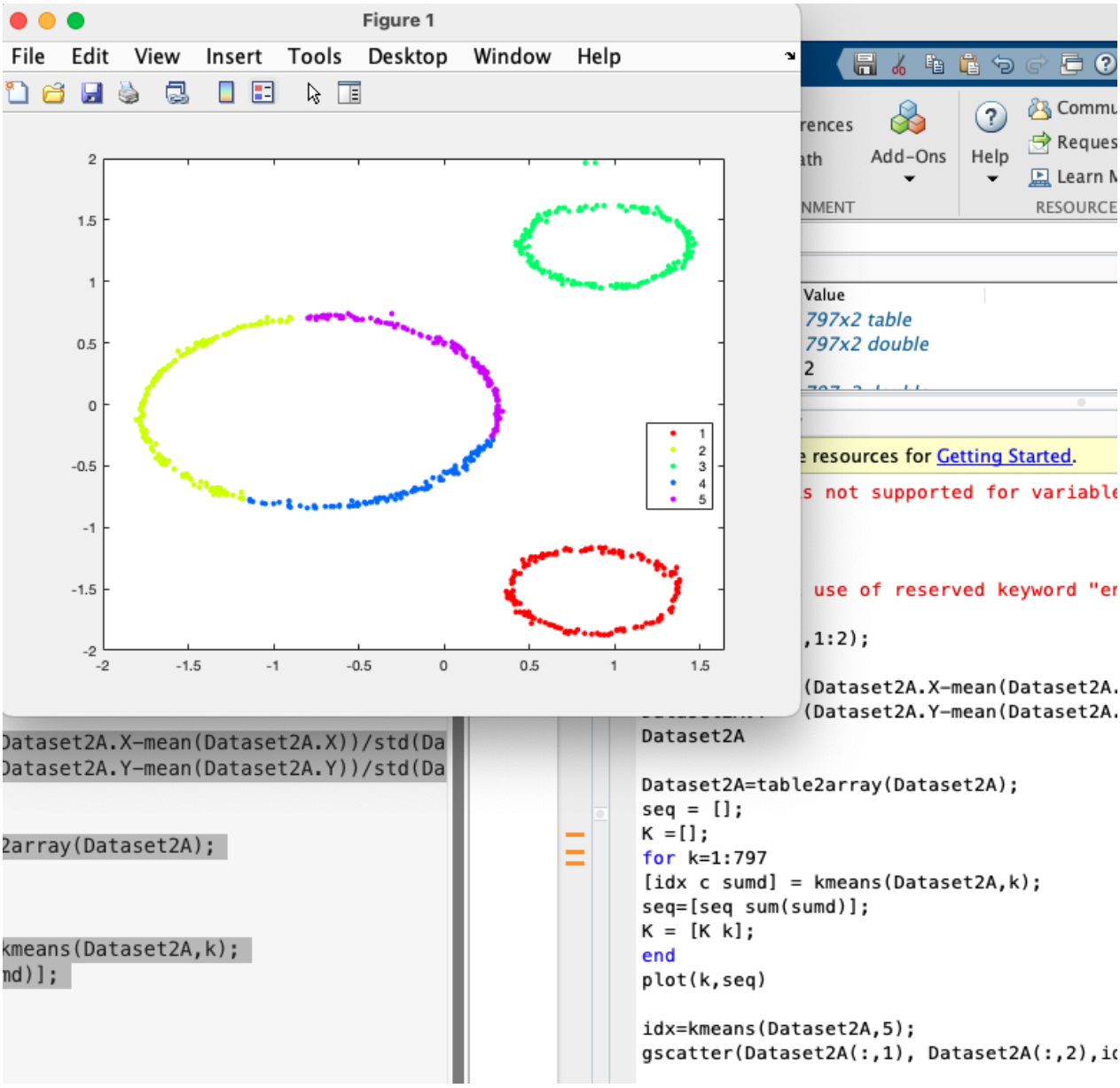


Figure 1
File Edit View Insert Tools Desktop Window Help
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2 -1.5 -1 -0.5 0 0.5 1 1.5
1
2
3
4
5
Add-Ons
Help
Value
797x2 table
797x2 double
2
resources for Getting Started.
not supported for variable
use of reserved keyword "er
,1:2);
(Dataset2A.X-mean(Dataset2A.
(Dataset2A.Y-mean(Dataset2A.
Dataset2A
Dataset2A=table2array(Dataset2A);
seq = [];
K =[];
for k=1:797
[idx c sumd] = kmeans(Dataset2A,k);
seq=[seq sum(sumd)];
K = [K k];
end
plot(k,seq)
idx=kmeans(Dataset2A,5);
gscatter(Dataset2A(:,1), Dataset2A(:,2),id