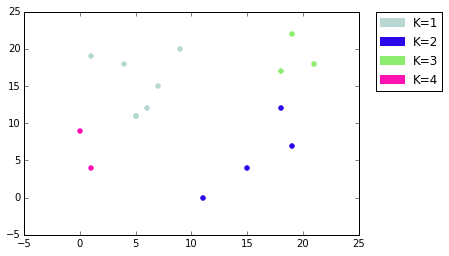
Data Analysis HW4

# Question 1 Answers

 A.) Clustering plot when data points are plotted in order.

s

Cluster Centers:

(Xave, Yave, NumPoints)

{1: (5.2857142857142865, 15.142857142857142, 7),

2: (15.75, 5.75, 4),

3: (19.333333333333332, 19.0, 3),

4: (0.5, 6.5, 2)}

Data point order:

X Y Cluster

0 6 12 1

1 19 7 2

2 15 4 2

3 11 0 3

4 18 12 2

5 9 20 1

6 19 22 2

7 18 17 2

8 5 11 1

9 4 18 1

10 7 15 1

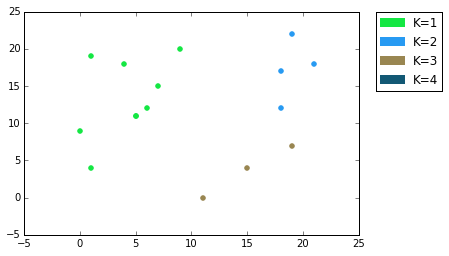
11 21 18 2

12 1 19 4

13 1 4 1

14 0 9 1

15 5 11 1

 B.) Clustering plot when data points are plotted in reverse order.

Cluster Centers:

(Xave, Yave, NumPoints)

{1: (4.2222222222222223, 13.222222222222221, 9),

2: (19.0, 17.25, 4),

3: (15.0, 3.666666666666667, 3)}

Data points order:

X Y Cluster

15 5 11 1

14 0 9 1

13 1 4 1

12 1 19 4

11 21 18 2

10 7 15 1

9 4 18 1

8 5 11 1

7 18 17 2

6 19 22 2

5 9 20 1

4 18 12 2

3 11 0 3

2 15 4 2

1 19 7 2

0 6 12 1

C.) Use Rand index to find the difference between the two clusterings obtained in (a) and (b). Informally, identify the cluster in (a) that has been altered the most in clustering done in (b). Give informal explanation of why this cluster broke apart the most.

**Rand Index**

Rand Index: 0.566666666667

**Cluster identification and explanation:** The cluster from part A that was changed the most was cluster 1 and 2. Clusters 1 and 2 both merged in the second clustering to form one large cluster. This makes sense when analyzing two parts of the algorithm. First is the fact that distance to a cluster is based off of the centroid of the cluster. Second is the order that the points are evaluated. When moving through the original data point list, points (1, 4), (0, 9), and (5, 11) aren’t evaluated until after most of the other points. By this time, the centroid of cluster 1 has moved far enough away that points (1, 4) and (0, 9) are no longer close enough to be considered to be a part of cluster 1. So, they are put in their own cluster. When the algorithm iterates through the list of data points backwards, points (1, 4), (0, 9), and (5, 11) are evaluated first. Thus, the centroid for cluster 1 is close enough to all of these points to include them in the same cluster. Furthermore, because these points are evaluated first, the centroid for cluster 1 is allowed to move to a position close enough to include all of the data points

**Python Code for Question 1**

def prob1():

global prob1Data

global clusterCenters

global clusterCenters2

global prob1Data2

listdat = [(6, 12), (19, 7), (15, 4), (11, 0),

(18, 12), (9, 20), (19, 22), (18, 17),

(5, 11), (4, 18), (7, 15), (21, 18), (1, 19),

(1, 4), (0, 9), (5, 11)]

#Initialize the data

prob1Data = pd.DataFrame(listdat, columns=['X','Y'])

prob1Data['Cluster'] = pd.Series(np.zeros(prob1Data.shape[0]))

#Reverse the data order and create a new data set for it.

listdat.reverse()

prob1Data2 = pd.DataFrame(listdat, columns=['X','Y'])

prob1Data2['Cluster'] = pd.Series(np.zeros(prob1Data.shape[0]))

#RUN PART A

clusterCenters = sequantialClusteringAlgorithm(prob1Data)

#Plot the data on a scatter plot.

colorHandles = []

#Make a list to hold the plotted cluster radi.

clusterCircles = []

#Create each cluster scatter plot.

for cluster in range(1, 5):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob1Data[prob1Data.Cluster == cluster].X, prob1Data[prob1Data.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

clusterCircles.append(plt.Circle((clusterCenters[cluster][0], clusterCenters[cluster][1]), 12, color=randColor, fill=False, clip\_on=False))

#plt.scatter(prob1Data[prob1Data.Cluster == 1].X, prob1Data[prob1Data.Cluster == 1].Y, label='Cluster 1', color=)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#RUN PART B

plt.clf()

clusterCenters2 = sequantialClusteringAlgorithm(prob1Data2)

#Plot the data on a scatter plot.

colorHandles = []

#Create each cluster scatter plot.

for cluster in range(1, 5):

randColor = [random.random(), random.random(), random.random()]

colorHandles.append(matplotlib.patches.Patch(color=randColor, label='K=' + str(cluster)))

plt.scatter(prob1Data2[prob1Data2.Cluster == cluster].X, prob1Data2[prob1Data2.Cluster == cluster].Y, label="Cluster " + str(cluster), color=randColor)

plt.legend(handles=colorHandles, bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()

#RUN PART C

randindex = randIndex(prob1Data, prob1Data2)

print("Rand Index: " + str(randindex))

'''

Clusters 1, 2, and 3.

Using incremental average newave = oldave + (an−oldave)/n.

'''

def sequantialClusteringAlgorithm(dataSet):

clusterCenters = {}

#Algorithm Parameters.

theta = 12

maxClusters = 4

numClusters = 1

#Set first point to its own cluster.

#For each data point

#1) Calculate distance to closest cluster center

#2) If dist < alg and numClusters < 4

#Create new cluster with data point.

#3) Else, add data point to cluster.

clusterCenters[numClusters] = (dataSet.ix[0].X, dataSet.ix[0].Y, 1)

dataSet['Cluster'].loc[0] = numClusters

for index in range(1, dataSet.shape[0]):

distance = 999999999

clusterToAssign = 0

for cluster in clusterCenters.keys():

tempDist = dist(dataSet.ix[index], clusterCenters[cluster])

if tempDist < distance:

distance = tempDist

clusterToAssign = cluster

if distance <= theta and numClusters <= maxClusters:

dataSet['Cluster'].loc[index] = clusterToAssign

newX = clusterCenters[clusterToAssign][0] + (dataSet.ix[index].X - clusterCenters[clusterToAssign][0])/(clusterCenters[clusterToAssign][2]+1.0)

newY = clusterCenters[clusterToAssign][1] + (dataSet.ix[index].Y - clusterCenters[clusterToAssign][1])/(clusterCenters[clusterToAssign][2]+1.0)

newSize = clusterCenters[clusterToAssign][2] + 1

clusterCenters[clusterToAssign] = (newX, newY, newSize)

dataSet['Cluster'].loc[index] = clusterToAssign

elif distance > theta and numClusters < maxClusters:

numClusters += 1

dataSet['Cluster'].loc[index] = numClusters

clusterCenters[numClusters] = (dataSet.ix[index].X, dataSet.ix[index].Y, 1)

dataSet['Cluster'].loc[index] = numClusters

elif numClusters >= maxClusters:

#print("At Max")

dataSet.ix[index].Cluster = clusterToAssign

newX = clusterCenters[clusterToAssign][0] + (dataSet.ix[index].X - clusterCenters[clusterToAssign][0])/(clusterCenters[clusterToAssign][2]+1.0)

newY = clusterCenters[clusterToAssign][1] + (dataSet.ix[index].Y - clusterCenters[clusterToAssign][1])/(clusterCenters[clusterToAssign][2]+1.0)

newSize = clusterCenters[clusterToAssign][2] + 1

clusterCenters[clusterToAssign] = (newX, newY, newSize)

dataSet['Cluster'].loc[index] = clusterToAssign

return clusterCenters

def dist(point1, point2):

sumsq = 0

for index in range(0, len(point1) - 1):

sumsq += math.pow(point1[index] - point2[index], 2)

return math.pow(sumsq,.5)

def randIndex(clustering1, clustering2):

f00 = 0

f01 = 0

f10 = 0

f11 = 0

for firstIndex in range(0, clustering1.shape[0] - 1):

for secondIndex in range(firstIndex + 1, clustering1.shape[0]):

if clustering1.iloc[firstIndex].Cluster != clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster != clustering2.iloc[secondIndex].Cluster:

f00 += 1

elif clustering1.iloc[firstIndex].Cluster != clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster == clustering2.iloc[secondIndex].Cluster:

f01 += 1

elif clustering1.iloc[firstIndex].Cluster == clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster != clustering2.iloc[secondIndex].Cluster:

f10 += 1

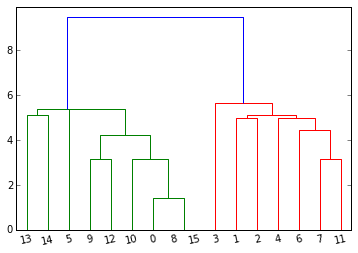
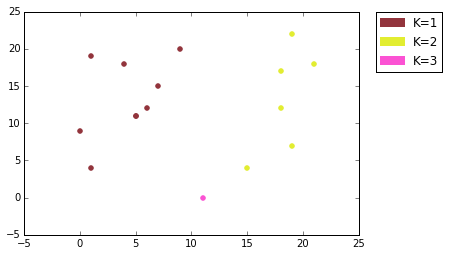
elif clustering1.iloc[firstIndex].Cluster == clustering1.iloc[secondIndex].Cluster and clustering2.iloc[firstIndex].Cluster == clustering2.iloc[secondIndex].Cluster:

f11 += 1

return (f00 + f11) / float(f00 + f01 + f10 + f11)

# Question 2 Answers

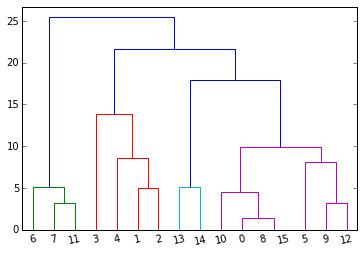
A.) Single Link Hierarchical Clustering

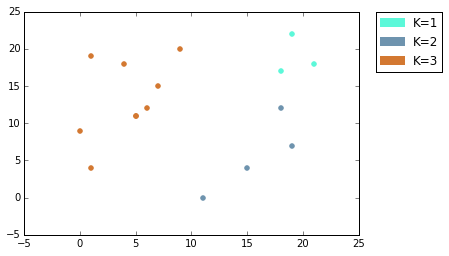


Distance at 5.4 to split data into 3 clusters.

B.) Complete Link Hierarchical Clustering

Distance at 20 to split data into 3 clusters.





C.) Sum Squared Errors

**Comparison and Comment on Clustering SSEs:** The first clustering has a much higher SSE than the second clustering. Looking at the graphs of points, this makes sense because the first clustering has more spread out points in cluster 2 than the second clustering. Since cluster 2 in the first clustering incorporates more points further away from its mean, and since the measurement is a squared value, the first clustering has a higher sum squared error. This also makes sense from the conceptual standpoint. Reduced SSE means that the clusters are more globular. As we can see in the second clustering graph, its clusters are much more globular than the first clustering’s. Thus, we would expect to see a smaller SSE for the second clustering.

Sum squared error for single link: 551.777777778

Cluster **contributing most** to SSE: 1

Cluster SSE: 293.111111111

Sum squared error for complete link: 427.277777778

Cluster **contributing most** to SSE: 3

Cluster SSE: 293.111111111

**Note:** Look at each graph individually above to identify which cluster is labeled 1, and which is labeled 3. Each clustering has labeled the clusters with differing numeric labels.