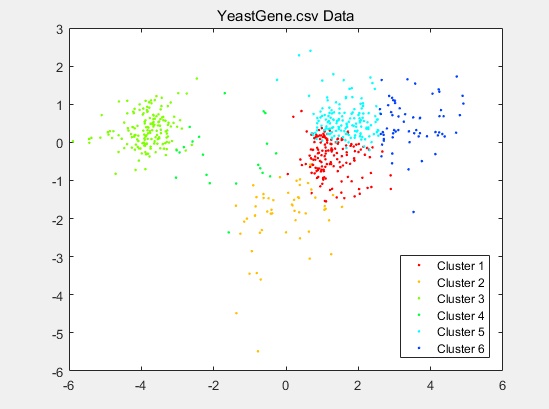
1. The cluster centroids obtained on YeastGene dataset after the T iterations.

New Centroid:   
Cluster 0 : -0.241271428571 -0.12875 0.06225 0.17335 0.217914285714 1.65169285714 1.90532142857   
Cluster 1 : -0.953509803922 -1.47164705882 0.0775294117647 -0.179490196078 -1.00482352941 1.15121568627 0.968803921569   
Cluster 2 : 0.165103658537 0.0916524390244 -0.103884146341 -0.552585365854 -0.630085365854 -1.72318292683 -1.75481097561   
Cluster 3 : 0.0232857142857 0.25080952381 -0.276952380952 -0.364 -0.735428571429 -0.894619047619 0.700142857143   
Cluster 4 : -0.00155555555556 0.15650617284 0.35625308642 0.701580246914 1.00971604938 1.84231481481 1.64341975309   
Cluster 5 : -0.0393289473684 0.153947368421 0.436078947368 1.10581578947 1.44871052632 3.01634210526 2.82938157895





1. The order of merging in the hierarchical clustering on the utilities dataset

[[12, 21], 23]  
[[10, 13], 24]  
[[4, 24], 25]  
[[7, 23], 26]  
[[20, 25], 27]  
[[14, 19], 28]  
[[1, 18], 29]  
[[15, 26], 30]  
[[28, 29], 31]  
[[2, 27], 32]  
[[8, 16], 33]  
[[30, 32], 34]  
[[22, 34], 35]  
[[9, 31], 36]  
[[35, 36], 37]  
[[6, 37], 38]  
[[3, 38], 39]  
[[33, 39], 40]  
[[17, 40], 41]  
[[11, 41], 42]  
[[5, 42], 43]

1. The codes of your K-means and hierarchical clustering algorithm implementation

[H-Clustering]

**import** sys  
**import** math  
**import** csv  
**from** collections **import** defaultdict  
**from** numpy **import** \*  
input\_file = None  
output\_file = None  
mData = 0  
nData = 0  
data = []  
link\_matrix = []  
  
**def** config(args = [**"Utilities.csv"**,**"output.txt"**]):  
 **global** input\_file, output\_file, mData, nData  
 **global** count, data  
 input\_file = args[0]  
 output\_file = args[1]  
 **with** open (input\_file, **'r'**) **as** f:  
 f = csv.reader(f, delimiter=**','**)  
 **for** line **in** f:  
 nData = len(line)  
 mData += 1  
 vec\_line = [float(i) **for** i **in** line]  
 data.append(vec\_line)  
  
  
**def** execute():  
 *# Calculate Distance Matrix  
 # To make it easier finding 2 objects with min distance,   
 # You should initialize distance[i][i] = positive\_infinity  
 # Note: there is no sys.maxint in python3, use float(inf)  
  
 # Repeat until only one cluster remain* distance=defaultdict(dict)  
  
 sub={}  
 t1=0  
 **for** row,a **in** enumerate(data):  
  
 t1=t1+1  
 **for** col,b **in** enumerate(data):  
 **if** row==col:  
 distance[row][col]=float(**"inf"**)  
 **continue** distance[row][col]=euclide\_dist(a,b)  
  
  
 **print** t1  
  
 **for** t **in** range(mData-1):  
  
  
 **for** i **in** range(mData):  
  
 allv=[]  
 mini=0  
 mx=0  
 my=0  
 **print** distance  
 **for** xx **in** distance:  
 **for** yy **in** distance:  
  
 allv.append(distance[xx][yy])  
 mini=min(allv)  
  
 **if** mini==float(**'inf'**):  
 **return** *#get min coordinate* **for** xx1 **in** distance:  
 **for** yy1 **in** distance:  
 **if** distance[xx1][yy1]==mini:  
 mx=xx1  
 my=yy1  
 **print** mx,my  
 link\_matrix.append([[my+1,mx+1],t1+1])  
  
 **for** xx3 **in** range(t1):  
 **if** xx3==mx **or** my:  
 distance[t1][xx3]=float(**'inf'**)  
  
 distance[t1][xx3]=min(distance[mx][xx3],distance[my][xx3])  
 distance[xx3][t1]=min(distance[mx][xx3],distance[my][xx3])  
 distance[t1][t1]=float(**'inf'**)  
  
 *#removing value* **for** xx2 **in** distance:  
 **for** yy2 **in** distance:  
 **if** xx2==mx:  
 **for** i1 **in** distance:  
 distance[xx2][i1]=float(**'inf'**)  
 distance[i1][xx2]=float(**'inf'**)  
 **if** yy2==my:  
 **for** i2 **in** distance:  
 distance[i2][yy2]=float(**'inf'**)  
 distance[yy2][i2]=float(**'inf'**)  
  
 t1=t1+1  
  
  
  
  
 *# find 2 closest cluster i1 i2  
  
 # Merge data of i1 and i2  
 # One way to do is updating row and col of i1 and setting row and col of i2 to be positive\_infinity***def** euclide\_dist(a\_list,b\_list):  
 **return** math.sqrt(sum((a\_list[i]-b\_list[i])\*\*2 **for** i **in** range(min(len(a\_list),len(b\_list)))))  
  
**def** output():  
 **with** open(output\_file,**'w'**) **as** f:  
 f.write(**"Linkage: \n"**)  
 **for** i **in** link\_matrix:  
 f.write(str(i))  
 f.write(**'\n'**)  
 f.close()  
  
**def** main():  
 arg = sys.argv[1:]  
 **if** len(arg) > 1:  
 config(arg)  
 **else**:  
 config()  
 execute()  
 output()  
  
main()

[K-means]

**import** sys  
**import** math  
**import** csv  
**from** collections **import** defaultdict  
**global** n\_inter, m\_data, n\_data, n\_cluster  
**global** data, centroid\_data, new\_cluster\_centroid  
**global** input\_file, centroi\_file, output\_file  
**global** cluster\_id  
  
**import** numpy **as** np  
  
**def** config(args = [**"YeastGene.csv"**,**"YeastGene\_initial\_Centroids.csv"**,7,**"output.txt"**]):  
 **global** n\_inter, m\_data, n\_data, n\_cluster  
 **global** data, centroid\_data, new\_cluster\_centroid  
 **global** input\_file, centroi\_file, output\_file  
 data = []  
 centroid\_data = []  
 input\_file = args[0]  
 centroi\_file = args[1]  
 output\_file = args[3]  
 n\_inter = args[2]  
  
 m\_data = 0  
 n\_data = 0  
 **with** open(input\_file, **'rb'**) **as** f:  
 f = csv.reader(f,delimiter = **','**)  
 **for** line **in** f:  
 m\_data += 1  
 n\_data = len(line)  
 vec = [float(i) **for** i **in** line]  
 data.append(vec)  
  
 **print "Input configuration "** + str(m\_data) + **" objects "** + str(n\_data) + **" - dimenion, "** n\_cluster = 0  
 **with** open(centroi\_file,**'rb'**) **as** f:  
 f = csv.reader(f, delimiter=**','**)  
 **for** line **in** f:  
 n\_cluster += 1  
 vec = [float(i) **for** i **in** line]  
 centroid\_data.append(vec)  
  
 **print** (**"Input configuration: "** + str(n\_cluster) + **" clusters"**)  
  
**def** euclide\_dist(a\_list,b\_list):  
 **return** math.sqrt(sum((a\_list[i]-b\_list[i])\*\*2 **for** i **in** range(min(len(a\_list),len(b\_list)))))  
  
  
**def** execute():  
 *#\*\*-------------------Fill in here------------------------\*\*   
 # Repeat for nIteration times:* **global** data, centroid\_data, new\_cluster\_centroid  
 **global** cluster\_id  
 new\_cluster\_centroid=defaultdict(dict)  
 **global** n\_inter, m\_data, n\_data, n\_cluster  
  
 cluster\_id=[]  
  
  
 *# new\_clust\_centroiid[n\_cluster(3)][n\_data(4)=4 dimensions]* **for** a\_iter **in** range(n\_inter):  
 closest=defaultdict(list)  
 **for** x **in** data:  
 dlist=[]  
 **for** s **in** centroid\_data:  
 q=euclide\_dist(x,s)  
 dlist.append(q)  
 minc=dlist.index((min(dlist)))  
 cluster\_id.append(minc)  
 closest[minc].append(x)  
 centroid\_data=[]  
 **for** a1 **in** range(n\_cluster):  
 pp=[]  
 **for** b1 **in** range(n\_data):  
 pp.append(sum(j[b1] **for** j **in** closest[a1])/len(closest[a1]))  
 centroid\_data.append(pp)  
 **for** a **in** range(n\_cluster):  
 **for** b **in** range(n\_data):  
 new\_cluster\_centroid[a][b]=sum(i[b] **for** i **in** closest[a])/len(closest[a])  
  
  
  
  
  
  
  
  
  
  
  
  
 *# Find the minimum distance and assign object to new cluster  
  
 # Calculate new centroid of each cluster by calculating the total sum of all objects in a cluster  
 # then divide it by number of objects in that cluster  
  
 # Update the centroid matrix for next iteration  
 # new\_clust\_centroiid[n\_cluster][d\_data]***def** output():  
 **global** n\_inter, m\_data, n\_data, n\_cluster  
 **global** data, centroid\_data, new\_cluster\_centroid  
 **global** input\_file, centroi\_file, output\_file,cluster\_id  
 **with** open(output\_file,**'w'**) **as** f:  
 f.write(**"New Centroid: \n"**)  
 **for** i **in** range(n\_cluster):  
 f.write(**"Cluster "** + str(i) + **" : "**)  
 **for** j **in** range(n\_data):  
 f.write(str(new\_cluster\_centroid[i][j]) + **' '**)  
 f.write(**'\n'**)  
 f.write(**'\n'**)  
 f.write(**'cluster ID: \n'**)  
 **for** i **in** range(m\_data):  
 f.write(str(cluster\_id[i]) + **' '**)  
  
 f.close()  
  
  
**def** main():  
 arg = sys.argv[1:]  
 **if** len(arg) > 3:  
 arg[2] = int(arg[2])  
 config(arg)  
 **else**:  
 config()  
 execute()  
 output()  
  
main()