COL 761 | DATA MINING
HOME WORK 3

ANIKET SHETTY (2018EE10443)
DIPEN KUMAR (2018CS50098)
UMESH PARMAR (2018CS50424)

Q1) [20 Points]

a) Draw the dendrogram for single linkage clustering on the data below. Show all steps. (5 points):

Point	x	y
1	0.4	0.53
2	0.22	0.38
3	0.35	0.32
4	0.26	0.19
5	0.08	0.41
6	0.45	0.30

⇒ We will start with creating a 2D matrix of pairwise distance of all points.

d(x, y)	1	2	3	4	5	6
1	0.000	0.234	0.216	0.368	0.342	0.235
2	0.234	0.000	0.143	0.194	0.143	0.244
3	0.216	0.143	0.000	0.158	0.285	0.102
4	0.368	0.194	0.158	0.000	0.284	0.220
5	0.342	0.143	0.285	0.284	0.000	0.386
6	0.235	0.244	0.102	0.220	0.386	0.000

⇒ Here dendrogram for single linkage clustering, we will cluster points by merging two different clusters who are closest to each other at each step and continue till we have only one cluster left. Here distance between two clusters is defined as shortest distance between any two points with one point in each cluster. Initially all points are in different separate clusters.

⇒ Shown below are the steps involved in creating the dendrogram

1. Find the closest pair of clusters

d(x, y)	1	2	3	4	5	6
1	0.000	0.234	0.216	0.368	0.342	0.235
2	0.234	0.000	0.143	0.194	0.143	0.244
3	0.216	0.143	0.000	0.158	0.285	0.102
4	0.368	0.194	0.158	0.000	0.284	0.220
5	0.342	0.143	0.285	0.284	0.000	0.386
6	0.235	0.244	0.102	0.220	0.386	0.000

Closest pair is {3} and {6}, hence merge them to get a single cluster {3, 6}

2. Find the closest pair of clusters among remaining set of clusters

d(x, y)	{1}	{2}	{3, 6}	{4}	{5}
{1}	0.000	0.234	0.216	0.368	0.342
{2}	0.234	0.000	0.143	0.194	0.143
{3, 6}	0.216	0.143	0.000	0.158	0.285
{4}	0.368	0.194	0.158	0.000	0.284
{5}	0.342	0.143	0.285	0.284	0.000

Here we have tie between [{2} and {5}] and [{2} and {3, 6}]. In order to reduce height of the tree and prefer clusters of uniform size we will merge {2} and {5} in one to get {2, 5}

3. Find the closest pair of clusters among remaining set of clusters and merge them

d(x, y)	{1}	{2, 5}	{3, 6}	{4}
{1}	0.000	0.234	0.216	0.368
{2, 5}	0.234	0.000	0.143	0.194
{3, 6}	0.216	0.143	0.000	0.158
{4}	0.368	0.194	0.158	0.000

Closest pair is {2, 5} and {3, 6}, hence merge them to get a single cluster {2, 3, 5, 6}

4. Find closest pair of cluster among remaining set of clusters and merge them in one

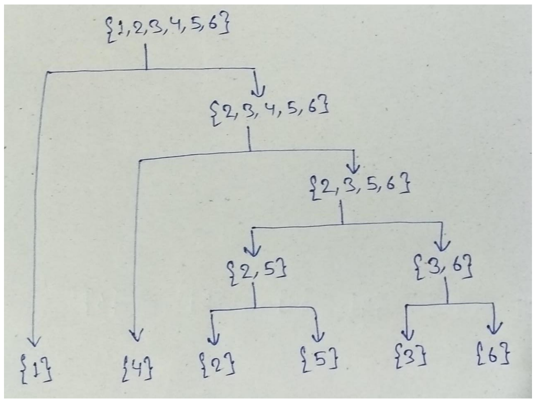
d(x, y)	{1}	{2, 3, 5, 6}	{4}
{1}	0.000	0.216	0.368
{2, 3, 5, 6}	0.216	0.000	0.158
{4}	0.368	0.158	0.000

Closest pair is {2, 3, 5, 6} and {4}, hence merge them to get a single cluster {2, 3, 4, 5, 6}

5. Finally merge the last two pair of clusters

d(x, y)	{1}	{2, 3, 4, 5, 6}
{1}	0.000	0.216
{2, 3, 4, 5, 6}	0.216	0.000

Finally we get all points into a single cluster i.e. {1, 2, 3, 4, 5, 6} completing the dendrogram shown below



b) Time complexity Analysis for single-linkage hierarchical clustering of n points without using the support of any other data structure is shown below

1. Compute pairwise distance between all clusters
[O(n^2) in running time since in order to compute pairwise distance between all clusters we will end up computing pairwise distance between all points except those belonging to same cluster. And since initially all are in different clusters of size 1 implies no two points belong to same cluster implies for n points we get O(n^2) distinct pair. Although we see with progress this time reduces over iterations but if we work its maths out, we find over all on average each iteration still takes O(n^2) time.]
2. Iterate over all pairs of clusters and find closest pair of clusters i.e. minimum single linkage distance between the two clusters. Do linear search
[O(n^2) since O(n^2) different pairs to iterate over during linear search. As already said above in previous step that we see with progress, the number of distinct pair of clusters reduces over iterations but if we work its maths out, we find over all on average each iteration still scan O(n^2) distinct pairs and hence takes O(n^2) time using linear search.]
3. Merge the pair of two clusters in one
[O(n) since need to copy cluster B to cluster A, suppose we are merging A and B. We then delete the cluster B. Both copying and deleting will take time proportional to the size of the cluster B that can be as big as (n-1) points and as small as just 1 point. Hence in worst case it will take O(n) time. This step is not that important to analyze cause it is the fastest among step1 and step2 that will eventually decide the overall running time complexity for single iteration. But for fun we can see that if we end up creating more balanced dendrogram with uniform clusters then we need to do less copying as compared to skewed. Furthermore analyzing both the best and the worst scenario, we see asymptotically one take O(1) time and the other O(n) time on average for each iteration. Hence considering the worst case scenario we say this step will take O(n) time.]
4. Repeat {1, 2, 3} till we have only one cluster left
[Since at each step we reduce the number of clusters by 1 because we merge two in one [-2 +1 = -1]. This implies if we start with "n" number of clusters i.e. number of points each in separate different cluster we have to iterate "n-1" times to merge every points in one cluster with each step reducing the number of clusters by 1. Hence O(n) is the number of iterations our program will run. Each iteration will run above mentioned step1, step2, and step3]

Hence over all time complexity becomes [(O(n^2) + O(n^2) + O(n)) * O(n) = O(n^3)]

c) We here propose the fastest possible algorithm that will run in O(n^2) time

$\Rightarrow$ In the above naive approach we saw that we do (n-1) iterations and in each we reduce the total number of clusters by 1. Hence we obtain one final cluster with all points when started with n clusters each with one point initially. In side this loop that is running from i = 1 to n-1, we are computing pairwise distance between clusters that takes O(n^2) time at each iteration. Therefore the overall complexity of our naive approach was O(n) * O(n^2) = O(n^3). This is important to know because now we will propose a faster algorithm which will save time at each iteration. Instead of spending O(n^2) time at each iteration and computing pairwise distance of clusters each time, We will compute it initially once and at each iteration we will merge the closest pair of clusters in O(n) time and update the new formed super cluster distance with other clusters in O(n) time. Hence, we reduced the time from O(n^2) to O(n) for each iteration and keeping the outer loop as before with O(n) iterations we get over all running time complexity of O(n^2) Pseudocode is written below.

```
# clusters is a dictionary with key = cluster_name and value = points_list.
for i in clusters: #O(n)
      for j in clusters: #O(n)
            # initially all cluster contain only one point and hence size = 1
            d[i,j] = distance(clusters[i][0], clusters[j][0])
            # d[i,j] is distance between cluster i and cluster j
      closest[i] = for j in clusters where j!=i and d[i,j] is minimum #O(n)
      # closest[i] is the cluster closest to cluster i

n = clusters.size()
for step in range(n-1): #O(n)
      # we will merge the closest pair of clusters
      i = for j in clusters where d[j,closest[j]] is minimum #O(n)
      clusters[i].extend(clusters[closest[i]]) #O(n)
      clusters.remove(closest[i])
      print("cluster {closest[i]} merged to cluster {i}")
      # update the new formed super cluster distance with others
      for j in clusters: #O(n)
            d[i,j] = min(d[i,j], d[closest[i],j])
            d[j,i] = min(d[i,j], d[closest[i],j])
      closest[i] = for j in clusters where j!=i and d[i,j] is minimum #O(n)
```

⇒ Time complexity of the above algorithm is straight forward to analyze.
We have two outer loops each with O(n) iteration
Each of these two outer loops have O(n) nested loops
$O(n)*[O(n)+O(n)] + O(n)*[O(n)+O(n)+O(n)+O(n)] = O(n^2)$

Q2 [40 Points]

1. We used graph attention networks.
2. Used MLP to convert our data points from original graph to embedding space which was further passed to GAT network for training.
3. Parameters used in GAT are in_channels = 2*128, hidden_channels = 128, num_layers = 3, out_channels = 1, dropout = 0.0
4. RMSE found with our trained model in testing was found approximately equal to 0.005, on multiple run we saw as high as 0.007
5. We are assuming following python packages been installed on the machine that would be running our code as HPC didn't had available modules for these. We installed on our user on HPC
   a. pip install torch-scatter -f https://pytorch-geometric.com/whl/torch-${TORCH}+${CUDA}.html
   b. pip install torch-sparse -f https://pytorch-geometric.com/whl/torch-${TORCH}+${CUDA}.html
   c. pip install torch-cluster -f https://pytorch-geometric.com/whl/torch-${TORCH}+${CUDA}.html
   d. pip install torch-spline-conv -f https://pytorch-geometric.com/whl/torch-${TORCH}+${CUDA}.html
   e. pip install torch-geometric -f https://pytorch-geometric.com/whl/torch-${TORCH}+${CUDA}.html