Assignment 3 Report

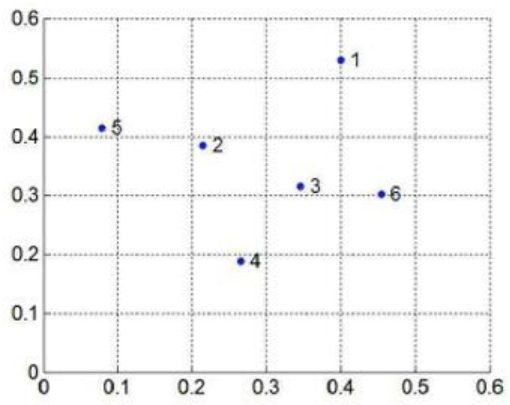
Shreyans J Nagori - 2018CS10390

Rajat Singh - 2020CSZ8507

Chhavi Agarwal - 2020CSY7654

Q1 Part a) Draw the dendrogram for single linkage clustering on the data below.

Point	x	y
1	0.40	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



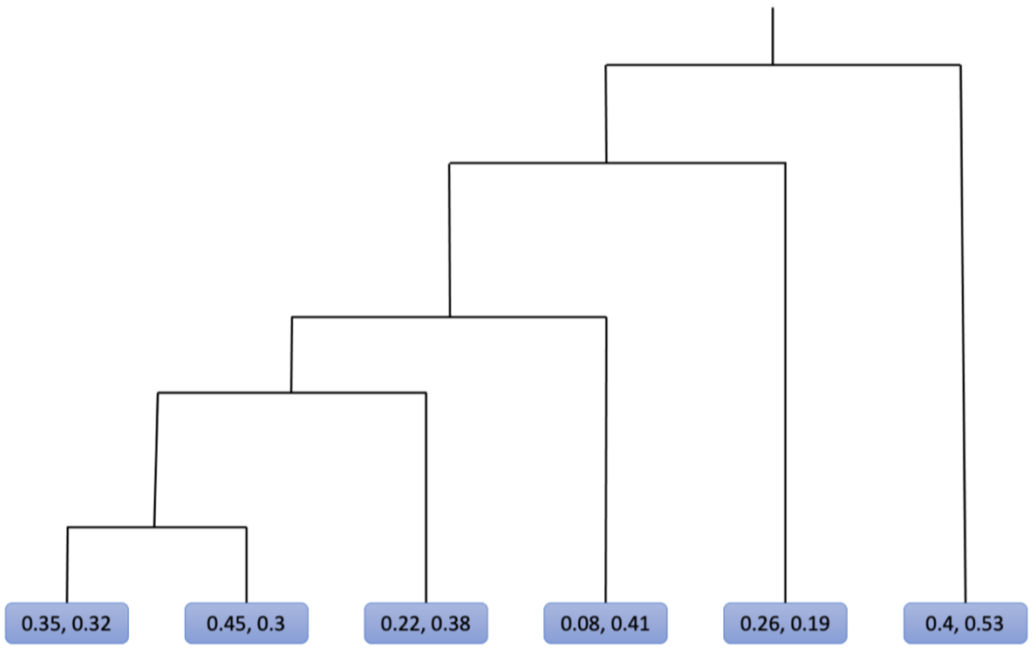


Fig 1: dendrogram

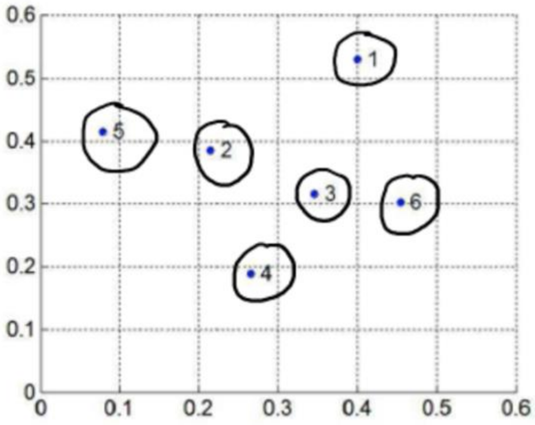


Fig 2: Step 1

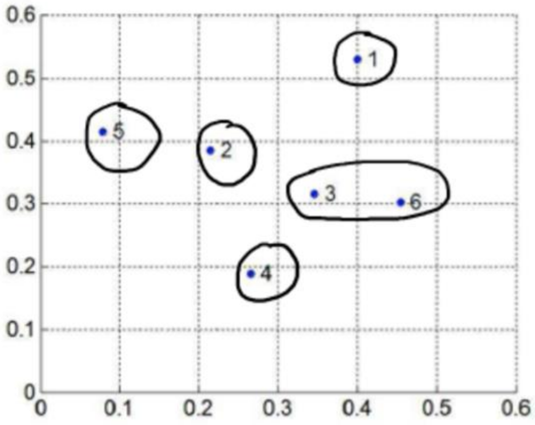


Fig 3: Step 2 Fig 3: Step 2

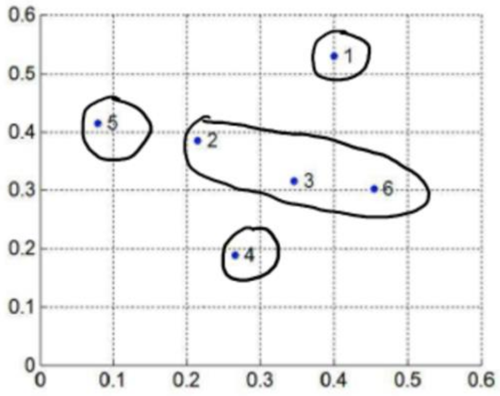


Fig 4: Step 3

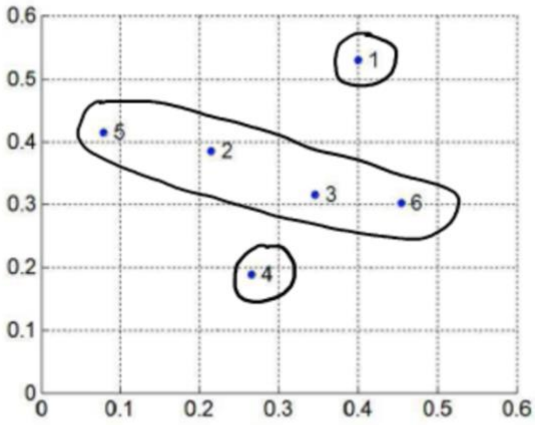


Fig 5: Step 4 Fig 5: Step 4

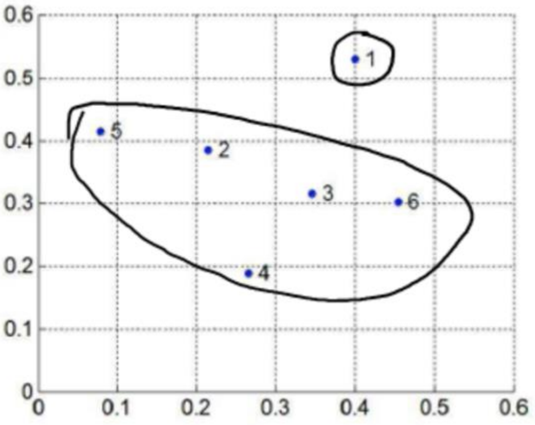


Fig 6: Step 5

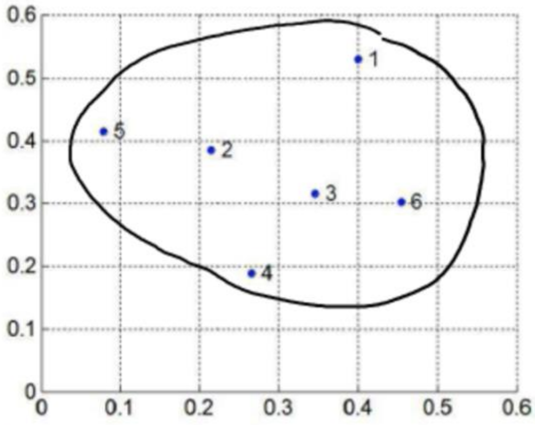


Fig 7: Step 6 Fig 7: Step 6

Step 1: Each node act like a single cluster initially

Step 2: Single-linkage distance between cluster {3} and {6} is least so merge these two clusters to a single cluster.

Step 3: Single-linkage distance between cluster {2} and {3,6} is least so merge these two clusters to a single cluster.

Note: distance between cluster {2} and {3,6} and distance between cluster {5} and {2} are same so we randomly choose one pair of clusters and merge them.

Step 4: Single-linkage distance between cluster {5} and {3,6,2} is least so merge these two clusters to a single cluster.

Step 5: Single-linkage distance between cluster {4} and {3,6,2,5} is least so merge these two clusters to a single cluster.

Step 6: Single-linkage distance between cluster {1} and {3,6,2,5,4} is least so merge these two clusters to a single cluster.

We can draw the above dendrogram in another way also as shown in the Fig 8.

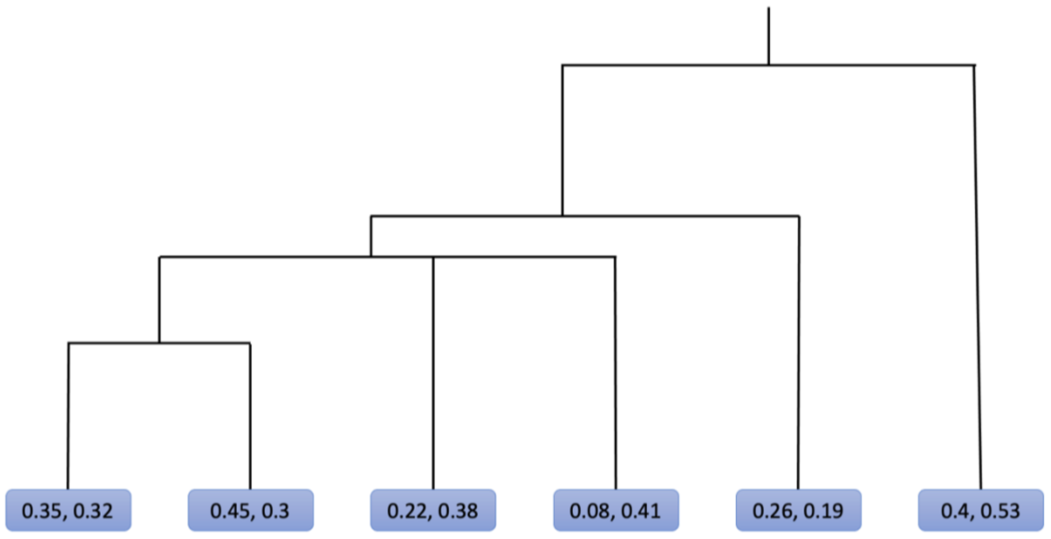


Fig 8: Dendrogram by representing the distance between clusters as the height of dendrogram

Q1 Part b) What is the computation complexity of single-linkage hierarchical clustering of n points without using the support of any other data structure? Explain your reasoning.

Ans: Since we are not going to use any data structure except one data structure i.e., to store the clusters. Thus:

1. We must perform $O(n^2)$ computation for find the shortest distance among all the clusters.
2. Among all the clusters we merge one cluster with smallest cluster distance. Now again we must perform step 1 with new clusters until only one cluster is formed.

Thus, the complexity of this algorithm is $O(n*n^2)$ as we are going to perform step 1 for n times.

Q1 Part c) What is the complexity of the fastest possible algorithm? Give pseudocode of your algorithm and its complexity analysis.

Ans: Q1.c. The time complexity of fastest single linkage clustering algorithm [1] is O(nB) where n is the number of data points in the dataset and B is the number of clusters or data points that goes into single hash entry.

This algorithm uses Locality-Sensitive Hashing to fasten the process of merging clusters.

It is executed in multiple layers; it takes as input 'r' which is the threshold. If the distance between any two clusters is less than r, then those clusters can be merged. The value of r keeps increasing for the next layers.

To find the approximate nearest neighbours quicker, we use LSH. In each layer 'l' hash functions are chosen, where each hash function has k random bits. If a data point is d dimension with max value of each dimension as C, then each data point x(x1, x2, x3 ,.. , xd) can be represented as Unary(x1) Unary(x2) … Unary(xd).

Unary(x1) = x1 sequence of 1 followed by C-x1 sequence of 0s.

Unary(x) will be Unary(x1) followed by Unary(x2) and so on.

LSH hash function thus will be any k random bits chosen between 1 and Cd. Corresponding hash table for all data points can be computed for that hash function by concatenation corresponding k bits of the data point whose index are mentioned in the hash function.

In each layer 'l' hash functions are chosen, where each hash function has k random bits. As we keep increasing 'r' per layers, simultaneously we keep decreasing 'k'.

Nearest neighbour search for any point q:

- 'l' hash values are computed for the data point q.
- For all the data points in dataset if m_i be the set of points in the same bucket as q for same hash function, then all the data points in all 'l' hash functions that are in the same bucket as q(m_1 U m_2 U.. m_l), will become the candidates for its neighbour. Compute the actual distance between q and these data points, whoever has distance less than the threshold 'r' will be the neighbours of q.

Pseudo Code:

1. Compute hash table by computing hash values for all data points on all ‘l’ hash functions. The data point p will be stored in bucket with index $h_i(p)$ where h_i is the i^{th} hash function. If any data point of same cluster has the same index value for that hash function, then it does not get stored in the bucket. This prevents from performance degrading.
2. For each data point, find all its l buckets and compute its distance from all the data points that are also present in those buckets. Now, find those points whose distance is less than ‘r’.
3. Merge the clusters obtained in previous point.
4. If the number of clusters remain is 1, then terminate the algorithm. Else change the value of r to Ar, where A>1, and decrease the value of k to k/A. A can be set according to the granularity of the clustering needed. For large values clustering will be faster and more points will be merged in 1 step.
5. Generate new ‘l’ hash functions with this new value of ‘k’ and return to step 1.

If ‘l’, I.e., the number of hash functions is too small, then it will not cover all the points within ‘r’ distance in the same bucket. Whereas if ‘l’ is too large, then it incurs extra overhead. It is chosen using L1 distance such that all the all the pairs within r distance(L1) are in the same bucket. It approximates for all the further L2 or more distances.

Analysis of Time Complexity:

The time complexity of this single linkage clustering algorithm is O(nB) where B is the maximum number of points that are part of a single hash bucket during the execution of the algorithm. Let s be the layers in which the program converges to one single cluster.

Following operations are performed in the algorithm:

- Computation of hash table
- Search clusters that can be merged
- Merge clusters

Computation of hash table: For each data point, l hash functions need to be computed. For each hash function, k bits need to be computed. There are total n data points, so this step will take O(nlk) time. But at each level k decreases by 1/A as mentioned in algorithm. So total time taken in s layers is:

$O(nlk + nlk/A + ... + nlk/A^{s-1}) \leq O(\Sigma_{i=0 \text{ to } \infty} nlk/A^i) \leq O(Anlk/A-1)$

Here ‘l’, ‘k’ and ‘A' are constants independent of n. Hence the complexity of this step is **O(n)**

Search clusters that can be merged: At most B data points are there in the bucket of a hash function and there are ‘l’ hash functions. So, for each data point O(lB) time will be taken to compute the distance of that data point from its potential neighbours. So, per phase it will take O(nlB) time.

Since, there are s phases, time complexity of this step is O(snlB). Here l is constant so that can be ignored. It can also be proved that s will be some constant independent of n. If r_{max} is the maximum distance between any 2 data points, then at max ‘r’ can be r_{max} before it terminates. So, $A^{s-1}R \geq r_{max}$ ’s’ can be written as $s \leq 1 + \log_A(r_{max}/r)$. Here r_{max} is independent of the number of data points. Hence ‘s’ is also a constant independent of n. Hence time taken in this step is **O(nB)**.

Merge clusters: At the end all data points need to be merged into one single cluster. Maximum clusters that can be merged is O(nlB). So for s phases, total time taken at this step is O(snlB). Again as it is proved earlier that 's' and 'l' can be ignored that they are constants independent of n. Hence time complexity is **O(nB)**.

So, time complexity of the algorithm is O(n + nB + nB) = **O(nB)**

Q2) Using GNN models from pytorch for computing RWR distance on CORA Dataset:

Firstly, we make one observation, let P be the transition matrix of random walk for a graph G. Then for random walks of length 4 with fixed restart, we can find out probability of being in node v after starting out in node u averaged across time by taking the v^{th} element of $T*I_u$, where $T= (I+P+P^2+P^3+P^4)/5$, is averaged transition matrix across 5 epochs of the random walk (as the random walk is periodic with period 5), and I_u is the row vector with value at index u =1 and other indices 0. Using this transition matrix, we sample out for entire training dataset, and entire validation dataset. For testing dataset, we sample out as mentioned in assignment description to maintain fairness.

Here we use 2 GAT layers, one with in_channels =1 and out_channels =250 and other with both in_channels and out_channels are set to 250. Then we take the pair list of vertices for which RWR distances is calculated, and then concatenate them. Finally, we pass the concatenated output through a MLP to obtain the required RWR distance.

We tested out various sampling strategies and found that our model is scaling for all pairs from training nodes and giving the best score on that.

The loss we get on test data is 9.27845e-05.

Time taken to train the model is approximately 8 mins.

We have used MSE (Mean Square error) as the loss function for our model.

The parameters for which we get the best results are follows:

- Number of train per class = 150
- First GATv2Conv layer (-1*250)
- Second GATv2Conv layer (250*250)
- Learning rate = 0.0001
- Weight decay=1e-5
- Number of epoch = 10

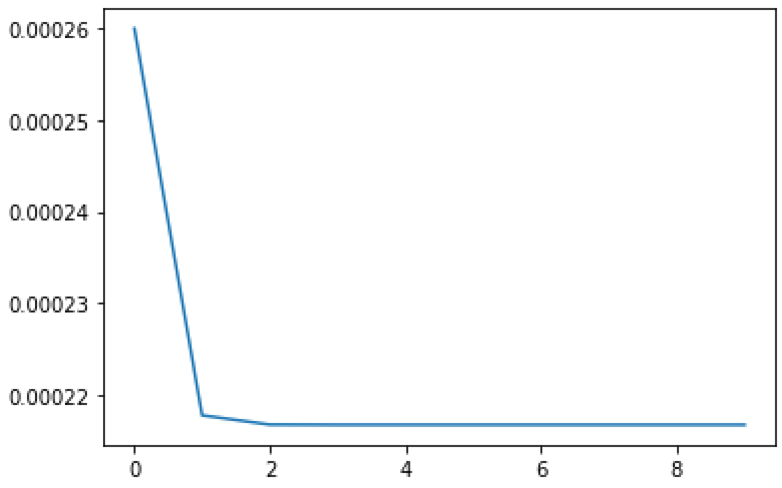


Fig 9: plot of number of epoch (x-axis) vs validation loss

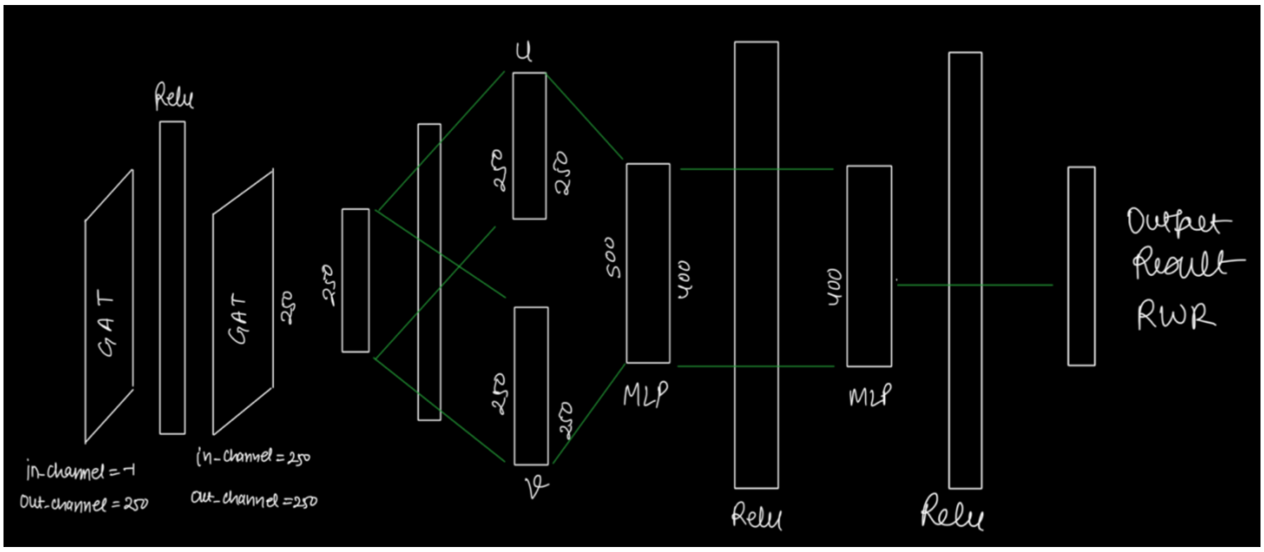


Fig 10: Architecture of the model.

References:

1. Koga, H., Ishibashi, T. & Watanabe, T. Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing. *Knowl Inf Syst* 12, 25–53 (2007).