# C3BA25 : Hierarchical Clustering-Agglomeration

IIT2018177,IIT2018178,IIT2018179
*IV Semester, B.Tech IT*
*Indian Institute of Information Technology, Allahabad*

***Abstract : In this paper we have devised an algorithm for Agglomerative Hierarchical Clustering.***
***Also, we have discussed the time and space complexity of the Algorithm using both Apriori and Aposteriori analysis.***
***IIT2018177 contributed to writing the report, he had done apriori and aposteriori analysis in the report as well as algorithm description, also he determined which algorithm should be followed for coding.***
***IIT2018178 contribution consists of designing the code to follow the suggested algorithm. He also contributed to the report by writing the most of the proposed model in the report.***
***IIT2018179 contribution consists of helping IIT2018178 in design of code and has done Introduction, keywords and conclusion in this report. IIT2018179 also helped in writing the proposed model in the report.***

## I. KEYWORDS

Cluster : An individual point or a group of points.

Linkage : Algorithms which are used in calculating distance matrix in our agglomerative clustering algorithm.

## II. INTRODUCTION

Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters.

There are two types of Hierarchical clustering :
Agglomerative Clustering
Divisive Clustering
Since, we will be working with only Agglomerative clustering, we will introduce this algorithm here.
In an agglomerative hierarchical clustering algorithm each individual element is supposed to be a cluster. These clusters are merged iteratively until all the elements belong to one cluster.
These algorithms will be used in calculating distance matrix in our agglomerative clustering algorithm.
There are several type of linkages :

Single Linking : We take the closest clusters and link those clusters and then recompute

the distance between the newly created cluster and the remaining clusters. Also, the clusters that form the new clusters are removed from the distance matrix.
For distance between two clusters that contain more than one element, the minimum distance between the points in the cluster is chosen.

$$L(R,S) = min(D(i,j)), i\epsilon R, j\epsilon S$$

Complete Linkage : For two clusters R and S, the single linkage returns the maximum distance between two points i and j such that i belongs to R and j belongs to S.

$$L(R,S) = max(D(i,j)), i\epsilon R, j\epsilon S$$

Average Linkage : For two clusters R and S, first for the distance between any data-point i in R and any data-point j in S and then the arithmetic mean of these distances are calculated. Average Linkage returns this value of the arithmetic mean.

$$L(R,S) = \frac{1}{n_R+n_S}\sum_{i=1}^{n_R}\sum_{j=1}^{n_S} D(i,j), i\epsilon R, j\epsilon S$$

Ward : This approach uses a minimised sum of squared differences between two clusters.

## III. ALGORITHM DESCRIPTION

In this algorithm, we first take the number of points the user wants to enter. Next, we consider each of these points itself to be an individual cluster and calculate the distance between each of these clusters and store it in a matrix called “Distance”. The algorithms we use to calculate distance are commonly referred to as linkages which are available for merging these clusters.

We will be implementing the single linkage approach in our algorithm.

As soon as we compute our initial distance matrix we move on the recursive algorithm. Firstly, we choose two clusters which are nearest to each other, and merge them. Secondly, we update our distance matrix, using the single linkage approach. The above two steps are repeated until we are left with only one cluster. At the end we graph this cluster, which is commonly referred to as a dendrogram.

## IV. PROPOSED MODEL

STEP 1 : We input the number of points and then the points itself. Also,we create a list to label these input points starting from 0 till 1 less than the number of points.
Here,
nosOfPoints : number of points the user wants to provide.
matrixofPoints : This list stores coordinates.
pointID : we store an ID which we use to refer to the coordinates stored in matrixofPoints

```
nosOfPoints = input()
matrixOfPoints = list()
pointID = list()
for i in range(nosOfPoints):
        temp = list(input(point1,point2))
        matrixOfPoints.append(temp)
        pointID.append(i)
```

STEP 2 : Next, we create our initial distance matrix that stores distance between all the points.
distanceMatrix = stores distance between each cluster.
sys.maxsize = Maximum size for an integer.(We use this for initialising distance between a cluster and itself)
dist : used as a temporary variable that stores distance between two clusters before value inserted in distanceMatrix

```
distanceMatrix = [0]

for i in range(nosOfPoints):
    for j in range(nosOfPoints):
        dist =
```

$$\sqrt{(point1(x) - point2(x))^2 + (point1(y) - point2(y))^2}$$

```
        if i == j:
            dist = sys.maxsize
        distanceMatrix[i][j] = dist
```

STEP 3 : We now find the closest cluster and combine those clusters.Here the minimumDistance function returns the i'th and j'th value of the least distance cell. These indices will fetch us cluster 1 and 2 respectively. We combine and store the cluster in clusteredPoints.
cluster1,cluster2 : stores the ID's of the coordinates with the shortest distance.
tempList : Is a temporary list that stores the remaining cluster and the new cluster formed.

```
cluster1,cluster2 =
minimumDistance(distanceMatrix)
clusterList = list[cluster1,cluster2]
clusteredPoints =
combine(pointID,clusterList)

combine :
tempList = []
   for i in range(pointID):
       if(i not equal to cluster1 or cluster2)
           tempList.append(i)
tempList.append(clusterList)
return tempList
```

STEP 4 : Now we can follow the algorithm given below until we are left with only one cluster. We use the new list of points(clusteredPoints) to create a distance matrix using the single linkage approach, then choose the clusters with minimum distance. And, then we update our clusteredPoints list by deleting the two clusters obtained and add the new cluster. (obtained by combining the two old clusters.) We keep on doing this until we are left with only one cluster.
cluster1,cluster2 :It is similar to our previous cluster1,cluster2, the only difference is that these can be of type list now.
computeDistanceMatrix : This function now computes minimum distance between the new clustered points.
poc1,poc2 : They are abbreviated as points of cluster 1 and 2 respectively, they display that these are two cluster lists which may have more than one element in them.
The formula in computeDistMatrix tells us to choose the distance between two clusters

by taking the minimum distance possible between the points in those two clusters.

```
Length = length(clusteredPoints)
while(Length is not 1){

distanceMatrix =
computeDistMatrix(clusteredPoints)

cluster1,cluster2 =
minimumDistance(distanceMatrix)

clusterList = list[cluster1,cluster2]
clusteredPoints =
combine(clusteredPoints,clusterList)
Length = length(clusteredPoints)
}
```

```
computeDistMatrix :
    for i in range(clusteredPoints)
        for j in range(clusteredPoints):
            dist =
min(√((poc1(x) − poc2(x))² + (poc1(y) − poc2(y))²))
            if i == j:
              dist = sys.maxsize
            distMatrix = dist
```

## V. PRIORI ANALYSIS

To store all points, there is a 'for' loop with range 0 to number of Points and thus time complexity for this loop is $O(n)$.

Now, for calculating distance between all pairs of point we used two nested 'for' loops with range again equal to 0 to number of Points, thus time complexity for this would be $O(n^2) + O(n)$.

Now a 'while' loop is used for making clusters of all data points. In the worst case, on every iteration of the 'while' loop the smallest cluster possible is made and the number of clusters get reduced by 1 unit, thus time complexity for this 'while' loop is $O(n)$.

Inside this 'while' loop, we need to update the distance matrix because two clusters of all clusters available are combined and become a cluster. We need to make a distance matrix between the new clustered points and considering the number of clustered points to be k (k <= n as points combine to form clusters the number of available clusters decrease), the time complexity of this function would be $O(nk^2)$. But, even in clusters we need to compare the points to find minimum distance. Thus, at the end the time complexity of this function turns out to be $O(n^3)$.

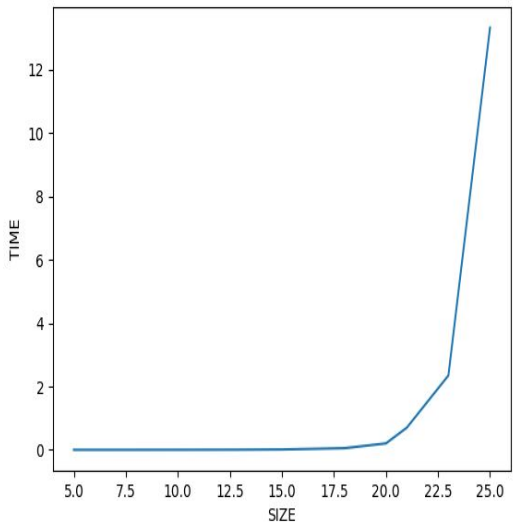
Again inside the "while" loop, a function is called to find the two clusters that form the new cluster. These functions linearly search for the new clusters, thus having time complexity of $O(n^2)$.

So final time complexity is $O(n^3) + O(n^2) + O(n)$. nFor a larger value of n, the time complexity is assumed to be equivalent to $O(n^3)$.

## VI. POSTERIORI ANALYSIS

| INPUT | TIME |
|---|---|
| 5 | 0.000127053 |
| 10 | 0.001099872 |
| 20 | 0.203758883 |
| 25 | 13.33288073 |

For time vs number of points the graph :




In the above graph, we notice that the graph takes the form of $O(n^3)$. Thus, our apriori and aposteriori analysis match.

## VII. CONCLUSION

From this paper, we can conclude that agglomerative hierarchical algorithm requires $O(n^3)$ time to cluster all the points given to them. Clustering is a mathematical tool that attempts to discover structures or certain patterns in a dataset, where the objects inside each cluster show a certain degree of similarity. The algorithm used here is the same algorithm implemented in scikit learn, though it would be implemented in a better manner.

## VIII. REFERENCES


Article on Agglomerative Hierarchical clustering
Geeks for Geeks article
Wikipedia page
Research paper on hierarchical Clustering

## IX. ANNEXURE-I

We would suggest to copy-paste this code in a sublime text editor so as to save the indentation, otherwise it might not run.
To run this python program the following module and libraries should be installed in your system:-

**Python3 :** As we have developed this program in python3 platform. It might happen that some modules and functions may not be present in lower or higher versions of python so it is recommended to use python3 for interpreting this program. It can be downloaded using command
sudo apt-get install python3

**Pip3 :** It has to be installed in your system to download the modules given below. It is not used in the program. It can be downloaded by command
sudo apt-get install python3-pip.

**Matplotlib :** This is used to plot a graph of the number of input vs time required to calculate cluster using agglomerative method in second. This module has to be installed with the command given below.
pip3 install matplotlib

**Numpy :** It stands for Numerical Python.It is used for efficient operation on homogeneous data that are stored in arrays.It can be downloaded using command
pip3 install numpy.

**Scipy :** It stands for Scientific python.it is actually a collection of tools for Python. These tools support operations like integration, differentiation, gradient optimization, and much more.It can be downloaded using command, here it is used with plotly for getting creating dendrogram.
pip3 install scipy

**Plotly :** It enables Python users to create beautiful interactive web-based visualizations.It can be downloaded using command.
pip3 install plotly

After installing all above modules, save the file below with .py extension, open terminal and navigate to the directory in which you have saved the file and run the command
python3 <filename>.py .

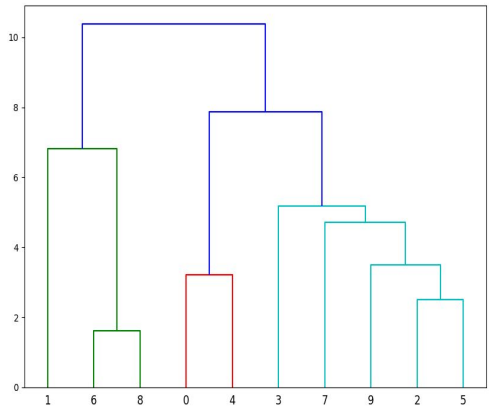
Considering a sample run, let the filename be daac3.py that is saved in home directory.
We run the command :
python3 daac3.py
Next, we input the nos of points, and in the following lines we input the points.

Input :

10
17.594379084664382 9.869760100329765
4.482120813255356 18.827252397647683
6.134613086988093 1.14405518572392
11.214267792385881 2.19200350975028
17.70296694072032 6.662845180815913
4.768409730231438 3.2363637603569835
11.272454306812657 18.254652652835233
2.1127625710418068 7.127341177648455

12.69426947413327 19.026590698994735
6.9285990412480025 5.9950599278259675

We have our dendrogram as output.




```
import math
import sys
import numpy as np
from scipy.cluster.hierarchy import
dendrogram, linkage
from matplotlib import pyplot as plt
import plotly.figure_factory as ff

#Function to find minimum arguments of a
2D array
def returnminArgs(arr, pointsID):
        minD = sys.maxsize
        for i in range(len(arr)):
                for j in range(i):
                        if arr[i][j] < minD:
                                minD =
arr[i][j]
                                argX =
pointsID[i]
                                argY =
pointsID[j]
        return argX, argY

#Function to combine a pointsID and
minArgs array to form a clustered array
def combine(pointsID,minArgs):
        clusteredPoints = list(pointsID)

        for elements in minArgs:

clusteredPoints.remove(elements)

        clusteredPoints.append(minArgs)
        return clusteredPoints

def returnSimpleList(compList, simpleList):
        for elements in compList:
                if isinstance(elements,list):

simpleList.extend(returnSimpleList(element
s,simpleList))
                else:

simpleList.append(elements)
        return simpleList

#input
nosOfPoints = int(input())
matrixOfPoints = list()
pointID = list()
for i in range(nosOfPoints):
        temp = list(map(float,input().split()))
        matrixOfPoints.append(temp)
        pointID.append(i)

#initialization of basic distance between two
IDs
distanceBetweenPoints = [ [0]*nosOfPoints
for i in range(nosOfPoints) ]
```

```
for i in range(nosOfPoints):
        for j in range(nosOfPoints):
                distanceBetweenPoints[i][j] =
math.sqrt((matrixOfPoints[i][0]-matrixOfPo
ints[j][0])**2+(matrixOfPoints[i][1]-matrix
OfPoints[j][1])**2)

clusteredPoints = pointID

while len(clusteredPoints) != 1:
        #Initialization of distance matrix
        distMatrix = [
[0]*len(clusteredPoints) for i in
range(len(clusteredPoints)) ]
        for i in range(len(clusteredPoints)):
                distMatrix[i][i] = 0



        for i in range(len(clusteredPoints)):
                for j in
range(len(clusteredPoints)):
                        if i==j:
                                continue
                        yAxis =
[clusteredPoints[j]]
                        xAxis =
[clusteredPoints[i]]
                        distMin = sys.maxsize
                        if
isinstance(clusteredPoints[i],list):
                                xAxis =
returnSimpleList(clusteredPoints[i],list())
                        if
isinstance(clusteredPoints[j],list):
                                yAxis =
returnSimpleList(clusteredPoints[j],list())
                        for elementsX in
xAxis:
                                for elementsY
in yAxis:
                                        if
distMin >
distanceBetweenPoints[elementsX][element
sY]:

distMin =
distanceBetweenPoints[elementsX][element
sY]
                                distMatrix[i][j] =
distMin


        minArgs =
list(returnminArgs(distMatrix,clusteredPoint
s))
        clusteredPoints =
combine(clusteredPoints,minArgs)

X = np.array(matrixOfPoints)

linked = linkage(X, 'single')

labelList = range(1, 11)

plt.figure(figsize=(10, 7))
dendrogram(linked,
        orientation='top',
        distance_sort='ascending',
        show_leaf_counts=True)
plt.show()
```