

**NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY**

(AN AUTONOMOUS INSTITUTION AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM)

**Data Mining(17IS62)**

**6th semester**

**LA - 2**

Programming Assignment on:

**“Agglomerative clustering algorithm”**

Submitted To:

Mr. Aditya Shastry

Submitted By:

D Jagath Singh

(USN:1NT17IS042, Sec:A)

Code:

# -\*- coding: utf-8 -\*-

"""

Created on Tue Apr 7 22:45:01 2020

@author: Jagath

"""

#for faster arrays compared to list

import numpy as np

#for dataframes

import pandas as pd

#for pattern detection

import re

#for masking the lower triangular matrix

from numpy import nan

#the built in min() doesn't allow float values. So, user-defined to find min(list)

def min1(l):

min = 99999

for i in range(len(l)):

if(l[i]<min):

min = l[i]

return min

#input the number of nodes in n

n = int(input("Enter the no. of points"))

#read the distance matrix

#NOTE:MATRIX MUST BE EITHER UPPER TRIANGULAR OR COMPLETE. DON'T INPUT LOWER TRIANGULAR.

print("Enter the distance matrix")

oarr = np.zeros((n,n),dtype = 'float32')

for i in range(n):

oarr[i] = input().split()

#a list l that holds the groups/clusters formed in each iteration

#initially, it is given values starting from a.

#NOTE : MY ALGORITHM WON'T SUPPORT A HUGE NUMBER OF NODES, SINCE THE LABELS ARE LIMITED TO ASCII LIMIT.(255 NODES ONLY) AFTER THAT THE NODE LABELS WILL REPEAT AND CAUSE CONFUSION

l = []

for i in range(n):

l.append(chr(ord('a')+i))

#copy of the labels list to be used later to check working of algorithm, using scikit-learn to verify answer.

labels = l.copy()

#convert the ndarray to Dataframe for easier visualization

odf = pd.DataFrame(oarr,index = l, columns = l)

#a copy of the initial dataframe. This will changed according to the clustering algo. iterations

df = odf.copy()

#a Queue which is used to keep track of the order of grouping/clustering

queue = []

#print the dataframe corresponding to the input given

print('data frame is:')

print(df)

#This function represents one iteration of the clustering algorithm

def myMinAggClustering(df):

#initialize min(to find min value in dataframe) to a high number

min = 99999

#minr and minc will hold the row and column label of the min value

minr = 'a'

minc = 'a'

#mask is a boolean matrix which is true only for upper triangular values

m = df.shape[0]

r = np.arange(m)

mask = r[:,None] < r

#use df values only where mask is True

df = df.where(mask)

#iterate through the dataframe to get the minimum value, store the lables in minr and minc

for i in df.index:

for j in df.columns:

if(min > df[j][i]):

min = df[j][i]

minr = i

minc = j

#edge holds the nodes that are grouped in the current iteration

edge = '('+minr+','+minc+')'

#insert into queue the group.

queue.insert(0,edge)

#remove the two nodes that were grouped from the list 'l', since they were grouped.

l.remove(minr)

l.remove(minc)

#push the grouped node to queue

l.insert(0,edge)

#a new dataframe which will be returned to be used for the next iteration.

ndf = pd.DataFrame(columns = l, index = l)

#loop to calculate the new distance matrix after grouping

for i in ndf.index:

for j in ndf.columns:

#holds the list of nodes in the LHS

fromlist = []

#List of nodes in the RHS

tolist = []

#Holds all the distances between fromlist and tolist, we need to put this in our min1()

minlist = []

#eg: if i was (d,f), then fromlist will have ['d','f']

fromlist = list("".join(re.split("[^a-zA-Z]\*",i)))

#similar to fromlist, only with j rather than i

tolist = list("".join(re.split("[^a-zA-Z]\*",j)))

#iterate through the fromlist and tolist, and calculate minlist.

for a in fromlist:

for b in tolist:

#if statement to make sure that values are taken from upper triangular matrix only..

if(a<b):

minlist.append(odf[b][a])

else:

minlist.append(odf[a][b])

#res has the least of the minlist.

res = min1(minlist)

ndf[j][i] = res

return ndf

#I iterate minAggClustering n-2 times, since we need n-2 iterations to get the dataframe to have only 2 rows and 2 columns from n rows n columns

for k in range(n-2):

df = myMinAggClustering(df)

print('data frame is:')

print(df)

#push the last grouping into the queue

queue.insert(0,'('+l[0]+','+l[1]+')')

#print the order we grouped our nodes.

print('The order of grouping is:')

for u in range(len(queue)):

print(queue.pop())

#Do the same thing using scipy

import scipy.cluster.hierarchy as sch

import numpy as np

import sys

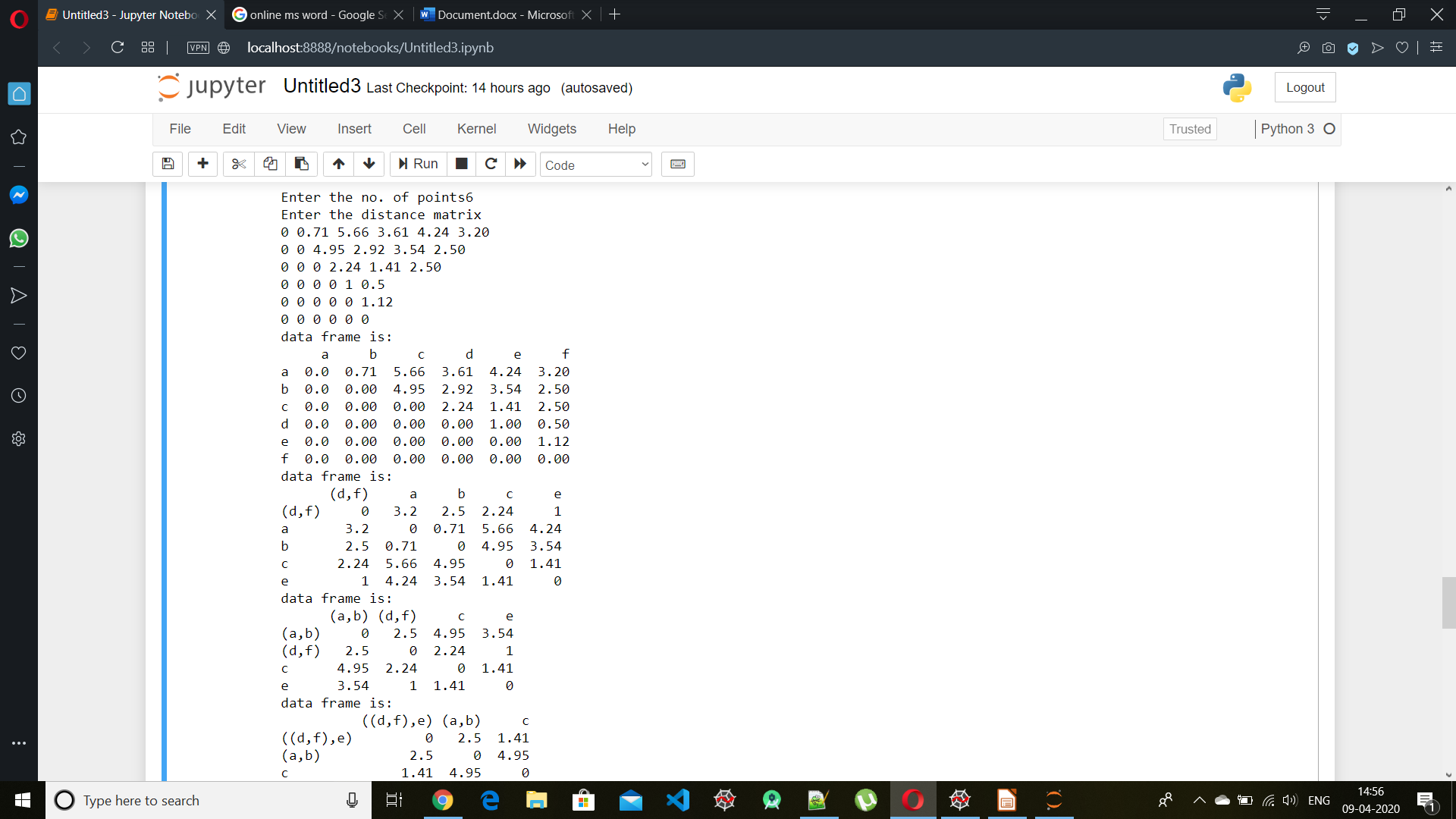
#calculate lincage matrix

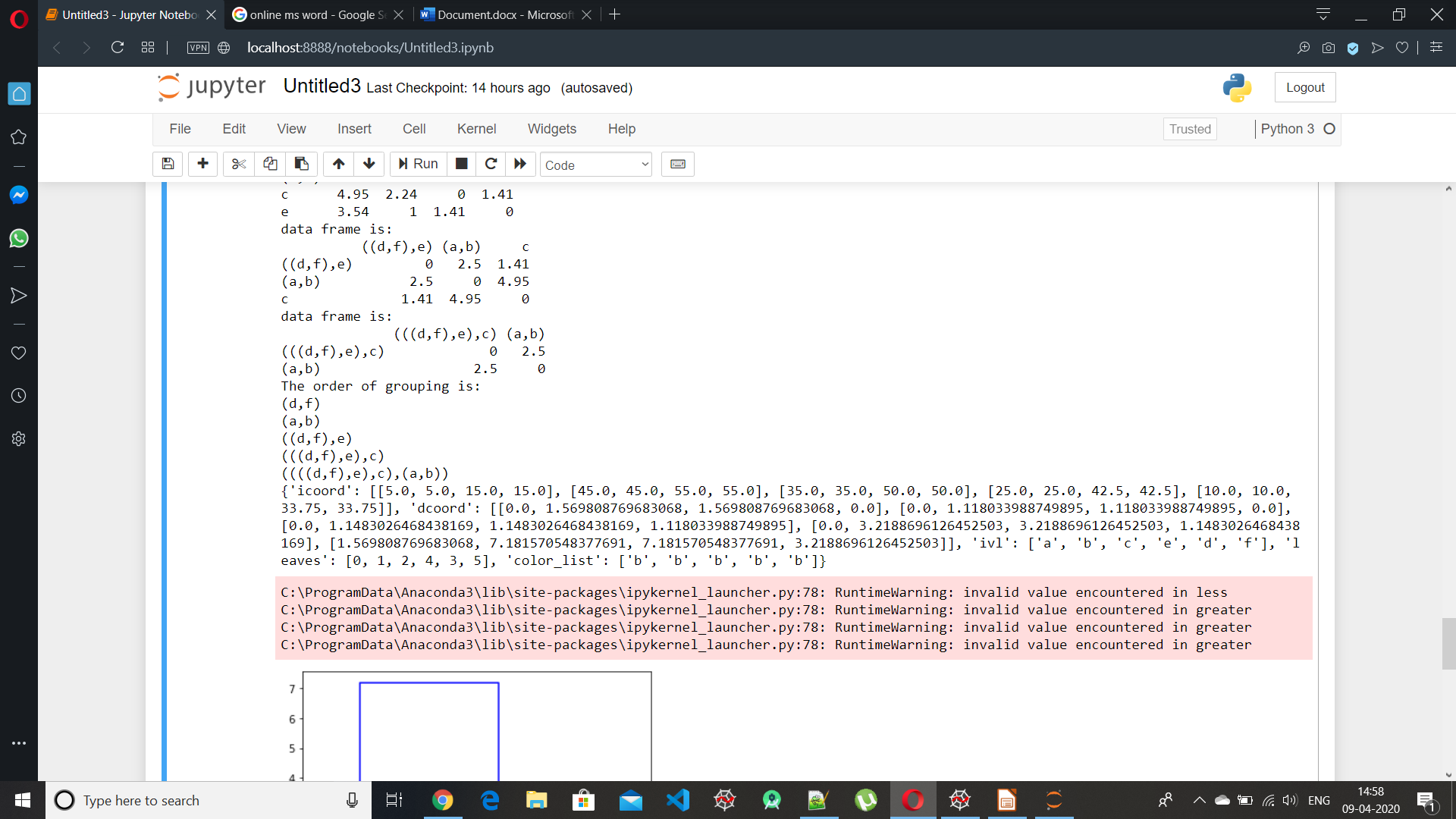
Z=sch.linkage(oarr,'average')

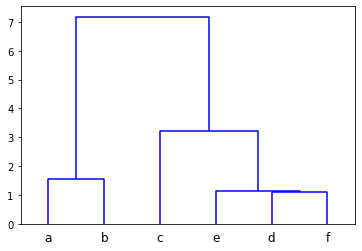
#plot the dendogram

print(sch.dendrogram(Z,labels = labels, color\_threshold=1,show\_leaf\_counts=True))

Sample output:







Observing the dendrogram, it is clear that the output we obtained in our code is accurate.

Algorithm for my code:

Input – The distance matrix of all the nodes to be clustered – oarr[][]

Input – The total number of nodes

make two dataframes – odf[][] and df[][] using oarr[][]

make a list l which has all the labels of nodes

initialize an empty queue - queue[] which will be used to keep in track the groupings

Repeat( n - 2 times ):

{ mask/hide the lower triangular part of df

find the minimum element in df

store the corresponding row label and column label of the min element in minr and minc

from l, remove minr and minc

push (minr,minc) into the queue and also into l

create a new dataframe - ndf - which has index(row names) and columns(column names)same as in l.

calculate ndf:

for i in ndf.index:

for j in ndf.columns:

fromlist = [] - holds all the nodes present in the row index of dataframe

tolist = [] - holds all the nodes present in the column index of dataframe

minlist = [] - Holds all the distances between fromlist and tolist.

eg: if i was (d,f), then fromlist will have ['d','f']

fromlist = all the labels present in i

tolist = all the labels present in j

for a in fromlist:

for b in tolist:

#if statement to make sure that values are taken from upper triangular matrix only..

upper triangular matrix => the row index must be lesser than the column index

if(a<b):

minlist.append(odf[b][a])

else:

minlist.append(odf[a][b])

res = min1(minlist) - has the least of the minlist.

ndf[j][i] = res

}

Notes

This code supports unique labels to nodes only up to 255 nodes. After that, the labels repeat because of the limitation of the number of ASCII characters available.

The input distance matrix given must be an upper triangular matrix or a complete matrix. Don’t give lower triangular matrix as input.