**IE 529**

**COMPUTATIONAL ASSIGNMENT 2**

**Karthik Venkata**

**NET ID-kvn3**

**Question 1: Lloyd’s Algorithm**

1. Output clustering results for both sets of data
2. Result evaluation for a range of k values-

We could identify different types of clusters and depending on how many natural clusters, identifying them based on the value of k. In the case of ShapedData.csv, no clustering based on natural clustering was observed. In spite of varying the values of k from 2, the number of apparent clusters being distinguished are varying but the number of natural clusters still remain the same.

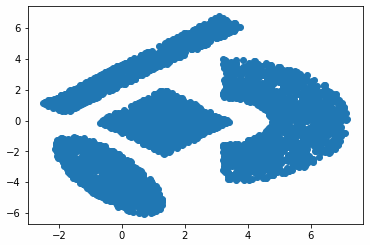
1. For the K-means algorithm, how was convergence tested for and why was this chosen as a test.

The convergence was tested based on the distance function which evaluated the average distance from the cluster centres and as the different values of k were used and the iterations performed, one could test for the convergence criteria.

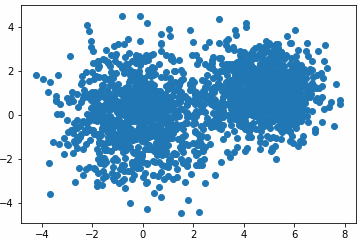
1. Brief Summary paragraph of findings

The clustering was done for different values of k and then natural clusters were observed for both the data sets.

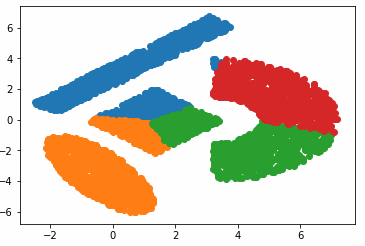
1. Scatter plot of ShapedData.csv



Scatter plot of Clustering.csv

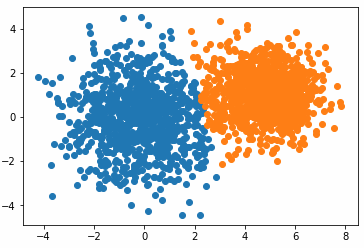


K=4 ShapedData.csv Scatter plot



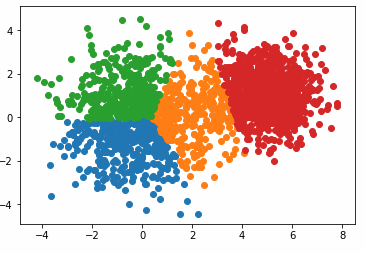
Natural Clusters – 4

K=2 Clustering.csv Scatter plot



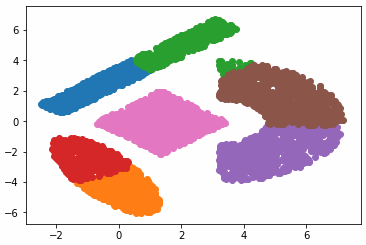
Natural Clusters-2

K=4 Clustering.csv Scatter plot



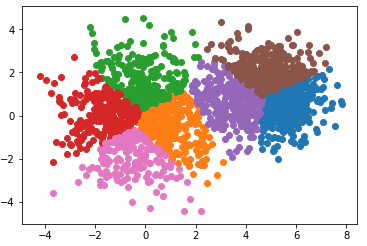
Natural Clusters -2

K=7 ShapedData.csv



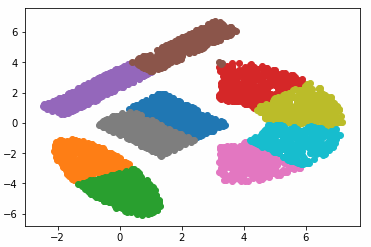
Natural Clusters-4

K=7 Clustering.csv



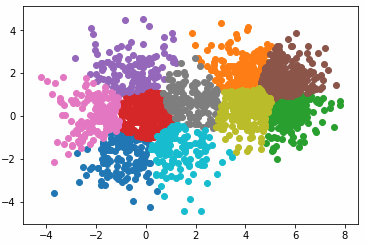
Natural Clusters-2

K=10 ShapedData.csv



Natural Clusters-4

K=10 Clustering.csv



Natural Clusters-2

**Pseudocode for K Means Clustering Algorithm**

**Input:** k(The number of clusters)

D(a set of lift ratios)

**Output:** a set of k clusters

**Method:**

Arbitrarily choose k objects from D as the initial cluster centers;

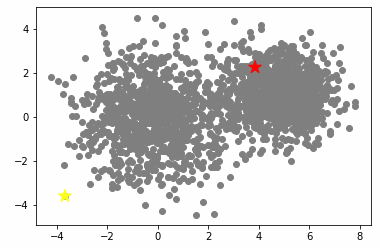
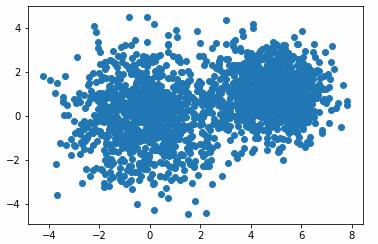
**Repeat:**

1. (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
2. Update the cluster means, i.e. calculate the mean value of the objects for each cluster

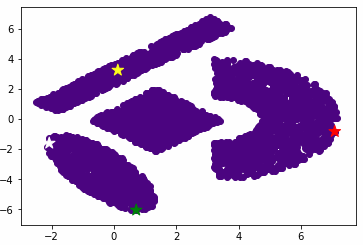
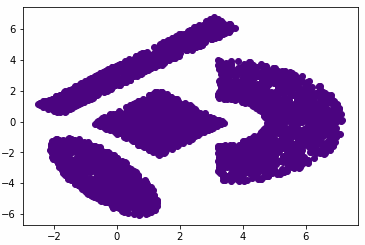
**Until** no change;

**Question 2:**

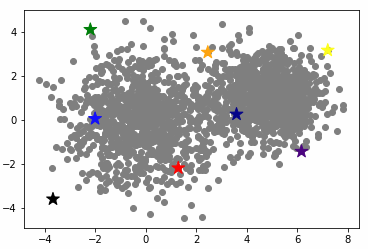
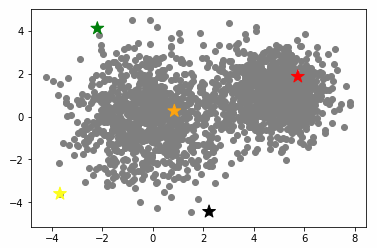
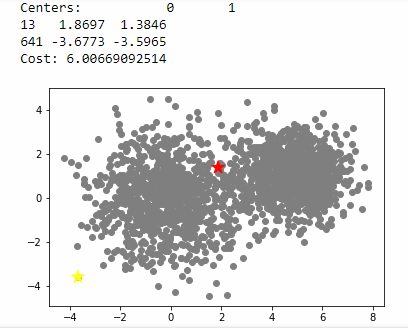
**Scatter plot of Data 1(Clustering.csv)**



**Scatter plot of Data2(ShapedData.csv)**

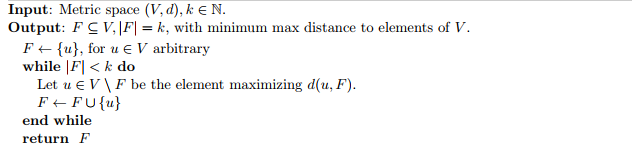


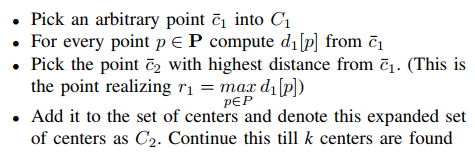
**Plots for k =2, 5 and 8**



Natural Clusters-2

**Pseudocode for Greedy Algorithm**





* Choose the first centre arbitrarily.
* At every step, choose the vertex that is furthest from the current centre to become a centre.
* Continue until k centres are chosen

**Pseudocode for Single Swap:**

BEGIN FUNCTION SWAP

INPUT x, y

OUTPUT x, y (values switched)

t = x

x = y

y = t

RETURN

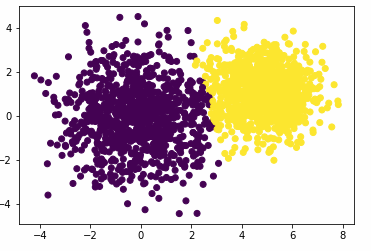
END FUNCTION swap

**Pseudocode for Spectral Clustering:**

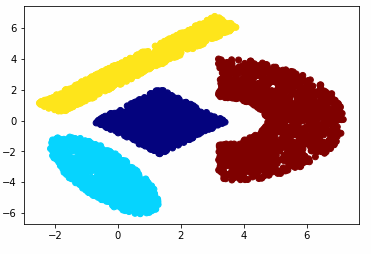
1. project your data into R^{n} 
2. define an A*ffinity* matrix A  , using a Gaussian Kernel K  or say just an Adjacency matrix (i.e. A_{i,j}=\delta_{i,j}) 
3. construct the Graph Laplacian from A  (i.e. decide on a normalization)
4. solve an Eigenvalue problem , such as L v=\lambda v   (or a Generalized Eigenvalue problem L v=\lambda D v )
5. select k eigenvectors \{ v_{i}, i=1, k \}  corresponding to the k lowest (or highest) eigenvalues  \{ \lambda_{i}, i=1, k \} , to define a k-dimensional subspace P^{t}LP 
6. form clusters in this subspace using, say, k-means

**QUESTION 3: Spectral Clustering**

**Scatter plot for Clustering.csv**

 **Natural clusters - 2**

**Scatter Plot of Shapeddata.csv**

 **Natural clusters - 4**

**SHORT ANALYSIS OF THE COMPUTATIONAL EFFORT**

Short analysis:

Lloyd’s algorithm: The computational complexity of each iteration is O(Nkd) , complexity of the worst case is : 

Greedy K Centers algorithm: The computational complexity of each iteration is O(Nkd)

Single-swap algorithm: The algorithm will terminate within  steps.

Spectral clustering: The computational complexity of the similarity matrix is  , and the complexity of eigenvalues and eigenvectors is 

Expectation Maximization: The computational complexity should be  **,** where m is the number of iterations.

The single swap algorithm is the most time consuming one.

**Appendix[]:**

**Question 1: Lloyd’s Algorithm and K-Means Clustering code**

import random

import math

import numpy as np

def random\_centers(data,k):

data\_ = np.array(data)

centres\_list = []

for x in range(0,k):

temp\_rand = np.random.randint( 0,high = 1999)

temp = data\_[temp\_rand]

centres\_list.append(temp)

data\_ = np.delete(data\_,temp,axis = 0)

return((centres\_list))

def assign\_data\_centers(data, k,centres):

thePartition = [[] for \_ in range(0,k)] *# list of k empty lists*

c = np.array(centres)

dp = np.array(data)

for a in dp:

temp\_norm = np.linalg.norm((c-a),axis =1 )

minn = temp\_norm.min()

temp\_list = list(temp\_norm)

thePartition[temp\_list.index(minn)].append(a)

return thePartition

def revalaute\_centres(data\_with\_centres,k):

data = data\_with\_centres

new\_centres=[]

for x in data:

temp = np.array(x)

temp\_m = temp.mean(axis = 0)

new\_centres.append(temp\_m)

return(new\_centres)

import pandas as pd

data = pd.read\_csv('ShapedData.csv',header = None)

k = 4

C = random\_centers(data,k)

D\_C\_2 = assign\_data\_centers(data,k,C)

tol = 1

while tol > 0.00001:

N\_C = revalaute\_centres(D\_C\_2,k)

D\_C\_1 = D\_C\_2

D\_C\_2 = assign\_data\_centers(data,k,N\_C)

temp\_sum = 0

tol = tol/10

import matplotlib.pyplot as plt

plt.plot()

for i in D\_C\_2:

plt.scatter(pd.DataFrame(i)[0],pd.DataFrame(i)[1])

plt.show()

import pandas as pd

data\_2 = pd.read\_csv('clustering.csv', header=None)

k = 4

C = random\_centers(data\_2,k)

D\_C\_2 = assign\_data\_centers(data\_2,k,C)

tol = 1

while tol > 0.00001:

N\_C = revalaute\_centres(D\_C\_2,k)

D\_C\_1 = D\_C\_2

D\_C\_2 = assign\_data\_centers(data\_2,k,N\_C)

temp\_sum = 0

tol = tol/10

import matplotlib.pyplot as plt

plt.plot()

for i in D\_C\_2:

plt.scatter(pd.DataFrame(i)[0],pd.DataFrame(i)[1])

plt.show()

**ALTERNATIVELY the Distance function in the K Means Algorithm may also be written as follows:**

def k\_means(data, centres, k, tol):

"""

Kmeans algorihtm

input data

output: new clusters, centres

"""

n = data.shape[0]

e = 100

c = 0

for epoch in range(0,e):

#calculated distances

print (centres.shape)

dist = distance\_cal (data, centres)

#assign cluster labels

cluster\_labels = np.argmin(dist, axis = 1) # n\*1

#new centroids

centres\_updates = np.zeros(shape = centres.shape)

for j in range(0, k):

points = []

if sum(cluster\_labels == j) == 0:

centres\_updates[j] = centres[j]

else:

# print (data[0])

for d,label in zip(data, cluster\_labels):

if label == j:

points.append(list(d))

centres\_updates[j] = np.mean(points,axis = 0)

np.mean(data[cluster\_labels == j, :], axis=0)

if (np.mean(np.amin(euclidean\_distances(centres, centres\_updates), axis = 1)) <= tol):

print ("Convergence, iterations:", c)

break

c+=1

centres = centres\_updates

return({"Iterations":c, "Centroids": centres,

"Labels": cluster\_labels, "Cost": (np.mean(np.amin(euclidean\_distances(data, centres), axis = 1)))})

**Question 2[]:**

#Importing libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import random as rd

from sklearn.metrics.pairwise import euclidean\_distances

data1 = pd.read\_csv("C:/Users/Karthik/Desktop/IE\_529/clustering.csv", header = None)

data2 = pd.read\_csv("C:/Users/Karthik/Desktop/IE\_529/ShapedData.csv", header = None)

def greedy\_kcenters( data1, k ):

copy = data1.copy()

Init\_center = copy.sample( 1 )

copy.drop(Init\_center.index,inplace = True)

copy.index = list(range( copy.shape[0] ) )

while Init\_center.shape[0] < k:

#Get index of data point that has maximum minimum distance from any centre

ind = np.argmax(np.amin( euclidean\_distances(copy, Init\_center), axis=1 ))

#Append data in temp\_df at index ind into C

Init\_center = Init\_center.append( copy.loc[ind] )

#Remove that row from temp\_df

copy.drop(ind,inplace = True)

#and change indices to 0,1,2,...,n-1

copy.index = list( range( copy.shape[0] ) )

del copy

return( Init\_center, np.amax(np.amin(euclidean\_distances( data1, Init\_center ), axis=1 )) )

plt.scatter(data1[0],data1[1])

plt.show()

centers, dist\_cost = greedy\_kcenters( data1, 2 )

print("Centers:", pd.DataFrame(centers))

print("Cost:", dist\_cost)

plt.scatter(data1[0], data1[1], color = 'grey')

plt.scatter(centers[0],centers[1], marker = "\*", color = ['red', 'yellow'], s =180)

plt.show()

plt.scatter(data1[0], data1[1], color = 'grey')

plt.scatter(centers[0],centers[1], marker = "\*", color = ['red', 'yellow'], s =180)

plt.show()

centers2, dist\_cost2 = greedy\_kcenters( data2, 4 )

print("Centers:", pd.DataFrame(centers2))

print("Cost:", dist\_cost2)

norm = plt.Normalize()

plt.scatter(data2[0], data2[1], color = 'indigo')

plt.scatter(centers2[0],centers2[1], marker = "\*", color = ['yellow', 'green','red','white'], s =150)

plt.show()

k = [2,4,5,7,8,9]

for i in k:

centers, dist\_cost = greedy\_kcenters( data1, i )

print("Centers:", pd.DataFrame(centers))

print("Cost:", dist\_cost)

plt.scatter(data1[0], data1[1], color = 'grey')

plt.scatter(centers[0],centers[1], marker = "\*", color = ['red', 'yellow','green','black', 'orange','indigo', 'blue', 'darkblue', 'lightgreen', 'purple'], s =180)

plt.show()

def sswap\_centers( data, k ):

centers,dist = greedy\_kcenters( data, k )

C0 = pd.DataFrame(euclidean\_distances( data, centers )).min(axis=1).sum()

print("Gk centers cost= ", C0)

counter = centers.shape[0]

beginning = False

while counter > 0:

for i in range( centers.shape[0] ):

if beginning == True:

beginning = False

break

#Create a copy of the dataframe df without the centers Q

copy = pd.concat( [data,centers] )

copy.drop\_duplicates(keep=False)

copy.index = list( range( copy.shape[0] ) )

#Swap

while copy.shape[0] != 0:

centers\_new = pd.concat( [ centers.drop(centers.index[i]), copy.loc[[0]] ] )

#Get cost of new centers

Cost\_new = pd.DataFrame(euclidean\_distances( data, centers\_new )).min(axis=1).sum()

#Check for reduced cost of more than gamma(which is taken to be 0.05)

if Cost\_new <= 0.95 \* C0:

print('1')

centers = centers\_new.copy()

print("SS\_cost= ", Cost\_new)

copy = pd.concat( [data,centers] )

copy.drop\_duplicates(keep=False)

copy.index = list( range( copy.shape[0] ) )

beginning = True

counter = centers.shape[0]

break

else:

copy.drop(0, inplace=True)

copy.index = list( range( copy.shape[0] ) )

if beginning == False:

print(counter)

counter -= 1

#Return the centers

return( centers )

sscenters = sswap\_centers(data1,2)

plt.scatter(data1[0], data1[1], color = 'grey')

plt.scatter(sscenters[0],sscenters[1], marker = "\*", color = ['red', 'yellow'], s =180)

plt.show()

sscenters2 = sswap\_centers(data2,2)

plt.scatter(data2[0], data2[1], color = 'grey')

plt.scatter(sscenters2[0],sscenters2[1], marker = "\*", color = ['red', 'yellow'], s =180)

plt.show()

**QUESTION 3: SPECTRAL CLUSTERING**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import euclidean\_distances

import warnings

warnings.filterwarnings("ignore")

#This cell is for Lloyds Algorithm

def cluster\_index( U, Y ):

#Get the distance matrix which measures euclidean distance between rows of U and Y

distanceMatrix = euclidean\_distances( U, Y )

#Return the index of the column that has minimun value as this would identify the closest centroid

#and the average distance

return ( np.argmin( distanceMatrix, axis=1 ), np.mean( np.amin( distanceMatrix, axis=1 ) ) )

def lloyds\_kmean( U, k, tol ):

#Convert U to pandas dataframe

U = pd.DataFrame( U )

#Initialization

Y = U.sample( k ) #samples k points from df uniformly and stores them into Y

Y.index = list( range( len( Y.index ) ) ) #Replacing index values with regular 0,1,2,3,..,k-1

#Initializing D to have very large value and D\_ to have value less than D - tolerance to allow looping

D = 1e5

D\_ = D - (tol + 7)

while D - D\_ > tol:

D = D\_

#Assigning cluster index to each row of U and getting average distance

C, D\_ = cluster\_index( U, Y )

#Update Y with new centroids by calculating means of clusters

for i in Y.index:

Y.loc[i] = U.loc[U.index[C == i]].mean()

#Return Y, C and D\_

return( Y, C, D\_ )

#This function returns a matrix with Gaussian similarity between rows of dataframe

def gaussian\_similarity\_matrix( df ):

#Ask the user for a sigma value

sigma = float( input( "Please input a value for sigma:\nRemember that the parameter sigma controls the width of the neighborhoods" ) )

#Create Euclidean distance matrix

distance\_Matrix = euclidean\_distances( df, df )

#Modify distance matrix to represent Gaussian similarity matrix

distance\_Matrix = np.exp( -np.square(distance\_Matrix) / ( 2 \* np.square(sigma) ) )

#Return the gaussian-similarity matrix

return distance\_Matrix

#This function returns a matrix where each row has binary values and 1 representing that the row has that row as its nearest neighbor

def k\_Nearest\_Neighbor\_matrix( df ):

#Ask the user to input a value for k

k = int( input( "Please input a value for k") )

#Create a numpy array of zeros of size no. of rows in df x no. of rows in df

k\_NN\_structure = np.zeros((df.shape[0],df.shape[0]))

#Fill in 1 at all k nearest neighbors of each row

k\_NN\_structure[np.array([np.arange(df.shape[0])]).T,\

np.array(list(map( lambda x: np.argsort(euclidean\_distances( x, df ))[:,1:k+1][0], np.array(df) )))] = 1

#Ask the user what kind of undirected k-Nearest Neighbor structure does he/she want

structure\_type = input("What kind of undirected structure do you desire?\n" \

+ "Press M for 'Mutual Nearest Neighbor Graph' which means that " \

+ "vertices vi and vj are connected if both vi is among the k-nearest" \

+ "neighbors of vj and vj is among the k-nearest neighbors of vi\n" \

+ "Press S to simply ignore the directions of the edges, that is " \

+ "connect vi and vj with an undirected edge if vi is among" \

+ "the k-nearest neighbors of vj or if vj is among the k-nearest neighbors of vi.")

if structure\_type == 'M':

k\_NN\_structure = k\_NN\_structure \* k\_NN\_structure.T

elif structure\_type =='S':

k\_NN\_structure[k\_NN\_structure + k\_NN\_structure.T > 0] = 1

#Return the k-Nearest Neighbor structure matrix

return k\_NN\_structure

def adjacency\_matrix\_creation( df ):

#Next two lines not required. Remove later

#Ask the user what kind of similarity function would they like

#similarity\_type = input( "Please select smilarity type of the graph:\nType G for the Gaussian Similarity function\nType K for the k-nearest neighborhood structure\n")

#Get the k Nearest Neighbor Structure

k\_nearest\_neighbor\_structure = k\_Nearest\_Neighbor\_matrix( df )

#Get the Gaussian Similarity matrix

gaussian\_similarity = gaussian\_similarity\_matrix( df )

#Multiply the two together to get the Weighted adjacency matrix W

W = k\_nearest\_neighbor\_structure \* gaussian\_similarity

#Return the Weighted adjacency matrix W

return W

#This function performs un-normalized graph Laplacian Spectral Clustering

def spectral\_clustering( path, k ):

#Read the data

df = pd.read\_csv( path, header = None )

#W = adjacency matrix

W = adjacency\_matrix\_creation( df )

#Get the degree matrix D

D = np.diag( np.sum( W, axis = 1 ) )

#Get the graph Laplacian matrix

L = D - W

#Get the eigen values and eigen vectors of L

w, v = np.linalg.eig( L )

#Sort the eigen vectors based on ascending order of the eigen values

v = v[ : , np.argsort( w ) ]

#Ask the user to input a value of K which would help us get the first K eigen vectors

K = int(input( "Please input a value for K to select the first K eigen vectors: "))

#Store the first K eigen vectors in U

print(v.shape)

U = v[:,:K]

#Ask the user to input a value for tolerance for Lloyds Algorithm

tol = float(input( "Please input a value for tolerance for Lloyds Algorithm" ))

#Run the Lloyds k-Means algorithm to return centers, cluster index vector and the objective function cost

return lloyds\_kmean( U, k, tol )

centers, Cluster\_index, cost = spectral\_clustering( "clustering.csv", 2 )

trial\_df = pd.read\_csv("clustering.csv", header=None)

plt.scatter(trial\_df[0], trial\_df[1], c=Cluster\_index)

plt.show()

centers, Cluster\_index, cost = spectral\_clustering( "ShapedData.csv", 4 )

trial\_df = pd.read\_csv("ShapedData.csv", header=None)

norm = plt.Normalize()

plt.scatter(trial\_df[0], trial\_df[1], c = plt.cm.jet(norm(Cluster\_index)))

plt.show()

print(np.square(1.22474487139), \

np.square(0.894427191))