PROGRAMMING IN DATA SCIENCE : ASSIGNMENT-1 a

Question 1a: Implement a Python program for Nearest Neighbour Classifier that can classify an unknown data sample to one of the given classes. For example, there are 2 classes Red and Blue, and x is an unknown data sample (i.e., we do not know x is red or blue). After calculating all distances between x and all data samples in the 2 classes, we find a data sample in the Red class that has shortest distance to x, so x is classified as a red data sample. Requirements: Your program reads data samples from 2 text files for 2 classes and unknown data samples from another text file, runs the Nearest Neighbour Classifier algorithm as demonstrated in the screenshots below, and outputs all unknown data samples and their classified label to screen and to another text file. Your program should work with any data dimension D > 1 and any number of unknown data samples > 0. For Python programming, use a tuple to store a data sample, a list to store all data samples, and modules to store functions. The main program includes only function calls and does not include any function implementations. Please do not use other versions of Nearest Neighbour Classifier you can find on websites or research articles, and do not import any external packages (except tkinter) to this project.

###

Importing read file and utilty method

```
import io_module as io

import utility as util

#loading dataset

red_data = ["C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/red_2d.txt","C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/red_4d.txt","C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/red_8d.txt"]

blue_data = ["C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/blue_2d.txt","C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/blue_4d.txt","C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/blue_8d.txt"]

unknown_data = ["C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/unknown_2d.txt","C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/unknown_4d.txt","C:/Users/kkabi/Assignment1_final/Template/Assignment 1/datasets/unknown_8d.txt"]


red_dataset = None
```

```
blue_dataset = None
unknown_dataset = None


### Reading the dataset
while True:
  try:
    select_dimension = int(input("Select the Dimension for the Red and blue dataset (2,4,8): "))
    if select_dimension == 2:
      print("Reading 2D datasets...")
      red_dataset = io.read_data_file(red_data[0])
      blue_dataset = io.read_data_file(blue_data[0])
      unknown_dataset = io.read_data_file(unknown_data[0])
      break
    elif select_dimension == 4:
      print("Reading 4D datasets...")
      red_dataset = io.read_data_file(red_data[1])
      blue_dataset = io.read_data_file(blue_data[1])
      unknown_dataset = io.read_data_file(unknown_data[1])
      break
    elif select_dimension == 8:
      print("Reading 8D datasets...")
      red_dataset = io.read_data_file(red_data[2])
      blue_dataset = io.read_data_file(blue_data[2])
      unknown_dataset =io.read_data_file(unknown_data[2])
      break
    else:
      raise ValueError
```

```
    except ValueError:
        print("Please enter a valid number (2, 4, or 8).")



        print(red_dataset)
        print(blue_dataset)
        print(unknown_dataset)



#classify each unknown sample
def classify_unknown_sample(unknown_sample, red_dataset, blue_dataset):
    # Calculate distances to the red and blue datasets
    distances_to_red = [util.calculate_distance(unknown_sample, red_sample) for red_sample in red_dataset]
    min_distance_red = min(distances_to_red)  # Minimum distance to red dataset
    distances_to_blue = [util.calculate_distance(unknown_sample, blue_sample) for blue_sample in blue_dataset]
    min_distance_blue = min(distances_to_blue)  # Minimum distance to blue dataset


    # Classify based on which dataset the sample is closer to
    if min_distance_red < min_distance_blue:
        return "red"
    else:
        return "blue"
#here we are cl;assifying all the unknown datasets to blue or red
for unknown_sample in unknown_dataset:
    classification = classify_unknown_sample(unknown_sample, red_dataset, blue_dataset)
print(f"The unknown point {unknown_sample} is falls in  {classification} class")
```

```
# Here we are making output file where all the list will go
output_filename = "classified_unknown_samples_final_output.txt"
with open(output_filename, "w") as output_file:
    # Write header to the output file
    output_file.write("Unknown Sample, Classified Label\n")

    ### Here we are classifying all the  unknown samples and output results
    for unknown_sample in unknown_dataset:
        classification = classify_unknown_sample(unknown_sample, red_dataset, blue_dataset)

        # Print the classification to the screen
        print(f"The unknown point {unknown_sample} is falls in  {classification} class")

        # Write the classification to the output file
        output_file.write(f"{unknown_sample}, {classification}\n")

print(f"Classification results is  saved to {output_filename}")
```

##

PROGRAMMING IN DATA SCIENCE : ASSIGNMENT-1 b

Question 2: Implement a Python program for K-Means Clustering that can group data samples to clusters. For example, you are given a set of data samples to group them into 2 clusters. The K-means clustering algorithm generates 2 cluster centres at random, groups data samples that are nearest to the first cluster centre to form a cluster then do the same with the second one to form another cluster. The algorithm will generate new cluster centres by averaging data samples in the same cluster. If the difference between the 2 old cluster centres and the 2 new cluster centres are not significant, the algorithm will stop, otherwise it removes the old cluster centres and re-groups data samples for the new cluster centres as seen above to form new clusters. The process repeats until the difference between the old and new cluster centres is not significant. Requirements: Your

program reads data samples from a text file, runs K-means Clustering algorithm as demonstrated in the screenshots below, and outputs all data samples with cluster centres to screen as below. Your program should work with any data dimension D > 1 and any number of clusters K > 1. For Python programming, use tkinter to display data samples and cluster centres on a canvas, a tuple to store a data sample or a cluster centre, a list to store all data samples or all cluster centres, and modules to store functions. The main program includes only function calls and does not include any function implementations. Please do not use other versions of K-Means Clustering that you can find on websites or research articles to implement this project. Please do not import any external packages (except tkinter) to this project.

```
#########################################################################

import io_module as io

import clustering as cluster

import tkinter

import math

import random


# Reading the data files

data_2c_2d = "C:/Users/kkabi/Assignment1_final/Template/Assignment
1/datasets/data_2c_2d.txt"

data_4c_2d = "C:/Users/kkabi/Assignment1_final/Template/Assignment
1/datasets/data_4c_2d.txt"

data_2c_4d = "C:/Users/kkabi/Assignment1_final/Template/Assignment
1/datasets/data_2c_4d.txt"

data_4c_4d = "C:/Users/kkabi/Assignment1_final/Template/Assignment
1/datasets/data_4c_4d.txt"


read_data = io.read_data_file(data_4c_4d)


## To find the dimention and sample size

num_of_dimension = len(read_data[0]) if read_data else 0  # Number of features per
sample
```

```
sample_size = len(read_data)


print(f"Number of dimensions: {num_of_dimension}")
print(f"Sample size: {sample_size}")


### need to input clusters number how much we want
cluster_size = int(input("Input the cluster size: "))
while cluster_size <= 1:
    cluster_size = int(input("Please enter a number greater than 1: "))


## Createing initial random cluster centers
cluster_centers = cluster.generate_K_cluster_centres(cluster_size, num_of_dimension,
read_data)
print("Initial cluster centers:")
for i, center in enumerate(cluster_centers):
    print(f"Cluster {i+1}: {center}")


## Threshold to a small value
threshold = 0.001  # Convergence threshold


### K-means algorithm implementation
def k_means(data, centers, threshold):
    previous_centers = None
    iteration = 0
    max_iterations = 100


    while True:
        iteration += 1
```

```
print(f"\nIteration {iteration}")

# Assigning each point to the nearest cluster
clusters = [[] for _ in range(len(centers))]
for point in data:
    distances = [cluster.euclidean_distance(point, center) for center in centers]
    closest = distances.index(min(distances))
    clusters[closest].append(point)

# Calculating new centers
new_centers = []
for i in range(len(centers)):
    if not clusters[i]:  # If cluster is empty, keep previous center
        new_centers.append(centers[i])
        continue

    # Calculating mean for each dimension
    new_center = tuple(
        sum(point[d] for point in clusters[i])/len(clusters[i])
        for d in range(num_of_dimension)
    )
    new_centers.append(new_center)

# Check here for convergence
if previous_centers:
    converged = True
    for old, new in zip(previous_centers, new_centers):
        if cluster.euclidean_distance(old, new) > threshold:
```

```
                converged = False
                break

        if converged or iteration >= max_iterations:
            break

    previous_centers = centers
    centers = new_centers

    # Print current centers
    print(f"Current cluster centers (iteration {iteration}):")
    for i, center in enumerate(centers):
        print(f"Cluster {i+1}: {center}")

  return centers, clusters

## Run K-means algorithm here
final_centers, final_clusters = k_means(read_data, cluster_centers, threshold)

## Print final results for the final cluster
print("\nFinal cluster centers:")
for i, center in enumerate(final_centers):
  print(f"Cluster {i+1}: {center}")

print("\nCluster assignments:")
for i, cluster_points in enumerate(final_clusters):
  print(f"\nCluster {i+1} ({len(cluster_points)} points):")
  for point in cluster_points:
```

```
        print(point)

### Ploting the data points and the cluster centers with connecting lines here
top = tkinter.Tk()
C = tkinter.Canvas(top, bg="white", height=700, width=700)

# Choose first two dimensions for visualization
if num_of_dimension >= 2:
    ix, iy = 0, 1  # Using first two dimensions for display

    # Transform data for display
    (sx, sy, tx, ty) = cluster.transform_data_for_canvas_display(
        read_data, idx=ix, idy=iy, canvas_height=700, canvas_width=700
    )

    # Define colors for clusters
    colors = ['red', 'blue', 'green', 'yellow', 'purple', 'orange', 'pink', 'brown']

    # First drawing the connecting lines
    for i, (cluster_points, center) in enumerate(zip(final_clusters, final_centers)):
        color = colors[i % len(colors)]
        center_x = center[ix] * sx + tx
        center_y = center[iy] * sy + ty
        for point in cluster_points:
            x = point[ix] * sx + tx
            y = point[iy] * sy + ty
            C.create_line(center_x, center_y, x, y, fill=color, width=1, dash=(2,2))
```

```
    # Then ploting data points
    for i, cluster_points in enumerate(final_clusters):
        color = colors[i % len(colors)]
        for point in cluster_points:
            x = point[ix] * sx + tx
            y = point[iy] * sy + ty
            C.create_oval(x-3, y-3, x+3, y+3, fill=color, outline='black')

    # Finally plot cluster centers
    for i, center in enumerate(final_centers):
        color = colors[i % len(colors)]
        x = center[ix] * sx + tx
        y = center[iy] * sy + ty
        C.create_rectangle(x-6, y-6, x+6, y+6, fill=color, outline='black')
        # Add center label
        C.create_text(x, y-10, text=f"C{i+1}", fill='black', font=('Arial', 10, 'bold'))
else:
    C.create_text(350, 350, text="Cannot visualize - data needs at least 2 dimensions",
                  font=('Arial', 12), fill='black')

C.pack()
top.mainloop()
```

#Used Function:

1. ###function to calculate distance

```
def calculate_distance(point1, point2):
```

```
    """Calculate Euclidean distance between two points (lists or tuples of numbers)."""
    return sum((float(p1) - float(p2)) ** 2 for p1, p2 in zip(point1, point2)) ** 0.5
```

2. #Function to transform data to display on canvas

```
def transform_data_for_canvas_display(data_list, idx=0, idy=1,
canvas_width=800, canvas_height=600):
    #data_list is a list of nD samples, n > 2
    maxW = -1000000.0 #max width
    minW = 1000000.0 #min width
    maxH = -1000000.0 #max height
    minH = 1000000.0 #min height
    for sample in data_list:
        if maxW < sample[idx]:
            maxW = sample[idx]
        if minW > sample[idx]:
            minW = sample[idx]
        if maxH < sample[idy]:
            maxH = sample[idy]
        if minH > sample[idy]:
            minH = sample[idy]
    sx = canvas_width / (maxW - minW)
    tx = canvas_width * minW / (minW - maxW)
    sy = canvas_height / (maxH - minH)
    ty = canvas_height * minH  / (minH - maxH)
    return (sx, sy, tx, ty)
```

3. #Function to display data on canvas

```
def display_data(data_list, xi=0, yi=1, colour='red', shape='circle',
canvas=None, r=5, sx=150, sy=150, tx=300, ty=200):
    for sample in data_list:
        x = sample[xi]
        y = sample[yi]
        x = x*sx + tx
        y = y*sy + ty
        if canvas != None:
            if shape == 'circle':
                canvas.create_oval(x-r, y-r, x+r, y+r, outline = colour,
fill=colour)
            elif shape == 'square':
                canvas.create_rectangle(x-r, y-r, x+r, y+r, outline=colour,
fill=colour)
            elif shape == 'triangle':
                canvas.create_polygon(x, y-r, x-r, y+r, x+r, y+r,
outline=colour, fill=colour)
            else: #if input shape is unknown, draw circle
                canvas.create_oval(x-r, y-r, x+r, y+r, outline = colour,
fill=colour)
#end of function


import math
```

4.
```
def generate_K_cluster_centres(K, dimension, data):
    """Generate K random cluster centers within data range"""
    if not data:
        return []
```

```
    # Find min and max for each dimension
    mins = [min(sample[i] for sample in data) for i in range(dimension)]
    maxs = [max(sample[i] for sample in data) for i in range(dimension)]

    # Generate random centers within data range
    import random
    return [tuple(random.uniform(mins[i], maxs[i]) for i in range(dimension))
            for _ in range(K)]

def euclidean_distance(a, b):
    """Calculate Euclidean distance between two points"""
    return math.sqrt(sum((x - y)**2 for x, y in zip(a, b)))

def transform_data_for_canvas_display(data, idx, idy, canvas_height, canvas_width):
    """Scale data to fit canvas dimensions"""
    if not data:
        return (1, 1, 0, 0)  # Default scaling if no data

    # Find min and max values
    x_values = [sample[idx] for sample in data]
    y_values = [sample[idy] for sample in data]

    min_x, max_x = min(x_values), max(x_values)
    min_y, max_y = min(y_values), max(y_values)

    # Calculate scaling factors
    sx = (canvas_width - 20) / (max_x - min_x) if max_x != min_x else 1
```

```
sy = (canvas_height - 20) / (max_y - min_y) if max_y != min_y else 1


# Calculate translation offsets
tx = 10 - min_x * sx
ty = 10 - min_y * sy


return (sx, sy, tx, ty)
```

5. #Function to read multi-dimension data from a file

```
#Function to read 2D data from file and save data to a list of tuples
# Function to read multi-dimensional data from a file
# Function to read multi-dimensional data from a file
def read_data_file(filename):
  dataset = []  # dataset is a python list
  f = None
  try:
    f = open(filename, 'r')
    while True:
      line = f.readline()
      if len(line) == 0:  # end of file
        break
      line = line.replace('\n', '')  # remove end of line \n character
      string_list = line.split(' ')  # Split by space to separate coordinates

      # Convert each element to float and append as a tuple (for multi-
dimensional data)
      dataset.append(tuple(map(float, string_list)))  # Converts all coordinates to
float and creates a tuple

  except Exception as ex:
    print(ex.args)
  finally:
    if f:
      f.close()

  return dataset
```

6. ###function to calculate distance
   def calculate_distance(point1, point2):
   """Calculate Euclidean distance between two points (lists or tuples of numbers)."""
   return sum((float(p1) - float(p2)) ** 2 for p1, p2 in zip(point1, point2)) ** 0.5

Final output for Assignment 1a

This is the result when we choose 2 dimensions of dataset

```
In [11]: runfile('C:/Users/kkabi/Assignment1_final/Template/Assignment 1/assignment-1a.py', wdir='C:/Users/kkabi/Assignment1_final/
Template/Assignment 1')
Select the Dimension for the Red and blue dataset (2,4,8): 2
Reading 2D datasets...
The unknown point (0.678713, 0.951598) is falls in  blue class
The unknown point (-1.169512, -0.957855) is falls in  blue class
The unknown point (0.631947, 1.236489) is falls in  blue class
The unknown point (-0.131799, -0.324218) is falls in  blue class
The unknown point (-0.199565, -0.229171) is falls in  blue class
The unknown point (0.900907, 0.448131) is falls in  blue class
The unknown point (0.699395, 0.414314) is falls in  blue class
The unknown point (-1.132646, -1.013298) is falls in  blue class
The unknown point (1.004178, 1.355361) is falls in  blue class
The unknown point (-0.127378, -0.097121) is falls in  blue class
The unknown point (6.08804, 3.457729) is falls in  red class
The unknown point (4.147974, 5.275341) is falls in  red class
The unknown point (6.538759, 3.670323) is falls in  red class
The unknown point (4.579573, 4.03559) is falls in  red class
The unknown point (4.756026, 4.184762) is falls in  red class
The unknown point (5.221742, 2.872705) is falls in  red class
The unknown point (5.271773, 3.158064) is falls in  red class
The unknown point (4.046376, 5.19232) is falls in  red class
The unknown point (6.530952, 3.171413) is falls in  red class
The unknown point (4.918007, 4.142507) is falls in  red class
The unknown point (0.678713, 0.951598) is falls in  blue class
The unknown point (-1.169512, -0.957855) is falls in  blue class
The unknown point (0.631947, 1.236489) is falls in  blue class
The unknown point (-0.131799, -0.324218) is falls in  blue class
The unknown point (-0.199565, -0.229171) is falls in  blue class
The unknown point (0.900907, 0.448131) is falls in  blue class
The unknown point (0.699395, 0.414314) is falls in  blue class
The unknown point (-1.132646, -1.013298) is falls in  blue class
The unknown point (1.004178, 1.355361) is falls in  blue class
The unknown point (-0.127378, -0.097121) is falls in  blue class
The unknown point (6.08804, 3.457729) is falls in  red class
The unknown point (4.147974, 5.275341) is falls in  red class
The unknown point (6.538759, 3.670323) is falls in  red class
The unknown point (4.579573, 4.03559) is falls in  red class
The unknown point (4.756026, 4.184762) is falls in  red class
The unknown point (5.221742, 2.872705) is falls in  red class
The unknown point (5.271773, 3.158064) is falls in  red class
The unknown point (4.046376, 5.19232) is falls in  red class
The unknown point (6.530952, 3.171413) is falls in  red class
The unknown point (4.918007, 4.142507) is falls in  red class
Classification results is  saved to classified_unknown_samples_final_output.txt
```

This is the result when we choose 4 dimensions of dataset

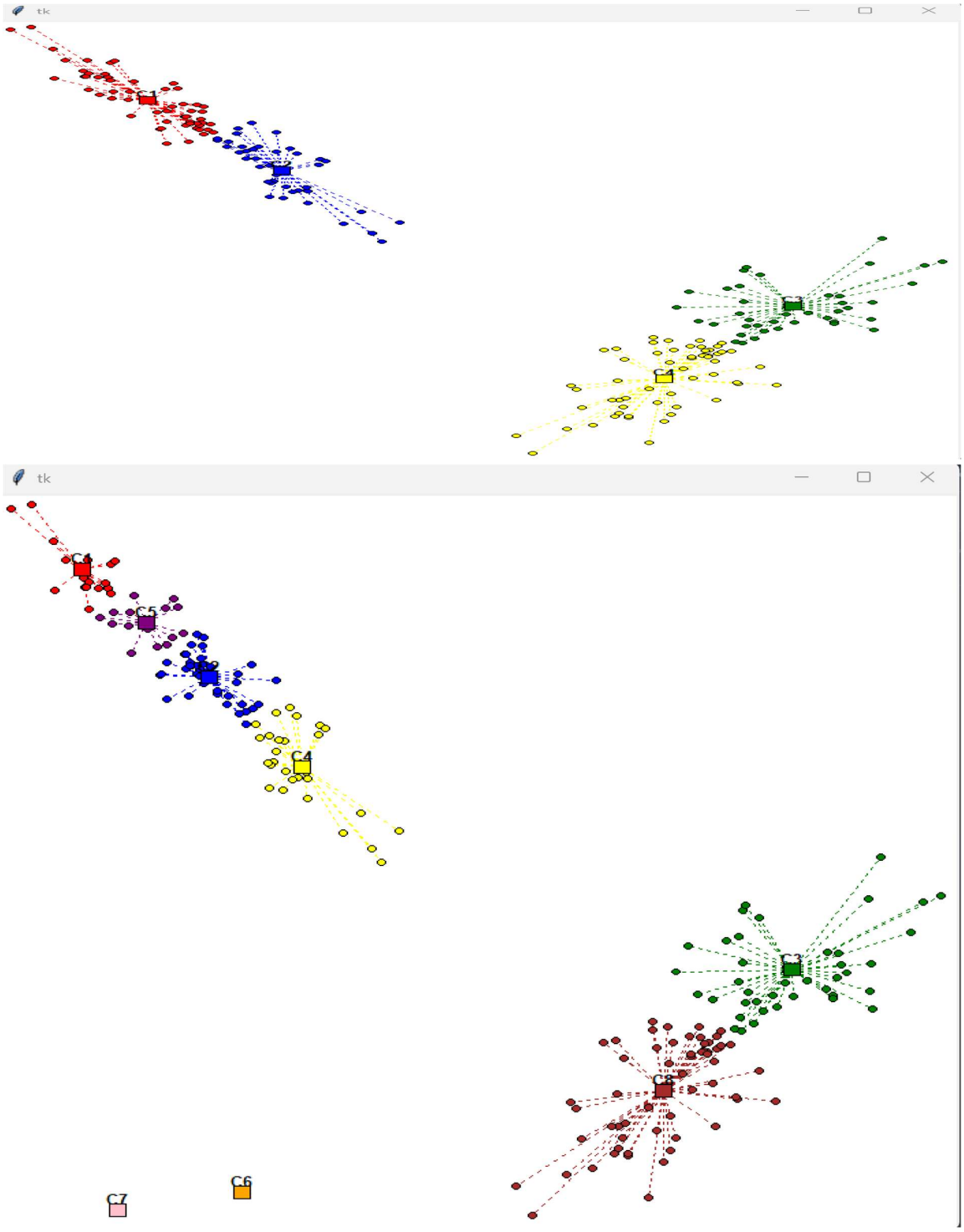
```
In [12]: runfile('C:/Users/kkabi/Assignment1_final/Template/Assignment 1/assignment-1a.py', wdir='C:/Users/kkabi/Assignment1_final/
Template/Assignment 1')
Reloaded modules: io_module, utility
Select the Dimension for the Red and blue dataset (2,4,8): 4
Reading 4D datasets...
The unknown point (5.5, 2.4, 3.8, 1.1) is falls in  blue class
The unknown point (5.7, 2.8, 4.1, 1.3) is falls in  blue class
The unknown point (5.8, 2.6, 4.0, 1.2) is falls in  blue class
The unknown point (6.2, 2.9, 4.3, 1.3) is falls in  blue class
The unknown point (6.1, 2.7, 4.3, 1.3) is falls in  blue class
The unknown point (5.5, 2.4, 3.7, 1.0) is falls in  blue class
The unknown point (5.8, 2.7, 3.9, 1.2) is falls in  blue class
The unknown point (6.0, 2.7, 5.1, 1.6) is falls in  blue class
The unknown point (5.4, 3.0, 4.5, 1.5) is falls in  blue class
The unknown point (6.3, 2.3, 4.4, 1.3) is falls in  blue class
The unknown point (5.6, 3.0, 4.1, 1.3) is falls in  red class
The unknown point (5.5, 2.5, 4.0, 1.3) is falls in  red class
The unknown point (5.5, 2.6, 4.4, 1.2) is falls in  red class
The unknown point (6.1, 3.0, 4.6, 1.4) is falls in  red class
The unknown point (5.0, 2.3, 3.3, 1.0) is falls in  red class
The unknown point (5.6, 2.7, 4.2, 1.3) is falls in  red class
The unknown point (5.7, 3.0, 4.2, 1.2) is falls in  red class
The unknown point (5.7, 2.9, 4.2, 1.3) is falls in  red class
The unknown point (5.1, 2.5, 3.0, 1.1) is falls in  red class
The unknown point (6.0, 3.4, 4.5, 1.6) is falls in  red class
The unknown point (5.5, 2.4, 3.8, 1.1) is falls in  blue class
The unknown point (5.7, 2.8, 4.1, 1.3) is falls in  blue class
The unknown point (5.8, 2.6, 4.0, 1.2) is falls in  blue class
The unknown point (6.2, 2.9, 4.3, 1.3) is falls in  blue class
The unknown point (6.1, 2.7, 4.3, 1.3) is falls in  blue class
The unknown point (5.5, 2.4, 3.7, 1.0) is falls in  blue class
The unknown point (5.8, 2.7, 3.9, 1.2) is falls in  blue class
The unknown point (6.0, 2.7, 5.1, 1.6) is falls in  blue class
The unknown point (5.4, 3.0, 4.5, 1.5) is falls in  blue class
The unknown point (6.3, 2.3, 4.4, 1.3) is falls in  blue class
The unknown point (5.6, 3.0, 4.1, 1.3) is falls in  red class
The unknown point (5.5, 2.5, 4.0, 1.3) is falls in  red class
The unknown point (5.5, 2.6, 4.4, 1.2) is falls in  red class
The unknown point (6.1, 3.0, 4.6, 1.4) is falls in  red class
The unknown point (5.0, 2.3, 3.3, 1.0) is falls in  red class
The unknown point (5.6, 2.7, 4.2, 1.3) is falls in  red class
The unknown point (5.7, 3.0, 4.2, 1.2) is falls in  red class
The unknown point (5.7, 2.9, 4.2, 1.3) is falls in  red class
The unknown point (5.1, 2.5, 3.0, 1.1) is falls in  red class
The unknown point (6.0, 3.4, 4.5, 1.6) is falls in  red class
Classification results is  saved to classified_unknown_samples_final_output.txt
```

This is the result when we choose 8 dimensions of dataset

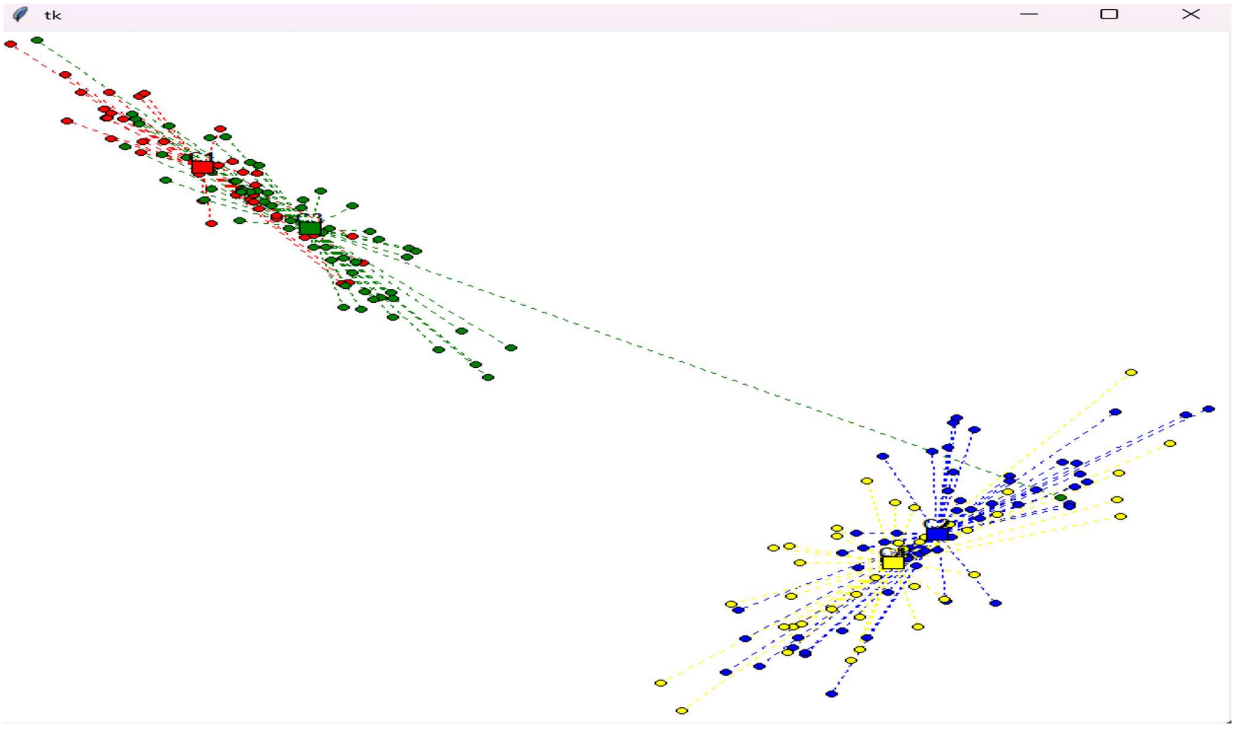
```
In [13]: runfile('C:/Users/kkabi/Assignment1_final/Template/Assignment 1/assignment-1a.py', wdir='C:/Users/kkabi/Assignment1_final/
Template/Assignment 1')
Reloaded modules: io_module, utility
Select the Dimension for the Red and blue dataset (2,4,8): 8
Reading 8D datasets...
The unknown point (-1.252161, 4.595982, 0.403397, 4.664438, 1.184887, 4.729431, 0.483914, 4.661279) is falls in  blue class
The unknown point (1.058662, 4.139604, -0.743436, 5.450736, -0.274353, 4.145079, -0.467963, 4.333563) is falls in  blue class
The unknown point (-0.835754, 4.782662, -1.13831, 5.45584, 0.135823, 4.216405, -0.231157, 5.910545) is falls in  blue class
The unknown point (1.603099, 4.092962, 0.79424, 5.514881, -1.259234, 5.508506, -0.812622, 4.289069) is falls in  blue class
The unknown point (-0.288341, 4.550324, -0.597914, 4.54299, -1.532651, 5.006298, 0.86868, 4.734013) is falls in  blue class
The unknown point (-0.918091, 5.803641, -0.406679, 4.369684, -0.598748, 5.700556, -0.294587, 4.781506) is falls in  blue class
The unknown point (0.920304, 4.52264, 0.313416, 4.17575, -1.23142, 4.807567, 0.873939, 4.097733) is falls in  blue class
The unknown point (-1.892713, 5.419602, 0.145955, 5.497594, -0.847049, 4.472751, 0.649144, 5.086061) is falls in  blue class
The unknown point (0.990046, 4.553275, 0.740694, 4.660983, -0.074056, 4.290081, -0.216103, 4.231506) is falls in  blue class
The unknown point (-0.623098, 4.631027, 0.470979, 5.102245, 1.139348, 5.049803, 1.562672, 4.929757) is falls in  blue class
The unknown point (-1.917901, 1.557697, 1.61902, 2.636847, 0.345654, 2.062426, 0.32132, 2.998103) is falls in  red class
The unknown point (-0.385049, 3.148221, -0.72842, 1.82968, -0.049924, 2.331142, 0.851898, 2.402296) is falls in  red class
The unknown point (1.576337, 2.411215, -0.884658, 2.606353, -1.166365, 2.880788, 0.888455, 1.649488) is falls in  red class
The unknown point (0.300287, 1.865714, -0.168243, 2.290775, 1.491714, 2.283442, -1.825572, 2.360293) is falls in  red class
The unknown point (0.510346, 1.917645, 0.657498, 1.74974, 1.799944, 2.373934, -1.568937, 2.991486) is falls in  red class
The unknown point (-0.586001, 2.052022, -0.665066, 3.081557, 0.885694, 1.791796, 1.468184, 2.046193) is falls in  red class
The unknown point (1.818772, 2.372645, -1.262745, 2.019086, 0.305039, 2.753358, -0.487803, 1.778518) is falls in  red class
The unknown point (1.358644, 2.984491, -0.061581, 1.98725, 0.688312, 1.813675, 0.969963, 2.196707) is falls in  red class
The unknown point (0.246576, 2.359044, -1.153771, 2.375196, 0.141793, 2.199259, 0.52315, 3.053531) is falls in  red class
The unknown point (0.374774, 2.540291, -0.706522, 2.680273, -1.404407, 2.011466, 1.878639, 2.524433) is falls in  red class
The unknown point (-1.252161, 4.595982, 0.403397, 4.664438, 1.184887, 4.729431, 0.483914, 4.661279) is falls in  blue class
The unknown point (1.058662, 4.139604, -0.743436, 5.450736, -0.274353, 4.145079, -0.467963, 4.333563) is falls in  blue class
The unknown point (-0.835754, 4.782662, -1.13831, 5.45584, 0.135823, 4.216405, -0.231157, 5.910545) is falls in  blue class
The unknown point (1.603099, 4.092962, 0.79424, 5.514881, -1.259234, 5.508506, -0.812622, 4.289069) is falls in  blue class
The unknown point (-0.288341, 4.550324, -0.597914, 4.54299, -1.532651, 5.006298, 0.86868, 4.734013) is falls in  blue class
The unknown point (-0.918091, 5.803641, -0.406679, 4.369684, -0.598748, 5.700556, -0.294587, 4.781506) is falls in  blue class
The unknown point (0.920304, 4.52264, 0.313416, 4.17575, -1.23142, 4.807567, 0.873939, 4.097733) is falls in  blue class
The unknown point (-1.892713, 5.419602, 0.145955, 5.497594, -0.847049, 4.472751, 0.649144, 5.086061) is falls in  blue class
The unknown point (0.990046, 4.553275, 0.740694, 4.660983, -0.074056, 4.290081, -0.216103, 4.231506) is falls in  blue class
The unknown point (-0.623098, 4.631027, 0.470979, 5.102245, 1.139348, 5.049803, 1.562672, 4.929757) is falls in  blue class
The unknown point (-1.917901, 1.557697, 1.61902, 2.636847, 0.345654, 2.062426, 0.32132, 2.998103) is falls in  red class
The unknown point (-0.385049, 3.148221, -0.72842, 1.82968, -0.049924, 2.331142, 0.851898, 2.402296) is falls in  red class
The unknown point (1.576337, 2.411215, -0.884658, 2.606353, -1.166365, 2.880788, 0.888455, 1.649488) is falls in  red class
The unknown point (0.300287, 1.865714, -0.168243, 2.290775, 1.491714, 2.283442, -1.825572, 2.360293) is falls in  red class
The unknown point (0.510346, 1.917645, 0.657498, 1.74974, 1.799944, 2.373934, -1.568937, 2.991486) is falls in  red class
The unknown point (-0.586001, 2.052022, -0.665066, 3.081557, 0.885694, 1.791796, 1.468184, 2.046193) is falls in  red class
The unknown point (1.818772, 2.372645, -1.262745, 2.019086, 0.305039, 2.753358, -0.487803, 1.778518) is falls in  red class
The unknown point (1.358644, 2.984491, -0.061581, 1.98725, 0.688312, 1.813675, 0.969963, 2.196707) is falls in  red class
The unknown point (0.246576, 2.359044, -1.153771, 2.375196, 0.141793, 2.199259, 0.52315, 3.053531) is falls in  red class
The unknown point (0.374774, 2.540291, -0.706522, 2.680273, -1.404407, 2.011466, 1.878639, 2.524433) is falls in  red class
Classification results is  saved to classified_unknown_samples_final_output.txt
```

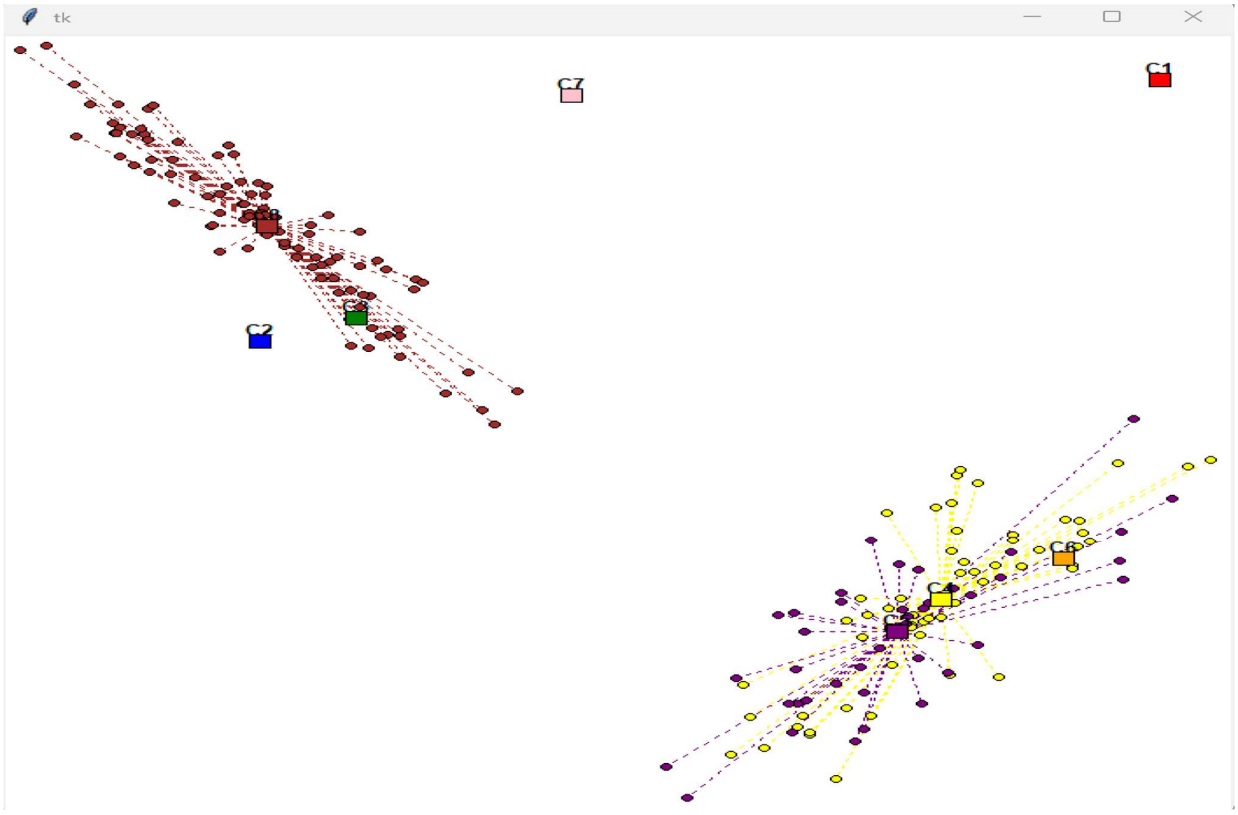
Final output for Assignment 1b

This is the result when we choose dataset named (data_2c_2d) and for 4 and 8 clusters

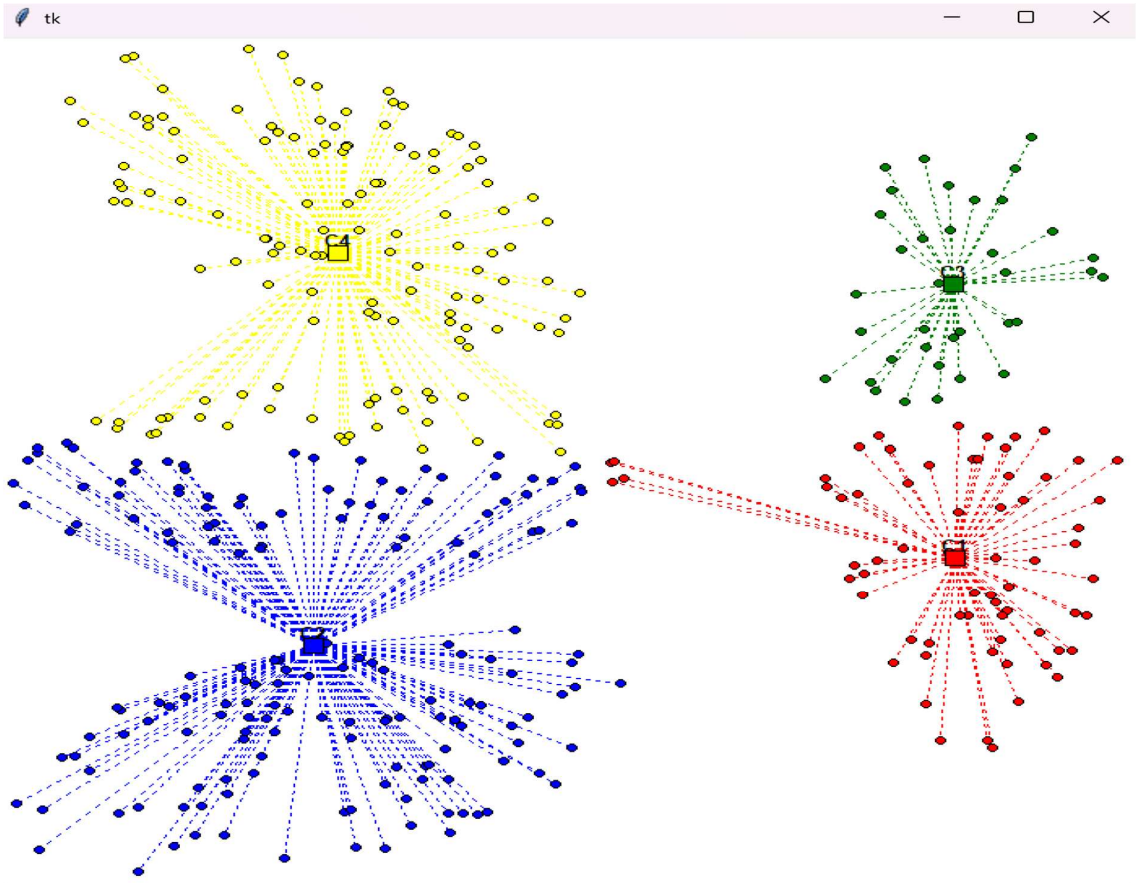


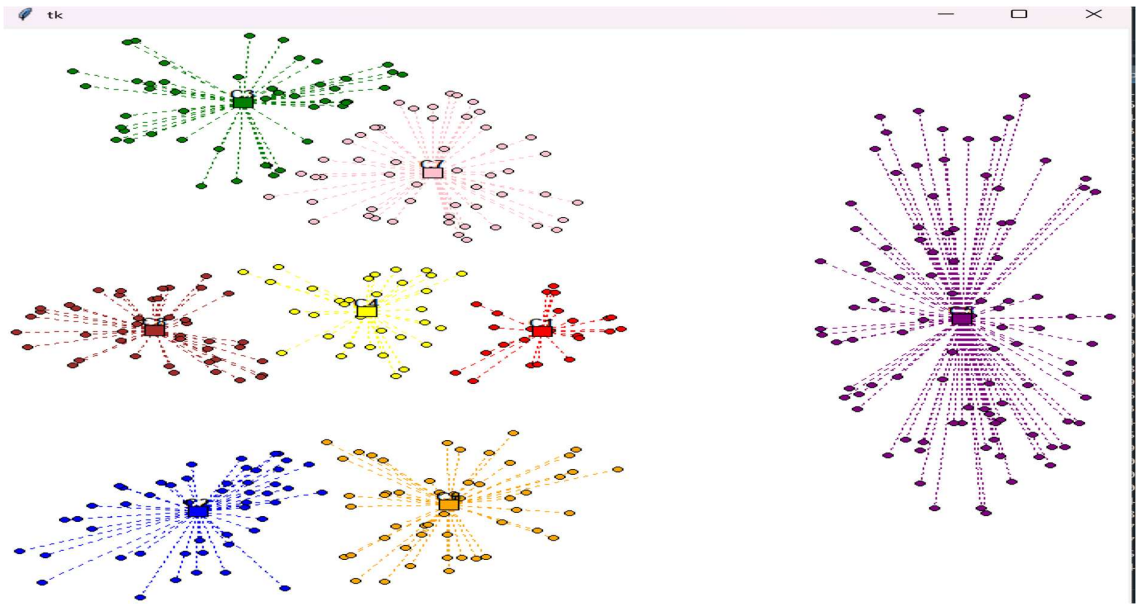
This is the result when we choose dataset named (data_2c_4d) and for 4 and 8 clusters





This is the result when we choose dataset named (data_4c_2d) and for 4 and 8 clusters





This is the result when we choose dataset named (data_4c_4d) and for 4 and 8 clusters