Bisecting K-Means Clustering

Briton A. Powe

Data Mining

Project 2

4/18/2018

THIS IS THE FILE FOR EUCLIDEAN BISECTING K-MEANS

```
"""
Programmer: Briton A. Powe              Program Homework Assignment #2
Date: 4/18/18                           Class: Data Mining
Filename: bisectingKMeansEuclidean.py   Version: 1.7.3
----------------------------------------------------------------------
--
Program Description:
Generates k number of clusters in a Euclidean space and outputs
cluster analysis.
**This Program uses Python 3.6.4***

"""


import random
import math
import matplotlib.pyplot as pl
import copy

#Function to create 20 points with a defined seed
def generate_points(seed):
    points = []

    #Setting the seed
    random.seed(seed)

    #Creating points
    for _ in range(20):
        x = round(random.uniform(1.0, 100.0), 2)
        y = round(random.uniform(1.0, 100.0), 2)

        points.append((x,y))

    return points
```

```
#Function to calculate Euclidean distance
def calculate_euclidean(centroid, point):
    distance = round(math.sqrt(((centroid[0]-point[0])**2)+
((centroid[1]-point[1])**2)), 2)
    return distance
```

```
    distance = round(abs(centroid[0]-point[0]) + abs(centroid[1]-
point[1]), 2)
    return distance
```

```
#Function to calculate all Euclidean distances in a cluster
def calculate_distances_euclidean(centroid, points):
    euclidean_distances = []

    #Finding each distance
    for point in points:
        euclidean_distances.append(calculate_euclidean(centroid,
point))

    return euclidean_distances

#Function to find the centroid based on mean of distances
def calculate_centroid(points):
    sum_of_x = 0.0
    sum_of_y = 0.0
    x_coor = 0.0
    y_coor = 0.0

    #Finding sums
    for point in points:
        sum_of_x += point[0]
        sum_of_y += point[1]

    #Calculating centroid coordinates
```

```
    x_coor = round((1/len(points))*sum_of_x, 2)
    y_coor = round((1/len(points))*sum_of_y, 2)
    centroid = (x_coor, y_coor)

    return centroid

#Function to calcualte SSE
def calculate_SSE(points):
    centroid = calculate_centroid(points)
    SSE = 0
    distances = calculate_distances_euclidean(centroid, points)

    #Summing the squares of distances
    for distance in distances:
        SSE += round((distance)**2, 2)

    return round(SSE, 2)

#Function for selecting centroids
def select_centroids(points):

    #Random select a point in the list
    centroid = random.choice(points)
    points.remove(centroid)

    distances = calculate_distances_euclidean(centroid, points)

    #Set the second centroid as the furtherest point from first
centroid
    second_centroid = points[distances.index(max(distances))]

    return centroid, second_centroid

#Function to generate clusters
def create_clusters(centroids, points):
    cluster_1 = [centroids[0]]
```

```
    cluster_2 = [centroids[1]]
    points = list(set(points)-set(centroids))

    #Organizing points based on proximity to either centroid
    for point in points:
        if calculate_euclidean(centroids[0], point) <=
calculate_euclidean(centroids[1], point):
            cluster_1.append(point)
        else:
            cluster_2.append(point)

    return cluster_1, cluster_2

#Function to split x and y coordinates for screen output
def split_coordinates(points):
    x_values = []
    y_values = []

    for point in points:
        x_values.append(point[0])
        y_values.append(point[1])

    return x_values, y_values

#Function to output clusters in coordinate plane
def print_points(clusters):
    count = 0
    marker = ''

    #Plotting points of each cluster with defined color
    for cluster in clusters:
        x_values, y_values = split_coordinates(cluster)
        if count == 0:
            marker = 'bo'
        elif count == 1:
            marker = 'go'
```

```
        elif count == 2:
            marker = 'ro'
        elif count == 3:
            marker = 'ko'
        elif count == 4:
            marker = 'yo'
        pl.ylabel('Y', {'color': 'y', 'fontsize': 16})
        pl.xlabel('X', {'color': 'y', 'fontsize': 16})
        pl.plot(x_values, y_values, marker)
        count += 1

    pl.axis([0.0, 100.0, 0.0, 100.0])

    #Outputting graph
    pl.show()

#Function to create cluster pairs and chose best pair(Inner loop of 
bisecting k-means)
def create_cluster_list(points):
    collected_clusters = []
    SSE_totals = []
    for _ in range(5):
        #Defining centroids
        first_centroid, second_centroid = 
select_centroids(copy.deepcopy(points))
        centroids = [first_centroid, second_centroid]

        #Bisecting clusters
        cluster_1, cluster_2 = create_clusters(centroids, 
copy.deepcopy(points))

        #Adding to list of potential cluster pairs
        collected_clusters.append([cluster_1, cluster_2])

        #Calculating and noting SSE of pairs
        SSE_cluster_1 = calculate_SSE(cluster_1)
```

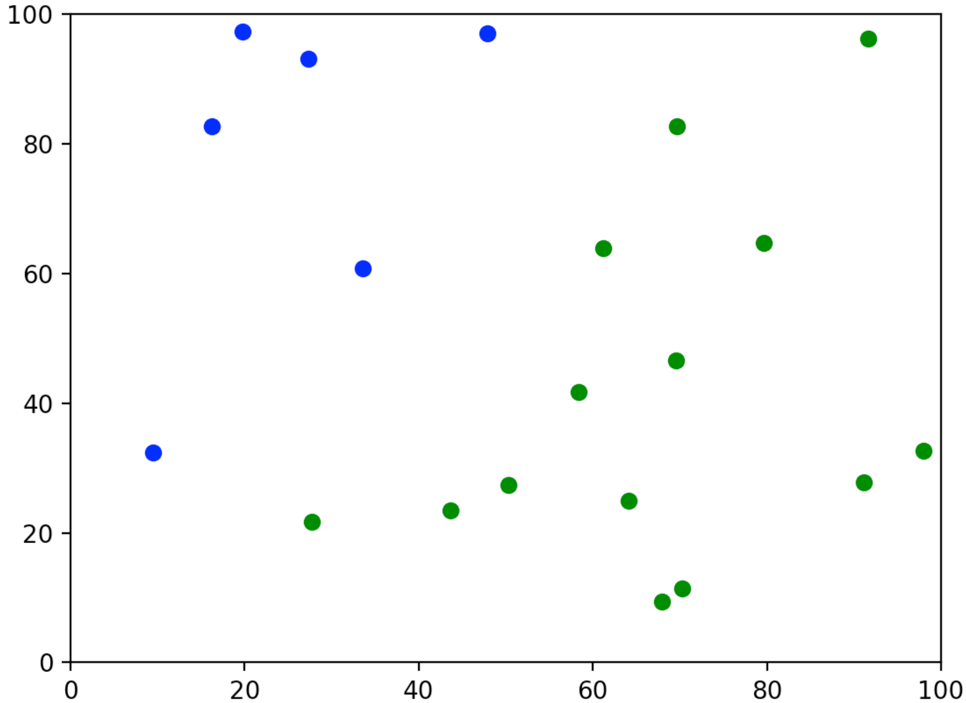
```
        SSE_cluster_2 = calculate_SSE(cluster_2)
        SSE_totals.append(SSE_cluster_1+SSE_cluster_2)


    #Defining the index of best pair
    index = SSE_totals.index(min(SSE_totals))

    #Return best pair
    return collected_clusters[index]

#Function to find all intra-cluster distances of each cluster
def calculate_intracluster(clusters):
    intracluster_distances = []
    total_distance = 0.0

    for cluster in clusters:
        centroid = calculate_centroid(cluster)
        distances = calculate_distances_euclidean(centroid, cluster)
        for distance in distances:
            total_distance += distance
        intracluster_distances.append(round((total_distance/
len(cluster)), 2))
        total_distance = 0

    return intracluster_distances

#Function to find inter-cluster distances between 2 clusters using d
MIN(ij)
def calculate_intercluster_MIN(cluster_1, cluster_2):
    distances = []

    #Finding the distances between every point between the culsters
    for point_1 in cluster_1:
        for point_2 in cluster_2:
            distances.append(calculate_euclidean(point_1, point_2))
```

```
    return min(distances)

#Function to print out the cluster analysis
def output_results(clusters):

    #Generating the legend that cooresponds to outputed graph
    results = "Legend:\n"
    for x in range(len(clusters)):
        if x == 0:
            marker = 'Blue'
        elif x == 1:
            marker = 'Green'
        elif x == 2:
            marker = 'Red'
        elif x == 3:
            marker = 'Black'
        elif x == 4:
            marker = 'Yellow'
        results += "Cluster "+str(x+1)+" - "+marker+": "+str(", 
".join(str(e) for e in clusters[x]))+"\n"

    #Generating intra-cluster distances
    distances = calculate_intracluster(clusters)
    counter = 1
    results += "\nIntra-Cluster Distances:\n"
    for distance in distances:
        results += "Cluster "+str(counter)+" : "+str(distance)+"\n"
        counter += 1
    #Generating intra-cluster sum of all clusters
    results += "\nSum of Intra-Cluster Distances:
\n"+str(round(sum(distances), 2))+"\n"

    #Generating inter-cluster distances
    results += "\nInter-Cluster Distances:\n"
    for index_1 in range(len(clusters)-1):
        for index_2 in range(index_1+1,len(clusters)):
```

```
            distance = calculate_intercluster_MIN(clusters[index_1],
clusters[index_2])
            results += "Distance between Clusters "+str(index_1+1)+"
and "+str(index_2+1)+" : "+str(distance)+"\n"

    print(results)
    
    #Outputing graph
    print_points(clusters)

#Main bisecting k-means functions
def bisecting_k_means(points, k):
    clusters = []
    all_sizes = []
    while len(clusters) != k:
        
        clusters.extend(create_cluster_list(copy.deepcopy(points)))

        if len(clusters) != k:
            for each in clusters:
                all_sizes.append(len(each))
            points = clusters[all_sizes.index(max(all_sizes))]
            clusters.remove(points)
            all_sizes = []

    print("\n\nResults for", k, "Clusters:\n\n")
    output_results(clusters)

#Defined seed
seed = 50

#Initial points
point_list = generate_points(seed)

k = 2
bisecting_k_means(point_list, k)
```

```
k = 4
bisecting_k_means(point_list, k)
```

RESULTS FOR K = 2



Legend:
Cluster 1 - Blue: (70.31, 11.47), (9.43, 32.42), (27.7, 21.72), (64.1, 25.01), (50.26, 27.35), (58.33, 41.76), (69.55, 46.54), (67.97, 9.44), (97.99, 32.61), (91.17, 27.84), (61.14, 63.86), (79.59, 64.67), (43.6, 23.49)

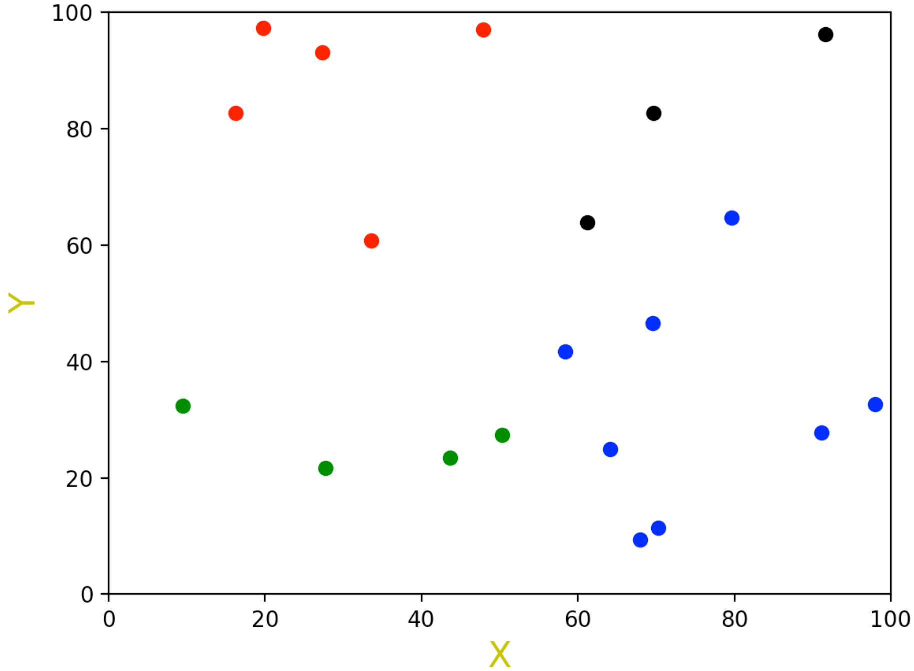
Cluster 2 - Green: (19.71, 97.27), (91.62, 96.28), (47.84, 97.06), (27.35, 93.07), (69.62, 82.67), (33.56, 60.77), (16.18, 82.7)

Intra-Cluster Distances:
Cluster 1 : 25.82
Cluster 2 : 26.49

Sum of Intra-Cluster Distances:
52.31

Inter-Cluster Distances:
Distance between Clusters 1 and 2 (Min): 20.58
Distance between Clusters 1 and 2(Max): 104.08

RESULTS FOR K = 4



Legend:
Cluster 1 - Blue: (97.99, 32.61), (91.17, 27.84), (64.1, 25.01), (70.31, 11.47), (79.59, 64.67), (58.33, 41.76), (69.55, 46.54), (67.97, 9.44)
Cluster 2 - Green: (9.43, 32.42), (27.7, 21.72), (50.26, 27.35), (43.6, 23.49)
Cluster 3 - Red: (19.71, 97.27), (47.84, 97.06), (33.56, 60.77), (16.18, 82.7), (27.35, 93.07)
Cluster 4 - Black: (91.62, 96.28), (61.14, 63.86), (69.62, 82.67)

Intra-Cluster Distances:
Cluster 1 : 20.65
Cluster 2 : 14.91
Cluster 3 : 16.47
Cluster 4 : 16.52

Sum of Intra-Cluster Distances:
68.55

Inter-Cluster Distances:
Distance between Clusters 1 and 2(Min): 14.04
Distance between Clusters 1 and 2(Max): 88.56
Distance between Clusters 1 and 3(Min): 31.22
Distance between Clusters 1 and 3(Max): 101.53
Distance between Clusters 1 and 4(Min): 18.47
Distance between Clusters 1 and 4(Max): 90.0
Distance between Clusters 2 and 3(Min): 37.23
Distance between Clusters 2 and 3(Max): 77.99
Distance between Clusters 2 and 4(Min): 38.1

Distance between Clusters 2 and 4(Max): 104.08
Distance between Clusters 3 and 4(Min): 26.1
Distance between Clusters 3 and 4(Max): 76.65

THIS IS THE FILE FOR MANHATTAN BISECTING K-MEANS

```
"""
Programmer: Briton A. Powe                  Program Homework Assignment #2
Date: 4/18/18                               Class: Data Mining
Filename: bisectingKMeansManhattan.py       Version: 1.7.3
------------------------------------------------------------------------
Program Description:
Generates k number of clusters in a Manhattan space and outputs
cluster analysis.
**This Program uses Python 3.6.4***

"""

import random
import math
import matplotlib.pyplot as pl
import copy

#Function to create 20 points with a defined seed
def generate_points(seed):
    points = []

    #Setting the seed
    random.seed(seed)

    #Creating points
    for _ in range(20):
        x = round(random.uniform(1.0, 100.0), 2)
        y = round(random.uniform(1.0, 100.0), 2)

        points.append((x,y))

    return points
```

```
#Function to calcualte Manhattan distance
def calculate_manhattan(centroid, point):
    distance = round(abs(centroid[0]-point[0]) + abs(centroid[1]-
point[1]), 2)
    return distance

#Function to calculate all Manhattan distances in a cluster
def calculate_distances_manhattan(centroid, points):
    manhattan_distances = []

    #Finding all distances
    for point in points:
        manhattan_distances.append(calculate_manhattan(centroid,
point))

    return manhattan_distances

#Function to find the centroid based on mean of distances
def calculate_centroid(points):
    sum_of_x = 0.0
    sum_of_y = 0.0
    x_coor = 0.0
    y_coor = 0.0

    #Finding sums
    for point in points:
        sum_of_x += point[0]
        sum_of_y += point[1]

    #Calculating centroid coordinates
    x_coor = round((1/len(points))*sum_of_x, 2)
    y_coor = round((1/len(points))*sum_of_y, 2)
    centroid = (x_coor, y_coor)

    return centroid
```

```
#Function to calcualte SSE
def calculate_SSE(points):
    centroid = calculate_centroid(points)
    SSE = 0
    distances = calculate_distances_manhattan(centroid, points)

    #Summing the squares of distances
    for distance in distances:
        SSE += round((distance)**2, 2)

    return round(SSE, 2)

#Function for selecting centroids
def select_centroids(points):

    #Random select a point in the list
    centroid = random.choice(points)
    points.remove(centroid)

    distances = calculate_distances_manhattan(centroid, points)

    #Set the second centroid as the furtherest point from first
centroid
    second_centroid = points[distances.index(max(distances))]

    return centroid, second_centroid

#Function to generate clusters
def create_clusters(centroids, points):
    cluster_1 = [centroids[0]]
    cluster_2 = [centroids[1]]
    points = list(set(points)-set(centroids))

    #Organizing points based on proximity to either centroid
    for point in points:
```

```
        if calculate_manhattan(centroids[0], point) <= 
calculate_manhattan(centroids[1], point):
            cluster_1.append(point)
        else:
            cluster_2.append(point)
    
    return cluster_1, cluster_2
```

```
#Function to split x and y coordinates for screen output
def split_coordinates(points):
    x_values = []
    y_values = []
```

```
    for point in points:
        x_values.append(point[0])
        y_values.append(point[1])
    
    return x_values, y_values
```

```
#Function to output clusters in coordinate plane
def print_points(clusters):
    count = 0
    marker = ''
    
    #Plotting points of each cluster with defined color
    for cluster in clusters:
        x_values, y_values = split_coordinates(cluster)
        if count == 0:
            marker = 'bo'
        elif count == 1:
            marker = 'go'
        elif count == 2:
            marker = 'ro'
        elif count == 3:
            marker = 'ko'
        elif count == 4:
```

```
            marker = 'yo'
        pl.ylabel('Y', {'color': 'y', 'fontsize': 16})
        pl.xlabel('X', {'color': 'y', 'fontsize': 16})
        pl.plot(x_values, y_values, marker)
        count += 1

    pl.axis([0.0, 100.0, 0.0, 100.0])

    #Outputting graph
    pl.show()
```

```
#Function to create cluster pairs and chose best pair(Inner loop of
bisecting k-means)
def create_cluster_list(points):
    collected_clusters = []
    SSE_totals = []
    for _ in range(5):
        #Defining centroids
        first_centroid, second_centroid =
select_centroids(copy.deepcopy(points))
        centroids = [first_centroid, second_centroid]

        #Bisecting clusters
        cluster_1, cluster_2 = create_clusters(centroids,
copy.deepcopy(points))
```

```
        #Adding to list of potential cluster pairs
        collected_clusters.append([cluster_1, cluster_2])

        #Calculating and noting SSE of pairs
        SSE_cluster_1 = calculate_SSE(cluster_1)
        SSE_cluster_2 = calculate_SSE(cluster_2)
        SSE_totals.append(SSE_cluster_1+SSE_cluster_2)
```

```
    #Defining the index of best pair
```

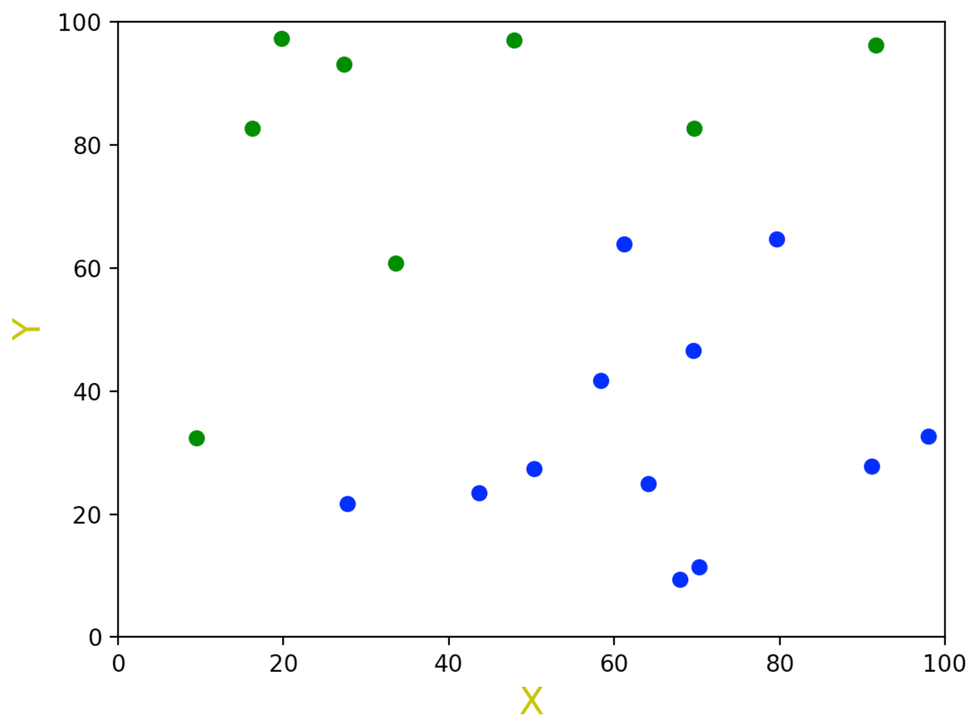
```
    index = SSE_totals.index(min(SSE_totals))

    #Return best pair
    return collected_clusters[index]

#Function to find all intra-cluster distances of each cluster
def calculate_intracluster(clusters):
    intracluster_distances = []
    total_distance = 0.0

    for cluster in clusters:
        centroid = calculate_centroid(cluster)
        distances = calculate_distances_manhattan(centroid, cluster)
        for distance in distances:
            total_distance += distance
        intracluster_distances.append(round((total_distance/
len(cluster)), 2))
        total_distance = 0

    return intracluster_distances

#Function to find inter-cluster distances between 2 clusters using d
MIN(ij)
def calculate_intercluster_MIN(cluster_1, cluster_2):
    distances = []

    #Finding the distances between every point between the culsters
    for point_1 in cluster_1:
        for point_2 in cluster_2:
            distances.append(calculate_manhattan(point_1, point_2))

    return min(distances)

#Function to print out the cluster analysis
def output_results(clusters):
```

```
    #Generating the legend that cooresponds to outputed graph
    results = "Legend:\n"
    for x in range(len(clusters)):
        if x == 0:
            marker = 'Blue'
        elif x == 1:
            marker = 'Green'
        elif x == 2:
            marker = 'Red'
        elif x == 3:
            marker = 'Black'
        elif x == 4:
            marker = 'Yellow'
        results += "Cluster "+str(x+1)+" - "+marker+": "+str(", 
".join(str(e) for e in clusters[x]))+"\n"
```

```
    #Generating intra-cluster distances
    distances = calculate_intracluster(clusters)
    counter = 1
    results += "\nIntra-Cluster Distances:\n"
    for distance in distances:
        results += "Cluster "+str(counter)+" : "+str(distance)+"\n"
        counter += 1
    #Generating intra-cluster sum of all clusters
    results += "\nSum of Intra-Cluster Distances:
\n"+str(round(sum(distances), 2))+"\n"

    #Generating inter-cluster distances
    results += "\nInter-Cluster Distances:\n"
    for index_1 in range(len(clusters)-1):
        for index_2 in range(index_1+1,len(clusters)):
            distance = calculate_intercluster_MIN(clusters[index_1],
clusters[index_2])
            results += "Distance between Clusters "+str(index_1+1)+"
and "+str(index_2+1)+" : "+str(distance)+"\n"
```

```
    print(results)

    #Outputing graph
    print_points(clusters)

#Main bisecting k-means functions
def bisecting_k_means(points, k):
    clusters = []
    all_sizes = []
    while len(clusters) != k:

        clusters.extend(create_cluster_list(copy.deepcopy(points)))

        if len(clusters) != k:
            for each in clusters:
                all_sizes.append(len(each))
            points = clusters[all_sizes.index(max(all_sizes))]
            clusters.remove(points)
            all_sizes = []

    print("\n\nResults for", k, "Clusters:\n\n")
    output_results(clusters)

#Defined seed
seed = 50

#Initial points
point_list = generate_points(seed)

k = 2
bisecting_k_means(point_list, k)

k = 4
bisecting_k_means(point_list, k)
```

RESULTS FOR K = 2



Legend:
Cluster 1 - Blue: (70.31, 11.47), (27.7, 21.72), (64.1, 25.01), (50.26, 27.35), (58.33, 41.76), (69.55, 46.54), (67.97, 9.44), (97.99, 32.61), (91.17, 27.84), (61.14, 63.86), (79.59, 64.67), (43.6, 23.49)

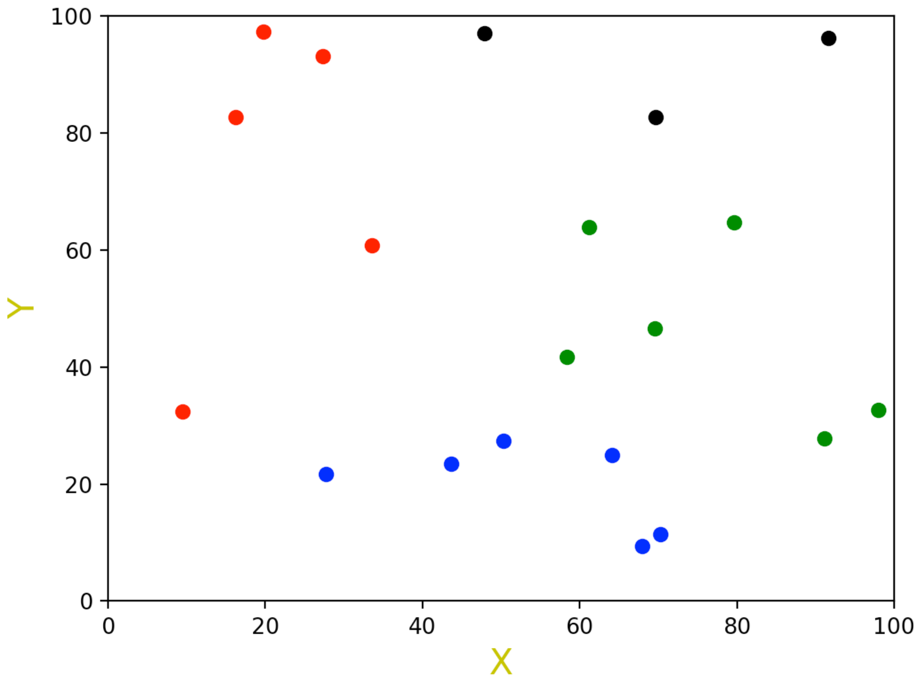
Cluster 2 - Green: (19.71, 97.27), (91.62, 96.28), (9.43, 32.42), (47.84, 97.06), (27.35, 93.07), (69.62, 82.67), (33.56, 60.77), (16.18, 82.7)

Intra-Cluster Distances:
Cluster 1 : 28.44
Cluster 2 : 39.55

Sum of Intra-Cluster Distances:
67.99

Inter-Cluster Distances:
Distance between Clusters 1 and 2(Min): 27.29
Distance between Clusters 1 and 2(Max): 142.94

RESULTS FOR K = 4



Legend:
Cluster 1 - Blue: (27.7, 21.72), (64.1, 25.01), (70.31, 11.47), (50.26, 27.35), (43.6, 23.49), (67.97, 9.44)

Cluster 2 - Green: (79.59, 64.67), (91.17, 27.84), (61.14, 63.86), (58.33, 41.76), (69.55, 46.54), (97.99, 32.61)

Cluster 3 - Red: (33.56, 60.77), (9.43, 32.42), (19.71, 97.27), (16.18, 82.7), (27.35, 93.07)

Cluster 4 - Black: (91.62, 96.28), (69.62, 82.67), (47.84, 97.06)

Intra-Cluster Distances:
Cluster 1 : 19.66
Cluster 2 : 25.43
Cluster 3 : 28.69
Cluster 4 : 20.84

Sum of Intra-Cluster Distances:
94.62

Inter-Cluster Distances:
Distance between Clusters 1 and 2(Min): 22.48
Distance between Clusters 1 and 2(Max): 94.84
Distance between Clusters 1 and 3(Min): 28.97
Distance between Clusters 1 and 3(Max): 136.4
Distance between Clusters 1 and 4(Min): 63.18
Distance between Clusters 1 and 4(Max): 138.48

Distance between Clusters 2 and 3(Min): 30.67
Distance between Clusters 2 and 3(Max): 142.94
Distance between Clusters 2 and 4(Min): 27.29
Distance between Clusters 2 and 4(Max): 114.6
Distance between Clusters 3 and 4(Min): 24.48
Distance between Clusters 3 and 4(Max): 146.05