CS 6200 4/13/18

Che Shian Hung

Programming Assignment 3

Minimax & Kmeans Algorithms

The program implements minimax and kmeans cluster classifying algorithms to classify randomly generated samples, and display the testing results. The samples are generated from five circle centers. Each circle center is used to generate ten samples within a given radius. In this case, the five circle centers are hard coded as (6, -6), (-6, -6), (10, 6), (-10, 6), and (0, 10). The distance between two circle centers are all greater than ten. While the program is running, the user can modify several parameters to test the impact on the result. For each program run, a new set of samples will be randomly generated, and the user can test the sample data over and over again through the terminal.

For minimax algorithm, starting with one class center, after each iteration, it will find a potential class center and test the point with the previous class centers and see if the algorithm is finished or not. For kmeans algorithm, before the algorithm runs, the user can input a number to indicate the number of classes are going to classify during the algorithm. Initially, the algorithm will randomly select the same number of samples as initial class centers, and reclassifying the samples and calculating the class centers again ang again until the generated class centers are the same as previous class centers.

In this program, the similarity measure is calculated based on the distance between two points. For the testing report, since the samples are generated by each circle center sequentially, the result samples generated from the same circle center is separated from others in the report.

In this report, I will demonstrate the test result for each algorithm on two different dataset. One is generated with circle radius 4, and the second dataset is generated with circle radius 15. The two dataset is used to observer the algorithm’s behavior changes when the samples’ cohesion changes. For kmeans algorithm, due to the nature of kmeans algorithm, I will test it several times and pick the testing result with the best performance.

**Dataset 1 (Radius = 4)**

display all samples:

( 6.85, -5.69)

( 5.43, -6.94)

( 4.04, -7.22)

( 6.87, -5.18)

( 8.05, -7.77)

( 6.81, -6.53)

( 6.30, -6.04)

( 9.06, -6.28)

( 6.51, -5.23)

( 3.49, -4.73)

( -3.54, -4.79)

( -6.28, -5.71)

( -4.32, -2.68)

( -5.59, -6.01)

( -6.37, -3.32)

( -4.22, -6.35)

( -8.57, -4.92)

( -8.49, -6.87)

( -4.84, -5.99)

( -5.64, -6.18)

( 8.64, 2.35)

( 9.66, 7.35)

( 10.87, 6.74)

( 12.12, 5.44)

( 11.75, 3.41)

( 12.47, 5.45)

( 10.65, 5.46)

( 8.20, 5.76)

( 8.35, 3.14)

( 10.93, 3.34)

( -9.36, 6.43)

(-10.61, 5.35)

( -9.24, 6.02)

(-11.59, 8.98)

( -9.02, 5.47)

(-10.25, 4.75)

( -9.90, 7.14)

( -9.13, 8.78)

( -9.28, 6.93)

(-10.64, 6.14)

( 1.33, 9.10)

( -0.66, 10.68)

( 2.06, 10.97)

( -1.15, 9.85)

( 1.88, 9.09)

( 0.01, 10.01)

( 0.75, 10.97)

( -0.14, 10.14)

( -1.95, 9.76)

( 0.11, 11.21)

**Dataset 2 (Radius = 15)**

display all samples:

( 6.20, -5.56)

( 5.31, -12.65)

( 6.65, -3.70)

( 10.60, -7.65)

( 14.00, -1.23)

( 5.56, -14.46)

( 14.90, -3.27)

( 7.44, 4.24)

( 3.14, -10.15)

( 6.44, -10.32)

(-18.17, -3.04)

( -6.01, -6.09)

(-14.22, -5.54)

( 6.63, 1.06)

( -1.67, 6.49)

( -5.95, -5.03)

( -6.31, -5.38)

( -2.34, -17.02)

( -5.88, -7.64)

( -5.40, -8.32)

( 5.66, 11.35)

( 1.27, 3.85)

( 13.00, -0.03)

( 19.45, 0.39)

( 10.89, 12.29)

( 9.62, 3.66)

( 0.05, -0.08)

( 12.69, -0.09)

( 12.09, 15.32)

( 4.75, -1.19)

(-10.64, 9.30)

(-19.95, 1.25)

(-11.17, 7.24)

( 2.95, 6.93)

( -9.38, -6.32)

(-23.37, 12.75)

(-12.36, 5.58)

(-15.52, -7.88)

( -9.04, 5.60)

(-12.85, 7.22)

( 0.96, 23.20)

( 0.03, 9.93)

( 0.42, 10.51)

( -2.55, -0.76)

(-10.88, 4.38)

( -2.14, 9.39)

( 12.97, 7.25)

( -2.72, 9.90)

( 9.10, 11.32)

( 0.21, 10.41)

These are the samples for two dataset. The First ten samples are generated from the first circle center, and the second ten samples are generated from the second circle center. We can observe which circle center the sample belongs to easily from the first dataset. However, it is hard to observe anything from the second data set since the samples are generated within radius 15, which exceeds the minimum distance between the circle centers. Even if we draw all samples on a 2d grid, it will be hard to visualize any cluster among the data. The sample data are directly copied from the program output from the terminal. Now we are going to look at the testing result from minimax algorithm on two dataset.

**Minimax Algorithm – Dataset 1 (Radius 4)**

Input 1 or 2 to run minimax(1) or kmeans(2): 1

display minimaxResult:

number of class: 5

number of iteration: 4

( 6.85, -5.69) => class 1

( 5.43, -6.94) => class 1

( 4.04, -7.22) => class 1

( 6.87, -5.18) => class 1

( 8.05, -7.77) => class 1

( 6.81, -6.53) => class 1

( 6.30, -6.04) => class 1

( 9.06, -6.28) => class 1

( 6.51, -5.23) => class 1

( 3.49, -4.73) => class 1

( -3.54, -4.79) => class 3

( -6.28, -5.71) => class 3

( -4.32, -2.68) => class 3

( -5.59, -6.01) => class 3

( -6.37, -3.32) => class 3

( -4.22, -6.35) => class 3

( -8.57, -4.92) => class 3

( -8.49, -6.87) => class 3

( -4.84, -5.99) => class 3

( -5.64, -6.18) => class 3

( 8.64, 2.35) => class 5

( 9.66, 7.35) => class 5

( 10.87, 6.74) => class 5

( 12.12, 5.44) => class 5

( 11.75, 3.41) => class 5

( 12.47, 5.45) => class 5

( 10.65, 5.46) => class 5

( 8.20, 5.76) => class 5

( 8.35, 3.14) => class 5

( 10.93, 3.34) => class 5

( -9.36, 6.43) => class 2

(-10.61, 5.35) => class 2

( -9.24, 6.02) => class 2

(-11.59, 8.98) => class 2

( -9.02, 5.47) => class 2

(-10.25, 4.75) => class 2

( -9.90, 7.14) => class 2

( -9.13, 8.78) => class 2

( -9.28, 6.93) => class 2

(-10.64, 6.14) => class 2

( 1.33, 9.10) => class 4

( -0.66, 10.68) => class 4

( 2.06, 10.97) => class 4

( -1.15, 9.85) => class 4

( 1.88, 9.09) => class 4

( 0.01, 10.01) => class 4

( 0.75, 10.97) => class 4

( -0.14, 10.14) => class 4

( -1.95, 9.76) => class 4

( 0.11, 11.21) => class 4

display class centers:

class 1: ( 6.34, -6.16), original index: 0

class 2: ( -9.90, 6.60), original index: 33

class 3: ( -5.79, -5.28), original index: 17

class 4: ( 0.22, 10.18), original index: 42

class 5: ( 10.36, 4.84), original index: 25

For the testing result on the first dataset, we can see minimax algorithm classifies the samples exactly based on the circle centers. Remember in this case, the class number does not matter since we only focus on which samples are classified in the same cluster. Since the radius for sample generation is small, if we draw all the samples on a 2d grid, there is no ambiguous of classifying any specific samples. However, while testing with few dataset, the testing result might not be as we expect. Samples generated from different circle centers can be classified into same cluster, and the number of classes will be 4 instead of 5. Since each iteration will determine whether there is a new cluster center, the number of iteration is always the number of classes minus one. At the end of the testing report, the class enters are lists. At the end of minimax algorithm, we used the final class centers, which are the original sample index here, to classify all sample again, and the average position of each cluster is the final class centers listed here.

**Minimax Algorithm – Dataset 2 (Radius 15)**

Input 1 or 2 to run minimax(1) or kmeans(2): 1

display minimaxResult:

number of class: 8

number of iteration: 7

( 6.20, -5.56) => class 1

( 5.31, -12.65) => class 1

( 6.65, -3.70) => class 1

( 10.60, -7.65) => class 1

( 14.00, -1.23) => class 7

( 5.56, -14.46) => class 8

( 14.90, -3.27) => class 7

( 7.44, 4.24) => class 6

( 3.14, -10.15) => class 1

( 6.44, -10.32) => class 1

(-18.17, -3.04) => class 4

( -6.01, -6.09) => class 4

(-14.22, -5.54) => class 4

( 6.63, 1.06) => class 1

( -1.67, 6.49) => class 5

( -5.95, -5.03) => class 4

( -6.31, -5.38) => class 4

( -2.34, -17.02) => class 8

( -5.88, -7.64) => class 4

( -5.40, -8.32) => class 8

( 5.66, 11.35) => class 6

( 1.27, 3.85) => class 5

( 13.00, -0.03) => class 7

( 19.45, 0.39) => class 7

( 10.89, 12.29) => class 6

( 9.62, 3.66) => class 6

( 0.05, -0.08) => class 1

( 12.69, -0.09) => class 7

( 12.09, 15.32) => class 6

( 4.75, -1.19) => class 1

(-10.64, 9.30) => class 5

(-19.95, 1.25) => class 4

(-11.17, 7.24) => class 5

( 2.95, 6.93) => class 6

( -9.38, -6.32) => class 4

(-23.37, 12.75) => class 2

(-12.36, 5.58) => class 5

(-15.52, -7.88) => class 4

( -9.04, 5.60) => class 5

(-12.85, 7.22) => class 5

( 0.96, 23.20) => class 3

( 0.03, 9.93) => class 5

( 0.42, 10.51) => class 6

( -2.55, -0.76) => class 5

(-10.88, 4.38) => class 5

( -2.14, 9.39) => class 5

( 12.97, 7.25) => class 6

( -2.72, 9.90) => class 5

( 9.10, 11.32) => class 6

( 0.21, 10.41) => class 5

display class centers:

class 1: ( 8.30, -8.37), original index: 0

class 2: ( -3.90, 2.13), original index: 35

class 3: ( 0.16, 3.87), original index: 40

class 4: (-16.90, -7.61), original index: 37

class 5: (-12.42, 14.75), original index: 38

class 6: ( 11.85, 13.81), original index: 24

class 7: ( 12.34, -0.70), original index: 23

class 8: ( -0.36, -6.63), original index: 17

Since the data is chaotic, there is no more optimal number of clusters unless we analyze further into the dataset. Anyway, minimax algorithm classified the data into 8 classes in this case.

**2-means Algorithm – Dataset 1 (Radius 4)**

Input 1 or 2 to run minimax(1) or kmeans(2): 2

Input the number of k for running kmeans (2 - 7): 2

initial center index:

11 22

display kmeansResult:

number of class: 2

number of iteration: 4

( 6.85, -5.69) => class 2

( 5.43, -6.94) => class 2

( 4.04, -7.22) => class 2

( 6.87, -5.18) => class 2

( 8.05, -7.77) => class 2

( 6.81, -6.53) => class 2

( 6.30, -6.04) => class 2

( 9.06, -6.28) => class 2

( 6.51, -5.23) => class 2

( 3.49, -4.73) => class 2

( -3.54, -4.79) => class 1

( -6.28, -5.71) => class 1

( -4.32, -2.68) => class 1

( -5.59, -6.01) => class 1

( -6.37, -3.32) => class 1

( -4.22, -6.35) => class 1

( -8.57, -4.92) => class 1

( -8.49, -6.87) => class 1

( -4.84, -5.99) => class 1

( -5.64, -6.18) => class 1

( 8.64, 2.35) => class 2

( 9.66, 7.35) => class 2

( 10.87, 6.74) => class 2

( 12.12, 5.44) => class 2

( 11.75, 3.41) => class 2

( 12.47, 5.45) => class 2

( 10.65, 5.46) => class 2

( 8.20, 5.76) => class 2

( 8.35, 3.14) => class 2

( 10.93, 3.34) => class 2

( -9.36, 6.43) => class 1

(-10.61, 5.35) => class 1

( -9.24, 6.02) => class 1

(-11.59, 8.98) => class 1

( -9.02, 5.47) => class 1

(-10.25, 4.75) => class 1

( -9.90, 7.14) => class 1

( -9.13, 8.78) => class 1

( -9.28, 6.93) => class 1

(-10.64, 6.14) => class 1

( 1.33, 9.10) => class 2

( -0.66, 10.68) => class 2

( 2.06, 10.97) => class 2

( -1.15, 9.85) => class 2

( 1.88, 9.09) => class 2

( 0.01, 10.01) => class 2

( 0.75, 10.97) => class 2

( -0.14, 10.14) => class 2

( -1.95, 9.76) => class 2

( 0.11, 11.21) => class 2

display class centers:

class 1: ( -7.84, 0.66)

class 2: ( 5.64, 2.95)

With 2-means algorithm, we found that the algorithm split whole cluster into half. The final class centers are list at the bottom, which we can see the first class center dominates the left half of samples, and the second class center dominates the right half the samples. Even if the number of samples in each cluster is not the same, each sample has the smallest distance between itself and the cluster center it belongs to compare with another cluster center. The initial center indexes are listed at the top, meaning that at the beginning of the program, the algorithm chose to assign the initial class centers as the 11th sample and the 22th sample. Depends on different initial class centers, the result can be turned out different. In this case, it takes four iterations to perform 2-means algorithm.

**3-means Algorithm – Dataset 1 (Radius 4)**

Input 1 or 2 to run minimax(1) or kmeans(2): 2

Input the number of k for running kmeans (2 - 7): 3

initial center index:

28 15 26

display kmeansResult:

number of class: 3

number of iteration: 3

( 6.85, -5.69) => class 1

( 5.43, -6.94) => class 1

( 4.04, -7.22) => class 1

( 6.87, -5.18) => class 1

( 8.05, -7.77) => class 1

( 6.81, -6.53) => class 1

( 6.30, -6.04) => class 1

( 9.06, -6.28) => class 1

( 6.51, -5.23) => class 1

( 3.49, -4.73) => class 1

( -3.54, -4.79) => class 2

( -6.28, -5.71) => class 2

( -4.32, -2.68) => class 2

( -5.59, -6.01) => class 2

( -6.37, -3.32) => class 2

( -4.22, -6.35) => class 2

( -8.57, -4.92) => class 2

( -8.49, -6.87) => class 2

( -4.84, -5.99) => class 2

( -5.64, -6.18) => class 2

( 8.64, 2.35) => class 3

( 9.66, 7.35) => class 3

( 10.87, 6.74) => class 3

( 12.12, 5.44) => class 3

( 11.75, 3.41) => class 3

( 12.47, 5.45) => class 3

( 10.65, 5.46) => class 3

( 8.20, 5.76) => class 3

( 8.35, 3.14) => class 3

( 10.93, 3.34) => class 3

( -9.36, 6.43) => class 2

(-10.61, 5.35) => class 2

( -9.24, 6.02) => class 2

(-11.59, 8.98) => class 2

( -9.02, 5.47) => class 2

(-10.25, 4.75) => class 2

( -9.90, 7.14) => class 2

( -9.13, 8.78) => class 2

( -9.28, 6.93) => class 2

(-10.64, 6.14) => class 2

( 1.33, 9.10) => class 1

( -0.66, 10.68) => class 1

( 2.06, 10.97) => class 1

( -1.15, 9.85) => class 1

( 1.88, 9.09) => class 1

( 0.01, 10.01) => class 1

( 0.75, 10.97) => class 1

( -0.14, 10.14) => class 1

( -1.95, 9.76) => class 1

( 0.11, 11.21) => class 1

display class centers:

class 1: ( 3.28, 2.01)

class 2: ( -7.84, 0.66)

class 3: ( 10.36, 4.84)

In this case, kmeans algorithm classifies the samples into three classes. Still, it tries to group the samples generated from the same circle center together. This time, it took only three iterations for 3-means algorithm.

**5-means Algorithm – Dataset 1 (Radius 4)**

Input 1 or 2 to run minimax(1) or kmeans(2): 2

Input the number of k for running kmeans (2 - 7): 5

initial center index:

23 12 11 6 22

display kmeansResult:

number of class: 5

number of iteration: 3

( 6.85, -5.69) => class 4

( 5.43, -6.94) => class 4

( 4.04, -7.22) => class 4

( 6.87, -5.18) => class 4

( 8.05, -7.77) => class 4

( 6.81, -6.53) => class 4

( 6.30, -6.04) => class 4

( 9.06, -6.28) => class 4

( 6.51, -5.23) => class 4

( 3.49, -4.73) => class 4

( -3.54, -4.79) => class 3

( -6.28, -5.71) => class 3

( -4.32, -2.68) => class 3

( -5.59, -6.01) => class 3

( -6.37, -3.32) => class 3

( -4.22, -6.35) => class 3

( -8.57, -4.92) => class 3

( -8.49, -6.87) => class 3

( -4.84, -5.99) => class 3

( -5.64, -6.18) => class 3

( 8.64, 2.35) => class 1

( 9.66, 7.35) => class 1

( 10.87, 6.74) => class 1

( 12.12, 5.44) => class 1

( 11.75, 3.41) => class 1

( 12.47, 5.45) => class 1

( 10.65, 5.46) => class 1

( 8.20, 5.76) => class 1

( 8.35, 3.14) => class 1

( 10.93, 3.34) => class 1

( -9.36, 6.43) => class 2

(-10.61, 5.35) => class 2

( -9.24, 6.02) => class 2

(-11.59, 8.98) => class 2

( -9.02, 5.47) => class 2

(-10.25, 4.75) => class 2

( -9.90, 7.14) => class 2

( -9.13, 8.78) => class 2

( -9.28, 6.93) => class 2

(-10.64, 6.14) => class 2

( 1.33, 9.10) => class 5

( -0.66, 10.68) => class 5

( 2.06, 10.97) => class 5

( -1.15, 9.85) => class 5

( 1.88, 9.09) => class 5

( 0.01, 10.01) => class 5

( 0.75, 10.97) => class 5

( -0.14, 10.14) => class 5

( -1.95, 9.76) => class 5

( 0.11, 11.21) => class 5

display class centers:

class 1: ( 10.36, 4.84)

class 2: ( -9.90, 6.60)

class 3: ( -5.79, -5.28)

class 4: ( 6.34, -6.16)

class 5: ( 0.22, 10.18)

In this case, kmeans algorithm classifies the samples into five classes. With this combination of initial class centers (listed at top), the algorithm is able to classify the samples as we expect. However, most of the time, the algorithm fails to group the samples generated from the same circle center into one cluster. This can be caused by the undesired initial center positions are used, which just converges as long as it finds a ‘balanced’ point. Therefore, we need to test the same sample data with different initial centers many times and pick the best clustering result.

**7-means Algorithm – Dataset 1 (Radius 4)**

Input 1 or 2 to run minimax(1) or kmeans(2): 2

Input the number of k for running kmeans (2 - 7): 7

initial center index:

2 21 6 23 20 11 18

display kmeansResult:

number of class: 7

number of iteration: 3

( 6.85, -5.69) => class 2

( 5.43, -6.94) => class 2

( 4.04, -7.22) => class 0

( 6.87, -5.18) => class 2

( 8.05, -7.77) => class 2

( 6.81, -6.53) => class 2

( 6.30, -6.04) => class 2

( 9.06, -6.28) => class 2

( 6.51, -5.23) => class 2

( 3.49, -4.73) => class 0

( -3.54, -4.79) => class 6

( -6.28, -5.71) => class 6

( -4.32, -2.68) => class 6

( -5.59, -6.01) => class 6

( -6.37, -3.32) => class 6

( -4.22, -6.35) => class 6

( -8.57, -4.92) => class 6

( -8.49, -6.87) => class 6

( -4.84, -5.99) => class 6

( -5.64, -6.18) => class 6

( 8.64, 2.35) => class 4

( 9.66, 7.35) => class 3

( 10.87, 6.74) => class 3

( 12.12, 5.44) => class 3

( 11.75, 3.41) => class 3

( 12.47, 5.45) => class 3

( 10.65, 5.46) => class 3

( 8.20, 5.76) => class 4

( 8.35, 3.14) => class 4

( 10.93, 3.34) => class 3

( -9.36, 6.43) => class 5

(-10.61, 5.35) => class 5

( -9.24, 6.02) => class 5

(-11.59, 8.98) => class 5

( -9.02, 5.47) => class 5

(-10.25, 4.75) => class 5

( -9.90, 7.14) => class 5

( -9.13, 8.78) => class 5

( -9.28, 6.93) => class 5

(-10.64, 6.14) => class 5

( 1.33, 9.10) => class 1

( -0.66, 10.68) => class 1

( 2.06, 10.97) => class 1

( -1.15, 9.85) => class 1

( 1.88, 9.09) => class 1

( 0.01, 10.01) => class 1

( 0.75, 10.97) => class 1

( -0.14, 10.14) => class 1

( -1.95, 9.76) => class 1

( 0.11, 11.21) => class 1

display class centers:

class 1: ( 3.77, -5.97)

class 2: ( 0.22, 10.18)

class 3: ( 6.99, -6.21)

class 4: ( 11.21, 5.31)

class 5: ( 8.40, 3.75)

class 6: ( -9.90, 6.60)

class 7: ( -5.79, -5.28)

With 7-means algorithm, we can the samples generated from two circle centers are each classified into two clusters. This makes sense since we are classifying the samples into 7 classes. Also, 7-means algorithm takes 3 iterations to finish. Overall, even it is hard to get the expected result on the first run, we can still trace the classified result with different number of k to correctly classify the data. For instance, if the samples are classified as different classes with 2-meas and 3-means, then we should know that those samples should really belong to different classes.

**2-means Algorithm – Dataset 2 (Radius 15)**

Input 1 or 2 to run minimax(1) or kmeans(2): 2

Input the number of k for running kmeans (2 - 7): 2

initial center index:

9 24

display kmeansResult:

number of class: 2

number of iteration: 8

( 6.20, -5.56) => class 1

( 5.31, -12.65) => class 1

( 6.65, -3.70) => class 1

( 10.60, -7.65) => class 1

( 14.00, -1.23) => class 1

( 5.56, -14.46) => class 1

( 14.90, -3.27) => class 1

( 7.44, 4.24) => class 1

( 3.14, -10.15) => class 1

( 6.44, -10.32) => class 1

(-18.17, -3.04) => class 2

( -6.01, -6.09) => class 1

(-14.22, -5.54) => class 1

( 6.63, 1.06) => class 1

( -1.67, 6.49) => class 2

( -5.95, -5.03) => class 1

( -6.31, -5.38) => class 1

( -2.34, -17.02) => class 1

( -5.88, -7.64) => class 1

( -5.40, -8.32) => class 1

( 5.66, 11.35) => class 2

( 1.27, 3.85) => class 2

( 13.00, -0.03) => class 1

( 19.45, 0.39) => class 1

( 10.89, 12.29) => class 2

( 9.62, 3.66) => class 1

( 0.05, -0.08) => class 1

( 12.69, -0.09) => class 1

( 12.09, 15.32) => class 2

( 4.75, -1.19) => class 1

(-10.64, 9.30) => class 2

(-19.95, 1.25) => class 2

(-11.17, 7.24) => class 2

( 2.95, 6.93) => class 2

( -9.38, -6.32) => class 1

(-23.37, 12.75) => class 2

(-12.36, 5.58) => class 2

(-15.52, -7.88) => class 1

( -9.04, 5.60) => class 2

(-12.85, 7.22) => class 2

( 0.96, 23.20) => class 2

( 0.03, 9.93) => class 2

( 0.42, 10.51) => class 2

( -2.55, -0.76) => class 1

(-10.88, 4.38) => class 2

( -2.14, 9.39) => class 2

( 12.97, 7.25) => class 1

( -2.72, 9.90) => class 2

( 9.10, 11.32) => class 2

( 0.21, 10.41) => class 2

display class centers:

class 1: ( 3.07, -4.42)

class 2: ( -4.15, 8.69)

Pretty chaotic classifying result. It takes more number of iteration compare to others.

**3-means Algorithm – Dataset 2 (Radius 15)**

Input 1 or 2 to run minimax(1) or kmeans(2): 2

Input the number of k for running kmeans (2 - 7): 3

initial center index:

29 22 11

display kmeansResult:

number of class: 3

number of iteration: 3

( 6.20, -5.56) => class 1

( 5.31, -12.65) => class 1

( 6.65, -3.70) => class 1

( 10.60, -7.65) => class 2

( 14.00, -1.23) => class 2

( 5.56, -14.46) => class 1

( 14.90, -3.27) => class 2

( 7.44, 4.24) => class 1

( 3.14, -10.15) => class 1

( 6.44, -10.32) => class 1

(-18.17, -3.04) => class 3

( -6.01, -6.09) => class 3

(-14.22, -5.54) => class 3

( 6.63, 1.06) => class 1

( -1.67, 6.49) => class 1

( -5.95, -5.03) => class 3

( -6.31, -5.38) => class 3

( -2.34, -17.02) => class 3

( -5.88, -7.64) => class 3

( -5.40, -8.32) => class 3

( 5.66, 11.35) => class 1

( 1.27, 3.85) => class 1

( 13.00, -0.03) => class 2

( 19.45, 0.39) => class 2

( 10.89, 12.29) => class 2

( 9.62, 3.66) => class 2

( 0.05, -0.08) => class 1

( 12.69, -0.09) => class 2

( 12.09, 15.32) => class 2

( 4.75, -1.19) => class 1

(-10.64, 9.30) => class 3

(-19.95, 1.25) => class 3

(-11.17, 7.24) => class 3

( 2.95, 6.93) => class 1

( -9.38, -6.32) => class 3

(-23.37, 12.75) => class 3

(-12.36, 5.58) => class 3

(-15.52, -7.88) => class 3

( -9.04, 5.60) => class 3

(-12.85, 7.22) => class 3

( 0.96, 23.20) => class 1

( 0.03, 9.93) => class 1

( 0.42, 10.51) => class 1

( -2.55, -0.76) => class 1

(-10.88, 4.38) => class 3

( -2.14, 9.39) => class 1

( 12.97, 7.25) => class 2

( -2.72, 9.90) => class 1

( 9.10, 11.32) => class 2

( 0.21, 10.41) => class 1

display class centers:

class 1: ( 2.60, 2.30)

class 2: ( 12.66, 3.45)

class 3: (-11.08, -1.05)

Not much to say about this one.

**5-means Algorithm – Dataset 2 (Radius 15)**

Input 1 or 2 to run minimax(1) or kmeans(2): 2

Input the number of k for running kmeans (2 - 7): 5

initial center index:

9 3 2 27 11

display kmeansResult:

number of class: 5

number of iteration: 8

( 6.20, -5.56) => class 2

( 5.31, -12.65) => class 1

( 6.65, -3.70) => class 2

( 10.60, -7.65) => class 2

( 14.00, -1.23) => class 2

( 5.56, -14.46) => class 1

( 14.90, -3.27) => class 2

( 7.44, 4.24) => class 4

( 3.14, -10.15) => class 1

( 6.44, -10.32) => class 1

(-18.17, -3.04) => class 5

( -6.01, -6.09) => class 1

(-14.22, -5.54) => class 5

( 6.63, 1.06) => class 2

( -1.67, 6.49) => class 3

( -5.95, -5.03) => class 1

( -6.31, -5.38) => class 1

( -2.34, -17.02) => class 1

( -5.88, -7.64) => class 1

( -5.40, -8.32) => class 1

( 5.66, 11.35) => class 4

( 1.27, 3.85) => class 3

( 13.00, -0.03) => class 2

( 19.45, 0.39) => class 2

( 10.89, 12.29) => class 4

( 9.62, 3.66) => class 4

( 0.05, -0.08) => class 1

( 12.69, -0.09) => class 2

( 12.09, 15.32) => class 4

( 4.75, -1.19) => class 2

(-10.64, 9.30) => class 5

(-19.95, 1.25) => class 5

(-11.17, 7.24) => class 5

( 2.95, 6.93) => class 3

( -9.38, -6.32) => class 1

(-23.37, 12.75) => class 5

(-12.36, 5.58) => class 5

(-15.52, -7.88) => class 5

( -9.04, 5.60) => class 5

(-12.85, 7.22) => class 5

( 0.96, 23.20) => class 3

( 0.03, 9.93) => class 3

( 0.42, 10.51) => class 3

( -2.55, -0.76) => class 1

(-10.88, 4.38) => class 5

( -2.14, 9.39) => class 3

( 12.97, 7.25) => class 4

( -2.72, 9.90) => class 3

( 9.10, 11.32) => class 4

( 0.21, 10.41) => class 3

display class centers:

class 1: ( -1.79, -8.02)

class 2: ( 10.89, -2.13)

class 3: ( -0.08, 10.07)

class 4: ( 9.68, 9.35)

class 5: (-14.38, 3.35)

Not much to say about this one.

**7-means Algorithm – Dataset 2 (Radius 15)**

Input 1 or 2 to run minimax(1) or kmeans(2): 2

Input the number of k for running kmeans (2 - 7): 7

initial center index:

19 3 2 4 0 25 26

display kmeansResult:

number of class: 7

number of iteration: 7

( 6.20, -5.56) => class 2

( 5.31, -12.65) => class 5

( 6.65, -3.70) => class 2

( 10.60, -7.65) => class 2

( 14.00, -1.23) => class 4

( 5.56, -14.46) => class 5

( 14.90, -3.27) => class 4

( 7.44, 4.24) => class 3

( 3.14, -10.15) => class 5

( 6.44, -10.32) => class 2

(-18.17, -3.04) => class 1

( -6.01, -6.09) => class 1

(-14.22, -5.54) => class 1

( 6.63, 1.06) => class 3

( -1.67, 6.49) => class 3

( -5.95, -5.03) => class 1

( -6.31, -5.38) => class 1

( -2.34, -17.02) => class 5

( -5.88, -7.64) => class 1

( -5.40, -8.32) => class 1

( 5.66, 11.35) => class 6

( 1.27, 3.85) => class 3

( 13.00, -0.03) => class 4

( 19.45, 0.39) => class 4

( 10.89, 12.29) => class 6

( 9.62, 3.66) => class 4

( 0.05, -0.08) => class 3

( 12.69, -0.09) => class 4

( 12.09, 15.32) => class 6

( 4.75, -1.19) => class 3

(-10.64, 9.30) => class 7

(-19.95, 1.25) => class 7

(-11.17, 7.24) => class 7

( 2.95, 6.93) => class 3

( -9.38, -6.32) => class 1

(-23.37, 12.75) => class 7

(-12.36, 5.58) => class 7

(-15.52, -7.88) => class 1

( -9.04, 5.60) => class 7

(-12.85, 7.22) => class 7

( 0.96, 23.20) => class 6

( 0.03, 9.93) => class 6

( 0.42, 10.51) => class 6

( -2.55, -0.76) => class 3

(-10.88, 4.38) => class 7

( -2.14, 9.39) => class 6

( 12.97, 7.25) => class 4

( -2.72, 9.90) => class 6

( 9.10, 11.32) => class 6

( 0.21, 10.41) => class 6

display class centers:

class 1: ( -9.65, -6.14)

class 2: ( 7.48, -6.81)

class 3: ( 2.36, 2.57)

class 4: ( 13.80, 0.95)

class 5: ( 2.92, -13.57)

class 6: ( 3.45, 12.36)

class 7: (-13.78, 6.67)

Not much to say about this one. Overall, the testing result for the second dataset are hard to analyze. However, I do believe the impact of initial centers is greater than running kmeans algorithm on the first dataset.

**Source Code**

/\* CS6200 Che Shian Hung 3/12/2018

Programming Assignment 3

Purpose: This program implements minimax and kmeans classifying algorithms to classify

randomly generated samples, and display the testing results. The samples are

generated from five circle centers. Each circle center is used to generated

10 samples within a given radius. The user can test with different radius

in the interactive interface. The user can also test with different numbers

of algorithm setting to perform testing through interactive interface. Each

program run will generate different set of testing samples. However, we can

test with the sampe data again and agian in the terminal. In this program,

the distance between two points is the similarity measure for both algorithm.

For minimax algo., it will find a potential class center in each iteration,

and include the new class center into the result when it passes the test

with the previous class centers, and the testing result and stats are captured

by several global variables. For kmeans algo, before the algorithm runs, the

user can indicate the number of classes given for kmeans algorithm. Initially,

the algorithm will randomly select the same number of samples as initial class

centers, and reclassifying the samples and calculating the class centers again

and again until the result converge. The output of the each test result is

format as how we generate data. Since the data is generated for each circle

center sequencially, the samples for each circle center are separated from

each other in the test result.

\*/

#define \_USE\_MATH\_DEFINES

// Import libraries and constants

#include<iostream>

#include<stdlib.h>

#include<time.h>

#include<cmath>

#include<math.h>

#include<string>

#define CENTER\_NUM 5 // Number of circle centers

#define DIMENSION 2 // Dimensionality for each sample

#define SAMPLE\_SIZE 10 // Number of sample generate for each class

#define MAX\_ITERATION 1000 // Iteration limit for training with backpropagation network

using namespace std;

// Linklist node to capture the runtime generated class center in minimax algorithm

struct centerIndexNode {

int index;

centerIndexNode\* next;

};

const double sphereCenter[CENTER\_NUM][DIMENSION] = {{6, -6}, { -6, -6}, {10, 6}, {-10, 6}, {0, 10}}; // Hard coded circle centers

double samples[CENTER\_NUM \* SAMPLE\_SIZE][DIMENSION]; // Sample for all classes

int minimaxClassNum = 0; // Number of classes after applying minimax

int minimaxResultClass[CENTER\_NUM \* SAMPLE\_SIZE]; // Arrays that capture the classifying result after applying minimax

int\* minimaxCenterIndex = NULL; // Arrays of the classes center index from the samples after applying minimax

double\*\* minimaxClassCenters = NULL; // 2D arrays of finalized class centers after applying minimax

int kmeansClassNum; // Number of classes after applying kmeans

int kmeansIterationNum; // Number of iteration took from previous kmeans

int kmeansResultClass[CENTER\_NUM \* SAMPLE\_SIZE]; // Arrays that capture the classifying result after applying kmeans

double\*\* kmeansClassCenters = NULL; // 2D arrays of finalized class centers after applying kmeans

void generateAllSamples(int radius); // Given a radius, generate all samples randomly within each class center

void displayAllSamples(); // Output all samples

void displayMinimaxResult(); // Display testing result after applying minimax

void displayKmeansResult(); // Display testing result after applying kmeans

void deletePointers(); // Delete global pointers

int userInput(int mode); // Ask user for input along with different question modes and return an integer answer

void minimax(); // Perfrom minimax algo.

void kmeans(int cNum); // Perform kmeans algo.

double distance(double\* a, double\* b); // Compute the distance between two points

centerIndexNode\* appendToIndex(centerIndexNode\* list, int index); // Append a new centerIndexNode to the linklist while running minimax

int main() {

srand(time(NULL));

generateAllSamples(userInput(1)); // Generate sample data

displayAllSamples(); // Display sample data

do {

if (userInput(2) == 1) { // Ask user to decide which algorithm to run

minimax(); // Run minimax

displayMinimaxResult(); // Display minimax result

}

else {

kmeans(userInput(3)); // Run kmeans

displayKmeansResult(); // Display kmeans result

}

} while (userInput(4) == 2); // Ask user to see if continue testing or exit the program

deletePointers(); // Delete all pointers

system("pause");

return 0;

}

void generateAllSamples(int radius) {

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++) {

double theta = rand() % 6282 / double(1000);

double r = rand() % (radius \* 1000) / double(1000);

samples[i][0] = sphereCenter[i / SAMPLE\_SIZE][0] + r \* cos(theta);

samples[i][1] = sphereCenter[i / SAMPLE\_SIZE][1] + r \* sin(theta);

}

}

void displayAllSamples() {

printf("\ndisplay all samples:\n");

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++)

printf("(%6.2f, %6.2f)\n", samples[i][0], samples[i][1]);

printf("\n\n");

}

void displayMinimaxResult() {

printf("\ndisplay minimaxResult:\n\n");

printf("number of class: %d\n\n", minimaxClassNum);

printf("number of iteration: %d\n\n", minimaxClassNum - 1);

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++) {

printf("(%6.2f, %6.2f) => class %d\n", samples[i][0], samples[i][1], minimaxResultClass[i]);

if (i % SAMPLE\_SIZE == SAMPLE\_SIZE - 1) printf("\n");

}

printf("\n\n");

printf("display class centers:\n\n");

for (int i = 0; i < minimaxClassNum; i++) {

printf("class %d: (%6.2f, %6.2f), original index: %d\n", i + 1, minimaxClassCenters[i][0], minimaxClassCenters[i][1], minimaxCenterIndex[i]);

}

printf("\n\n");

}

void displayKmeansResult() {

printf("display kmeansResult:\n\n");

printf("number of class: %d\n\n", kmeansClassNum);

printf("number of iteration: %d\n\n", kmeansIterationNum);

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++) {

printf(" (%6.2f, %6.2f) => class %d\n", samples[i][0], samples[i][1], kmeansResultClass[i]);

if (i % SAMPLE\_SIZE == SAMPLE\_SIZE - 1) printf("\n");

}

printf("\n\n");

printf("display class centers:\n\n");

for (int i = 0; i < kmeansClassNum; i++) {

printf("class %d: (%6.2f, %6.2f)\n", i + 1, kmeansClassCenters[i][0], kmeansClassCenters[i][1]);

}

printf("\n\n");

}

void minimax() {

// Initialize variables

int cNum = 1;

double centerDistanceTotal = 0;

bool done = false;

bool isCenter[CENTER\_NUM \* SAMPLE\_SIZE];

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++) {

isCenter[i] = false;

minimaxResultClass[i] = 0;

}

isCenter[0] = true;

// Contruct centerIndexNode linklist to capture runtime generated class centers

centerIndexNode\* indexHead = new centerIndexNode();

indexHead->index = 0;

indexHead->next = NULL;

// Each minimax iteration

while (!done && cNum < MAX\_ITERATION){

double maxResult[2] = { 0, -1 }; // Capture the result for each iteration

// Compute and compare distance between each sample and classCenters

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++) {

if (!isCenter[i]) {

centerIndexNode\* h = indexHead;

double minResult[2];

minResult[0] = distance(samples[i], samples[h->index]);

minResult[1] = (double) i;

while (h->next) {

h = h->next;

double d = distance(samples[i], samples[h->index]);

if (d < minResult[0]) {

minResult[0] = d;

minResult[1] = i;

}

}

h = NULL;

if (minResult[0] > maxResult[0]) {

maxResult[0] = minResult[0];

maxResult[1] = minResult[1];

}

}

}

// Check if there is a new cluster center

if (cNum != 1 && maxResult[0] <= centerDistanceTotal / (cNum \* (cNum - 1) / 2) / 2) {

done = true;

}

// If yes, add the new cluster center and update stats

if (!done) {

centerIndexNode\* h = indexHead;

for (int i = 0; i < cNum; i++) {

centerDistanceTotal += distance(samples[h->index], samples[(int) maxResult[1]]);

h = h->next;

}

h = NULL;

cNum++;

indexHead = appendToIndex(indexHead, (int) maxResult[1]);

isCenter[(int) maxResult[1]] = true;

}

}

// Put cNum linked list into an array and build the array for calculating cluster centers

if (minimaxCenterIndex) delete[] minimaxCenterIndex;

minimaxCenterIndex = new int[cNum];

double\*\* classTotalPosition = new double\*[cNum];

centerIndexNode\* indexTracer = indexHead;

for (int i = 0; i < cNum; i++) {

classTotalPosition[i] = new double[2];

classTotalPosition[i][0] = 0;

classTotalPosition[i][1] = 0;

}

for (int i = 0; i < cNum; i++) {

classTotalPosition[i][0] += samples[indexTracer->index][0];

classTotalPosition[i][1] += samples[indexTracer->index][1];

minimaxCenterIndex[i] = indexTracer->index;

minimaxResultClass[indexTracer->index] = i + 1;

indexTracer = indexTracer->next;

}

// Classifying samples based on distances and sum up cluster positions for calculating cluster centers

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++) {

if (minimaxResultClass[i] == 0) {

int minIndex = 0;

double min = distance(samples[i], samples[minimaxCenterIndex[0]]);

for (int j = 1; j < cNum; j++) {

double d = distance(samples[i], samples[minimaxCenterIndex[j]]);

if (d < min) {

min = d;

minIndex = j;

}

}

minimaxResultClass[i] = minIndex + 1;

classTotalPosition[minIndex][0] += samples[i][0];

classTotalPosition[minIndex][1] += samples[i][1];

}

}

// Assign cluster centers

minimaxClassNum = cNum;

if (minimaxClassCenters) delete[] minimaxClassCenters;

minimaxClassCenters = new double\*[cNum];

for (int i = 0; i < cNum; i++) {

minimaxClassCenters[i] = new double[2];

minimaxClassCenters[i][0] = classTotalPosition[i][0] / (CENTER\_NUM \* SAMPLE\_SIZE / cNum);

minimaxClassCenters[i][1] = classTotalPosition[i][1] / (CENTER\_NUM \* SAMPLE\_SIZE / cNum);

}

// Delete pointers

indexTracer = indexHead;

while (indexTracer) {

indexHead = indexTracer;

indexTracer = indexTracer->next;

delete indexHead;

}

indexTracer = NULL;

indexHead = NULL;

for (int i = 0; i < cNum; i++)

delete[] classTotalPosition[i];

delete[] classTotalPosition;

}

void kmeans(int cNum) {

// Initialize variables and randomly select initial class centers

int iteration = 0;

bool done = false;

int\* cCounter = new int[cNum];

double\*\* cCenters = new double\*[cNum];

double\*\* cPreCenters = new double\*[cNum];

double\*\* cCenterTotal = new double\*[cNum];

printf("initial center index: \n");

for (int i = 0; i < cNum; i++) {

bool repeat = true;

while (repeat) {

repeat = false;

int r = rand() % 30;

for (int j = 0; j < i; j++) {

if (cCounter[j] == r) {

repeat = true;

break;

}

}

if(!repeat) cCounter[i] = r;

}

cCenters[i] = new double[2];

cPreCenters[i] = new double[2];

cCenterTotal[i] = new double[2];

cCenters[i][0] = samples[cCounter[i]][0];

cCenters[i][1] = samples[cCounter[i]][1];

cPreCenters[i][0] = samples[cCounter[i]][0];

cPreCenters[i][1] = samples[cCounter[i]][1];

cCenterTotal[i][0] = 0;

cCenterTotal [i][1] = 0;

printf("%d ", cCounter[i]);

}

printf("\n\n");

for (int i = 0; i < cNum; i++) cCounter[i] = 0;

if (kmeansClassCenters) {

for (int i = 0; i < kmeansClassNum; i++)

delete[] kmeansClassCenters[i];

delete[] kmeansClassCenters;

kmeansClassCenters = new double\*[cNum];

}

else kmeansClassCenters = new double\*[cNum];

kmeansClassNum = cNum;

// Each kmeans iteration

while (!done && iteration < MAX\_ITERATION) {

// Initialize variables

for (int i = 0; i < cNum; i++) {

cCounter[i] = 0;

cCenterTotal[i][0] = 0;

cCenterTotal[i][1] = 0;

}

// Classify samples and sum up stats for eahc class

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++) {

int index = 0;

double min = distance(samples[i], cCenters[0]);

for (int j = 1; j < cNum; j++) {

double d = distance(samples[i], cCenters[j]);

if (d < min) {

min = d;

index = j;

}

}

cCounter[index]++;

cCenterTotal[index][0] += samples[i][0];

cCenterTotal[index][1] += samples[i][1];

}

// Calculate class centers and check with previous class centers

done = true;

for (int i = 0; i < cNum; i++) {

cCenters[i][0] = cCenterTotal[i][0] / cCounter[i];

cCenters[i][1] = cCenterTotal[i][1] / cCounter[i];

if (done && (cCenters[i][0] != cPreCenters[i][0] || cCenters[i][1] != cPreCenters[i][1]))

done = false;

}

// Update previous class centers if the iteration is not done

if (!done) {

for (int i = 0; i < cNum; i++) {

cPreCenters[i][0] = cCenters[i][0];

cPreCenters[i][1] = cCenters[i][1];

}

}

iteration++;

}

// Assign kmeans stats result

kmeansIterationNum = iteration;

for (int i = 0; i < cNum; i++) {

kmeansClassCenters[i] = new double[2];

kmeansClassCenters[i][0] = cCenters[i][0];

kmeansClassCenters[i][1] = cCenters[i][1];

}

// Classify each sample since we did not store the result while running

for (int i = 0; i < CENTER\_NUM \* SAMPLE\_SIZE; i++) {

int index = 0;

double min = distance(samples[i], cCenters[0]);

for (int j = 1; j < cNum; j++) {

double d = distance(samples[i], cCenters[j]);

if (d < min) {

min = d;

index = j;

}

}

kmeansResultClass[i] = index + 1;

}

// Delete pointers

delete[] cCounter;

for (int i = 0; i < cNum; i++) {

delete[] cCenters[i];

delete[] cPreCenters[i];

delete[] cCenterTotal[i];

}

delete[] cCenters;

delete[] cPreCenters;

delete[] cCenterTotal;

cCounter = NULL;

cCenters = NULL;

cPreCenters = NULL;

cCenterTotal = NULL;

}

double distance(double\* a, double\* b) {

return sqrt(pow(a[0] - b[0], 2) + pow(a[1] - b[1], 2));

}

centerIndexNode\* appendToIndex(centerIndexNode\* list, int index) {

centerIndexNode\* newNode = new centerIndexNode();

newNode->index = index;

newNode->next = NULL;

centerIndexNode\* h = list;

if (!h) return newNode;

while (h->next) h = h->next;

h->next = newNode;

return list;

}

void deletePointers() {

if (minimaxClassCenters) {

for (int i = 0; i < minimaxClassNum; i++) {

delete[] minimaxClassCenters[i];

}

delete[] minimaxClassCenters;

}

if (kmeansClassCenters) {

for (int i = 0; i < kmeansClassNum; i++) {

delete[] kmeansClassCenters[i];

}

delete[] kmeansClassCenters;

}

if(minimaxCenterIndex)

delete[] minimaxCenterIndex;

}

int userInput(int mode) {

// mode 1: ask for radius for data gen

// mode 2: ask for using minimax or kmeans

// mode 3: ask for k for kmeans algo

// mode 4: ask for exit

string input;

switch (mode){

case 1:

printf("For data generation, input the radius of the circle (1 - 15): ");

break;

case 2:

printf("Input 1 or 2 to run minimax(1) or kmeans(2): ");

break;

case 3:

printf("Input the number of k for running kmeans (2 - 7): ");

break;

case 4:

printf("Done testing and want to exit the program? (y / n): ");

break;

}

cin >> input;

switch (mode) {

case 1:

while (input != "4" && input != "15" && input != "1" && input != "2" && input != "3" && input != "5" && input != "6" && input != "7" && input != "8" && input != "9" && input != "10" && input != "11" && input != "12" && input != "13" && input != "14") {

printf("Incorrect input. For data generation, please input the number of the radius (1 - 15): ");

cin >> input;

}

break;

case 2:

while (input != "1" && input != "2") {

printf("Incorrect input. Please enter 1 to run minimax algo. or enter 2 to run kmeans algo (1 / 2): ");

cin >> input;

}

break;

case 3:

while (input != "2" && input != "3" && input != "5" && input != "7" && input != "4" && input != "6") {

printf("Incorrect input. Please enter number 2 to 7 to determine the number of classes in kmeas algo. (2 - 7): ");

cin >> input;

}

break;

case 4:

while (input != "y" && input != "n" && input != "Y" && input != "N" && input != "yes" && input != "no" && input != "Yes" && input != "No") {

printf("Incorrect input. Please enter 'y' to exit the program, or enter 'n' to keep testing (y / n): ");

cin >> input;

}

if (input == "y" || input == "Y" || input == "yes" || input == "Yes") input = "1";

else input = "2";

break;

}

return stoi(input);

}