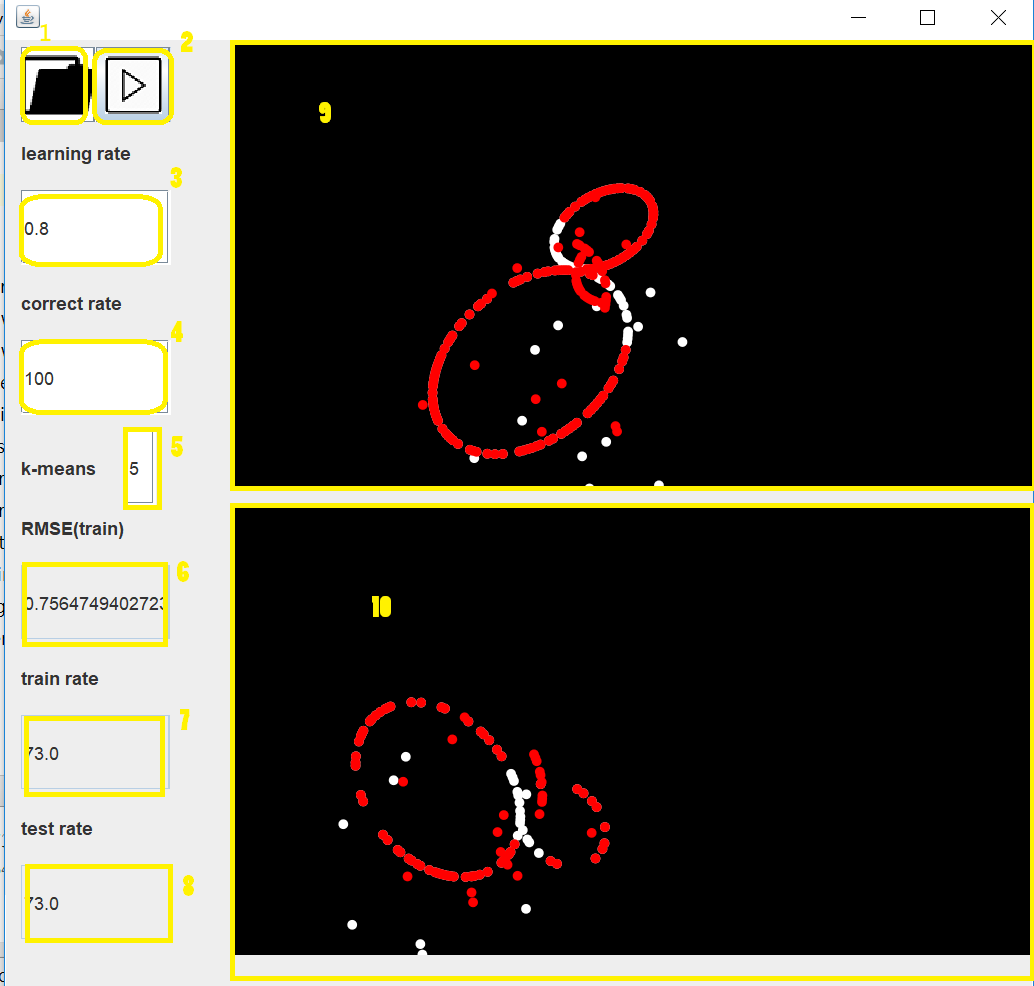
104502002 朱熙媛 HW2

1. 實做的網路架構:RBFN
2. 程式執行說明

執行檔位置:out->artifacts->hw2\_jar->lib->hw2.jar

1. 
2. 點選此鈕選取檔案
3. 點選此鈕開始訓練(先確認是否已輸入終止條件、群數和學習率)
4. 輸入學習率
5. 輸入終止條件(辨別率)
6. 輸入K-MEANS的群數
7. 顯示TRAIN DATA的RMSE
8. 顯示TRAINDATA辨識率
9. 顯示TESTDATA辨識率
10. 顯示TRAINDATA圖形
11. 顯示TESTDATA圖形

操作說明:可用滑鼠旋轉圖形，alt+滑鼠左鍵拖移可縮放圖形(一開始圖片會很大，無法全部顯示，需先進行縮放)

(圖形說明: 資料的第一個數值當X值 第二個數值當Y值 第三個數值當Z值 畫於CANVAS3D上，原始點為白色，預測的點為紅色，若預測的點=原始點，紅色會蓋過白色)

這個地方有個BUG:CANVAS3D不能重複畫超過32次，會出現RuntimeException，需重新開啟

1. 程式簡介

Startup Class:為初始介面，輸入學習率、群數、終止條件並按下執行鈕後，呼叫下一個class

Kmeans Class:利用k-means演算法將資料分類

Drawtrain Class:將traindata畫在Canvas3D上

Drawtest Class:將testdata畫在Canvas3D上

Traindata Class:訓練資料集，調整w和 θ

Correctness class:計算rmse

Testdata class:用調整過後的w和θ 計算testdata的輸出

1. 重點程式碼說明

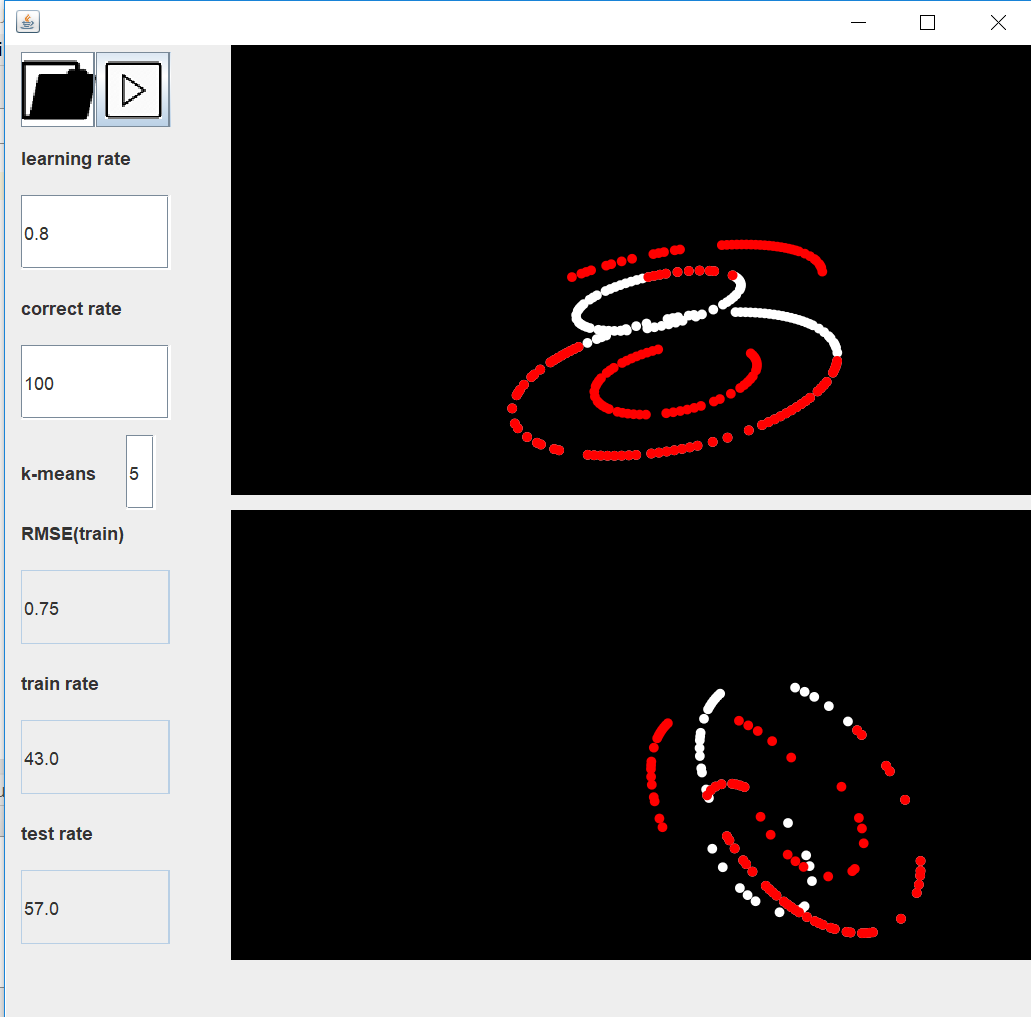
In kmeans class:

**public void** inicenter(){  
 **boolean**[] find = **new boolean**[20];  
 **for**(**int** i=0;i<**kmean**;i++){  
 find[i] = **false**;  
 }  
 Random ran = **new** Random();  
 **int** start1 = **kmean**/2;  
 **for**(**int** i=0;i<start1;i++){ *//為避免都找到同一群的點當中心點* **while**(find[i] == **false**){  
 **int** a = ran.nextInt(**traintime**); //隨機選取點當中心點  
 **if**(**traindata**[a][2] == **n1**){  
 **for**(**int** j = 0;j<3;j++){  
 **center**[i][j] = **traindata**[a][j];  
 }  
 find[i] = **true**;  
 }  
 **else** find[i] = **false**;  
 }  
 }  
 **for**(**int** i=start1;i<**kmean**;i++){*//為避免都找到同一群的點當中心點*  
 **while**(find[i] == **false**){  
 **int** a = ran.nextInt(**traintime**); //隨機選取點當中心點  
 **if**(**traindata**[a][2] == **n2**){  
 **for**(**int** j = 0;j<3;j++){  
 **center**[i][j] = **traindata**[a][j];  
 }  
 find[i] = **true**;  
 }  
 **else** find[i] = **false**;  
 }  
 }  
  
 classify();  
  
}**public void** classify(){  
 **double** mu = 1E-10;  
 **double** difference = 1000;  
 **int** time = 10;*//疊代次數* **int** t = 0;  
 **for**(**int** i=0;i<**kmean**;i++){  
 **for**(**int** j=0;j<2;j++){  
 **weight**[i][0][j] = **center**[i][j];  
 }  
 }  
  
  
 **while**(difference >mu &&t <time){  
 **int** []index = **new int**[20];  
 **double** []sumx = **new double**[20];  
 **double** []sumy = **new double**[20];  
 **double** []meanx = **new double**[20];  
 **double** []meany = **new double**[20];  
 **double** []differencee = **new double**[20];  
 **int** [][]cen = **new int**[20][3000];  
 **double** []distance = **new double**[20];  
 **double** min = 100;  
  
 **for**(**int** i=0;i<**kmean**;i++){  
 index[i] = 0;  
 }  
 **for**(**int** i=0;i<**traintime**;i++){//依照歐基里德距離決定要分配到哪群  
 **for**(**int** j=0;j<**kmean**;j++){  
 **double** xdifference = **weight**[j][t][0]-**traindata**[i][0];  
 **double** ydifference = **weight**[j][t][1]-**traindata**[i][1];  
 distance[j] = Math.*sqrt*(Math.*pow*(xdifference,2)+Math.*pow*(ydifference,2));  
 **if**(distance[j]<min){  
 min = distance[j];  
 **traindata**[i][3] = j;  
 }  
 }  
 **int** a = (**int**)(**traindata**[i][3]);  
 cen[a][index[a]] = i;  
 index[a]++;  
 }  
  
 **for**(**int** i=0;i<**kmean**;i++){//調整中心點位置  
 **for**(**int** j=0;j<index[i];j++){  
 sumx[i] = sumx[i]+**traindata**[(cen[i][j])][0];  
 sumy[i] = sumy[i]+**traindata**[(cen[i][j])][1];  
 }  
 meanx[i] = sumx[i]/(index[i]+1);  
 meany[i] = sumy[i]/(index[i]+1);  
 **weight**[i][t+1][0] = meanx[i];  
 **weight**[i][t+1][1] = meany[i];  
 differencee[i] = Math.*sqrt*(Math.*pow*((**weight**[i][t+1][0]-**weight**[i][t][0]),2)+Math.*pow*(**weight**[i][t+1][1]-**weight**[i][t][1],2));  
 }  
 **double** max = 0;  
 **for**(**int** i=0;i<**kmean**;i++){  
 **if**(differencee[i]>max){  
 max = differencee[i];  
 }  
 }  
 difference = max;  
  
 t++;  
  
 }  
 **float** [][]c = **new float**[20][3];  
 **for**(**int** i= 0;i<**kmean**;i++){  
 **for**(**int** j=0;j<2;j++){  
 c[i][j] = (**float**)**weight**[i][t][j];  
 }  
 c[i][2] = 0;  
 }  
  
}

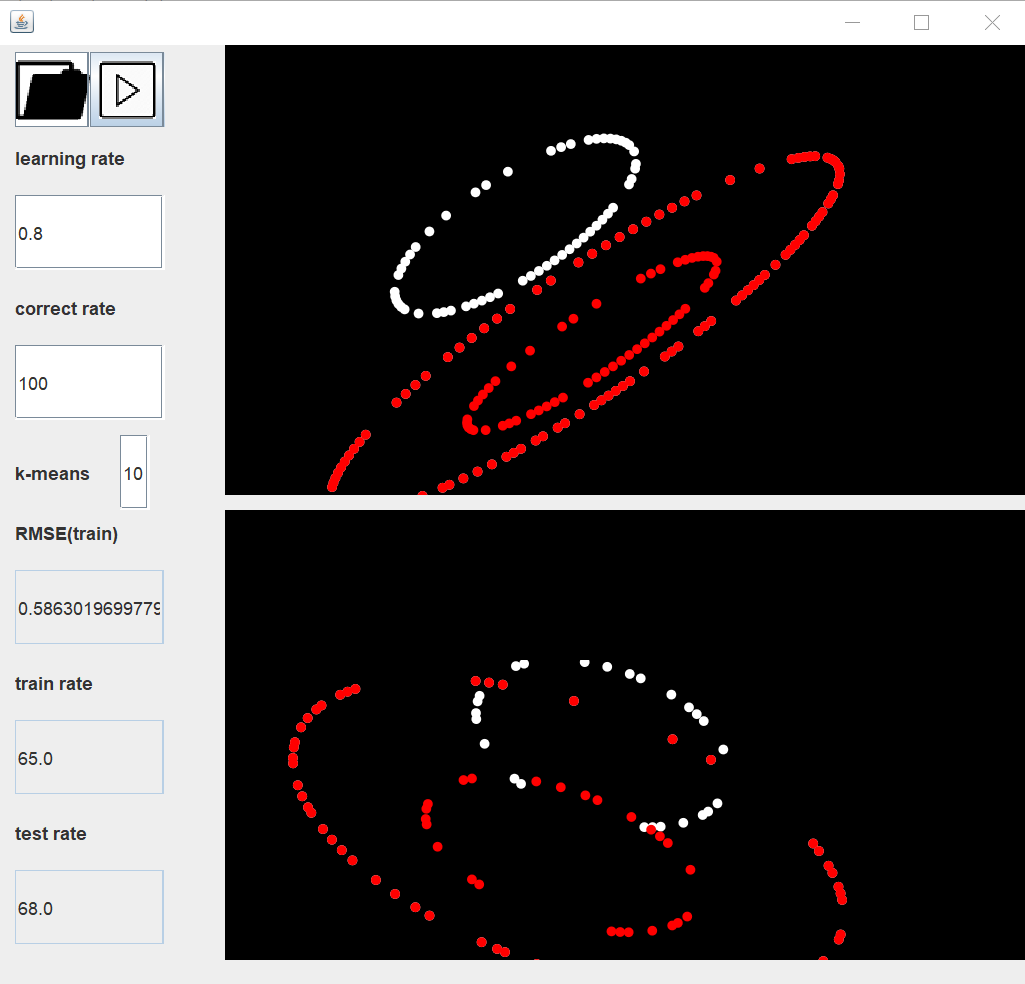
in traindata class:

**public void** train(){  
 **int** time = 0;  
 **int** time1 = 0;  
 calvarx();*//計算變異數* calvary();//計算變異數  
 **while**(time1<10000){  
  
 **if**(time>=**traintime**){  
 time= time-**traintime**;  
 }  
 sgn(time);//計算基底函數的值  
 **if**(correct(time1)>**max**){ //每次調整完鍵結值就算一次train correctrate，將最好的結果存起來  
 **max** = correct(time1);  
 **correctrate** = correct(time1);   
 **for**(**int** i=0;i<2;i++){  
 **finalweight**[i] = **weight**[time1][i];  
 }  
 **finalsita** = **sita**[time1];  
 }  
 **if**(**max**>=**corr**){  
 **break**;  
 }  
 **for**(**int** i=0;i<2;i++){//調整鍵結值  
 **weight**[time1+1][i] = **weight**[time1][i]+**learnrate**\*(**traindata**[time][2]-output(time,time1))\***s**[i];  
 }  
  
  
 **sita**[time1+1] = **sita**[time1]+**learnrate**\*(**traindata**[time][2]-output(time,time1));//調整theta值  
 time1++;  
 time++;  
  
 }

1. 實驗結果
2. 2Ccircle1

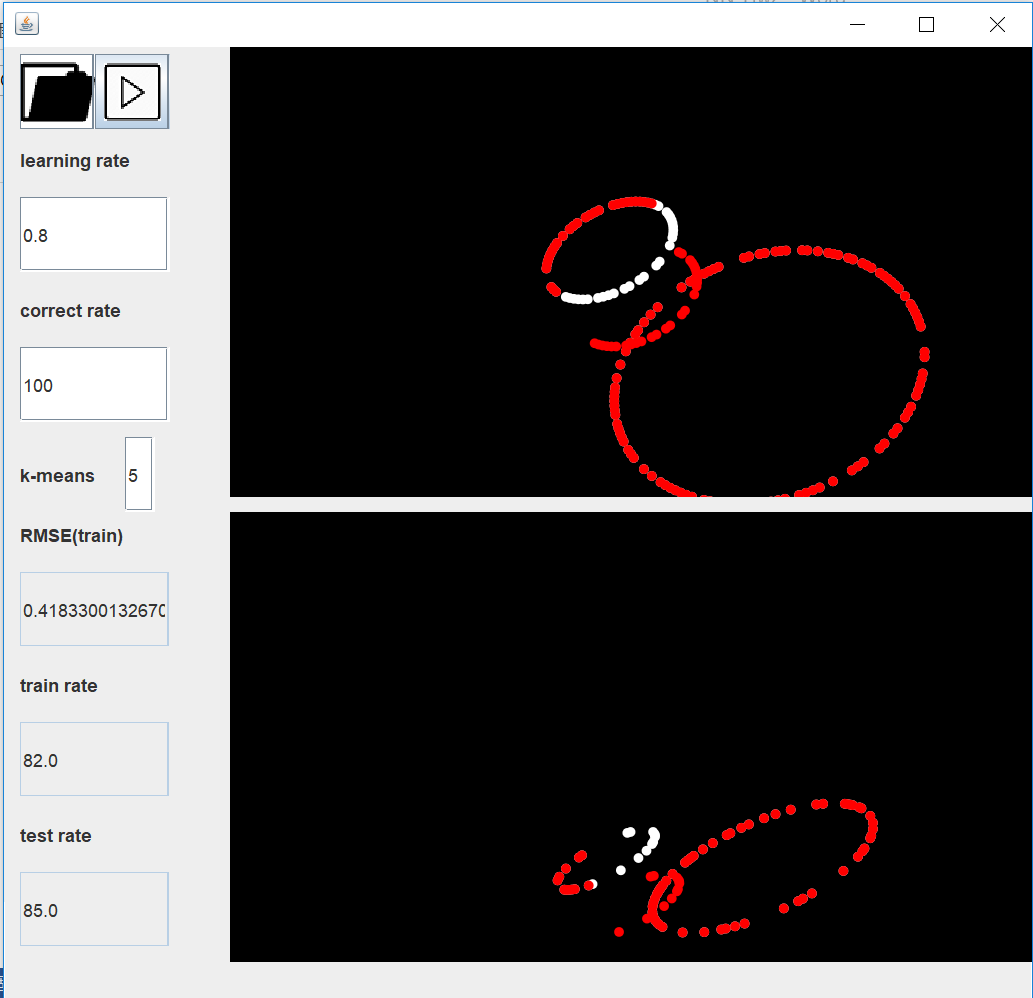
kmeans=5

Kmeans = 10

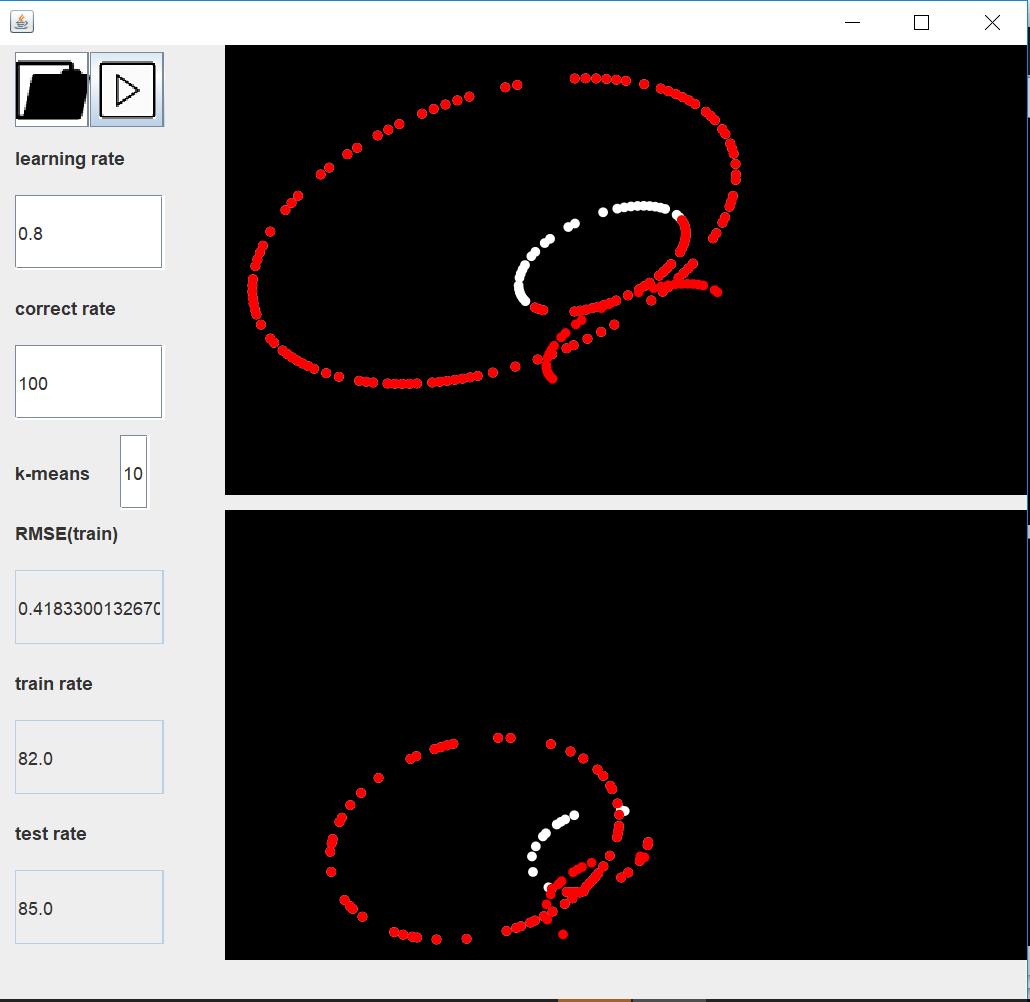


1. 2Circle1

Kmeans = 5

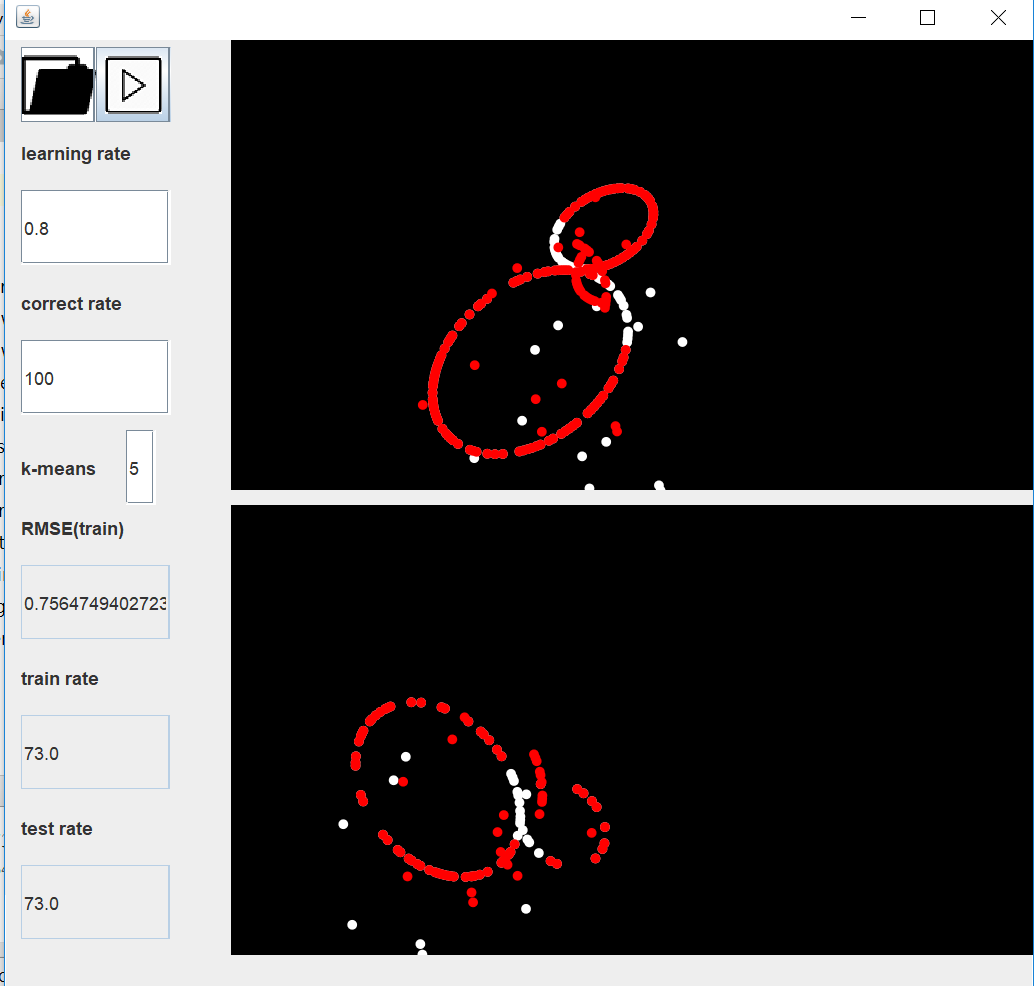


Kmeans = 10

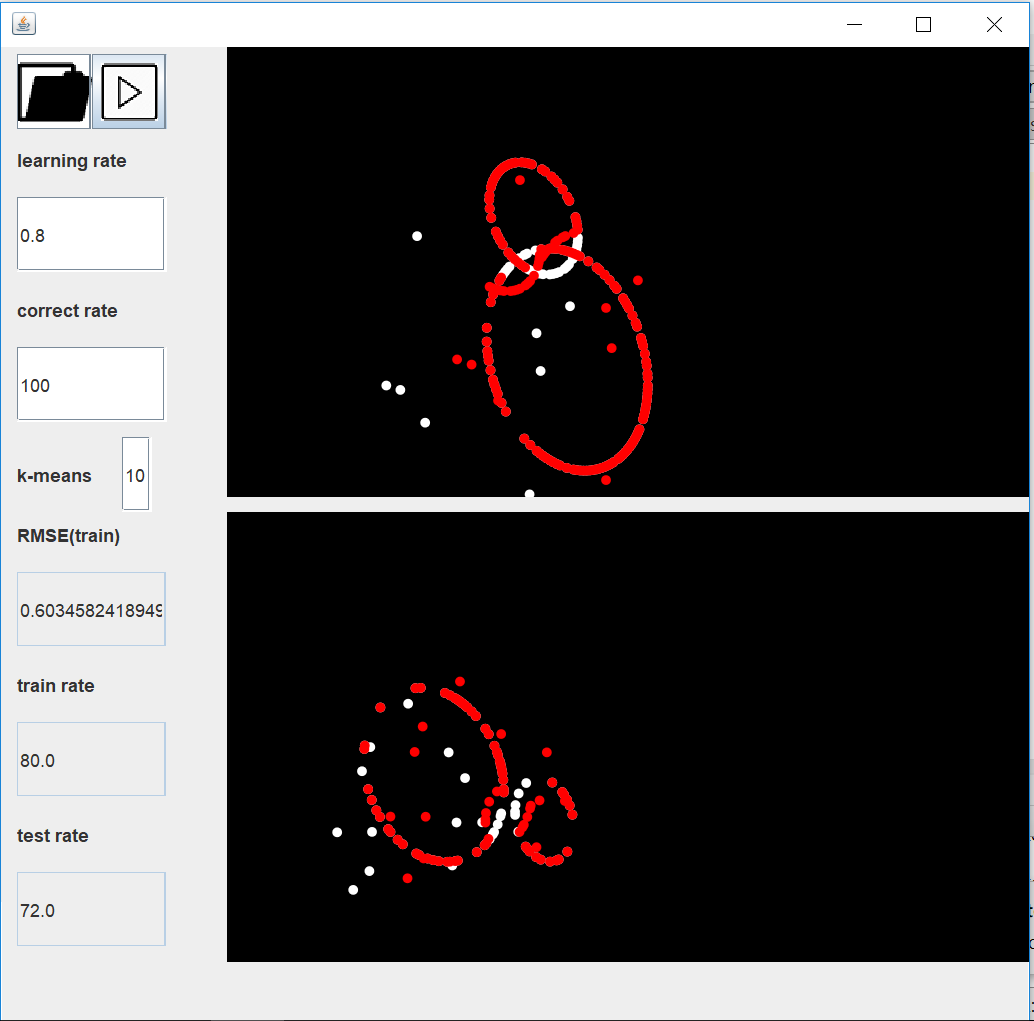


1. 2Circle2

Kmeans = 5

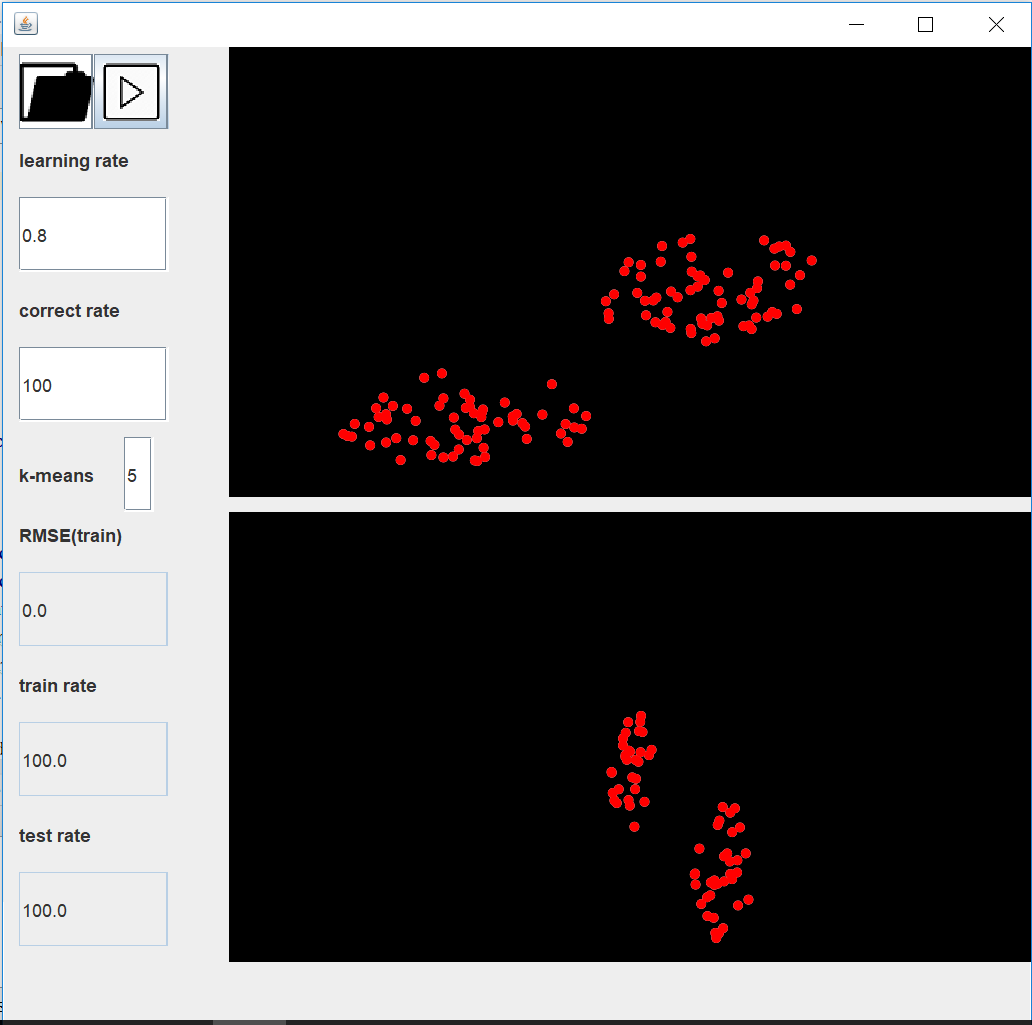


Kmeans = 10

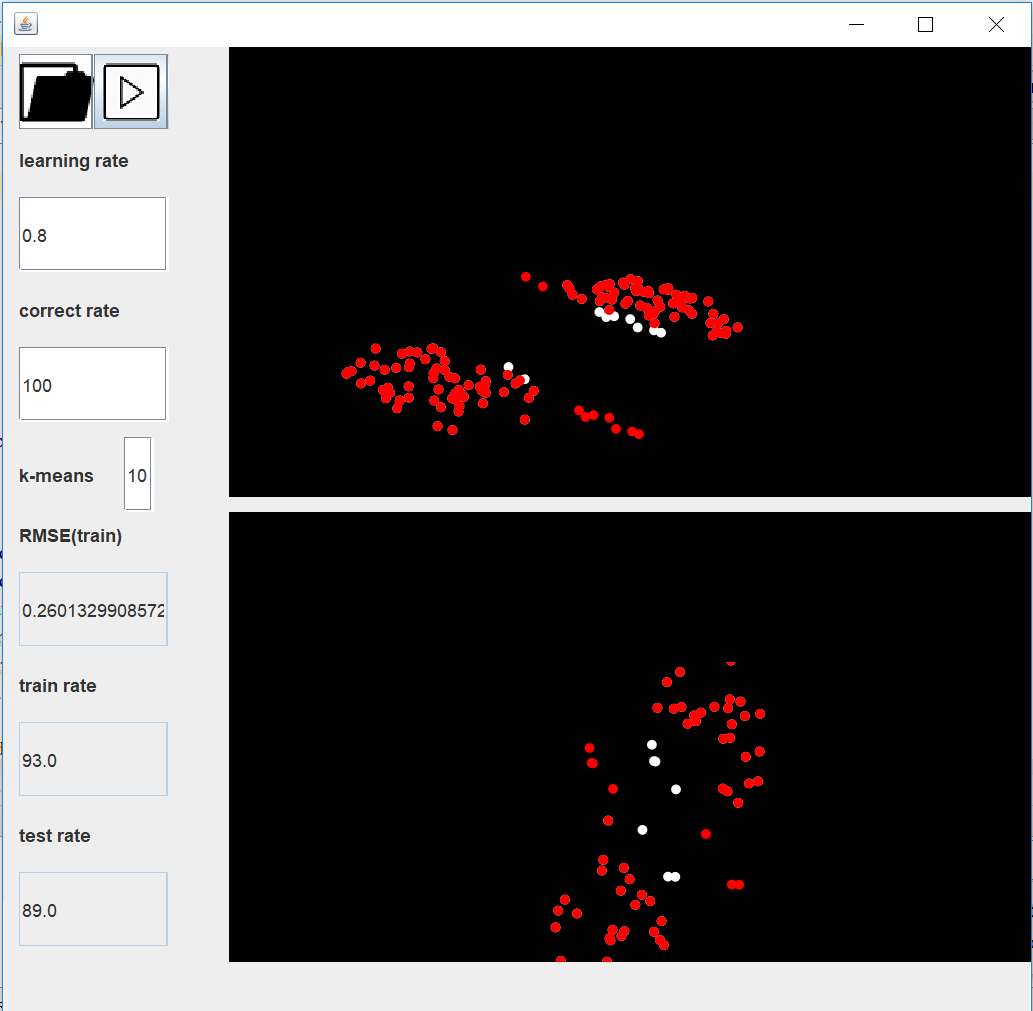


1. 2CloseS

Kmeans = 5

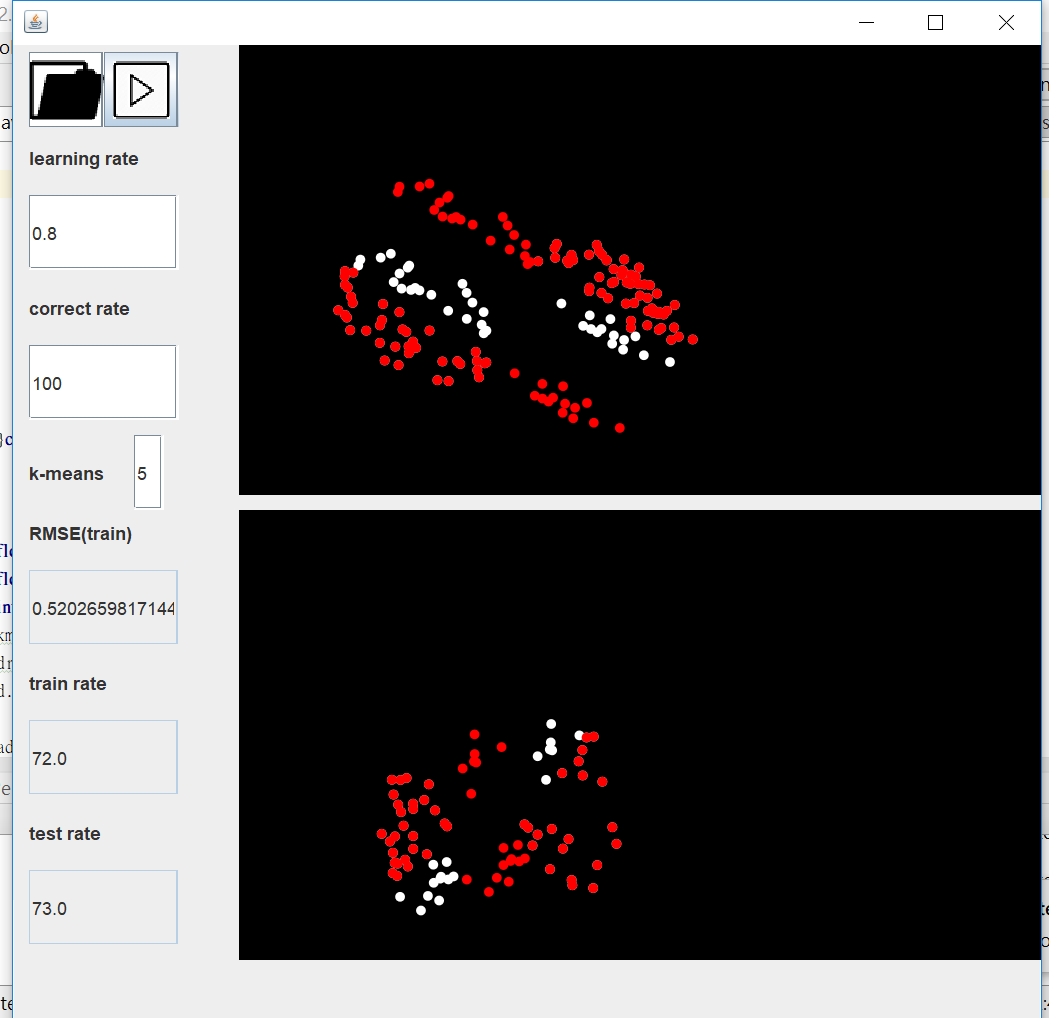


Kmeans = 10

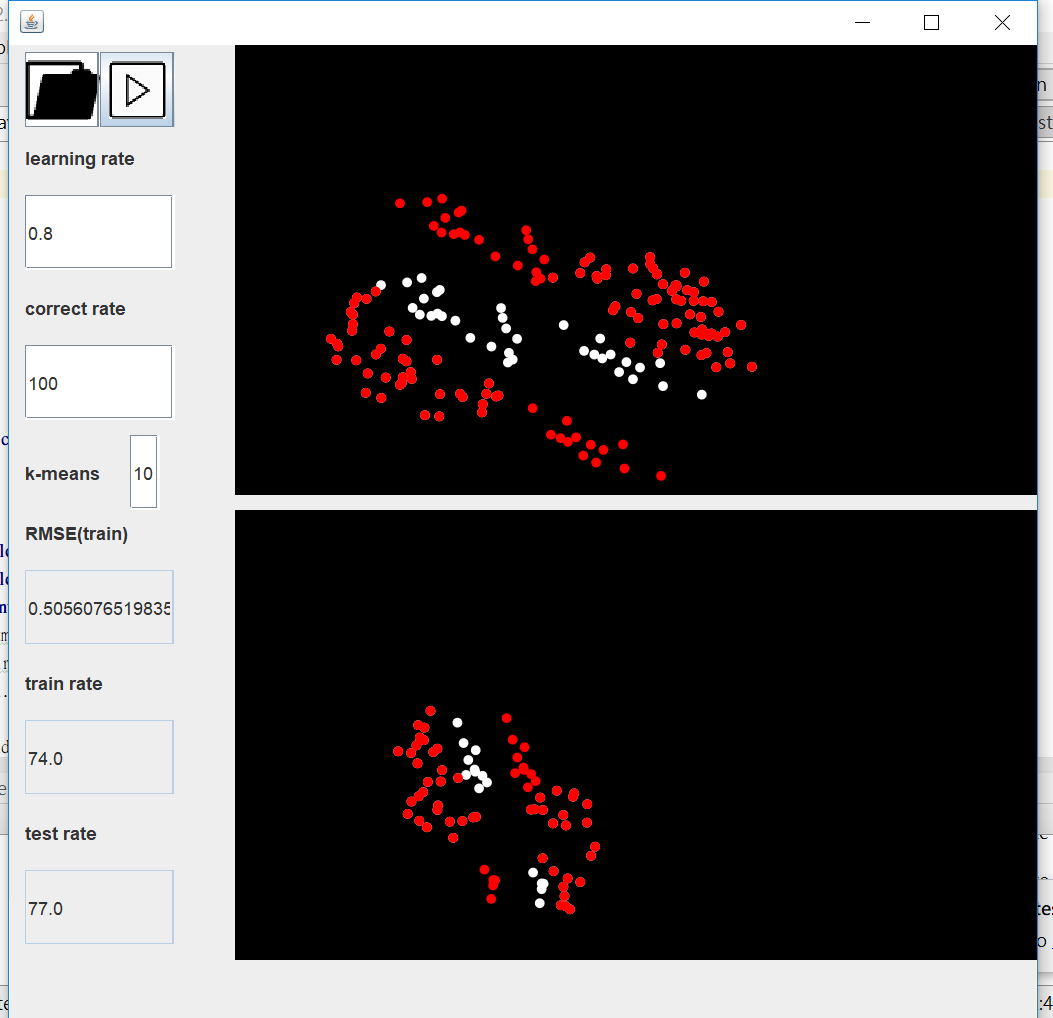


1. 2CloseS2

Kmeans = 5

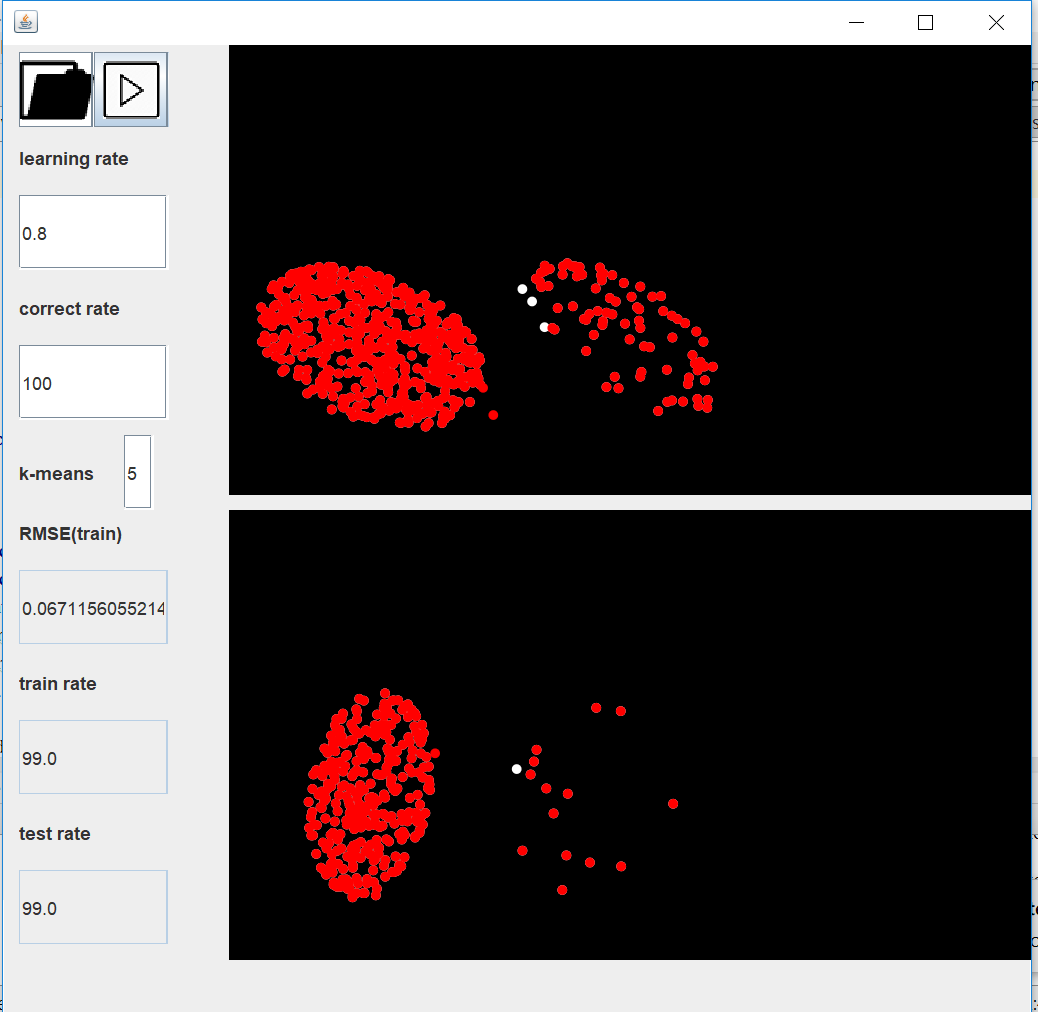


Kmeans = 10

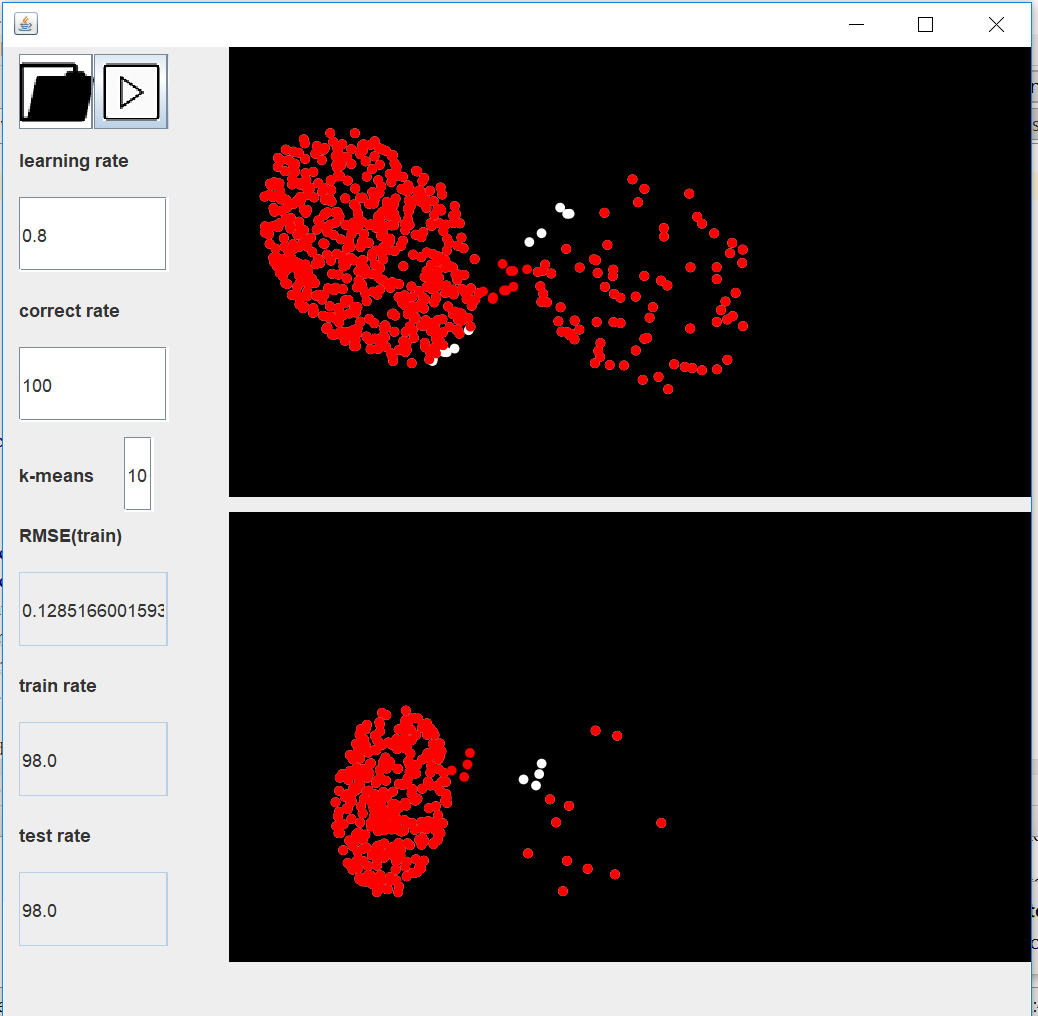


1. 2CloseS3

Kmeans = 5

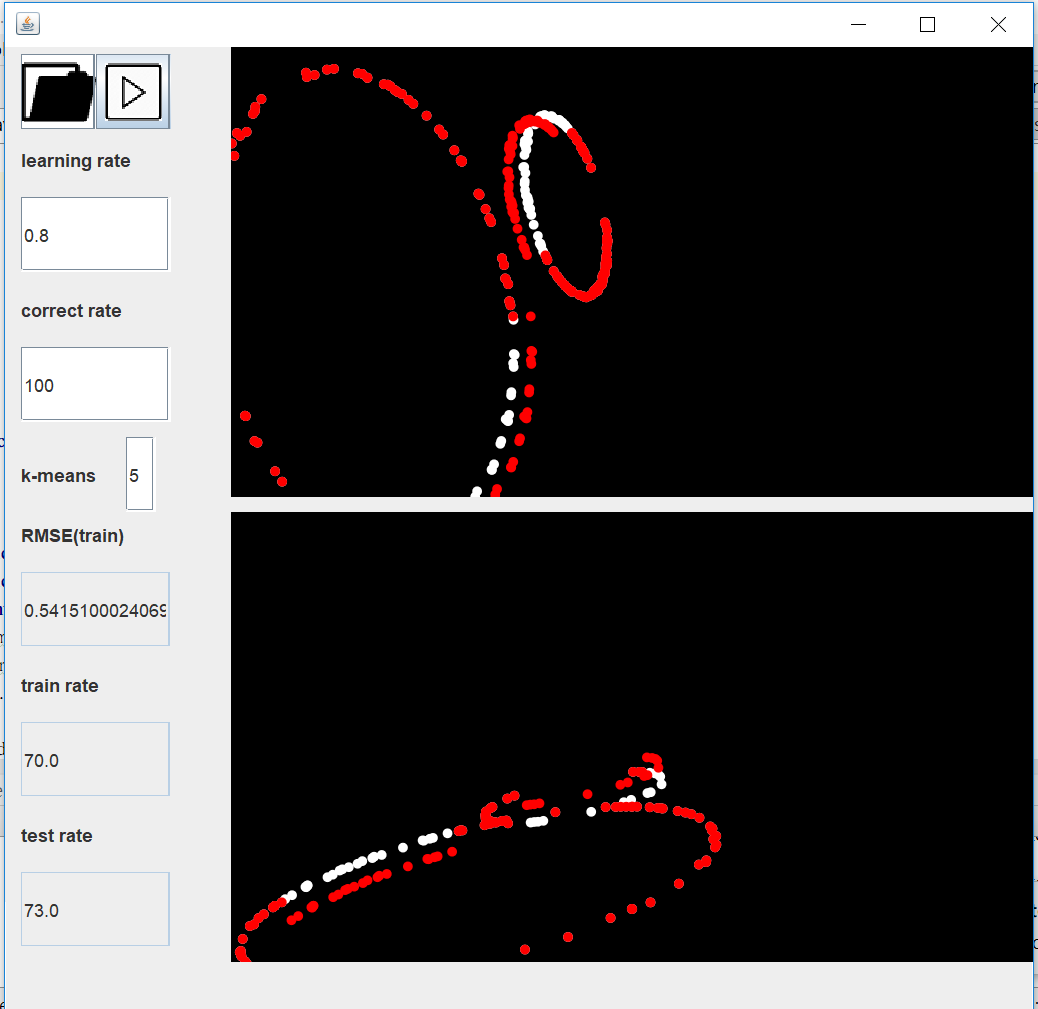


Kmeans = 10

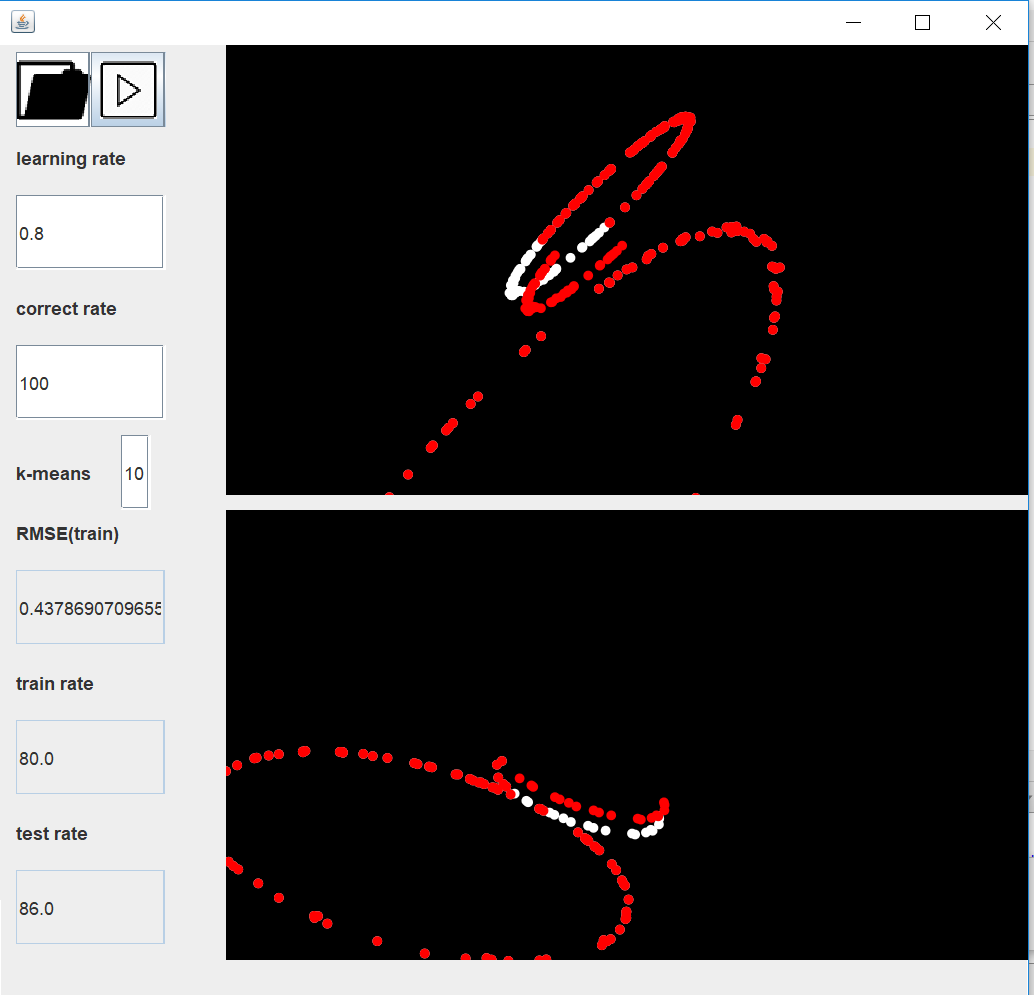


1. 2cring

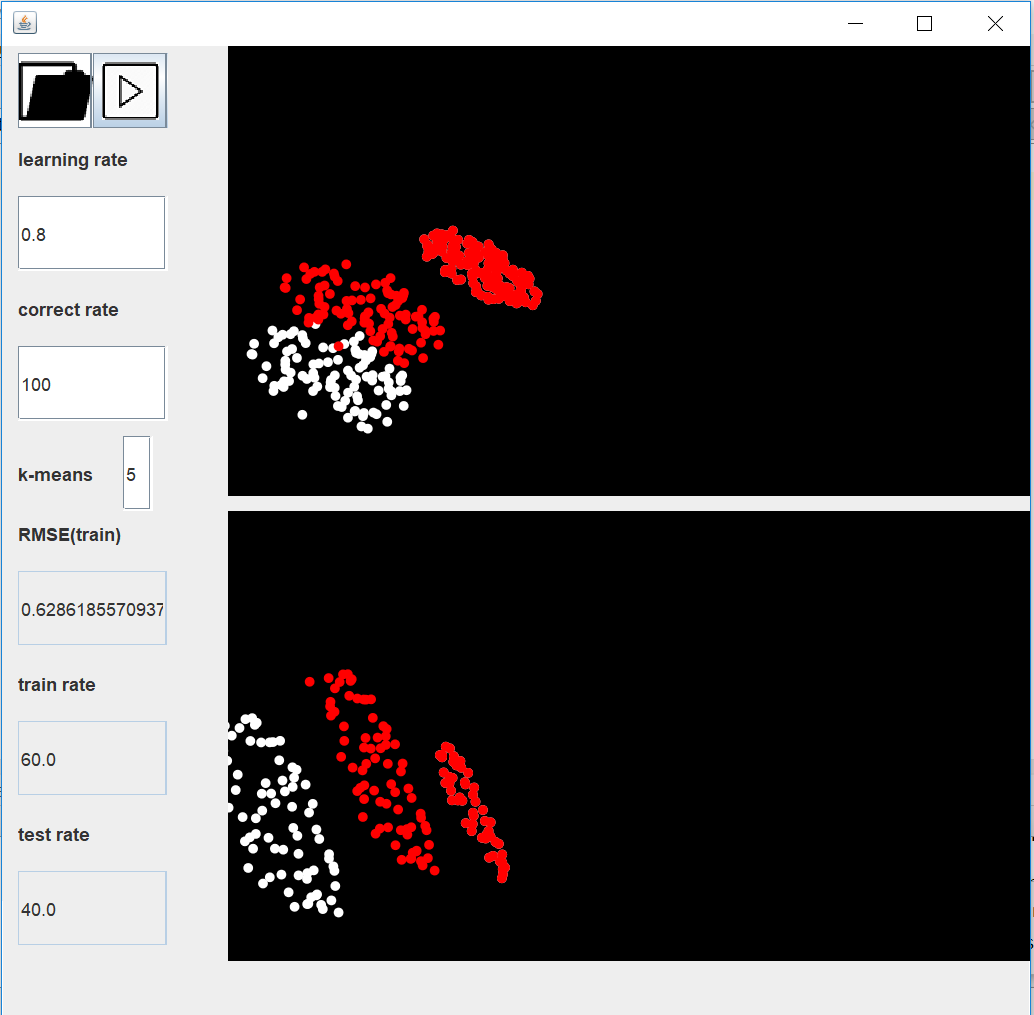
Kmeans = 5



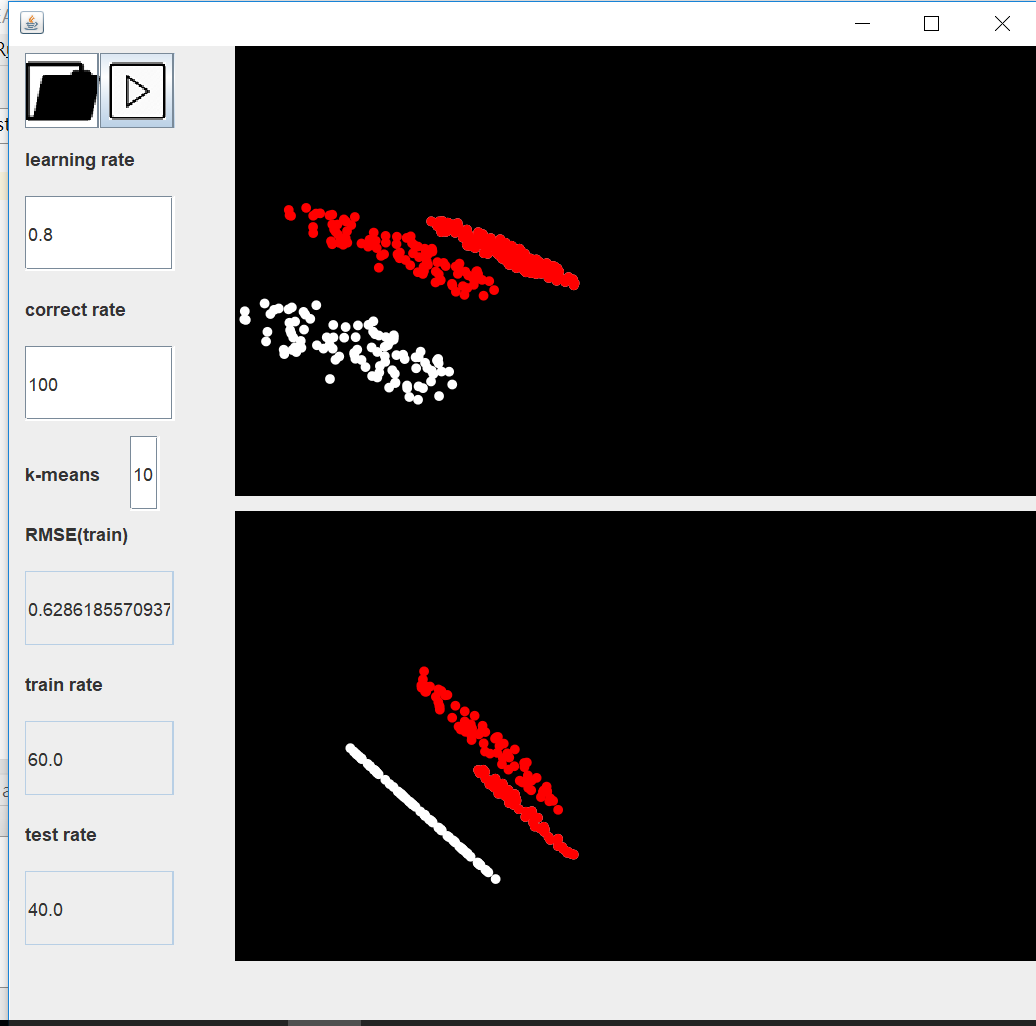
Kmeans = 10



(8)2CS

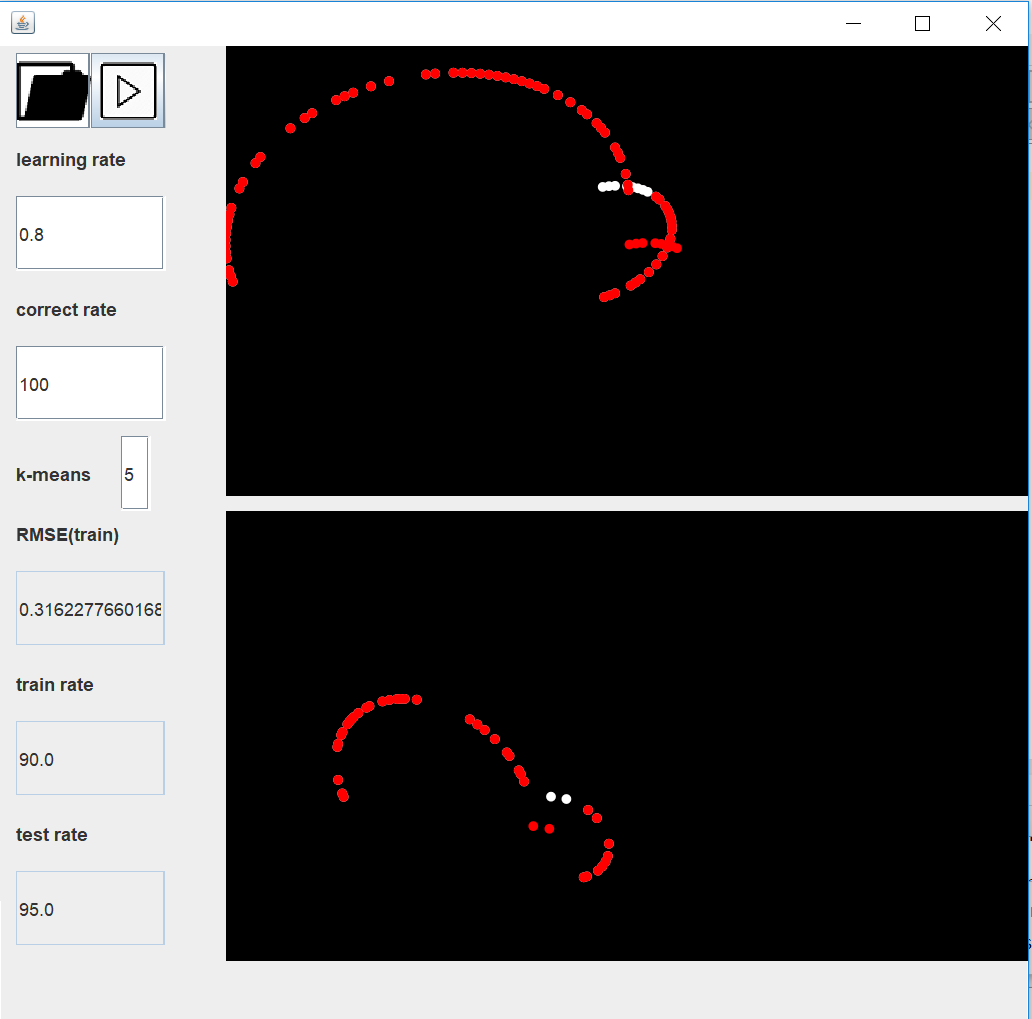
Kmeans=5

Kmeans = 10

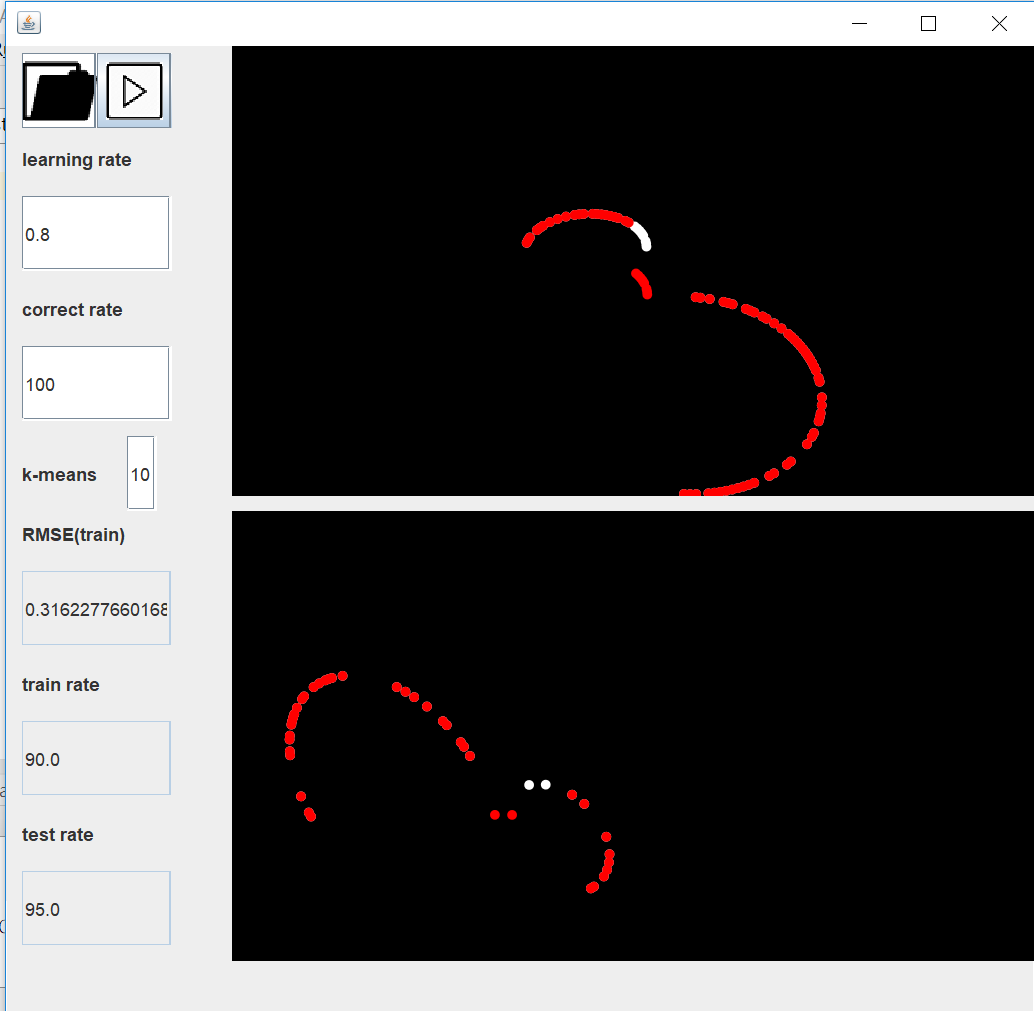


1. 2Hcircle1

Kmeans = 5

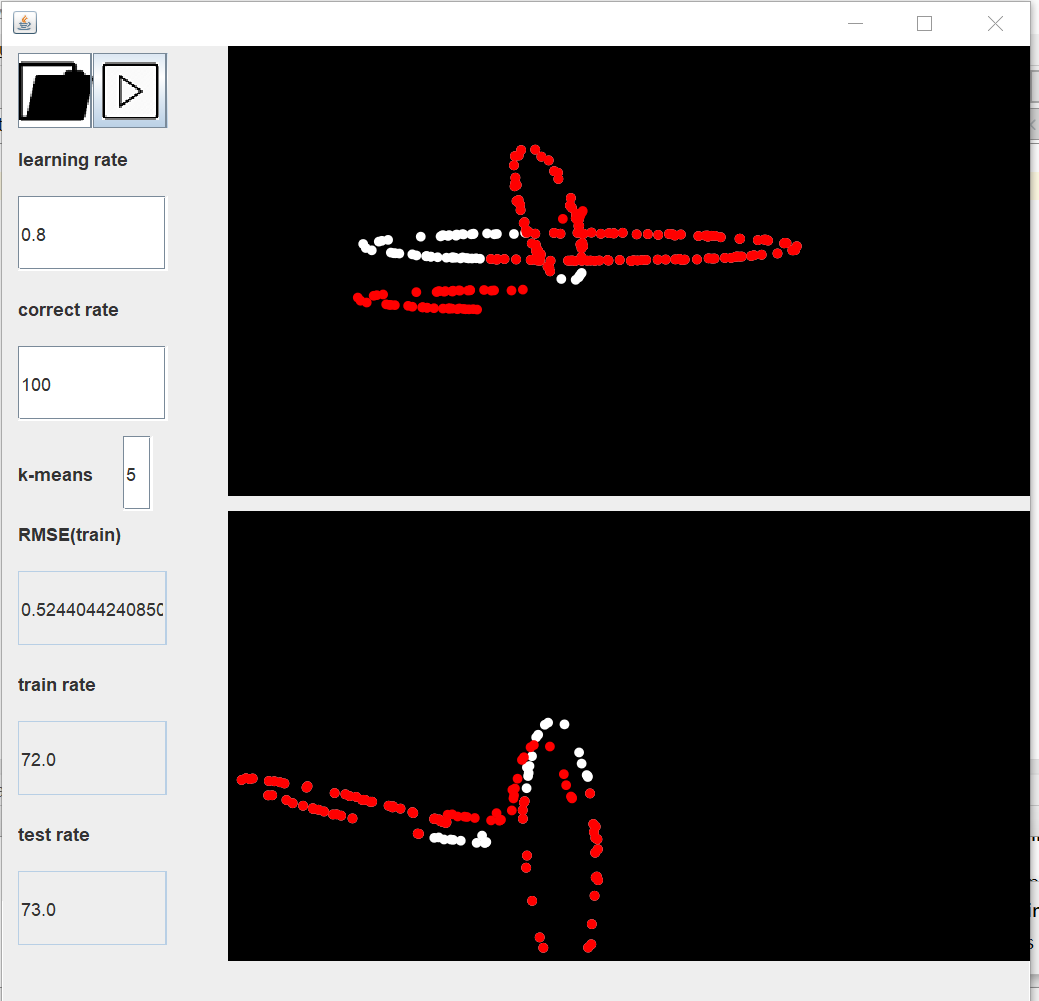


Kmeans = 10

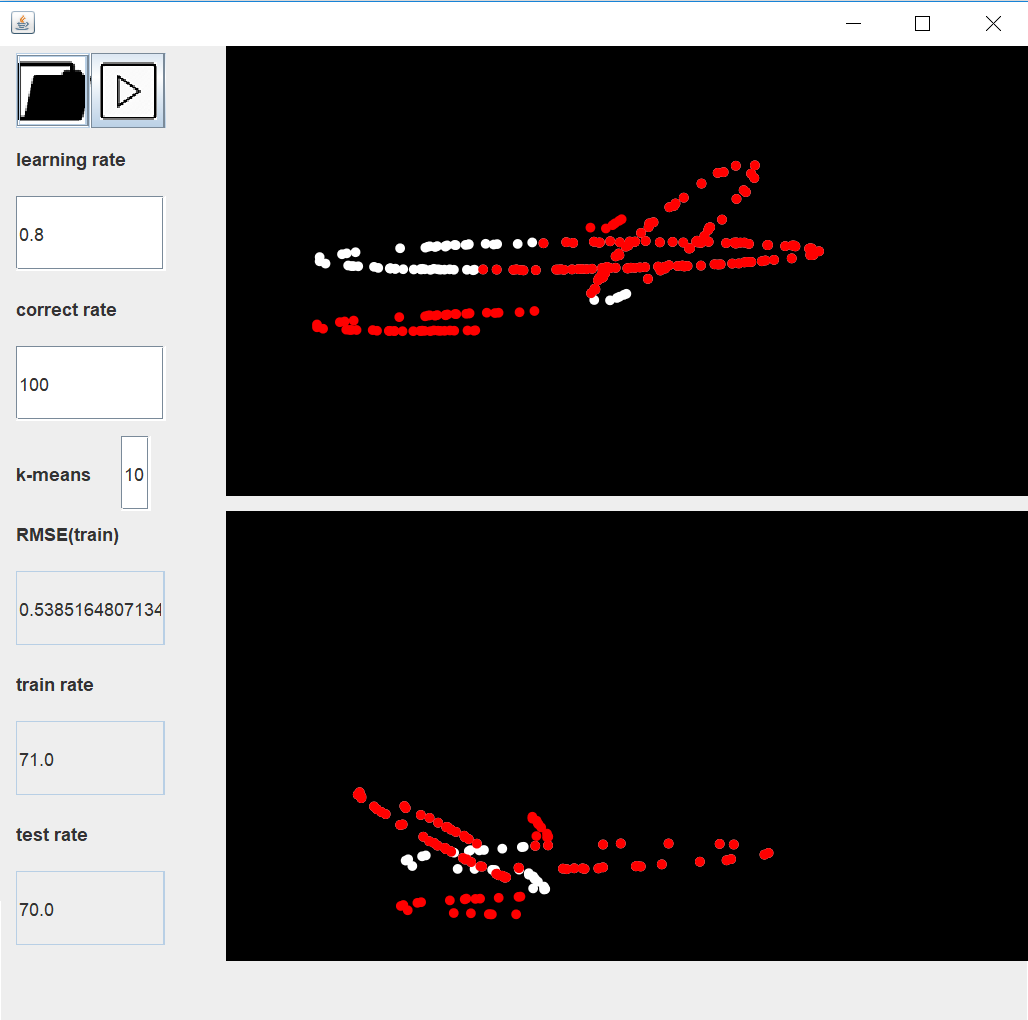


(9)2ring

Kmeans = 5

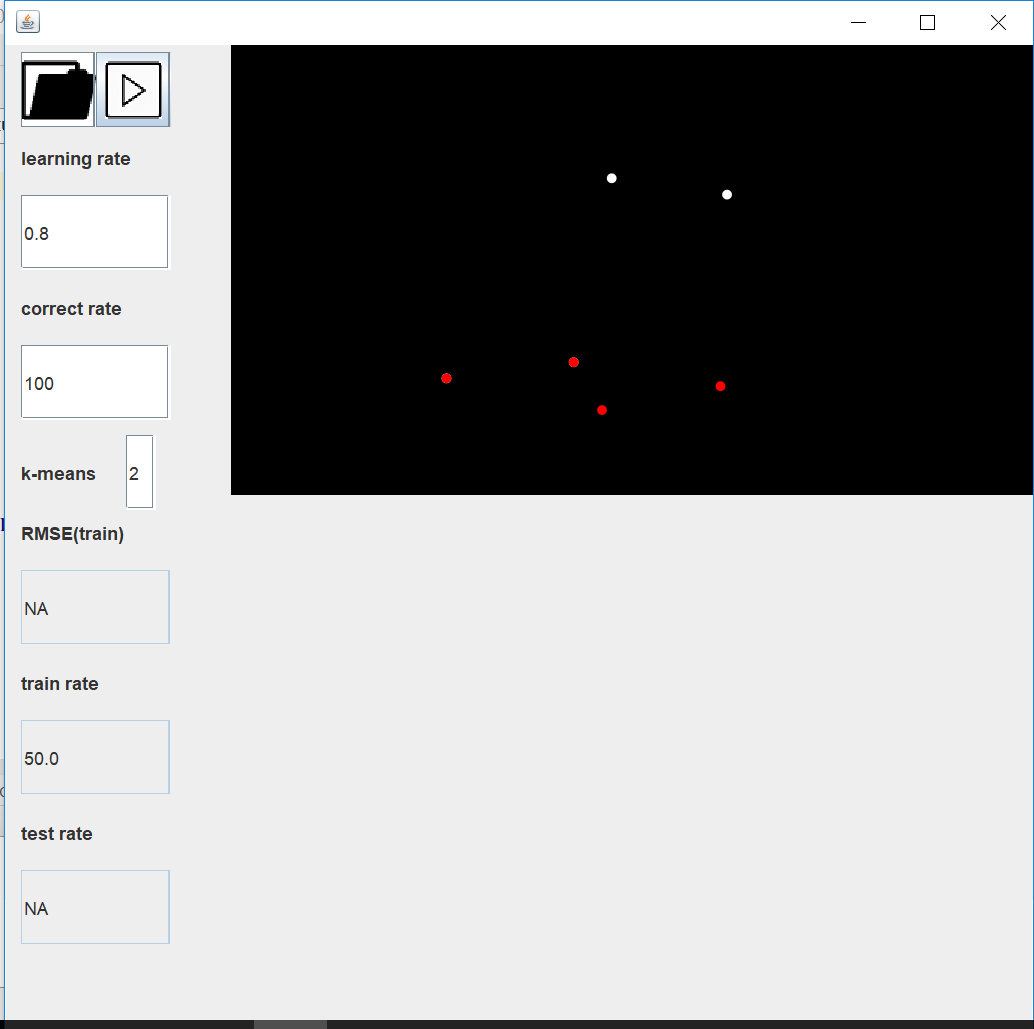


Kmeans = 10



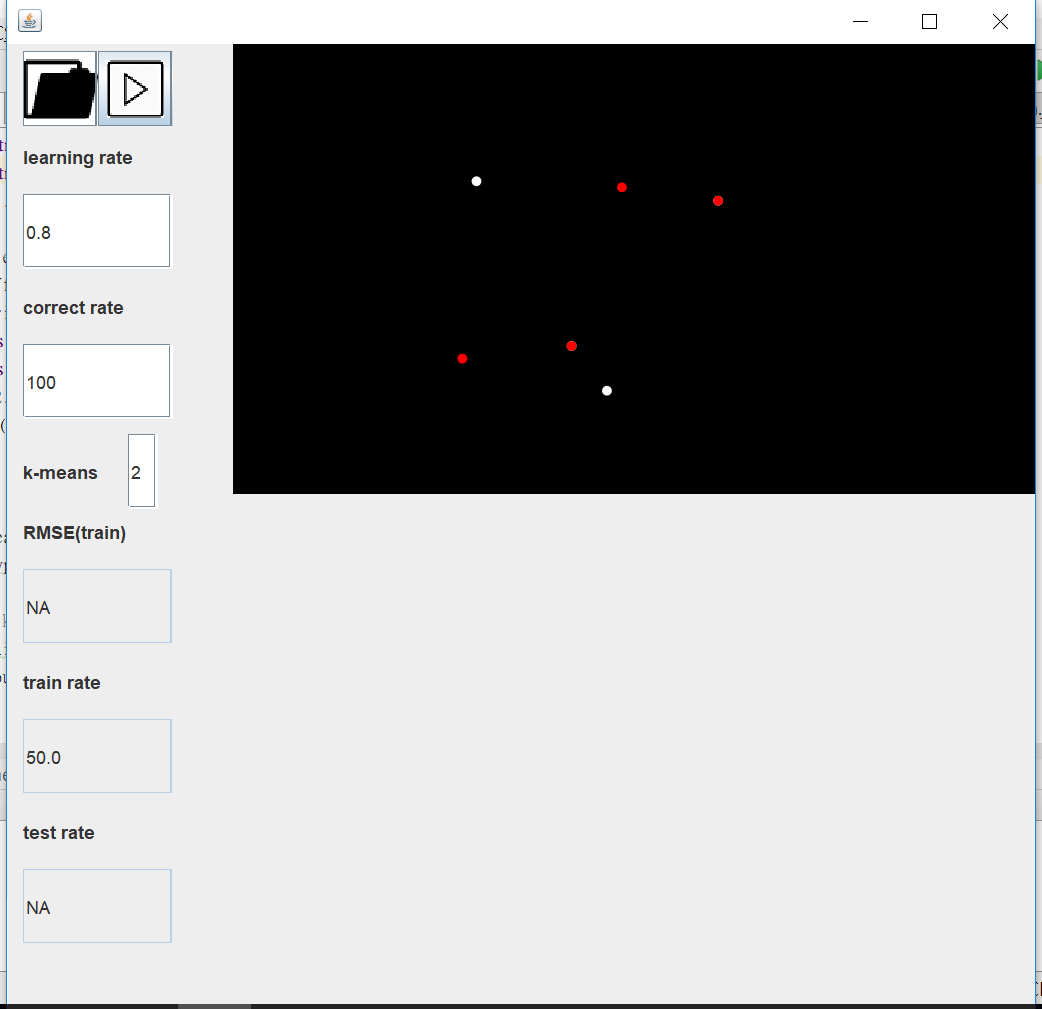
1. perceptron1

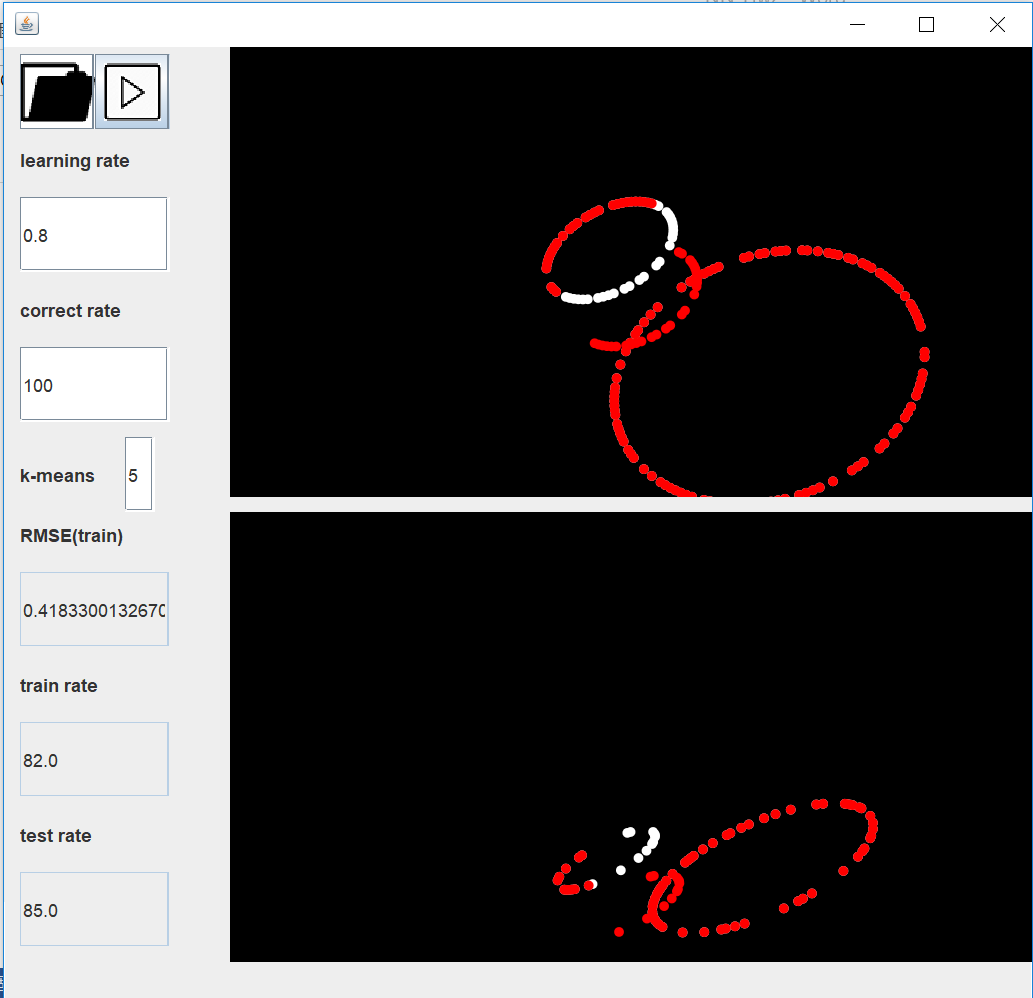
kmeans = 2



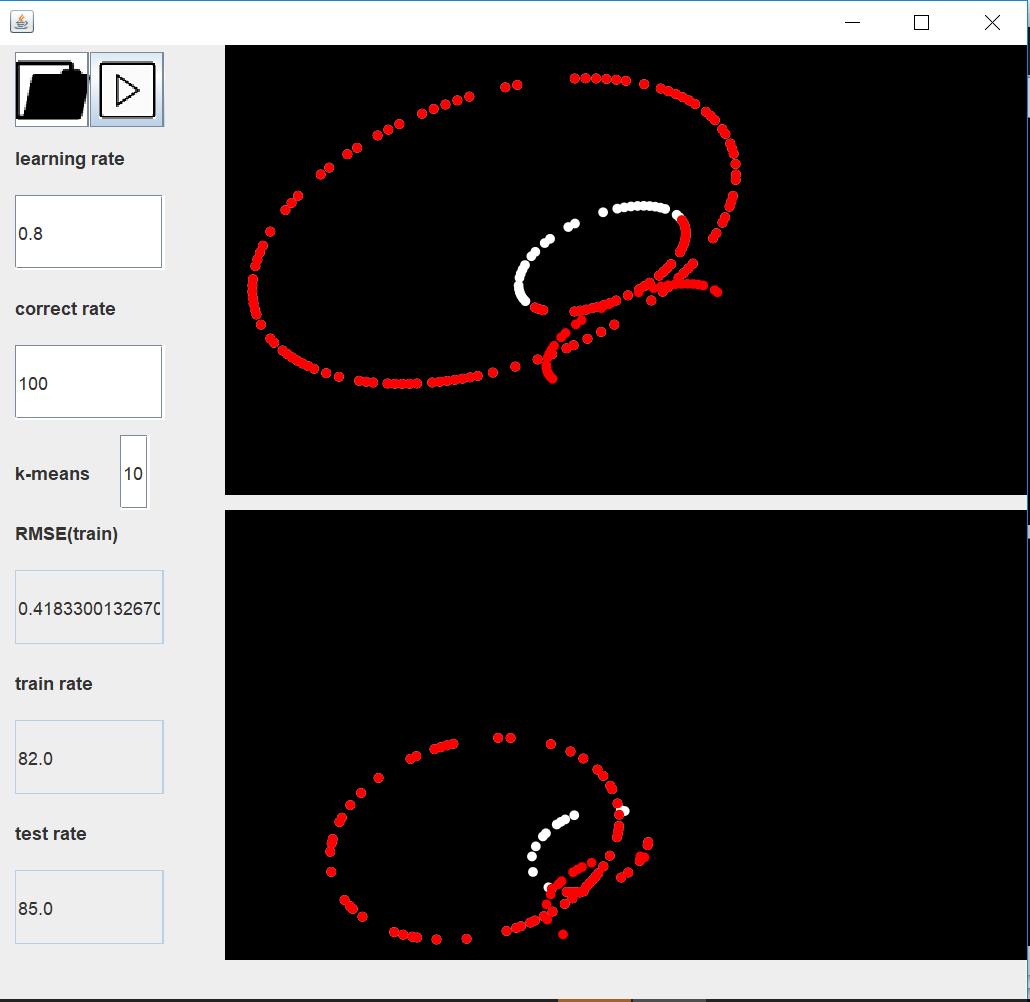
1. perceptron2

kmeans = 2





Kmeans = 10



1. 結果分析及討論

中心點、群數對於結果的影響:

1. 降低中心點的個數可以降低網路的複雜度，也可過濾雜訊
2. 中心點的位置對於網路訓練的收斂速度與穩定性有很大的影響，中心點的選取不足或選取不當可能導致實際值與預測值有明顯誤差
3. 需具備大量的訓練資料才能夠經由隨機選取獲得具有代表性的中心點

學習率對於結果的影響:

1. 若學習率低，結果會更加精確可靠，但可能會耗費較長的時間
2. 若學習率高，學習可能會較快，但也有可能訓練不會收斂，甚至會發散。

此程式使用LMS法(只修正輸出層的參數)，無修正隱藏層中心點位置、放射狀基底函數的形狀…等等，可能因選取的中心點不恰當…等原因而造成模擬結果較不精確。