CS 6200

Che Shian Hung

Programming Assignment 1

Report

The purpose of the program is using perceptron algorithm to train the classifiers for three classes. Each sample is represented as a three dimensional vector, and there are 30 sample data randomly generated for each class whenever the program runs. The first 15 samples for each class are used for training, and rest 15 samples for each class are used for testing the classifiers after training. There are three fixed sphere centers in the program that are used to generate the data, and the distances between the center points have to be greater than 15. In the program, the data will be randomly generated inside each sphere with radius unit 2, and the generated sample data will be different whenever the program runs, but the sphere centers are hard coded in the program. In the beginning of the program, the user can decide to train the classifiers with separable or non-separable data. If the user chooses to train with non-separable data, the last sample for both training and testing for each class will be swapped to the next class. For instance, the samples for class 1 will now have the samples for class 3.

At the beginning, I picked (10, 0, 0), (-5, 8, 0), and (-5, -8, 0) as sphere centers for easily visualization purpose. However, I found that with this set of data, the perceptron algorithm only requires no more than 3 iterations to train the classifier for each class since the bias terms do not need to change much. Therefore, I changed the centers to (0, 20, 0), (7.5, 5, 0), and (-7.5, 5, 0). Since now all three points are in the positive side of y axis, it requires more iterations to train the classifiers with this set of data.

Since the sample data is changed for each program runs, I just screen capture the output results, and analysis these specific cases.

For separable case:

Trained with non-separable data set? (y/n) n

display all samples:

display class 1 samples:

1.000000, -0.085199, 18.012153, -0.202991

1.000000, -1.330491, 21.492619, 0.043380

1.000000, 0.806885, 18.260693, -0.568989

1.000000, 0.867825, 18.214421, 0.242047

1.000000, 0.119164, 19.229461, -1.841757

1.000000, -1.341223, 19.235001, 1.271179

1.000000, -0.278322, 18.020785, 0.072430

1.000000, -1.318184, 21.037691, -1.088848

1.000000, -1.511701, 20.054993, -1.308333

1.000000, 0.483848, 18.060844, 0.074612

1.000000, 1.772445, 19.246102, 0.538586

1.000000, -1.439751, 21.375667, 0.186169

1.000000, 0.604805, 19.181741, 1.721819

1.000000, 0.619083, 21.654842, 0.937141

1.000000, -1.629875, 19.654738, -1.106482

1.000000, 1.136163, 21.644667, 0.064832

1.000000, 0.370226, 18.041999, -0.170774

1.000000, -0.094805, 21.231756, 1.572828

1.000000, -0.273804, 18.182316, -0.788070

1.000000, -1.493028, 18.700128, 0.284960

1.000000, 1.305781, 18.669991, 0.725266

1.000000, -1.578453, 19.712005, 1.193962

1.000000, -0.461382, 18.195034, 0.727477

1.000000, -0.023405, 21.999823, 0.012622

1.000000, 0.671261, 18.408191, -1.007747

1.000000, 0.622383, 21.577164, -1.060751

1.000000, 1.130836, 21.548463, 0.568747

1.000000, 1.721179, 19.733792, 0.983197

1.000000, -0.392597, 21.924145, 0.378859

1.000000, -0.988489, 21.516337, 0.850653

---------------------------------

display class 2 samples:

1.000000, 6.187042, 3.500744, 0.168439

1.000000, 6.879546, 3.247041, 0.736324

1.000000, 6.206884, 4.498413, -1.440923

1.000000, 7.837136, 3.365642, 1.102366

1.000000, 7.523783, 6.994028, -0.152597

1.000000, 6.672596, 3.187354, -0.172386

1.000000, 9.080290, 4.799338, -1.209305

1.000000, 7.063077, 3.182316, 0.710719

1.000000, 7.915184, 6.549728, -1.194138

1.000000, 7.617471, 3.081973, 0.554412

1.000000, 8.164619, 6.868749, 0.257017

1.000000, 8.568768, 5.488023, -1.618508

1.000000, 5.938584, 5.639565, -1.073749

1.000000, 8.682563, 6.516337, -0.549788

1.000000, 5.892056, 3.924949, -0.508706

1.000000, 7.363714, 6.994182, 0.068304

1.000000, 8.682093, 4.666564, 1.578441

1.000000, 8.458381, 3.976020, 1.425822

1.000000, 7.838340, 6.969000, 0.092547

1.000000, 8.660065, 3.482359, -0.592466

1.000000, 8.035431, 5.744626, -1.777314

1.000000, 7.489548, 6.999947, -0.010184

1.000000, 6.196781, 3.494144, 0.184440

1.000000, 6.048872, 5.985940, 0.960287

1.000000, 7.717027, 6.438191, 1.372773

1.000000, 6.401834, 6.439580, -0.849493

1.000000, 7.240258, 3.559032, -1.362404

1.000000, 7.298159, 5.989418, -1.726358

1.000000, 6.332805, 4.277702, 1.454627

1.000000, 7.130263, 6.952968, -0.221836

---------------------------------

display class 3 samples:

1.000000, -5.514825, 4.887060, 0.215236

1.000000, -7.585476, 6.675883, -1.088169

1.000000, -5.909805, 5.386552, 1.149721

1.000000, -7.317915, 6.857109, 0.719715

1.000000, -5.848315, 4.000171, 0.521805

1.000000, -5.538771, 5.206631, -0.332996

1.000000, -7.433038, 6.545928, 1.267131

1.000000, -8.172216, 4.070238, -1.638191

1.000000, -8.459795, 3.452804, 0.827633

1.000000, -7.434706, 6.325522, -1.496238

1.000000, -5.740023, 4.064930, 0.167703

1.000000, -7.618578, 3.005517, 0.089322

1.000000, -7.539696, 6.991115, -0.184082

1.000000, -8.587635, 5.540206, 1.589097

1.000000, -7.210927, 3.765095, 1.546430

1.000000, -7.153235, 3.118050, 0.581392

1.000000, -7.071097, 3.192458, 0.740832

1.000000, -7.555525, 6.998534, 0.052731

1.000000, -6.322314, 3.552115, 0.718809

1.000000, -6.645597, 3.306868, -0.635059

1.000000, -6.734523, 3.158109, -0.146570

1.000000, -8.497761, 3.267647, 0.058528

1.000000, -5.749128, 5.136893, -0.956927

1.000000, -9.382136, 4.901040, 0.669156

1.000000, -6.459054, 6.681319, 0.299328

1.000000, -6.289586, 6.442354, -0.674175

1.000000, -7.567518, 6.893830, 0.639413

1.000000, -7.691167, 3.009459, -0.034670

1.000000, -9.292281, 5.582436, 0.669698

1.000000, -7.008081, 6.912264, -0.318217

---------------------------------

17 iterations for class 1 trained data.

2 iterations for class 2 trained data.

2 iterations for class 3 trained data.

display all weights:

display class 1 weights:

-35.200000, -0.667446, 2.994261, 2.630477

display class 2 weights:

0.800000, 10.521430, -9.011494, 0.886203

display class 3 weights:

0.800000, -10.412082, -5.159368, -0.535953

display test result:

result for class 1:

1

1

1

1

1

1

1

1

1

1

1

1

1

1

1

result for class 2:

2

2

2

2

2

2

2

2

2

2

2

2

2

2

2

result for class 3:

3

3

3

3

3

3

3

3

3

3

3

3

3

3

3

--------------------------------

Press any key to continue . . .

In the output, the sample data is listed first. There are four items stored in each data set since this is easier for me to program the algorithm while performing perceptron and testing with data. The second, third, and fourth row each represents x, y, z values in the coordinate system. Therefore, we can see the values are close to the corresponding center points due to radius restriction.

For the next output, I list the number of iteration took for each classifier and the result weights. The number of iteration for the first classifier is more than the other two, and it makes sense because the first class sits on the top of other two classes. With fixed zeta value, the first classifier requires more moves to shift all zero weight vector to the top where separates class 1 from other classes. For the weight vector, the columns from left to right each represent: w0 (bias), w1(x), w2(y), w3(z).

Next, the rest of the output shows the testing result of the trained classifiers. I use the last 15 sample data for each class to test. During testing, each sample vector is multiplied with all three classifiers, and the one computes the greatest value means that the sample is been classified to the class. Since the clusters are very far away from each other, the compute result usually leads out one positive value and two negative values. However, depends on the cluster distance and the radius restriction, the sign of compute values can differ.

Now I am going to analyze the result from non-linearly separable case.

Trained with non-separable data set? (y/n) y

display all samples:

display class 1 samples:

1.000000, -1.269980, 18.619984, 0.694771

1.000000, -0.887270, 18.209048, -0.072390

1.000000, -0.172223, 18.039193, -0.354365

1.000000, 0.941506, 20.816434, 1.564290

1.000000, -0.088074, 18.120089, -0.676888

1.000000, -1.443757, 21.319521, 0.417649

1.000000, 1.422699, 21.290728, 0.556729

1.000000, -0.043535, 18.970870, -1.714350

1.000000, -1.799682, 20.832833, -0.259872

1.000000, 0.511114, 21.819349, 0.654775

1.000000, -0.264646, 19.084434, -1.758323

1.000000, -0.569295, 18.677476, 1.388104

1.000000, 0.614226, 19.349077, 1.788582

1.000000, 0.311447, 18.025960, 0.078531

\*\*\*\*1.000000, 0.693414, 18.474571, 1.091899

1.000000, 0.987517, 19.504223, 1.667038

1.000000, -0.326703, 20.382627, 1.935681

1.000000, -0.893553, 19.400368, -1.685825

1.000000, 0.566562, 19.132771, 1.710825

1.000000, -0.893806, 18.440208, 0.876447

1.000000, -1.637660, 20.567111, -0.998225

1.000000, 0.515740, 18.109409, -0.399595

1.000000, -0.513691, 18.078053, 0.205528

1.000000, 0.085232, 18.012374, 0.205131

1.000000, 0.518418, 21.863716, 0.507747

1.000000, -0.298230, 18.026933, -0.134402

1.000000, 0.419920, 20.276115, -1.935827

1.000000, -0.656290, 18.123525, 0.219378

1.000000, -0.729654, 19.811281, 1.852563

\*\*\*\*1.000000, -1.941016, 19.891054, -0.469669

---------------------------------

display class 2 samples:

1.000000, 7.514342, 6.995995, -0.125693

1.000000, 7.538184, 5.792631, 1.835832

1.000000, 8.483739, 3.280786, -0.276696

1.000000, 7.738147, 3.038020, -0.306462

1.000000, 7.817862, 5.495777, 1.911326

1.000000, 7.140736, 6.967222, -0.031121

1.000000, 6.906370, 3.777720, -1.467527

1.000000, 9.416127, 4.546966, -0.351024

1.000000, 8.095996, 3.571573, 1.266643

1.000000, 7.852747, 3.145875, 0.661657

1.000000, 6.908333, 3.102979, -0.226365

1.000000, 7.090373, 6.837971, 0.673846

1.000000, 9.373355, 4.603641, 0.577445

1.000000, 6.836506, 3.119407, -0.152142

\*\*\*\*1.000000, 8.195134, 3.266648, 0.715737

1.000000, 6.858442, 5.809124, -1.712811

1.000000, 7.450726, 3.772976, 1.578602

1.000000, 7.396623, 3.131964, 0.706934

1.000000, 8.187487, 3.127711, 0.147971

1.000000, 7.771331, 3.031352, -0.225396

1.000000, 8.058259, 6.920279, 0.029615

1.000000, 5.652278, 4.383220, 0.453329

1.000000, 7.784063, 6.856366, -0.687905

1.000000, 6.860960, 3.349663, 0.931674

1.000000, 8.279237, 6.034270, -1.524164

1.000000, 7.519445, 6.998609, 0.071996

1.000000, 7.884087, 6.941098, -0.290886

1.000000, 8.353537, 6.797140, 0.204357

1.000000, 8.152475, 6.755839, -0.700931

\*\*\*\*1.000000, 7.822114, 3.036104, 0.198377

---------------------------------

display class 3 samples:

1.000000, -8.183217, 6.011929, 1.584050

1.000000, -8.493770, 5.295910, -1.710222

1.000000, -8.655968, 5.720433, 1.464484

1.000000, -8.822963, 3.691032, 0.732374

1.000000, -5.817469, 5.688613, 0.833608

1.000000, -8.079669, 3.598426, -1.303677

1.000000, -8.057293, 5.011000, -1.920756

1.000000, -9.003633, 3.953773, -0.802807

1.000000, -7.799435, 4.925018, -1.976036

1.000000, -7.444052, 6.999199, -0.008466

1.000000, -8.569109, 6.685638, -0.125017

1.000000, -9.019759, 3.919894, -0.723673

1.000000, -7.029224, 6.293781, 1.450690

1.000000, -6.603650, 3.397358, 0.792525

\*\*\*\*1.000000, -8.232848, 3.587057, 1.211002

1.000000, -6.919803, 3.399754, 1.050041

1.000000, -7.027314, 6.939644, -0.119783

1.000000, -7.266482, 6.513726, -1.286119

1.000000, -5.631521, 5.228500, 0.675703

1.000000, -7.908290, 3.057942, 0.248415

1.000000, -6.493720, 4.059631, -1.450209

1.000000, -6.139518, 4.702113, -1.435393

1.000000, -7.401162, 3.025960, 0.305610

1.000000, -9.287871, 5.626285, -0.641315

1.000000, -7.479075, 3.000801, 0.052574

1.000000, -6.209953, 3.972586, 1.131459

1.000000, -8.318657, 3.570174, -1.133754

1.000000, -9.147076, 5.913787, 0.672408

1.000000, -7.247381, 6.935177, 0.437349

\*\*\*\*1.000000, -8.820655, 3.930013, 1.054039

---------------------------------

display all samples:

display class 1 samples:

1.000000, -1.269980, 18.619984, 0.694771

1.000000, -0.887270, 18.209048, -0.072390

1.000000, -0.172223, 18.039193, -0.354365

1.000000, 0.941506, 20.816434, 1.564290

1.000000, -0.088074, 18.120089, -0.676888

1.000000, -1.443757, 21.319521, 0.417649

1.000000, 1.422699, 21.290728, 0.556729

1.000000, -0.043535, 18.970870, -1.714350

1.000000, -1.799682, 20.832833, -0.259872

1.000000, 0.511114, 21.819349, 0.654775

1.000000, -0.264646, 19.084434, -1.758323

1.000000, -0.569295, 18.677476, 1.388104

1.000000, 0.614226, 19.349077, 1.788582

1.000000, 0.311447, 18.025960, 0.078531

\*\*\*\*1.000000, -8.232848, 3.587057, 1.211002

1.000000, 0.987517, 19.504223, 1.667038

1.000000, -0.326703, 20.382627, 1.935681

1.000000, -0.893553, 19.400368, -1.685825

1.000000, 0.566562, 19.132771, 1.710825

1.000000, -0.893806, 18.440208, 0.876447

1.000000, -1.637660, 20.567111, -0.998225

1.000000, 0.515740, 18.109409, -0.399595

1.000000, -0.513691, 18.078053, 0.205528

1.000000, 0.085232, 18.012374, 0.205131

1.000000, 0.518418, 21.863716, 0.507747

1.000000, -0.298230, 18.026933, -0.134402

1.000000, 0.419920, 20.276115, -1.935827

1.000000, -0.656290, 18.123525, 0.219378

1.000000, -0.729654, 19.811281, 1.852563

\*\*\*\*1.000000, -8.820655, 3.930013, 1.054039

---------------------------------

display class 2 samples:

1.000000, 7.514342, 6.995995, -0.125693

1.000000, 7.538184, 5.792631, 1.835832

1.000000, 8.483739, 3.280786, -0.276696

1.000000, 7.738147, 3.038020, -0.306462

1.000000, 7.817862, 5.495777, 1.911326

1.000000, 7.140736, 6.967222, -0.031121

1.000000, 6.906370, 3.777720, -1.467527

1.000000, 9.416127, 4.546966, -0.351024

1.000000, 8.095996, 3.571573, 1.266643

1.000000, 7.852747, 3.145875, 0.661657

1.000000, 6.908333, 3.102979, -0.226365

1.000000, 7.090373, 6.837971, 0.673846

1.000000, 9.373355, 4.603641, 0.577445

1.000000, 6.836506, 3.119407, -0.152142

\*\*\*\*1.000000, 0.693414, 18.474571, 1.091899

1.000000, 6.858442, 5.809124, -1.712811

1.000000, 7.450726, 3.772976, 1.578602

1.000000, 7.396623, 3.131964, 0.706934

1.000000, 8.187487, 3.127711, 0.147971

1.000000, 7.771331, 3.031352, -0.225396

1.000000, 8.058259, 6.920279, 0.029615

1.000000, 5.652278, 4.383220, 0.453329

1.000000, 7.784063, 6.856366, -0.687905

1.000000, 6.860960, 3.349663, 0.931674

1.000000, 8.279237, 6.034270, -1.524164

1.000000, 7.519445, 6.998609, 0.071996

1.000000, 7.884087, 6.941098, -0.290886

1.000000, 8.353537, 6.797140, 0.204357

1.000000, 8.152475, 6.755839, -0.700931

\*\*\*\*1.000000, -1.941016, 19.891054, -0.469669

---------------------------------

display class 3 samples:

1.000000, -8.183217, 6.011929, 1.584050

1.000000, -8.493770, 5.295910, -1.710222

1.000000, -8.655968, 5.720433, 1.464484

1.000000, -8.822963, 3.691032, 0.732374

1.000000, -5.817469, 5.688613, 0.833608

1.000000, -8.079669, 3.598426, -1.303677

1.000000, -8.057293, 5.011000, -1.920756

1.000000, -9.003633, 3.953773, -0.802807

1.000000, -7.799435, 4.925018, -1.976036

1.000000, -7.444052, 6.999199, -0.008466

1.000000, -8.569109, 6.685638, -0.125017

1.000000, -9.019759, 3.919894, -0.723673

1.000000, -7.029224, 6.293781, 1.450690

1.000000, -6.603650, 3.397358, 0.792525

\*\*\*\*1.000000, 8.195134, 3.266648, 0.715737

1.000000, -6.919803, 3.399754, 1.050041

1.000000, -7.027314, 6.939644, -0.119783

1.000000, -7.266482, 6.513726, -1.286119

1.000000, -5.631521, 5.228500, 0.675703

1.000000, -7.908290, 3.057942, 0.248415

1.000000, -6.493720, 4.059631, -1.450209

1.000000, -6.139518, 4.702113, -1.435393

1.000000, -7.401162, 3.025960, 0.305610

1.000000, -9.287871, 5.626285, -0.641315

1.000000, -7.479075, 3.000801, 0.052574

1.000000, -6.209953, 3.972586, 1.131459

1.000000, -8.318657, 3.570174, -1.133754

1.000000, -9.147076, 5.913787, 0.672408

1.000000, -7.247381, 6.935177, 0.437349

\*\*\*\*1.000000, 7.822114, 3.036104, 0.198377

---------------------------------

3000 iterations for class 1 trained data.

3000 iterations for class 2 trained data.

3000 iterations for class 3 trained data.

display all weights:

display class 1 weights:

-291.200000, -24.805921, 8.267281, -12.663479

display class 2 weights:

-379.200000, 48.353123, 24.141840, -14.099993

display class 3 weights:

57.600000, -0.840676, -9.814149, 8.517729

display test result:

result for class 1:

2

2

2

2

2

2

2

2

2

2

2

2

2

2

3

result for class 2:

2

2

2

2

2

2

3

2

3

2

2

2

2

2

2

result for class 3:

3

3

3

3

3

3

3

3

3

3

3

3

3

3

2

--------------------------------

Press any key to continue . . .

At the beginning of the output, the sample data is printed twice. One before the sample swaps between classes, and one after swapping. I use ‘\*\*\*\*’ to mark the sample data that is swapped. So the 15th and 30th data from class1 will be shift to class 2, and the data from class 2 will be shift to class 3, and the data in class 3 will be shift to class 1. The difference can be observed between two versions.

Since perceptron algorithm cannot solve non-linearly separable problems, I set the iteration limit to 3000, so that I assume perceptron algorithm will give me ‘good enough’ classifier when the number of iteration reaches 3000. Based on the output, the algorithm still cannot figure out the classifier when the iteration limit had been met, which makes sense since the problem is non-linearly separable. The weight vectors seemed to be modified multiple times based on the bias term.

In the testing result, we can see errors in the output. Remember, the 30th sample data for each class is also swapped with others. But still, all of the result for data from class 1 is wrong. This might be caused of the swapped sample. The 30th sample data is actually from class 3, so in this case, it makes sense that the classifiers think the swapped sample comes from class 3. For class 2 and class 3, the error seems to be less than class 1. It might be caused by the starting weight, which is set to all zeros at the beginning. If we change the initial weight to other vectors, the errors for each class can be different.

Besides these two cases, I also played around the parameter and observed the behavior. For instance, if I change the radius from 2 to 20. The result then becomes:  
  
display test result:

result for class 1:

2

3

3

2

2

3

3

3

3

3

3

3

2

3

3

result for class 2:

3

2

2

2

3

1

3

2

2

3

2

2

2

2

2

result for class 3:

3

3

3

2

2

3

3

2

3

2

3

3

2

3

3

In this case, the perceptron also passes the iteration limit, and it make sense because if the radius is greater than the center distance, it is highly likely that the problem can become non-linearly separable, which causing errors in the testing result without even swapping the samples.

Source codes:

/\* CS6200 Che Shian Hung 2/18/2018

Programming Assignment 1

Purpose: This program uses perceptron algorithm to train the classifiers for three classes.

Each sample is represented as a three dimensional vector, and there are 30 sample data

randomly generated for each class everytime the program runs. The first half of data

is used for training, and the second half of data is used for testing the classifiers

after training. There are three fixed sphere centers that are used to generate the data, and

the distances between them have to be greater than 15. The data will then be randomly

generated inside each sphere with radius 2. In the beginning, the user can decide to

train the classifiers with separable or non-separable data. If training with

non-separable data, the last sample for both training and testing for each class will

be swap to the next class. For instance, the samples for class 1 will now have the samples

for class 3.

Architecture: There are mainly three steps in the program: data generation, perceptron algorithm,

and testing. Each step has been encapsulated in few functions. Also, for testing purpose,

each function has been designed for reusability. For instance, we can output any information

for a specific class.

Data Structure: The data information like sample vectors, weight vectors, and result are all stored

in statically allocated arrays. The arrays are all multidimensional for easy programming

purpose. All data information are set as global variables so that all the functions are

able to modify the information without passing any arrays into the function. In the program,

I defined as many as constant as opssible at the top of the program, so that we can easily

modify the constant for different testing purposes such as adding more sample data. However,

CLASSNUM and DIMENSION can only be changed after modifying the program

\*/

#define \_USE\_MATH\_DEFINES

// Import libraries and constants

#include<iostream>

#include<stdlib.h>

#include<time.h>

#include<cmath>

#include<string>

#define RADIUS 2

#define TESTSIZE 15

#define CLASSNUM 3

#define DIMENSION 3

#define SAMPLESIZE 30

#define WEIRDSYMBOL 0.8 // The greek symbol in gradient decent algorithm (zeta...???)

#define MODIFYINDEX 15 // The index of sample data that is chose to be non-separable while training

#define MAXITERATION 3000

using namespace std;

// Declare global constant variable

const double sphereCenter[CLASSNUM][DIMENSION] = { {0, 20, 0}, {7.5, 5, 0}, {-7.5, 5, 0} }; // Hard coded sphere centers

// Declare global variables

double classSamples[CLASSNUM][SAMPLESIZE][DIMENSION + 1]; // Sample for all three classes, including training data and testing data

double classWeights[CLASSNUM][DIMENSION + 1]; // Weights for all three classes, can be trained and untrained

int resultClass[CLASSNUM][SAMPLESIZE - TESTSIZE]; // Captures the testing result after testing with trained classifiers

bool modifyData = false; // A switch capture by the user to modify few data

void generateAllSamples(); // Generate samples randomly for all classes

void generateRandomSamples(double samples[SAMPLESIZE][DIMENSION + 1], int classNum); // Generate samples randomly for a specific class

void displayAllSamples(); // Display samples for all classes

void displaySamples(int classNum); // Display samples for a specific class

void displayAllWeights(); // Display weights for all classes

void displayWeights(int classNum); // Display weights for a specific class

void perceptron(); // Apply perceptron algorithm and update weights

void testWeights(); // Test the classifiers with data and update test result

void initializeWeights(); // Initialize weights' values for all classifiers

void displayTestResult(); // Display test result for each class

void modifyTrainedData(int modifyIndex); // Swap a specific sample between classes

double vectorMultiplication(double a[DIMENSION + 1], double b[DIMENSION + 1]); // Perform vector multiplication between two vectors

int main() {

// Capture input from user to determine if the data is linearly separable or not

string input = "";

do {

if (input != "") printf("Invalid input. Please try again.\n");

printf("Trained with non-separable data set? (y/n) ");

cin >> input;

} while (input != "y" && input != "n" && input != "Y" && input != "N");

if (input == "y" || input == "Y") modifyData = true;

srand(time(NULL));

// Generate sample data and display for each class

generateAllSamples();

displayAllSamples();

// If need to modify data, modify it and display again

if (modifyData) {

modifyTrainedData(MODIFYINDEX);

modifyTrainedData(SAMPLESIZE);

displayAllSamples();

}

// Perform perceptron algorithm and display the result weights for each classifier

perceptron();

displayAllWeights();

// Test the tarined classifiers with second half of data and display testing result

testWeights();

displayTestResult();

system("pause");

return 0;

}

void generateAllSamples() {

for (int i = 0; i < 3; i++) {

generateRandomSamples(classSamples[i], i + 1);

}

}

void generateRandomSamples(double sample[SAMPLESIZE][DIMENSION + 1], int classNum) {

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

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

double phi = rand() % 3141 / double(1000) - 1.5705;

sample[j][0] = 1;

sample[j][1] = sphereCenter[classNum - 1][0] + RADIUS \* cos(theta) \* cos(phi);

sample[j][2] = sphereCenter[classNum - 1][1] + RADIUS \* sin(phi);

sample[j][3] = sphereCenter[classNum - 1][2] + RADIUS \* sin(theta) \* cos(phi);

}

}

void displayAllSamples() {

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

for (int i = 1; i < DIMENSION + 1; i++) displaySamples(i);

printf("\n\n");

}

void displaySamples(int classNum) {

printf("display class %d samples:\n", classNum);

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

if ((i == MODIFYINDEX - 1 && modifyData) || (i == SAMPLESIZE - 1 && modifyData)) printf("\*\*\*\*");

for (int j = 0; j < DIMENSION + 1; j++) {

if (j != 3)

printf("%f, ", classSamples[classNum - 1][i][j]);

else

printf("%f\n", classSamples[classNum - 1][i][j]);

}

}

printf("---------------------------------\n\n");

}

void displayAllWeights() {

printf("display all weights:\n");

for (int i = 1; i < CLASSNUM + 1; i++) displayWeights(i);

printf("\n\n");

}

void displayWeights(int classNum) {

printf("display class %d weights:\n", classNum);

for (int i = 0; i < DIMENSION + 1; i++) {

if (i != 3)

printf("%f, ", classWeights[classNum - 1][i]);

else

printf("%f\n", classWeights[classNum - 1][i]);

}

printf("\n");

}

void perceptron() {

bool allGreaterThanZero;

initializeWeights();

// For each classifier

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

int iterationCount = 0;

do {

allGreaterThanZero = true;

iterationCount++;

// Test with data for each class

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

// Test with all the data used to train for each class

for (int k = 0; k < TESTSIZE; k++) {

double val = vectorMultiplication(classWeights[i], classSamples[j][k]);

if (i != j) val = val \* -1;

if (val <= 0) {

for (int l = 0; l < DIMENSION + 1; l++) {

if (i != j)

classWeights[i][l] -= classSamples[j][k][l] \* WEIRDSYMBOL;

else

classWeights[i][l] += classSamples[j][k][l] \* WEIRDSYMBOL;

}

allGreaterThanZero = false;

}

}

}

} while (!allGreaterThanZero && iterationCount < MAXITERATION);

printf("%d iterations for class %d trained data.\n", iterationCount, i + 1);

}

printf("\n\n");

}

void testWeights() {

for (int i = 0; i < 3; i++) {

for (int j = TESTSIZE; j < SAMPLESIZE; j++) {

int classNum = 1;

double testValue[CLASSNUM];

// Store the values after testing with each classifier, the values represent the distance between the point and each decision boundary

for (int k = 0; k < CLASSNUM; k++) {

testValue[k] = vectorMultiplication(classWeights[k], classSamples[i][j]);

}

// Find the maximum value, which is the maximum distance among three classifieres

double max = testValue[0];

for (int k = 1; k < CLASSNUM; k++) {

if (max < testValue[k]) {

max = testValue[k];

classNum = k + 1;

}

}

// Store the result class, which is the one with the maximum distance

resultClass[i][j - TESTSIZE] = classNum;

}

}

}

void initializeWeights() {

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

for (int j = 0; j < DIMENSION + 1; j++) {

classWeights[i][j] = 0;

}

}

}

void displayTestResult() {

printf("display test result:\n");

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

printf("result for class %d: \n", i + 1);

for (int j = 0; j < SAMPLESIZE - TESTSIZE; j++) {

printf("%d\n", resultClass[i][j]);

}

printf("\n");

}

printf("--------------------------------\n\n");

}

void modifyTrainedData(int modifyIndex) {

modifyIndex--;

double class12Sample[2][DIMENSION] = { {classSamples[0][modifyIndex][1], classSamples[0][modifyIndex][2], classSamples[0][modifyIndex][3]},{ classSamples[1][modifyIndex][1], classSamples[1][modifyIndex][2], classSamples[1][modifyIndex][3] } };

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

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

if (i == 0) classSamples[i][modifyIndex][j] = classSamples[2][modifyIndex][j];

else classSamples[i][modifyIndex][j] = class12Sample[i - 1][j - 1];

}

}

}

double vectorMultiplication(double a[DIMENSION + 1], double b[DIMENSION + 1]) {

double sum = 0;

for (int i = 0; i < DIMENSION + 1; i++) {

sum += a[i] \* b[i];

}

return sum;

}