CSE 5523 HW 3

Lilong Jiang (jiang.573)

1. Problem 1

I use a HoldOut method for cross validation. 90% of training dataset is for training, 10% of training dataset is to calculate training error.

Bagged\_10 means there are 10 number of trees.

Boosted\_100 means 100 ensemble learning cycles.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Decision | Bagged\_10 | Bagged\_15 | Bagged\_20 | Boosted\_50 | Boosted\_100 | Boosted\_150 | Boosted\_200 |
| Training Error | 0.0550 | 0.0600 | 0.0300 | 0.0200 | 0.0600 | 0.0350 | 0.0300 | 0.0350 |
| Testing Error | 0.0475 | 0.0350 | 0.0210 | 0.0245 | 0.0615 | 0.0505 | 0.0495 | 0.0435 |

Usually, as the increase of number of tree increases or number of learning cycles, both training error and testing error will decrease and better than decision tree.

**Decision Tree Code:**

Y = [ones(1000, 1); -ones(1000, 1)];

load('train79.mat');

trainData = d79;

load('test79.mat');

testData = d79;

N = size(trainData, 1);

M = size(trainData, 2);

percent = 0.1;

[TrainInd, TestInd] = crossvalind(N, percent);

trainTrainN = size(TrainInd, 1);

trainTestN = size(TestInd, 1);

trainTrainData = zeros(trainTrainN, M);

trainTrainYs = zeros(trainTrainN, 1);

trainTestData = zeros(trainTestN, M);

for i = 1: trainTrainN

trainTrainData(i, :) = trainData(TrainInd(i), :);

if TrainInd(i) <= 1000

trainTrainYs(i) = 1;

else

trainTrainYs(i) = -1;

end

end

for i = 1: trainTestN

trainTestData(i, :) = trainData(TestInd(i), :);

end

tree = fitctree(trainTrainData, trainTrainYs);

predTrainLabels = predict(tree, trainTestData);

trainError = 0;

for i = 1 : trainTestN

if TestInd(i) <= 1000

trueLabel = 1;

else

trueLabel = -1;

end

if predTrainLabels(i) ~= trueLabel

trainError = trainError + 1;

end

end

trainError = trainError / trainTestN

testN = size(testData, 1);

predTestLabels = predict(tree, testData);

testError = 0;

for i = 1: testN

if i <= 1000

trueLabel = 1;

else

trueLabel = -1;

end

if predTestLabels(i) ~= trueLabel

testError = testError + 1;

end

end

testError = testError / testN

**Bagged Tree Code:**

treeNum = 20;

B = TreeBagger(treeNum, trainTrainData, trainTrainYs);

predTrainLabels = predict(B, trainTestData);

trainError = 0;

for i = 1 : trainTestN

if TestInd(i) <= 1000

trueLabel = 1;

else

trueLabel = -1;

end

if str2double(predTrainLabels{i}) ~= trueLabel

trainError = trainError + 1;

end

end

trainError = trainError \* 1.0 / trainTestN

testN = size(testData, 1);

predTestLabels = predict(B, testData);

testError = 0;

for i = 1: testN

if i <= 1000

trueLabel = 1;

else

trueLabel = -1;

end

if str2double(predTestLabels{i}) ~= trueLabel

testError = testError + 1;

end

end

testError = testError \* 1.0 / testN

**Boosted Tree Code:**

ens = fitensemble(trainTrainData, trainTrainYs, 'AdaBoostM1', 200, 'Tree');

predTrainLabels = predict(ens, trainTestData);

trainError = 0;

for i = 1 : trainTestN

if TestInd(i) <= 1000

trueLabel = 1;

else

trueLabel = -1;

end

if predTrainLabels(i) ~= trueLabel

trainError = trainError + 1;

end

end

trainError = trainError \* 1.0 / trainTestN

testN = size(testData, 1);

predTestLabels = predict(ens, testData);

testError = 0;

for i = 1: testN

if i <= 1000

trueLabel = 1;

else

trueLabel = -1;

end

if predTestLabels(i) ~= trueLabel

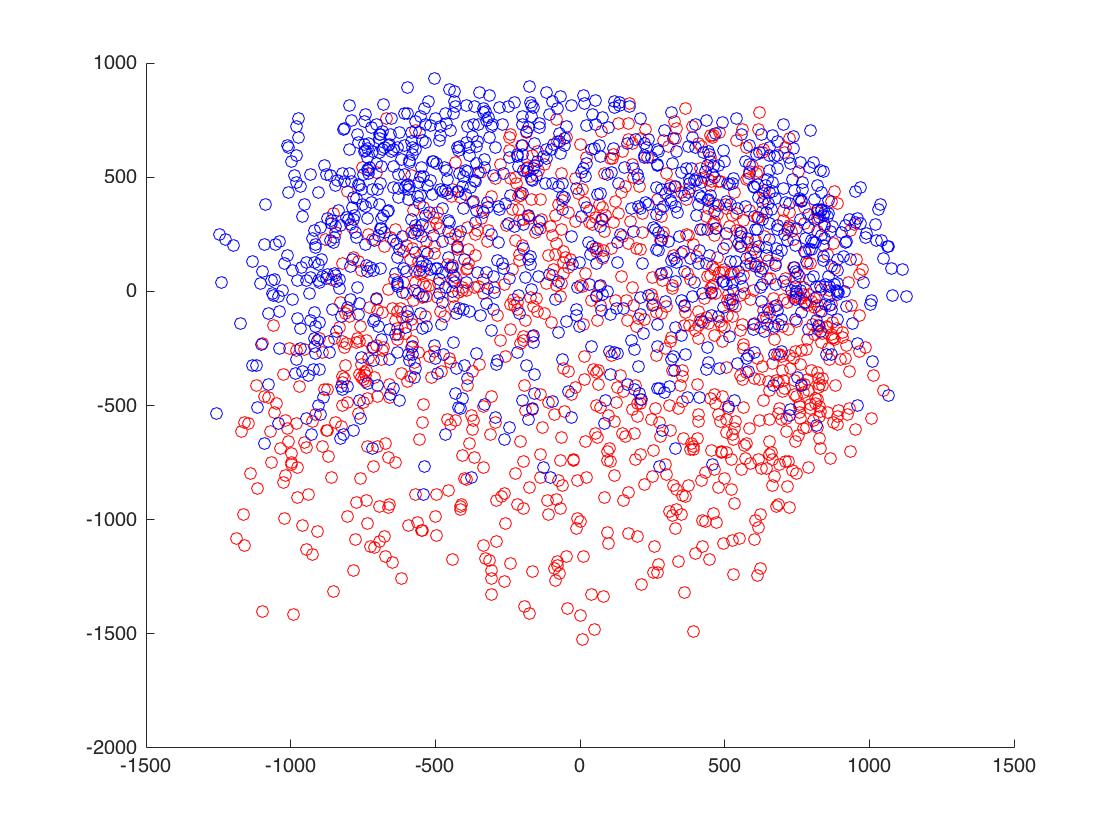
testError = testError + 1;

end

end

testError = testError \* 1.0 / testN

1. Problem 2
2. The visualization figure is shown below. This is performed on the training data.



**Code is shown below:**

load('train79.mat');

trainData = d79;

trainData = bsxfun(@minus, trainData, mean(trainData, 1));

C = cov(trainData);

[V D] = eig(C);

[D order] = sort(diag(D), 'descend');

V = V(:, order);

trainData = trainData \* V(:, 1: 2);

cdata = [ones(1000, 1) \* [1 0 0]; ones(1000, 1) \* [0 0 1]];

size(trainData)

scatter(trainData(:, 1), trainData(:, 2), 'o', 'cdata', cdata)

1. Figure of Class 7 is shown below. It seems that it captures two type of 7s in each figure.

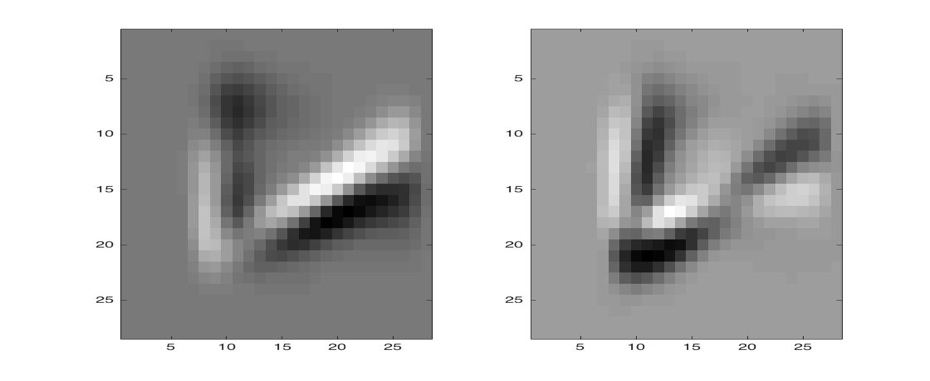


Figure of Class 9 is shown below. It seems that it captures two type of 9s in each figure.

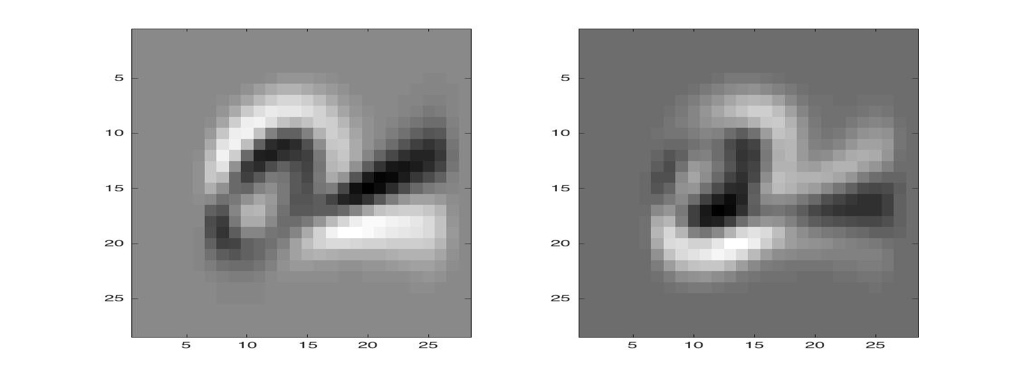
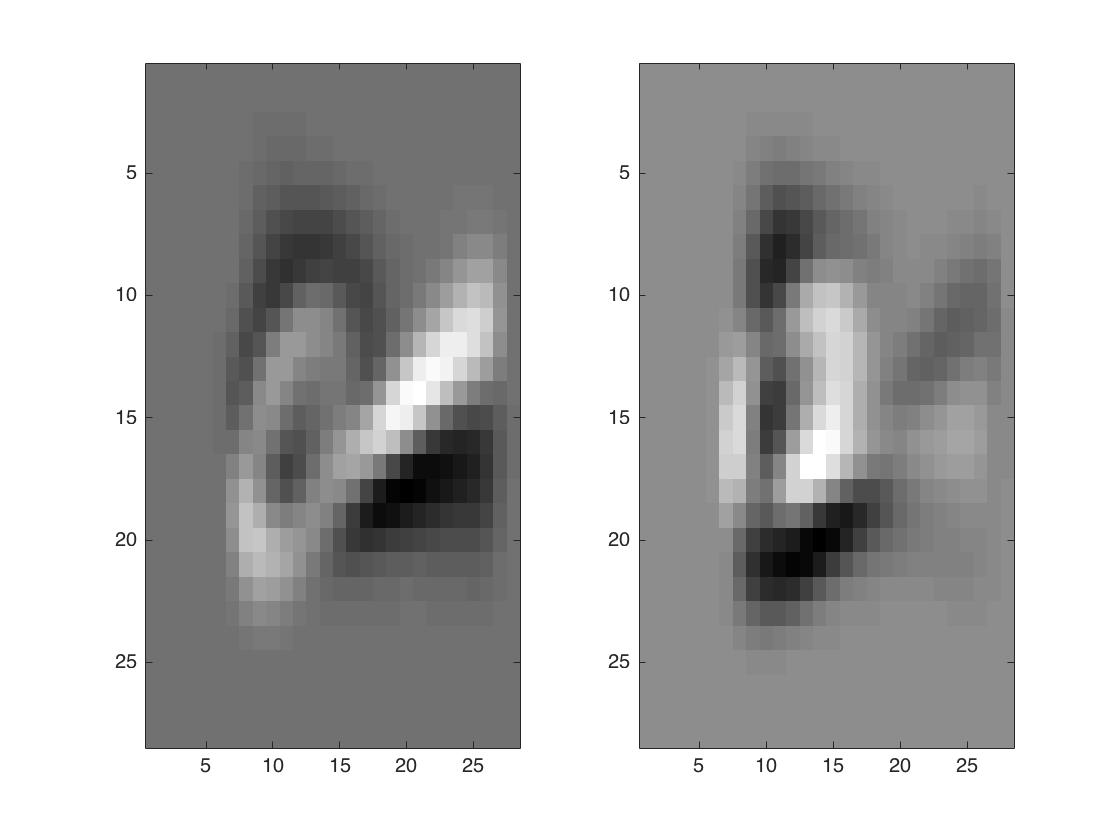


Figure of both classes are shown below. It seems the left figure captures 9 and another figure captures 7.



**Code is shown below:**

load('train79.mat');

trainData = d79;

load('test79.mat');

testData = d79;

X1 = [trainData(1: 1000, :); testData(1: 1000, :)];

X2 = [trainData(1001: 2000, :); testData(1001: 2000, :)];

X3 = [X1; X2];

X = X3;

X = X';

X = bsxfun(@minus, X, mean(X, 2));

s = cov(X');

[V, D] = eig(s);

[D order] = sort(diag(D), 'descend');

V = V(:, order);

figure,subplot(1, 2, 1)

colormap gray

for i = 1:2

subplot(1, 2, i)

imagesc(reshape(V(:, i), 28, 28))

end

1. Problem 3

I run clustering algorithms on test dataset.

The error rate is calculated in the following way, for the first 1000 images, I will pick up the largest cluster, all points out of the largest cluster is classified as error. The same for the later 1000 pages. The error rate is shown below. It shows that single-linkage is much better than k-means, possible reason is that cluster function constructs a maximum of n clusters using the 'distance' criterion and don’t guarantee that the number of clusters generated is same as setting. Both method’s error rate is increasing with cluster num, which is a characteristic of our metric.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of clusters | 2 | 5 | 10 | 50 |
| k-means | 0.3965 | 0.7400 | 0.7700 | 0.9325 |
| Single-linkage | 5.0000e-04 | 0.0020 | 0.0050 | 0.0260 |

**Code for K-means:**

load('test79.mat');

testData = d79;

ks = [2, 5, 10, 50];

for kIdx = 1: size(ks, 2)

error = 0;

k = ks(kIdx);

clusterLabels = kmeans(testData, k);

clusterNums = zeros(k, 1);

for i = 1: 1000

label = clusterLabels(i);

clusterNums(label) = clusterNums(label) + 1;

end

[maxVal, maxIdx] = max(clusterNums);

error = 1000 - maxVal;

clusterNums = zeros(k, 1);

for i = 1001: 2000

label = clusterLabels(i);

clusterNums(label) = clusterNums(label) + 1;

end

[maxVal, maxIdx] = max(clusterNums);

error = error + 1000 - maxVal;

error = error \* 1.0 / 2000

end

**Code for single-linkage clustering:**

load('test79.mat');

testData = d79;

distMat = pdist(testData);

Z = linkage(distMat, 'single');

% dendrogram(Z)

ks = [2, 5, 10, 50];

for kIdx = 1: size(ks, 2)

error = 0;

k = ks(kIdx);

clusterLabels = cluster(Z,'maxclust', k) ;

clusterNums = zeros(k, 1);

for i = 1: 1000

label = clusterLabels(i);

clusterNums(label) = clusterNums(label) + 1;

end

[maxVal, maxIdx] = max(clusterNums);

error = 1000 - maxVal;

clusterNums = zeros(k, 1);

for i = 1001: 2000

label = clusterLabels(i);

clusterNums(label) = clusterNums(label) + 1;

end

[maxVal, maxIdx] = max(clusterNums);

error = error + 1000 - maxVal;

error = error \* 1.0 / 2000

end