Problem 1. Apply (1) decision trees, (2) bagged and (3) boosted decision trees to the digit datasets from Homework 2. (You may use the standard libraries of Matlab or download Matlab code from the Web.) Use appropriate cross validation on the training set. Compare performance.

***Code:***

train = load('train79.mat');

train=train.d79;

test = load('test79.mat');

test=test.d79;

label = vertcat(ones(1000,1)\*1, ones(1000,1)\*-1);

N=2000;

%% Decision trees

DT = fitctree(train,label, 'CrossVal','on');

crossValFun = @(x)sum(x.IsBranch);

crossValResult = cellfun(crossValFun, DT.Trained);

figure;

max(crossValResult) % Find the maximum split.

lossList = zeros(38,1);

for split=1:38

sDT=fitctree(train, label, 'CrossVal','on','MaxNumSplits',split);

lossDT = kfoldLoss(sDT);

lossList(split)=lossDT;

end

[lossListSorted , inx] = sort(lossList);

sDT=fitctree(train, label, 'MaxNumSplits',inx(1));

testDTResult = sDT.predict(test);

DTDiff = testDTResult - label;

DTLoss = transpose(DTDiff)\*DTDiff/4/N

%% Bagged trees

BT = fitcensemble(train,label,'Method','Bag','NumLearningCycles',500,'CrossVal','on');

kFoldLossList=kfoldLoss(BT,'mode','cumulative');

[optimalBT, index]=sort(kFoldLossList);

BTOptimal=fitcensemble(train,label,'NumLearningCycles',index,'Method','Bag');

testBTResult=BTOptimal.predict(test);

BTDiff = testBTResult - label;

BTLoss = transpose(BTDiff)\*BTDiff/4/N

%% Boosted trees

BoT = fitcensemble(train,label,'Method','AdaBoostM1','NumLearningCycles',500,'CrossVal','on');

kFoldLossBoTList=kfoldLoss(BoT,'mode','cumulative');

[optimalBoT, index2]=sort(kFoldLossBoTList);

BoTOptimal=fitcensemble(train,label,'NumLearningCycles',index2,'Method','AdaBoostM1');

testBoTResult=BoTOptimal.predict(test);

BoTDiff = testBoTResult - label;

BoTLoss = transpose(BoTDiff)\*BoTDiff/4/N

***Analysis:***

For a single decision tree, a major parameter that decide its performance and whether it is overfitting is its depth, which can be also measured by split number. The result shows the largest split count is 38.

Based on the maximum, we controlled “MaxNumSplits” during the process of cross validation and found the best loss rate and corresponding classifier. Then we predicted the optimal model using the test dataset.

For bagged and boosted trees, it is hard to control maximum number of splits. Instead, I optimized the classifiers over the learning cycles with maximum of 500. Figure 1 and Figure 2 shows the learning process of bagged and boosted trees.

Table 1 shows the final error rate obtained by cross validation. Single decision tree’s performance is worse than bagged trees and AdaBoosted trees. It proves that boosted classifiers outperform single classifier.

Table 1 Final error rate for decision trees, bagged trees, and boosted trees.

|  |  |
| --- | --- |
| Classifier | Cross validation error |
| Decision trees | 0.0460 |
| Bagged trees | 0.0180 |
| Boosted trees | 0.0160 |

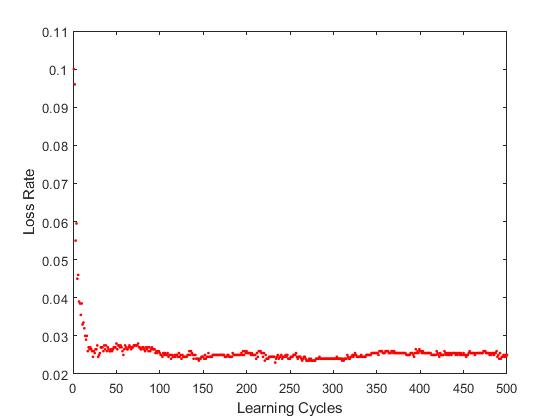


Figure 1 Bagged tree's loss rate with learning cycles

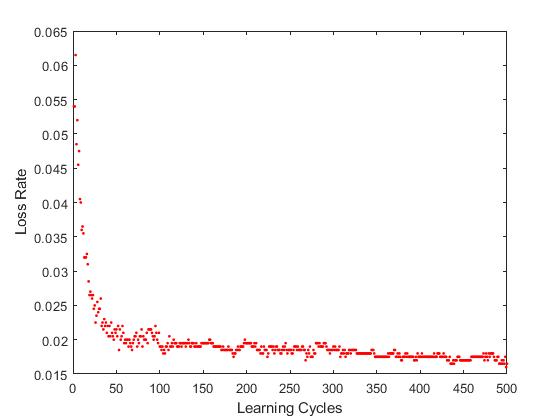


Figure 2 Boosted tree's loss rate with learning cycles

Problem 2.

1. Implement PCA and apply it to the digit data, reducing the dimension to two. Visualize the data after dimensionality reduction using colors for different classes.

***Code:***

function [PCA, newTrain] = PCA\_eig(train, k)

normalizedTrain = train - repelem(mean(train),size(train,1),1);

covTrain = normalizedTrain'\*normalizedTrain/size(train,1);

[eigenvectors,eigenvaluesMatrix] = eig(covTrain);

[eigenvaluesOrdered,ind] = sort(diag(eigenvaluesMatrix),'descend');

eigenvectorsOrdered = eigenvectors(:,ind);

PCA = eigenvectorsOrdered(:, 1:k);

newTrain = train\*PCA;

end

test=load('test79.mat');

test=test.d79;

train=load('train79.mat');

train=train.d79;

label = vertcat(ones(1000,1)\*1, ones(1000,1)\*-1);

N=2000;

d=784;

k=2;

train\_7 = train (1:1000,:);

train\_9 = train (1001:2000, :);

[PCA, newTrain] = PCA\_eig(train, k)

%% Visualization

figure(1)

scatter(newTrain(1:1000,1),newTrain(1:1000,2),'.b');

hold on

scatter(newTrain(1001:2000,1),newTrain(1001:2000,2),'.r');

***Analysis:***

Figure 3 shows the distribution of data after PCA in a 2-D space. We can see two classes are somewhat separable: sevens are red dots which resides in the top, while nines are blue which resides in the bottom. However, we can also observe there is a huge mixed cluster in the middle. This is due to the massive information loss during the process of PCA.

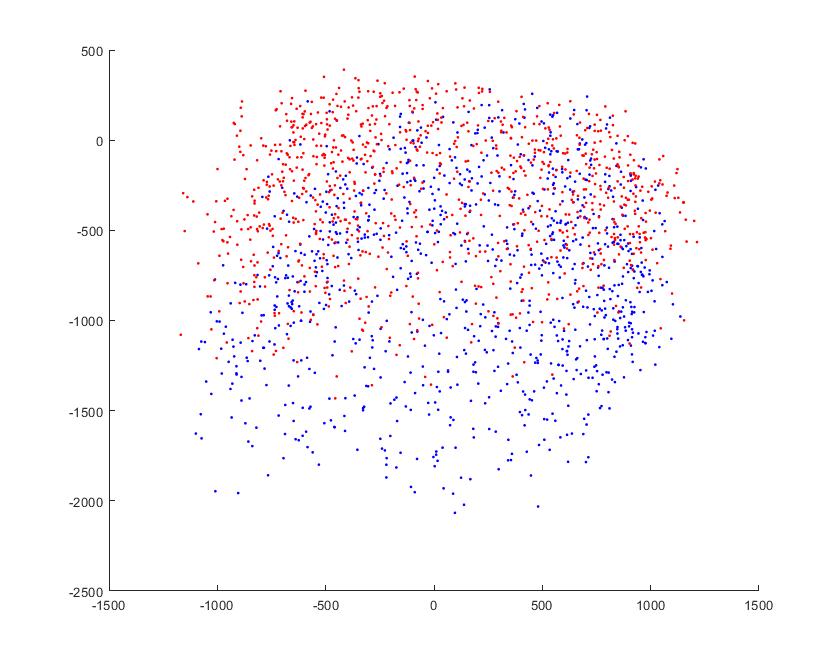


Figure 3 The visualization of reduced data after PCA in 2D space. (Red: seven; Blue: nine)

1. Produce pictures of “eigendigits” for the dataset combining both classes and for each class separately. Observations?

***Code:***

figure(2)

subplot(1,2,1)

x = reshape (PCA(:,1),28,28);

pcolor(x);

title('Eigendigit 1');

daspect([1 1 1])

subplot(1,2,2)

x = reshape (PCA(:,2),28,28);

pcolor(x);

title('Eigendigit 2');

daspect([1 1 1])

***Analysis:***

Figure 4 shows the ‘Eigendigits” picture for dataset combining both sevens and nines classes. The first picture shows there is a highlighted “9” shape, with a darkened “9” in the back ground. The other cells in the outskirts shows no significant high or low values. The second picture shows a darkened “7” shape in the front, with a blurred highlighted “9” shape in the back. The intersection of two shapes are blurred and mixed.

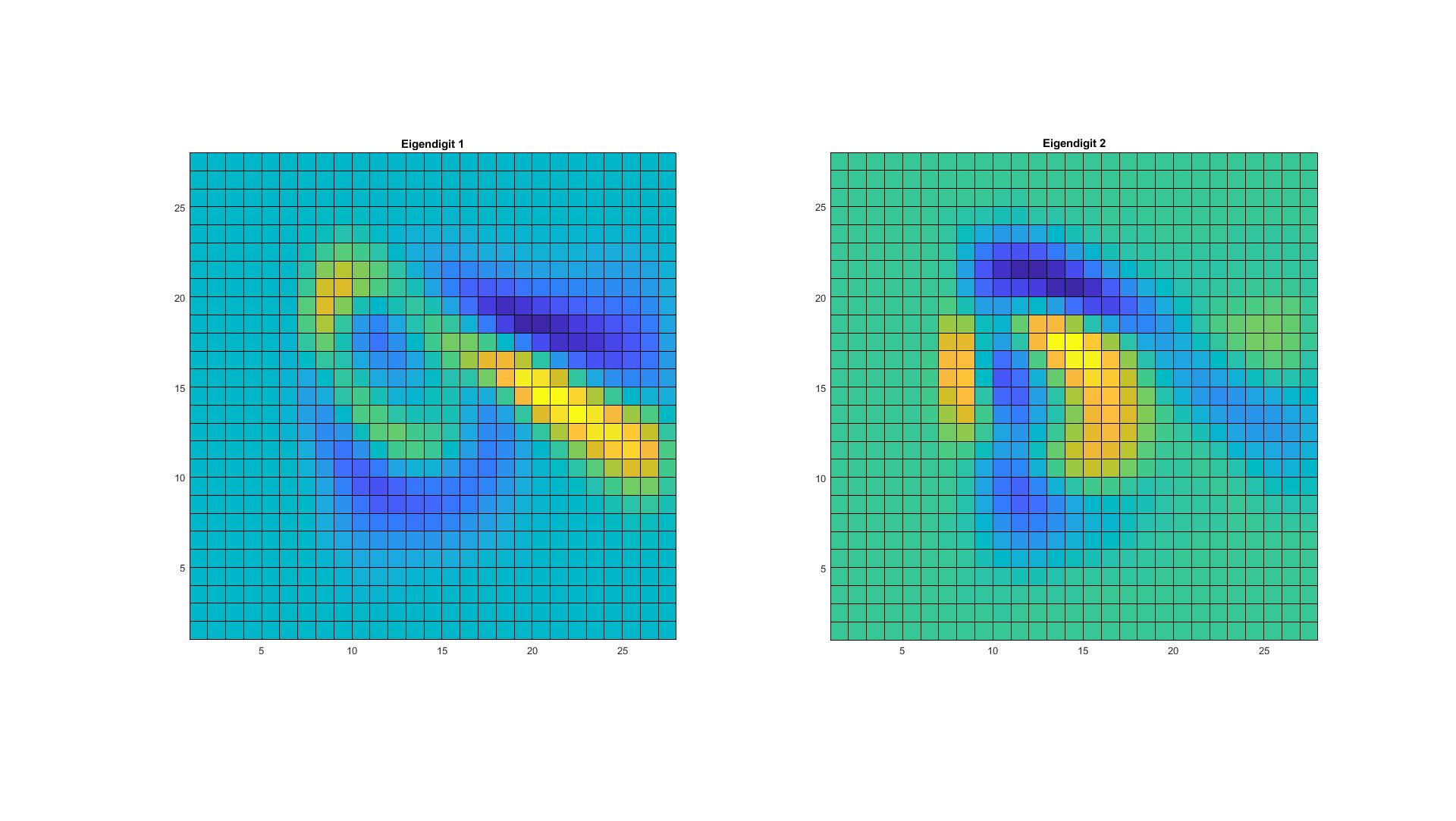


Figure 4 Eigendigits for dataset combining both classes.

Figure 5 and Figure 6 shows eigendigits for dataset of sevens and dataset of nines. First and second pictures all show a clear pattern of “7”, both for front and back layers. Also for Figure 6, which shows “9” in the front and back layers for both pictures.

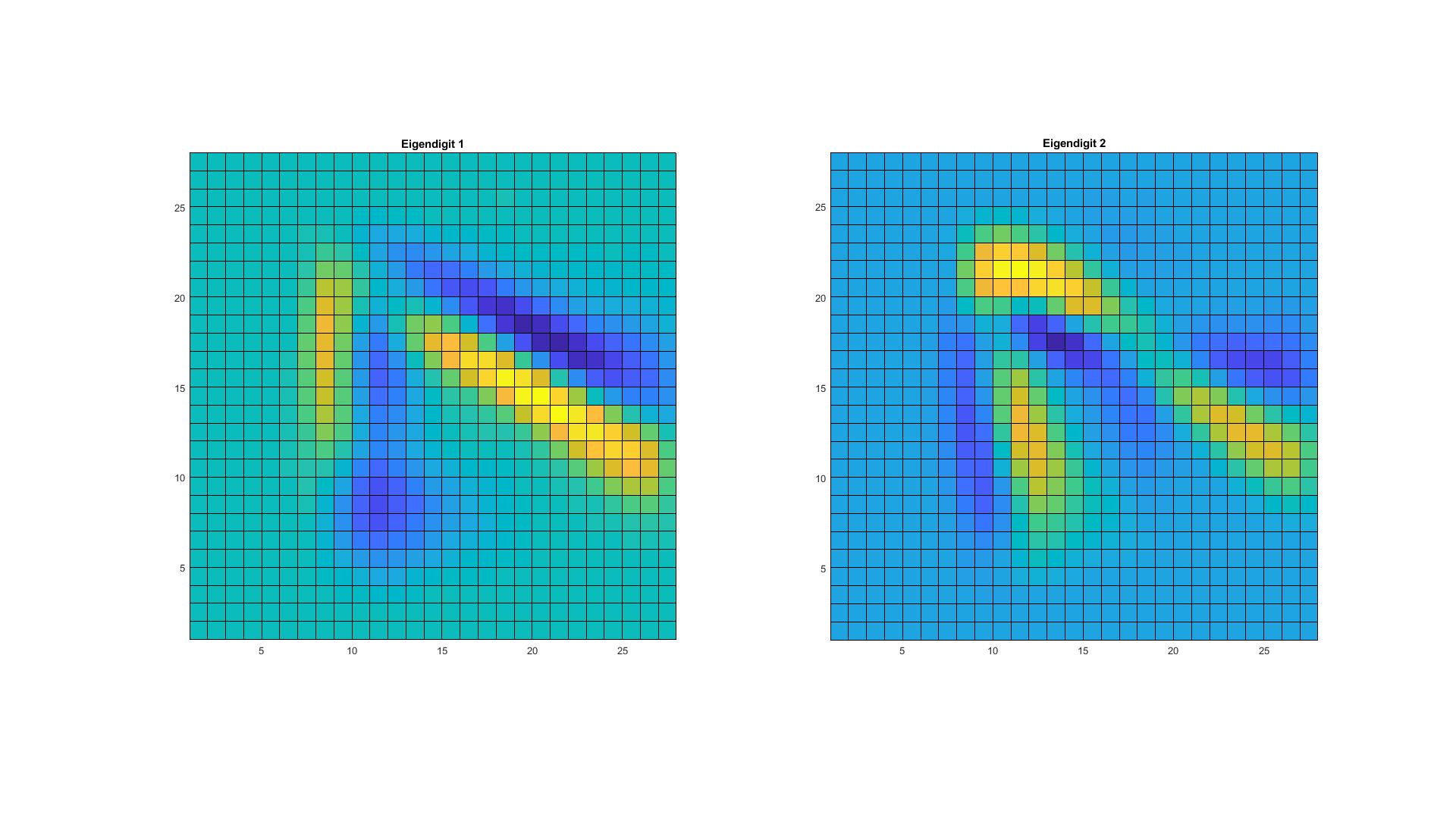


Figure 5 Eigendigits for dataset of sevens

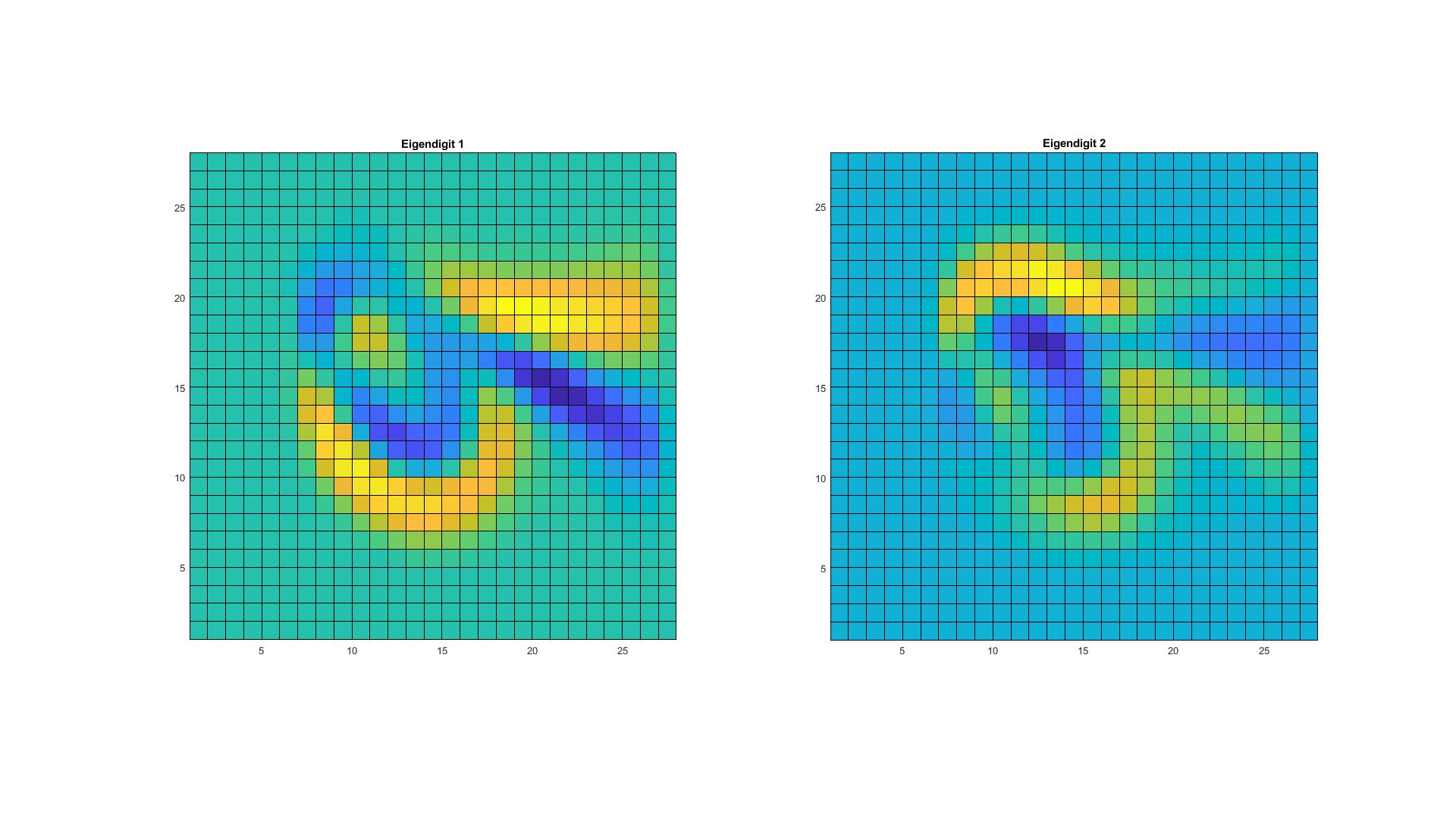


Figure 6 Eigendigits for dataset of nines

Problem 3. Apply k-means clustering to the digits data set for k = 2, 5, 10, 50. How well does it identify the different digits? (Note that clustering is unsupervised – how do you compare classification and clustering results?)

***Code:***

test=load('test79.mat');

test=test.d79;

train=load('train79.mat');

train=train.d79;

label = vertcat(ones(1000,1)\*1, ones(1000,1)\*-1);

n=size(train, 1);

d=size(train, 2);

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

lossList=ones(length(kList),1);

for i=1:length(kList)

k=kList(i);

[kMeansResult] = kmeans(train,k); % K-means

for cluster=1:k

isMember=ismember(kMeansResult,cluster);

% number of points belong to 7 and 9

seven=sum(isMember(1:1000,1));

nine=sum(isMember(1001:2000,1));

if seven>=nine

thisClass=7;

kMeansResult(isMember)=1;

else

thisClass=9;

kMeansResult(isMember)=-1;

end

end

diff = kMeansResult - label;

loss = diff'\*diff/4/n;

lossList(i)=loss;

end

***Analysis:***

To do the classification, I first applied K-means to the dataset without label, since K-means is an unsupervised method. For each cluster, assign seven or nine according to which class the cluster belong to. It assumes each cluster is like a characteristic of a class, such as the sharp turning in “7”.

Figure 7 and Table 1 show the relationship between k and loss rate. It is obvious that the loss becomes less when k increases. Smaller k means smaller clusters’ size. With more delicate structures, the digit pictures can be easier to be recognized by the algorithm. More clusters also increase the resolution of the classification, making the whole process less ambiguous.

For k = 2, the loss is 0.4275, which is pretty close to random guessing. However, for k = 25, the loss is 0.0675, which is close to decision tree in problem 1.

Table 2 Relationship between k and loss rate

|  |  |
| --- | --- |
| k | Loss |
| 2 | 0.4275 |
| 5 | 0.2735 |
| 10 | 0.1720 |
| 25 | 0.0675 |

Figure 7 Relationship between k and loss rate