Title: Classifying Digits

Author: Wendy Jiang

Abstract: This assignment is to classify digits of images. We are given 0-to-9-digit images. We should reshape the data and do different types of classification of the training data: LDA, SVM, and the decision trees. Then we calculate the accuracy of each method by the test data and compare the differences among those classification methods.

Section I. Introduction and Overview

This assignment aims to apply how to do a linear discriminant analysis (LDA) to identify 0-to-9, totally 10 digits. To achieve our goal, we need to reshape data first, set each image into a column vector and each column of our data matrix is a different image. Then, use SVD to analysis in order to get the singular value spectrum and make sure the rank of the digit space. After that, we build the classifier, using the mode we got before as the feature we use, we can reasonably identify digits. So, we test our classifier's accuracy by comparing two or three digits at the same time, then we compare the difference of each label to the actual labels to determine how accurate it is. Then we could summary which two digits are most easy to separate and which twos are most difficult to separate. Then we have other methods to build classifier in MATLAB as well, such as SVM and decision trees, compare their performance and pros and cons is a meaningful task.

Section II. Theoretical Background

1. The strategy to build an image classifier: (1) Use a wavelet transform on each image to detect the image. (2) Find the principal component of the wavelet transforms to see how dogs and cats differ in principal component basis. (3) Use linear discriminant analysis to determine some threshold that separates images. (4) Test the algorithm on new data (test data) to see its accuracy.

2. We can use discreate wavelet transform to gain information about the edges in out images. Then, we need to find a statistical method to tell the difference among images' edges. Principal component analysis (PCA) can tell us about the ways in which we have the most variation in our data, to pick out some of the most important features of our data.

3. Goal of LDA: Find a suitable projection that maximizes the distance between inter-class data while minimizing the intra-class data. To find the right subspace to project onto, we should calculate the means for each of groups first, called means . Note that are column vectors. If we consider two Here is the between-class scatter matrix:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T. \quad \dots (1)$$

   This is a measure of the variance between the groups (between the means). Define the within-class scatter matrix:

$$S_w = \sum_{j=1}^{2} \sum_{X} (X - \mu_j)(X - \mu_j)^T \ldots (2)$$

This is a measure of the variance within each group. The goal is then to find a vector such that $w = argmax \frac{w^T S_B w}{w^T S_w w} \ldots (3)$

The vector maximizes the above quotient is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem: $S_B w = \lambda S_B w \ldots (4)$

Once we have using the above (4) operations, we can choose a cutoff between two thresholds for decision making.

There are functions of changing scatter matrices for multiple groups:

$$S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T \ldots (5)$$

$$S_w = \sum_{j=1}^{N} \sum_{X} (X - \mu_j)(X - \mu_j)^T \ldots (6)$$

4. It would always not good enough with supervised machine learning to do a good job on the training data. We could easily get them all right by having the computer memorize the image and look them up in a table. When we use the classification and produce the determinant to distinguish the different images, remember we need to first perform a wavelet transform, then PCA, then project onto the line we found from LDA. We can check which images are classified as the images we want be classified by using the threshold found during the training. And we can check errors using the hiddenlabels vector included with the testing data set, also the success on the testing data.

5. Classification and Regression Tree (CART) analysis is a very common modeling technique used to make prediction on a variable (Y), based upon several explanatory variables, X1,X2,...Xp. The term Classification Tree is used when the response variable is categorical, while Regression Tree is used when the response variable is continuous. CART analysis is very common because it works well for a wide variety of data sets and is easy to interpret. In addition, it does not require model assumptions (i.e. the variables do not need to follow any distributional patterns).

6. SVM classifier: For greater accuracy and kernel-function choices on low- through medium-dimensional data sets, train a binary SVM model or a multiclass error-correcting output codes (ECOC) model containing SVM binary learners using the Classification Learner app. For greater flexibility, use the command-line interface to train a binary SVM model using fitcsvm or train a multiclass ECOC model composed of binary SVM learners using fitcecoc.

Algorithm Development

We deal with the training data first, build the classifier and use the same command for the test data. Before doing the classifier, we use minst_parse to load the data set into the workspace. Here we would get a set with 60000 images and the dimensions of each digit is 28 x 28. We need to reshape the image into a column vector, and each column of the data matrix is a different image. With the data matrix, we subtracted the mean of each row from the data to center the data. Then we use singular value decomposition (SVD). The SVD is a factorization that generalizes the eigen decomposition of a square normal matrix to any matrix via an extension of the polar decomposition. Specifically, the SVD of an matrix is a factorization of the from . Thus the SVD decomposition breaks down any invertible linear transformation into a composition of three geometrical transformations: a rotation or reflection (), followed by a coordinate-by-coordinate scaling (), followed by another rotation or reflection (U). In short, the columns of , , , and are orthonormal bases. Then we could draw the singular value spectrum and figure out how many modes are necessary for good image instruction as the feature. Here we set feature equals 50. To find the projection of each digits, we multiply the with the training data and then separate the projections of every digits. Then we create a 3-D plot of projections onto three selected V-modes, here I select the first three columns.

Then we build the classifier function using a linear discriminant analysis (LDA). We pick two digits of test data first. We set a new matrix which including these two digits' data and then we reuse SVD. We start LDA by taking advantage of the SVD that we already computed by projecting onto some number of PCA modes. Then calculate the scatter matrix and find using the eig() MATLAB command with two arguments. To project onto we just multiply our data with . Then find the threshold for our classifier. Later on, it will be easier for us to make decision if we have consistency for whether our images data above or below the threshold. We can see overlap by the plot of two digits' values. Then we do the process of testing. We select the same two digits as before and combine these two digits' data in a matrix. Here's a array called "hiddenlabels" which is a vector of 0's and 1's (0 for the first digit and 1 for the second digit) which we can use to check our performance against without having to look through each image individually. After that, we use classification procedure to determine which images were classified as digits we selected. We want to 0 if we are below the threshold and 1 if we are above the threshold. We could check the errors using the hiddenlabels vector included with the testing data set, also we can get the success rate, or called the accuracy of our testing. Same process here when we select three different digits and get the success rate of testing. Based on the success rate (accuracy), we can figure out which two digits are the easiest to separate and which two are the most difficult to separate.

Then we use other method to build the classifier, such as the decision tree and SVM. We compare the difference with these methods.

Computational Result

Figure 1 is the plot of singular spectrum. Singular values encode magnitude of the semiaxis, while singular vectors encode direction. As we see, the value was decreasing to around 100 after 50 modes. After we use SVD command, the diagonal value of matrix S is the Singular Value.

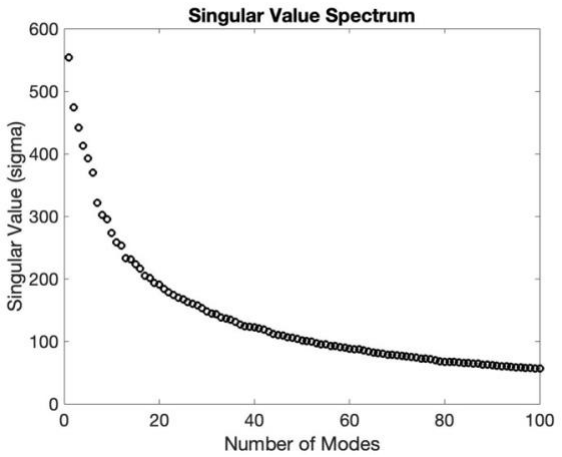

Figure 1. Singular Value Spectrum

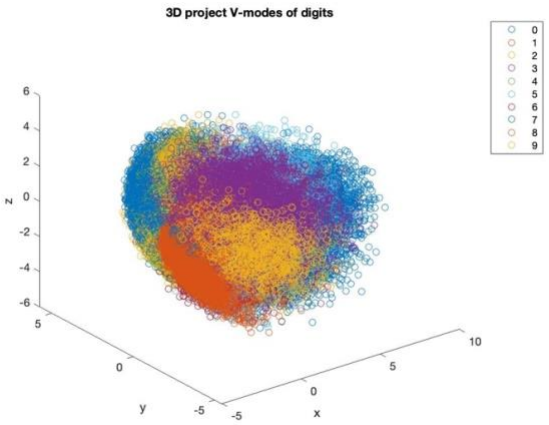

Figure 2. 3-D plot of project onto three V-modes

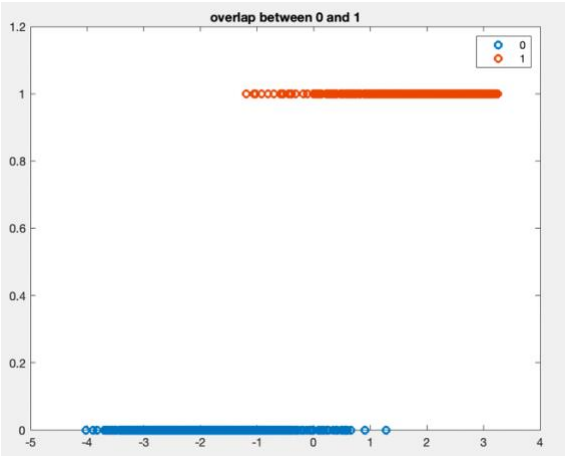

Figure 3. Overlap between digit 0 and 1

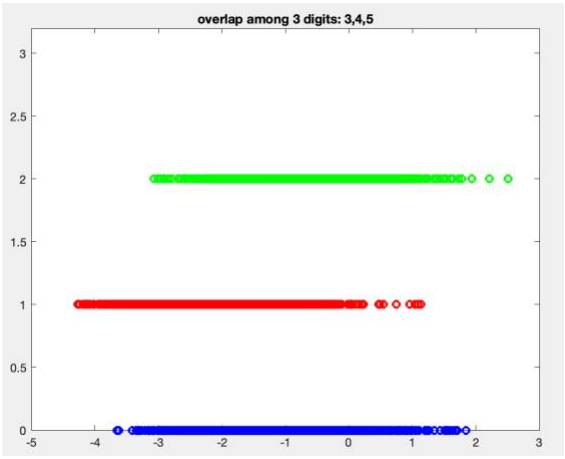

Figure 4. Overlap among digit 3(blue), 4(red) and 5(green)

Figure 2 is the projections of each digit. It seems like a cloud. For each digit, their projections are close. So, each digit's projections are a gobbet and 10 gobbets being together in this plot. I choose the first three columns projection data to plot. The projection data comes from the first three columns of U multiple by the training data, and then set the label of each digit, for example, find(label == 0) to find the whole projections of digit 0. After that we plot the first three columns of the projections.

Figure 3 is when we pick two digits and build the LDA to reasonable identify these digits. Here I select 0 and 1. We could see there's overlap exists. Figure 4 is when we pick three digits, I choose 3, 4, and 5 here. Overlap also exist. Compare these lines, 3 and 5 have much more overlap points.

When we choose digits 0 and 1, the threshold equals to 0.0026. When we select 4 and 9 two digits, the threshold equals to -0.2791(features always be 50). As the result, the easiest digits to separate is 0 and 1, and the hardest digits to separate is 4 and 9.Here is the chart of testing the test data and find the easiest and hardest two digits:

Digits	Success Rate	threshold
0 and 1 (highest)	0.9962	0.0026
4 and 9 (lowest)	0.9543	-0.2791

When we do the decision tree classifier, we use the command fitctree() to get the decision tree. Inside this command, we set the transpose of the trainingdata and label of the training data as the input. Then we use the kflodLoss, it returns loss obtained by cross-validated classification model obj. For every fold, this method computes classification loss for in-fold observations using a model trained on out-of-fold observations. We can set `[~,A] = min(classError)` to get the most minimum error in this set. Then we predict on the test data. And get the labels. Then we create a for loop to compare the difference between our prediction with the original label to get the accuracy. In my code, the accuracy of this method is clearly lower than the SVM and LDA classification. Although every time it would return different result, the accuracy is between 0.5 and 0.6 when I input the whole training data set. But if I only choose the two digits, it would be get much higher result, the accuracy over 0.9. The advantages of the decision tree classification are that compared to other algorithms decision trees requires less effort for data preparation during pre-processing and the decision tree does not require normalization of data. However, there are some disadvantages, such as A small change in the data can cause a large change in the structure of the decision tree causing instability.

Then we use the SVM classifier. We build the model classifier first and use the classifier to predict as well. `Mdl = fitcecoc(trainingdata',traingnd,'CrossVal','on')` This is the command to build the classifier. We set the transform of the training data and the label of the training data as input.

The SVM has a better performance than decision tree and LDA. If we select the two groups of digits we used before, we can compare their accuracy, SVM always the most accurate one. LDA classifier is more stable than the decision tree classifier.

If we do the whole dataset, using the whole training data to build the classifier and test all testing data, SVM has accuracy more than 0.95, but decision tree only has 0.56 or so. There's a big difference. However, the SVM would take much more time.

Here is the chart for the accuracy when we compare the same two digits by different methods:

Digits	Decision tree	LDA	SVM
4 and 9	0.9221	0.9543	0.9709
0 and 1	0.9976	0.9962	0.9991

Summary
In this assignment, we practice how to build classifier of 0 to 9 digits in three different methods: LDA, the decision tree, and SVM. After we build the classifier of the training data, we use testing data to test the accuracy of each method. SVM is the most accurate way to build classifier. LDA is more complex since we need to build the function by hand. Then the decision tree classifier is the one been influence easiest, in other word, the less stable one if we change the data of it.

Appendix A. MATLAB function

- label = kfoldPredict(obj): returns class labels predicted by obj, a cross-validated classification. For every fold, kfoldPredict predicts class labels for in-fold observations using a model trained on out-of-fold observations.
- `[trainingdata, traingnd] = mnist_parse()` : load in the implementation on the MNIST database
- `find(X)` : Returns a vector of non-zero indices in a matrix. It enables to locate the bright spots of each images. Also, it corresponded to moving object.
- `diag(X)`: Returns a column vector of the diagonal values of a matrix. It can generate the variance number by SVD.
- `M = mean(A)`: returns the mean of the elements of A along the first array dimension whose size does not equal 1.
- B = repmat(A,n): returns an array containing n copies of A in the row and column dimensions. The size of B is size(A)*n when A is a matrix.
- label = predict(Mdl,X) :returns a vector of predicted class labels for the predictor data in the table or matrix X, based on the trained, full or compact classification tree Mdl.
- Mdl = fitcecoc(Tbl,Y) :returns an ECOC model using the predictors in table Tbl and the class labels in vector Y.

Appendix B. MATLAB code

```
%%
% clear workspace
close all; clear all; clc
%%
% Reshape data (each column is a picture)
[trainingdata, traingnd] = mnist_parse('train-images.idx3-ubyte', 'train-
labels-idx1-ubyte');
trainingdata = im2double(reshape(trainingdata, size(trainingdata,1) *
size(trainingdata,2), []));
traingnd = double(traingnd);
mn =  mean(trainingdata,2);
trainingdata = trainingdata - repmat(mn,1,60000);
```

```
[testdata, testgnd] = mnist_parse('t10k-images.idx3-ubyte', 't10k-
labels.idx1-ubyte');
testdata = im2double(reshape(testdata, size(testdata,1) * size(testdata,2),
[]));
testgnd = double(testgnd);
testdata = testdata - repmat(mn,1,10000);
%%
[U,S,V] = svd(trainingdata, 'econ');
lambda = diag(S).^2;


figure(1)
plot(diag(S),'ko','Linewidth',2)
set(gca,'Fontsize',16,"Xlim",[0,100])
ylabel("Singular Value (sigma)")
xlabel("Number of Modes")
title("Singular Value Spectrum")
%%
figure(2)
plot(lambda/sum(lambda),'bo','Linewidth',2);
set(gca,'Fontsize',16);
%%
proj = U(:,[1,2,3])'* trainingdata;

proj_0 = proj(:,traingnd == 0);
proj_1 = proj(:,traingnd == 1);
proj_2 = proj(:,traingnd == 2);
proj_3 = proj(:,traingnd == 3);
proj_4 = proj(:,traingnd == 4);
proj_5 = proj(:,traingnd == 5);
proj_6 = proj(:,traingnd == 6);
proj_7 = proj(:,traingnd == 7);
proj_8 = proj(:,traingnd == 8);
proj_9 = proj(:,traingnd == 9);

figure(2)
plot3(proj_0(1,:),proj_0(2,:),proj_0(3,:),"o");hold on
plot3(proj_1(1,:),proj_1(2,:),proj_1(3,:),"o");hold on
plot3(proj_2(1,:),proj_2(2,:),proj_2(3,:),"o");hold on
plot3(proj_3(1,:),proj_3(2,:),proj_3(3,:),"o");hold on
plot3(proj_4(1,:),proj_4(2,:),proj_4(3,:),"o");hold on
plot3(proj_5(1,:),proj_5(2,:),proj_5(3,:),"o");hold on
plot3(proj_6(1,:),proj_6(2,:),proj_6(3,:),"o");hold on
plot3(proj_7(1,:),proj_7(2,:),proj_7(3,:),"o");hold on
plot3(proj_8(1,:),proj_8(2,:),proj_8(3,:),"o");hold on
plot3(proj_9(1,:),proj_9(2,:),proj_9(3,:),"o");


title('3D project V-modes of digits')
legend('0','1','2','3','4','5','6','7','8','9');
xlabel('x');ylabel('y');zlabel('z');

% figure(2)
% semilogy(diag(s),'ko','Linewidth',2)
% set(gca,'Fontsize',16,'Xlim',[0 196])
```

```
%% pick two digits and build LDA

selected_digits = [4 9];
digit1_indices = find(traingnd == selected_digits(1));
digit1_train = trainingdata(:,digit1_indices');

digit2_indices = find(traingnd == selected_digits(2));
digit2_train = trainingdata(:,digit2_indices');

[threshold, u, s, v, w, sort_digit1, sort_digit2] =
digit_trainer(digit1_train, digit2_train);

% figure(3)
% plot(v_digit1, zeros(length(v_digit1)),'ob','Linewidth',2)
% hold on
% plot(v_digit2, ones(length(v_digit2)),'dr','Linewidth',2)
% ylim([0 1.2])

figure(5)
subplot(1,2,1)
histogram(sort_digit1,30); hold on, plot([threshold threshold], [0 1000],'r')
set(gca,'Xlim',[-10 10],'Ylim',[0 1000],'Fontsize',14)
title('the first digit')
subplot(1,2,2)
histogram(sort_digit2,30); hold on, plot([threshold threshold], [0 1000],'r')
set(gca,'Xlim',[-10 10],'Ylim',[0 1000],'Fontsize',14)
title('the second digit')


%% Check accuracy on test set
digit1_test_indices = find(testgnd == selected_digits(1));
digit1_test = testdata(:,digit1_test_indices');
n1_test = size(digit1_test, 2);
digit2_test_indices = find(testgnd == selected_digits(2));
digit2_test = testdata(:,digit2_test_indices');
n2_test = size(digit2_test, 2);

digits_test = [digit1_test digit2_test];
hiddenlabels = zeros(1, size(digits_test, 2));
hiddenlabels(n1_test + 1:n1_test + n2_test) = 1; %pick the samples when
predict
testNum = size(digits_test,2);
testMat = u' * digits_test;
pval = w' * testMat;

% digit2 = 1, digit1 = 0
ResVec = (pval > threshold);
err = abs(ResVec - hiddenlabels);
errNum = sum(err);
sucRate = 1 - errNum/testNum;


%% Pick three digits
```

```
feature = 50;
selected_digits = [1,2,3];
digit1_indices = find(traingnd == selected_digits(1));
digit2_indices = find(traingnd == selected_digits(2));
digit3_indices = find(traingnd == selected_digits(3));

digit1_train = trainingdata(:,digit1_indices');

digit2_train = trainingdata(:,digit2_indices');

digit3_train = trainingdata(:,digit3_indices');

n1 = size(digit1_train, 2);
n2 = size(digit2_train, 2);
n3 = size(digit3_train, 2);

[u3,s3,v3] = svd([digit1_train digit2_train digit3_train], 'econ');
digits = s3 * v3';

digit1 = digits(1:feature,1:n1);
digit2 = digits(1:feature,n1+1:n1+n2);
digit3 = digits(1:feature,n1+n2+1:n1+n2+n3);

%% LDA for three digits
m1 = mean(digit1, 2);
m2 = mean(digit2, 2);
m3 = mean(digit3, 2);
m = [m1 m2 m3];
overall_m = (m1 + m2 + m3) / 3;

Sw = 0; % within class variances
for k = 1:length(digit1)
    Sw = Sw + (digit1(:,k) - m1) * (digit1(:,k) - m1)';
end

for k = 1:length(digit2)
    Sw = Sw + (digit2(:,k) - m2) * (digit2(:,k) - m2)';
end

for k = length(digit3)
    Sw = Sw + (digit3(:,k) - m3) * (digit3(:,k) - m3)';
end

Sb = 0; % between class
for i = 1:3
    Sb = Sb + (m(:, i) - overall_m) * (m(:, i) - overall_m)';
end

[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

v_digit1 = w'*digit1;
```

```
v_digit2 = w'*digit2;
v_digit3 = w'*digit3;

if mean(v_digit1) > mean(v_digit2)
    w = -w;
    v_digit1 = -v_digit1;
    v_digit2 = -v_digit2;
end

if mean(v_digit2) > mean(v_digit3)
    w = -w;
    v_digit2 = -v_digit2;
    v_digit3 = -v_digit3;
end

figure(3)
plot(v_digit1, zeros(length(v_digit1)),'ob','Linewidth',2)
hold on
plot(v_digit2, ones(length(v_digit2)),'dr','Linewidth',2)
hold on
plot(v_digit3, ones(length(v_digit3)) + 1,'g','Linewidth',2)
ylim([0 3])
%% Classification
sort_digit1 = sort(v_digit1);
sort_digit2 = sort(v_digit2);
sort_digit3 = sort(v_digit3);

% Compare between digit 1 and digit 2
t1 = length(sort_digit1);
t2 = 1;
while sort_digit1(t1) > sort_digit2(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sort_digit1(t1) + sort_digit2(t2)) / 2;

figure(4)
subplot(1,2,1)
histogram(sort_digit1,30); hold on, plot([threshold threshold], [0 1000],'r')
set(gca,'Xlim',[-8 4],'Ylim',[0 1000],'Fontsize',14)
title('the first digit')
subplot(1,2,2)
histogram(sort_digit2,30); hold on, plot([threshold threshold], [0 1000],'r')
set(gca,'Xlim',[-8 4],'Ylim',[0 1000],'Fontsize',14)
title('the second digit')

% Compare between digit 2 and digit 3
t2 = length(sort_digit2);
t3 = 1;
while sort_digit2(t2) > sort_digit3(t3)
    t2 = t2 - 1;
    t3 = t3 + 1;
end
threshold = (sort_digit2(t2) + sort_digit3(t3)) / 2;
```

```
figure(5)
subplot(1,2,1)
histogram(sort_digit2,30); hold on, plot([threshold threshold], [0 1000],'r')
set(gca,'Xlim',[-5 7],'Ylim',[0 1000],'Fontsize',14)
title('the second digit')
subplot(1,2,2)
histogram(sort_digit3,30); hold on, plot([threshold threshold], [0 1000],'r')
set(gca,'Xlim',[-5 7],'Ylim',[0 1000],'Fontsize',14)
title('the third digit')

% Compare between digit 1 and digit 3
t1 = length(sort_digit1);
t3 = 1;
while sort_digit1(t1) > sort_digit3(t3)
    t1 = t1 - 1;
    t3 = t3 + 1;
end
threshold = (sort_digit1(t1) + sort_digit3(t3)) / 2;

figure(6)
subplot(1,2,1)
histogram(sort_digit1,30); hold on, plot([threshold threshold], [0 1000],'r')
set(gca,'Xlim',[-6 7],'Ylim',[0 1000],'Fontsize',14)
title('the first digit')
subplot(1,2,2)
histogram(sort_digit3,30); hold on, plot([threshold threshold], [0 1000],'r')
set(gca,'Xlim',[-6 7],'Ylim',[0 1000],'Fontsize',14)
title('the second digit');


%% classification tree on fisheriris data
tree = fitctree(trainingdata',traingnd,'MaxNumSplits',10,'CrossVal','on');
view(tree.Trained{1},'Mode','graph')
classError = kfoldLoss(tree,'mode','individual');
[~,A] = min(classError);
%%
test_prediction = predict(tree.Trained{A},testdata');
i = 0;
for j = 1: length(testgnd)
    if testgnd(j) == test_prediction(j)
        i = i +1;
    end
end
accuracy = i/length(testgnd)

%% two digits of decision tree
digits_easy = [0 1]
d1_indices = find(traingnd ==digits_easy(1) );
d2_indices = find(traingnd ==digits_easy(2) );
d1_train = trainingdata(:,d1_indices');
d2_train = trainingdata(:,d2_indices');
d1_train_label = traingnd(d1_indices);
d2_train_label = traingnd(d2_indices);
easy = [d1_train d2_train];

easy_test_ind1 = find(testgnd == digits_easy(1));
```

```
easy_test_ind2 = find(testgnd == digits_easy(2));
d1_test = testdata(:,easy_test_ind1');
d2_test = testdata(:,easy_test_ind2');
test = [d1_test d2_test];

tree2 = fitctree(easy', [d1_train_label;
d2_train_label],'MaxNumSplits',10,'CrossVal','on');
classError2 = kfoldLoss(tree2,'mode','individual');
[~,B] = min(classError2);

test_prediction = predict(tree2.Trained{B},test');
easy_label = [testgnd(easy_test_ind1);testgnd(easy_test_ind2)];
err = test_prediction - easy_label;
correctnum = find(err == 0);

accuracy = length(correctnum)/length(test_prediction);
%% svm
% SVM classifier with training data, labels and test set
Mdl = fitcecoc(trainingdata',traingnd);
test_labels = predict(Mdl,testdata');

%%
m = 0;
for k = 1: length(testgnd)
    if testgnd(k) == test_labels(k)
        m = m +1;
    end
end
accuracy_svm = m/length(testgnd)
%%
Mdl = fitcecoc(easy', [d1_train_label; d2_train_label]);
test_labels = predict(Mdl,test');
%%
err = test_labels - easy_label;
correctnum = find(err == 0);

accuracy_svm2 = length(correctnum)/length(test_prediction);


%% three digits
feature = 60;
% Number of observations of each class
n_3 = 5923;%0
n_4 = 6265;%7
n_5 = 5421;
N = n_3 + n_4 + n_5;
%Mean of each class

P = S*V'; % projection onto principal components: X = USV' --> U'X = SV'
threes = P(1:feature,traingnd== 0);
fours = P(1:feature,traingnd == 7);
fives = P(1:feature, traingnd == 5);

m_3 = mean(threes,2);
m_4 = mean(fours,2);
```

```
m_5 = mean(fives,2);

%%
Sw = 0; % within class variances
for k = 1:n_3
    Sw = Sw + (ones(:,k) - m_3)*(ones(:,k) - m_3)';
end
for k = 1:n_4
   Sw =  Sw + (sevens(:,k) - m_4)*(sevens(:,k) - m_4)';
end
for k = 1:n_5
   Sw =  Sw + (sevens(:,k) - m_5)*(sevens(:,k) - m_5)';
end
%
me = [m_3,m_4,m_5];
overall_m = (m_3+m_4+m_5)/3;
Sb = 0;
for j = 1:3
    Sb = Sb + (me(:,j)-overall_m)*(me(:,j)-overall_m)';
end
%% Find the best projection line

[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

%% Project onto w

v_3 = w'*threes;
v_4 = w'*fours;
v_5 = w'*fives;

%% Make dogs below the threshold

% if mean(v_1) > mean(v_7)
%     w = -w;
%     v_1 = -v_1;
%     v_7 = -v_7;
% end

%% Plot dog/cat projections (not for function)

figure(4)
plot(v_3,threes(6131),'ob','Linewidth',2)
hold on
plot(v_4,fours(5842),'or','Linewidth',2)
hold on
plot(v_5,fives(5421),'og','Linewidth',2)
%% Find the threshold value

sort_threes = sort(v_3);
sort_fours = sort(v_4);
sort_fives = sort(v_5);
```

```
t3 = length(sort_threes);
t4 = 1;
while sort_threes(t3) > sort_fours(t4)
    t3 = t3 - 1;
    t4 = t4 + 1;
end
threshold = (sort_threes(t3) + sort_fours(t4))/2;


%%
figure(5)
subplot(1,2,1)
histogram(sort_threes,40); hold on, plot([threshold threshold], [0 500],'r')
set(gca,'Fontsize',14)
title('digit3')
subplot(1,2,2)
histogram(sort_fours,40); hold on, plot([threshold threshold], [0 500],'r')
set(gca,'Fontsize',14)
title('digit4')

%%
t3 = length(sort_threes);
t5 = 1;
while sort_threes(t4) > sort_fives(t5)
    t3 = t3 - 1;
    t5 = t5 + 1;
end
threshold = (sort_threes(t3) + sort_fives(t5))/2;


figure(5)
subplot(1,2,1)
histogram(sort_threes,40); hold on, plot([threshold threshold], [0 500],'r')
set(gca,'Fontsize',14)
title('digit:3')
subplot(1,2,2)
histogram(sort_fives,40); hold on, plot([threshold threshold], [0 500],'r')
set(gca,'Fontsize',14)
title('digit:5')









%% function
```

```
function [threshold, u, s, v, w, sort_digit1, sort_digit2] =
digit_trainer(digit1_train, digit2_train)
    feature = 60;
    data = [digit1_train digit2_train];
    [u,s,v] = svd(data, 'econ');
    u = u(:,1:feature);
    n1 = size(digit1_train, 2);
    n2 = size(digit2_train, 2);
    digits = s * v';

    digit1 = digits(1:feature,1:n1);
    digit2 = digits(1:feature,n1+1:n1+n2);

    m1 = mean(digit1,2);
    m2 = mean(digit2,2);

    Sw = 0; % within class variances
    for k = 1:n1
        Sw = Sw + (digit1(:,k) - m1) * (digit1(:,k) - m1)';
    end

    for k = 1:n2
        Sw =  Sw + (digit2(:,k) - m2) * (digit2(:,k) - m2)';
    end

    Sb = (m1-m2)*(m2-m1)'; % between class

    [V2, D] = eig(Sb,Sw); % linear disciminant analysis
    [~, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);

    v_digit1 = w'*digit1;
    v_digit2 = w'*digit2;

    if mean(v_digit1) > mean(v_digit2)
        w = -w;
        v_digit1 = -v_digit1;
        v_digit2 = -v_digit2;
    end

    sort_digit1 = sort(v_digit1);
    sort_digit2 = sort(v_digit2);
    t1 = length(sort_digit1);
    t2 = 1;

    while sort_digit1(t1) > sort_digit2(t2)
        t1 = t1 - 1;
        t2 = t2 + 1;
    end
    threshold = (sort_digit1(t1) + sort_digit2(t2)) / 2;
end
```