Hanze Zhang
Dr. Bramburger
AMATH 482
Assignment 4
03/10/2021

Abstract:
Given a dataset containing images of handwritten digits, is it possible for us to classify each image successfully? In this assignment, we are given a dataset with 60000 images of handwritten digits, and we are trying to classify each image with Linear Discriminant Analysis (LDA), decision trees and support vector machines (SVM). We will compare the accuracy between these three different classification methods.

Introduction and Overview:
In this assignment, we are using the MNIST dataset, which contains 60000 images of handwritten digits for training and 10000 images for testing. There are 10 handwritten digits in total, from 0 to 9.

First, we need to perform an SVD analysis on the dataset in order to see how many features are appropriate for our classifier. Then, we can use the number of features we find to build up the LDA classifier to classify images of selected handwritten digits. After we train a model for classification, we can use the model to predict images in the test set and compare results with those true test labels. In this way, we can obtain a success rate, indicating the performance of our model.

Then, we can train a decision tree model as well as a SVM model and use them to predict on the tests set. We obtain both success rates so that we can compare their performance in classifying images of handwritten digits with that of LDA.

Theoretical Background:
In this assignment, two most important techniques are singular value decomposition (SVD) and LDA. SVD can help us find the appropriate number of modes that we are going to project our data onto, while LDA can help us find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data.

Given a matrix $A \in \mathbb{R}^{m*n}$, if we perform SVD on A:

$$A = U\Sigma V^* \quad (1)$$

In the above equation, $U \in \mathbb{R}^{m*m}$ and $V^* \in \mathbb{R}^{n*n}$ are unitary matrices and $\Sigma \in \mathbb{R}^{m*n}$ is a diagonal matrix with non-negative σ_j ordered from largest to smallest on its diagonals. Then, we can project our data X onto principal components by:

$$X = U\Sigma V^* \quad (2)$$

$$U'X = \Sigma V^* \quad (3)$$

In order to figure out the right subspace to project onto with LDA, we first need to calculate means for each of our groups for each feature. With two groups, we use the following equations:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \quad (4)$$

In the above equation, μ_1 is the column vector containing the means of each row in the first group, and μ_2 is the same vector but representing the second group. S_B is then called between-class scatter matrix, as it measures the variance between the groups. Then, we need to calculate a within-class scatter matrix with the following equation:

$$S_w = \sum_{j=1}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T \quad (5)$$

This measures the variance within each group.

With these two matrices, we can find a vector w such that:

$$w = argmax \frac{w^T S_B w}{w^T S_W w} \quad (6)$$

Since we know that the vector w is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem, we can use the equation below to help us find the vector w:

$$S_B w = \lambda S_w w \quad (7)$$

We can also generalize LDA to more than 2 groups. The only place we need to change is the way to calculate S_B and S_w. With groups more than two, the equation to calculate S_B becomes:

$$S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T \quad (8)$$

In the above equation, μ is the overall mean and μ_j is the vector of means we mentioned above for each group. Similarly, S_w now is equal to:

$$S_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T \quad (9)$$

We do not need to change the way we find w.

Algorithm Implementation and Development:

The first step we took was to load both the MNIST training and testing dataset to MATLAB with the function $mnist\_parse$. Then, we reshaped our training and testing data such that each column in the matrix represents an image of handwritten digit and each row is a feature. While reshaping, we also called the function $im2double$ to transform the image data from type uint8 to double, so we can better analysis them with MATLAB. After reshaping, the training matrix is $784 * 60000$ and the testing matrix is $784 * 10000$. To center our data, we subtract both means of each row of training data from both the training data and the testing data.

Next, we start performing the SVD analysis on the whole training dataset. After

performing SVD on the training data and obtained three matrices U, Σ, V, we can now plot the singular value spectrum and the captured energy diagram to help determine the appropriate number of modes (Figure 1 and Figure 2). Then, we can find projections by multiply U with the training data. Now, we can plot our projections onto three selected V-modes, columns 2, 3 and 5 (Figure 3).
According to singular value and energy diagram, I determine the appropriate number of features should be 50. We can now pick two digits and start LDA. We use the function $find$ to find indices of the selected digit in both the training and testing label. Then we extract corresponding images from both the training and testing data. We combine the data of two digits together and form our data matrix. Then we perform SVD again and obtain their projections. After we find their projections, we can then start calculating our between-class scatter matrix S_B and within-class scatter matrix S_w. In order to find these two, we need means from both groups' data. Now, we can use the function $eig()$ along with $\max()$ to help us find w. After we find the vector w, we can now project our data onto the LDA subspace.

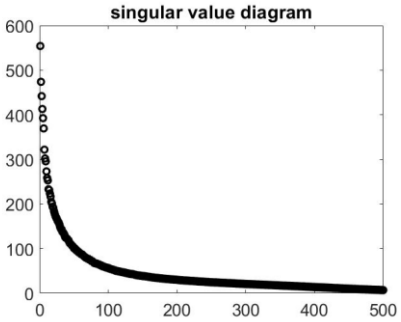
Next step is to find a threshold to classify the training data. To make this step easier, we make a small change in our projected data to make sure the digit 1 is always lower than digit 2. We sort each projected digits data. If the digit 1 value is less than the digit 2 value, we move in one on each. We continue until the digit 2 value is less than the digit 1 value, so we can now set our threshold as their midpoint.

After we find our threshold, we start using the threshold to classify each image in the testing data. We still need to perform PCA and LDA projection on our testing data. Then we can check if the current value is larger than the threshold or not. If it is, we consider it as digit 1. Then, we can find our success rate by dividing the number of correct results by the total number of images.

If we are classifying 3 digits, we will follow the same procedure above. The only difference is that we use different way to obtain our S_B and S_w, and we need to compare three times.

Finally, we can start training our decision tree and SVM models. We can train our decision tree model by using $fitctree$. We also set the cross validation on. With cross validation, we have to find the tree with lease validation error. Then, we use that tree to classify images in the test data and calculate the success rate. Similarly, we use $fitcecoc$ to train our SVM model. We can use the model to classify images in the testing set directly.

Computational Results:
Three matrices obtained by SVD is named U, S, V. U represents the principal components (left singular value), while S stores the singular value. Singular values stored in S tell us how much weight each principal component carries. V stores the right singular value. The singular value diagram and energy diagram are attached below:



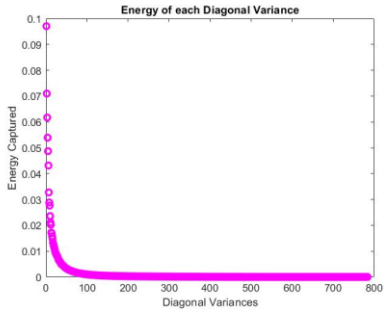


Figure 1 - Singular value diagram of training data **Figure 2 - energy diagram of training data**

From above diagrams, we determine to use 50 as our feature number. Projection onto three selected V-modes (columns 2, 3, 5) are attached below:

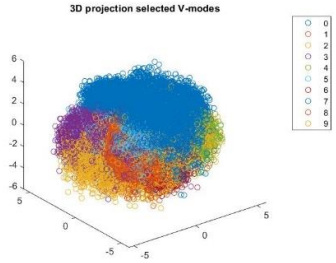


Figure 3 - 3D projection of selected V-modes (Columns 2, 3, 5)

Two digits I picked are 0 and 1. After running LDA, we have the following histogram:

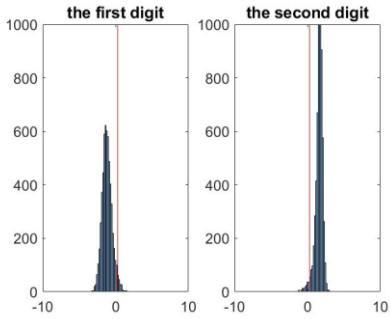


Figure 4 - Histogram of result of LDA on digits 0 (left) and 1 (right)

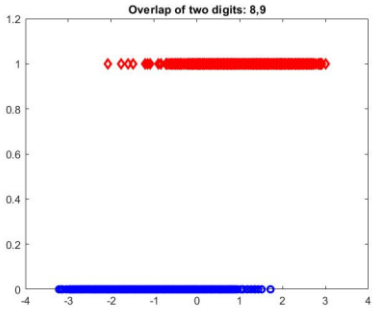


Figure5 - overlap of two digits: 8, 9

From Figure 4 and Figure 5, we can observe that there are some overlaps, and LDA classify some images wrong. The success rate of classifying 8 and 9 is 0.9543, which is not too bad. Then, we need to determine which two digits are easiest to be classified and which two digits are hardest to be classified. After trying all 45 combinations, 0 and 1 are easiest to be classified by LDA with a success rate 0.9962, while 4 and 9 are hardest to be classified by LDA with a success rate 0.9543. The above results are

obtained from the testing set. If we are testing our LDA model on the training set, the easiest pair of digits is 6 and 7 with a success rate of 0.9980. while the hardest pair is 7 and 9 with a success rate of 0.9481.

To see how LDA perform in classifying three digits, I pick digits 1, 6, 3. Three histograms are attached below:

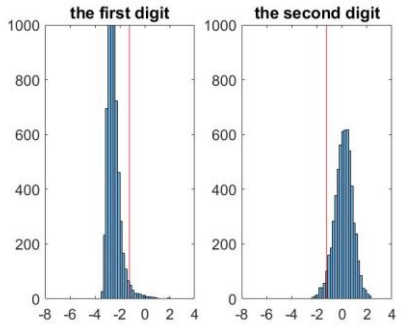


Figure 6 - Histogram of classifying three digits (Compare b/w 1 (left) and 2 (right))

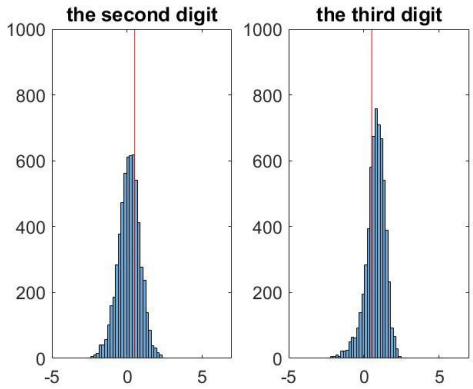


Figure 7 - Histogram of classifying three digits (Compare b/w 2 (left) and 3 (right))

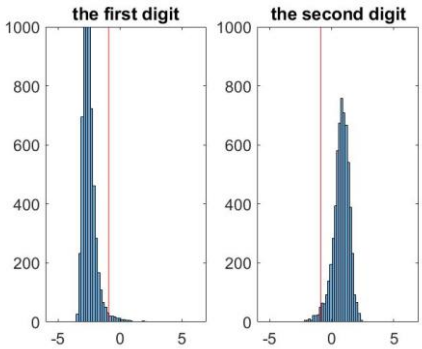


Figure 8 - Histogram of classifying three digits (Compare b/w 1 (left) and 3 (right))

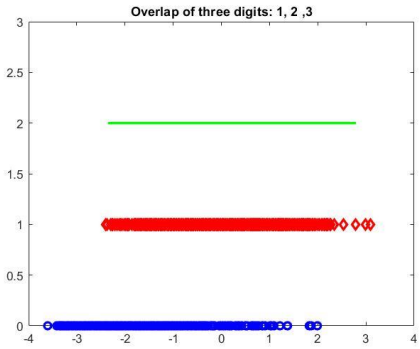


Figure 9 - overlap of three digits (1 (bottom), 2 (middle), 3 (top))

From above histograms (Figure 6, 7, 8) and overlap diagrams (Figure 9), we can observe that there are a lot of overlaps between three digits, indicating that If we are trying to classify the whole training set with decision tree and SVM, not picking out digits, the decision tree will give us a success rate of 0.5779 and SVM will give us a success rate of 0.9438. The success rate of SVM is much higher than that of decision tree.

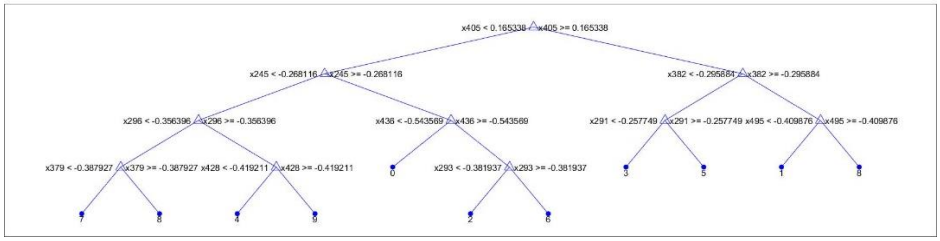


Figure 10 – The trained Decision tree with lowest validation error

We can compare the performance of LDA, decision tree and SVM using hardest and easiest pair of digits to separate in the testing set. On the easiest pair of digits, decision tree has a success rate of 0.9962, while SVM has a success rate of 0.9991. On the hardest pair of digits in the testing set, decision tree has a success rate of 0.9196, while SVM has a success rate of 0.9709.

As for the training set, decision tree has a success rate of 0.99 in distinguishing the easiest pair of digits (6 and 7), while SVM has a success rate of 1. If we are using these two methods to classify the hardest pair, the decision tree gives us a success rate of 0.9546 and SVM gives us a success rate of 0.9745.

Summary and Conclusions:

From comparisons above, we can see that in the testing set, on the easiest pair of digits, decision tree and LDA have the same success rate, while SVM has a higher success rate then those two. On the hardest pair of digits in the testing set, LDA performs better than decision tree, while SVM still performs the best. In the training set, on the easiest pair of digits, SVM has the highest success rate and LDA has a higher success rate than decision tree. On the hardest pair, SVM still has the highest success rate and decision tree has a higher success rate than LDA. From above results, we can see that SVM performs the best in every condition, while decision tree and LDA do not have too much

difference.

Appendix 1 (MATLAB Functions):

- $minst\_parse$: transform MNIST data to a form that we can analysis them in MATLAB
- $im2double(I)$: converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a binary image. $im2double$ rescales the output from integer data types to the range $[0, 1]$.
- $find(X)$: returns a vector containing the linear indices of each nonzero element in array X.
- $min(A)$: returns the minimum elements of an array
- $[U, S, V] = svd(A)$: performs a singular value decomposition of matrix A, such that $A = U * S * V'$.
- $reshape$: Reshape array.
- $fitctree$: Fit binary decision tree for multiclass classification
- $predict$: Predict labels using classification model.
- $fitcecoc$: Fit multiclass models for support vector machines or other classifiers
- $fitcsvm$: Train support vector machine (SVM) classifier for one-class and binary classification

Appendix 2 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);

[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);

% subtract mean
mn =  mean(trainingdata,2);
trainingdata = trainingdata - repmat(mn,1,60000);

testdata = testdata - repmat(mn,1,10000);
%%
[u,s,v] = svd(trainingdata, 'econ');
lambda = diag(s).^2;

figure(1)
plot(diag(s),'ko','Linewidth',2)
title("singular value diagram")
set(gca,'Fontsize',16,'Xlim',[0 500])
figure(2)
plot(1:784, lambda/sum(lambda), 'mo', 'Linewidth', 2);
title("Energy of each Diagonal Variance");
xlabel("Diagonal Variances"); ylabel("Energy Captured");

%%
proj = u(:,[2,3,5])'* 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(3)
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 projection selected V-modes')
legend('0','1','2','3','4','5','6','7','8','9');
%% pick two digits and build LDA
% selected_digits = randperm(10, 2) - 1;
highest_digit1 = 0;
highest_digit2 = 0;
highest_succRate = 0;

lowest_digit1 = 0;
lowest_digit2 = 0;
lowest_succRate = 2;

% try all combinations
for i = 0:8
    for j = i + 1:9
        selected_digits = [i j];
        digit1_indices = find(traingnd == selected_digits(1));
        digit1_train = trainingdata(:,digit1_indices');

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

        [v_digit1, v_digit2, threshold, u, s, v, w, sort_digit1,
sort_digit2] = digit_trainer(digit1_train, digit2_train);

        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];
        digits_train = [digit1_train digit2_train];
        n1_train = size(digit1_train, 2);
        n2_train = size(digit2_train, 2);
        hiddenlabels = zeros(1, size(digits_train, 2));
        hiddenlabels(n1_train + 1:n1_train + n2_train) = 1;
```

```
        testNum = size(digits_train,2);
        testMat = u' * digits_train;
        pval = w' * testMat;

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

        if sucRate > highest_succRate
            highest_succRate = sucRate;
            highest_digit1 = i;
            highest_digit2 = j;
        end

        if sucRate < lowest_succRate
            lowest_succRate = sucRate;
            lowest_digit1 = i;
            lowest_digit2 = j;
        end
    end
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)
ylim([0 1.2])
title("Overlap of two digits: 8,9")

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 for two digits 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;
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 = randperm(10, 3) - 1;
selected_digits = [1 2 3];
digit1_indices = find(traingnd == selected_digits(1));
digit1_train = trainingdata(:,digit1_indices');

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

digit3_indices = find(traingnd == selected_digits(3));
digit3_train = trainingdata(:,digit3_indices');

data = [digit1_train digit2_train digit3_train];
[u,s,v] = svd(data, 'econ');

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

digits = s * v';

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:n1
    Sw = Sw + (digit1(:,k) - m1) * (digit1(:,k) - m1)';
end

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

for k = 1:n3
    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])
title("Overlap of three digits: 1, 2 ,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')

%% SVM and decision tree
tree = fitctree(trainingdata',traingnd,
'MaxNumSplits',10,'CrossVal','on');
view(tree.Trained{1},'Mode','graph');
classError = kfoldLoss(tree, 'mode', 'individual');
[~, k] = min(classError);
testSetPredictions = predict(tree.Trained{k}, testdata');
err_tree = immse(testSetPredictions, testgnd);
```

```
diff = testSetPredictions - testgnd;
correct_pred = find(diff == 0);
sucRate_tree = length(correct_pred) / length(testgnd);

% SVM classifier with training data, labels and test set
Mdl = fitcecoc(trainingdata',traingnd);
testlabels = predict(Mdl,testdata');

diff_svm = testlabels - testgnd;
correct_pred = find(diff_svm == 0);
sucRate_svm = length(correct_pred) / length(testgnd);

%% Decision tree and svm on pairs of digits
easiest_digits = [7 9];

% Decision Tree
digit1_indices = find(traingnd == easiest_digits(1));
digit1_train = trainingdata(:,digit1_indices');
digit1_train_label = traingnd(digit1_indices);

digit2_indices = find(traingnd == easiest_digits(2));
digit2_train = trainingdata(:,digit2_indices');
digit2_train_label = traingnd(digit2_indices);

easiest_data = [digit1_train digit2_train];

easiest_test_digit1_ind = find(testgnd == easiest_digits(1));
digit1_test = testdata(:,easiest_test_digit1_ind');

easiest_test_digit2_ind = find(testgnd == easiest_digits(2));
digit2_test = testdata(:,easiest_test_digit2_ind');

digits_test = [digit1_test digit2_test];

tree = fitctree(easiest_data',[digit1_train_label;
digit2_train_label], 'MaxNumSplits',10,'CrossVal','on');
view(tree.Trained{1},'Mode','graph');
classError = kfoldLoss(tree, 'mode', 'individual');
[~, k] = min(classError);
testSetPredictions = predict(tree.Trained{k}, easiest_data');

train_label = [digit1_train_label; digit2_train_label];
easiest_test_label = [testgnd(easiest_test_digit1_ind);
```

```
testgnd(easiest_test_digit2_ind)];

diff = testSetPredictions - train_label;
correct_pred = find(diff == 0);
sucRate_tree = length(correct_pred) /
length(testSetPredictions);

% SVM
Mdl = fitcsvm(easiest_data',[digit1_train_label;
digit2_train_label]);
testlabels = predict(Mdl,easiest_data');

diff_svm = testlabels - train_label;
correct_pred = find(diff_svm == 0);
sucRate_svm = length(correct_pred) / length(train_label);

function [v_digit1, v_digit2, threshold, u, s, v, w, sort_digit1,
sort_digit2] = digit_trainer(digit1_train, digit2_train)
    feature = 50;
    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
```