AMATH 482 Assignment 4

Carter Peyton

March 10, 2021

Abstract

This assignment aims to build an image classifier based on the MNIST handwritten digit set. Through the SVD, we are able to understand the key components of each digit the commonalities/differences between the digits. We then use linear discriminant analysis, support vector machines, and decision tree classifiers to identify these digits. The results illustrate a computer's ability to identify hand-drawn digits based on their core components.

1 Introduction and Overview

Using the MNIST data set (hand-drawn images of digits), this assignment aims to use the SVD, LDA, SVM, and decision tree classifiers to explore and identify the properties of hand-drawn digits. The SVD is used to identify principal components of the digits to figure out what the important "essences" of each digit are. LDA is used to create a linear classifier based on the results of SVD to classify the images of digits. Lastly, SVM and decision tree classifiers will be used to compare between LDA.

2 Theoretical Background

2.1 The Singular Value Decomposition

The singular value decomposition (SVD) is, as the name suggests, a method to decompose a matrix A into unitary matrices U, V^* and diagonal matrix Σ, so that $A = U\Sigma V^*$. The diagonal elements of Σ, which are non negative and in descending order, are called the singular values (σ_j). U and V^* are rotation matrices that apply a change of basis to the original data. The columns of these matrices correspond to the respective singular values, which in turn give an idea of the information capture of these columns in their respective bases. Note that the mean of each row in A must be subtracted from the row.

2.1.1 SVD Projection in Image Analysis

The U matrix of SVD in image analysis corresponds to the essences of the images. Therefore we want to project our original data onto these simpler "essences." We accomplish this by noticing that $UX = SV'$. Moreover, we can control the amount of features (corresponding to significant singular values) we want by only projecting over a subset of columns.

2.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a method that takes some groupings of data and projects them onto a line to maximize separation between the data, while minimizing the spread in each of the groups. To accomplish this we construct a between class variance matrix where

$$S_b = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{1}$$

Note: for more than two classes we need to consider the overall mean, μ and S_b becomes

$$S_b = \sum_{j=1}^{n} (\mu_j - \mu)(\mu_j - \mu)^T \tag{2}$$

We also must construct a within class variance matrix

$$S_w = \sum_{j=1}^{n} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{3}$$

where each class (j) is a cluster of data. We construct our projection line, w, by solving

$$S_b w = \lambda S_w w \tag{4}$$

Lastly we project each "cluster" of data onto the line w. This results in a way to linearly classify the data based on the position/groupings along the line.

3 Algorithm Implementation and Development

3.1 Initialization

The images from the testing and training sets from the MNIST database are loaded in and processed into 28^2 x 1 vectors, which are then placed into a data matrix.

3.2 SVD

3.2.1 SVD Spectrum Plotting

The SVD of the data matrix is taken with the 'econ' parameter. Then the energies of each singular value are computed and plotted alongside the singular values. This creates a singular value spectrum. We then use this spectrum to determine how many features (significant singular values) are necessary to capture 90% of the information in the data matrix.

3.2.2 SVD Mode Projection

Using the labels vector, we find the indices of each of the digits. For example there is a vector that contains all the indices of hand-drawn sevens. For each of the digits, they are projected onto three different modes of V. From there we plot the three projections for each of the digits using the plot3 command. The resulting figure shows clusters, where each color corresponds to images of a certain digit.

3.3 LDA

In order to employ LDA, a new data matrix is created where new data = $S * V'$. We then construct vectors of r features (where r is the amount of singular values that captures at least 90% of the information in the system) for each of the digits that we want to classify. Within class (Sw) and between class (Sb) matrices are created. The eigenvalues of these matrices are computed and are used to project our data (whatever digits we want to classify) onto a line, w. We then plot the resulting projections to see the classification.

3.4 SVM and Decision Trees

For each method, SVM and decision trees, a model is trained based on the training data and then tested on the test images. We do not feed in the entire data matrix, instead we use a reduced data matrix $S * V'$ with r number of features that we determined via the SVD spectrum. This data is scaled by the maximum value of itself in order to be in the range of [-1 1], this reduces runtime.

4 Computational Results

4.1 SVD Analysis

4.1.1 Plotting the SVD Spectrum

In order to reach a good energy capture, approximately 64 singular values are used (this captures 91.7% of the original energy). We use the number 64 because it is a power of 2, which may come in hand.

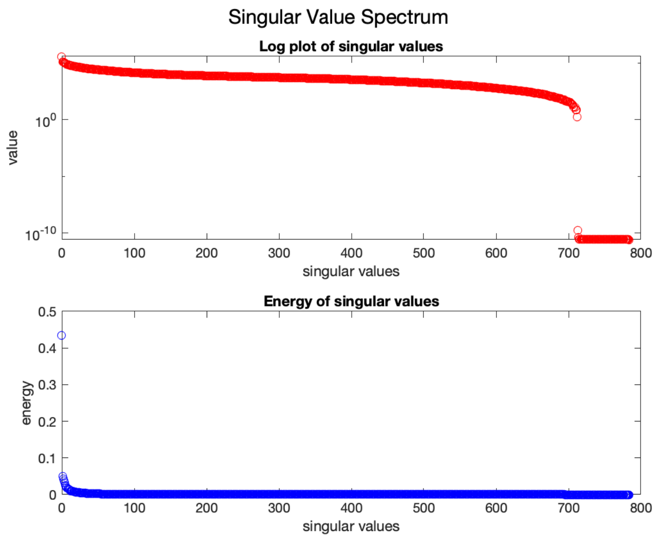

Figure 1: This figure displays the numerical value and the energy capture of each singular value.

The first mode contains much of the information, however, the second mode and on have a relatively flat drop off until around the 700th mode.

4.1.2 Interpretation of the U, Σ, and V matrices

The U matrix contains the base components of the images. Each column of U corresponds to different ”essences” of the images. For example, a column in U may represent a single vertical line. Σ serves as a scaling matrix for these modes. Lastly, V describes how much each image is present in each of the modes of U. For example, the first column of V tells how strongly each image is present in the first mode of U. We let the first mode of U be a vertical line (as an example), the first column of V would likely have large values at indices that correspond to digits with strong, vertical components. 1 and 7 may be strongly represented in this column of V.

4.1.3 Projection onto V-modes

For each digit (0 - 9), we project the respective values from the original data onto three selected modes of V. The modes 2, 3, and 5 produce the following figure.

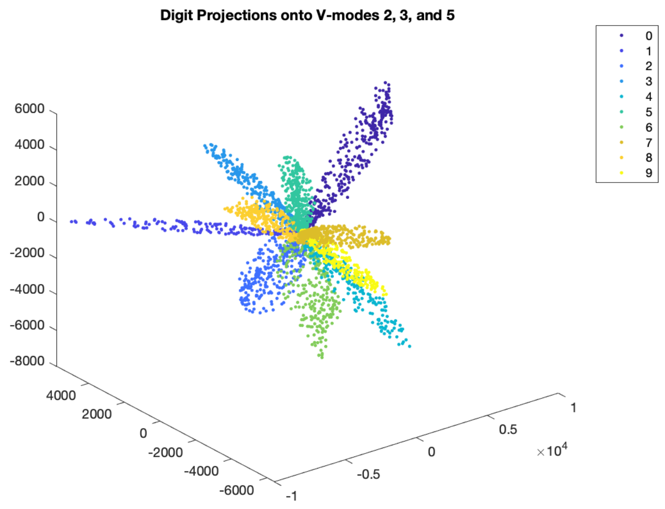

Figure 2: Each cluster corresponds to a digit, clusters farther from each other can be thought of as having different core components, while those that are close have similar core components.

We can see that certain clusters (like those of 4 and 9) are close together. The proximity to one another suggests that 4 and 9 share some core aspect. This makes sense considering the shape of 4 and 9, both digits have enclosed/circular tops, with vertical lines on the right side. Other digits (take 0 and 1) are much further apart, suggesting that there is little commonality between them. This is also consistent with the shapes of 0 and 1, as 0 is primarily a circle and 1 is a vertical line.

4.2 Linear Discriminant Analysis

4.2.1 2 Digit LDA

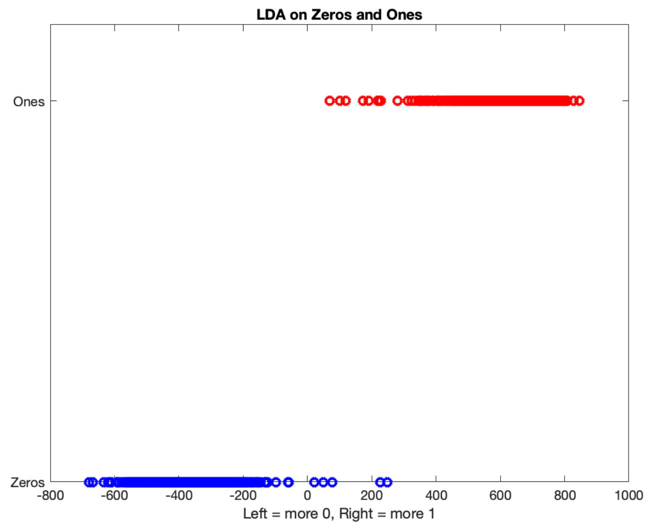

Figure 3: Results of performing LDA on the hand-drawn 0s and 1s.

Given the digits 0 and 1 and using LDA, we are very successful at classifying the two with accuracy rates of 99.8% and 99.9% for 0s and 1s respectively.

4.2.2 Three digit LDA

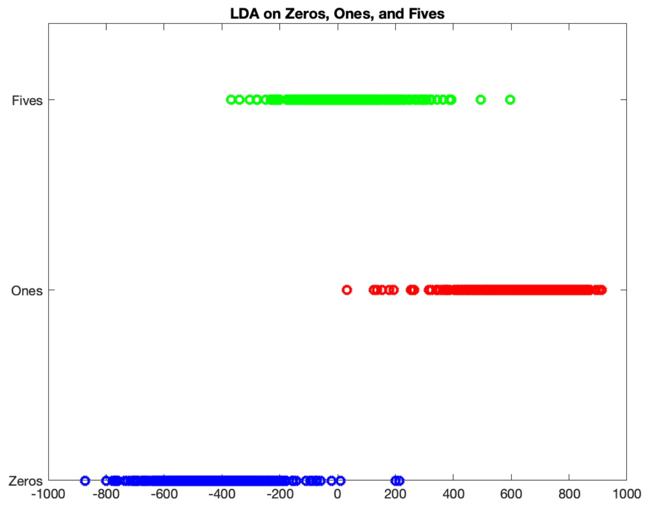

Figure 4: Results of performing LDA on the hand-drawn 0s, 1s, and 5s.

Using LDA on three different digits results we see that the three digits have their own distinct means. However, the variances of each group creates a lot of overlap. The figure suggests that there are elements of a hand-drawn 5 that are in both 0s and 1s. However,

4.2.3 Hardest two digit pair to separate

In order to find the hardest two digit pair to separate, we compute the distances between clusters seen in figure 2. The minimum distance represents the two digits that are hardest to distinguish. This appears to be 4 and 9. Using LDA, we obtain an accuracy of 96.1% and 96.2% for 4s and 9s respectively.

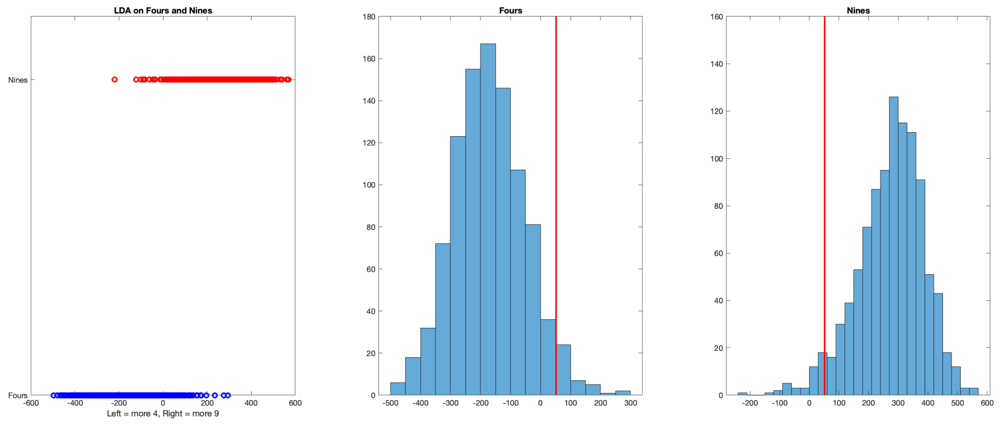

Figure 5: LDA results on 4s and 9s.

4.2.4 Easiest two digit pair to separate

Conversely, the easiest digits to distinguish corresponds to the clusters with the most distance between them. In this case, it appears that 0 and 1 have the most separation between the center of their clusters. In order to quantify this we reference the results of LDA on these digits in Figure 3. To further the distinct separation between the 0s and 1s, we construct a histogram with a threshold value between the two.

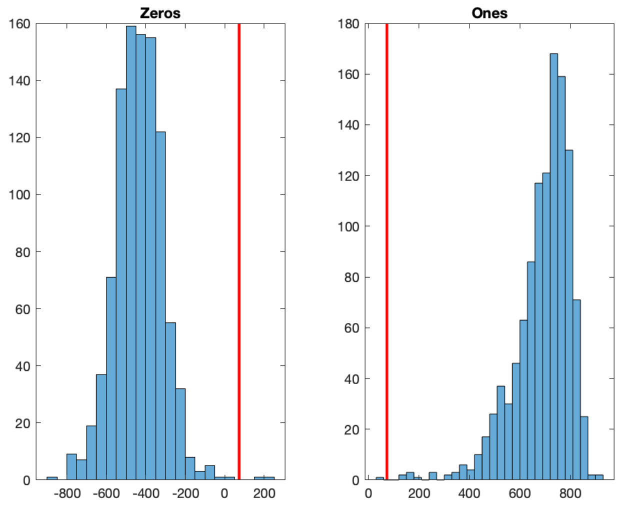

Figure 6: Histogram results of LDA on 0s and 1s.

4.3 SVM, Decision Trees, and Comparison

When comparing all digits, decision trees seems to have the upper hand on SVM as SVM is largely limited to binary classification. In order to compare data in a meaningful way, we consider the accuracy between 0s and 1s and the accuracy between 4s and 9s.

Method	0s and 1s	4s and 9s
LDA	99.8% and 99.9%	96.1% and 96.2%
SVM	100% and 89.3%	41.9% and 66.8 %
Decision Tree	98.1% and 99.0%	93.5% and 93.8%

We see that LDA and decision trees have a relatively similar accuracy. However, SVM seems to fall behind these two in terms of accuracy (we consider 100% accuracy for 0s as an outlier).

5 Summary and Conclusion

Through the use of SVD, LDA, SVM, and decision trees, we were able to classify images of hand-drawn digits. The SVD allowed us to look at certain ”essences” of these digits and project our data onto them. LDA, SVM, and decision trees served as algorithms to classify these projections. LDA and decision trees have similar runtimes and accuracies, while SVM seems to fall behind these methods in accuracy (more evident for similar digits).

Appendix A MATLAB Functions

Appendix B MATLAB Code

```
%% Part 0 - Init and process
clear all; clc
% can load test data by changing train  to test
[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
num_img = size(images,3);
data = zeros(28*28,num_img);
for i = 1:num_img
    data(:,i) = reshape(images(:,:,i),[28*28 1]);
end

%% Part 1 - Perform SVD
[U,S,V] = svd(data,'econ');

%% Part 1 - Compute energy and 90% r value (choose 64 for simplicity)
sig = diag(S);
for l = 1:length(sig)
    energy(l) = sig(l)^2/sum(sig.^2);
end

r = 64;
sum(energy(1:r))

%% Part 1 - plot singular value spectrum
sig_vect = linspace(0,length(sig),length(sig));
subplot(2,1,1)
semilogy(sig_vect,sig,'ro')
xlabel('singular values'); ylabel('value'); title('Log plot of singular values')
subplot(2,1,2)
plot(sig_vect,energy,'bo')
xlabel('singular values'); ylabel('energy'); title('Energy of singular values')
sgtitle('Singular Value Spectrum')
print('-dpng','sing_spect.png')

%% Part 1 - project onto 3 V-modes
% create indices vectors for each digit
zero = find(labels==0);
one = find(labels==1);
two = find(labels==2);
three = find(labels==3);
four = find(labels==4);
five = find(labels==5);
six = find(labels==6);
seven = find(labels==7);
eight = find(labels==8);
nine = find(labels==9);

% modes
m1 = 2;
m2 = 3;
m3 = 5;
```

```
% Projections and plot
cmap = colormap(parula(10));

project0a = V(zero,m1)'*data(:,zero)'; project0b = V(zero,m2)'*data(:,zero)'; project0c = V(zero,m3)'*da
plot3(project0a,project0b,project0c,'.','Color',cmap(1,:))
hold on

project1a = V(one,m1)'*data(:,one)'; project1b = V(one,m2)'*data(:,one)'; project1c = V(one,m3)'*data(:
plot3(project1a,project1b,project1c,'.','Color',cmap(2,:))

project2a = V(two,m1)'*data(:,two)'; project2b = V(two,m2)'*data(:,two)'; project2c = V(two,m3)'*data(:
plot3(project2a,project2b,project2c,'.','Color',cmap(3,:))

project3a = V(three,m1)'*data(:,three)'; project3b = V(three,m2)'*data(:,three)'; project3c = V(three,m
plot3(project3a,project3b,project3c,'.','Color',cmap(4,:))

project4a = V(four,m1)'*data(:,four)'; project4b = V(four,m2)'*data(:,four)'; project4c = V(four,m3)'*da
plot3(project4a,project4b,project4c,'.','Color',cmap(5,:))

project5a = V(five,m1)'*data(:,five)'; project5b = V(five,m2)'*data(:,five)'; project5c = V(five,m3)'*da
plot3(project5a,project5b,project5c,'.','Color',cmap(6,:))

project6a = V(six,m1)'*data(:,six)'; project6b = V(six,m2)'*data(:,six)'; project6c = V(six,m3)'*data(:
plot3(project6a,project6b,project6c,'.','Color',cmap(7,:))

project7a = V(seven,m1)'*data(:,seven)'; project7b = V(seven,m2)'*data(:,seven)'; project7c = V(seven,m
plot3(project7a,project7b,project7c,'.','Color',cmap(8,:))

project8a = V(eight,m1)'*data(:,eight)'; project8b = V(eight,m2)'*data(:,eight)'; project8c = V(eight,m
plot3(project8a,project8b,project8c,'.','Color',cmap(9,:))

project9a = V(nine,m1)'*data(:,nine)'; project9b = V(nine,m2)'*data(:,nine)'; project9c = V(nine,m3)'*da
plot3(project9a,project9b,project9c,'.','Color',cmap(10,:))

legend('0','1','2','3','4','5','6','7','8','9')
title(['Digit Projections onto V-modes ',num2str(m1),', ',num2str(m2),', and ',num2str(m3)])

print('-dpng','mode_projection.png')

%% Part 2 - LDA

[images, labels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
num_img = size(images,3);
data = zeros(28*28,num_img);
for i = 1:num_img
    data(:,i) = reshape(images(:,:,i),[28*28 1]);
end
[U,S,V] = svd(data,'econ');
zero = find(labels==0);
one = find(labels==1);
two = find(labels==2);
three = find(labels==3);
four = find(labels==4);
```

```
five = find(labels==5);
six = find(labels==6);
seven = find(labels==7);
eight = find(labels==8);
nine = find(labels==9);
r = 64;
% LDA between 0 and 1 (with r modes from part 1)
new_data = S*V';
zeroS = new_data(1:r,zero);
oneS = new_data(1:r,one);

n0 = size(zeroS,2);
n1 = size(oneS,2);

md = mean(zeroS,2);
mc = mean(oneS,2);

Sw = 0; % within class variances
for k = 1:n0
    Sw = Sw + (zeroS(:,k) - md)*(zeroS(:,k) - md)';
end
for k = 1:n1
    Sw =  Sw + (oneS(:,k) - mc)*(oneS(:,k) - mc)';
end

Sb = (md-mc)*(md-mc)'; % between class

[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
vzeros = w'*zeroS;
vones = w'*oneS;
sort0 = sort(vzeros);
sort1 = sort(vones);
t1 = length(sort0);
t2 = 1;
while sort0(t1) > sort1(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sort0(t1) + sort1(t2))/2;
count_zero = 0;
for i = 1:length(vzeros)
    if (vzeros(i) < threshold)
        count_zero = count_zero + 1;
    end
end
count_zero/length(vzeros)
count_ones = 0;
for i = 1:length(vones)
    if (vones(i) > threshold)
        count_ones = count_ones + 1;
    end
```

```
end
count_ones/length(vones)


%% Part 2 - Plot the 0 and 1 LDA results
plot(vzeros,zeros(n0),'ob','Linewidth',2)
hold on
plot(vones,ones(n1),'or','Linewidth',2)
ylim([0 1.2]); title('LDA on Zeros and Ones'); xlabel('Left = more 0, Right = more 1')
yticks([0 1]); yticklabels({'Zeros','Ones'})

print('-dpng','zero_one_LDA.png')

%% Part 3 - Make histogram for easiest to seperate part

subplot(1,2,1)
histogram(sort(vzeros)); hold on, plot([threshold threshold], [0 160],'r','Linewidth',2)
title('Zeros')
subplot(1,2,2)
histogram(sort(vones)); hold on, plot([threshold threshold], [0 180],'r','Linewidth',2)
title('Ones')

print('-dpng','hist_0_1.png')

%% Part 2 - LDA with three variables
% LDA between 0, 1, and 5 (with r modes from part 1)
new_data = S*V';
zeroS = new_data(1:r,zero);
oneS = new_data(1:r,one);
fiveS = new_data(1:r,five);

n0 = size(zeroS,2);
n1 = size(oneS,2);
n5 = size(fiveS,2);

m0 = mean(zeroS,2);
m1 = mean(oneS,2);
m5 = mean(fiveS,2);
m = (m0+m1+m5)/3;

Sw = 0; % within class variances
for k = 1:n0
    Sw = Sw + (zeroS(:,k) - m0)*(zeroS(:,k) - m0)';
end
for k = 1:n1
    Sw =  Sw + (oneS(:,k) - m1)*(oneS(:,k) - m1)';
end
for k = 1:n5
    Sw =  Sw + (fiveS(:,k) - m5)*(fiveS(:,k) - m5)';
end
sb1 = (m0-m)*(m0-m)';
sb2 = (m1-m)*(m1-m)';
sb3 = (m5-m)*(m5-m)';
Sb = (sb1+sb2+sb3)/3;
```

```
[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
vzeros = w'*zeroS;
vones = w'*oneS;
vfives = w'*fiveS;

vzeros = sort(vzeros); vones = sort(vones); vfives = sort(vfives);

%% Plot the 3 LDA
plot(vzeros,zeros(n0),'ob','Linewidth',2)
hold on
plot(vones,ones(n1),'or','Linewidth',2)
plot(vfives,2*ones(n5),'og','Linewidth',2)
ylim([0 2.4]); title('LDA on Zeros, Ones, and Fives');
yticks([0 1 2]); yticklabels({'Zeros','Ones','Fives'})
print('-dpng','lda3cases.png')

%% Part 3 - Hardest digits to seperate
% LDA between 4 and 9, I use all the zero and one nomenclature to speed up
% the code writing process! (zeroS would be analagous to fourS)
new_data = S*V';
zeroS = new_data(1:r,four);
oneS = new_data(1:r,nine);

n0 = size(zeroS,2);
n1 = size(oneS,2);

md = mean(zeroS,2);
mc = mean(oneS,2);

Sw = 0; % within class variances
for k = 1:n0
    Sw = Sw + (zeroS(:,k) - md)*(zeroS(:,k) - md)';
end
for k = 1:n1
    Sw =  Sw + (oneS(:,k) - mc)*(oneS(:,k) - mc)';
end

Sb = (md-mc)*(md-mc)'; % between class

[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
vzeros = w'*zeroS;
vones = w'*oneS;
sort0 = sort(vzeros);
sort1 = sort(vones);
t1 = length(sort0);
t2 = 1;
while sort0(t1) > sort1(t2)
```

```
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sort0(t1) + sort1(t2))/2;
count_zero = 0;
for i = 1:length(vzeros)
    if (vzeros(i) < threshold)
        count_zero = count_zero + 1;
    end
end
count_zero/length(vzeros)
count_ones = 0;
for i = 1:length(vones)
    if (vones(i) > threshold)
        count_ones = count_ones + 1;
    end
end
count_ones/length(vones)

%% Part 3 - Make histogram for hardest to seperate part
figure1 = figure('Position', [100, 100, 1600, 600]);
subplot(1,3,1)
plot(vzeros,zeros(n0),'ob','Linewidth',2)
hold on
plot(vones,ones(n1),'or','Linewidth',2)
ylim([0 1.2]); title('LDA on Fours and Nines'); xlabel('Left = more 4, Right = more 9')
yticks([0 1]); yticklabels({'Fours','Nines'})
threshold = (sort0(t1) + sort1(t2))/2;
subplot(1,3,2)
histogram(sort(vzeros)); hold on, plot([threshold threshold], [0 180],'r','Linewidth',2)
title('Fours')
subplot(1,3,3)
histogram(sort(vones)); hold on, plot([threshold threshold], [0 160],'r','Linewidth',2)
title('Nines')

print('-dpng','plots_4_9.png')

%% Part 4 - Load in training data
% Load
[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
num_img = size(images,3);
data = zeros(28*28,num_img);
for i = 1:num_img
    data(:,i) = reshape(images(:,:,i),[28*28 1]);
end

zero = find(labels==0);
one = find(labels==1);
four = find(labels==4);
nine = find(labels==9);

% Compute SVD and project
r = 64;
[U,S,V] = svd(data,'econ');
```

```
sv = S*V';
proj_data = sv ./ max(sv(:));
proj_data = proj_data(1:r,:);

svm_dat = cat(2,proj_data(:,four),proj_data(:,nine));
svm_lab = cat(1,labels(four),labels(nine));

%% Compute Decision tree and SVM on training data
tree=fitctree(proj_data',labels);
%%
Mdl = fitcecoc(svm_dat',svm_lab);

%% Predict with test data
% Load in test data
[images, labels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
num_img = size(images,3);
data = zeros(28*28,num_img);
for i = 1:num_img
    data(:,i) = reshape(images(:,:,i),[28*28 1]);
end

% Compute SVD and project
[U,S,V] = svd(data,'econ');
sv = S*V';
proj_data = sv ./ max(sv(:));
proj_data = proj_data(1:r,:);

% Redefine indices for test data
zero = find(labels==0);
one = find(labels==1);
four = find(labels==4);
nine = find(labels==9);

svm_lab = cat(1,labels(zero),labels(one));
svm_dat = cat(2,proj_data(:,four),proj_data(:,nine));

%% Test decision tree and compute accuracies
test_tree = predict(tree,proj_data');
count_zero = 0;
tester = test_tree(four);
for i = 1:length(tester)
    if (tester(i) == 4)
        count_zero = count_zero + 1;
    end
end
count_zero / length(tester)

count_ones = 0;
tester = test_tree(nine);
for i = 1:length(tester)
    if (tester(i) == 9)
        count_ones = count_ones + 1;
    end
end
```

```
count_ones / length(tester)

%% Test SVM data
test_svm = predict(Mdl,svm_dat');

zeroSV = 1:length(proj_data(zero));
oneSV = length(zero)+1:length(svm_dat);
fourSV = 1:length(proj_data(four));
nineSV = length(four)+1:length(svm_dat);

count_zero = 0;
tester = test_svm(fourSV);
for i = 1:length(tester)
    if (tester(i) == 4)
        count_zero = count_zero + 1;
    end
end
count_zero / length(tester)

count_ones = 0;
tester = test_svm(nineSV);
for i = 1:length(tester)
    if (tester(i) == 9)
        count_ones = count_ones + 1;
    end
end
count_ones / length(tester)
```

Code from homework4.m