John Curtis

Assignment 4: Classifying Digits

Abstract

This paper explores using the principal component analysis to determine the key components that differentiate images of handwritten digits, as well as the use of the linear discriminant analysis to classify between digits. Furthermore, the LDA method was compared with the decision trees and support vector machines methods of classification. The results showed that the LDA classification performs significantly better when classifying between just 2 digits but not more, while decision trees and SVM generally does well classifying any number of digits.

Introduction and Overview

Linear discriminant analysis is a method of finding projections of sets of data to minimise the overlap between the sets. In conjunction with principal component analysis, it can be used to classify data through a supervised machine learning algorithm. In this paper we will use the principal component analysis to determine the key features of images of hand drawn numeral digits and then use linear discriminant analysis to classify the images digits based on these features.

Theoretical Background

The principal component analysis (PCA) can be used to decompose a set of data into matrices U, V and Σ that represent the fundamental modes that make it up. These modes can be used to determine what features are most important in determining the difference between sets of data. Specifically, the matrix U is comprised of vectors that represent the spatial modes, the components that make up each image, while the matrix V is comprised of vectors that represent the time modes, the components that show how the space modes vary from image to image. The matrix Σ contains the singular values that dictate how significant the different modes are in comprising the initial data.

For example, in this case of images of digits, performing PCA will allow us to see the modes of the digits and then use them to find what components differentiate between the digits. If we keep only the modes with the highest corresponding singular values, we can find a low rank representation of the differences between the digits.

If we have data that is comprised of different groups or classes, linear discriminant analysis (LDA) is a process that optimises a projection of the data such that the space between groups is maximised but the spread within each group is minimised. In other words, for 2 groups if we define the between-class scatter matrix and within-class scatter matrix respectively to be:

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

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

where μ_j is the mean of group j, and x is the values for one instance of the data (ie. one digit). Then by finding a projection W such that:

$$W = argmax\left(\frac{W^T S_B W}{W^T S_w W}\right) \quad Eq. 3$$

This is usually difficult but can be done relatively quickly by a computer using the eigenvalue equation:

$$S_B W = \lambda S_w W \quad Eq. 4$$

We can also generalise this for cases where there are more than 2 classes, in which case the scatter matrices become:

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

$$S_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T \quad Eq. 6$$

and W is found the same way as above. W is then the optimal vector to project our data on to minimise the spread within groups and maximise the distance between groups.

The by using this projection we can find a threshold value where values on one side belong to one group and values on the other side belong to the other group. This threshold can be found by making it such that the number of incorrect classifications for both groups is equal.

Algorithm Implementation and Development

First I loaded all the training image data for the digits and transformed them so that each column corresponded to the data of each image. I subtracted the mean of each image from the image to normalise and remove the background.

Then I performed the singular value decomposition (SVD) and projected the training data onto the PCA basis. I plotted all the singular values of the SVD to determine how many modes to use. I also plotted the first few modes and the projection of the image data onto 3 arbitrary modes to see if the modes do describe some differences between digits.

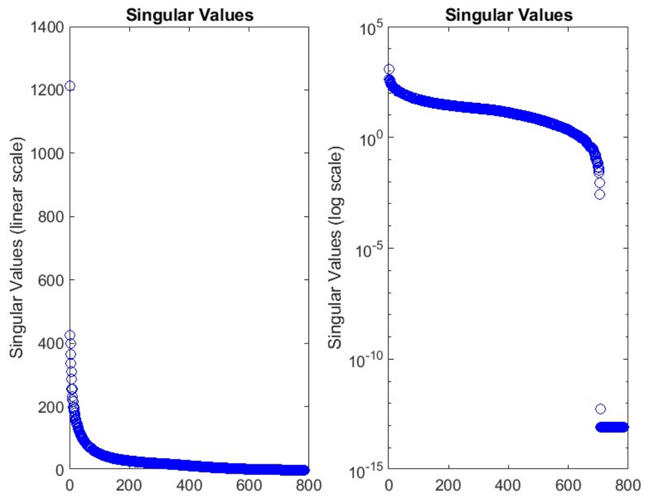
I created a function that calculates the LDA for any number of digits to project. This was done by first calculating the scatter matrices as defined in equations 1,2,5 and 6, solving the eigenvalue equation of equation 4 to find the projection vector W, and then finding the threshold using the iterative method described earlier.

I used this function to train the LDA model on the training data for 2 digits. Then I found the projection of the test data for 2 digits onto W and used the threshold value to classify the digits. After that I calculated the accuracy of the classifications of both training and test data by dividing the number of correct classifications by the total number of image data instances.

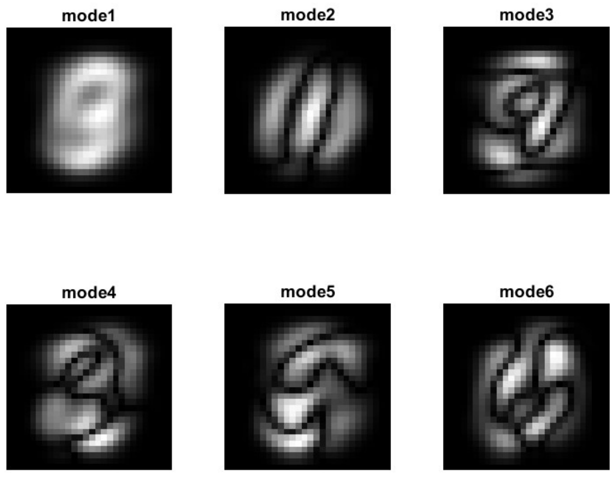
I then repeated this process for 3 digits, calculating the LDA, projecting the test data onto W, and finding the accuracy of the classifications. I also iteratively performed the 2 digit classification for all combinations of digits, finding the classification accuracy of each, to find the combination of digits that are easiest and hardest to differentiate with the classifier.

Then I used MatLab’s built-in functions to create and find the accuracy of classifiers of all 10 digits using decision trees and support vector machines (SVM). After which I used these 2 classifiers to classify the easiest and hardest to differentiate digits found earlier, comparing these results to the LDA.

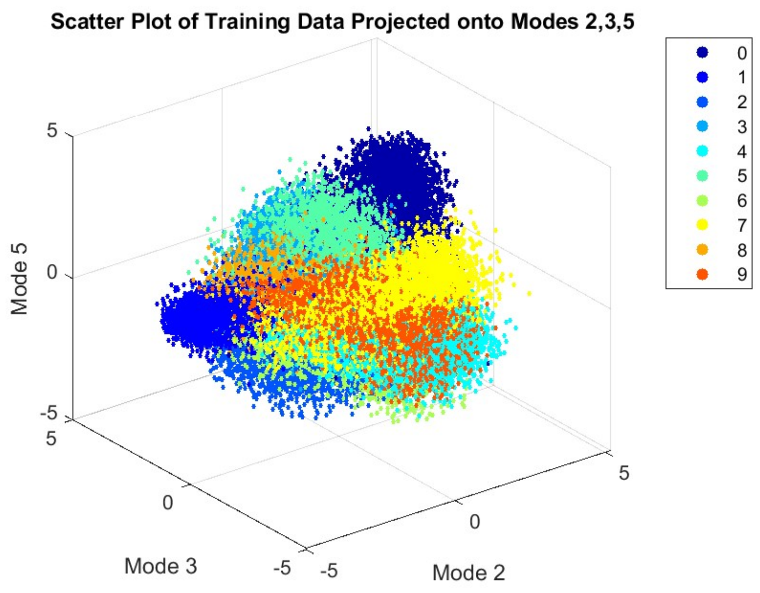
Computational Results



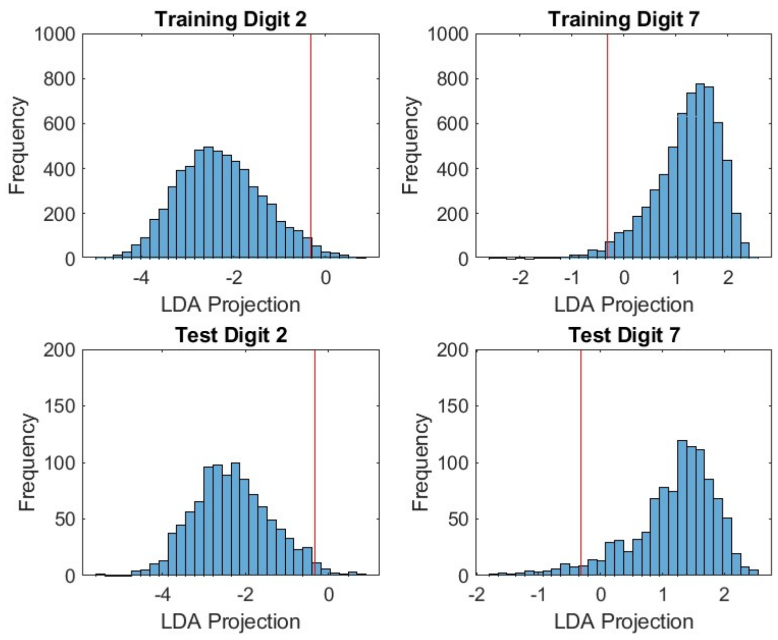
From this we can see that the first singular value contains a significant portion of the information but then there is a very gradual decrease so there are many other important singular values. From this plot and the visualisation of the modes, I decided to use the first 20 features.



This visualisation of the first 6 spatial modes shows show the PCA components that make up and differentiate each digit. We can see that the shapes in these modes do resemble the general shapes of the digits.

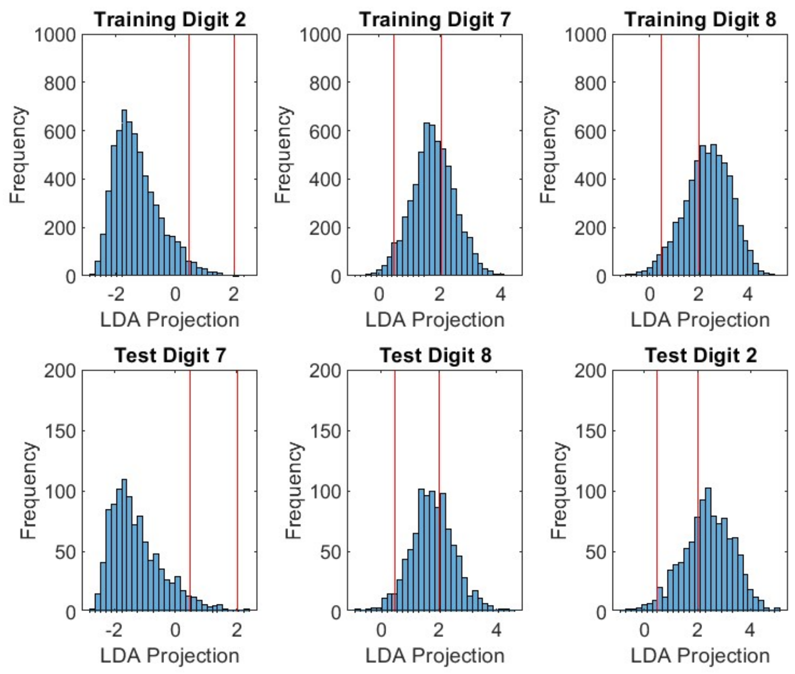


We can see from this scatter plot that even with just 3 arbitrary modes, there is some clustering of the digits 0 to 9, which shows that even a few modes can be used to differentiate the digits to an extent.



These histograms show the distribution of the LDA projection values for the training and test images of digits 2 and 7 (chosen arbitrarily), along with the red line at the threshold value. From this histogram we can see that the LDA did a relatively good job classifying the digits.

The accuracy of the training data classifications was 0.9973 and the accuracy of the test data classifications was 0.9689. We see that the classifier did really well for both training and test data but did very slightly better for the training dataset as expected since that is the dataset it used to optimise the projection vector W.



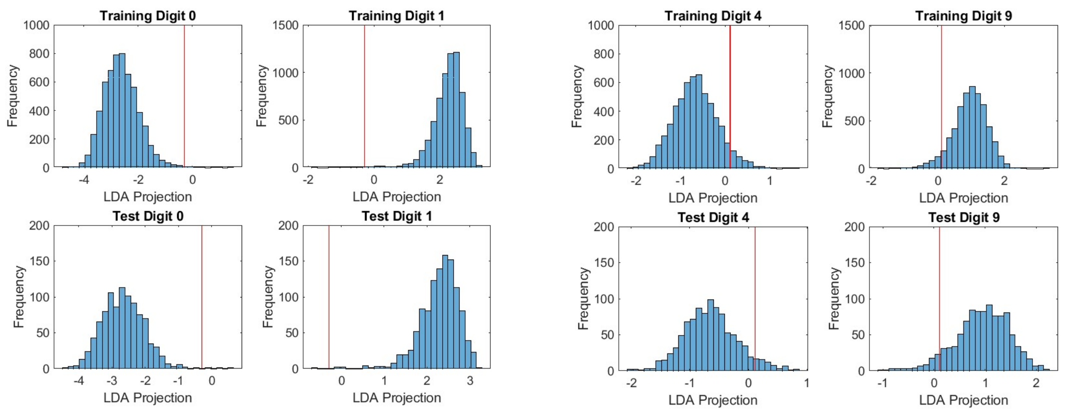
These histograms show the distribution of the LDA projection values for the training and test images of digits 2, 7 and 8 (chosen arbitrarily), along with the red lines at the threshold values. The accuracy of the training data classifications was 0.7505 and the accuracy of the test data classifications was 0.7390.

From this histogram we can see that the LDA had more trouble classifying the digits compared to the 2 digit case. This is likely due to how there is much more overlap between the projections when the data of another digit is added, so it is much more difficult to find a threshold value that cleanly separates them.

To determine the easiest and hardest to classify digits, the test data was used. After iterating through the 2-digit LDA classification for all combinations of digits in the test data, the results are as follows:

	Easiest Classification		Hardest Classification	
Digits	0	1	4	9
Accuracy	0.9981		0.9342	

First of all we see that even for the harder to classify digits, the LDA can classify them with a relatively high accuracy which shows that there aren't any digits that seem indistinguishable to the classifier. Intuitively it makes sense that the easiest to classify digits are 1 and 0 since the shapes are very different, 1 is just a straight line while 0 is a curved loop, there aren't any similar structures to the digits. It also makes sense that the hardest to classify digits are 4 and 9 since the shapes are similar when hand drawn, a loop in the top left area with a tail coming straight down, in fact depending on how one draws them they may look identical except for a segment from the top removed from the 9 to make a 4.



This is supported by the histograms of the easiest and hardest classification. We can see there is significantly more overlap and error with the threshold for digits 4 and 9, making it harder to classify them precisely.

When introducing the use of decision trees and SVM to classify the digits, we get the following accuracies of classification:

	Decision Trees	Support Vector Machines
All Digits	0.8561	0.9087
Easiest Digits	0.8662	0.9674
Hardest Digits	0.8525	0.9418

Firstly we see that both of these classifiers do relatively well classifying all 10 digits, while the LDA classifier had a lower accuracy for even just 3 digits. This is most likely because the LDA classifies using a projection onto a 1-dimensional vector, so as we increase the number of digits, there is much more overlap in the projections since we are only looking at a 1-dimensional representation of 10 different classes of images.

However, we see that the LDA classification does significantly better (~10%) than the decision trees for both the hard and easy digits, which suggests that while decision trees are better at classifying which of all 10 digits it may be, the LDA is better at distinguishing between just 2 specific digits. We also see that the easy digit accuracy is higher for LDA than SVM, while the hard digit accuracy is higher for SVM than LDA. It is not exactly clear why this is the case but it suggests that SVM is more consistent in its accuracy classifications while the LDA is more influenced by how similar the digits are. This again may be because of the 1-dimensional nature of the LDA.

Summary and Conclusion

The LDA classification was able to classify between 2 digits relatively easily with minimal difference in the accuracy for easy and hard to distinguish digits. However, when there were 3 digits, the accuracy significantly dropped due to the 1-dimensional projection of the data causing more overlaps. Both decision trees and SVM were able to classify between more than 2 digits much more effectively than the LDA, but the LDA classifier was still more accurate in certain cases. This goes to show that there is no one best classifier but it depends on what the purpose is. Where LDA may be better for classifying between 2 groups, but decision trees and SVM may be better when there are more than 2 classes to involved.

Appendix A: MatLab Functions Used

- **[U,S,V] = svd(A,'econ')** – Performs a reduced singular value decomposition of matrix A
  - https://au.mathworks.com/help/matlab/ref/double.svd.html
- **tree = fitctree(Tbl,Y)** – Returns a fitted binary classification decision tree based on the input variables contained in the table Tbl and output in vector Y.
  - https://au.mathworks.com/help/stats/fitctree.html
- **Mdl = fitcecoc(Tbl,Y)** – Returns an ECOC model using the predictors in table Tbl and the class labels in vector Y.
  - https://au.mathworks.com/help/stats/fitcecoc.html
- **label = predict(Mdl,X)** – Returns a vector of predicted class labels (label) for the predictor data in the table or matrix X, based on the model Mdl.
  - https://au.mathworks.com/help/stats/classificationecoc.predict.html
- **[V,D] = eig(A)** – Returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors.
  - https://au.mathworks.com/help/matlab/ref/eig.html
- **M = mean(A)** – Returns the mean of the elements of A.
  - https://au.mathworks.com/help/matlab/ref/mean.html
- **B = sort(A)** - Sorts the elements of A in ascending order.
  - https://au.mathworks.com/help/matlab/ref/sort.html
- **B = reshape(A,sz)** – Reshapes A using the size vector, sz, to define size(B).
  - https://au.mathworks.com/help/matlab/ref/reshape.html
- **sz = size(A)** – Returns a row vector whose elements are the lengths of the corresponding dimensions of A.
  - https://au.mathworks.com/help/matlab/ref/size.html
- **I2 = im2double(I)** – Converts the image I to double precision.
  - https://au.mathworks.com/help/matlab/ref/im2double.html
- **x = diag(A)** – Returns a column vector of the main diagonal elements of A.
  - https://au.mathworks.com/help/matlab/ref/diag.html
- **Y = zeros(X)** – Returns an n-by-n matrix of zeros.
  - https://au.mathworks.com/help/matlab/ref/zeros.html
- **[M,I] = max(A,[],’all’,'linear')** – Returns the maximum value in A as well as the linear index into A that corresponds to the maximum value in A.
  - https://au.mathworks.com/help/matlab/ref/max.html
- **[I1,I2,...,In] = ind2sub(sz,ind)** – Returns the equivalent multidimensional subscripts corresponding to the linear indices ind for a multidimensional array of size sz.
  - https://au.mathworks.com/help/matlab/ref/ind2sub.html
- **S = sum(A)** – Returns the sum of the elements of A.
  - https://au.mathworks.com/help/matlab/ref/sum.html
- **plot(X,Y,LineSpec)** – Creates a 2-D line plot of the data in Y versus the corresponding values in X.
  - https://au.mathworks.com/help/matlab/ref/plot.html
- **semilogy(X,Y,LineSpec)** – Plots x- and y-coordinates using a linear scale on the x-axis and a base-10 logarithmic scale on the y-axis.
  - https://au.mathworks.com/help/matlab/ref/semilogy.html

- **scatter3(X,Y,Z,S,C)** – Displays circles at the locations specified by the vectors X, Y, and Z with size S and color C.
    - https://au.mathworks.com/help/matlab/ref/scatter3.html
- **histogram(X,nbins)** – Creates a histogram plot of X with number of bins specified by the scalar, nbins.
    - https://au.mathworks.com/help/matlab/ref/matlab.graphics.chart.primitive.histogram.html
- **print(filename,formattype)** – Saves the current figure to a file using the specified file format.
    - https://au.mathworks.com/help/matlab/ref/print.html

Appendix B: MatLab Code

```
clear; close all; clc;

%% Prepare Training Data
[trainImg, trainLbl] = mnist_parse('train-images.idx3-
ubyte', 'train-labels.idx1-ubyte');
trainImg =
reshape(trainImg,[size(trainImg,1)*size(trainImg,2),size(
trainImg,3)]);

numTrain = size(trainImg,2);
for i = 1:numTrain
    trainImg(:,i) = trainImg(:,i) - mean(trainImg(:,i));
end
trainImg = im2double(trainImg);

%% Perform SVD
[U,S,V] = svd(trainImg,'econ');
sig = diag(S);
trainProj = S*V';

%% Plot Singular Values
x = 1:length(sig);
t = 1:numTrain;
figure(1);
subplot(1,2,1);
plot(diag(S),'bo');
title('Singular Values');
ylabel('Singular Values (linear scale)');
subplot(1,2,2);
semilogy(diag(S),'bo');
title('Singular Values');
ylabel('Singular Values (log scale)');
print('svals','-dpng');

figure(2);
for i = 1:6
    subplot(2,3,i);
    imshow(rescale(abs(reshape(U(:,i),28,28))));
    title(['mode' num2str(i)]);
end
print('pca_modes','-dpng');

figure(3);
proj = trainProj([2,3,5],:);
CM = jet(10);
for i = 0:9
```

```
    plotting = proj(:,(trainLbl==i));

scatter3(plotting(1,:),plotting(2,:),plotting(3,:),5,CM(i
+1,:),'filled');
    hold on;
end
title('Scatter Plot of Training Data Projected onto Modes
2,3,5');
xlabel('Mode 2'); ylabel('Mode 3'); zlabel('Mode 5');
legend('0','1','2','3','4','5','6','7','8','9');
print('proj_scatter','-dpng');

feature = 20;

%% Prepare Test Data
[testImg, testLbl] = mnist_parse('t10k-images.idx3-
ubyte', 't10k-labels.idx1-ubyte');
testImg =
reshape(testImg,[size(testImg,1)*size(testImg,2),size(tes
tImg,3)]);

numTest = size(testImg,2);
for i = 1:numTest
    testImg(:,i) = testImg(:,i) - mean(testImg(:,i));
end
testImg = im2double(testImg);

%% Perform LDA With 2 Digits
digits = {2,7};
dA = trainProj(1:feature,(trainLbl==digits{1}));
dB = trainProj(1:feature,(trainLbl==digits{2}));
[thresh,w,sortV,order] = digit_trainer({dA,dB});

figure(4);
subplot(2,2,1);
histogram(sortV{1},30); hold on, plot([thresh{1}
thresh{1}], [0 1000],'r');
title(['Training Digit ',num2str(digits{1})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,2,2);
histogram(sortV{2},30); hold on, plot([thresh{1}
thresh{1}], [0 1000],'r');
title(['Training Digit ',num2str(digits{2})]);
xlabel('LDA Projection'); ylabel('Frequency');
trainAcc =
(sum(sortV{1}<thresh{1})+sum(sortV{2}>thresh{1}))/(size(d
A,2)+size(dB,2))
```

```
testProj = U'*testImg;
tA = w'*testProj(1:feature,(testLbl==digits{order(1)}));
tB = w'*testProj(1:feature,(testLbl==digits{order(2)}));
subplot(2,2,3);
histogram(tA,30); hold on, plot([thresh{1} thresh{1}], [0
200],'r');
title(['Test Digit ',num2str(digits{order(1)})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,2,4);
histogram(tB,30); hold on, plot([thresh{1} thresh{1}], [0
200],'r');
title(['Test Digit ',num2str(digits{order(2)})]);
xlabel('LDA Projection'); ylabel('Frequency');
testAcc =
(sum(tA<thresh{1})+sum(tB>thresh{1}))/(size(tA,2)+size(tB
,2))
print('lda_hist2','-dpng');

%% Perform LDA With 3 Digits
digits = {2,7,8};
dA = trainProj(1:feature,(trainLbl==digits{1}));
dB = trainProj(1:feature,(trainLbl==digits{2}));
dC = trainProj(1:feature,(trainLbl==digits{3}));
[thresh,w,sortV,order] = digit_trainer({dA,dB,dC});

figure(5);
subplot(2,3,1);
histogram(sortV{1},30);
hold on, plot([thresh{1} thresh{1}], [0 1000],'r'),
plot([thresh{2} thresh{2}], [0 1000],'r');
title(['Training Digit ',num2str(digits{1})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,3,2);
histogram(sortV{2},30);
hold on, plot([thresh{1} thresh{1}], [0 1000],'r'),
plot([thresh{2} thresh{2}], [0 1000],'r');
title(['Training Digit ',num2str(digits{2})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,3,3);
histogram(sortV{3},30);
hold on, plot([thresh{1} thresh{1}], [0 1000],'r'),
plot([thresh{2} thresh{2}], [0 1000],'r');
title(['Training Digit ',num2str(digits{3})]);
xlabel('LDA Projection'); ylabel('Frequency');
trainAcc =
(sum(sortV{1}<thresh{1})+sum(sortV{2}>thresh{1}&sortV{2}<
thresh{2})+sum(sortV{3}>thresh{2}))/(size(dA,2)+size(dB,2
)+size(dC,2))

testProj = U'*testImg;
tA = w'*testProj(1:feature,(testLbl==digits{order(1)}));
tB = w'*testProj(1:feature,(testLbl==digits{order(2)}));
tC= w'*testProj(1:feature,(testLbl==digits{order(3)}));
subplot(2,3,4);
histogram(tA,30);
hold on, plot([thresh{1} thresh{1}], [0 200],'r'),
plot([thresh{2} thresh{2}], [0 200],'r');
title(['Test Digit ',num2str(digits{order(1)})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,3,5);
histogram(tB,30);
hold on, plot([thresh{1} thresh{1}], [0 200],'r'),
plot([thresh{2} thresh{2}], [0 200],'r');
title(['Test Digit ',num2str(digits{order(2)})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,3,6);
histogram(tC,30);
hold on, plot([thresh{1} thresh{1}], [0 200],'r'),
plot([thresh{2} thresh{2}], [0 200],'r');
title(['Test Digit ',num2str(digits{order(3)})]);
xlabel('LDA Projection'); ylabel('Frequency');
print('lda_hist3','-dpng');
testAcc =
(sum(tA<thresh{1})+sum(tB>thresh{1}&tB<thresh{2})+sum(tC>
thresh{2}))/(size(tA,2)+size(tB,2)+size(tC,2))

%% Compare Digits
accuracies = zeros(10,10,2);
for A = 0:9
    for B = A+1:9
        digits = {A,B};
        dA = trainProj(1:feature,(trainLbl==A));
        dB = trainProj(1:feature,(trainLbl==B));
        [thresh,w,sortV,order] = digit_trainer({dA,dB});
        tA =
w'*testProj(1:feature,(testLbl==digits{order(1)}));
        tB =
w'*testProj(1:feature,(testLbl==digits{order(2)}));

        trainAcc =
(sum(sortV{1}<thresh{1})+sum(sortV{2}>thresh{1}))/(size(d
A,2)+size(dB,2));
```

```
        testAcc =
(sum(tA<thresh{1})+sum(tB>thresh{1}))/(size(tA,2)+size(tB
,2));
        accuracies(A+1,B+1,1) = trainAcc;
        accuracies(A+1,B+1,2) = testAcc;
    end
end
accuracies(accuracies == 0) = NaN;
[accMax,Imax] = max(accuracies(:,:,2),[],'all','linear');
[maxX,maxY] = ind2sub([10,10],Imax);
[accMin,Imin] = min(accuracies(:,:,2),[],'all','linear');
[minX,minY] = ind2sub([10,10],Imin);
maxX=maxX-1; maxY=maxY-1; minX=minX-1; minY=minY-1;

%% Plot Best and Worst Digits
digits = {0,1};
dA = trainProj(1:feature,(trainLbl==digits{1}));
dB = trainProj(1:feature,(trainLbl==digits{2}));
[thresh,w,sortV,order] = digit_trainer({dA,dB});

figure(6);
subplot(2,2,1);
histogram(sortV{1},30); hold on, plot([thresh{1}
thresh{1}], [0 1000],'r');
title(['Training Digit ',num2str(digits{1})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,2,2);
histogram(sortV{2},30); hold on, plot([thresh{1}
thresh{1}], [0 1500],'r');
title(['Training Digit ',num2str(digits{2})]);
xlabel('LDA Projection'); ylabel('Frequency');

testProj = U'*testImg;
tA = w'*testProj(1:feature,(testLbl==digits{order(1)}));
tB = w'*testProj(1:feature,(testLbl==digits{order(2)}));
subplot(2,2,3);
histogram(tA,30); hold on, plot([thresh{1} thresh{1}], [0
200],'r');
title(['Test Digit ',num2str(digits{order(1)})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,2,4);
histogram(tB,30); hold on, plot([thresh{1} thresh{1}], [0
200],'r');
title(['Test Digit ',num2str(digits{order(2)})]);
xlabel('LDA Projection'); ylabel('Frequency');
print('lda_hist_best','-dpng')
```

```
digits = {4,9};
dA = trainProj(1:feature,(trainLbl==digits{1}));
dB = trainProj(1:feature,(trainLbl==digits{2}));
[thresh,w,sortV,order] = digit_trainer({dA,dB});

figure(7);
subplot(2,2,1);
histogram(sortV{1},30); hold on, plot([thresh{1}
thresh{1}], [0 1000],'r');
title(['Training Digit ',num2str(digits{1})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,2,2);
histogram(sortV{2},30); hold on, plot([thresh{1}
thresh{1}], [0 1500],'r');
title(['Training Digit ',num2str(digits{2})]);
xlabel('LDA Projection'); ylabel('Frequency');

testProj = U'*testImg;
tA = w'*testProj(1:feature,(testLbl==digits{order(1)}));
tB = w'*testProj(1:feature,(testLbl==digits{order(2)}));
subplot(2,2,3);
histogram(tA,30); hold on, plot([thresh{1} thresh{1}], [0
200],'r');
title(['Test Digit ',num2str(digits{order(1)})]);
xlabel('LDA Projection'); ylabel('Frequency');
subplot(2,2,4);
histogram(tB,30); hold on, plot([thresh{1} thresh{1}], [0
200],'r');
title(['Test Digit ',num2str(digits{order(2)})]);
xlabel('LDA Projection'); ylabel('Frequency');
print('lda_hist_worst','-dpng')

%% Predict Using Decision Tree
tree =
fitctree(array2table(trainProj(1:feature,:)'),trainLbl');
predicted = predict(tree,testProj(1:feature,:)');
error = sum(predicted == testLbl);
treeAcc = error/length(predicted)

%% Predict Using SVM
Mdl =
fitcecoc(array2table(trainProj(1:feature,:)'),trainLbl');
predicted = predict(Mdl,testProj(1:feature,:)');
error = sum(predicted == testLbl);
svmAcc = error/length(predicted)

%% Compare Best and Worst
```

```
subsetImg = trainProj(:,(testLbl==0|testLbl==1));
subsetLbl = trainLbl(testLbl==0|testLbl==1);

tree =
fitctree(array2table(subsetImg(1:feature,:)'),subsetLbl')
;
predicted =
predict(tree,testProj(1:feature,(testLbl==0|testLbl==1))'
);
error = sum(predicted == testLbl(testLbl==0|testLbl==1));
treeAccBest = error/length(predicted)

Mdl =
fitcecoc(array2table(subsetImg(1:feature,:)'),subsetLbl')
;
predicted =
predict(Mdl,testProj(1:feature,(testLbl==0|testLbl==1))')
;
error = sum(predicted == testLbl(testLbl==0|testLbl==1));
svmAccBest = error/length(predicted)

subsetImg = trainProj(:,(testLbl==4|testLbl==9));
subsetLbl = trainLbl(testLbl==4|testLbl==9);

tree =
fitctree(array2table(subsetImg(1:feature,:)'),subsetLbl')
;
predicted =
predict(tree,testProj(1:feature,(testLbl==0|testLbl==1))'
);
error = sum(predicted == testLbl(testLbl==0|testLbl==1));
treeAccWorst = error/length(predicted)

Mdl =
fitcecoc(array2table(subsetImg(1:feature,:)'),subsetLbl')
;
predicted =
predict(Mdl,testProj(1:feature,(testLbl==0|testLbl==1))')
;
error = sum(predicted == testLbl(testLbl==0|testLbl==1));
svmAccWorst = error/length(predicted)
```

```
function [thresh,w,sortV,order] = digit_trainer(digits)

    count = length(digits);
    means = cell(count,1);
    for i = 1:count
        means{i} = mean(digits{i},2);
    end
    overallMean = mean(cat(2,means{:}),2);

    Sw = 0;
    Sb = 0;
    for i = 1:count
        d = digits{i};
        for j = 1:size(d,2)
            Sw = Sw + (means{i}-d(:,j))*(means{i}-
d(:,j))';
        end
        Sb = Sb + (means{i}-overallMean)*(means{i}-
overallMean)';
    end

    [V2, D] = eig(Sb,Sw);
    [lambda, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    v = cell(count,1);
    sorts = cell(count,1);
    temp = zeros(count,1);
    for i = 1:count
        v{i} = w'*digits{i};
        sorts{i} = sort(v{i});
        temp(i) = mean(v{i});
    end
    [sortMeans,order] = sort(temp);
    sortV = sorts(order);

    thresh = cell(count-1,1);
    for i = 1:count-1
        t1 = length(sortV{i});
        t2 = 1;
        while sortV{i}(t1) > sortV{i+1}(t2)
            t1 = t1 - 1;
            t2 = t2 + 1;
        end
        thresh{i} = (sortV{i}(t1) + sortV{i+1}(t2))/2;
    end
```