AMATH 482 Homework 4

Biyao Li

Mar 6, 2021

Abstract

In this assignment, we will use the Singular Value Decomposition(SVD) to perform a Linear Discriminant Analysis(LDA) on the images of the MNIST digit training set. The analysis will result in a classifier which can be used to distinguish the digit images in the MNIST testing set.

1 Introduction and Overview

After loading the MNIST training data into MATLAB, we will get a 28 by 28 by 60000 matrix which represents 60000 28 by 28 images where each image contains a digit from 0 to 9. In order to perform the SVD on this data, we need to transform the data into a new matrix where each column represent a image and this will results in a 784 by 60000 matrix. Moreover, we are given a 1 by 60000 labels matrix that tell us the digit corresponding to each image. After applying the SVD to the data, we can project the data onto principal components and the projected data will allow us to compute the LDA classifer. Furthermore, the LDA will give us a threshold that can be used to separate two or more digits and it will allow us to identify the corresponding digits in the testing set.

2 Theoretical Background

2.1 Singular Value Decomposition(SVD)

For any matrix $A \in C^{m\times n}$, we can apply the Singular Value Decomposition to A such that

$$A = U\Sigma V^* \tag{1}$$

Where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are unitary matrices, and $\Sigma \in \mathbb{R}^{m\times n}$ diagonal matrix. The columns of U are the left singular vectors of A and the columns of V are the right singular vectors of A. The values on the diagonal of Σ are the singular values of A and if we let σ_1 be the first diagonal value and so on, the singular values of A are always ordered from the largest to smallest such that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$. Moreover, since U and V are unitary matrix, it is true that $UU^* = I$ and $VV^* = I$ where I is the identity matrix.

2.2 Low rank approximation

After applying the SVD to a matrix A, with r as the rank of A, we can define A as $\sum_{j=1}^{r} \sigma_j u_j v_j^*$. Furthermore, for any value s where $1 \leq s \leq r$, we define $\sum_{j=1}^{s} \sigma_j u_j v_j^*$ as the rank s approximation of the data. If we have images stored in each column of A, the rank s approximation of A will allow us to approximate each image with fewer pixels. Moreover, we can find a value n that is the smallest rank of a approximation that keeps the majority information of the image while having the least amount of pixels. This is the low rank approximation. In this assignment, after we found the appropriate value of n, we will project the data matrix onto n principal components modes and we refer to these modes as the "features".

2.3 Linear Discriminant Analysis(LDA)

In this assignment, one of the goal is to apply the Linear Discriminant Analysis on any two digits. This can be done by projecting the data of two digits onto a line and hopefully, these data on the new line are completely separated such that we can define a threshold value that can be used to determine whether a image is more of the first digit or the second digit. In order to achieve of good separation of data, we need to ensure that we are maximizing the distance between the mean of two data groups and minimizing the variance within each data group. This can be done by the following procedure:

- Define S_B as $(\mu_2 - \mu_1)(\mu_1 - \mu_1)^T$ where S_B is the between class scatter matrix that measures the variance between the groups and μ_i is the means of each group for each features.
- Define S_w as $\sum_{n=1}^{2} \sum_x (x - \mu_i)(x - \mu_i)^T$ where S_w is the within class scatter matrix that measures the variance within each group and x is each column of the projected data.
- Based on this, we want to find the vector w that maximize $\frac{w^T S_B w}{w^T S_w w}$. This is because in order to obtain a large value, we need to have a large value of S_B which means the variance between the class is large and have a small value of S_w such that the variance within each class is small. This can be solved by finding the eigenvector corresponding to the largest eigenvalue in the equation $S_B w = \lambda S_w w$. In the end, the best projection line will be spanned by the vector w.

After we find the vector w, we will need to project the projected data of each digit onto w which will results in n points on a line where n is the sum of the images for the two digits. Moreover, in order to find the threshold, we just need to find the midpoint of the n points on the line.

2.4 Support vector machine(SVM)

The support vector machine allows us to find a hyperplane in the Nth dimension that is able to classify all the data points where N is the number of features.

2.5 Decision tree

Decision tree is a method for classification where it keep splitting the entire data set into subparts based on some values.

3 Algorithm Implementation and Development

3.1 Perform SVD on the images of the training set

Recall that before we apply the SVD on the data matrix, we first need to reshape the given data into a 784 by 60000 matrix. This can be done by iterating through every image and use the **reshape** command to transform each 28 by 28 matrix in a 784 by 1 column vector. Then we can use the **svd** command to apply the Singular Value Decomposition to the data matrix. This part of the code in appendix B is from line 4 to 13.

3.2 Rank analysis

As mentioned before, we will be projecting our data onto n principle component where n can be found by low rank approximation. In this assignment, in order to find the value of n, I approximated the first image which is the digit 5 using 15 values of rank which range from 20 to 90 increments by 5. Then for each matrix of the approximated data, I transformed the first column into a 28 by 28 image using the **reshape** and **imshow** command. By looking through each approximation, we can determine which rank is a good value for n. This part of the code in appendix B is from line 23 to 31.

3.3 U, Sigma, V matrices and V-modes projection

After we apply the **svd** command on the training data set, we will get three matrices: U, Σ, V. In the context of this assignment, the U matrix is the principal component basis and the Σ and V matrix together give us the information regarding the direction and the magnitude of each principal component basis vector for each of the 60000 images.

In this assignment, we are asked to project three columns of V matrix onto a 3-D plane. From this plot, we can analysis how each images spread out in space for each of the 3 modes. In appendix B, the code from this projection is from line 33 to 41 and we will be projecting onto the first, second, third columns of V. Moreover, we can see that I color each digit with a unique color so that we can visualize which digits are separated the least and the most.

3.4 2 digits classifier

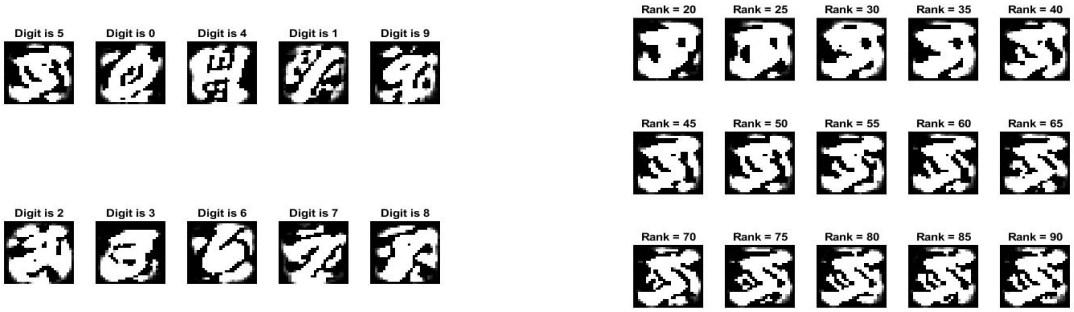
Creating a 2 digits classifier is easy since we just need to follow the steps in section 2.3. In this assignment, I created a MATLAB function called **digitLDA** that contains the code for creating a 2 digits classifier. This function will return the threshold value and w and it has 4 parameters: the first digit, the second digit, the entire training data set, and the label matrix. This function makes it easier to compute the 2 digits classifier and efficiently avoid redundancy in our code. Moreover, I plotted the projected points for one digit on the line of y = 0 and another digit on the line of y = 1 so that we can visualize the separation.

3.5 3 digits classifier

To create a 3 digits classifier, we need to make some adjustment to how we computed S_B and S_w in section 2.3. For S_B, we redefine it as $\sum_{j=1}^{N}(\mu_j - \mu)(\mu_j - \mu)^T$ where μ is the average of all μ_j. For S_w, we just need to change the upper bound of the outer summation to 3. An example of the 3 digits classifier in the code of appendix B is from line 83 to 124 which created a classifier for digits 3,6,8.

3.6 Difficult and easy 2 digits classifier

In this assignment, we need to create two classifiers where one is for digits that is difficult to separate and another one is for digits that is easy to separate. From the rank analysis, I found the value n for the low rank analysis is 50. I used this value to create a plot that consists of 10 images for the rank 50 approximation of the 10 digits. From images on the left hand side of figure 1, we can see that images for digits 2 and 5 have a similar structure and images for digit 0 and 9 differ the most. Hence I predict that the classifier for 2 and 5 is going to be difficult to separate while the classifier for 0 and 9 is easy to separate.



(a) Rank 50 approximations

(b) Rank 20 to 90 approximations

Figure 1: Rank 50 approximation for each of the 10 digits on the right and rank 20 to 90 approximation for digit 5.

To see how good the separation is, we need to apply the classifier onto the testing set. For the difficult digit classifier, this can be done with the following procedure:

- For the testing set, reshape the data matrix into the 784 by 10000 matrix where each column represent a image and project this matrix onto PCA basis.
- Pick all columns that represent a image with the digit of 2 or 5. Call this new matrix as A'.
- Project the data in A' onto w where w is the projection line span vector that is returned from the **digitLDA** function.
- Using the threshold that is returned from the **digitLDA** function, we can determine whether a image in A' is 2 or 5.
- With the label matrix for the testing set, we can determine the actual number of images that display a digit of 2 or 5. Then we can calculate the accuracy of the classifier by dividing the number of images classified as 2 and 5 with the total count.

The same process can be applied to the easy digits and this part of the code in appendix are line 57 to 68 for the difficult classifier and line 70 to 81 for the easy classifier.

3.7 SVM and decision tree

Recall from the theoretical background section, I mentioned two other classification methods: Support vector machine(SVM) and the decision tree. For the last part of this assignment, we will be comparing the accuracy of the three classification methods used in this assignment. For SVM, we will use the **fitcecoc** function for 10 digits separation and **fitcsvm** function for 2 digits separation. For decision tree, we will be using the **fitctree** method function for both 10 digits and 2 digits separation. For both SVM and decision tree, I will perform the following procedure:

- Separate between all 10 digits.
- Separate the two easy digits, 0 and 9.
- Separate the two difficult digits, 2 and 5.

Moreover, I will apply the classifiers on the testing set to obtain the accuracy. All code related to the computation for SVM is from line 157 to 180 and all code related to the computation for the decision tree is from line 136 to 156 in appendix B.

4 Computational Results

4.1 Low rank approximation analysis

In the section 3.6, I mentioned that the appropriate value of n for the low rank approximation is 50. We can see this to be true from the images on the right hand side of figure 1. Based on the images, we can see that the image for digit 5 start to get less changes from rank 50. This implies that we are able to capture majority of the information in each image with the rank 50 approximation.

4.2 V-modes projection

If we follow the instruction in section 3.3, we should be getting the 3-D plot in figure 2. From the figure, we can see that the points for 0(blue) and points for 9(yellowish green) are mostly separated while the points for 2(yellow) and 5(cyan) are intersecting with each other in the middle of the plot. This means that if we apply the LDA onto the digits of the training data, digit 0 and 9 is probably easier to separate than digit 2 and 5. We should be expecting this characteristic show up when we try to apply the LDA classifier onto the testing set.

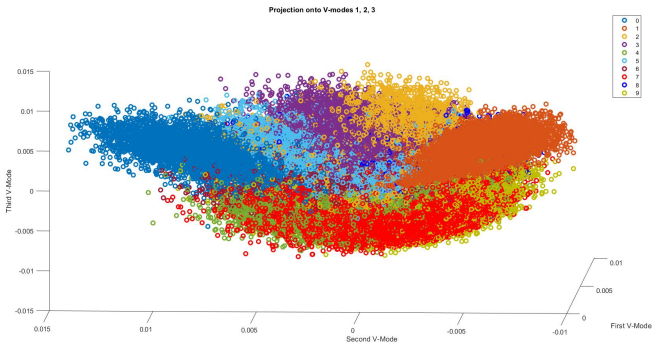
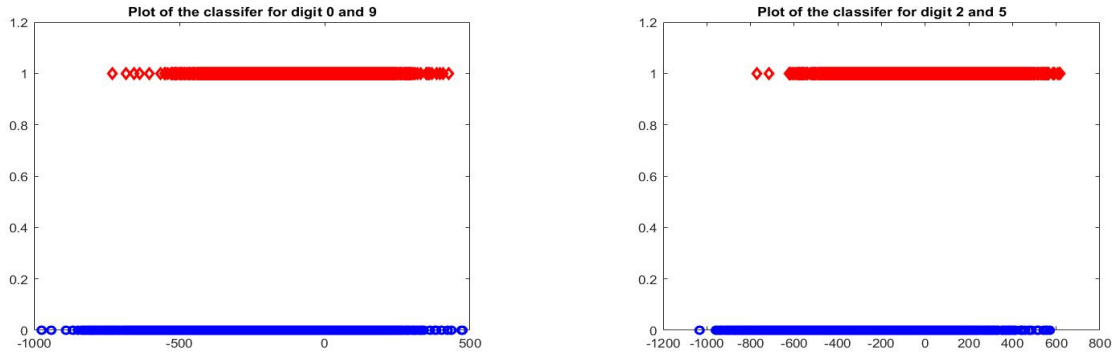


Figure 2: First, second, and third columns of V projection

4.3 Easy 2 digits classifier

Following my prediction in section 3.6, we can set up the classifier for digits 0 and 9 and calculate the accuracy of the classifier on testing data set by using the code on line 70 to 75. Recall that from section 3.4, the **digitLDA** function will create a plot for the points that is projected onto the vector w and for this classifier, we should be getting a plot like the one on the left of figure 3.

We can see that even for two digits that seems to differ by a lot in pixel image, it is still hard to separate two digits using LDA. Furthermore, we can check the accuracy of this classifier on the testing data. With a total of 1989 images of 0 and 9 in the testing data, there are 1014 images that is correctly classified which gives us a success rate of 51%. Moreover, we can apply this classifier onto the training set and if we run the code on line 77 to 81 in appendix B, we will get an accuracy of 50.1%. The amount of images for 0 and 9 in the training set is 11872 while the accuracy is almost the same. This make sense since the accuracy of the classifier shouldn't be depended on the number of items that it is classifying.



(a) Projection of 0 and 9

(b) Projection of 2 and 5

Figure 3: Separating for two digits onto the projection line

4.4 Difficult 2 digits classifier

Recall that we are also going to compute a classifier for digits 2 and 5 which is what I predicted to be hard to separate. By running the code on line 57 to 62 we should be getting the plot like the plot on the right of figure 3. From the plot, we can see that the LDA is still having a hard time to separate the two digits. In order to compare the LDA for the easy digits and this difficult digits, we can calculate the accuracy of this classifier and compare it with the previous classifier. With a total of 1924 images of 0 and 9 in the testing data, there are 906 images that is correctly classified which gives us a success rate of 47%. We can see that the accuracy of this classifier is a bit lower than the accuracy of the classifier for two easy digits

which makes sense since we should expect more images get correctly classified for the digits that is easier to separate. Similarly, we can apply this classifier onto the training set and by running the code on line 64 to 68 in appendix B, we will get an accuracy of 47% which is the same accuracy for the testing set. Again, this make sense since the amount of images shouldn't have an affect on the accuracy.

4.5 SVM and decision tree result

For the SVM classification, I got the following results:

- For the classifier on all 10 digits, out of 10000 images in the testing set, 8554 images are correctly classified which give us an accuracy rate of 85.54%.
- For the classifier for digits 0 and 9, out of 1989 images of 0 and 9 in the testing set, 1971 images are correctly classified which give us an accuracy rate of 99.1%.
- For the classifier for digits 2 and 5, out of 1924 images of 2 and 5, 1838 images are correctly classified which give us an accuracy rate of 95.53%.

We can see that the accuracy for the easy digits is higher than the accuracy for the difficult digits which is what we expected. Moreover, every accuracy is higher than the accuracy from the LDA classifiers.

For the decision tree classifier, I got the following results:

- For the classifier on all 10 digits, out of 10000 images in the testing set, 8492 images are correctly classified which give us an accuracy rate of 84.92%.
- For the classifier for digits 0 and 9, out of 1989 images of 0 and 9 in the testing set, 1957 images are correctly classified which give us an accuracy rate of 98.39%.
- For the classifier for digits 2 and 5, out of 1924 images of 2 and 5, 1856 images are correctly classified which give us an accuracy rate of 96.47%.

Based on these results, we can conclude that SVM and the decision tree classifiers both have significantly better performance than the LDA classifier.

5 Summary and Conclusions

In this assignment, we used three different classification techniques to classify digits from 0 to 9. Moreover, we compared the accuracy of the classifier on two digits that is easy to separate and two digits that is hard to separate. From the result, we can conclude that the LDA classifiers have a much lower accuracy than the classifiers of SVM and decision tree. Furthermore, the SVM and decision tree classifiers for 2 digits perform better than the classifier for all 10 digits which make sense since it is a lot hard to separate all 10 digits.

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation. This is how to make an **unordered** list:

- `B = reshape(A,sz)` reshapes A using the size vector, sz, to define size(B).
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `M = mean(A,dim)` returns the mean along dimension dim.
- `imshow(I)` displays the grayscale image I in a figure.
- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array X.
- `plot3(X,Y,Z)` plots coordinates in 3-D space.

- `[M,I] = max(A)` return a maximum value M in the matrix A and the corresponding index of M.
- `X = zeros(n)` returns an n-by-n matrix of zeros.
- `Mdl = fitcecoc(X,Y)` returns a trained ECOC model using the predictors X and the class labels Y.
- `Mdl = fitcsvm(X,Y)` returns an SVM classifier trained using the predictors in the matrix X and the class labels in vector Y for one-class or two-class classification.
- `tree = fitctree(X,Y)` returns a fitted binary classification decision tree based on the input variables contained in matrix X and output Y.
- `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 discriminant analysis classification model Mdl.

Appendix B MATLAB Code

My MATLAB code for this assignment including the MATLAB function for two digit LDA

```
%%
close all; clear;
%% Load train data
[images1, labels1] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
data1 = [];
for i = 1:60000
    img = reshape(images1(:,:,i),784,1);
    data1 = [data1, img];
end
data1 = double(data1);
%% Part1: SVD
[u,s,v] = svd(data1,'econ');
digits = s*v';
%% Analysis of the singular values
subplot(2,1,1)
plot(diag(s),'ko','Linewidth',2)
subplot(2,1,2)
semilogy(diag(s),'ko','Linewidth',2)
%% Continue
sig = diag(s);
plot(sig.^2/sum(sig.^2),'ko','Linewidth',2)
%% Part 2: approximations
range = 20:5:90;
for i = 1:15
    rank = range(i);
    approx_1 = u(:,1:rank)*s(1:rank,1:rank)*v(:,1:rank)';
    digit1 = reshape(approx_1(:,1),28,28);
    subplot(3,5,i);
    imshow(digit1)
    title(['Rank = ',num2str(rank)]);
end
%% Part 4
colors = [0, 0.4470, 0.7410;0.8500, 0.3250, 0.0980;0.9290, 0.6940, 0.1250;0.4940, 0.1840, 0.5560;0.4660
for label=0:9
    indices = find(labels1 == label);
    plot3(v(indices, 1)', v(indices, 2)', v(indices, 3)', 'o', 'color',colors(label+1,:),'DisplayName',
    hold on
```

```
end
xlabel('First V-Mode'), ylabel('Second V-Mode'), zlabel('Third V-Mode')
title('Projection onto V-modes 1, 2, 3')
legend
%% Two digits LDA
index1 = find(labels1 == 0)';
index2 = find(labels1 == 8)';
digit1 = digits(1:50,index1(1:1000));
digit2 = digits(1:50,index2(1:1000));
[threshold, w,u,s,v] = digitLDA(0, 8, labels1, data1);
%% Load test data
[images2, labels2] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
data2 = [];
for i = 1:10000
    img = reshape(images2(:,:,i),784,1);
    data2 = [data2, img];
end
data2 = double(data2);
%% Hard two digits test
[threshold,w,u,s,v] = digitLDA(2,5,labels1,data1);
test2or5 = find(labels2 == 5 | labels2 == 2);
test_digit = u'*data2;
pval = w'*test_digit(:,test2or5);
total = pval > threshold;
successRate = sum(total)/size(total,2);
%% check the accuracy of the train data
test2or5Train = find(labels1 == 2 | labels1 == 5);
test_digit_Train = u'*data1;
pvalTrain = w'*test_digit_Train(:,test2or5Train);
totalTrain = pvalTrain > threshold;
successRateTrain = sum(totalTrain)/size(totalTrain,2);
%% Easy two digits test
[threshold,w,u,s,v] = digitLDA(0,9,labels1,data1);
test0or9 = find(labels2 == 0 | labels2 == 9);
test_digit = u'*data2;
pval = w'*test_digit(:,test0or9);
total2 = pval > threshold;
successRate2 = sum(total2)/size(total2,2);
%% check the accuracy of the train data
test0or9Train = find(labels1 == 0 | labels1 == 9);
test_digit_Train = u'*data1;
pvalTrain = w'*test_digit_Train(:,test0or9Train);
total2Train = pvalTrain > threshold;
successRate2Train = sum(total2Train)/size(total2Train,2);
%% Three digits LDA
index1 = find(labels1 == 3)';
index2 = find(labels1 == 8)';
digit1 = digits(1:50,index1(1:1000));
digit2 = digits(1:50,index2(1:1000));
mean1 = mean(digit1,2);
mean2 = mean(digit2,2);
index3 = find(labels1 == 6)';
digit3 = digits(1:50,index3(1:1000));
mean3 = mean(digit3,2);
```

```
overall_mean = (mean1+mean2+mean3) / 3;
Sb = 0;
Sb = (mean1 - overall_mean)*(mean1 - overall_mean)' + (mean2 - overall_mean)*(mean2 - overall_mean)' +(
for k = 1:100
   Sw =  Sw + (digit3(:,k) - mean3)*(digit3(:,k) - mean3)';
end
[V2, D] = eig(Sb,Sw);
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
v1 = w'*digit1;
v2 = w'*digit2;
v3 = w'*digit3;
% make v2 bigger than v1
if mean(v1) > mean(v2)
    w = -w;
    v1 = -v1;
    v2 = -v2;
end
% make v3 bigger than v2
if mean(v2) > mean(v3)
    w = -w;
    v2 = -v2;
    v3 = -v3;
end
twos = ones(1000)+1;
p1 = plot(v1,zeros(1000),'ob','Linewidth',2);
hold on
p2 = plot(v2,ones(1000),'dr','Linewidth',2);
p3 = plot(v3,twos,'dk','Linewidth',2);
h = [p1(1),p2(1),p3(1)];
legend(h,'digit 3','digit 8', 'digit 6');
ylim([0 3.2])
%%
digits = [1, 2,3,4,5,6,8,14,16,18];
label = labels1';
for i = 1:10
    approx_1 = u(:,1:50)*s(1:50,1:50)*v(:,1:50)';
    digit1 = reshape(approx_1(:,digits(i)),28,28);
    subplot(2,5,i);
    imshow(digit1)
     title(['Digit is ',num2str(label(digits(i)))]);
end
%% Decision tree for all 10 digits
digitsTree = u(:,1:50)'*data1;
tree = fitctree(digitsTree',labels1);
digitsTest1 = u(:,1:50)'*data2;
testlabelTree1 = predict(tree, digitsTest1');
successrateTree1 = sum(testlabelTree1 == labels2)/10000;
%% Decision tree on two easy digits
indexTree1 = find(labels1 == 0 | labels1 == 9);
digitsTree1 = digitsTree(:,indexTree1);
tree1 = fitctree(digitsTree1', labels1(indexTree1));
indexTestTree1 = find(labels2 == 0 | labels2 == 9);
```

```
digitsTest2 = digitsTest1(:,indexTestTree1);
testlabelTree2 = predict(tree1, digitsTest2');
successrateTree2 = sum(testlabelTree2 == labels2(indexTestTree1))/1989;
%% Decision tree on two difficult digits
indexTree2 = find(labels1 == 2 | labels1 == 5);
digitsTree2 = digitsTree(:,indexTree2);
tree2 = fitctree(digitsTree2', labels1(indexTree2));
indexTestTree2 = find(labels2 == 2 | labels2 == 5);
digitsTest3 = digitsTest1(:,indexTestTree2);
testlabelTree3 = predict(tree2, digitsTest3');
successrateTree3 = sum(testlabelTree3 == labels2(indexTestTree2))/1924;
%% SVM classifier with training data, labels and test set
[u,s,v] = svd(data1,'econ');
digits = u(:,1:50)'*data1;
digits = digits ./ max(digits(:));
Mdl1 = fitcecoc(digits',labels1);
testdigits1 = u(:,1:50)'*data2;
testlabel1 = predict(Mdl1, testdigits1');
successrate = sum(testlabel1 == labels2)/10000;
%% SVM for two difficult digits
index = find(labels1 == 2 | labels1 == 5);
digitsfor2and5 = digits(:,index);
Mdl2 = fitcsvm(digitsfor2and5', labels1(index));
index2 = find(labels2 == 2 | labels2 == 5);
testdigits2 = testdigits1(:, index2);
testlabel2 = predict(Mdl2, testdigits2');
successrate2 = sum(testlabel2 == labels2(index2))/1924;
%% SVM for two easy digits
index = find(labels1 == 0 | labels1 == 9);
digitsfor0and9 = digits(:,index);
Mdl3 = fitcsvm(digitsfor0and9', labels1(index));
index2 = find(labels2 == 0 | labels2 == 9);
testdigits3 = testdigits1(:, index2);
testlabel3 = predict(Mdl3, testdigits3');
successrate3 = sum(testlabel3 == labels2(index2))/1989;

function [threshold,w,u,s,v] = digitLDA(digit1,digit2,labels,data)
    indexofd1 = find(labels == digit1);
    indexofd2 = find(labels == digit2);
    [u,s,v] = svd(data,'econ');
    digits = s*v';
    u = u(:,1:50);
    digit1 = digits(1:50,indexofd1);
    digit2 = digits(1:50,indexofd2);
    mean1 = mean(digit1,2);
    mean2 = mean(digit2,2);
    Sw = 0;
    for k = 1:size(indexofd1,2)
        Sw = Sw + (digit1(:,k) - mean1)*(digit1(:,k) - mean1)';
    end
    for k = 1:size(indexofd2,2)
       Sw =  Sw + (digit2(:,k) - mean2)*(digit2(:,k) - mean2)';
    end
    Sb = (mean1-mean2)*(mean1-mean2)';
```

```
    [V2, D] = eig(Sb,Sw);
    [lambda, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    v1 = w'*digit1;
    v2 = w'*digit2;
    if mean(v1) > mean(v2)
        w = -w;
        v1 = -v1;
        v2 = -v2;
    end
    plot(v1,zeros(size(indexofd1,2)),'ob','Linewidth',2)
    hold on
    plot(v2,ones(size(indexofd2,2)),'dr','Linewidth',2)
    ylim([0 1.2])
    title('Plot of the classifer for digit 0 and 9');
    sortd1 = sort(v1);
    sortd2 = sort(v2);
    t1 = length(sortd1);
    t2 = 1;
    while sortd1(t1) > sortd2(t2)
        t1 = t1 - 1;
        t2 = t2 + 1;
    end
    threshold = (sortd1(t1) + sortd2(t2))/2;
end
```