AMATH 482 Homework 4

Jessica Shi, 1762333

March 6, 2021

Abstract

In this homework, we use the Linear Discriminant Analysis(LDA) to analyze the MNIST data set and do classifications for data with different digit labels. We also use the Support Vector Machines(SVM) and decision tree classifiers and compare the result of classifying on the same data set.

1 Introduction and Overview

In this homework, I have the MNIST data set with training data, testing data and labels. I would first perform an analysis on the data by doing an SVD analysis and project the data into PCA space. Also, I would plot the related information about the singular values and plot the projection onto 3 selected V-modes. Then, I would implement several classifiers to identify data with different labels. I would first use the Linear Discriminant Analysis to classify between 2 labels. Then, similarly, by changing the old method, I would perform LDA to classify 3 different labels. I would try to find the pair of digits that are most difficult and easiest to separate. Besides, I would use the Support Vector Machines and the decision tree classifiers to do the same classification on the data. And I would compare the performance of each classifier on both training and testing sets.

2 Theoretical Background

The fundamental concept used in this homework is the Principal Component Analysis and also the Linear Discriminant Analysis. And they are both derived method based on the Singular Value Decomposition.

2.1 Principal Component Analysis

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

This is the SVD of a matrix $\mathbf{A}$.Note that $\mathbf{V^*}$ is the transpose of $\mathbf{V}$. We would use this to implement the Principal Component Analysis. We introduce the covariance matrix.

$$\mathbf{A} = \frac{1}{\sqrt{n-1}}\mathbf{X} \tag{2}$$

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{XX^*} = \mathbf{AA^*} \tag{3}$$

Also, we know that

$$\mathbf{C_X} = \mathbf{AA^*} = \mathbf{U\Sigma^2U^*} \tag{4}$$

So here, the eigenvalues of the covariance matrix are the squares of the scaled singular values. And recall that we used a change of basis to work in the basis of principal components. To do this, I multiply the data by $\mathbf{U^{-1}} = \mathbf{U^*}$:

$$\mathbf{Y} = \mathbf{U}^{-1}\mathbf{X} \tag{5}$$

The covariance $\mathbf{Y}$ is:

$$\mathbf{C_Y} = \frac{1}{n-1}\mathbf{YY}^T = \frac{1}{n-1}\mathbf{U}^T\mathbf{XX}^T\mathbf{U} = \mathbf{U}^T\mathbf{AA}^T\mathbf{U} = \mathbf{U}^T\mathbf{U}\Sigma^2\mathbf{U}^T\mathbf{U} = \Sigma^2 \tag{6}$$

In the actual implementation, I would use the builtin function `svd(A)` to get the corresponding $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V^*}$ matrices for $\mathbf{A}$.

2.2 Linear Discriminant Analysis

The Linear Discriminant Analysis(LDA) helps us to find a proper chosen subspace to project our data sets on in order to have a clear separation between them. The goal is to find a projection that maximized the distance between the inter-class data while minimizing the intra-class data.

Then we define the between-class scatter matrix:

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{7}$$

and also the within-class scatter matrix:

$$\mathbf{S}_w = \sum_{j=1}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{8}$$

Our goal is to find a vector $\mathbf{w}$ such that:

$$\mathbf{w} = argmax \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}} \tag{9}$$

And it turns out that this vector is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem:

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w} \tag{10}$$

So, we could actually use MATLAB to solve for this vector and find proper projection.

3 Algorithm Implementation and Development

According to the specification, I will divide the implementation section into 2 parts.

3.1 SVD Analysis of Data

First, I load the data into MATLAB using the given `mnist_parse` function to get both the training data and the test data, and their corresponding labels. Then I reshape each image into a column vector and perform the PCA on the training data. I use the `svd` function to get the $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ matrix. The matrix $\mathbf{U}$ represents the modes themselves. The matrix $\mathbf{\Sigma}$ gives us the singular value to calculate the energy in each mode and to determine how many modes are necessary for image reconstruction. And the columns in $\mathbf{V}$ represents projections onto each mode. Lastly, I plot the projection onto three selected $\mathbf{V}$-modes and color the data by their digit label.

3.2 Classifier Implementation

After the basic analysis of the data, I start to implement the classifiers.

First, I implement the linear classifier for 2 classes using LDA. I use the code from lecture 19 and write a function `trainer`. And it works properly for classifying between 2 different labels. Here, I take the label 4 and label 5 as examples. I also write the function `getData` to extract the data with certain labels. After the training process, I plot the histogram for each label and draw a vertical line for the decision-making threshold. From the `trainer` function, I get the proper $\mathbf{U}$ matrix so that I could use it for the projection of test data. Then I apply this matrix to test data and find the final accuracy for the testing process.

Then, I start to implement the linear classifier for 3 classes. Basically, the idea is similar to the 2-class classifier. So, I revise the function `trainer` to get a proper classifier for this. I write a new function `trainer3` to classify among 3 different labels and I also have the function `getThreshold` to get the 2 different thresholds between these 3 groups. Also, I use the function `getData3` to get the data for the 3 selected labels.

Then I use the builtin decision tree classifier and the SVM to separate the data according to the labels as well. I first get the models using the training data and then use the model to get the predicted labels for test data. Next, I calculate the accuracy for both the training and testing process. The detail for the implementation is in the Appendix B.

4 Computational Results

4.1 SVD Analysis Result

By doing the SVD analysis of the data, I get the following plot for the singular value and the corresponding energies.

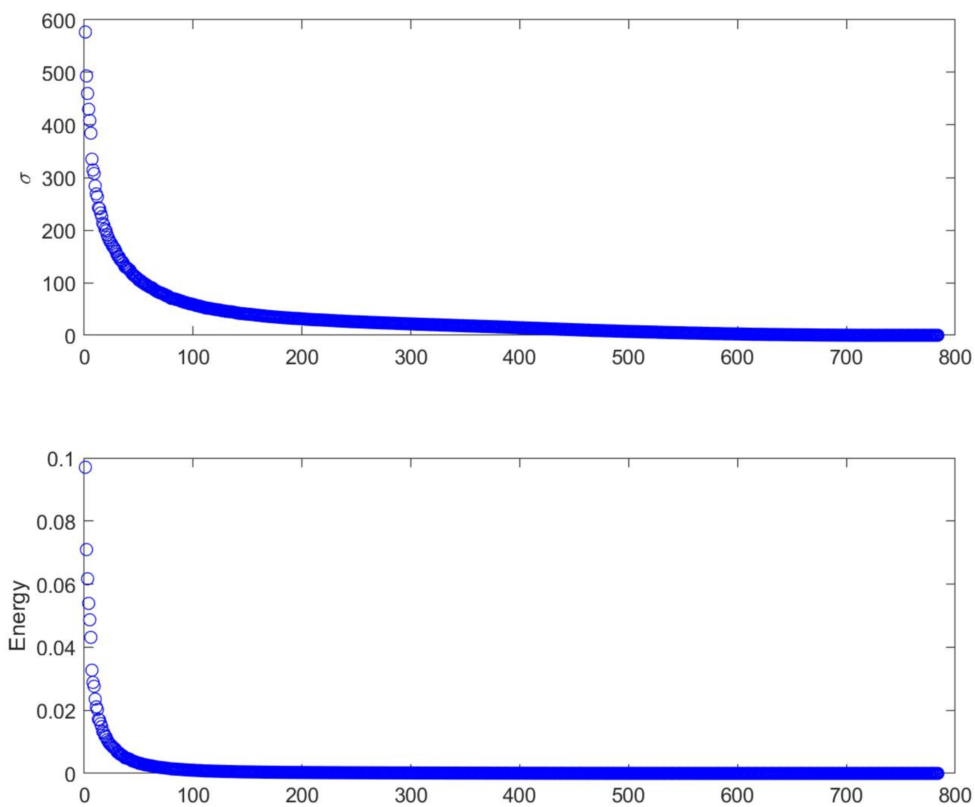

Figure 1: Singular Value and Energy

I then calculate the rank for the digit space. I set the threshold for energy to be 90% and I get a rank of 87. So we could say that 87 modes are needed for the image reconstruction for this data set.

Also, I plot the projection onto 3 selected V-modes, which are column 2, 3 and 5. And I use different colors to represent the data with different labels. Below is the plot.

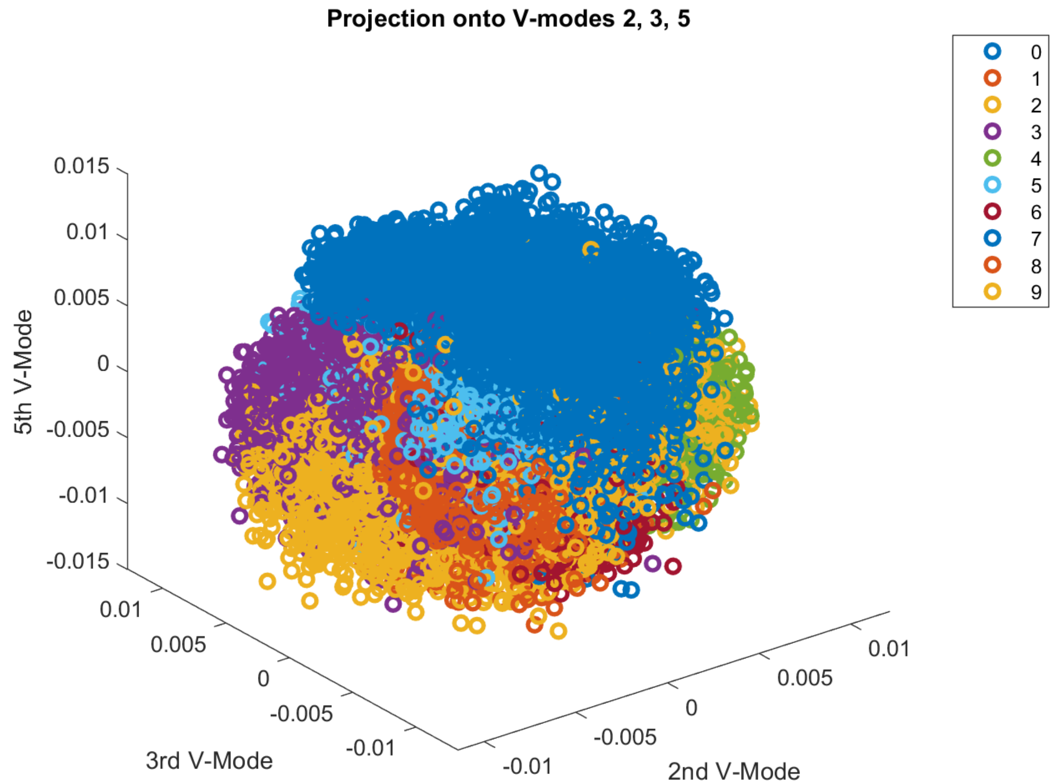

Figure 2: Projection on V-modes

4.2 Classification Result

First, I choose the digit 4 and 5 to do one-time LDA on them. I plot the histogram of the classification result on the train data. And I also calculate the training accuracy, which is 98.37% for label 4 and 98.23% for label 5. We could also see from the Figure 3 that the accuracy should be high.

Then, by applying the **U** matrix to the test data, I also do the classification on test data and get an accuracy of 47.6%. And I loop over all the digits to try different combinations and I found that the highest accuracy appears in the pair of label 0 and label 1, which is 73.00%. And the lowest accuracy appears in the pair of label 2 and label 3, which is 35.31%. This might mean that pair (0,1) would be the easiest to separate while pair (2,3) would be the most difficult to separate.

For the 3 classes classifier, I choose the digit 4, 5 and 9. I plot the classification result histogram, which is shown in Figure 4. While we get a relatively good result for digit 5 and 9, we get undesired result for digit 4 and 9. The accuracy is relatively low. The accuracy on test data is 34.06%. This is reasonable since it is hard to use a linear classifier for 3 classes.

Next, I use the builtin function of decision tree classifier and use the function `predict` to get the prediction of labels for test data and check the accuracy. The highest accuracy appears in the pair of label 1 and label 2, which is 15.60%. And the lowest accuracy appears in the pair of label 0 and label 9, which is 82.40%. This means that pair (1,2) would be the most difficult to separate while pair (0,9) would be the easiest to separate.

Lastly, I use the Support Vector Machine(SVM). By similar approach, first I train on the training set and then use the resulting model for the test data. By looping over all the combinations of digits, I get the accuracy table. According to the table, the highest accuracy appears in the pair of label 1 and label 9, which

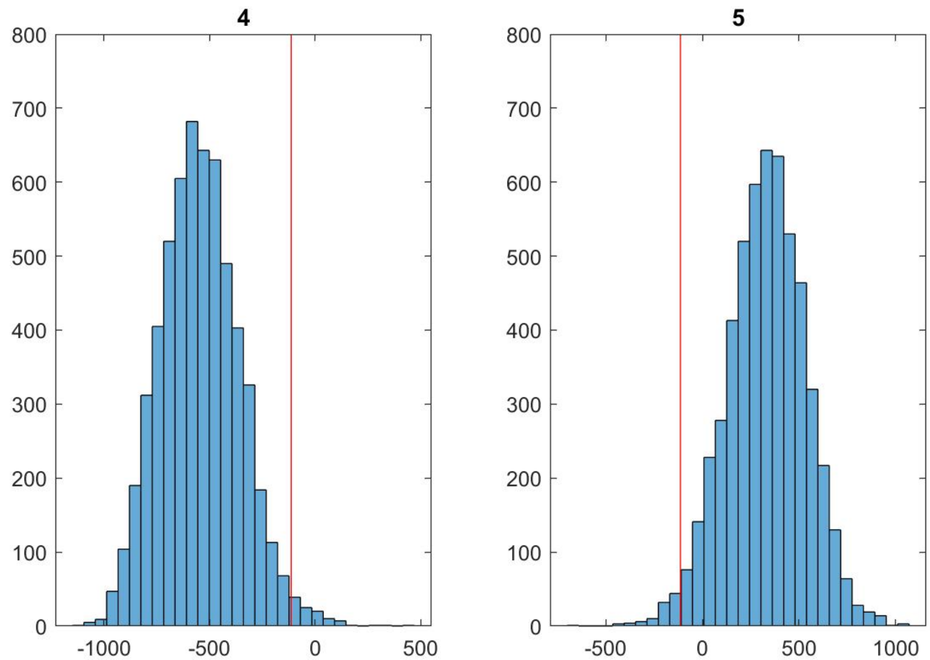

Figure 3: Classification of Train Data between label 4 and 5

is 78.64%. And the lowest accuracy appears in the pair of label 1 and label 2, which is 20.54%. This might mean that pair (1,9) would be the easiest to separate while pair (1,2) would be the most difficult to separate for this classifier.

5 Summary and Conclusions

To conclude, in order to classify images with different labels, we could use Linear Discriminant Analysis as well as other models such as decision tree and Support Vector Machine. The linear classifier is useful for classifying between 2 groups of data, but might be less useful when we have 3 classes. Also, different classifiers or different models might have various performance on the same data set. From the computational result we know that the pair of digits that are easiest and most difficult to separate are not same for LDA, decision tree and SVM. They all perform well on the training accuracy but get an accuracy of about 40-60% for the test data, which are not as good as the training set. And for different pairs of digits, they have diverse performances.

Appendix A MATLAB Functions

- `x = diag(A)` returns a column vector of the main diagonal elements of `A`.
- `plot(X,Y)` creates a 2-D line plot of the data in `Y` versus the corresponding values in `X`.
- `plot3(X,Y,Z)` plots coordinates in 3-D space.
- `[row,col] = find(__)` returns the row and column subscripts of each nonzero element in array `X` using any of the input arguments in previous syntaxes.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix `A`, such that `A = U*S*V'`.

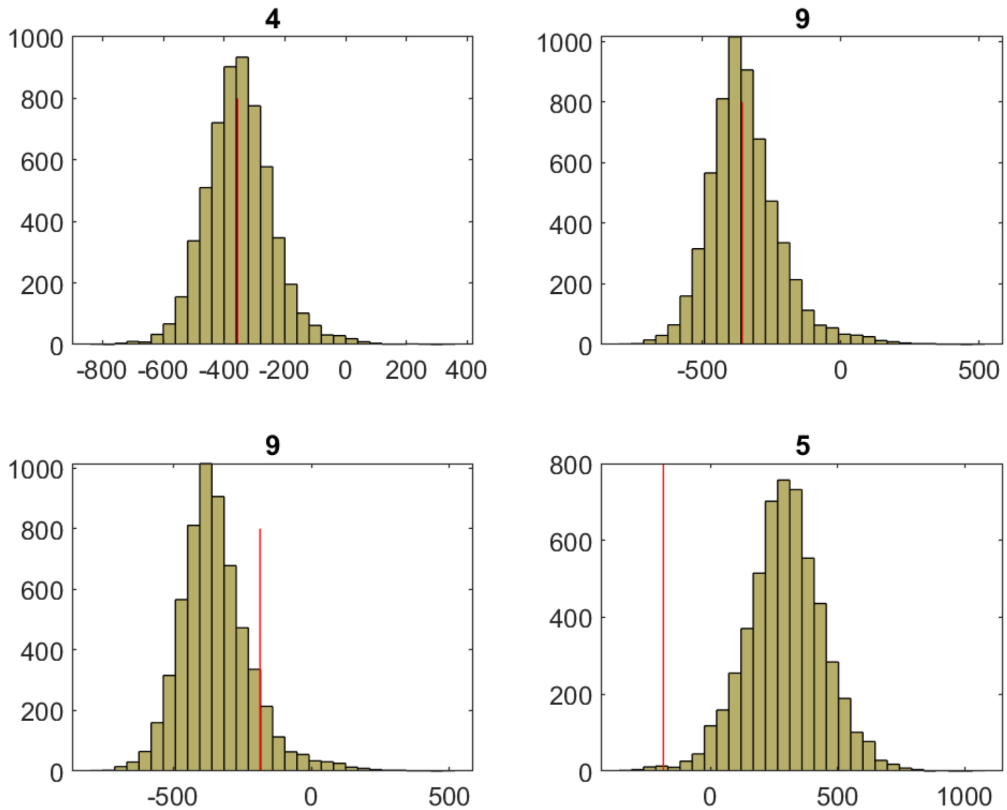

Figure 4: Classification of Train Data between label 4, 5, 9

- `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`.
- `Mdl = fitcsvm(Tbl,Y)` returns an SVM classifier trained using the predictor variables in the table `Tbl` and the class labels in vector `Y`.
- `label = predict(SVMModel,X)` returns a vector of predicted class labels for the predictor data in the table or matrix `X`, based on the trained support vector machine (SVM) classification model `SVMModel`. The trained SVM model can either be full or compact.
- `loss = kfoldLoss(CVMdl)` returns the classification loss obtained by the cross-validated, binary kernel model (ClassificationPartitionedKernel) `CVMdl`. For every fold, `kfoldLoss` computes the classification loss for validation-fold observations using a model trained on training-fold observations. By default, kfoldLoss returns the classification error.

Appendix B MATLAB Code

```
clear all; close all; clc

%% load the training data
% [images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
[train_data, train_label] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
[test_data, test_label] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');
```

```
% SVD analysis
% reshape the train data: each column is an image
train_data = double(reshape(train_data, size(train_data,1)*size(train_data,2), []));
train_label = double(train_label);
test_data = double(reshape(test_data, size(test_data,1)*size(test_data,2), []));
test_label = double(test_label);

% use PCA
[M, N] = size(train_data);
train_data = train_data - repmat(mean(train_data, 2), 1, N);
[U, S, V] = svd(train_data/sqrt(N-1), 'econ');
% [U, S, V] = svd(train_data, 'econ');
sig = diag(S);
proj_data = U' * train_data;

% calculate the rank of the digit space
energy = 0;
total = sum(diag(S).^2);
thre = 0.9;
r = 0;
while energy < thre
    r = r+1;
    energy = energy + (S(r,r).^2)/total;
end

%% plot the singular value and energy
figure(1)
subplot(2, 1, 1)
plot(sig, 'bo', 'Linewidth', 0.5)
ylabel('\sigma')

subplot(2, 1, 2)
plot(sig.^2 / sum(sig.^2), 'bo', 'Linewidth', 0.5)
ylabel('Energy')

%% Projection onto 3 V-modes
% 2, 3, 5 for now
for label=0:9
    label_indices = find(train_label == label);
    plot3(V(label_indices, 2), V(label_indices, 3), V(label_indices, 5),...
        'o', 'DisplayName', sprintf('%i',label), 'Linewidth', 2)
    hold on
end
xlabel('2nd V-Mode'), ylabel('3rd V-Mode'), zlabel('5th V-Mode')
title('Projection onto V-modes 2, 3, 5')
legend
set(gca,'Fontsize', 10)

% -----------------------------------------------------------------------
%% Implement LDA for 2 digits -- 2 selected digits
% get the data with label 4 and 5:
l1 = 4;
l2 = 5;
proj_data = U' * train_data;
[data1, data2, n1, n2, label] = getData(l1, l2, train_label, proj_data);

feature = 87;

[U,S,V,threshold,w,sort1,sort2] = trainer(data1, data2, feature);

% calculate the accuracy for train data
result1 = sort1 > threshold;
result2 = sort2 < threshold;
err1 = size(find(result1==1));
err2 = size(find(result2==1));
acc1 = 1 - err1/size(sort1);
acc2 = 1 - err2/size(sort2);
```

```matlab

% plot the result for train data:
figure(2)
subplot(1,2,1)
histogram(sort1,30); hold on, plot([threshold threshold], [0 800],'r')
%set(gca,'Xlim',[-3 4],'Ylim',[0 10],'Fontsize',14)
title(l1)
subplot(1,2,2)
histogram(sort2,30); hold on, plot([threshold threshold], [0 800],'r')
%set(gca,'Xlim',[-3 4],'Ylim',[0 10],'Fontsize',14)
title(l2)

% check the accuracy for test data-----------------------------------------
% get the test data with label 4 and 5:
[data1, data2, n1, n2, label] = getData(l1, l2, test_label, test_data);
test = [data1 data2];

% change first label to 0, second to 1 to do comparison
label = zeros(n1+n2,1);
label(1:n1) = 0;
label(n1+1:n1+n2) = 1;
TestNum = size(test, 2);
TestMat = U' * test;
pval = w' * TestMat;
ResVec = (pval > threshold);
% 0's are correct, 1s are incorrect
err = abs(ResVec - label');
errNum = sum(err);
accuracy = 1 - errNum/TestNum;
% -------------------------------------------------------------------------
%% Implement LDA for 2 digits: try different combinations:
% get the data with label 4 and 5:
accu_LDA = zeros(10, 10);
for i = 0:9
    for j = i+1:9
        l1 = i;
        l2 = j;

        [data1, data2, n1, n2, label] = getData(l1, l2, train_label, proj_data);

        feature = 87;

        [U,S,V,threshold,w,sort1,sort2] = trainer(data1, data2, feature);

        % calculate the accuracy for train data
        result1 = sort1 > threshold;
        result2 = sort2 < threshold;
        err1 = size(find(result1==1));
        err2 = size(find(result2==1));
        acc1 = 1 - err1/size(sort1);
        acc2 = 1 - err2/size(sort2);

        % check the accuracy for test data-----------------------------------------
        % get the test data
        [data1, data2, n1, n2, label] = getData(l1, l2, test_label, test_data);
        test = [data1 data2];

        % change first label to 0, second to 1 to do comparison
        label = zeros(n1+n2,1);
        label(1:n1) = 0;
        label(n1+1:n1+n2) = 1;
        TestNum = size(test, 2);
        TestMat = U' * test;
        pval = w' * TestMat;
        ResVec = (pval > threshold);
        % 0's are correct, 1s are incorrect
        err = abs(ResVec - label');
        errNum = sum(err);
```

```
        accuracy = 1 - errNum/TestNum;
        accu_LDA(i+1,j+1) = accuracy;
    end
end

%% Implement LDA for 3 digits: (select 4,5,9 as example)
l1 = 4;
l2 = 5;
l3 = 9;
[data1, data2, data3, n1, n2, n3, label] = getData3(l1, l2, l3, train_label, proj_data);
feature = 87;

[U, S, V, w, sort1, sort2, sort3] = trainer3(data1, data2, data3, feature);

threshold1 = getThreshold(sort2, sort3);
threshold2 = getThreshold(sort3, sort1);


% plot the result for train data:
figure(2)
subplot(2,2,1)
histogram(sort1,30); hold on, plot([threshold2 threshold2], [0 800],'r')
title(l1)
subplot(2,2,2)
histogram(sort3,30); hold on, plot([threshold2 threshold2], [0 800],'r')
title(l3)
subplot(2,2,3)
histogram(sort3,30); hold on, plot([threshold1 threshold1], [0 800],'r')
title(l3)
subplot(2,2,4)
histogram(sort2,30); hold on, plot([threshold1 threshold1], [0 800],'r')
title(l2)

% check the accuracy for test data-----------------------------------------
% get the test data with label 4 and 5:
[data1, data2, data3, n1, n2, n3, label] = getData3(l1, l2, l3, test_label, test_data);
test = [data1 data2 data3];

% change first label to 0, second to 1 to do comparison
TestNum = size(test, 2);
TestMat = U' * test;
pval = w' * TestMat;
ResVec = zeros(1, TestNum);
for i = 1:TestNum
    if pval(i) > threshold2
        ResVec(i) = l1;  % label1
    elseif pval(i) < threshold1
        ResVec(i) = l2;  % label2
    else
        ResVec(i) = l3;
    end
end
err_num = 0;
for i = 1:TestNum
    if (ResVec(i) ≠ label(i))
        err_num = err_num + 1;
    end
end
sucRate = 1 - err_num/TestNum;

%% decision tree: different combination of digits
% choose data with labels 4 & 5:
accu_DecTree = zeros(10,10);
for i = 0:9
    for j = i+1:9
        l1 = i;
        l2 = j;
        [data1, data2, n1, n2, label] = getData(l1, l2, train_label, proj_data);
```

```
        data = [data1 data2];
        tree = fitctree(data', label, 'CrossVal', 'On');
        % view(tree.Trained{1}, 'Mode', 'graph');
        classError = kfoldLoss(tree);

        [data1, data2, n1, n2, label] = getData(l1, l2, test_label, test_data);
        test = [data1 data2];
        test_labels = predict(tree.Trained{1}, test');
        accu = sum(test_labels == label)/length(label);
        accu_DecTree(i+1, j+1) = accu;
    end
end

%% SVM classifier: different combination of digits
% multiplying SV by the inverse of the largest/first singular value.
% choose data with labels 4 & 5:
accu_SVM = zeros(10,10);
for i = 0:9
    for j = i+1:9
        l1 = i;
        l2 = j;
        [data1, data2, n1, n2, label] = getData(l1, l2, train_label, proj_data);
        data = [data1 data2];
        data = data ./ max(data(:));
        Mdl = fitcsvm(data', label);
        [data1, data2, n1, n2, label] = getData(l1, l2, test_label, test_data);
        test = [data1 data2];
        test_labels = predict(Mdl, test');

        % 0 false; 1 true
        check = label==test_labels;
        err = size(find(check==0), 1);
        accu = 1 - err/size(test_labels, 1);
        accu_SVM(i+1,j+1) = accu;
    end
end
```

```
function [data1, data2, n1, n2, label] = getData(label1, label2, labels, data)
    ind1 = find(labels==label1);
    ind2 = find(labels==label2);
    n1 = size(ind1, 1);
    n2 = size(ind2, 1);
    data1 = zeros(784, n1);
    data2 = zeros(784, n2);
    for i = 1:n1
        ind = ind1(i);
        data1(:,i) = data(:, ind);
    end

    for i = 1:n2
        ind = ind2(i);
        data2(:,i) = data(:, ind);
    end
    label = zeros(n1+n2,1);
    label(1:n1) = label1;
    label(n1+1:n1+n2) = label2;

end
```

```
function [data1, data2, data3, n1, n2, n3, label] = getData3(label1, label2, label3, ...
    labels, data)
    ind1 = find(labels==label1);
    ind2 = find(labels==label2);
    ind3 = find(labels==label3);
```

```
    n1 = size(ind1, 1);
    n2 = size(ind2, 1);
    n3 = size(ind3, 1);
    data1 = zeros(784, n1);
    data2 = zeros(784, n2);
    data3 = zeros(784, n3);
    for i = 1:n1
        ind = ind1(i);
        data1(:,i) = data(:, ind);
    end

    for i = 1:n2
        ind = ind2(i);
        data2(:,i) = data(:, ind);
    end

    for i = 1:n3
        ind = ind3(i);
        data3(:,i) = data(:, ind);
    end
    label = zeros(n1+n2+n3,1);
    label(1:n1) = label1;
    label(n1+1:n1+n2) = label2;
    label(n1+n2+1:n1+n2+n3) = label3;

end
```

```
function [U, S, V, threshold, w, sort1, sort2] = trainer(data1, data2, feature)

    n1 = size(data1, 2);
    n2 = size(data2, 2);
    [U, S, V] = svd([data1 data2], 'econ');
    % projection onto principal componenet: X = USV' -> U'X = SV'
    proj = S*V';
    U = U(:, 1:feature);
    d1 = proj(1:feature, 1:n1);
    d2 = proj(1:feature, n1+1:n1+n2);
    m1 = mean(d1, 2);
    m2 = mean(d2, 2);

    Sw = 0;

    for k = 1:n1
        Sw = Sw + (d1(:,k)-m1)*(d1(:,k) - m1)';
    end
    for k = 1:n2
        Sw = Sw + (d2(:,k)-m2)*(d2(:,k) - m2)';
    end
    Sb = (m1 - m2)*(m1 - m2)';

    [V2, D] = eig(Sb, Sw);
    [lambda, ind] = max(abs(diag(D)));
    w = V2(:, ind);
    w = w/norm(w,2);
    v1 = w' * d1;
    v2 = w' * d2;

    if mean(v1) > mean(v2)
        w = -w;
        v1 = -v1;
        v2 = -v2;
    end

    sort1 = sort(v1);
    sort2 = sort(v2);
    t1 = length(sort1);
```

```
    t2 = 1;
    while sort1(t1) > sort2(t2)
        t1 = t1-1;
        t2 = t2+1;
    end

    threshold = (sort1(t1) + sort2(t2))/2;
end
```

```
function [U, S, V, w, sort1, sort2, sort3] = trainer3(data1, data2, data3, feature)

    n1 = size(data1, 2);
    n2 = size(data2, 2);
    n3 = size(data3, 2);
    [U, S, V] = svd([data1 data2 data3], 'econ');
    % projection onto principal componenet: X = USV' -> U'X = SV'
    proj = S*V';
    U = U(:, 1:feature);
    d1 = proj(1:feature, 1:n1);
    d2 = proj(1:feature, n1+1:n1+n2);
    d3 = proj(1:feature, n1+n2+1:n1+n2+n3);
    m1 = mean(d1, 2);
    m2 = mean(d2, 2);
    m3 = mean(d3, 2);
    mu = mean([m1 m2 m3], 2);

    % Calculate within-class matrix
    Sw = 0;
    for k = 1:n1
        Sw = Sw + (d1(:,k)-m1)*(d1(:,k) - m1)';
    end
    for k = 1:n2
        Sw = Sw + (d2(:,k)-m2)*(d2(:,k) - m2)';
    end
    for k = 1:n3
        Sw = Sw + (d3(:,k)-m3)*(d3(:,k) - m3)';
    end

    % Calculate between-class matrix
    Sb = (mu - m1)*(mu - m1)' + (mu - m2)*(mu - m2)' +(mu - m3)*(mu - m3)';

    [V2, D] = eig(Sb, Sw);
    [lambda, ind] = max(abs(diag(D)));
    w = V2(:, ind);
    w = w/norm(w,2);
    v1 = w' * d1;
    v2 = w' * d2;
    v3 = w' * d3;

    sort1 = sort(v1);
    sort2 = sort(v2);
    sort3 = sort(v3);

end
```

```
function threshold = getThreshold(sort1, sort2)
    t1 = length(sort1);
    t2 = 1;
    while sort1(t1) > sort2(t2)
        t1 = t1-1;
        t2 = t2+1;
    end
    threshold = (sort1(t1) + sort2(t2))/2;
end
```

```
function [images, labels] = mnist_parse(path_to_digits, path_to_labels)

% The function is curtesy of stackoverflow user rayryeng from Sept. 20,
% 2016. Link: ...
    https://stackoverflow.com/questions/39580926/how-do-i-load-in-the-mnist-digits-and-label-data-in-matlab

% Open files
fid1 = fopen(path_to_digits, 'r');

% The labels file
fid2 = fopen(path_to_labels, 'r');

% Read in magic numbers for both files
A = fread(fid1, 1, 'uint32');
magicNumber1 = swapbytes(uint32(A)); % Should be 2051
fprintf('Magic Number - Images: %d\n', magicNumber1);

A = fread(fid2, 1, 'uint32');
magicNumber2 = swapbytes(uint32(A)); % Should be 2049
fprintf('Magic Number - Labels: %d\n', magicNumber2);

% Read in total number of images
% Ensure that this number matches with the labels file
A = fread(fid1, 1, 'uint32');
totalImages = swapbytes(uint32(A));
A = fread(fid2, 1, 'uint32');
if totalImages ≠ swapbytes(uint32(A))
    error('Total number of images read from images and labels files are not the same');
end
fprintf('Total number of images: %d\n', totalImages);

% Read in number of rows
A = fread(fid1, 1, 'uint32');
numRows = swapbytes(uint32(A));

% Read in number of columns
A = fread(fid1, 1, 'uint32');
numCols = swapbytes(uint32(A));

fprintf('Dimensions of each digit: %d x %d\n', numRows, numCols);

% For each image, store into an individual slice
images = zeros(numRows, numCols, totalImages, 'uint8');
for k = 1 : totalImages
    % Read in numRows*numCols pixels at a time
    A = fread(fid1, numRows*numCols, 'uint8');

    % Reshape so that it becomes a matrix
    % We are actually reading this in column major format
    % so we need to transpose this at the end
    images(:,:,k) = reshape(uint8(A), numCols, numRows).';
end

% Read in the labels
labels = fread(fid2, totalImages, 'uint8');

% Close the files
fclose(fid1);
fclose(fid2);

end
```