Classifying digits with Linear Discriminant Analysis

Crystal Yang

ABSTRACT

In this article, models are trained using Linear Discriminant Analysis (LDA), decision tree and support vector machines (SVM) to be able to classify the different digits from the images that contain the handwritten digits. These methods are implemented in MATLAB.

INTRODUCTION

MNIST dataset is a dataset that contains 70,000 pictures, each picture contains one digit in handwritten, and the digits it contains are from 0 to 9. It got split into two sets: one set contains 60,000 images for training the model, and 10,000 for testing the model. Our task is to use Singular Value Decomposition to recognize the most important features and use the SVD result to the LDA to allows LDA explicitly separate classes of objects (different digits), so by 'looking' at the image of the digit, the model should be able to tell which digit it is.

BACKGROUND THEOREM

Singular Value Decomposition is an analysis tool that decomposes a matrix that contains high-dimensional data into three component matrices: U, S, V where U matrix is the left singular vector matrix of the original matrix and it contains the direction of the projection of the variable; the S matrix is a diagonal matrix that contains the variance explained by each variable, and V matrix is the matrix that contains vectors of projection and we can use V to reconstruct the original matrix, this matrix is also the right singular vector matrix of the original matrix. Thus, from SVD, we can extract the most important features in the dataset according to the singular values matrix S and represent the system(dataset) using U and V matrix.

Linear Discriminant Analysis is closely related to the SVD above because it make uses of the importance features SVD find to implement a threshold between two classes of objects and use this threshold to classify between the two classes, this is also called Linear classification, that is, we draw a line between two groups of points, and the data points that are to the left of the line is classified as a group whereas the data points over the other side of the line got recognize as the other group. We will use this method to train our LDA model in order to recognize between two different digits. So, how can we find this 'line' and treat it as our classifier, and also find the right dimensions to project it onto?

Suppose we are trying to classify between two groups of objects, and their group mean for each of the features are μ_1 and μ_2(μ is a column vectors since it contains mean across each row), we can define the between-class scatter matrix as:

$$S_B = (\mu_2 - \mu_1) * (\mu_2 - \mu_1)^T$$

This measures the variance between groups, now we also need the within-class scatter matrix:

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

These two variance matrices help us to find a vector w such that:

$$w \;=\; argmax \frac{w^T S_B w}{w^T S_w w}$$

and we can show that this w that maximize the above function is the eigenvector of:

$$S_B w \;=\; \lambda S_w w$$

This w vector that we found will be able to help us to find a threshold between two classes.

ALGORITHM IMPLEMENTATION & DEVELOPMENT

After loading the MNIST dataset into MATLAB, first we are interested in the SVD analysis of the dataset. I have reshaped the dataset so that the column represents sample digits, and the row of the dataset corresponds to the features of the image. The training dataset contains 784 rows and 60,000 columns where the 784 is the 784 features of every digit image, and 60,000 is the digit images, and for test dataset the dimension is 784x10,000. By using the `svd(X,'econ')` command in MATLAB, we will be able to get a reduced SVD matrices from the training dataset, and we are going to use this U,S,V decomposed matrix in training our data.

To project the selected V mode onto a 3D plot, I random sampled 2000 images from the 60,000 training images using `randsample()` in MTALAB, and extract the needed column from the U matrix, then got the projection space by multiplication between U transpose and the training dataset.

Identifying between 2 and 3 digits using LDA

First, I want to build a linear classifier by using LDA between any two digits, the two digits that I picked is 2 and 3. To increase the runtime, I still pick 2000 out of the 60,000 images by using random sample, and I used the first 200 features to build my classifier, I choose to use the first 200 features because when summing up the energy of the first 200 singular value, the explained variance is 97% which is sufficient for us to draw the conclusion that the first 200 modes are enough to represent the dataset. Then, I got the projection space by S*V', so that I can start to pick the images that contains 2 and 3. I followed the previous implementation for defining the two group means, between-class variance matrix and the within-class variance matrix. To find w vector, I used `eig()` command, and w will be the max among the eigenvalue. Then, the transpose of w times the projection space matrix of the two group will give me the groups of value that I can used to build a classifier between. After I found these two groups of value, the algorithm for me to build a classifier between is that I sorted these two groups, and I got two pointers, one points to the first value in group 1, the other points to the last value of group 2, and I start to compare these two values and move the pointer until I find a place that the value of group 1 is larger than the value of group 2. This value will be my classifier to separate the two groups.

To identify between three digits, the implementation is the same as above, it is just when we are calculating the between-group variance matrix, we need to subtract from the overall mean the three group means. Then we can find two classifiers to build the threshold for identifying between three group.

For validation using the test data, I grab the images that contains 2 and 3 from the test dataset and multiply the test dataset with the transpose of U on the left to get the PCA projection space matrix, then multiply this matrix with the transpose of w on the left to get a vector of the value that we can compared with our threshold to distinguish its class (whether it is digit 2 or 3). The correct rate is calculated by comparing the prediction between the test data labels.

I use nested for loop to build a classifier between every pair of digits to find the pair of digits that are the easiest and hardest to separate.

Decision tree classifier and SVM

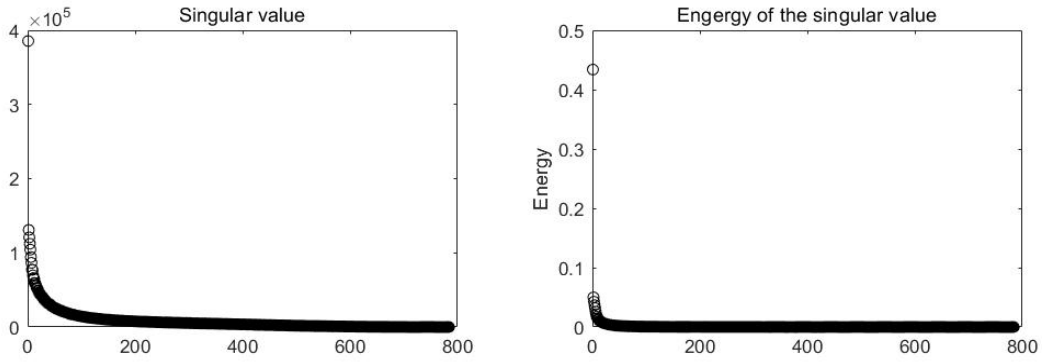
To build a decision tree classifier between these ten digits, I use the MATLAB built-in command `fitctree(X, y, 'MaxNumSplits',10,'CrossVal','on')` to built a classifier for the 10 digits dataset, and the prediction error in the training set can be viewed by using `kfoldLoss()` in MATLAB, the performance of testing set is validated by using `predict()`.

For SVM, first I find the images that contains the two digits that I to build a classifier for, and then use `fitcsvm()` command in MATLAB to find the classifier, and use `predict()` command to find the correct rate of prediction. I used nested for loop to build a classifier between every pair of digits.

COMPUTATIONAL RESULTS

After we perform SVD on the training dataset, we have to be clear about the interpretation of the three decomposed matrices: The U matrix is the matrix that contains the 'eigendigits' of the original data matrix that each column of the U matrix specify the direction of the most important variable; The S matrix contains the variances that explained by each mode, and the V matrix contains coefficient of the linear combination to reconstruct the original matrix. Thus, we can make U and V to construct the PCA projection space.

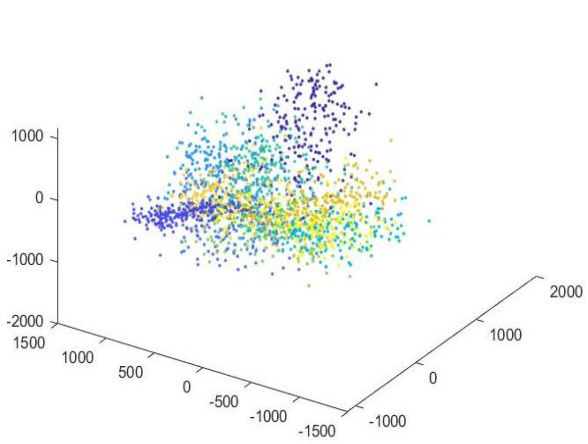
Now let's look at the singular value spectrum from the SVD result of the training dataset:



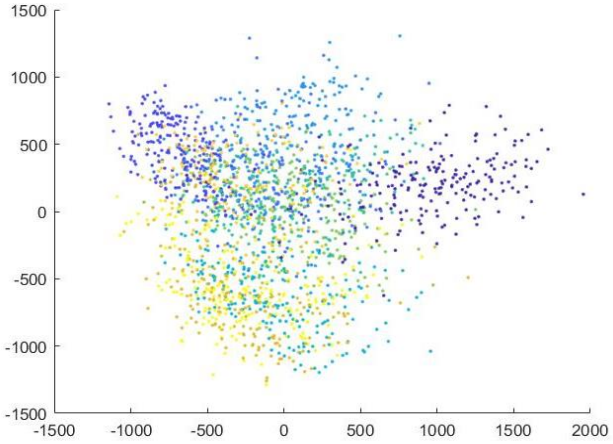
Singular value spectrum of MNIST dataset

We can see that there is a fast-decreasing trend in the singular value, and the singular value drops to zero after the 200 modes, thus, I choose to use the first 200 modes to represent the dataset, and we can see that it is sufficient.

Next, picking the mode 2,3,5 from the U matrix, I want to project the data onto these three modes in 3D plot:



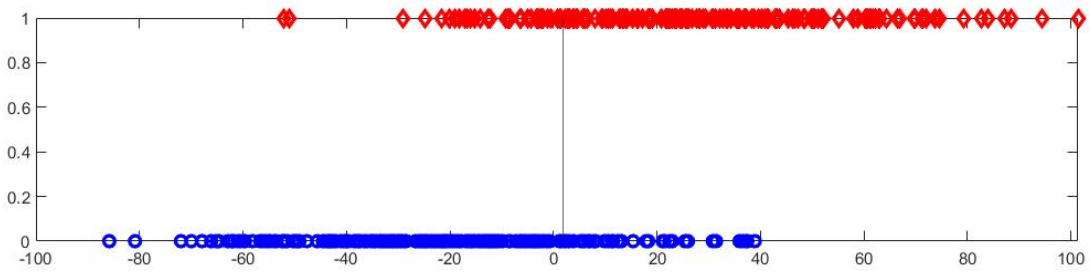
Projection of 3 V-modes (looking from side)



Projection of 3 V-modes (looking from above)

The data points in the above graph are colored differently according to their label, so the image with the same label got the same color, from the above graph, we can see that there are some clustering existed when we project the data onto 3 features, and this is our start point of building the linear classifier.

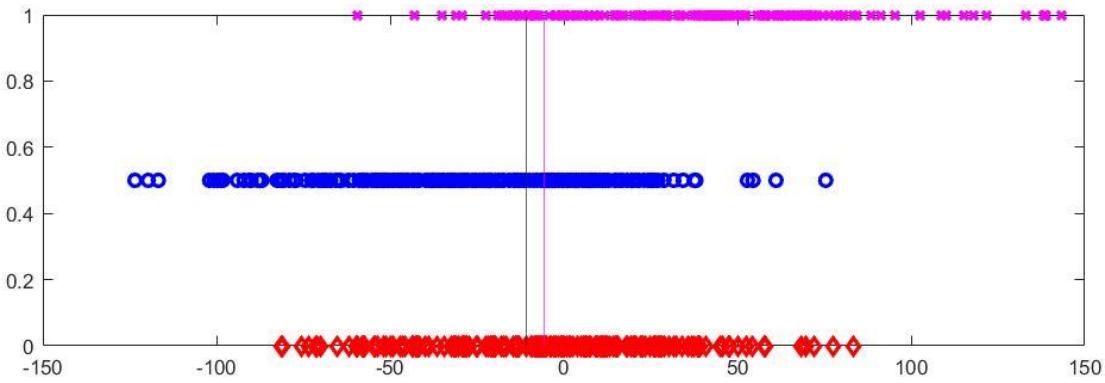
Building a linear classifier using LDA between two digits 2 and 3 results in finding a threshold looks like this:



Threshold between two digits

The blue dots correspond to all the data points that are 2, whereas the red diamonds correspond to all the data points that are 3. We can see there is a vertical line that splits these two groups of data and although this threshold is not perfect because we can see that there are still datapoints that are 3 but be miss classified as 2, it draws the line where splits between the most data points of different groups.

Building linear classifier between three groups of data is very different:



Threshold between 3 digits

The magenta points correspond to the digit 1, and the blue points represents the digit 2, and the red diamonds represent the digit 3. It is very hard to draw the line between them since we have three variables here, we have two thresholds. The first one is the magenta line, to the left of the line the data will be identify as 2 and to the right will be 1. The black line draws the difference between digit 2 and 3, to the left will be identify as 2 whereas to the right will be identify as 3. Now identifying digit 3 will be tricky because in order to not to mix digit 3 and 1, then we treat the data points that have a value between the two thresholds to be digit 3, this results in a lot of digit 3 miss classified as digit 1 or 2.

Using for loop to build classifier between every pair of two digits and the performance of LDA on both training and testing data are:

-	0	1	2	3	4	5	6	7	8	9
0	0	0.6274	0.6317	0.6337	0.6182	0.6522	0.6109	0.6330	0.6249	0.6159
1	0.6274	0	0.6248	0.6256	0.6316	0.6334	0.6331	0.6399	0.6264	0.6203
2	0.6317	0.6248	0	0.6401	0.6261	0.6305	0.6201	0.6204	0.6386	0.6227
3	0.6337	0.6256	0.6401	0	0.6355	0.6393	0.6372	0.6408	0.6361	0.6295
4	0.6182	0.6316	0.6261	0.6355	0	0.6291	0.6216	0.6219	0.6278	0.6233
5	0.6522	0.6334	0.6305	0.6393	0.6291	0	0.6216	0.6286	0.6324	0.6307
6	0.6109	0.6331	0.6201	0.6372	0.6216	0.6216	0	0.6208	0.6284	0.5933
7	0.6330	0.6399	0.6204	0.6408	0.6219	0.6286	0.6208	0	0.6194	0.6220
8	0.6249	0.6264	0.6386	0.6361	0.6278	0.6324	0.6284	0.6194	0	0.6188
9	0.6159	0.6203	0.6227	0.6295	0.6233	0.6307	0.5933	0.6220	0.6188	0

LDA performance on training data

-	0	1	2	3	4	5	6	7	8	9
0	0	0.6435	0.4756	0.5085	0.5673	0.5721	0.4546	0.5896	0.5420	0.5018
2	0.6435	0	0.4555	0.4340	0.5243	0.4396	0.5485	0.6181	0.5130	0.5620
2	0.4756	0.4555	0	0.4701	0.4707	0.4511	0.3889	0.5000	0.4826	0.4875
3	0.5085	0.4340	0.4701	0	0.5060	0.4695	0.4619	0.4750	0.4057	0.4453
4	0.5673	0.5243	0.4707	0.5060	0	0.4402	0.4964	0.5493	0.5399	0.5384
5	0.5721	0.4396	0.4511	0.4695	0.4402	0	0.4778	0.4714	0.5155	0.4971
6	0.4546	0.5485	0.3889	0.4619	0.4964	0.4778	0	0.5081	0.4881	0.4525
7	0.5896	0.6181	0.5000	0.4750	0.5493	0.4714	0.5081	0	0.5395	0.5169
8	0.5420	0.5130	0.4826	0.4057	0.5399	0.5155	0.4881	0.5395	0	0.4992
9	0.5018	0.5620	0.4875	0.4453	0.5384	0.4971	0.4525	0.5169	0.4992	0

LDA performance on testing data

The first row and column represent the digit we are comparing, for example, the cell (2,3) represents the correct rate of identifying between digit 2 and 3, so this is a symmetric matrix since the result of classifying between 2 and 3 is the same as classifying between 3 and 2. We can see from the table that LDA model is doing overall well on training data than the test data, and this is expected since the model is 'trained' using the training data so it meant to work well on training data. From the testing data result, we can conclude that the easiest digits to be separated is (0,1) with a successful classifying rate of 64.35%, and the hardest pair of digits to be separated is (2,6) with a classifying rate of 38.89%.

With these two pair of digits, we are also interested in the performance on identifying (0,1) and (2,6) by using SVM and classification tree. By running SVM and classification tree on these two pairs, we can get the following results:

	Dataset performed	Classifying (0,1)	Classifying (2,6)
LDA	Train	62.74%	62.01%
	Test	64.35%	38.89%
Classification tree	Train	56.90%	56.90%
	Test	57.72%	57.72%
SVM	Train	100%	100%
	Test	99.91%	97.79%

From this table, we can say that when identifying between (0,1) and (2,6), SVM got the highest accuracy rate whereas the classification tree got the lowest accuracy. SVM is the relatively good model compared with LDA and classification, we can see this from performing SVM on every single pair of digits, the successful rate are:

-	0	1	2	3	4	5	6	7	8	9
0	0	0.9991	0.9836	0.9925	0.9949	0.9786	0.9840	0.9955	0.9893	0.9915
1	0.9991	0	0.9862	0.9916	0.9962	0.9906	0.9976	0.9917	0.9815	0.9944
2	0.9836	0.9862	0	0.8908	0.9702	0.9688	0.9779	0.9670	0.9088	0.9809
3	0.9925	0.9916	0.8908	0	0.9935	0.8386	0.9939	0.9764	0.8327	0.9747
4	0.9949	0.9962	0.9702	0.9935	0	0.9893	0.9902	0.9831	0.9913	0.8795
5	0.9786	0.9906	0.9688	0.8386	0.9893	0	0.9616	0.9865	0.8730	0.9774
6	0.9840	0.9976	0.9779	0.9939	0.9902	0.9616	0	0.9985	0.9855	0.9959
7	0.9955	0.9917	0.9670	0.9764	0.9831	0.9865	0.9985	0	0.9885	0.7653
8	0.9893	0.9815	0.9088	0.8327	0.9913	0.8730	0.9855	0.9880	0	0.9657
9	0.9915	0.9944	0.9809	0.9747	0.8795	0.9774	0.9959	0.7653	0.9657	0

CONCLUSION

Building classification between two classes can be done by using Linear Discriminant Analysis, and we are making use of the fact that the SVD can give us the most important principal component in the data set. But the performance of LDA may not be the best among other classification methods like SVM and classification tree.

Appendix. A MATLAB functions used and brief implementation explanation

`Randsample(n,k)`: returns k integer samples uniformly from the range of 1 to n.

`Fitctree(X, label)`: fits a binary classification decision tree based on the training sample X and the correct labels.

`Fitcsvm(X, label)`: fits a support vector machine classifier based on the training sample X and the correct labels.

`Loss(X, Test, Testlb)`: returns the incorrect rate of the model X on predicting the label of the Test dataset given the correct test labels.

`Predict(X, Test)`: returns the predicted labels using the classification model X.

`Find(X)`: returns the vector of indices that satisfy some given conditions in the dataset X.

`Repelem(v,n)`: return n copies of the given vector v.

Appendix. B MATLAB Code

```
clear all; close all; clc
% Extract the needed data
gunzip('train-labels-idx1-ubyte.gz');
gunzip('train-images-idx3-ubyte.gz');
gunzip('t10k-images-idx3-ubyte.gz');
gunzip('t10k-labels-idx1-ubyte.gz');
[train_images, train_labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-
idx1-ubyte');
[test_images, test_labels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-
ubyte');

%% Reshape
train_images = double(reshape(train_images,
size(train_images,1)*size(train_images,2), []).');
train_labels = double(train_labels);
%% SVD
% feature/mode = row
% image on column, 1 column is the image of digit
trainingdata = transpose(train_images);
traininglb = transpose(train_labels);
[U,S,V] = svd(trainingdata,'econ');

%% rank of digit space equals to the non zero singular value
subplot(1,2,1)
plot(diag(S),'ko')
title("Singular value")
r = rank(S);
sig = diag(S);
subplot(1,2,2)
plot(sig.^2/sum(sig.^2),'ko')
title("Engergy of the singular value")
ylabel('Energy')
%% Calculate energy of the first 200 features
energy = sig.^2/sum(sig.^2);
sum(energy(1:200))

%% Selected V-mode 2,3,5
% random sample of the data -to be less to show
ind = randsample(60000,2000);
l = traininglb(ind); % label
cmap = colormap(parula(10));
proj(:,1) = U(:,2);
proj(:,2) = U(:,3);
proj(:,3) = U(:,5);
m = proj'*trainingdata(:,ind);
for j = 1:10
    plot3(m(1,find(l==(j-1))),m(2,find(l==(j-1))),m(3,find(l==(j-
1))),'.','Color',cmap(j,:)), hold on
end
```

```
%% LDA of 2 digits(group)
% random sample 2000 out of the 60000
ind = randsample(60000,2000);
l = traininglb(ind); % label
feature = 200; % features we used
digits = S*V';
ind_2 = find(l == 2);
ind_3 = find(l == 3);
digit_2 = digits(1:feature,ind_2);
digit_3 = digits(1:feature,ind_3);
m2 = mean(digit_2,2);
m3 = mean(digit_3,2);
% within-class variance
Sw = 0;
for k = 1:length(ind_2)
    Sw = Sw + (digit_2(:,k) - m2)*(digit_2(:,k) - m2)';
end
for k = 1:length(ind_3)
    Sw = Sw + (digit_3(:,k) - m3)*(digit_3(:,k) - m3)';
end
% between class
Sb = (m2-m3)*(m2-m3)';

[V2, D] = eig(Sb,Sw);
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

v2 = w'*digit_2;
v3 = w'*digit_3;

if mean(v2)>mean(v3)
    w = -w;
    v2 = -v2;
    v3 = -v3;
end

plot(v2,zeros(length(ind_2)),'ob','Linewidth',2), hold on
plot(v3,ones(length(ind_3)),'dr','Linewidth',2)

sort2 = sort(v2);
sort3 = sort(v3);
t1 = length(sort2);
t2 = 1;
while sort2(t1)>sort3(t2) % compare the last to the first
    t1 = t1 - 1;
    t2 = t2 + 1;
end

threshold = (sort2(t1) + sort3(t2))/2;
xline(threshold)
miss_classify = length(find(v2 > threshold));
miss_classify = miss_classify + length(find(v3 < threshold));
```

```
%% LDA of 3 digits
% random sample 2000 out of the 60000
ind = randsample(60000,2000);
l = traininglb(ind); % label
feature = 200; % features we used
digits = S*V';
ind_1 = find(l == 1);
ind_2 = find(l == 2);
ind_3 = find(l == 3);
digit_1 = digits(1:feature,ind_1);
digit_2 = digits(1:feature,ind_2);
digit_3 = digits(1:feature,ind_3);
m1 = mean(digit_1,2);
m2 = mean(digit_2,2);
m3 = mean(digit_3,2);
% within-class variance
Sw = 0;
for k = 1:length(ind_1)
    Sw = Sw + (digit_1(:,k) - m1)*(digit_1(:,k) - m1)';
end
for k = 1:length(ind_2)
    Sw = Sw + (digit_2(:,k) - m2)*(digit_2(:,k) - m2)';
end
for k = 1:length(ind_3)
    Sw = Sw + (digit_3(:,k) - m3)*(digit_3(:,k) - m3)';
end
% between class
Sb = 0;
mu = (m1+m2+m3)/3;
Sb = (m1 - mu)*(m1 - mu)';
Sb = Sb + (m2 - mu)*(m2 - mu)';
Sb = Sb + (m3 - mu)*(m3 - mu)';

[V2, D] = eig(Sb,Sw);
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

v1 = w'*digit_1;
v2 = w'*digit_2;
v3 = w'*digit_3;

if mean(v2)>mean(v3)
    w = -w;
    v2 = -v2;
    v3 = -v3;
end

plot(v1, ones(length(ind_1)), 'mx', 'Linewidth',2), hold on
plot(v2,ones(length(ind_2))*0.5,'ob','Linewidth',2),
plot(v3,zeros(length(ind_3)),'dr','Linewidth',2)
% classifier between digit 1 and 2
sort1 = sort(v1);
sort2 = sort(v2);
t1 = length(sort1);
```

```
t2 = 1;
while sort1(t1)>sort2(t2) % compare the last to the first
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sort1(t1) + sort2(t2))/2;
xline(threshold,'-m')
sort2 = sort(v2);
sort3 = sort(v3);
t1 = length(sort2);
t2 = 1;
while sort2(t1)>sort3(t2) % compare the last to the first
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sort2(t1) + sort3(t2))/2;
xline(threshold)

%% Reshape the testing data
test_images = double(reshape(test_images, size(test_images,1)*size(test_images,2),
[]).');
test_labels = double(test_labels);
% feature/mode = row
% image on column, 1 column is the image of digit
testdata = transpose(test_images);
testlb = transpose(test_labels);
%% train the data to test on the test data
feature = 200; % how many features we want to use
training_rate = zeros(10,10);
correct_rate_LDA = zeros(10,10);
miss_classify = 0;
for j = 1:10
    for k = 1:10
        miss_classify = 0;
        if j ~= k
            % build a classifier between digit m and digit n
            m = j - 1;
            n = k - 1;
            digits = S*V';
            nj = find(testlb == m);
            nk = find(testlb == n);
            digit_j = digits(1:feature,nj);
            digit_k = digits(1:feature,nk);
            mj = mean(digit_j,2);
            mk = mean(digit_k,2);
           % within-class variance
            Sw = 0;
            for c = 1:length(nj)
                Sw = Sw + (digit_j(:,c) - mj)*(digit_j(:,c) - mj)';
            end
            for c = 1:length(nk)
                Sw = Sw + (digit_k(:,c) - mk)*(digit_k(:,c) - mk)';
            end
            % between class
            Sb = (mj-mk)*(mj-mk)';
```

```
[V2, D] = eig(Sb,Sw);
            [lambda, ind] = max(abs(diag(D)));
            w = V2(:,ind);
            w = w/norm(w,2);

            vj = w'*digit_j;
            vk = w'*digit_k;

            if mean(vj)>mean(vk)
                w = -w;
                vj = -vj;
                vk = -vk;
            end

            sortj = sort(vj);
            sortk = sort(vk);
            t1 = length(sortj);
            t2 = 1;
            while sortj(t1)>sortk(t2) % compare the last to the first
                t1 = t1 - 1;
                t2 = t2 + 1;
            end
            threshold = (sortj(t1) + sortk(t2))/2;
            miss_classify = length(find(vj > threshold));
            miss_classify = miss_classify + length(find(vk < threshold));
            training_rate(j,k) = 1 - miss_classify/(length(vj) + length(vk));

            % Validate using test data
            set_index1 = find(testlb == m);
            set_index2 = find(testlb == n);
            set_index = [set_index1 set_index2];
            right = [zeros(1,length(set_index1)) ones(1,length(set_index2))];
            testset = testdata(:,set_index);
            testnum = size(testset,2);
            U_1 = U(:,1:feature); % Add this in
            testmat = U_1'*testset;
            pval = w'*testmat;
            resvac = (pval>threshold); % 0 == digit j, 1 == digit k
            err = abs(resvac - right);
            correct_rate_LDA(j,k) = 1 - (sum(err)/testnum);
        end
    end
end

%% Tree
tree=fitctree(trainingdata',traininglb','MaxNumSplits',10);
%%
1 - loss(tree,trainingdata',traininglb') % accuracy rate on trainingdata
1 - loss(tree,testdata',testlb') % accuracy rate on test data
```

```
%% SVM for digits 1 and 2
ind_1 = find(traininglb==0);
ind_2 = find(traininglb==1);
ind = [ind_1,ind_2];
X = trainingdata(:,ind);
y = [repelem(0,length(ind_1)) repelem(1,length(ind_2))];
Mdl = fitcsvm(X',y');
%%
ind_1 = find(testlb == 2);
ind_2 = find(testlb == 6);
ind = [ind_1,ind_2];
right = [repelem(0,length(ind_1)) repelem(1,length(ind_2))];
test_labels = predict(Mdl,X');
%%
testnum = length(y);
% diff = abs(right' - test_labels);
% testnum - sum(right'==test_labels);
correct = sum(y'==test_labels)/testnum
%% SVM all digits
result = zeros(10,10);
for j = 1:10
    for k = 1:10
        if (j ~= k)
            m = j-1;
            n = k-1;
            ind_m = find(traininglb==m);
            ind_n = find(traininglb==n);
            ind = [ind_m ind_n];
            X = trainingdata(:,ind);
            y = [repelem(m,length(ind_m)) repelem(n,length(ind_n))];
            Mdl = fitcsvm(X',y' );
            % validate the classifier using test data
            test_ind_m = find(testlb==m);
            test_ind_n = find(testlb==n);
            test_ind = [test_ind_m test_ind_n];
            right = [repelem(m,length(test_ind_m)) repelem(n,length(test_ind_n))];
            test_labels = predict(Mdl,testdata(:,test_ind)');
            testnum = length(test_ind);
            % record the correct rate
            result(j,k) = 1-(testnum-sum(right'==test_labels))/testnum;
        end
    end
end
```