AMATH 482 Homework 4
Classifying Digits

Weien Guo

March 10, 2021

Abstract

In this assignment, we are given a database on hand-written digits. Our job is to use Singular Value Decomposition (SVD), Principle Component Analysis (PCA), and Linear Discriminant Analysis (LDA) to analyze them and identify the digits in the test dataset. The accuracy of the predictions are recorded and used for the evaluation of our models.

1 Introduction and Overview

The dataset for this assignment is from the MNIST database of handwritten digits, which has a training set of 60,000 examples, and a test set of 10, 000 examples. The labels for each image is also provided. We need to turn the image data into a matrix in MATLAB, and then we will do the SVD analysis using the singluar values. We wil also use the results to reconstruct the image.

For the second part, we will build a linear classifier (LDA) for classifying a pair of digits in the PCA space. With this model, we can determine the pair of digits that is the most difficult to identify, and the pair of digits that is the easiest to identify. We will also explore the LDA for three digits.

At the end, we will run some tests using MATLAB built-in functions on support vector machines (SVM) and decision tree classifiers. The results from these functions will put in comparison with the results from models developed by ourselves.

2 Theoretical Background

2.1 Singular Value Decomposition

SVD is a very powerful and useful tool in computational mathematics. A matrix, representing a transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$, can be decomposed to

$$A = U\Sigma V^*$$

with $U \in \mathbb{R}^{n\times m}$ and $V \in \mathbb{R}^{n\times n}$ are unitary, $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal.

The values on the diagonal of Σ, σ_i, are called singular values of the matrix A. Column in U are left singular values of A. Columns in V are right singular values of A. They also satisfy that $Av_i = \sigma_i u_i$ and $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$. σ_i are always non-negative and real numbers. The number of non-zero singular values are called the rank of A. MATLAB has built-in function `svd()` that calculates U, Σ, V matrices for us.

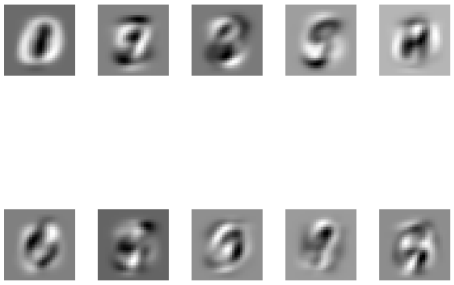


Figure 1: First ten principal components from SVD

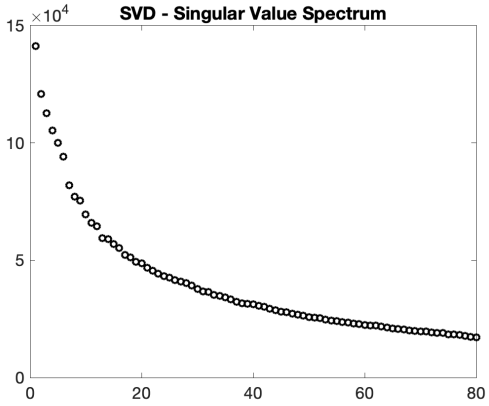


Figure 2: The singular values from SVD

2.2 Linear Discriminant Analysis (LDA)

Assume we have two classes in our data. The data is projected onto a new base. The LDA gives the best projection that maximizes the distance between the inter-class data while minimizong the intra-class data. Having two vectors, μ_1 and μ_2, we can calculate the **between-class scatter matrix**

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

which gives the variance between groups, and **within-class scatter matrix**

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

which gives the variance within each group. The vector

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

And using eigenvalues λ, we have

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

When we have more than two groups,

$$\mathbf{S}_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T$$

$$\mathbf{S}_W = \sum_{j=1}^{N} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T$$

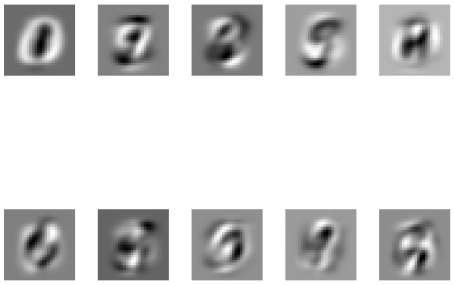


Figure 3: First ten principal components from SVD

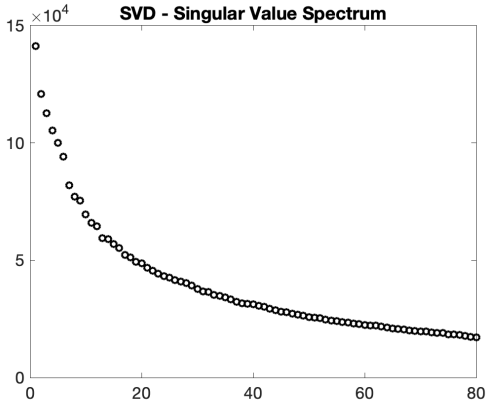
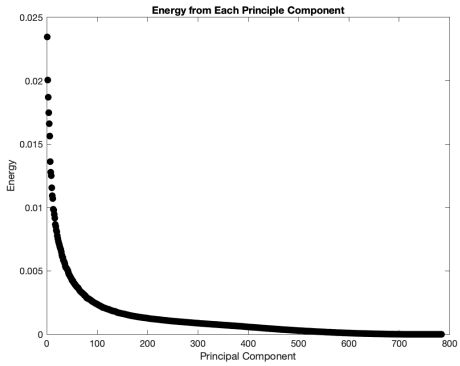
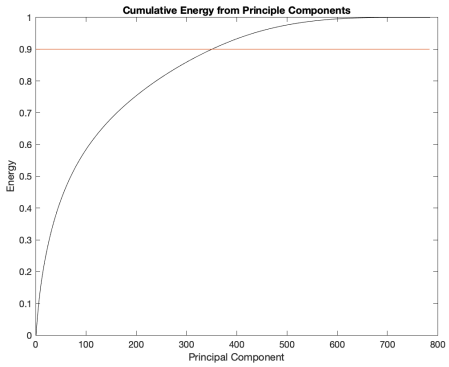


Figure 4: The singular values from SVD



(a) Energy captured in each principle component



(b) The cumulative energy. The red line is the 90% threshold

Figure 5

3 Algorithm Implementation and Development

3.1 Part 1

1. Load the database into MATLAB using `mnist_parse()`. Each image is of size 28*28. We reshape the image into vector representation. Then we have a matrix of size 784*60,000 in `double`. Then we subtract row-wise mean from this matrix for the PCA coming up.
2. Perform SVD on the matrix using `svd()`. We can calculate the energy for each principle component using the eigenvalues.
3. In order to reconstruct the image, we try some different rank values, and use the image to see which rank is enough for reconstruction.
4. Project each digit onto 3 selected V-modes in a 3-D plot.

3.2 Part 2

1. Load and prepare the testing data like we did for the training data in Part 1.

2. Multiply the test matrix with matrix U', which consists of transposed left singular values, and multiple matrix S with V', preparing for the PCA projection.

3. Use the rank approximation that we determined earlier, we run the LDA algorithm. We first calculate the scatter matrices and find the best projection line between data for two different digits. Then, we project the data onto the line and finds a threshold. With the threshold value, we can find the training error easily by locating the data with respect to threshold and compare the result with the true label. For the testing error, we need to project the testing data onto $\mathbf{w}$, and compare with the threshold value. Those points that do not lie in the right region will count towards the error.

Figure 6: Image reconstruction with rank 1, 10, 50, 70, 80, 100 respectively

4. LDA on three digits is similar. We modify the equation for $\mathbf{S}_B$ and $\mathbf{S}_w$. The rest follows the same logic. This time, since we have three digits, there would be two threshold values. Because the range for each digit relative to the threshold changes, so we will only run the model for one training combination.

5. MATLAB has its built-in functions for SVM and decision tree, `fitctree()` and `fitcsvm()`. We input the training data for fitting a decision tree, and use `predict()` to get the fitted digit for each image. The error is found by comparing the prediction and the true label. The decision tree is able to separate ten digits at the same time, while the SVM algorithm can only take two digits at a time. We will use it to check the hardest and easiest pair we found earlier.

4 Computational Results

4.1 Part 1

Singular value decomposition is performed on the training data. Figure 1 shows the first ten principle components, which are the first ten columns in matrix `U`. We can see that the first plot looks like number zero. That is probably because much of the variance in the handwritten digits are along that circle and in the center of the plot. That is reasonable. The rest of the plots show that the variance becomes more and more unrecognizable and more chaotic. That is because more

characteristics in the data have been captured by the first few principle components, and the rest becomes more and more trivial.

Figure 2 shows the singular values, which are on the diagonal of matrix `S`. The values decrease along a curved line, and does not show any dominant ones. So we can conclude that we would need many principle components for reconstructing the image later. The two plots in Figure 3 help us to determine the rank. Figure 3(a) shows the energy within each principle component. The value decreases rapidly in the 100 points and the rest is very close to zero. This means that after the 100th principle component, the rest would not contribute to some noticeable additional feature. Figure 3(b) is another way of identifying the rank, that is by setting a intended percentage of variance captured. In this case I set this value to 90%, meaning I would want my approximation to have 90% of the entire variance. The rank found here is 350.

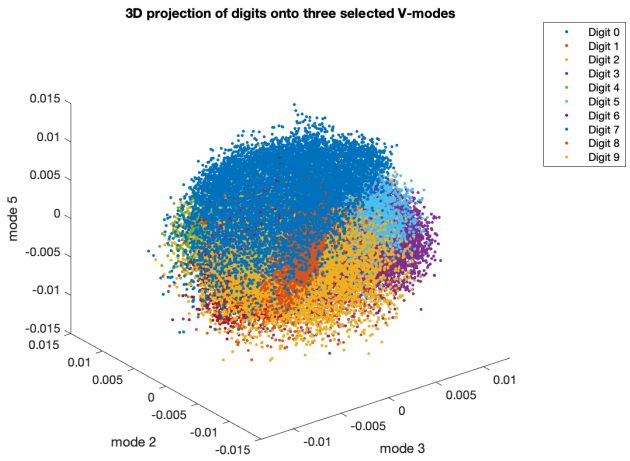


Figure 7: 3D projection of digits onto V-modes 3, 2, and 5

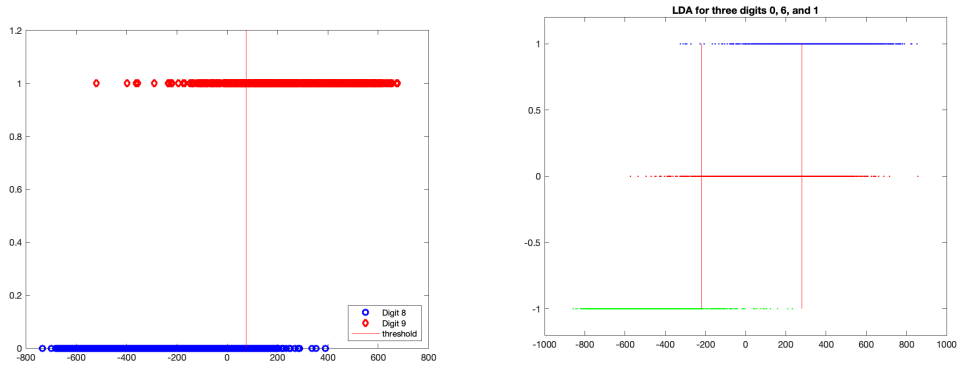
Image reconstruction is shown in Figure 4. It is done by using rank values 1, 10, 50, 70, 80, and 100 on the first handwritten digit, that is a 5. The first two images do not provide a clear and identifiable digit 5. The next two plots are blurry, and the last two are almost the same. From this observation, we can conclude that with rank 70, we can have a pretty good reconstruction of the image.

Matrix `U` is orthonormal, calculated from the singular value decomposition. Each column of U is a principle component, that is the features in the handwritten digits explained by this component. Matrix `S` has all the singular values, which can used for calculating the energy for each mode. Matrix `V` can be interpreted as the projection of the original matrix onto the principle components.

With this interpretation in mind, we can make a 3-D projection of the digits onto three selected V-modes, as in Figure 5. Here I chose V-modes 3, 2, and 5. Each color represents a digit label. We can clearly see that the digits are in clusters, meaning the digits can be classified.

4.2 Part 2

For classifying each pair of digits, we run the pair through the LDA algorithm. Training and testing error are stored in a matrix. As shown in Figure 6(a), digit pair (8, 9) is plotted in color blue and red respectively. The threshold is the red vertical line. For two digit pairs and using 70 features, we have the pair 5 and 8 being the most difficult to identity. Their testing error rate is 0.0477, in other words, accuracy is 95.23%. The easiest pair to identity is 0 and 4. Their testing error rate is 0.0015, in other words, accuracy is 99.85%.



(a) LDA for two digits. This test is run with 8 and 9 (b) LDA for three digits. This test is run with 0, 6, and 1

Figure 8

In the three digit pair model, we used 0, 6 and 1 for testing purposes. Threshold 1 is the red line on the left, and threshold 2 is on the right. The error rate in this example is 13.63%.

Lastly, we now use the MATLAB built-in functions. The decision tree with all ten digit classes gives training error rate 3.21%, and testing error rate 37.38%. When using the SVM function on the most difficult pair, 5 and 8, the testing error rate is 12.49%. The easiest pair, 0 and 4, gives testing error rate 0.46%. Both errors are slightly higher than the LDA model.

5 Summary and Conclusions

In Part 1, we performed SVD, and dived into singular value and principle component analysis. We have determined a rank value of 70 for a good image reconstruction. Then in Part 2, we used LDA to classify two digit pairs, and three digit pairs. The error rate is very small, indicating a good prediction. When using MATLAB built-in functions for SVM and decision tree, the testing error for classifying all 10 digits is not small, but two digit pair prediction remained satisfactory.

Appendix A MATLAB Functions

- `[U,S,V] = svd(X)` produces a diagonal matrix S with non-negative diagonal elements in decreasing order, and unitary matrices `U` and `V` so that $X = U * S * V'$.
- `E = eig(A)` produces a column vector `E` containing the eigenvalues of matrix `A`.
- `B = sort(A)` sorts `A` in ascending order.
- `TREE=fitctree(TBL,Y)` returns a classification decision tree for data in the predictor variables `TBL` and response `Y`.
- `MODEL=fitcsvm(TBL,Y)` returns an SVM model for data in the predictor variables `TBL` and response `Y`.
- `LABEL = predict(TREE,X)` returns predicted class labels `LABEL`.

Appendix B MATLAB Code

```
clear all; close all; clc
%%

[images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');

%%
image = zeros(784, 60000);

for i = 1:60000
    Im = images(:, :, i);
    image(:, i) = Im(:);
end
image = double(image);
% %%
avg = mean(image, 2);
image = image - repmat(avg,1,60000);
%%
[U,S,V] = svd(image,'econ');

%% Plot first ten principal components
figure()
for k = 1:10
   subplot(2,5,k)
   ut1 = reshape(U(:,k),28,28);
   ut2 = rescale(ut1);
   imshow(ut2)
end
saveas(gcf, 'PC.png')
clf
%% Singular Value Spectrum
plot(diag(S),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 80])
title('SVD - Singular Value Spectrum')
saveas(gcf, 'Singular Value Spectrum.png')
clf

%%
energy = diag(S)./sum(diag(S));
plot(energy, 'k.', 'MarkerSize', 20)
xlabel('Principal Component')
ylabel('Energy')
title('Energy from Each Principle Component')
saveas(gcf, 'Energy.png')
clf

energyCap = [0, cumsum(energy)'];
plot(energyCap, 'k-')
hold on
```

```
plot(1:784, ones(784, 1)*0.9)

xlabel('Principal Component')
ylabel('Energy')
title('Cumulative Energy from Principle Components')
saveas(gcf, 'Cumulative Energy.png')
clf

rank = find(cumsum(energy)>0.9);
rank = rank(1);
% rank = 350
%% image reconstruction
rank = [1, 10, 50, 70, 80, 100];
for i = 1:6
    subplot(2,3,i)
    r = rank(i);
    newU = U(:, 1:r);
    newS = S(1:r, 1:r);
    newV = V(1, 1:r);
    reconstruction = newU * newS * newV';
    ut1 = reshape(reconstruction,28,28);
    imshow(rescale(ut1))
end
saveas(gcf, 'rank.png')
clf
%%
zero = find(labels == 0);
one = find(labels == 1);
two = find(labels == 2);
three = find(labels == 3);
four = find(labels == 4);
five = find(labels == 5);
six = find(labels == 6);
seven = find(labels == 7);
eight = find(labels == 8);
nine = find(labels == 9);

clf
v = V';
plot3(v(3, zero), v(2, zero), v(5, zero), '.');
hold on
plot3(v(3, one), v(2, one), v(5, one), '.');
plot3(v(3, two), v(2, two), v(5, two), '.');
plot3(v(3, three), v(2, three), v(5, three), '.');
plot3(v(3, four), v(2, four), v(5, four), '.');
plot3(v(3, five), v(2, five), v(5, five), '.');
plot3(v(3, six), v(2, six), v(5, six), '.');
plot3(v(3, seven), v(2, seven), v(5, seven), '.');
plot3(v(3, eight), v(2, eight), v(5, eight), '.');
plot3(v(3, nine), v(2, nine), v(5, nine), '.');
legend('Digit 0', 'Digit 1', 'Digit 2', 'Digit 3', 'Digit 4', 'Digit 5', 'Digit 6', 'Di
```

```
xlabel('mode 3')
ylabel('mode 2')
zlabel('mode 5')
title('3D projection of digits onto three selected V-modes')
saveas(gcf, '3D.png')
clf

%% test error matrix
[testimages, testlabels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte
%%
testimage = zeros(784, 10000);

for i = 1:10000
    Im = testimages(:, :, i);
    testimage(:, i) = Im(:);
end
testimage = double(testimage);

avg = mean(testimage, 2);
testimage = testimage - repmat(avg,1,10000);

TestMat = U'*testimage; % PCA projection


%%
ErrorMat = zeros(45, 4);
count = 1;

feature = 70;

digits = S*V';

for i = 0:8
    for j = (i+1) :9
        first = find(labels == i);
        second = find(labels == j);

        D1 = digits(1:feature,first);
        D2 = digits(1:feature,second);

        m1 = mean(D1,2);
        m2 = mean(D2,2);

        Sw = 0;
        for k = 1:length(first)
            Sw = Sw + (D1(:,k) - m1)*(D1(:,k) - m1)';
        end
        for k = 1:length(second)
           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;

TrainError1 = sum(sort1 > threshold);
TrainError2 = sum(sort2 < threshold);
error = (TrainError1 + TrainError2)/(length(sort1)+length(sort2));

% test data
lookfor = [i, j];
exist = ismember(testlabels, lookfor);
testInd = find(exist);

curTestMat = TestMat(1:feature, testInd);

pval = w' * curTestMat;

ResVec = (pval > threshold);

digitcategory = ResVec * j;
digitcategory(digitcategory == 0) = i;

err = (digitcategory' ~= testlabels(testInd));
errNum = sum(err);
testError = errNum/length(testInd);

ErrorMat(count, 1) = i;
ErrorMat(count, 2) = j;
```

```
        ErrorMat(count, 3) = error;
        ErrorMat(count, 4) = testError;
        count = count+1;
    end
end

%%
plot(v1,zeros(1, length(v1)),'ob','Linewidth',2)
hold on
plot(v2,ones(1, length(v2)),'dr','Linewidth',2)
ylim([0 1.2])
plot([threshold, threshold], [0, 1.2], 'r')
legend('Digit 8', 'Digit 9', 'threshold', 'Location','southeast')
saveas(gcf, 'LDA.png')
clf

%%
[M, I] = max(ErrorMat(:, 3));
ErrorMat(I, [1, 2])
% 3 & 5
%%
[M, I] = min(ErrorMat(:, 3));
ErrorMat(I, [1, 2])
% 6 & 7
%%
[M, I] = max(ErrorMat(:, 4));
ErrorMat(I, [1, 2, 4])
% 5 & 8 error = 0.0477
%%
[M, I] = min(ErrorMat(:, 4));
ErrorMat(I, [1, 2, 4])
% 0 & 4 error = 0.0015
%%
feature = 70;

firstDigit = image(:, labels == 0);
secondDigit = image(:, labels == 6);
thirdDigit = image(:, labels == 1);

[U,S,V,threshold1,threshold2,w,sortd1,sortd2, sortd3] = three_digits_trainer(firstDigit
%%
plot(sortd1, ones(length(firstDigit), 1), 'b.', 'MarkerSize', 0.3)
ylim([-1.2 1.2])
hold on
plot(sortd2, zeros(length(secondDigit), 1), 'r.', 'MarkerSize', 0.3)
plot(sortd3, ones(length(thirdDigit), 1) * -1, 'g.', 'MarkerSize', 0.3)
plot([threshold1, threshold1], [-1, 1], 'r')
plot([threshold2, threshold2], [-1, 1], 'r')
title('LDA for three digits 0, 6, and 1')
saveas(gcf, 'three digits.png')
clf
```

```
TrainErrorN1 = sum(sortd1 < threshold2);
TrainErrorN2 = sum(sortd2 < threshold1) + sum(sortd2 > threshold2);
TrainErrorN3 = sum(sortd3 > threshold1);

error = (TrainErrorN1 + TrainErrorN2 + TrainErrorN3)/(length(firstDigit)+length(secondD
% 0.1363

%% SVM and decision tree
tree = fitctree(image', labels);
%% trainning error
pred = predict(tree, image');
sum(pred ~= labels) / 60000;
% 0.0321
%% test error
pred = predict(tree, testimage');
sum(pred ~= testlabels) / 10000;
% 0.3738

%% most difficult digit pair 5 & 8
i = 5;
j = 8;

lookfor = [i, j];
exist = ismember(labels, lookfor);
ind = find(exist);
y = labels(ind);
X = image(:, ind)';

SVMModel = fitcsvm(X,y);

exist = ismember(testlabels, lookfor);
ind = find(exist);
truelabel = testlabels(ind);
test = testimage(:, ind)';
test_labels = predict(SVMModel, test);

sum(test_labels ~= truelabel) / length(ind)
% 0.1249

%% easier digit pair 0 & 4
i = 0;
j = 4;

lookfor = [i, j];
exist = ismember(labels, lookfor);
ind = find(exist);
y = labels(ind);
X = image(:, ind)';
```

```
SVMModel = fitcsvm(X,y);

exist = ismember(testlabels, lookfor);
ind = find(exist);
truelabel = testlabels(ind);
test = testimage(:, ind)';
test_labels = predict(SVMModel, test);

sum(test_labels ~= truelabel) / length(ind)
% 0.0046

%% three digits trainer function

function [U,S,V,threshold1,threshold2,w,sortd1,sortd2, sortd3] = three_digits_trainer(d

    n1 = length(digit1);
    n2 = length(digit2);
    n3 = length(digit3);
    [U,S,V] = svd([digit1 digit2 digit3],'econ');
    digits = S*V';
    U = U(:,1:feature);

    D1 = digits(1:feature,1:n1);
    D2 = digits(1:feature,n1+1:n1+n2);
    D3 = digits(1:feature,n1+n2+1:n1+n2+n3);
    m1 = mean(D1,2);
    m2 = mean(D2,2);
    m3 = mean(D3,2);

    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
    for k=1:n3
        Sw = Sw + (D3(:,k)-m3)*(D3(:,k)-m3)';
    end

    meanAll = mean(digits(1:feature, :), 2);
    mean3 = [m1, m2, m3];
    Sb = 0;
    for j = 1:3
        Sb = (mean3(:, j) - meanAll) * (mean3(:, j) - meanAll)';
    end

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

```
    vd1 = w'*D1;
    vd2 = w'*D2;
    vd3 = w'*D3;

    sortd1 = sort(vd1);
    sortd2 = sort(vd2);
    sortd3 = sort(vd3);

    msortd1 = mean(sortd1);
    msortd2 = mean(sortd2);
    msortd3 = mean(sortd3);

    sortedmean = [msortd1, msortd2, msortd3];
    msort = sort(sortedmean);

    threshold1 = (msort(1)+msort(2))/2;
    threshold2 = (msort(2)+msort(3))/2;

end
```