AMATH 482 HW4 - Digit Classification

Winnie Shao

3/10/2021

1 Abstract

In this paper, we will explore the power and limitation of SVD in extracting fundamental structure from data. We will work with forming a low-rank approximation of a collection of hand-written 0 to 9 digit collection. Furthermore, we will explore the foundation of supervised machine learning and implement a classifier. To obtain our goal, we will apply singular value decomposition (SVD), and Linear discrimination analysis (LDA). Also, we are going to implement decision tree learning and support vector machine (SVM) to compare them with LDA on their performance.

Section I: Introduction

Nowadays, machine learning has become a hot topic. It has been applied widely in jobs of prediction, classification, recognition, detection, and recommendations. Image recognition is a important part of machine learning. In this paper, we are applying LDA, decision tree, and SVM on a collection of hand-written digits. Our goal is to design a code that can classify a picture of a digit to the right number. We will test the functionalities of our classifier, and explore the performance under different training and testing sets.

Section II: Theoretical Background

2.1 Quick Review on SVD

Singular value decomposition (SVD) is a factorization of a matrix into several specific components, as the following form:

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

with three components:

$$U \in \mathbf{C}^{m\times m} \text{ is unitary}$$
$$V \in \mathbf{C}^{n\times n} \text{ is unitary}$$
$$\Sigma \in \mathbf{R}^{m\times n} \text{ is diagonal}$$

The diagonal entries of Σ are nonnegative and ordered from largest to smallest so that $\sigma_1 \geq \sigma_2 \geq ... \geq 0$, and every matrix A has a singular value decomposition with singular values σ_j uniquely determined. The SVD of the matrix A shows that the matrix first applies a unitary transformation preserving the unit sphere via V^* . This is followed by a stretching operation that creates an ellipse with principal semi-axes given by the matrix Σ . Finally, the generated hyper-ellipse is rotated by the unitary transformation U.

2.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a method used in statistics to find a linear combination features that characterizes and separates two or more classes of objects. It focuses on maximizing the separability among known classes. LDA can be considered as a dimensional reduction technique and the results could be used in classification process.

LDA is trying to find a suitable projection that maximize the distance between inter-class data while minimizing the intra-class data. For two-class LDA specifically, we will have a projection w such that:

$$w = argmax\frac{w^T S_B w}{w^T S_W w} \tag{2}$$

where

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{3}$$

$$S_W = \sum_{j=1}^{2}\sum_{x}(x - \mu_j)(x - \mu_j)^T \tag{4}$$

These equations are for measuring the variance of the data sets as well as the variance of the difference in the meanings. w can be found via the generalized eigenvalue problem:

$$S_B w = \lambda S_W w \tag{5}$$

2.3 Multi-class LDA

Multi-class LDA is based on the analysis of two scatter matrices: within-class scatter matrix and between-class scatter matrix. The between-class scatter matrix S_B becomes the summation of distances between the center of the total scatter and the center of each category.

$$S_b = \sum_{k=1}^{m} n_k(\mu_k - \mu)(\mu_k - \mu)^T \tag{6}$$

where m is the number of classes, μ is the overall sample mean, and n_k is the number of samples in the k-th class.

2.4 Machine Learning in General

In the following, I will walk through the implementation of the classifier. The major for steps to conduct an image classifier are:

- Decompose images into wavelet basis function to provide an effective way for doing edge detection.
- Find the principal components associated with the categories respectively from the wavelet expaned images
- Design a decision threshold for discrimination between categories. The method to be used here will be linear discrimination analysis.
- Test the algorithm efficiency and accuracy by running through testing data.

Then I will give a brief introduction about SVM and DTL. SVM is essentially an optimization problem that tries to find an optimal hyperplane for all points x and classifies them by conditions according to the hyperplane. DTL follows a simple binary approach. For each variable in the system, scan over all possible values of that variable to be use as a separator of the data. After that, it will find the variable and its associated value that best divides the current collection of data into their respective groups.

Section III: Algorithm Implementation & Development

Step 1: Load the data & Exploratory Analysis

First of all, we load in the digit image (MNIST) data by using the function called mnist parse(). Since we need to apply SVD onto the data, we need to reshape the 3D matrix into 2D matrix. Then we need to subtract the data with row mean. We also need to calculate the r rank approximation judging form how much energy we can obtain by the approximation. Finally, we plot the 3D V-mode.

Step 2: SVD & LDA

Learning form the exploratory analysis, all the spectrogram information, we are going to design the trainer for the classifier. In this section, I will use the 2-class LDA as an example. We will aggregate the two spectrogram-information matrices of different categories into a huge matrix. We produce the economy-size decomposition on this aggregate matrix. Then we multiply S with the transpose of V to get the projections onto principal components. And then, we take feature number equal to rank which is 87 of each class for later classification. Once the singular value decomposition has been performed, the LDA is applied. We calculate the between-class S_b and within-class S_w. of the feature components. Then we use function eig() to get the eigenvalues and eigenvectors of S_b and S_w. we will extract the coefficient w by looking for the max ratio of the between-class scattering to the within class scattering, and multiply the coefficient with two feature component sets. To get the threshold, we trace computed values and get the average point of the boundary.

Step 3: Testing

After getting the threshold for the two classes, we load in the test data and test label. Then we apply the PCA projection and LDA projection on the test dataset to get the result

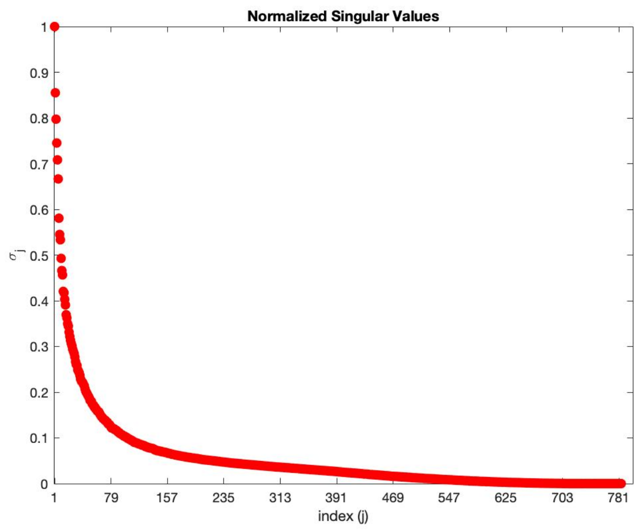

Figure 1: The Singular Value Spectrum

values. By comparing the result with the ground truth, we can examine the success rate for predictions.

Section IV: Computational Results & Conclusion

4.1 Analysis of the data set

First, I will talk about my interpretation for the U, S, V. The columns of U represent an orthogonal basis for the codomain of our data matrix X.Not just any basis, however, the BEST basis for the space where each successive column captures as much of the variance in the data as possible. In this case, as the co-domain of our matrix is the space in which column of the matrix exists, as each column is a face in our data set the co-domain is "face space" Thus, it follows that U is the "optimal" basis for face-space where the SVD's idea of what the important features that make up a face are will be those that capture the most variance. Furthermore, the columns of V then store the coordinates of each of our data measurements (i.e. each face) within this face space.

4.1.1 Singular Spectrum & rank approximation

In this section, we get the singular Spectrum: From this graph we can see from Figure 1 that if we want to obtain 90 percent of the energy in the original matrix, we need a rank at least more than 79. After using for loop, we get the rank = 87.

4.1.2 V-mode 3D plot

We can see from Figure 2 that the digit has some visible separation between each other after plotting the V modes.

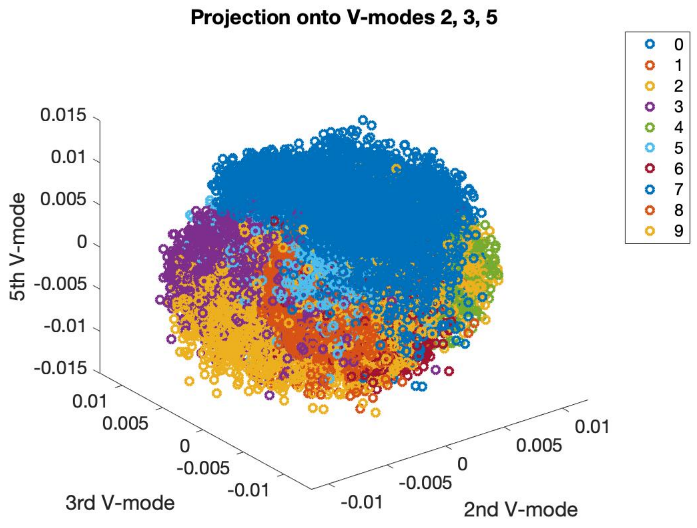

Figure 2: Vmode plot on column 2, 3, 5

4.2 Machine Learning

4.2.1 Identify between 2 digits

In this part, I choose to classify 3 and 7 and I get a success rate of 0.98. Therefore, I think the classifier identifies them reasonably.

4.2.2 Identify between 3 digits

In this part, I choose to classify 0, 4, and 9. I get a success rate of 0.7. The success rate has decreased but I think it is still in a reasonable range.

4.2.3 The most difficult/ easy to separate

For the most difficult to separate, I run a for loop and collect each error for each pair of classification and take the max error. I got the result that 4 and 9 is the most difficult pair to distinguish. For the easiest to separate, I run a for loop and collect each error for each pair and take the min error. I got the result that 0 and 4 is the easiest to distinguish.

4.2.4 Try out SVM and DTL

When performing classification on the whole test data set, SVM has success rate of 0.94 but DTL only has rate of 0.2685. We can learn from the success rate that SVM is good at separating between 10 digits but DTL is not.

The accuracy of SVM for the most difficult pair is 0.9709 and for the easiest pair is 0.9949. The accuarcy of DTL for the most difficult pair is 0.8895 and for the easiest pair is 0.9664. Therefore, at all aspect, SVM performs better.

Appendix A: MATLAB functions used

- svd(X): Calculates the Singular Value Decomposition of the given matrix X, returning U, Σ, V.
- fitcecoc(training data, labels): Generates a predictive model using multi-class SVM, training it with the given data and its associated labels.
- fitctree(training data, labels): Generates a predictive model using Decision Tree Learning training it with the given data and its associated labels.

Appendix B: MATLAB codes

```
clear all; clc; close all;

% Part 1 - load in and reshape train and test data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte',...
                                'train-labels-idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte',...
                              't10k-labels-idx1-ubyte');
% cast the images to double
images_train = im2double(images_train);
images_test = im2double(images_test);
% reshape to one image per column for train data
images_train = reshape(images_train, [28*28, 60000]);
[M, N] = size(images_train);
train_data = images_train - repmat(mean(images_train,2),1,N);
% reshape to one image per column for test data
images_test = reshape(images_test, [28*28, 10000]);
[M, N] = size(images_test);
test_data = images_test - repmat(mean(images_test,2),1,N);

% Part 1 - Perform SVD on train data
[U, S, V] = svd(train_data, 'econ');

% Part 1 - plot singular values
figure(1)
plot(diag(S) ./ max(diag(S)), 'r.', 'markersize', 20);
xticks(1:round(length(S)/10):length(S))
ylim([0,1])
title('Normalized Singular Values')
ylabel('\sigma_j')
xlabel('index (j)')

% Part 1 - choose rank
% We will choose the r that captures at least 90% of the energy of the
% system
energy = 0;
total = sum(diag(S));
% how much energy we want our modes to capture.
% 75% does alright; 90% does very good.
threshold = 0.9;
r = 0;
while energy < threshold
    r = r + 1;
    energy = energy + S(r,r)^2/sum(diag(S).^2);
end
rank = r;

% Part 1 - plot 3d v-modes on column 2, 3, 5
figure(2)
for label = 0:9
    label_indices = find(labels_train == 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', 14)

clc; clear all; close all;

% Part 1 - load in and reshape train and test data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte',...
                               'train-labels-idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte',...
                             't10k-labels-idx1-ubyte');
% cast the images to double
images_train = im2double(images_train);
images_test = im2double(images_test);
% reshape to one image per column for train data
images_train = reshape(images_train, [28*28, 60000]);
[M, N] = size(images_train);
train_data = images_train - repmat(mean(images_train,2),1,N);
% reshape to one image per column for test data
images_test = reshape(images_test, [28*28, 10000]);
[M, N] = size(images_test);
test_data = images_test - repmat(mean(images_test,2),1,N);
```

```
%% Part 2 - Q1: train data
% choose two digits 3 and 7
label3 = find(labels_train == 3);
digit3 = train_data(:,label3);
label7 = find(labels_train == 7);
digit7 = train_data(:,label7);
%%
% apply SVD to digit 3 and 7 data
[U,S,V] = svd([digit3, digit7], 'econ');
%%
% project on to PCA space
feature = 87;
n3 = size(digit3,2);
n7 = size(digit7,2);
digits = S*V';
digit3s = digits(1:feature,1:n3);
digit7s = digits(1:feature,n3+1:n3+n7);
%%
% Calculate scatter matrices
m3 = mean(digit3s,2);
m7 = mean(digit7s,2);
Sw = 0; % within class variances
for k = 1:n3
    Sw = Sw + (digit3s(:,k) - m3)*(digit3s(:,k) - m7)';
end
for k = 1:n7
   Sw =  Sw + (digit7s(:,k) - m7)*(digit7s(:,k) - m7)';
end
Sb = (m3-m7)*(m3-m7)'; % between class

% Find the best projection line
[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

% project on to w
vdigit3 = w'*digit3s;
vdigit7 = w'*digit7s;

% Make digit 3 below the threshold
if mean(vdigit3) > mean(vdigit7)
    w = -w;
     vdigit3 = -vdigit3;
     vdigit7 = -vdigit7;
```

```
end

% Find the threshold value
sortd3 = sort(vdigit3);
sortd7 = sort(vdigit7);

t1 = length(sortd3);
t2 = 1;
while sortd3(t1) > sortd7(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sortd3(t1) + sortd7(t2))/2;
%% Part 2 - Q1: test data
% Find digit 3 and digit 7 in the test data
label3_test = find(labels_test == 3);
digit3_test = test_data(:,label3_test);
label7_test = find(labels_test == 7);
digit7_test = test_data(:,label7_test);
test37 = [digit3_test digit7_test];
%% Part 2 - Q1: test
testNum = size(test37, 2);
testMat = U'* test37;
pval = w'*testMat(1:87,:);
ResVec = (pval > threshold);
trueRes = [3*ones(1,1010) 7*ones(1,1028)];
trueRes = (trueRes == 7);
err = abs(ResVec - trueRes);
errNum = sum(err);
sucRate = 1 - errNum/testNum;

clc; clear all; close all;

% Part 1 - load in and reshape train and test data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte',...
                                   'train-labels-idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte',...
                                 't10k-labels-idx1-ubyte');
% cast the images to double
images_train = im2double(images_train);
images_test = im2double(images_test);
% reshape to one image per column for train data
images_train = reshape(images_train, [28*28, 60000]);
[M, N] = size(images_train);
train_data = images_train - repmat(mean(images_train,2),1,N);
% reshape to one image per column for test data
```

```
images_test = reshape(images_test, [28*28, 10000]);
[M, N] = size(images_test);
test_data = images_test - repmat(mean(images_test,2),1,N);

%%
label0 = find(labels_train == 0);
digit0 = train_data(:,label0);
label4 = find(labels_train == 4);
digit4 = train_data(:,label4);
label9 = find(labels_train == 9);
digit9 = train_data(:,label9);
label0_test = find(labels_test == 0);
digit0_test = test_data(:,label0_test);
label4_test = find(labels_test == 4);
digit4_test = test_data(:,label4_test);
label9_test = find(labels_test == 9);
digit9_test = test_data(:,label9_test);
test_data049 = [digit0_test, digit4_test, digit9_test];
labels_test049 = [zeros(1,size(digit0_test,2)),...
    4*ones(1,size(digit4_test,2)),...
    9*ones(1,size(digit9_test,2))]';

%%
[U,S,V,w,vdigit1,vdigit2,vdigit3] = class3_trainer(digit0, digit4,...
    digit9, 87)

sort1 = sort(vdigit1);
sort2 = sort(vdigit2);
sort3 = sort(vdigit3);

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

TestNum = size(test_data049,2);
TestMat = U'*test_data049;  % PCA projection
pval = w'*TestMat;  % LDA projection

ResVec = zeros(1,TestNum); % Answer by the classifier model
for i = 1:TestNum
    if pval(i) > threshold2
        ResVec(i) = 0; % band1
    elseif pval(i) < threshold1
        ResVec(i) = 4; % band2
    else
        ResVec(i) = 9; % band3
    end
```

```
end

err_num = 0;
for i = 1:TestNum
    if (ResVec(i) ~= labels_test049(i))
        err_num = err_num + 1;
    end
end
%%
sucRate = 1-err_num/TestNum

clc; clear all; close all;

% Part 1 - load in and reshape train and test data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte',...
                               'train-labels-idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte',...
                             't10k-labels-idx1-ubyte');
% cast the images to double
images_train = im2double(images_train);
images_test = im2double(images_test);
% reshape to one image per column for train data
images_train = reshape(images_train, [28*28, 60000]);
[M, N] = size(images_train);
train_data = images_train - repmat(mean(images_train,2),1,N);
% reshape to one image per column for test data
images_test = reshape(images_test, [28*28, 10000]);
[M, N] = size(images_test);
test_data = images_test - repmat(mean(images_test,2),1,N);

%%
accuracy = [];
index = 1;
for i = 0:8
    label1 = find(labels_train == i);
    digit1 = train_data(:,label1);
    for j = i+1:9
        label2 = find(labels_train == j);
        digit2 = train_data(:,label2);
        [U,S,V,threshold,w,sort1,sort2] = digit_trainer(digit1,...
            digit2, 87);
        label1_test = find(labels_test == i);
        digit1_test = test_data(:,label1_test);
        label2_test = find(labels_test == j);
        digit2_test = test_data(:,label2_test);
        test12 = [digit1_test digit2_test];
        testNum = size(test12, 2);
        testMat = U'* test12;
        pval = w'*testMat;
        ResVec = (pval > threshold);
        trueRes = [i*ones(1,size(digit1_test,2)),...
            j*ones(1,size(digit2_test,2))];
        trueRes = (trueRes == j);
        err = abs(ResVec - trueRes);
        errNum = sum(err);
        sucRate = 1 - errNum/testNum;
        accuracy(index) = sucRate;
        index = index + 1;
    end
end

%%
[max, index_max] = max(accuracy);
[min, index_min] = min(accuracy);

clc; clear all; close all;

% Part 1 - load in and reshape train and test data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte',...
                                'train-labels-idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte',...
                                't10k-labels-idx1-ubyte');
% cast the images to double
images_train = im2double(images_train);
images_test = im2double(images_test);
% reshape to one image per column for train data
images_train = reshape(images_train, [28*28, 60000]);
[M, N] = size(images_train);
train_data = images_train - repmat(mean(images_train,2),1,N);
% reshape to one image per column for test data
images_test = reshape(images_test, [28*28, 10000]);
[M, N] = size(images_test);
test_data = images_test - repmat(mean(images_test,2),1,N);

%% tree
tree = fitctree(train_data',labels_train,'MaxNumSplits',3,'CrossVal','on');
test_labels = predict(tree.Trained{1}, test_data');
%%
tree_acc = sum(labels_test == test_labels)/ 10000;
%% svm
Mdl = fitcecoc(train_data',labels_train);
test_labels = predict(Mdl,test_data');
```

```
%% svm accuracy
svm_acc = sum(labels_test == test_labels)/ 10000;

clc; clear all; close all;

% Part 1 - load in and reshape train and test data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte',...
                              'train-labels-idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte',...
                            't10k-labels-idx1-ubyte');
% cast the images to double
images_train = im2double(images_train);
images_test = im2double(images_test);
% reshape to one image per column for train data
images_train = reshape(images_train, [28*28, 60000]);
[M, N] = size(images_train);
train_data = images_train - repmat(mean(images_train,2),1,N);
% reshape to one image per column for test data
images_test = reshape(images_test, [28*28, 10000]);
[M, N] = size(images_test);
test_data = images_test - repmat(mean(images_test,2),1,N);

%% easy data
label0 = find(labels_train == 0);
digit0 = train_data(:,label0);
label4 = find(labels_train == 4);
digit4 = train_data(:,label4);
train_data04 = [digit0, digit4];
labels_train04 = [zeros(1,size(digit0,2)),...
    4*ones(1,size(digit4,2))]';
label0_test = find(labels_test == 0);
digit0_test = test_data(:,label0_test);
label4_test = find(labels_test == 4);
digit4_test = test_data(:,label4_test);
test_data04 = [digit0_test, digit4_test];
labels_test04 = [zeros(1,size(digit0_test,2)),...
    4*ones(1,size(digit4_test,2))]';
%% easy tree
tree_04 = fitctree(train_data04',labels_train04,'MaxNumSplits',...
    3,'CrossVal','on');
test_labels = predict(tree_04.Trained{1}, test_data04');
tree_04_acc = sum(labels_test04 == test_labels)/ size(test_labels,1);

%% svm tree
Mdl_04 = fitcecoc(train_data04',labels_train04);
test_labels = predict(Mdl_04,test_data04');
```

```
svm_04_acc = sum(labels_test04 == test_labels)/size(test_labels,1);

%% difficult data
label4 = find(labels_train == 4);
digit4 = train_data(:,label4);
label9 = find(labels_train == 9);
digit9 = train_data(:,label9);
train_data49 = [digit4, digit9];
labels_train49 = [4*ones(1,size(digit4,2)),...
    9*ones(1,size(digit9,2))]';
label4_test = find(labels_test == 4);
digit4_test = test_data(:,label4_test);
label9_test = find(labels_test == 9);
digit9_test = test_data(:,label9_test);
test_data49 = [digit4_test, digit9_test];
labels_test49 = [4*ones(1,size(digit4_test,2)),...
    9*ones(1,size(digit9_test,2))]';

%% difficult tree
tree_49 = fitctree(train_data49',labels_train49,'MaxNumSplits',...
    3,'CrossVal','on');
test_labels = predict(tree_49.Trained{1}, test_data49');
tree_49_acc = sum(labels_test49 == test_labels)/ size(test_labels,1);

%% difficult svm
Mdl_49 = fitcecoc(train_data49',labels_train49);
test_labels = predict(Mdl_49,test_data49');
svm_49_acc = sum(labels_test49 == test_labels)/size(test_labels,1);

function [U,S,V,w, vdigit1, vdigit2, vdigit3] = class3_trainer(digit1,digit
    n1 = size(digit1,2);
    n2 = size(digit2,2);
    n3 = size(digit3,2);
    [U,S,V] = svd([digit1 digit2 digit3],'econ');
    genres = S*V'; % projection onto principal components
    U = U(:,1:feature);
    digit11 = genres(1:feature,1:n1);
    digit22 = genres(1:feature,n1+1:n1+n2);
    digit33 = genres(1:feature,n1+n2+1:n1+n2+n3);

    mdigit1 = mean(digit11,2);
    mdigit2 = mean(digit22,2);
    mdigit3 = mean(digit33,2);
    mtotal = mean([mdigit1 mdigit2 mdigit3],2);

    Sw = 0; % within class variances
    for k = 1:n1
        Sw = Sw + (digit11(:,k)-mdigit1)*(digit11(:,k)-mdigit1)';
    end
    for k = 1:n2
        Sw = Sw + (digit22(:,k)-mdigit2)*(digit22(:,k)-mdigit2)';
    end
    for k = 1:n3
        Sw = Sw + (digit33(:,k)-mdigit3)*(digit33(:,k)-mdigit3)';
    end
    Sb = size(digit1,2)*(mdigit1-mtotal)*(mdigit1-mtotal)' +...
        size(digit2,2)*(mdigit2-mtotal)*(mdigit2-mtotal)'...
        + size(digit3,2)*(mdigit3-mtotal)*(mdigit3-mtotal)';

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

    vdigit1 = w'*digit11;
    vdigit2 = w'*digit22;
    vdigit3 = w'*digit33;

end

function [U,S,V,threshold,w,sortdog,sortcat] = digit_trainer(dog_wave,cat_w

    nd = size(dog_wave,2);
    nc = size(cat_wave,2);
    [U,S,V] = svd([dog_wave cat_wave],'econ');
    animals = S*V';
    U = U(:,1:feature); % Add this in
    dogs = animals(1:feature,1:nd);
    cats = animals(1:feature,nd+1:nd+nc);
    md = mean(dogs,2);
    mc = mean(cats,2);

    Sw = 0;
    for k=1:nd
        Sw = Sw + (dogs(:,k)-md)*(dogs(:,k)-md)';
    end
    for k=1:nc
        Sw = Sw + (cats(:,k)-mc)*(cats(:,k)-mc)';
    end
    Sb = (md-mc)*(md-mc)';

    [V2,D] = eig(Sb,Sw);
    [lambda,ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    vdog = w'*dogs;
    vcat = w'*cats;

    if mean(vdog)>mean(vcat)
        w = -w;
        vdog = -vdog;
        vcat = -vcat;
    end

    % Don't need plotting here
    sortdog = sort(vdog);
    sortcat = sort(vcat);
    t1 = length(sortdog);
    t2 = 1;
    while sortdog(t1)>sortcat(t2)
    t1 = t1-1;
    t2 = t2+1;
    end
    threshold = (sortdog(t1)+sortcat(t2))/2;

    % We don't need to plot results
end

function threshold = get_threshold(s1, s2)
    t1 = length(s2);
    t2 = 1;
    while s2(t1) > s1(t2)
        t1 = t1-1;
        t2 = t2+1;
    end
    threshold = (s2(t1)+s1(t2))/2;
end
```