Amath 482 Hw 4 - Classifying Images: LDA

Ofir Kedar

March 10, 2021

Abstract

In this project we are looking to utilize Linear Discriminant Analysis (LDA) to classify images of numbers into their respective labels. Labeling images of the number 1 as "1" images of the number 2 as "2" and so on. We do this using Principal Component Analysis (PCA) through a Singular Value Decomposition (SVD) implementation. We see that the first rank 154 approximation out of 784 are all that is needed for an accurate approximation containing 95% of the information stored in the images. This results in a success rate of 86% at image classification on our given set.

1 Introduction and Overview

Using a set of 60,000 images of numbers ranging from 0 to 9, we are asked to compute and analyse the Linear Discriminant Analysis (LDA) method as we attempt to classify (name/label) each number. Using the singular value spectrum we can determine what is an appropriate rank approximation of our data to get our methods running at adequate effectiveness and efficiency. after we do this we must analyze the conceptual meaning of the SVD resulting matrices. We are then asked to compare this classification method with two others, Support Vector Machines (SVM) and Classification Tree, in various ways, specifically, looking at their worst performance cases.

2 Theoretical Background

2.1 Linear Discriminant Analysis (LDA)

After using SVD to numerically separate each image in multidimensional space, LDA attempts to simplify our problem of assigning each multidimensional position with a single label. It does this by linearization of all data points onto a single line, represented by the 2norm (line of best fit) of the data. In other words we project the data onto the line of best fit, **w**. This is done using the distance between the means of the independent Scatter matrices that each represent a classification. [1]

$$\text{Between-Class Scatter Matrix} : S_B = \sum_{j=1}^{N} (\mu_j - \mu) * (\mu_j - \mu)^T \tag{1}$$

$$\text{Within-Class Scatter Matrix} : S_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j) * (x - \mu_j)^T \tag{2}$$

$$\text{Evaluating w} : S_B * w = \lambda * S_w * w \tag{3}$$

Here, $\boldsymbol{\mu}$ is the overall mean and $\boldsymbol{\mu_j}$ is the mean of each of the N classifications. We then solve for with the relation

2.2 Support Vector Machines (SVM) and Classification Trees

While not covered in lecture much, we do know that SVM and Classification Trees were considered state of the art classifiers until 2014, when they were overtaken by superior methods. these methods are easily implemented in MatLab and will be used to draw comparisons to the LDA method. [1]

3 Algorithm Implementation and Development

3.1 Singular Value Modes

The standard percent of content from an original data source for accurate data analysis is 95%. This mean that We can extract the first n modes of the our data matrix that leave us with at least 95% of the original information. Luckily, matrix **S** from the SVD we compute on out data tells us how much useful information is contained per rank approximation. we use this to extract the number of modes needed to accurately analyse this data.

Algorithm 1: Linear Discriminant Analysis

```
diagS = diag(S)$^2/sum(diag(S)^2) * 100$;
percent = 0;
i = 1;
while percent $< 95$ do
    percent = percent + diagS(i);
    i = i+1;
end while
feature = i-1;
```

3.2 Linear Discriminant Analysis (LDA)

To preform analysis on our images we convert each image to a column vector in a matrix where each column represents one of the images. Taking the SVD of the this matrix minus it's mean will give us the principal component matrices. We extract training data by projecting our desired rank of the principle components onto our mean normalized data. We then project out test data onto the 2norm, and normalize the training and testing data about the max value in their data sets. **fitcdiscr** and **predict** then preform the back end computations described in section 2.1, where the data is projected onto the 2norm. after that's done the we can compare the test results with the true labels of each immage and see the % effectivenes of the our LDA classification approach.

Algorithm 2: Linear Discriminant Analysis

```
mean_val = mean(mat_image,2);
m,n= size(mat$_i$$mage$);
X = mat_image - repmat(mean_val, 1, n);

[U,S,V] = svd(X/sqrt(n-1),'econ');
u = U; s = S; v = V;

projTrain = u(:,1:154)' * X;

W = mat_imageTest - repmat(mean_val, 1, size(mat_imageTest,2));
projTest = u(:,1:154)' * W;
projTrain = projTrain/ max(s(:));
projTest = projTest/ max(s(:));

Mdl = fitcdiscr(projTrain(:,1:10000)',labels(1:10000,:)', 'discrimType', 'linear');
test_label = predict(Mdl,projTest');

TestNum = size(test_label,1);
err = abs(test_label - labelsTest);
err = err ¿ 0;

errNum = sum(err);
sucRate2 = 1 - errNum/TestNum;
cm = confusionchart(labelsTest,test_label);
```

3.3 Support Vector Machines (SVM) and Classification Trees (CT)

[h] Computing SVM and CT was very similar to LDA, as the preparation steps were exactly the same as LDA. The differences here was one MatLab command for each method, which executed the computations accordingly to its assigned method. LDA using the **fitcdiscr** command, SVM using **fitcecoc**, and CT using **fitctree**.

4 Computational Results

4.1 Singular Value Spectrum

Although the singular values do have an exponential drop in in energy, the highest single value energy only contains under of 10% of the total information as seen in figure 1. This immediately tells us that we are looking at a fairy high rank approximation to gather a sufficient amount of information about out matrix. Using algorithm 1 we find that we need 154 modes. This is actually still a great decrease from the original 784 modes contained in the matrix.

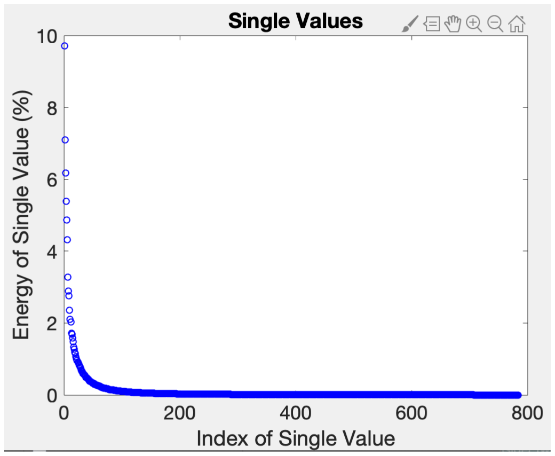


Figure 1: Single Value Spectrum

4.2 3D Scatter Matrix Representation

Using V modes 2, 3, and 5, we project our original data (mean reduced) onto our principal components. By plotting these projections of each mode separately we can see the variance in occupied space that each type of image lives in.

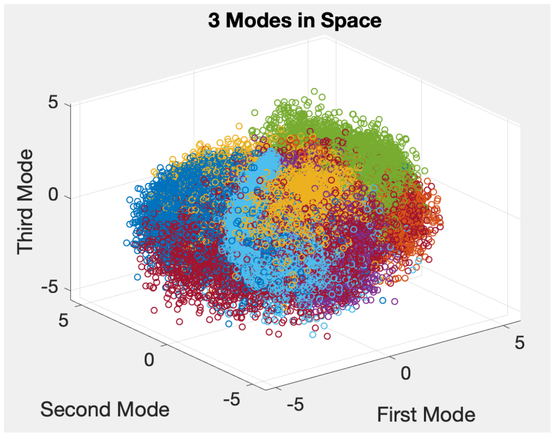
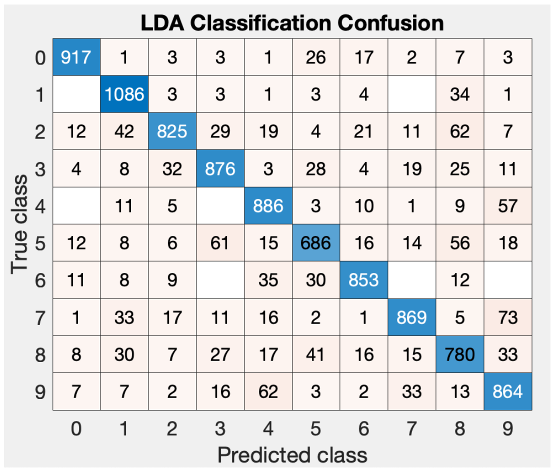


Figure 2: 3D Scatter Matrix

4.3 Digit Separation Confusion

After LDA is preformed on a data set of all 10 digits from 0 to 9, we can plot a confusion chart to see where our classifier is struggling most and least. It is a fair assumption to make that numbers that intuitively do not look alike will be mistaken for each other least, and visa versa. but to put this to the test we can simply find the largest numbers that are not on the diagonal and check. This is because the diagonal represents the number of images that were correctly labeled, and everything else is the number of miss classified images. We see several numbers that are almost never confused with each other, **1 and 0, 0 and 4, and 6 and 7, only being mixed up 1 once each in the entire labeling process**. On the other hand **when the true number was 2, LDA predicted it to be the number 8, 62 times**. The same was true **when the true number was 9, where LDA predicted it to be the number 4, 62 times**. This all from a test data set containing a total of 10,000 images.



True class \ Predicted class	0	1	2	3	4	5	6	7	8	9
0	917	1	3	3	1	26	17	2	7	3
1		1086	3	3	1	3	4		34	1
2	12	42	825	29	19	4	21	11	62	7
3	4	8	32	876	3	28	4	19	25	11
4		11	5		886	3	10	1	9	57
5	12	8	6	61	15	686	16	14	56	18
6	11	8	9		35	30	853		12	
7	1	33	17	11	16	2	1	869	5	73
8	8	30	7	27	17	41	16	15	780	33
9	7	7	2	16	62	3	2	33	13	864

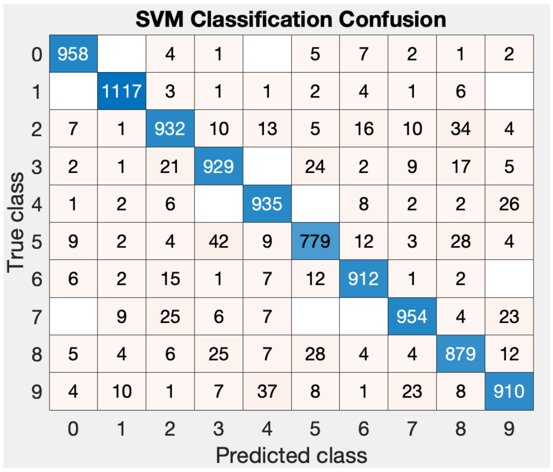
Figure 3: LDA Classification Confusion

4.4 Classification Method Performance

Content	LDA	SVM	CT (10 Splits)
Percent Accuracy (%)	86.42	93.05	51.01
Computation Time (sec)	1.62	13.40	12.57

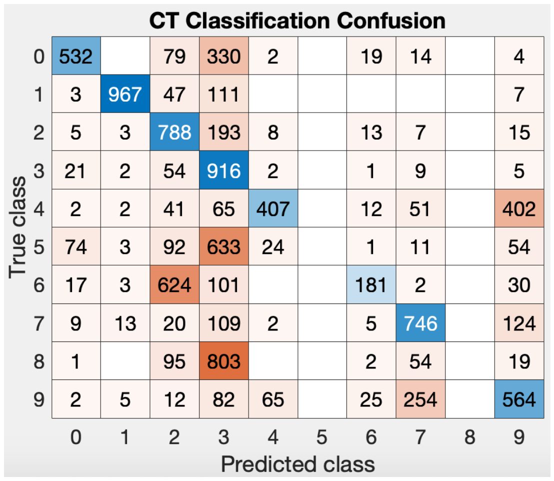
Table 1: Classification Methods in Contrast to True Classifications

Looking at the table above it's clear why in most cases LDA would be the method of choice over SVM, and definitely over CT. Although the accuracy of LDA is admittedly lower than that of SVM, it's computation time is magnitudes faster. Knowing the abilities of SVM and LDA it's almost embarrassing to the CT approach that given it's run time, it was able not only predict 1 to be 0 3 times, something that LDA and SVM both did 0 times, but it also somehow managed to miss label images of the number 5, every single time it came across it. Not a single 5 was labeled correctly. Even knowing CT's error rate from table 1 above, I find this surprising, and frankly quite humorous.



True class \ Predicted class	0	1	2	3	4	5	6	7	8	9
0	958		4	1		5	7	2	1	2
1		1117	3	1	1	2	4	1	6	
2	7	1	932	10	13	5	16	10	34	4
3	2	1	21	929		24	2	9	17	5
4	1	2	6		935		8	2	2	26
5	9	2	4	42	9	779	12	3	28	4
6	6	2	15	1	7	12	912	1	2	
7		9	25	6	7			954	4	23
8	5	4	6	25	7	28	4	4	879	12
9	4	10	1	7	37	8	1	23	8	910

Figure 4: SVM Classification Confusion



True class \ Predicted class	0	1	2	3	4	5	6	7	8	9
0	532		79	330	2		19	14		4
1	3	967	47	111						7
2	5	3	788	193	8		13	7		15
3	21	2	54	916	2		1	9		5
4	2	2	41	65	407		12	51		402
5	74	3	92	633	24		1	11		54
6	17	3	624	101			181	2		30
7	9	13	20	109	2		5	746		124
8	1		95	803			2	54		19
9	2	5	12	82	65		25	254		564

Figure 5: CT Classification Confusion

These alternative methods to LDA to struggle the most in different areas do generally maintain some of the same trends. By this I mean that the pairing that was most difficult for one method, may not be the most difficult for the other two, but may be the second or third most difficult. The same is true for easily separable numbers, for example 1 and 0 are barely mistaken for one another in any method. SVM most often mistakes 5 to be a 3, and CT mistakes most often 8 to be a 2, and appears to just think that 5's ARE 3's.

5 Summary and Conclusions

PCA through the use of SVD is a great tool to set up the ground works for several machine learning algorithms, specifically discussed in this report was mainly Linear Discriminant Analysis, and also Support Vector Machines and Classification Trees. In the resulting matrices U Σ and V, we can then categorize U to be the Principal Components of the over all data set containing every image, Σ to be the Singular Value energies, and V to be the set of confidence needed to reconstruct an image given the columns of U. We can use this information to create multidimensional scatter matrices of each image, and project these onto a line to be descretized on a single axis, then categorized by creating a numerical range value for each type of image in LDA. between the 3 described Machine learning methods, SVM had the highest rate of correct categorization of each image, coming in at 93.05% accurate. This is only 6.63% better than LDA, but took 8.27 times longer to compute. CT was out preformed in both computation time and in accuracy by orders of magnitude when given 10 splits; which is the number of categories it has. An interesting question that is then posed is if more modes could be added to LDA such that it contains for example 98% of information as apposed to 95%, and would then become more accurate but would still be faster than SVM. Furthermore, if it is possible to push this relation such that LDA becomes both faster and more accurate than SVM at a certain point.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

- `[images, labels] = mnist_parse('a', 'b')` takes ubyte files a and b and returns then in uint8 and double files respectively
- `images = im2double(images);` takes an image as input, and returns an image of class double
- `reshape(X,M,N) or reshape(X,[M,N])` returns the M-by-N matrix whose elements are taken columnwise from X. An error results if X does not have M*N elements.
- `scatter3(X,Y,Z,S,C)` displays colored circles at the locations specified by the vectors X,Y,Z
- `DISCR = fitcdiscr(X,Y)` accepts X as an N-by-P matrix of predictors with one row per observation and one column per predictor. Y is the response and is an array of N class labels.
- `OBJ = fitcecoc(TBL,Y)` fits K*(K-1)/2 binary SVM models using the "one versus one" encoding scheme for data in the table TBL and response Y. TBL contains the predictor variables. Y has K classes
- `TREE = fitctree(TBL,Y)` returns a classification decision tree for data in the table TBL and response Y. TBL contains the predictor variables.
- `LABEL = predict(MODEL,X)` returns predicted class labels LABEL and negated values of the average binary loss per class NEGLOSS for ECOC model MODEL and predictors X. X must be a table if MODEL was originally trained on a table, or a numeric matrix if MODEL was originally trained on a matrix
- `cm = confusionchart(m)` plots a confusion matrix and returns a ConfusionMatrixChart object. The matrix m must be a valid confusion matrix, that is, m must be a square matrix and its elements must be positive integers. The element m(i,j) is the number of times an observation in the ith true class was predicted to be in the jth class. Each colored cell of the chart corresponds to a single element of the confusion matrix.
- [Max_ave, Max_index] = max(abs(ave(:))) returns the max absolute value in matrix ave and it's index location
- B = repmat(A) creates a large matrix B the same size as A, tiling of copies of A. If A is a matrix, the size of B is [size(A,1)*M, size(A,2)*N].
- [U,S,V] = svd(X) produces a diagonal matrix S, of the same dimension as X and with non-negative diagonal elements in decreasing order, and unitary matrices U and V so that X = U*S*V'.

Appendix B MATLAB Code

```matlab
close all; clear all; clc
%%
[images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
[imagesTest, labelsTest] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');



%%

images = im2double(images);
[m,n,k] = size(images);



%%
for i = 1:k
    mat_image(:,i) = reshape(images(:,:,i),m*n,1);
end

imagesTest = im2double(imagesTest);
[m,n,k] = size(imagesTest);

for i = 1:k
    mat_imageTest(:,i) = reshape(imagesTest(:,:,i),m*n,1);
end

mat_image = im2double(mat_image);
mat_imageTest = im2double(mat_imageTest);

mat0 = mat_image(:,labels == 0);
mat1 = mat_image(:,labels == 1);
mat2 = mat_image(:,labels == 2);
mat3 = mat_image(:,labels == 3);
mat4 = mat_image(:,labels == 4);
mat5 = mat_image(:,labels == 5);
mat6 = mat_image(:,labels == 6);
mat7 = mat_image(:,labels == 7);
mat8 = mat_image(:,labels == 8);
mat9 = mat_image(:,labels == 9);

mean_val = mean(mat_image,2);
[m,n] = size(mat_image);

X = mat_image - repmat(mean_val, 1, n);
```

Listing 1:

```
%%
[U,S,V] = svd(X/sqrt(n-1),'econ');
u = U;
s = S;
v = V;

[Um,Sm,Vm] = svd(mat_image,'econ');
um = Um;
sm = Sm;
vm = Vm;


%%
plot((diag(S).^2/sum(diag(S).^2)*100), 'ob')
set(gca,'Fontsize',18)
title('Single Values')
xlabel('Index of Single Value')
ylabel('Energy of Single Value (%)')


%%
% must run all section up to this point to work
diagS = diag(S).^2/sum(diag(S).^2)*100;
percent = 0;
i = 1;
while percent < 95
    percent = percent + diagS(i);
    i = i+1;
end
feature = i-1;


%%
proj = U' * X;

scatter3(proj(1,labels==2), proj(2,labels==2), proj(3,labels==2), 'ko');
hold on
scatter3(proj(1,labels==3), proj(2,labels==3), proj(3,labels==3), 'ro');
hold on
scatter3(proj(1,labels==5), proj(2,labels==5), proj(3,labels==5), 'bo');

set(gca,'Fontsize',18)
title('Modes in 3D Space (units insignificant)')
xlabel('Space')
ylabel('Space')
zlabel('Space')
```

Listing 2:

```
%%
proj = U' * X;

for i = 1:9
    scatter3(proj(2,labels==i), proj(3,labels==i), proj(5,labels==i), 'o');
    hold on
end


%%
proj = U' * X;
scatter3(proj(1,labels==7), proj(2,labels==7), proj(3,labels==7), 'ko');
hold on
scatter3(proj(1,labels==3), proj(2,labels==3), proj(3,labels==3), 'ro');
hold on



%% Calculate scatter matrices

[U,S,V,threshold,w,sort1,sort2] = dc_trainer(mat1,mat2,feature);

mat1Test = mat_imageTest(:,labelsTest == 1);
mat2Test = mat_imageTest(:,labelsTest == 2);
TestSet = [mat1Test mat2Test];

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

ResVec = (pval>threshold);

a = zeros(1,size(mat1Test, 2));
b = ones(1, size(mat2Test, 2));

hiddenlabels = [a b];

err = abs(ResVec - hiddenlabels);
err = err > 0;

errNum = sum(err);
sucRate1 = 1 - errNum/TestNum;
```

Listing 3:

```
%%
%           LDA 10 digits

projTrain = u(:,1:154)' * X;

W = mat_imageTest - repmat(mean_val, 1, size(mat_imageTest,2));
projTest = u(:,1:154)' * W;
projTrain = projTrain/ max(s(:));
projTest = projTest/ max(s(:));

tic
Mdl = fitcdiscr(projTrain(:,1:60000)',labels(1:60000,:)', 'discrimType', 'linear');
test_label = predict(Mdl,projTest');
toc

TestNum = size(test_label,1);
err = abs(test_label - labelsTest);
err = err > 0;

errNum = sum(err);
sucRate2 = 1 - errNum/TestNum;
figure(1);
cm = confusionchart(labelsTest,test_label);
title('LDA Classification Confusion')
set(gca,'Fontsize',18)


%%
%           SVM classifier with training data, labels and test set

projTrain = u(:,1:154)' * X;
W = mat_imageTest - repmat(mean_val, 1, size(mat_imageTest,2));
projTest = u(:,1:154)' * W;
projTrain = projTrain/ max(s(:));
projTest = projTest/ max(s(:));

tic
Mdl = fitcecoc(projTrain(:,1:10000)',labels(1:10000,:)');
test_label = predict(Mdl,projTest');
toc

TestNum = size(test_label,1);
err = abs(test_label - labelsTest);
err = err > 0;

errNum = sum(err);
sucRate3 = 1 - errNum/TestNum;
figure(2);
cm = confusionchart(labelsTest,test_label);
title('SVM Classification Confusion')
set(gca,'Fontsize',18)
```

Listing 4:

```
%%

%           classification tree on fisheriris data

projTrain = u(:,1:154)' * X;
W = mat_imageTest - repmat(mean_val, 1, size(mat_imageTest,2));
projTest = u(:,1:154)' * W;
projTrain = projTrain/ max(s(:));
projTest = projTest/ max(s(:));

tic
Mdl = fitctree(projTrain',labels','MaxNumSplits',10);
%view(Mdl ,'Mode','graph');
test_label = predict(Mdl,projTest');
toc


TestNum = size(test_label,1);
err = abs(test_label - labelsTest);
err = err > 0;

errNum = sum(err);
sucRate4 = 1 - errNum/TestNum;
figure(3);
cm = confusionchart(labelsTest,test_label);
title('CT Classification Confusion')
set(gca,'Fontsize',18)
```

Listing 5:

```
function [images, labels] = mnist_parse(path_to_digits, path_to_labels)
    % The function is curtesy of stackoverflow user rayryeng from Sept. 20,

    % 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
```

Listing 6:

```matlab
function [U,S,V,threshold,w,sort1,sort2] = dc_trainer(data1,dta2,feature)

    n1 = size(data1,2);
    n2 = size(dta2,2);
    [U,S,V] = svd([data1 dta2],'econ');
    projection = S*V';
    U = U(:,1:feature); % Add this in
    data1 = projection(1:feature,1:n1);
    data2 = projection(1:feature,n1+1:n1+n2);
    m1 = mean(data1,2);
    m2 = mean(data2,2);

    Sw = 0;
    for k=1:n1
        Sw = Sw + (data1(:,k)-m1)*(data1(:,k)-m1)';
    end
    for k=1:n2
        Sw = Sw + (data2(:,k)-m2)*(data2(:,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'*data1;
    v2 = w'*data2;

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

    % Don't need plotting here
    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;

    % We don't need to plot results
end
```

Listing 7: