Classification Methods for Digit Images

Johnny Van

March 3, 2021

Abstract

We use linear discriminant analysis, SVM, and decision tree classifiers to "train" the computer such that when given an image of a digit from 0-9, the computer will be able to accurately classify the image with the correct digit label. This paper compares the three methods in terms of their accuracy in classification and how principal component analysis can complement these methods.

1 Introduction and Overview

When given an image, is there a way for computers to classify correctly what that image is? In this project we are given tens of thousands of images of handwritten digits from 0-9, and our objective is to train our computer to be able to classify each image with their correct digit label using linear discriminant analysis and two other methods called support vector machines(SVM) and decision tree classifiers. We create these classifiers and compare their performances to one another in terms of their accuracy. To efficiently create these classifiers, we employ the method of Principal Component Analysis to reduce run times while maintaining accuracy.

2 Theoretical Background

2.1 Background of Principal Component Analysis

Imagine that we have 4 row vectors each containing data about students' grades. Let's put these row vectors into a matrix called X. Then we subtract each row of X by its respective mean, and lets still define this as the matrix X. We compute all the variances and covariances between the rows of X using the equation.

$$C_X = \frac{1}{n-1} XX^T = \begin{bmatrix} \sigma_a^2 & \sigma_{ab}^2 & \sigma_{ac}^2 & \sigma_{ad}^2 \\ \sigma_{ba}^2 & \sigma_b^2 & \sigma_{bc}^2 & \sigma_{bd}^2 \\ \sigma_{ca}^2 & \sigma_{cb}^2 & \sigma_c^2 & \sigma_{cd}^2 \\ \sigma_{da}^2 & \sigma_{db}^2 & \sigma_{dc}^2 & \sigma_d^2 \end{bmatrix}. \tag{1}$$

The matrix, C_X, is called the covariance matrix which as you can see is square and symmetric. Next, we diagonalize our covariance matrix so that all of the off-diagonal elements in our matrix is zero.

$$C_X = V\Lambda V^{-1} \tag{2}$$

The basis of the eigenvectors contained in matrix V are the **principal components** which are orthogonal, indicating that they are uncorrelated. This is because C_X is a symmetric matrix, which causes its eigenvalues to be real and thus their eigenvectors will be orthogonal. Then the diagonal of the matrix, Λ are the eigenvalues of C_X, which represents the magnitude of the variances of these newly formed variables.

2.2 Connection to Singular Value Decomposition

The prinicpal component analysis has strong connections to the singular value decompsition. Let's say X is our data matrix. Then the SVD of matrix A is connected to the eigenvalue decomposition of AA^T. Consider,

$$A = \frac{1}{\sqrt{n-1}} X \tag{3}$$

Then it is known that,

$$C_X = \frac{1}{n-1} XX^T = AA^T \tag{4}$$

And from our previous work in the course,

$$C_X = AA^T = U\Sigma^2 U^T \tag{5}$$

After we get the matrices, U, Σ, V from SVD of A, then it is understood that Σ^2 contain the eigenvalues of the covariance matrix, and that the basis of eigenvectors or the columns in matrix U are the principal components, and V contains the coefficients of each principal components that is used to rebuilt each of the samples/image. So each row of V is set of coefficients needed to rebuilt each sample/image using columns of U. Then to project the data into a new basis of the principal components, you would use the equation:

$$Y = U^T X \tag{6}$$

2.3 Linear Discriminant Analysis

The goal of Linear Discriminant Analysis is to find an ideal projection that maximizes the distance between the inter-class data while minimizing the amount of this intra-class data.

Consider an example in which we are implementing LDA on two data sets. Let's say that μ_1 and μ_2 are the means of each data set respectively. From the textbook, [1], the first step in LDA is to define the **between-class scatter matrix** which is shown as Equation (7).

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

This matrix represents a measure of variance between the two data sets (between the means). Next we define the **within-class scatter matrix** by Equation (8).

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

This would represent the variance within each data set. The point of defining these two matrices is to find the vector w such that

$$\mathbf{w} = argmax(\frac{w^T S_B w}{w^T S_w w}) \tag{9}$$

This seems tough to compute; however it is known that the vector **w** that maximizes Equation (9) is simply the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem given by Equation (10).

$$S_B w = \lambda S_w w \tag{10}$$

Using Matlab, finding the vector w is rather simple, and once we have found w, we can choose the cutoff between the two data sets as a threshold for decision making.

2.4 Linear Discriminant Analysis for More Groups

Of course, using LDA for two datasets is useful for a wide variety of situations, but there are times when we need to classify between more than two groups. Luckily, adapting LDA to more than two groups isn't difficult. We simply have to change our scatter matrices, S_B to Equation (11),

$$\mathbf{S}_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu_1)^T \tag{11}$$

where μ is the overall mean and μ_j is the mean of the j-th group/data set. The within-class scatter matrix is then defined by Equation(12).

$$\mathbf{S}_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{12}$$

Then we find our w vector using the same equation given by Equation(10) and find a cutoff value.

2.5 Support Vector Machines (SVM)

Support Vector Machine or SVM is a supervised machine-learning algorithm used for mainly classification problems. The difference between SVM and our linear discriminant analysis is that SVM focuses on the data that are hard to classify while LDA focuses on all of the data. More specifically, SVM finds linear separators that separates the classes with the least error while LDA computes the mean of each class and finds the projection that maximizes the distance between the means.

2.6 Decision Tree Classifiers

The Decision Tree Classifier is also an outdated classification method. The basis of this classifier is a binary tree in which the test data is classified by transversing through this tree. Each sample is given a particular value and at each branch node, there is a conditional that either leads to you to either the right or left node based on your value. This keeps going on until you have reached a leaf node or the bottom of the tree and you are given your predicted class label.

3 Algorithm Implementation and Development

We are given four files of data. One of the files contain the training data, which consists of 60,000 28x28 pixel images of digits from 0-9, and another file contains the labels of each of these images. Similarly, we have a file containing the test data, consisting of 10,000 28 x 28 digit images, and another file containing their labels. First we reshape the data such that each column of the data matrix represents 1 image. So the training data will go from 28 x 28 x 60,000 matrix to 784 x 60,000 and test data will be 784 x 10,000.

3.1 Algorithm 1: Principal Component Analysis on Training Data

First we standardize our training data by subtracting the rows by their means, then divide by $\sqrt{n-1}$ where n is number of images in the data. Next, we compute the SVD decomposition of the standardized matrix which gives you the U,S,V matrices. U contains the principal components, S^2 contains the eigenvalues of each principal component which determines their individual importance to the data, and V contains coefficients that are used to rebuild any of the 60,000 images using the principle components. To project the training data using equation (6), we multiply X by only a certain number of columns of U, the number determined by how many singular values you need for 95% of the variance.

3.2 Algorithm 2: Linear Discriminant Analysis on 2 Digits

Now that we know how many PCA modes are required to capture at least 95% of the variance of the original data, we can start on the Linear Discriminant Analysis. Once we have build the LDA, it will be able to classify between two digits, such as if the image is a 4 or a 7.

1. From the training data, create two matrices each containing the data of only a single digit. These two digits are what the LDA classifier will distinguish between and we refer to each digit label/group as a class. Calculate how many images are in each matrix.

2. Concatenate the matrices of the data so they are side by side. Take the SVD of this resulting matrix and project the data using Equation (6). From the projection, take the first 154 rows (determined by number of singular values needed for 95% energy) and the number of columns determined by how many images were in class 1. Let's call this data set 1. Then for the second class, take the first 154 rows of the projection, and the remaining columns, and call this data set 2.

3. Find the means of the rows of each data set individually, then calculate the S_B and S_w scatter matrices by equation (7) and (8) respectively.

4. Using equation (10), find the vector w. Then multiply the tranpose of w by the data of the data matrix of class 1. Then do the same for class 2.

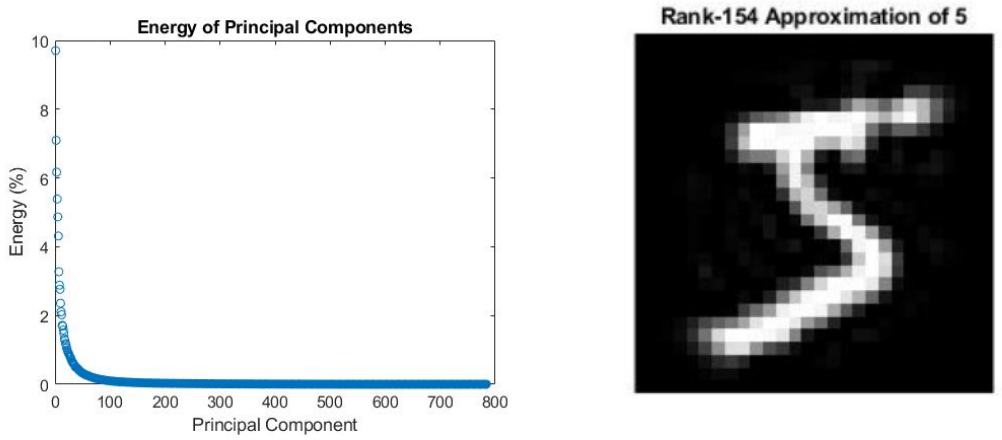

Figure 1: The left plot shows the energy percentages of each of the 784 principal components. The right image is the rank 154 approximation of an image of a 5.

5. Using these two resulting vectors, find the threshold or value that best separates the two classes, such that anything below the threshold is class 1, and anything above is class 2. Once you have the threshold, we compare our results to the accurate labels and find our accuracy percentage of our 2-digit LDA.

3.3 Algorithm 3: Data Preparation for Built-in functions

For 3-digit LDA, SVM, and decision tree classifiers, rather than creating them by hand, we will simply use the MATLAB built-in functions to create these classifiers and then use them on the test data. Rather than putting a 784 by 60,000 matrix in as inputs to these functions, it will be much more efficient to use it's PCA projection instead,

1. Follow Algorithm 1 in order to get the U,S, and V matrices from the SVD decomposition of the standardized training data.
2. Project the training data onto a certain number of columns of U such that the projection of training data has fewest dimensions possible while still having 95% of variance.
3. With the test data, subtract each row of the test data using the row means of the training data. Then project the standardized test data onto the first 154 columns of the same U matrix to get the projection of test data on the same principal components of training data.

Note: We built our LDA classifier based on the projected training data on the first 154 principal components of the SVD decomposition of the training data. In order to actually use this LDA classifier on the test data, we must project the test data on the same space or principal components as we did the training data.

3.4 Using MATLAB functions for LDA, SVM, and Decision Tree

From the section above, we created the projections of the training data and the test data. When we are creating the different classifiers using the built-in functions, the inputs to these functions will be the projected training data.. If we want to build classifiers for only a few digits, we put in a subset of the projected training data.

1. To create a LDA classifier capable of classifying more than 2 digits, use the `fitcsvm` command.
2. To create the SVM classifier, use the fitcvsm() command for a 2-digit SVM classifier, and use fitcecoc() for creating SVM classifiers that can classify more than 2 digits.

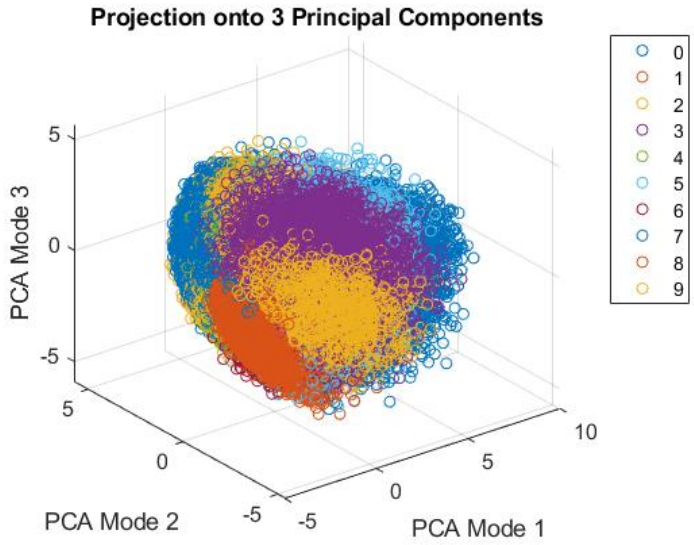

Figure 2: A 3-D plot showing the projection of training data on 3 modes colored by their digit labels.

3. To create a decision tree classifier, use the `fitctree` command for a tree classifier that is able to classify any number of digits.

After creating these classifiers, we use the `predict` function to use our classifiers on the projected test data in order to get a vector representing the predicted labels. Then we simply compare the predicted labels to the actual labels of the test data to find the accuracy percentage of these classifiers.

4 Computational Results

4.1 Principal Component Analysis on Training Data

Using Algorithm 1, we performed PCA on the training data and in Figure 1, the singular values plot is shown. We need 154 principal components or modes in order for our projection to contain at least 95% of the data's original variance. Instead of 784 pixels, we can now classify images based on only 154 pixels. In Figure 1, the right image is the rank 154 approximation of an image of the digit 5 which basically looks identical to the original image which is not surprising.

In Figure 2, the projection of the training data on only three modes is shown and each point is colored by their digit label. Clusters that are mixed with each other indicate that it will be difficult for classifiers to distinguish between the two digit images. The three PCA modes I chose are the 1st, 2nd, and 3rd modes because the clusters were most separated when projected on these three.

Accuracy of Various Classification Methods (%)			
Digits Classified	Linear Discriminant Analysis	SupportVector Machine(SVM)	Decision Tree
1,2	98.75 %	99.49 %	98.52 %
3,5	96.48 %	96.58 %	91.22 %
4,9	95.73 %	97.04 %	89.85 %
1,0	99.62 %	99.91 %	99.55 %
3,5,8	98.74 %	94.02 %	86.27 %
1,7,2	99.51 %	98.40 %	96.57 %
10 digits (test)	87.43 %	94.80 %	83.87 %
10 digits(training)	87.05 %	95.61 %	96.05 %

Table 1: A table containing the accuracy of LDA, SVM, Decision Tree classifiers used on test data

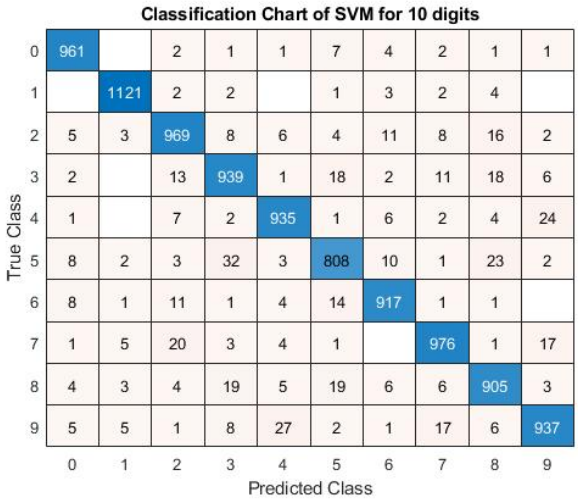

Figure 3: A confusion matrix showing the mislabeled images made by 10-digit SVM classifier on test data.

4.2 Linear Discriminant Analysis

Using Algorithm 2 with the help of the `dc.trainer` function, I built 2 and 3 digit LDA classifiers that performed very well as described in Table 1. The two digits in the data set that seem to be the most difficult for 2-digit LDA classifier to separate between are 4 and 9 since it has the lowest accuracy compared to all other pairs of digits. The easiest pair of digits to separate is the pair of 1 and 0. When I used the 10-digit LDA classifier on the training data instead of the test data, it's accuracy was about the same, showing that the size of the data set doesn't affect the accuracy of the LDA classifier significantly.

4.3 Support Vector Machine (SVM)

We created 2-digit and 3-digit SVM classifiers that usually performed better than LDA and decision tree as shown in Table 1, but most importantly, I built a SVM classifier that was able to classify all 10 digits! It had a very high accuracy of about 94% when used on the test data, but it was beat by decision tree when used on training data. Then Figure 3 shows exactly what errors were made. The pair of digits that had the most amount of errors was 4 and 9, while the pair of digits with no errors was 1 and 0. When used on either training or test data, it's accuracy is about the same which signifies that SVM does well with small and large data sets.

4.4 Decision Tree Classifier

We create a variety of decision tree classifiers which were used on the test data with results shown in Table 1. The accuracy percentages for 2 and 3 digit decision tree classifiers performed well but definitely worse than LDA and SVM. For classifying 10 digits, decision tree does the worst when used on test data. However, if we refer to Table 2, when decision tree is used on training data, decision tree beats the other two methods in almost all cases which is surprising. This must mean decision tree performs better on larger sample data sets. From Figure 6, we can also see that it has the hardest time separating 4 and 9 and easiest time separating 1 and 0.

5 Summary and Conclusions

Our results in Table 1 shows the accuracy of each methods across different cases. When used on test data, SVM did the best in separating the hardest and easiest pair of digits to separate. When used on the training data, SVM does the best with the easiest pair, but decision tree does best with the hardest pair. Overall, when used on the test data, SVM has the best performance, and when used on the training data, the decision tree has the best performance. These three classification methods have served their use in previous two decades, but today, neural networks are the standard classification method. Although neural networks are faster and much more accurate, these three methods still delivered reasonable results.

References

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

Appendix A Supplemental Figures

Accuracy of Various Classification Methods (%)			
Digits Classified	Linear Discriminant Analysis	SupportVector Machine(SVM)	Decision Tree
3,5	95.80 %	96.77 %	98.96 %
4,9	96.02 %	97.31 %	98.66 %
1,0	99.53 %	99.98 %	99.95 %
3,5,8	92.01 %	94.78 %	97.68 %
1,7,2	97.44 %	99.04 %	99.45 %
10 digits(training)	87.05 %	95.61 %	96.05 %

Table 2: A table containing the accuracy of LDA, SVM, Decision Tree classifiers used on training data

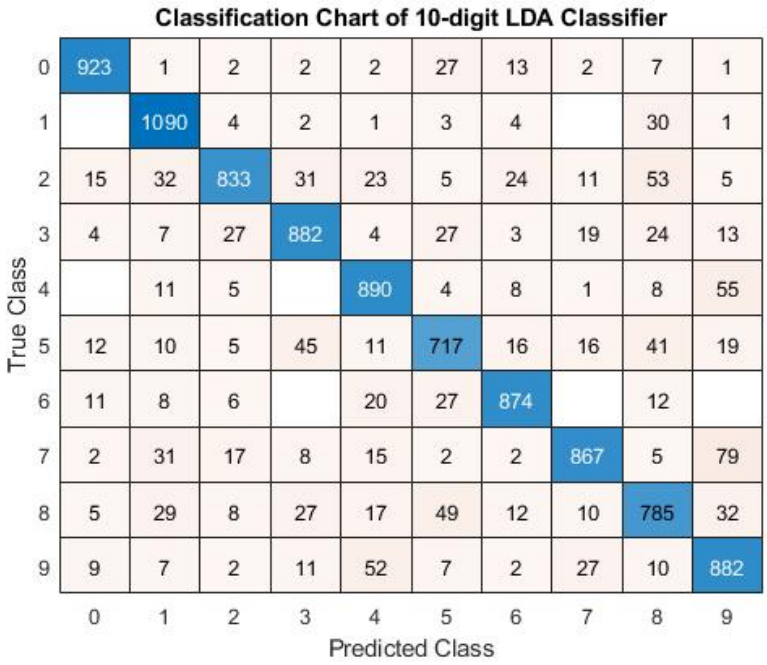

Figure 4: A confusion matrix showing the mislabeled images made by 10-digit LDA classifier when used on the test data which consists of 10,000 images.

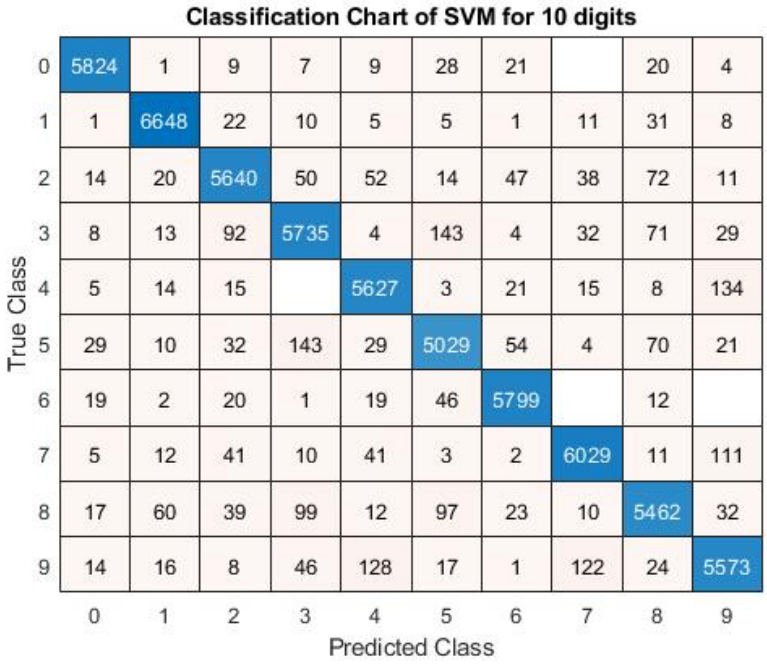

Figure 5: A confusion matrix showing the mislabeled images made by 10-digit SVM classifier when used on the training data which consists of 60,000 images.

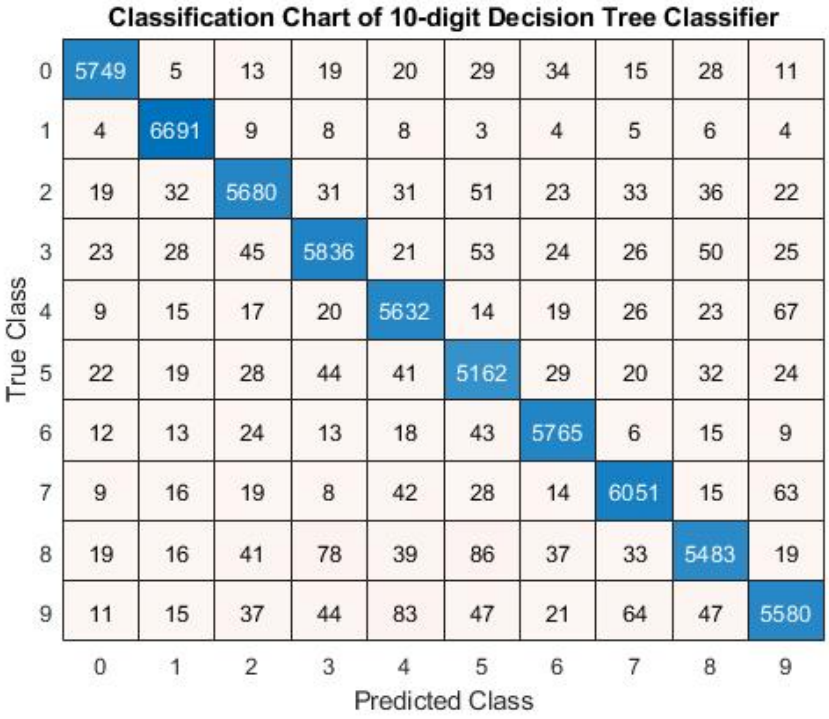

Figure 6: A confusion matrix showing the mislabeled images made by 10-digit Decision Tree classifier when used on the training data which consists of 60,000 images.

Appendix B MATLAB Functions

These are the important MATLAB functions we used in this project along with their implementation explaination.

- `y = reshape(X, [ m n ])` reshapes X to the size defined by m and n. So [m n] represents reshaping X to be a matrix with m rows and n columns.
- `[images, labels] = mnist.parse('images.file', 'label.file')` returns the data contained in the given files in matrix form. Used specifically for files taken from the MNIST database.
- `imshow(X)` returns the frame of the video, X, as a figure.
- `[m,n] = size(X)` stores the number of rows of X as m and the number of columns of X as n.
- `repmat(mean, 1, n)` returns an array containing n copies of mean in the row dimension specified by the 1.
- `[U,S,V] = svd(A, 'econ')` performs the singular value decomposition of matrix A and stores the decompsition as the three matrices U,S,V such that A = U*S*V'. The 'econ' means that the command returns an economy-size decomposition of A.
- `Mdl = fitcdiscr(X,Y, 'discrimType', 'linear')` returns a fitted discriminant analysis model based on the input variables contained in X, and output variables such as labels in Y. The type of discriminant analysis can be 'linear', 'pseudolinear', and 'diaglinear'.
- `Mdl = fitcsvm(X,Y)` returns a trained support vector machine (SVM) model for one-class or two-class classification based on the sample data in X, and their class lables in Y.
- `Mdl = fitcecoc(xtrain,labels)` returns a trained support vector machine (SVM) model for multi-class classification based on the sample data in X, and their class lables in Y.
- `Mdl = fitctree(X,Y, 'MaxNumSplits', 10)` returns a fitted binary classification decision tree based on the sample data from X, and their class labels in Y. 'MaxNumSplit' refers to how many splits the binary tree can have. Turn this input to 'off' if you want the highest accuracy MATLAB can give with a decision tree.
- `predicted.labels = predict(Md1, X)` returns a vector of the predicted class labels for the data given by X based on the trained classification model given by Mdl.
- `[U,S,V,threshold,w,sortOne,sortTwo] = dc.trainer(data 1,data 2,feature)` returns the matrices U,S,V from the SVD composition of the joined matrix of data 1 and data 2. Also returns the threshold and the vector w which comes from LDA of the two data sets.

Appendix C MATLAB Code

```
%% Load in training and test data from MNIST database.
clear all; close all; clc;
[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');

images = im2double(images);
[m,n,k] = size(images);
for i = 1:k
    rawData(:,i) = reshape(images(:,:,i), m*n,1);
end

[test_images, test_labels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');

test_images = im2double(test_images);
[m,n,k] = size(test_images);
for i = 1:k
    testData(:,i) = reshape(test_images(:,:,i), m*n,1);
end

%% Principal Component Analysis
close all;
[m,n] = size(rawData);
mn = mean(rawData, 2);
X = rawData - repmat(mn, 1, n); % Standardize the training data
A = X/sqrt(n-1); % Define A by equation (3)

[U,S,V] = svd(A,'econ'); % SVD of standardized training data

% Plot of the energy contained in each principal component
figure(1)
plot((diag(S).^2)/(sum(diag(S).^2))*100, 'o');
set(gca, 'Fontsize', 12);
ylabel("Energy (%)"); title("Energy of Principal Components");
xlabel("Principal Component");


% Calculating the number of PCA modes needed to capture 95% of variance in training data.
diagonal = (diag(S).^2)/(sum(diag(S).^2))*100;
sum_Sing = 0;
index = 0;

for i = 1:784
    sum_Sing = sum_Sing + diagonal(i);
    index = index + 1;
    if (sum_Sing > 90.00)
        break
    end
end
```

Listing 1: Principal Component Analysis of Training Data.

```
% Projection of training data onto 154 modes.
proj = U(:, 1:154)'*X;

% Plotting 3D cluster plot, each color representing a different digit.
figure(2)
for i = 0:9
   scatter3(proj(1,labels==i), proj(2,labels==i), proj(3,labels==i));
    hold on
end

set(gca, 'Fontsize', 12);
xlabel("PCA Mode 1"); ylabel("PCA Mode 2"); zlabel("PCA Mode 3");
title("Projection onto 3 Principal Components");
legend("0","1", "2", "3", "4", "5", "6", "7", "8", "9");

%% Separating the images into their own matrix.
% To build LDA for a different pair of digits, simply change these two matrices.
one_matrix = rawData(:, labels == 1);
two_matrix = rawData(:, labels == 2);

%% Constructing LDA classifier between two digits. Feature represents how many PCA modes
% we needed for 95% of variance in training data.

feature = 154;
[U,S,V,threshold,w,sortOne,sortTwo] = dc_trainer(one_matrix,two_matrix,feature);

% From the test data, creates a new matrix containing only the two digits we built the
%  classifier for. Also creates a new matrix containing the labels for the filterData matrix.
filterData = [];
filterLabel = [];

for i = 1:k
    if (test_labels(i) == 1)
        filterData = [filterData testData(:,i)];
        filterLabel = [filterLabel 0];
    end
    if(test_labels(i) == 2)
        filterData = [filterData testData(:,i)];
        filterLabel = [filterLabel 1];
    end
end

%% Calculates the error of the LDA classifier by comparing the predicted labels
% to the actual labels of the images.
TestNum = size(filterData,2);
TestMat = U'*(filterData); % PCA projection
pval = w'*TestMat;

ResVec = (pval>threshold);

err = abs(ResVec - filterLabel);
errNum = sum(err);
sucRate = 1 - errNum/TestNum %% Accuracy Percentage of LDA
```

Listing 2: Linear Discriminant Analysis of 2 digits

```
function [U,S,V,threshold,w,sortData1,sortData2] = dc_trainer(data1,data2,feature)

    nd = size(data1,2);
    nc = size(data2,2);
    [U,S,V] = svd([data1 data2],'econ');


    total = S*V';
    U = U(:,1:feature); % Add this in
    data1 = total(1:feature,1:nd); % Feature represents how many Principal components
                                   % you need for 95% of the energy of original data.
    data2 = total(1:feature,nd+1:nd+nc);
    md = mean(data1,2);
    mc = mean(data2,2);

    Sw = 0;
    for k=1:nd
        Sw = Sw + (data1(:,k)-md)*(data1(:,k)-md)';
    end
    for k=1:nc
        Sw = Sw + (data2(:,k)-mc)*(data2(:,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);
    vData1 = w'*data1;
    vData2 = w'*data2;

    if mean(vData1)>mean(vData2)
        w = -w;
        vData1 = -vData1;
        vData2 = -vData2;
    end

    % Don't need plotting here
    sortData1 = sort(vData1);
    sortData2 = sort(vData2);
    t1 = length(sortData1);
    t2 = 1;
    while sortData1(t1)>sortData2(t2)
    t1 = t1-1;
    t2 = t2+1;
    end
    threshold = (sortData1(t1)+sortData2(t2))/2;

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

Listing 3: The function used to perform LDA on the two classes / data sets.

```
%% For LDA-3 digits, SVM, and Decision Tree Classier, we use built in Matlab functions.
% The below code is Step 1 for each of the classication methods.


%% Loading in Data
clear all; close all; clc;

[images, 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');

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

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

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

for i = 1:k
    testData(:,i) = reshape(test_images(:,:,i), m*n,1);
end

%% PCA Projection

[m,n] = size(rawData);
mn = mean(rawData, 2);
X = rawData - repmat(mn, 1, n);
A = X/sqrt(n-1);

[U,S,V] = svd(A,'econ');

%% Project onto 154 PCA modes with the training data
projection_training = U(:, 1:154)'*X;

% Subtract each row of the test data with the means of the rows from training data.
[m, n] = size(testData);
test_avg = testData - repmat(mn, 1, n);

% Project onto the same 154 PCA modes with the standardized test data.
projection_test = U(:, 1:154)'*test_avg;
```

Listing 4: Step 1 Code for LDA-3 digits, SVM, and Decision Tree

```
%% Follow Step 1 on page 13.

%% This is the Step 2 code for Linear Discriminant Analysis (LDA) of 3 digits.
% We use the builtin Matlab function, fitcdiscr, and predict.




%% The three digits chosen for LDA

%Training Data
xtrain = projection_training(:, labels == 1 | labels == 7| labels == 2);
label = labels(labels == 1 | labels == 7 | labels == 2, :);
label = label';

% Test Data
testNum = size(xtrain, 2);
proj_test = projection_test(:, test_labels == 1 | test_labels == 7 | test_labels == 2);
true_label = test_labels(test_labels == 1 | test_labels == 7 | test_labels == 2, :);


%% LDA with 3 digits
clc;

Md1 = fitcdiscr(xtrain', label, 'discrimType', 'diaglinear');
approx_label = predict(Md1, proj_test');


err = abs(approx_label - true_label);
errTrue = err > 0;
errNum = sum(errTrue);
sucRate = 1 - errNum/testNum %% Accuracy percentage
```

Listing 5: Linear Discriminant Analysis for more than 2 digits.

```
%% Follow Step 1 on page 13.
%% This is the Step 2 code for Support Vector Machine (SVM) for 2 digits.
% We use the builtin Matlab function, fitcecsvm, and predict.



%% The two digits being chosen for SVM - Just change the two digits to what u want.

% Test Data and labels
proj_test = projection_test(:, test_labels == 1 | test_labels == 2);
true_label = test_labels(test_labels == 1 | test_labels == 2, :);

% Training Data and label
label = labels(labels == 1 | labels == 2, :);
label = label';
xtrain = projection_training(:, labels == 1 | labels == 2);

%%

% SVM classifier with training data, labels and test set
Mdl = fitcsvm(xtrain',label);

%%

testlabels = predict(Mdl,proj_test');

testNum = size(testlabels,1);
err = abs(testlabels - true_label);
err = err > 0;
errNum = sum(err);
sucRate = 1 - errNum/testNum %% Accuracy percentage
```

Listing 6: Support Vector Machine (SVM) for 2 digits

```
%% Follow Step 1 on page 13.

%% This is the Step 2 code for Support Vector Machine (SVM) for 10 digits.
% We use the builtin Matlab function, fitcecoc, and predict.



%% The data being used in SVM, since 10 digits, we are using all of it.

xtrain = projection_training;
label = labels';

proj_test = projection_test;
true_label = test_labels;

%%

% SVM classifier with training data, labels and test set
% Use xtrain(:, 1:4000)', label(:, 1:4000) for faster run time
% Increase the number, 4000, if accuracy is low.
Mdl = fitcecoc(xtrain',label);

%%

clc;
testlabels = predict(Mdl,proj_test');

testNum = size(testlabels,1);
err = abs(testlabels - true_label);
err = err > 0;
errNum = sum(err);
sucRate = 1 - errNum/testNum %% Accuracy percentage

confusionchart(true_label, testlabels);
title("Classification Chart of SVM for 10 digits");
```

Listing 7: Support Vector Machine (SVM) for 10 digits

```
%% Follow Step 1 on page 13.

%% This is the Step 2 code for Descision Tree Classifier for 10 digits.
% We use the builtin Matlab function, fitctree, and predict. We also explore another
% way of quantifying error with fitctree and kfoldLoss.



%% All 10 digits, if you only want 2, you have to change the code slightly
% xtrain = projection_training(:, labels == 1| labels == 2) gives you 1 and
% 2 matrix

xtrain = projection_training;
label = labels';

proj_test = projection_test;
true_label = test_labels;

%% Increase number of max splits for higher accuracy.
Md1 =fitctree(xtrain',label,'MaxNumSplits',10);
view(Md1,'Mode','graph');
% classError = kfoldLoss(Md1)

%% First way of calculating error Tree
clc;
approx_labels = predict(Md1, proj_test');

testNum = size(approx_labels,1);
err = abs(approx_labels - true_label);
err = err > 0;
errNum = sum(err);
sucRate = 1 - errNum/testNum %% Accuracy percentage

%% 2nd way of calculating error for Tree

tree = fitctree(xtrain', label, 'MaxNumSplits', 10, 'CrossVal', 'on');
classErrorTree = kfoldLoss(tree);
TreeError10 = 1 - classErrorTree %% Accuracy percentage
```

Listing 8: Decision Tree Classifier on all 10 digits