AMATH 482 Homework 4:
Digit Classification

Amelia Nathan

March 10, 2021

Abstract

This investigation explores the utility of three classification algorithms: Linear Discriminant Analysis, Support Vector Machines, and Decision Trees. Using MNIST Data, we compare the easiest and hardest digits to classify and discuss the relationship between SVD, principal components, and ultimately classifier success rates.

1 Introduction and Overview

1.1 Introduction to Classification and Supervised Machine Learning

Machine learning has become extremely valuable and subsequently excessively studied; it presents extensive potential for innovative data tasks, and classification is a common application that has proven and remains useful. Specifically, digit classification is a clear example that, in this investigation, will highlight the utility of machine learning in classifying images and handwriting samples.

1.2 Problem Overview

In this investigation we are tasked with building classifiers for recognizing and categorizing handwritten digits. Using a set of 60,000 training images and 10,000 test images from the MNIST Database, we are to analyze the data using Singular Value Decomposition and then use three types of classifiers to compare their utility: Linear Discriminant Analysis (LDA), Support Vector Machines (SVM), and Decision Trees. We must then quantify classification accuracy for the easiest and hardest digits to differentiate and analyze the overall performance of our three classifier methods.

2 Theoretical Background

2.1 Foundational Ideas in Linear Algebra

A key theory directly related to matrices that gives the SVD significance in practice is the following (from lecture notes):

If A is a matrix of rank r, then A is the sum of r rank 1 matrices:

$$A = \Sigma_{j=1}^{r} \sigma_j u_j v_j^* \tag{1}$$

where $u_j v_j^*$ is the outer product and the resulting matrix is rank 1. Ultimately, this foundational idea leads to the possibility of dimensionality reduction which, for high rank matrices, greatly reduces computational time. This notion of approximation greatly informs principal component analysis as an analytical technique. As will be discussed in the next theoretical background section, Singular Value Decomposition is a technique that is based on this idea of dimensionality reduction and approximation.

2.2 Singular Value Decomposition

Singular value decomposition is a factorization technique that stretches and transforms vectors by using two unitary matrices U and V and a diagonal matrix Σ. The simple definition of the SVD is given by the following equation where A is the matrix being deconstructed:

$$A = U\Sigma V^*, \tag{2}$$
$$U \in \mathbb{R}^{mxm}, \quad V \in \mathbb{R}^{nxn}, \quad \Sigma \in \mathbb{R}^{mxn}$$

The matrix Σ is a diagonal matrix with singular values along the diagonal. U and V are unitary and geometrically, do the following as described in lecture notes:

- Multiplying by V^* facilitates rotation
- Multiplying by Σ facilitates stretching
- Multiplying by U facilitates proper orientation

As previously mentioned, Σ contains singular values. U and V are unitary matrices; the columns of U contain left singular vectors and the columns of V are right singular vectors, all with respect to the factored matrix A. In finding the SVD, it is important to note that eigenvalues and eigenvectors form a key component. Calculating the eigenvalues of A^TA and AA^T is a key step as the singular values are the squares of these eigenvalues (both of these matrix products have the same eigenvalues).

2.3 Principal Component Analysis

Principal component analysis is an extension of singular value decomposition; in short, the principal components are values corresponding to best fit vectors that reduce the complexity and, when accurate, closely match the data to demonstrate trends. The best fit is found and defined as the minimum average squared distance from points to line. The key equation in principal component analysis that also demonstrates the connection to SVD as it incorporates the U matrix is:

$$Y = U^TX \tag{3}$$

This equation is put in context in the next theoretical section as we go over the statistical ideas that form the foundation for PCA.

2.4 Statistical Background

Variance and covariance are two significant statistical concepts that inform principal component analysis. Variance is the spread of the data and the square is given by

$$\sigma^2 = \frac{1}{n}\Sigma_{k=1}^{n}(a_k - \mu)^2 \tag{4}$$

If we assume we have subtracted the mean (μ above) and subtract one from our denominator to account for under-prediction, we can write variance as an inner product that will be used in the algorithm section and serves as an unbiased estimator:

$$\sigma^2 = \frac{1}{n-1}aa^T \tag{5}$$

Covariance measures variables' variance with respect to each other and is represented by the following equation. Here a and b are the variables interacting with each other.

$$\sigma_{ab}^2 = \frac{1}{n-1}ab^T \tag{6}$$

An important concept to note is that a covariance of 0 indicates the two variables are uncorrelated, or statistically independent. The final Sigma matrix, as described in 2.2 has 0 entries on all off-diagonals, indicating that the variables in Y are uncorrelated.

$$C_Y = \frac{1}{n-1}YY^T = \frac{1}{n-1}U^TXX^TU = U^TAA^TU = U^TU\Sigma^2U^TU = \Sigma^2 \tag{7}$$

2.5 Linear Discriminant Analysis

Linear Discriminant Analysis facilitates a separation of data through projection; its aim, as stated in lecture notes, is to maximize the distance between inter-class data and minimize the intra-class data. LDA is implemented by calculating means, between- and within-class scatter matrices, and using eigenvalues to solve for w which serves as a threshold for distinguishing between images, in this case handwritten digits. Linear discriminant analysis, generally, is as follows:

Between-class scatter matrix:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \text{where } \mu \text{ is a mean.} \tag{8}$$

The within-class scatter matrix:

$$S_w = \sum_{j=1}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{9}$$

$$w = argmax \frac{w^T S_B w}{w^T S_w w} \tag{10}$$

$$S_B w = \lambda S_w w \tag{11}$$

For larger comparison groups,

$$S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T \tag{12}$$

$$S_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{13}$$

2.6 Support Vector Machine (SVM) and Decision Tree Classifiers

In addition to LDA, SVM and Decision Trees are alternative supervised machine learning methods to classify data. Both use binary optioned decisions to facilitate classifications. While LDA is the main focus of this investigation, we will be using built-in matlab commands for SVM and decision trees that can be found in the appendix to enable a comparison of the methods. Decision trees have been critiqued for not being very robust; a small change in training data can significantly skew predictions. While SVM is considered a very robust method, when the number of features exceeds training samples, over-fitting can nonetheless occur.

3 Algorithm Implementation and Development

The first step in implementing our analysis was to load the data from the MNIST database and reshape the data into 784 x 60,000 and 784 x 10,000 matrices for the training and test images respectively. After reshaping the data I performed Singular Value Decomposition to understand how the singular values and principle components represented the data; this provided information about how many principle components, or features, were needed to accurately represent the digit images without needing to store and use all of the original information. The steps for this first component of implementation are shown in algorithm 1.
Before implementing LDA, SVM, and Decision Tree Classification, I also plotted singular value energies and the projection of training data of 3 modes onto a scatter plot to visualize the overlap of features of digits and gain intuition about what digits would be easiest and hardest to classify; this plot is included in the results section.

The following algorithm shows implementation of LDA. I listed the general steps, although in practice and as can be seen in the appendix of my matlab code, I separately coded cases for classifying two digits and three digits. It should be noted that for the LDA of 3 digits, SVM, and Decision Tree Methods I used a projection of the first 154 modes, as was determined to be the necessary number to capture 95% of the digit information, to aid in computational time.

Algorithm 1: Data Reshaping and Finding Feature Threshold

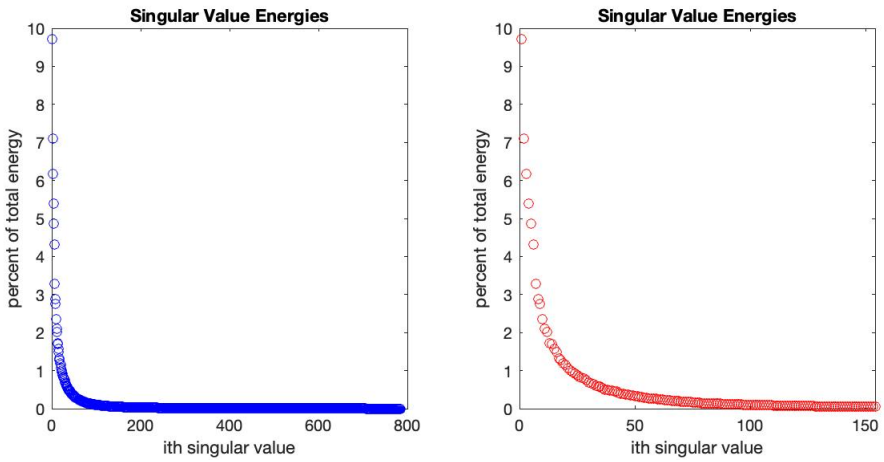
Load training and test images and labels.
Initialize matrices for reshaped data.
for every digit image **do**
  Use `im2double()` and `reshape` to prepare data for SVD analysis
end for
Calculate the mean of each training data column
Subtract mean from training data and divide by `sqrt(n-1)`
Use `svd` to get U, S, V matrices of training data
Define `lambda` as the vector of squared diagonal of singular values
for each singular value in S **do**
  Sum energy captured in singular values until a threshold is passed to find the number of singular values to include to capture the threshold of energy
end for

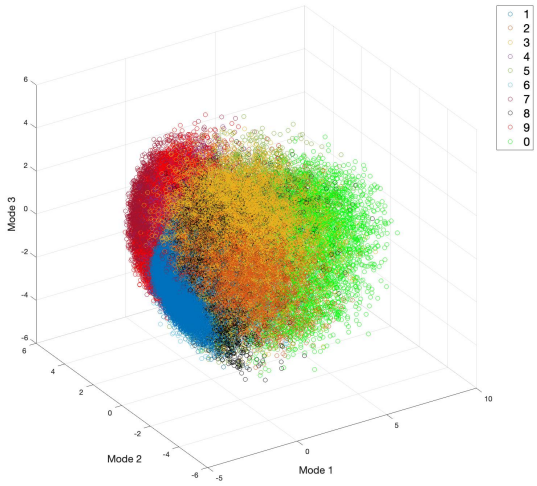
Algorithm 2: Linear Discriminant Analysis

Load and reshape MNIST Data
Isolate data for desired digits
Initialize true labels vector
Use `svd` to perform Singular Value Decomposition
Project using matrix S and V transpose
Calculate S_w
Calculate S_B
Find eigenvalues
Calculate vector w
Find threshold

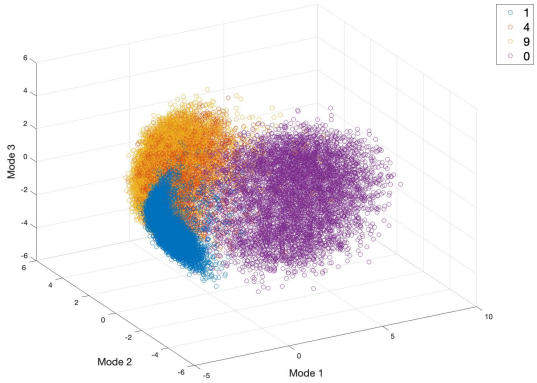
For each implementation of LDA, SVM, and Decision Tree Classifiers we also found the error to aid in comparing the methods. To implement SVM we used the command `fitcsvm` and `fitctree` for the decision tree; the specifics of these commands can be found in the matlab command appendix. Essentially, these are built-in commands to automatically perform these classification methods.

4 Computational Results

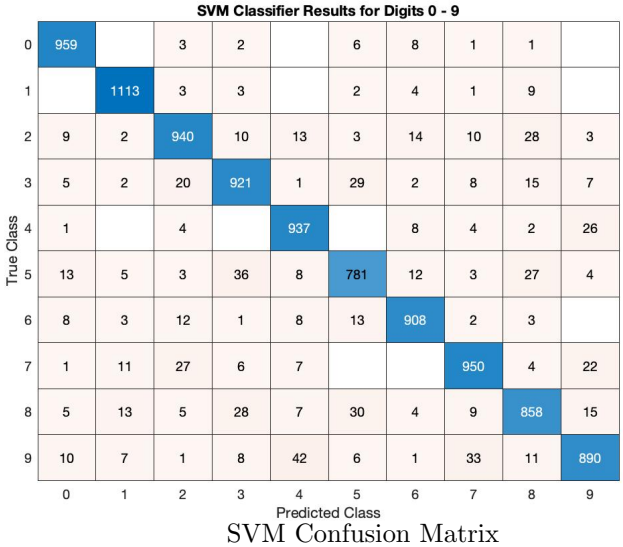




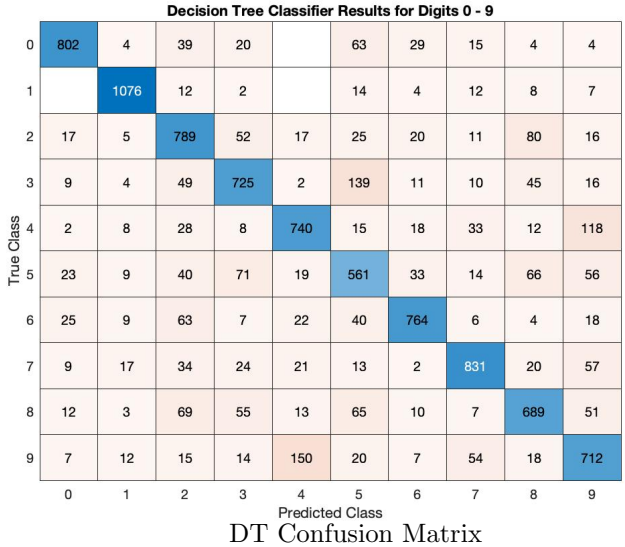
Principal Component Projection Modes 1-3



Principal Component Projection Modes 1-3



SVM Confusion Matrix



DT Confusion Matrix

Algorithm	Digits Classified	Test Success Rate	Training Set Success Rate
2 Digit LDA	1,2	0.9875	0.9843
2 Digit LDA	0,1	0.9962	0.9953
2 Digit LDA	4,9	0.9573	0.9602
3 Digit LDA	1,2,3	0.9367	-
3 Digit LDA	3,4,9	0.9227	-
2 Digit SVM	1,2	0.9931	-
2 Digit SVM	0,1	0.9986	-
2 Digit SVM	4,9	0.9704	-
2 Digit Tree	1,2	0.9852	-
2 Digit Tree	0,1	0.9967	-
2 Digit Tree	4,9	0.9031	-
10 Digit SVM	0-9	0.9257	0.9736
10 Digit Tree	0-9	0.7689	0.8393

Table 1: Classifier algorithm success rates

The first four plots show singular value energies and principal component projections. These indicate the information obtained through SVD and informed which digits would be easiest and hardest to classify, as well as how many principal components were necessary to obtain accurate representations of the digits without having to store and use all of the information provided in the image. I first plotted all 784 singular values. As the first plot shows, the energies have a tail that indicate more than the first singular values are necessary for capturing enough information. Algorithm 1 found that 154 modes were necessary to capture 95% of the information, so I plotted the first 154 singular values as well to demonstrate the energy distribution between each mode. I then used these 154 modes to plot projections for 3 columns - 1, 2, 3 - to visualize the digit information. As can be seen in the first plot of all 10 digits, some colors are more isolated and some overlap significantly. This plot helped me predict which digits would be most difficult and easiest to classify. The second 3D scatter plot shows just digits 0,1,4,9, the digits that I found to be easiest and most difficult for our algorithms to classify. As seen in the plot, the blue and purple projections correspond to 0 and 1 and are very isolated and distinct on the plot. The orange and red, corresponding to 4 and 9, on the other hand, significantly overlap, indicating similar features that the classifier might have difficulty distinguishing. After this initial implementation to better understand and visualize the data, I found the success rates for all 3 algorithms for different cases of 2, 3, and 10 digit classifications. As seen in the confusion matrices and Table 1 listing specifics of the tests I performed, 4 and 9 were difficult for our algorithms to classify. 3 and 5 were also challenging, whereas 1 and 0 were easiest, as would be expected based on the 3D scatter plot. I also used digits 1 and 2 as an intermediate case to better compare the performance of our classifiers for the easier and harder classification tests.

As summarized in table 1, the decision tree was least accurate, especially for 10 digits and classifying 4 and 9. As was expected, for all 3 methods for 2 digits the algorithms were most accurate for 0,1 and least accurate for 4,9 with 1,2 having a success rate in the middle. The SVM method performed best when classifying 2 digits, slightly better than our LDA algorithm. Overall, the decision tree method performed worst of all.

While the majority of this investigation tested the classifications of the test data after training the algorithms with the training data, I tested the algorithms on the training data to compare results as well. I did this for all three 2-digit cases for LDA and the 10 digit cases for SVM and the Decision Tree. These results are also listed in table 1. As shown, there was not much difference in LDA performance; in fact, the success rate was slightly lower on the training data. However, for the 10 digit cases, both SVM and Decision Tree Classifiers had substantially higher success rates. This makes sense: while the 10 digit classifiers were less successful on new test data because it was a very large and diverse test set of 10,000 images, when dealing with a large set it had already processed before, higher success is reasonable and expected.

5 Summary and Conclusions

This investigation highlighted the power and benefits, but also limitations, of machine learning classification. We were able to obtain relatively high success rates using SVM and LDA, especially for two digit classification of numbers with somewhat distinct projections and features, namely 0 and 1 and 1 and 2. However, even with a large training set we were unable to achieve perfect results, hinting at the distinction between human identification and using AI to classify digits. Nonetheless, machine learning for image recognition demonstrates potential and continues to be a valuable tool in data science.

Appendix A MATLAB Functions

- `min(X)` Finds the minimum value within X
- `sum(X)` Finds the sum of values in X
- `mean(X)` Finds the mean of values in X
- `[x, y] = size(X)` Returns the dimensions of a matrix X in terms of number of rows and columns
- `[U, S, V] = svd(A)` Computes the singular value decomposition of matrix A, returning a diagonal matrix S and two unitary matrices U and V such that X = U*S*V'
- `repmat(A,M,N)` replicates the value A in an mxn matrix
- `diag(V)` Creates a diagonal matrix with vector V along the diagonal
- `double` Casts values as type double
- `ind2sub` returns subscript indices for a location in a matrix
- `mnist.parse()` A function from stackoverflow to load MNIST data into matlab: https://stackoverflow.com/questions/39 do-i-load-in-the-mnist-digits-and-label-data-in-matlab
- `dctrainer` A machine learning trainer found in the textbook and used in class
- `scatter3` Creates a 3D scatter plot.
- `ones, zeros(m,n)` Creates a matrix of ones or zeros with dimensions m x n.
- `fitcdiscr()` Returns a classification decision tree
- `crossval()` Performs cross-validation for a function on training and test data
- `kfoldloss()` Returns cross-validation loss
- `fitcecoc()` Fits a multiclass model for SVM classifier
- `predict()` Returns predicted class labels
- `fitctree()` Returns a classification decision tree for data

Appendix B MATLAB Code

```
% Assignment 4 - Classifying Digits

% load data
[training_images, training_labels] = mnist_parse('train-images.idx3-ubyte',
'train-labels.idx1-ubyte');
[test_images, test_labels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-
labels.idx1-ubyte');

% reshape data
training = zeros(784, 60000);
for i = 1:60000
    training(:,i) = im2double(reshape(training_images(:,:,i), 784, 1));
end

% reshape data
tests = zeros(784, 10000);
for i = 1:10000
    tests(:,i) = im2double(reshape(test_images(:,:,i), 784, 1));
end

%%
% testing reshape to confirm images are the same - confirmed
for i = 1:3
subplot(2,3,i)
test = reshape(training(:,i),28,28,1);
imshow(test)
subplot(2,3,i+3)
test = training_images(:,:,i)
imshow(test)
end

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

[U, S, V] = svd(A, 'econ');
lambda = diag(S).^2; % vector of singular values
subplot(1,2,1)
plot(1:784, lambda/sum(lambda)*100, 'bo');
title('Singular Value Energies');
xlabel('ith singular value')
ylabel('percent of total energy')

subplot(1,2,2)
singular_values = lambda/sum(lambda)*100
plot(1:154, singular_values(1:154), 'ro');
title('Singular Value Energies');
xlabel('ith singular value')
ylabel('percent of total energy')

%%
% Use 154 modes to represent training and test image data
projection_training = U(:, 1:154)'*X; % projection of training data
W = tests-repmat(mn, 1, 10000);
projection_test = U(:, 1:154)'*W; % projection of test data
```

```
% Finding the ideal number of singular values to capture the energy above a
% threshold
energy_threshold = 95;
total = 0;
all = sum(lambda);
diagonal = lambda/sum(lambda)*100
for j = 1:size(S)
    total = diagonal(j) + total;
    if total > energy_threshold
        break
    end
end
num_vals = j; % the number of singular values to include to capture the
threshold of energy/information about image

%%
lambda = diag(S).^2;

labels = training_labels;
projection = U(:,1:154)'*X; % U is principal components, direction with most
variance
scatter3(projection(1, labels == 1), projection(2, labels == 1),projection(3,
labels == 1))
hold on
scatter3(projection(1, labels == 2), projection(2, labels == 2),projection(3,
labels == 2))
hold on
scatter3(projection(1, labels == 3), projection(2, labels == 3),projection(3,
labels == 3))
hold on
scatter3(projection(1, labels == 4), projection(2, labels == 4),projection(3,
labels == 4))
hold on
scatter3(projection(1, labels == 5), projection(2, labels == 5),projection(3,
labels == 5))
hold on
scatter3(projection(1, labels == 6), projection(2, labels == 6),projection(3,
labels == 6))
hold on
scatter3(projection(1, labels == 7), projection(2, labels == 7),projection(3,
labels == 7))
hold on
scatter3(projection(1, labels == 8), projection(2, labels == 8),projection(3,
labels == 8),'black')
hold on
scatter3(projection(1, labels == 9), projection(2, labels == 9),projection(3,
labels == 9),'red')
hold on
scatter3(projection(1, labels == 0), projection(2, labels == 0),projection(3,
labels == 0),'green')
legend('1', '2', '3','4', '5','6','7','8', '9', '0', 'Fontsize', 16)
xlabel('Mode 1','Fontsize', 14)
ylabel('Mode 2','Fontsize', 14)
zlabel('Mode 3','Fontsize', 14)
%% LDA 2 Digits
```

```
feature = j; % number of features/modes to capture

im1 = training(:, training_labels == 1);
im2 = training(:, training_labels == 2);

test_1 = tests(:, test_labels == 1); % using original data - in 'tests'
test_2 = tests(:, test_labels == 2);

% redefine to cross check with training dataset

test_1 = im1;
test_2 = im2;
test_set = [test_1 test_2]; % changed to cross test on training
hidden_labels = [zeros(1, size(test_1,2)) ones(1,size(test_2,2))];

[U,S,V,threshold,w,sortim1,sortim2] = dc_trainer(im1,im2,feature);

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

ResVec = (pval>threshold);

% 0s are correct and 1s are incorrect
err = abs(ResVec - hidden_labels);
errNum = sum(err);
sucRateLDA2 = 1 - errNum/TestNum % for 1 and 2

% testing additional digits

im1 = training(:, training_labels == 4);
im2 = training(:, training_labels == 9);

test_4 = tests(:, test_labels == 4); % using original data - in 'tests'
test_9 = tests(:, test_labels == 9);
test_4 = im1;
test_9 = im2;
test_set = [test_4 test_9];
hidden_labels = [zeros(1, size(test_4,2)) ones(1,size(test_9,2))];

[U,S,V,threshold,w,sortim1,sortim2] = dc_trainer(im1,im2,feature);

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

ResVec = (pval>threshold);

% 0s are correct and 1s are incorrect
err = abs(ResVec - hidden_labels);
errNum = sum(err);
sucRateLDA22 = 1 - errNum/TestNum % for 4 and 9

% testing additional digits
```

```
im1 = training(:, training_labels == 0);
im2 = training(:, training_labels == 1);

test_0 = tests(:, test_labels == 0); % using original data - in 'tests'
test_1 = tests(:, test_labels == 1);

test_0 = im1;
test_1 = im2;
test_set = [test_0 test_1];
hidden_labels = [zeros(1, size(test_0,2)) ones(1,size(test_1,2))];

[U,S,V,threshold,w,sortim1,sortim2] = dc_trainer(im1,im2,feature);

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

ResVec = (pval>threshold);

% 0s are correct and 1s are incorrect
err = abs(ResVec - hidden_labels);
errNum = sum(err);
sucRateLDA23 = 1 - errNum/TestNum % for 0 and 1

%% LDA 3 digits

xtrain = projection_training(:, training_labels == 3 | training_labels == 4 |
training_labels == 9);
label = training_labels(training_labels == 3 | training_labels == 4 |
training_labels == 9);
label = label';
mdl = fitcdiscr(xtrain', label, 'discrimType', 'diaglinear');
test = projection_test(:, test_labels == 3 | test_labels == 4 | test_labels
== 9);
label_approx = predict(mdl, test');

hidden_labels =  test_labels(test_labels == 3 | test_labels == 4 |
test_labels == 9, :);
TestNum = size(hidden_labels, 1);

err = abs(label_approx - hidden_labels);
err = err > 0;
errNum = sum(err);
sucRateLDA3 = 1 - errNum/TestNum

%% LDA 3 digits method 2 - comparing error methods

xtrain = projection_training(:, training_labels == 1 | training_labels == 2 |
training_labels == 3);
label = training_labels(training_labels == 1 | training_labels == 2 |
training_labels == 3);
label = label';
mdl = fitcdiscr(xtrain', label, 'discrimType', 'diaglinear');
cmdl = crossval(mdl);
classErrorLDA3 = kfoldLoss(cmdl);
sucRateLDA32 = 1 - classErrorLDA3
```

```
%% SVM -  Used projection, was more accurate
xtrain = projection_training(:, training_labels == 1 | training_labels == 2);
label = training_labels(training_labels == 1 | training_labels == 2);
label = label';
Mdl = fitcsvm(xtrain',label); % svm classifier
label_approx = predict(Mdl,projection_test(:, test_labels == 1 | test_labels
== 2)');

hidden_labels =  test_labels(test_labels == 1 | test_labels == 2, :);
TestNum = size(hidden_labels, 1);

err = abs(label_approx - hidden_labels);
err = err > 0;
errNum = sum(err);
sucRateSVM2 = 1 - errNum/TestNum

% for 4 and 9

xtrain = projection_training(:, training_labels == 4 | training_labels == 9);
label = training_labels(training_labels == 4 | training_labels == 9);
label = label';
Mdl = fitcsvm(xtrain',label); % svm classifier
label_approx = predict(Mdl,projection_test(:, test_labels == 4 | test_labels
== 9)');

hidden_labels =  test_labels(test_labels == 4 | test_labels == 9, :);
TestNum = size(hidden_labels, 1);

err = abs(label_approx - hidden_labels);
err = err > 0;
errNum = sum(err);
sucRateSVM22 = 1 - errNum/TestNum

% for 0 and 1 - easiest

xtrain = projection_training(:, training_labels == 0 | training_labels == 1);
label = training_labels(training_labels == 0 | training_labels == 1);
label = label';
Mdl = fitcsvm(xtrain',label); % svm classifier
label_approx = predict(Mdl,projection_test(:, test_labels == 0 | test_labels
== 1)');

hidden_labels =  test_labels(test_labels == 0 | test_labels == 1, :);
TestNum = size(hidden_labels, 1);

err = abs(label_approx - hidden_labels);
err = err > 0;
errNum = sum(err);
sucRateSVM23 = 1 - errNum/TestNum
%% for 10 digits

xtrain = projection_training(:,1:10000)./max(projection_training(:,1:10000));
label = training_labels(1:10000, :);
label = label';
Mdl = fitcecoc(xtrain',label); % svm classifier
```

```
project_test = projection_test(:,1:10000)./max(projection_test(:,1:10000));
label_approx_svm = predict(Mdl,xtrain');

hidden_labels_svm =  test_labels(1:10000,:);
hidden_labels_svm = label'; % swapped for training comparison
TestNum = size(hidden_labels_svm, 1);

err = abs(label_approx_svm - hidden_labels_svm);
err = err > 0;
errNum = sum(err);
sucRateSVM10 = 1 - errNum/TestNum
%
% figure()
% confusionchart(hidden_labels_svm,label_approx_svm);
% title('SVM Classifier Results for Digits 0 - 9');
%% Decision Tree - 2 digits
xtrain = projection_training(:, training_labels == 1 | training_labels == 2);
label = training_labels(training_labels == 1 | training_labels == 2);
label = label';
mdl=fitctree(xtrain',label,'MaxNumSplits', 200);
% view(mdl,'Mode','graph');
label_approx = predict(mdl, projection_test(:, test_labels == 1 | test_labels
== 2)');

hidden_labels =  test_labels(test_labels == 1 | test_labels == 2, :);
TestNum = size(hidden_labels, 1);

err = abs(label_approx - hidden_labels);
err = err > 0;
errNum = sum(err);
sucRateDT2 = 1 - errNum/TestNum

% next two digit test - 4,9

xtrain = projection_training(:, training_labels == 4 | training_labels == 9);
label = training_labels(training_labels == 4 | training_labels == 9);
label = label';
mdl=fitctree(xtrain',label,'MaxNumSplits', 200);
% view(mdl,'Mode','graph');
label_approx = predict(mdl, projection_test(:, test_labels == 4 | test_labels
== 9)');

hidden_labels =  test_labels(test_labels == 4 | test_labels == 9, :);
TestNum = size(hidden_labels, 1);

err = abs(label_approx - hidden_labels);
err = err > 0;
errNum = sum(err);
sucRateDT22 = 1 - errNum/TestNum

% next two digit test - 0, 1

xtrain = projection_training(:, training_labels == 0 | training_labels == 1);
label = training_labels(training_labels == 0 | training_labels == 1);
label = label';
mdl=fitctree(xtrain',label,'MaxNumSplits', 200);
```

```
% view(mdl,'Mode','graph');
label_approx = predict(mdl, projection_test(:, test_labels == 0 | test_labels
== 1)');

hidden_labels =  test_labels(test_labels == 0 | test_labels == 1, :);
TestNum = size(hidden_labels, 1);

err = abs(label_approx - hidden_labels);
err = err > 0;
errNum = sum(err);
sucRateDT23 = 1 - errNum/TestNum
%% Decision Tree - 10 digits
xtrain = projection_training(:,1:10000);
label = training_labels(1:10000,:);
label = label';
mdl=fitctree(xtrain',label,'MaxNumSplits', 200)% 'CrossVal', 'on');
label_approx_dt = predict(mdl, xtrain');
hidden_labels_dt = test_labels;
hidden_labels_dt = label';

TestNum = size(hidden_labels_dt, 1);

err = abs(label_approx_dt - hidden_labels_dt);
err = err > 0;
errNum = sum(err);
sucRateDT10 = 1 - errNum/TestNum

%classErrorTree = kfoldLoss(mdl); alternative error method
%sucRateDT10 = 1- classErrorTree


figure()
confusionchart(hidden_labels_dt, label_approx_dt);
title('Decision Tree Classifier Results for Digits 0 - 9');

%% visual comparison for two digit methods
figure()
plot([1 1.5 2], [sucRateLDA2, sucRateSVM2, sucRateDT2], 'm.', 'MarkerSize',
20)
xticklabels({'LDA', 'SVM', 'DT'})
hold on
plot([1 1.5 2], [sucRateLDA22, sucRateSVM22, sucRateDT22], 'g.',
'MarkerSize', 20)
xticks([1 1.5 2])
plot([1 1.5 2], [sucRateLDA23, sucRateSVM23, sucRateDT23], 'c.',
'MarkerSize', 20)
xticklabels({'LDA', 'SVM', 'DT'})
ylim([0.9 1])
legend('1,2', '4,9', '0,1')
```

Dc_trainer.m

```
function [U,S,V,threshold,w,sortim1,sortim2] = dc_trainer(im1,im2,feature)

    nd = size(im1,2);
    nc = size(im2,2);
    [U,S,V] = svd([im1 im2],'econ');
    projections = S*V';
    U = U(:,1:feature); % Add this in
    im1 = projections(1:feature,1:nd);
    im2 = projections(1:feature,nd+1:nd+nc);
    md = mean(im1,2);
    mc = mean(im2,2);

    Sw = 0;
    for k=1:nd
        Sw = Sw + (im1(:,k)-md)*(im1(:,k)-md)';
    end
    for k=1:nc
        Sw = Sw + (im2(:,k)-mc)*(im2(:,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);
    vim1 = w'*im1;
    vim2 = w'*im2;

    if mean(vim1)>mean(vim2)
        w = -w;
        vim1 = -vim1;
        vim2 = -vim2;
    end

    % Don't need plotting here
    sortim1 = sort(vim1);
    sortim2 = sort(vim2);
    t1 = length(sortim1);
    t2 = 1;
    while sortim1(t1)>sortim2(t2)
    t1 = t1-1;
    t2 = t2+1;
    end
    threshold = (sortim1(t1)+sortim2(t2))/2;

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

mnist_parse.m

```
function [images, labels] = mnist_parse(path_to_digits, path_to_labels)

% The function is curtesy of stackoverflow user rayryeng from Sept. 20,
% 2016. Link: https://stackoverflow.com/questions/39580926/how-do-i-load-in-
the-mnist-digits-and-label-data-in-matlab
```

```
% 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
```