# AMATH 482 Homework 4

Jeremy Lu

March 10, 2021

**Abstract**

Using the MNIST data set, which is a collection of handwritten numerical digits, we train a Linear Discriminant Analyzer (LDA) to identify individual digits in MATLAB. Comparing our results to a Support Vector Machine (SVM) and a binary classification tree trained on the same data, we find that the SVM and LDA have similar accuracies, and the decision tree is outperformed in some cases.

## 1 Introduction and Overview

Machine Learning is a field in mathematics and statistics, which aims to accomplish a variety of human tasks (and non-human), often through large data sets. The two main types of machine learning models, are supervised and unsupervised; with the main distinction being that supervised learning trains on pre-labelled data.This project investigates three supervised classification models: linear discriminant analysis, support vector machines, and binary decision trees.
Using a total of 60,000 handwritten digits from the MNIST dataset, these three models were trained and had their performance evaluated on 10,000 test digits. The LDA performed best in differentiating 0's and 4's, and did the worst differentiating 4's and 9's. Additionally, the SVM had a similar accuracy with the LDA, while the classification trees were inconsistent in predicting digits with high accuracy.

## 2 Theoretical Background

### 2.1 Singular Value Decomposition

Given an $m \times x$ matrix $A$, the Singular Value Decomposition, or SVD, rewrites the matrix $A$ in the form of $\mathbf{U}\ \Sigma\ \mathbf{V}^*$, where $\mathbf{U}$ and $\mathbf{V}$ are $m \times m$ and $n \times n$ unitary matrices and $\Sigma$ is a matrix containing the eigenvalues in the order $\sigma_1 < \sigma_2 < \cdots < \sigma_n$. Geometrically, this is a rotation, rescaling, and second rotation of the original matrix $A$. The SVD is a valuable tool in reducing data, and is used in this project to reduce the 784 features used for model development.

### 2.2 Linear Discriminant Analysis

Linear Discriminant Analysis is a supervised learning algorithm, used to make classifications on data. The goal of LDA is to find a suitable projection, that maximizes the distance between inter-class data while minimizing the distance between intra-class data.
Given two classes, with means $\mu_1$ and $\mu_2$, where these $\mu$ are column vectors (means across each row), we can define the between-class scatter matrix or $\mathbf{S}_B$ as:

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T. \tag{1}$$

This measures the variance between the two classes (between the means), and we can also define the within-class scatter matrix or $\mathbf{S}_w$ as:

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

This measures the variance within each group, and the goal then becomes finding a vector $\mathbf{w}$ such that

$$\mathbf{w} = \text{argmax} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}. \tag{3}$$

Using a generalized eigenvalue problem, we can find $\mathbf{w}$ by solving:

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w}. \tag{4}$$

Using the information from this process, we can create a threshold, which then classifies projections of inputted data points.

### 2.3 Support Vector Machine

The Support Vector Machine algorithm is based on constructing hyperplanes in $N$-dimensional space, used to classify data points. These planes are constructed with the objective of maximizing the distance from the data points to the planes.

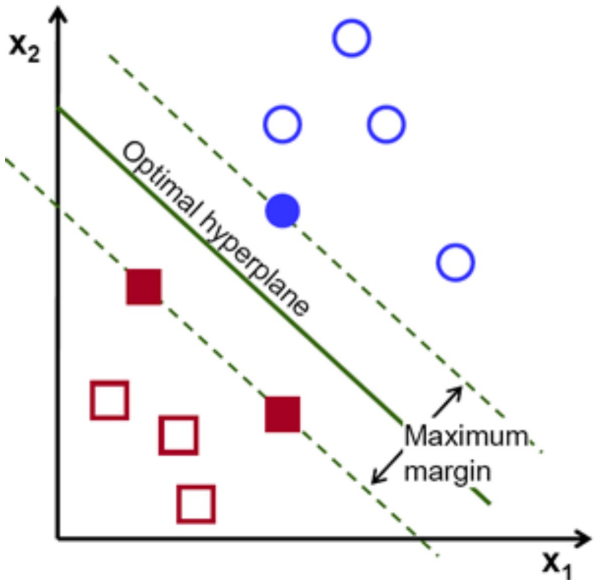


Figure 1: Example of vinary SVM in 2-D (Source)

### 2.4 Binary Decision Tree

Decision trees are another form of supervised learning. Given a training dataset, that data is split into subsets by thresholding features, with the 'best' splits (in MATLAB's `fitctree` command) being determined via Gini impurity. This process is repeated on each data subset until the data is entirely partitioned into distinct categories.

## 3 Algorithm Implementation and Development

We begin by reading in the 60,000 training digits and labels, along with the 10,000 test digits and labels. After preparing the data by reshaping them from 28x28 matrices into 784x1 vectors, we perform the SVD and plot both the energies and singular values:

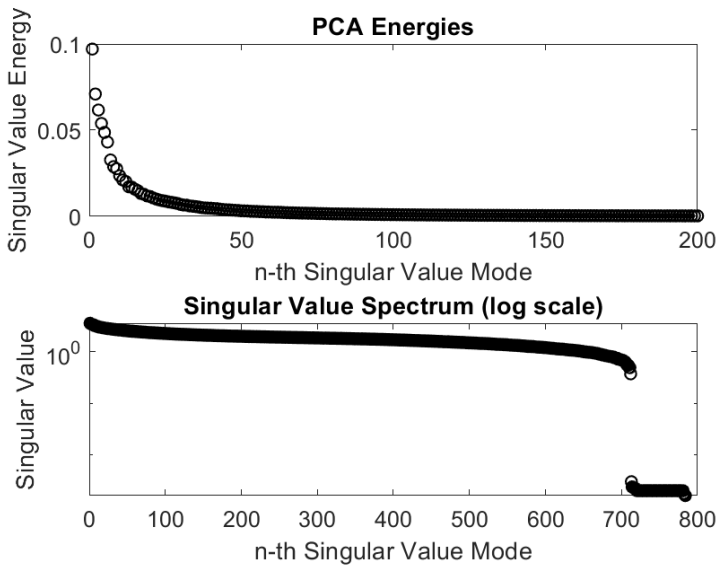


Figure 2: First 200 PCA Energies (top) and Singular Values on a log scale (bottom)

After 100 features, the slope of the PCA Energies reaches close to zero. Thus it was determined that using the first 100 features was most optimal. These features captured 91.46% of the energy, which means that most of the relevant information on our handwritten digits is present.
After the SVD, we can project onto 3 V-Nodes:

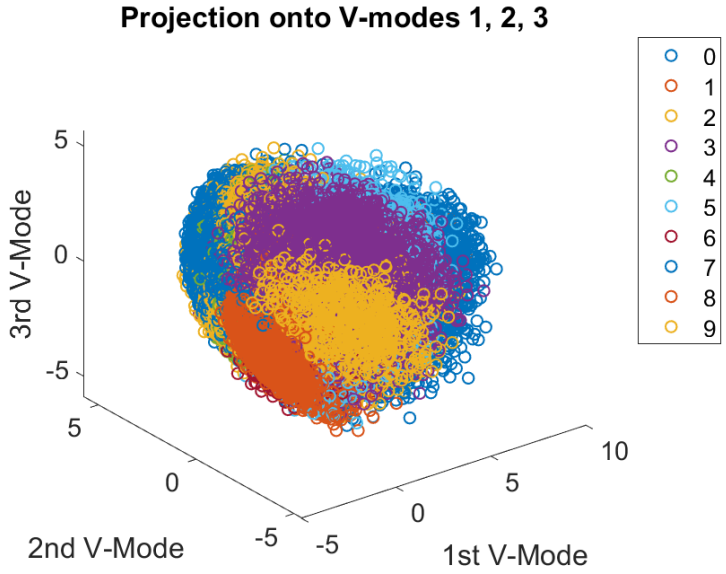


Figure 3: Projection of each digit onto first 3 V-Nodes

Looking at this plot, we can see that each digit seems to be clustered among itself, so our change of basis vector is ideal for reducing the number of features to be accounted for in our models.
With the number of features determined, we can proceed with finding the LDA performance on each pair of digits (along with best and worst performing pairs), using the following:

**Algorithm 1:** Finding LDA Performance on Pairs of Handwritten Digits

```
for i = 0 : 8 do
  for j = j + i : 9 do
    Train LDA on training digits i and j
    Evaluate performance on test digits
    Store digits and accuracy
  end for
end for
```

Additionally, we also use built-in MATLAB functions `fitcecoc` and `fitctree` to train a 10 digit classifying SVM and random forest, as well as training these two models on the easiest and hardest pairs to distinguish.

# 4 Computational Results

## 4.1 LDA Performance

The LDA performed best on the 0 and 4 pair, attaining an accuracy of 0.9985 on the test data, and it performed the worst on 4 and 9, with an accuracy of .9508 (again on test data). The latter does make sense, as 4 and 9 have similar shapes when written by hand. The rest of the accuracies are displayed in the following table:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.9976 | 0.9866 | 0.9940 | 0.9985 | 0.9877 | 0.9892 | 0.9920 | 0.9898 | 0.9910 |
| 1 | | 1 | 0.9866 | 0.9911 | 0.9957 | 0.9926 | 0.9967 | 0.9898 | 0.9796 | 0.9939 |
| 2 | | | 1 | 0.9755 | 0.9811 | 0.9756 | 0.9789 | 0.9791 | 0.9711 | 0.9829 |
| 3 | | | | 1 | 0.9930 | 0.9632 | 0.9964 | 0.9804 | 0.9632 | 0.9797 |
| 4 | | | | | 1 | 0.9872 | 0.9902 | 0.9836 | 0.9898 | 0.9508 |
| 5 | | | | | | 1 | 0.9708 | 0.9896 | 0.9555 | 0.9832 |
| 6 | | | | | | | 1 | 0.9965 | 0.9850 | 0.9969 |
| 7 | | | | | | | | 1 | 0.9845 | 0.9597 |
| 8 | | | | | | | | | 1 | 0.9738 |
| 9 | | | | | | | | | | 1 |

Table 1: LDA Performance on Pairs of Handwritten Digits (Test Data)

## 4.2 SVM Performance

The SVM had an accuracy of 0.9425 when classifying between all ten digits. For the pair one and four, SVM had an accuracy of 0.9985 on the test data, which is the same as the LDA. For the pair 4 and 9, the SVM had an accuracy of 0.9684 on the test data, slightly higher than the SVM. This increase in accuracy for 4 and 9 can be explained by the fact that the SVM during training focuses on the data points which are harder to distinguish, while the LDA focuses on the data as a whole.

## 4.3 Classification Tree Performance

The classification tree had an accuracy of 0.8383 when classifying between all digits. For the pair one and four, our tree had an accuracy of 0.9776 on the test data, which is slightly worse than the LDA and SVM. For the digits 4 and 9, it performed much worse than both the previous models, with an accuracy of 0.457, likely because the forest must make threshold decisions on individual features, rather than looking at the data holistically. The figure below shows the complexity of such decision trees, and how many decisions are made to come to a conclusion:

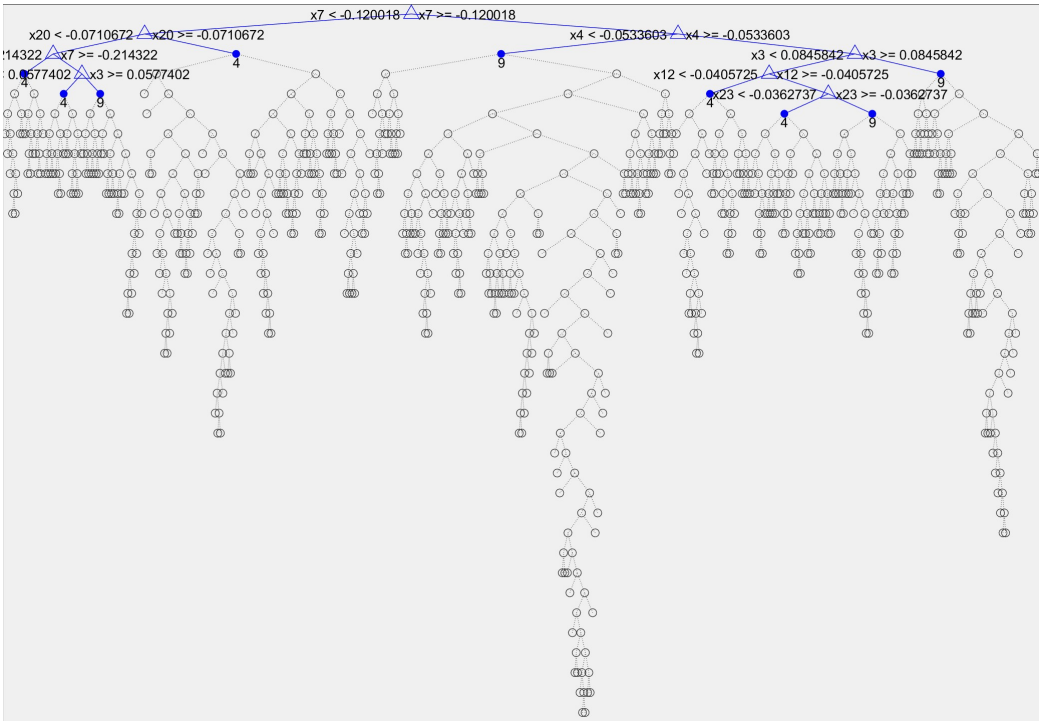


Figure 4: Classification Tree for the Digits 4 and 9 (Pruned to only display 5 out of 52 layers)

# 5 Summary and Conclusions

Through this project, we are able to see applications of the LDA, along with SVM and decision trees. Given images of handwritten digits, we were able to develop models that identify new, unseen images with high accuracy (mostly 90%+). We also utilized the SVD, in order to optimize and reduce the number of features used for model training.

The LDA performed best classifying between 0 and 4 (accuracy of 0.9985, while doing the worst for the pair of 4 and 9 (0.9508). Meanwhile, the SVM had accuracies of 0.9985 and 0.9684, respectively, and the decision tree had accuracies of 0.9776, and 0.457, respectively. Although the decision tree performed poorly in certain pairs, the overall accuracy for all 10 classes was relatively strong at 0.8383. Additionally, the SVM had an accuracy of 0.9425 for all classes. Overall, the SVM had the highest accuracy, followed closely by the LDA, and then the classification tree.

# Appendix A MATLAB Functions

- `abs(X)`: Returns the absolute value of a given input `X`.
- `diag(X)`: Returns the diagonal entries of a given a matrix `X`.
- `digit_trainer(X, Y)` Returns a trained binary linear discriminant analyzer given training data `X` and labels `Y`.
- `eig(X)`: Returns a column vector of eigenvalues corresponding to the matrix `X`.
- `find(X)==Y` Returns indices where `X` and `Y` are the same.
- `fitecoc(X,Y)`: Returns a trained, multiclass support vector machine, given data `X` and labels `Y`,
- `fitctree(X), Y`: Returns a trained, multiclass binary decision classification tree, given data `X` and labels `Y`.

- `im2double(X)`: Returns an image converted to double precision, given an image `X`.
- `max(X)`: Returns the maximum value in a given vector `X`.
- `mean(X, n)` Returns a vector of mean along the $n$ dimension of a given matrix `X`.
- `min(X)` Returns the minimum value in a given vector `X`.
- `mnist_parse(X)`: Returns given files downloaded from MNIST.
- `norm(X)`: Returns the Euclidean norm of a vector or matrix `X`.
- `numel(X)`: Returns the number of elements in a given matrix `X`.
- `predict(X, Y)` Returns class predictions, given a model `X` and data to predict on in `Y`.
- `repmat(X, m, n)`: Returns a matrix with copies of the given matrix `X` in an `m` $\times$ `n` size.
- `size(X)`: Returns the dimensions of a given matrix `X`.
- `sort(X)`: Returns a sorted array of input `X`, in ascending array.
- `sum(X)`: Returns the sum of all the elements in a given vector `X`.
- `svd(X)`: Performs SVD on a given matrix `X` and returns U, $\Sigma$, and V.
- `zeros(m,n)`: Returns an `m` $\times$ `n` matrix of zeroes.

# Appendix B MATLAB Code

```
%% Loading in data
[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.
    idx1-ubyte');
training_data = zeros(784,60000);

[timages, tlabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.
    idx1-ubyte');
testing_data = zeros(784,10000);

%% Reshaping and doing SVD
for i = 1:60000
    training_data(:,i) = im2double(reshape(images(:,:,i),784,1));
end
[m,n] = size(training_data);
mn = mean(training_data,2);
training_data = training_data - repmat(mn,1,n);
[U,S,V] = svd(training_data,'econ');
sig = diag(S);
lambda = diag(S).^2;

for i = 1:10000
    testing_data(:,i) = im2double(reshape(timages(:,:,i),784,1));
end
[m,n] = size(testing_data);
testing_data = testing_data - repmat(mn,1,n);

%%Plotting the energies
figure()
```

```
subplot(2,1,1);
plot(lambda/sum(lambda),'ko','Linewidth',1);
set(gca,'Fontsize',12, 'Xlim', [0 200]);
xlabel('n-th Singular Value Mode')
ylabel('Singular Value Energy')
title("PCA Energies")
subplot(2,1,2)
semilogy(diag(S),'ko','Linewidth',1);
set(gca,'Fontsize',12);
xlabel('n-th Singular Value Mode')
ylabel('Singular Value')
title("Singular Value Spectrum (log scale)")
saveas(gcf, 'PCA_data.png')

%% Projection onto 3 V-modes
projection = U(:,[1,2,3])'*training_data;
for label=0:9
    digit_proj = projection(:,find(labels == label));
    plot3(digit_proj(1,:),digit_proj(2,:),digit_proj(3,:),...
        'o', 'DisplayName', sprintf('%i',label), 'Linewidth', 1)
    hold on
end
xlabel('1st V-Mode'), ylabel('2nd V-Mode'), zlabel('3rd V-Mode')
title('Projection onto V-modes 1, 2, 3')
legend
set(gca,'Fontsize', 14)
saveas(gcf, 'V_node_projection.png')

%% Running LDA on all pairs
feature = 100;
n=1;
accuracies=zeros(3,45);
total_correct = 0;
total_predictions = 0;
for i = 0:8
    for j = i+1:9
        digitA = training_data(:,find(labels==i));
        digitB = training_data(:,find(labels==j));
        [U,S,V,threshold,w,sortedA,sortedB] = digit_trainer(digitA,digitB,
           feature);
        testA = testing_data(:,find(tlabels==i));
        correct = 0;
        alength = size(testA,2);
        for k = 1:alength
            digit = testA(:,k);
            imat = U' * digit;
            digitval = w' * imat;

            if digitval < threshold
                correct = correct + 1;
            end
        end
        testB = testing_data(:,find(tlabels==j));
        blength = size(testB,2);
```

```
        for k = 1:size(testB,2)
            digit = testB(:,k);
            imat = U' * digit;
            digitval = w' * imat;

            if digitval > threshold
                correct = correct + 1;
            end
        end

        accuracy = correct/(alength+blength);
        accuracies(1,n)=i;
        accuracies(2,n)=j;
        accuracies(3,n)=accuracy;
        n=n+1;
        total_predictions=total_predictions+alength+blength;
        total_correct = total_correct+correct;
    end
end
total_correct/total_predictions;

%% Train SVM
[U,S,V] = svd(training_data,'econ');
U=U(:,1:feature);
proj = S*V';
train_input = (U'*training_data)'./max(proj(:));
Mdl = fitcecoc(train_input,labels);
test_input = (U' * testing_data)'./max(proj(:));
testlabels = predict(Mdl,test_input);
correct = 0;
s = testlabels == tlabels;
sum(s)/numel(s);

%% Train Decision Tree
d_tree = fitctree(train_input,labels);

testlabels = predict(d_tree, test_input);
s = testlabels == tlabels;
sum(s)/numel(s);

%% SVM on easiest and hardest pairs
train_input = train_input';
train_zero = train_input(:,find(labels==0));
train_four = train_input(:,find(labels==4));
train_nine = train_input(:,find(labels==9));

label_zero = zeros(1,size(train_zero,2));
label_four = zeros(1,size(train_four,2)) + 4;
label_nine = zeros(1,size(train_nine,2)) + 9;

train_onefour = [train_zero train_four];
label_onefour = [label_zero label_four];
train_fournine = [train_four train_nine];
label_fournine = [label_four label_nine];
```

```
Mdl4 = fitcsvm(train_onefour',label_onefour);
Md49 = fitcsvm(train_fournine',label_fournine);

test_input=test_input';
test_zero = test_input(:,find(tlabels==0));
test_four = test_input(:,find(tlabels==4));
test_nine = test_input(:,find(tlabels==9));

tlabel_zero = zeros(1,size(test_zero,2));
tlabel_four = zeros(1,size(test_four,2)) + 4;
tlabel_nine = zeros(1,size(test_nine,2)) + 9;

test_onefour = [test_zero test_four];
tlabel_onefour = [tlabel_zero tlabel_four];
test_fournine = [test_four test_nine];
tlabel_fournine = [tlabel_four tlabel_nine];

testlabels = predict(Mdl4,test_onefour');
s = testlabels == tlabel_onefour';
sum(s)/numel(s);

testlabels = predict(Md49,test_fournine');
s = testlabels == tlabel_fournine';
sum(s)/numel(s)

%% Classification tree on easiest and hardest pairs + graph of tree
d_tree = fitctree(train_onefour',label_onefour);

testlabels = predict(d_tree,test_onefour');
s = testlabels == tlabel_onefour';
sum(s)/numel(s)

d_tree = fitctree(train_fournine', label_fournine);
testlabels = predict(d_tree,test_onefour');
s = testlabels == tlabel_onefour';
sum(s)/numel(s)

view(d_tree,'mode','graph')
```

## digit_trainer.m

```
function [U,S,V,threshold,w,sortedA,sortedB] = digit_trainer(digitA,
    digitB, feature)
     na = size(digitA, 2);
     nb = size(digitB, 2);
     [U,S,V] = svd([digitA digitB], 'econ');
     digits = S*V';
     U = U(:,1:feature);
     digA = digits(1:feature, 1:na);
     digB = digits(1:feature, na+1:na+nb);
     ma = mean(digA, 2);
     mb = mean(digB, 2);

     Sw=0;
```

```
    for k=1:na
        Sw = Sw + (digA(:,k)-ma)*(digA(:,k)-ma)';
    end
    for k=1:nb
        Sw = Sw + (digB(:,k)-mb)*(digB(:,k)-mb)';
    end
    Sb = (ma-mb)*(ma-mb)';

    [V2,D] = eig(Sb,Sw);
    [lambda, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    va = w'*digA;
    vb = w'*digB;

    if mean(va)>mean(vb)
        w=-w;
        va=-va;
        vb=-vb;
    end

    sortedA = sort(va);
    sortedB = sort(vb);
    t1 = length(sortedA);
    t2 = 1;

    while sortedA(t1)>sortedB(t2)
        t1 = t1-1;
        t2 = t2+1;
    end

    threshold = (sortedA(t1)+sortedB(t2))/2;

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
```