AMATH 482 Homework 4

Sandy Yang

March 10, 2020

Abstract

This report outlines the techniques and procedures necessary to complete a supervised machine learning problem. Below describes the steps taken to train classifiers to recognize different hand written digit.

1 Introduction and Overview

Add your introduction and overview here.

2 Theoretical Background

2.0.1 Singular Value Decomposition

Similar to the previous assignment, solving this problem relies heavily on the processes of Singular Value Decomposition/Principal Component Analysis. To recall, the reduced SVD is written mathematically as:

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

where U and V^T are unitary matrices that contain information on rotation and Σ is a diagonal matrix describing the stretch in each direction. The columns of U are the left singular vectors of A and the columns of V are called the right singular vectors of A. Each non-zero value in the Σ matrix is a singular valueof A and the total number of non-zero singular values is the rank of A.

2.0.2 Linear Discriminant Analysis

To make statistical sense out of the Singular Value Decomposition, this problem uses Linear Discriminant Analysis (LDA) in order to find a suitable projection that maximizes the distance between the MNist data while minimizing the MNist data. Mathematically, this goal is written:

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

with the between-class scatter matrix, S_B, and within-class scatter matrix, S_W,written as:

$$S_B = \sum_{j=1}^{n} m_j (\vec{\mu}_j - \vec{\mu})(\vec{\mu}_j - \vec{\mu})^T \tag{3}$$

$$S_W = \sum_{j=1}^{n} \sum_{\vec{x}} m_j (\vec{x} - \vec{\mu}_j)(\vec{x} - \vec{\mu}_j)^T \tag{4}$$

where m_j is the number of samples in class j, μ is the mean of all the data, and μ_j is the mean of the data in class j. Using these matrices we can find the solution to Equation (2) with a generalized eigenvalue problem:

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

The information provided by Linear Discriminant Analysis and solving this eigenvalue problem gives us the quantity of interest and the projection basis. The projections are what enable us to separate the different classes of data into categories mathematically.

2.0.3 Support Vector Machines

Support Vector Machines (SVM) is an optimization problem that tries to find the an optimal hyperplane $\vec{w}x+b=0$ where all points x satisfying $\vec{w}x+b<0$ are labeled as class one and all points satisfying $\vec{w}x+b>0$ are labeled as class two. SVM optimizes the decision line that makes the fewest labeling errors for the data, and optimizes the largest margin between the data. w and b must be optimized in a principle way, wihch is by stating a loss function:

$$\ell\left(\mathbf{y}_j, \overline{\mathbf{y}}_j\right) = \begin{cases} 0 \text{ if data is correctly labeled} \\ +1 \text{ if data is incorrectly labeled} \end{cases}$$

This can be simplified to:

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{j=1}^{m} H\left(\mathbf{y}_j, \overline{\mathbf{y}}_j\right) + \frac{1}{2}\|\mathbf{w}\|^2$$

subject to:

$$\min_j |\mathbf{x}_j \cdot \mathbf{w}| = 1$$

There are also kernel methods for SVD, which solves the curse of dimensinoality, when there are too many additional features. We define the kernel function as

$$K\left(\mathbf{x}_j, \mathbf{x}\right) = \Phi\left(\mathbf{x}_j\right) \cdot \Phi(\mathbf{x})$$

, and we get a new optimization problem

$$\underset{\boldsymbol{\alpha},b}{\operatorname{argmin}} \sum_{j=1}^{m} H\left(\mathbf{y}_j, \overline{\mathbf{y}}_j\right) + \frac{1}{2}\left\|\sum_{j=1}^{m} \alpha_j \Phi\left(\mathbf{x}_j\right)\right\|^2$$

subject to

$$\min_j |\mathbf{x}_j \cdot \mathbf{w}| = 1$$

. This allows us to represent Taylor series expansions of observables in a compact way. The radial basis function and polynomial kernel are commonly used.

2.0.4 Decision Tree

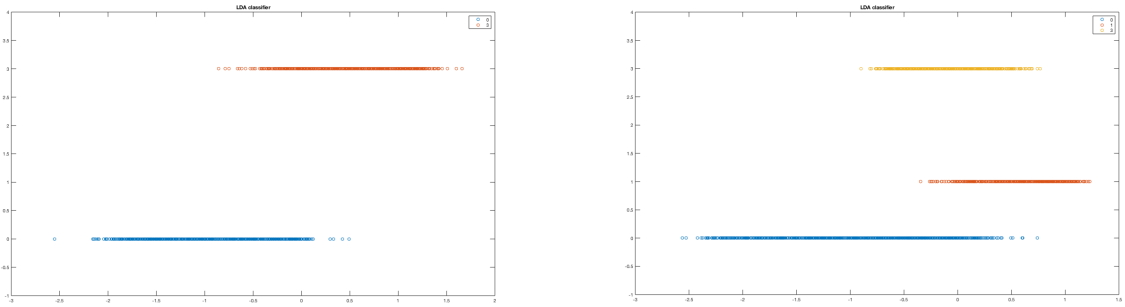
The decision tree is a hierarchical construct that optimally splits data for classification and regression. The model scans through each feature and compares the prediction accuracy across different features. It branches when it reaches a feature giving the best segmentation.

3 Algorithm Implementation and Development

This part includes the algorithm implementation and development for this assignment.

- Load the data, get the training data with dimension 784×60000 and the 1×60000 label.
- Subtract the row-wise mean of the training data set.
- Perform SVD. Get the U, S, V matrix. In this case, each column of U is principal component (eigen vector). S is the singular value, which is a measure of how important of column of U is. V' is the linear coefficient.
- With this understanding. We can use low rank approximation, plot singular values and determine principal components that adequately separate the data to develop a better coordinate system for classification. In this case, we got 90 percent of the energy at rank equal t 350.

- Plot out the 3D scatter plot with the first three component. (In spec, it's said plot out the V matrix but I went to office hour and TA state that we can also plot out the principal component so I did that.)
- Train LDA and SVM models using training dataset, training the model on the coordinates of each of these pieces in the new coordinate system determined by SVD in the previous steps.
- For LDA model, we can construct Si matrix of each selected digits. Construct Sw within class covariance and construct Sb between class covariance. Then, we perform Singular Value Decomposition of $\frac{Sw}{Sb}$. The following classifier on training data look like:

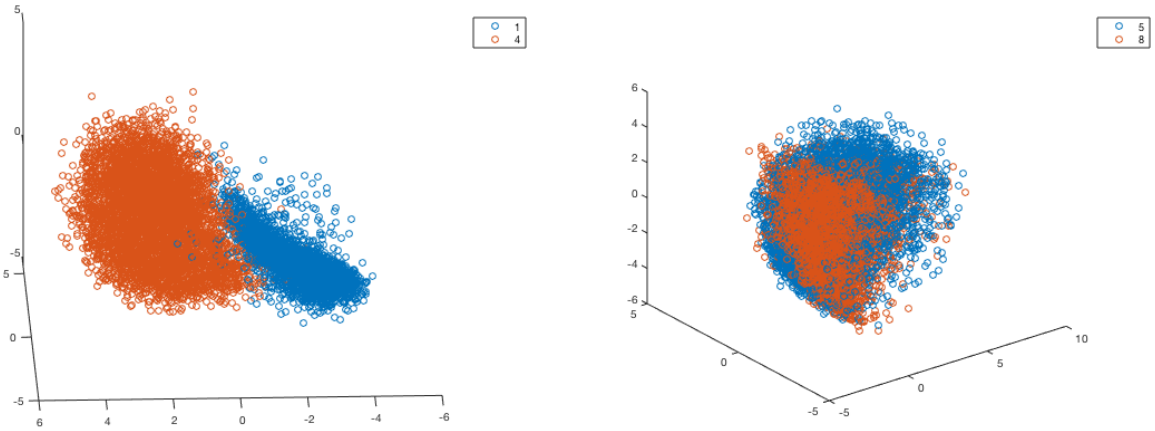


(a) Using digit 0 and 1

(b) Using digit 0, 1 and 3

Figure 1: The projected data of LDA classifier

- To decided the possible candidate of the 2 digits pair which is the easiest/hardest to separate, I look at at the graph of the PCA space. Go through all the pair of number, it seems like [1,4] are the easiest to separate and [5,8] are the most difficult. I also test other pairs such as [0,7] and [2,3] since their PCA graph also look like possible candidate. After decided the pairs, we can use these digits pair and build the LDA classifier as we did in above step.



(a) 1 and 4 on PCA space

(b) 5 and 8 on PCA space (very sad)

Figure 2: PCA space with 2 digit

- For SVM modle, instead of using `fitcsvm`, I choose to use `fitcecoc` since we have multiple variable.
- After building up the classifier/model, we can check accuracy of the model generated by projecting each piece of data in the test dataset and check the model's prediction.

- To calculate the accuracy, we compare our prediction data and our actual test label and see how many is the same and divided it by 10000(test data size).

4 Computational Results

For the first part of the assignment, we got the following graph for the PCA space. As we see, the PCA didn't look really good on classifying hand written digit. But we can still eye-balling the digit can classify which 2 digit is the hardest/easiest pair to separate.

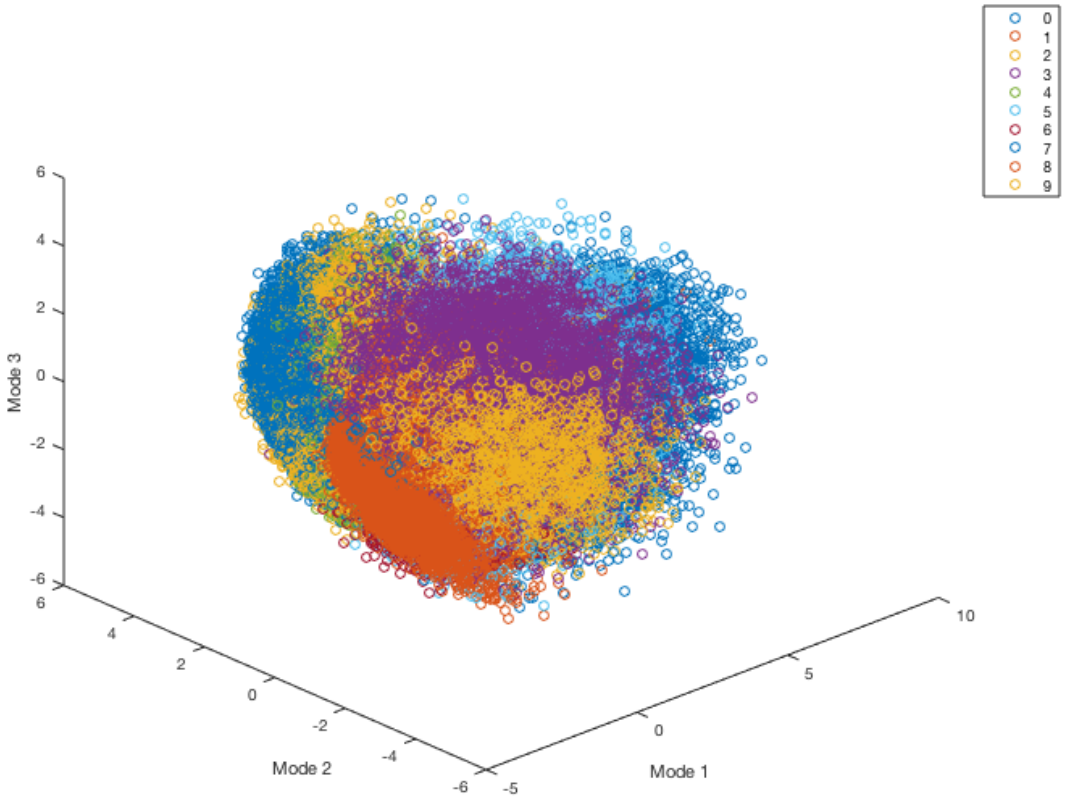


Figure 3: PCA projection for the first 3 mode

The 2 digit in the data set appear to be the most difficult to separate seems to be [5,8], which has the accuracy is 93.35 percent. The 2 digit in the data set appear to be the easiest to separate seems to be [1,4], which has the accuracy is 99.48 percent. Both of the result looks pretty good!

The result of SVM is around 83.31 percent (I tried to see the difference between one vs one and one vs all in the graph). I also tried to do the cross validation. Yet, due to device constrain, I can't get the result on time.

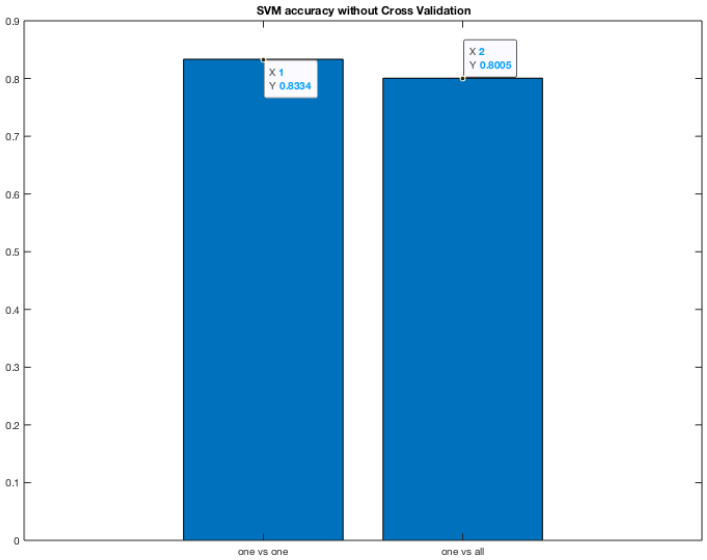


Figure 4: Decision Tree Classifier

The accuracy of decision tree is around 50.25 percent if we set the maxium number of splits to be 10. As we increase the number of splits, the accuracy goes to 82.81 percent. Yet, it took long time to calculate the result.

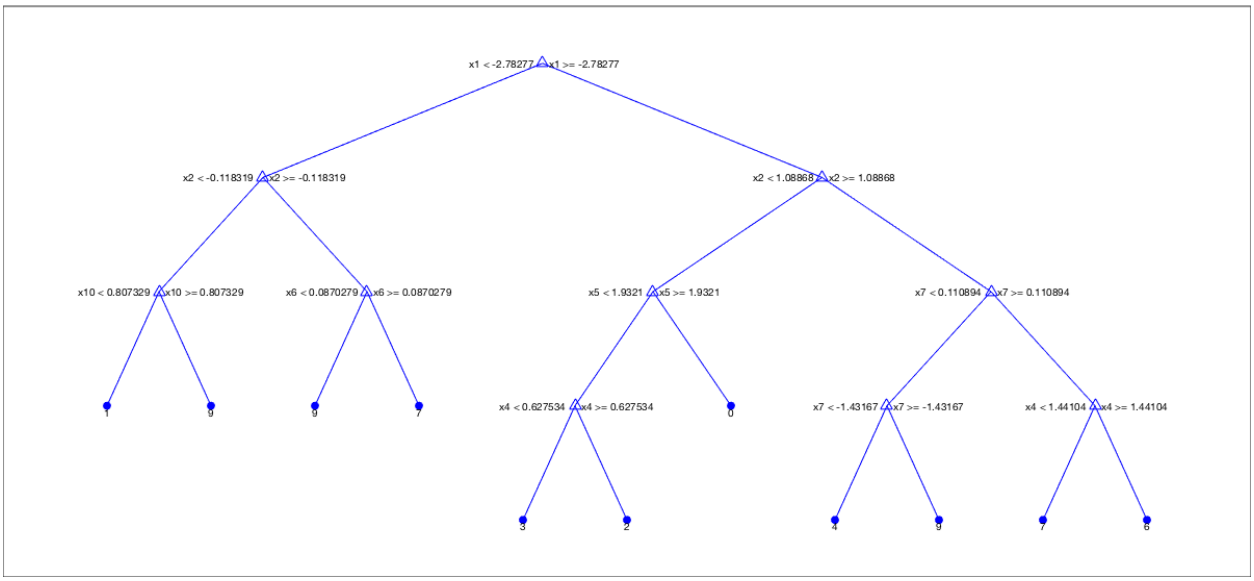


Figure 5: Decision Tree Classifier with maxium number of splits = 10. Notice it didn't classify 5 and 8 (the hardest pair to seperate)

For the easiest pair of digits to separate, [1,4], SVM got the result of 99.48 percent of accuracy and decision tree method got 98.74 percent of accuracy when not restricting its maxium number of splits (if we restrict the number of maxium step, the accuracy will be just a bit lower).

For the hardest pair of digits to separate, [5,8], SVM got the 74.39 percent of accuracy and the decision tree method has 91.35 percent percent of accuracy when not restricting its maxium number of splits.

Overall, for the easiest pair, all three classifier has the really good result with accuracy around 99 percent. Interestingly, when just classifying 2 number, decision tree perform better than SVM when we are testing the hardest pair, while LDA has the highest accuracy. Also, when classifying all 10 digit, SVM seems to perform better than decision tree method.

5 Summary and Conclusions

In this assignment, we see the good example of using SVD to build a classifier. With SVD, we can decomposed system to a classification problem and generate a low-rank approximation. From this approximation, we can pick out only the principal coordinates that best separate the data. Rather than working with the entire dataset, we only have to work with the coordinates of our data in this reduced-dimension space. We also see the difference between LDA, SVM and random forest. From the result we got, it seems like LDA is the best classifier for classifying hand written digit.

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation. This is how to make an **unordered** list:

- `find(X)`: Returns a vector of non-zero indeces in a matrix. We used this in our conditional matrix to find digit in the data.
- `fitcecoc(data, labels)`: Fits a multi-labeled Support Vector Machine with "data", using "labels" to create a model. Can specify cross validation with (..., "crossval", "on") We used this for our digit classification.
- `fitctree(data, labels)`: Fits a Random Forests model with "data", using "labels" to create a model. Can specify cross validation with (..., "crossval", "on") We used this for our music classifcation.
- `kFoldloss(Mdl, data, labels)`: Returns a number between 0 to 1 representing the loss (inaccuracy) of a given cross-volidated model "Mdl", where predicted labels are compared to actual ones ("labels"). We used this to assess the accuracy of our cross validated models.
- `loss(Mdl, data, labels)`: Returns a number between 0 to 1 representing the loss (inaccuracy) of a given model "Mdl", where predicted labels are compared to actual ones ("labels"). We used this to assess the accuracy of our various models.
- `reshape(A,sz)`: Reshapes the array A to the size "sz". We used this to resize our image matrices into a vector.
- `size(X)`: Returns the dimensions of the matrix "X". We used this to calculate size for audio.
- `[u,s,v] = svd(X)`: Returns the U, S, and V matrices corresponded with the singular value decomposition of "X". We used this for build the PCA space for our training data.

Appendix B MATLAB Code

The code for case 1 to 4 are very similar, the only difference is the number setting for threshold and the index number when filtering the image, so I only put the code for the first case.

```
clear all; close all; clc;
load fisheriris

[train_image, train_label] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte
    ');
train_image = im2double(reshape(train_image, size(train_image,1)*size(train_image,2), []).')
    ;
train_label = im2double(train_label);
train_image = train_image'; %784* 60000

[test_image,  test_label] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');
test_image = im2double(reshape(test_image, size(test_image,1)*size(test_image,2), []).');
test_label = im2double(test_label);
test_image = test_image';


% substract row-wise mean
mn = mean(train_image,2); % compute mean for each row
train_image = double(train_image)-repmat(mn,1,length(train_image)); % subtract mean

% Do Singular Value Decomposition(SVD) of matrix
% piled with 60,000 vectorized images and get Principal images
[U, S, V ] = svd(train_image, 'econ');

energy = 0;
total = sum(diag(S));
% how much energy we want our modes to capture.
```

```
% 75% does alright; 90% does very good.
threshold = 0.9;
r = 0;
while energy < threshold
    r = r + 1;
    energy = energy + S(r,r)/total;
end

rank = r;

train_image = (U(:, 1:rank))'*train_image; %project on to PCA compenent
test_image = (U(:, 1:rank))'*test_image;

lamda = diag(S).^2;

%% Plot in PCA space
for i = [0,7]
    Projection_idx = train_image(:, find(train_label == i));
    plot3(Projection_idx(1,:), Projection_idx(2,:), Projection_idx(3,:),...
          'o', 'DisplayName', num2str(i));
    hold on
end
legend show
%legend('0', '1', '2', '3','4', '5', '6','7', '8', '9')
%xlabel('Mode 1')
%ylabel('Mode 2')
%zlabel('Mode 3')

%% Variable Initialization

scrsz = get(groot, 'ScreenSize'); % Get screen width and height

% LDA Step 1. Prepare data matrix
X = train_image;
T = test_image;

% Retrieve data
dimension = size(train_image,1);
Sw = zeros(dimension);
Sb = zeros(dimension);   % Could consider sparse here
N = size(train_image, 2); % Train data size
Nt = size(test_image, 2); % Test data size
Mu = mean(train_image, 2);  % Get mean vector of train data

for i = [0,7] % construct matrix

    % LDA Step 2. Construct Si matrix of each category
    mask = (train_label ==  i);
    x = X(:, mask);
    ni = size(x, 2);
    pi = ni / N;
    mu_i = mean(x, 2);

    Si = (x - repmat(mu_i, [1,ni]))*(x - repmat(mu_i, [1,ni]))';

    % LDA Step 3. Construct Sw within class covariance
    Sw = Sw + Si ;

    % LDA Step 4. Construct Sb between class covariance
    Sb = Sb + (mu_i - Mu) * (mu_i - Mu)';
end

% LDA Step 5. Singular Value Decomposition of Sw\Sb
M = pinv(Sw) * Sb;  % Sw maybe singular, use pseudo-inverse
[U, D, V] = svd(M);

% ============================
%% Plot out the classifier
```

```
%
disp('Task 1: Visualize projected data to 2D and 3D plots');

G2 = U(:,1:rank);
Y2 = G2' * X;

% Plot 2d figure
data2d_fig = figure('Name', '2-D Plot');
set(data2d_fig, 'Position', [60 60 scrsz(3)-120 scrsz(4) - 140]);

for number = [0,7]

    mask = (train_label ==  number);
    a = Y2(1,mask);
    b = Y2(2,mask);

    d = [a'; b'];
    % Draw 2D visualization in separate view
    plot(d, 1*number*ones(size(d)),'o',...
        'DisplayName', num2str(number)); hold on
    title(['LDA classifier']);

end
legend show

%xlim([-0.02 0.02])
 ylim([-1 number+1])

 %% Accuracy Test for LDA
na = 0;
nb = 7;

Y = G2' * X;
Y_t = G2'* T;

train_n= Y(:, find(train_label == na|train_label ==nb));
test_n = Y_t(:, find(test_label == na |test_label ==nb));

accuracy = classifyNN(test_n, train_n,...
    test_label(find(test_label == na |test_label ==nb)), ...
    train_label(find(train_label == na |train_label ==nb)));

%% Defining function

function [accuracy] = classifyNN(test_data, train_data, test_label, train_label)
%
% Description:
% Classify test data using Nearest Neighbor method withEuclidean distance
% criteria.
%
% Usage:
% [accuracy] = classifyNN(test_data, train_data, test_label, train_label)
%
% Parameters:
% test_data = test images projected in reduced dimension  dxtn
% train_data = train images projected in reduced dimension dxN
% test_label = test labels for each data tn x 1
% train_label = train labels for each train data Nx1
%
% Returns:
% accuracy: a scalar number of the classification accuracy

train_size = size(train_data, 2);
test_size = size(test_data, 2);
counter = zeros(test_size, 1);

parfor test_digit = 1:1:test_size

```

```
    test_mat = repmat(test_data(:, test_digit), [1,train_size]);
    distance = sum(abs(test_mat - train_data).^2);
    [M,I] = min(distance);
    if train_label(I) == test_label(test_digit)
        counter(test_digit) = counter(test_digit) + 1;
    end
end

accuracy = double(sum(counter)) / test_size;
end
```

```
clear all; close all; clc;
load fisheriris

[train_image, train_label] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte
    ');
train_image = im2double(reshape(train_image, size(train_image,1)*size(train_image,2), []).')
    ;
train_label = im2double(train_label);
train_image = train_image'; %784* 60000

[test_image,  test_label] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');
test_image = im2double(reshape(test_image, size(test_image,1)*size(test_image,2), []).');
test_label = im2double(test_label);
test_image = test_image';


% substract row-wise mean
mn = mean(train_image,2); % compute mean for each row
train_image = double(train_image)-repmat(mn,1,length(train_image)); % subtract mean

% Do Singular Value Decomposition(SVD) of matrix
% piled with 60,000 vectorized images and get Principal images
[U, S, V ] = svd(train_image, 'econ');

energy = 0;
total = sum(diag(S));
% how much energy we want our modes to capture.
% 75% does alright; 90% does very good.
threshold = 0.9;
r = 0;
while energy < threshold
    r = r + 1;
    energy = energy + S(r,r)/total;
end

rank = r;

train_image = (U(:, 1:rank))'*train_image; %project onto PCA compenent
test_image = (U(:, 1:rank))'*test_image;
%% SVM Modeling

%t1 = templateSVM('KernelFunction','gaussian');
svm_model = fitcecoc(train_image',train_label);
%svm_model1 = fitcecoc(train_image',train_label,'Coding','onevsone','Learners',t);
%svm_model2 = fitcecoc(train_image',train_label,'Coding','onevsall','Learners',t);

result = predict(svm_model,test_image');
%result1 = predict(svm_model1,test_image');
%result2 = predict(svm_model2,test_image');
%svmL1 = loss(svm_model1, test_image.', test_label);
%svmL2 = loss(svm_model2, test_image.', test_label);

svmL = loss(svm_model2, test_image.', test_label);
figure()
bar(1-[svmL1 svmL2])
xticklabels( {'one vs one', 'one vs all'});
title("SVM accuracy without Cross Validation");
```

```
%% Decision Tree Modeling

rfMdl = fitctree(train_image.', train_label,'CrossVal','on');
rfL = kfoldLoss(rfMdl, 'LossFun','ClassifErr');
%view(rfMdl.Trained{1},'Mode','graph');
figure()
bar(1-rfL)
xticklabels([ "Accuracy of Decision Tree"]);
title("Decision Tree accuracy with Cross Validation");

%% Decided pair of digit
na = 1;
nb = 4;

train_2 = train_image(:, find(train_label == na|train_label == nb));
label_2 = train_label(find(train_label == na|train_label == nb));

test_2 = test_image(:, find(test_label == na|test_label == nb));
tlabel_2 = test_label(find(test_label == na|test_label == nb));

%% modeling SVM
svm_best = fitcecoc(train_2',label_2);
svml_best = loss(svm_best, test_2.', tlabel_2);

%% modeling Tree
rfMdl = fitctree(train_2.', label_2,'CrossVal','on');

rfL = kfoldLoss(rfMdl, 'LossFun','ClassifErr');
view(rfMdl.Trained{1},'Mode','graph');
```

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

% 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
```