Classifying Digits from the MNIST Data Set

Jiajun Bao

Abstract

This project aims to first perform an SVD analysis of the digit images from the MNIST data set, then using Linear Discriminant Analysis(LDA), support vector machines(SVM), and decision tree classifiers to identify individual digits in the training set and classify digits in the testing set.

1 Introduction and Overview

In this project, we are provided with a training set of 60,000 examples of handwritten digits and a test set of 10,000 examples handwritten digits. These digits have been size-normalized and centered in a fixed-size image. The goal is to classify digits in the testing set images from the MNIST data set with a linear classifier (LDA) and then compare the performance of two other classifier support vector machines(SVM) and decision tree classifiers on the same data set. In order to achieve this, we first perform an SVD analysis of this data set and project the data into PCA space. We build the linear classifier to identify individual digits in the training set, test the accuracy of the classification on both the training and test sets, and find the two digits in the data set that are most easy/difficult to separate. Then we compare the performance between LDA, SVM, and decision trees on the separation of between all ten digits and the hardest and easiest pair of digits.

2 Theoretical Background

2.1 The Singular Value Decomposition

The Singular Value Decomposition (SVD) is a very powerful tool that is widely used in data analysis to produce low-rank approximations of a data set, which is described as following:

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

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are unitary matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is diagonal. The values σ_n on the diagonal of Σ are the singular values of the matrix A. The vectors u_n which makes up the columns of U are the left singular vectors of A. The vectors v_n which make up the columns of V are the right singular vectors of A. The SVD tells us how to keep the most amount of information while keeping the fewest number of dimension of the data. The large singular values help us pick out which directions are most important.

2.2 Principal Component Analysis

The goal of principal component analysis is to find a new set of coordinates(a change of basis) so that the variables are now uncorrelated and each variables contains completely new information. This is very helpful in this project as each image is 28x28 pixels, and after reshaping, it turns to to 784x1 vector, and we have 60,000 images in the training data and 10,000 images in the testing data. By utilizing the PCA, we can increase the efficiency of computation while preserving the important characteristics of images. More details will be discussed in next section. To do this, we first compute the covariances matrix, and the eigenvalues of the covariance matrix are the squares of the scaled singular values, making it an unbiased estimator. The steps is shown in the following:

Consider

$$\mathbf{A} = \frac{\mathbf{1}}{\sqrt{\mathbf{n-1}}}\mathbf{X}, \mathbf{C_x} = \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{X}\mathbf{X^T} = \mathbf{A}\mathbf{A^T} = \mathbf{U}\mathbf{\Sigma^2}\mathbf{U^T} \tag{2}$$

The data in the new coordinates is

$$\mathbf{Y} = \mathbf{U^T X} \tag{3}$$

The covariance of $\mathbf{Y}$ is

$$\mathbf{C_x} = \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{Y Y^T} = \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{U^T X X^T U} = \mathbf{U^T A A^T U} = \mathbf{U^T U \Sigma^2 U^T U} = \mathbf{\Sigma^2} \tag{4}$$

And the variables in $\mathbf{Y}$ are uncorrelated as off-diagonal elements of Σ are zero.

2.3 Linear Discriminant Analysis(LDA)

Linear Discriminant Analysis(LDA) is used for dimensionality reduction and pattern classification of data sets by projecting onto the suitable subspace that maximizes the distance between the inter-class data while minimizing the intra-class data. For 2 datasets and more than two groups, we define the **between-class scatter matrix**, which measures the variance between groups as:

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T / \mathbf{S}_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T, N \geq 3 \tag{5}$$

where for two groups, μ_1 and μ_2 are means of each of groups, and this measures the variance between the groups; for three or more groups, μ is the overall mean and μ_j is the mean of each groups. Then we define the **within-class scatter matrix**, which is the measure of the variance within each group as:

$$\mathbf{S}_w = \sum_{j=1}^{2} \sum_{X} (X - \mu_j)(X - \mu_j)^T / \mathbf{S}_w = \sum_{j=1}^{N} \sum_{X} (X - \mu_j)(X - \mu_j)^T, N \geq 3 \tag{6}$$

We need the vector $\mathbf{w}$ such that $\mathbf{w} = argmax \frac{W^T S_B w}{W^T S_w w}$ and it is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem $S_B \mathbf{w} = \lambda\ S_w \mathbf{w}$, then we project onto w and choose cutoff values.

2.4 Support Vector Machines(SVM)

The Support vector machine (SVM) is a supervised learning algorithm used for many classification and regression problems. The objective of the SVM algorithm is to find a hyperplane that, to the best degree possible, separates data points of one class from those of another class. [1]

2.5 Decision Trees

Decision trees, or classification trees and regression trees, predict responses to data. To predict a response, follow the decisions in the tree from the root (beginning) node down to a leaf node. The leaf node contains the response. Classification trees give responses that are nominal, such as 'true' or 'false'. Regression trees give numeric responses. [2]

3 Algorithm Implementation and Development

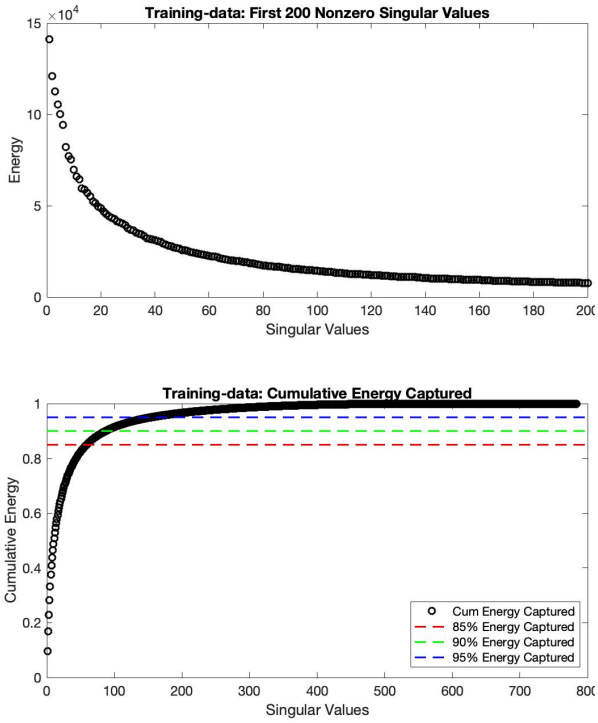
- Read from **train-images-idx3-ubyte** and **train-labels-idx1-ubyte** using helper function *mnist_parse* and get the 60000 28x28 pixels images of handwritten digits and their corresponding label representing the digit of each images.
- Reshape each image into a column vector and each column of the data matrix is a different image, then convert the data matrix to double and subtracting the row wise mean for SVD analysis. Then do the SVD analysis of the digit images data matrix using *svd*.
- Plot singular value spectrum where x coordinate is the singular values and y coordinate is the energy captured by each singular value, then also plot the graph with cumulative energy capture, and add three reference lines representing 85%, 90%, and 95% energy captured.

- Calculate numbers of modes needed for capturing 85%, 90%, and 95% energy, which means preserving 85%, 90%, and 95% of the original image using while loops. Then plot the first occurrence of each digit 0-9's reconstruction image in the data matrix and with 85%,90%,95% Energy.
- Plot the first nine principal components. Then we calculate the projection onto three selected V-modes (columns 2,3, and 5) of 10 digits and then plot the projection colored by their digit label using for loop. We choose to manually plot digit 7,8,9 with colors different form default plotting color, as in *plot3* there are only 7 default color.
- After performed the above analysis, we build the linear classifier(LDA) from the training data set that identify digits from testing data set. We choose to use 59 features, which is the number of nodes that capture 85% energy as it is determined in the previous plots 85% will give us a good image reconstruction.
- Read from **t10k-images-idx3-ubyte** and **t10k-labels-idx1-ubyte** using helper function *mnist_parse* and get the 10000 28x28 pixels images of handwritten digits and their corresponding label representing the digit of each images, this is the testing data set.
- Reshape and demean the testing data following same steps of training data, then do the SVD analysis of the digit images data matrix using *svd*.
- Sort the training data so that the test labels are in increasing order, then change the training data according to the sequence of test sorted test labels order.
- Use the *classify* function to find the prediction of the testing data and then plot the prediction for one of examples for identify two digits using *bar*. In here, we choose digit 0 and 1. We repeat the same procedures for identifying three digits 0,1, and 2.
- Run through all combination of two digits using a for loop and *combntns*, then we find the two digits that are most difficult to separate and two digits that are most easy to separate.
- Implement Matlab built-in funciton *fitctree* for decision tree classifier, then find the performance of the decision tree classifier on both the training and test sets using function *predict* and a for loop that compare each label in the prediction and original label vector. We repeat the same process for SVM Classifier using built-in funciton *fitcecoc*.
- Then we compute the performance of SVM and decision trees on the hardest and easiest pair of digits to separate (obtained from the previous LDA steps). We compare the performance between LDA, SVM and decision trees on these pairs by calculating the percentage of classification compared to original label vector.

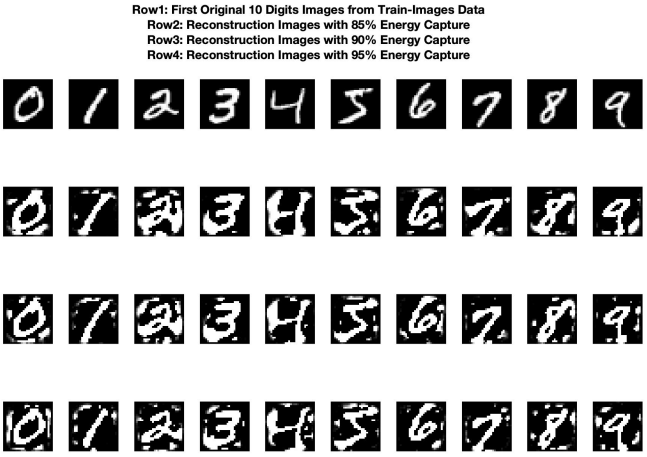
4 Computational Results

4.1 Singular Value Spectrum and Images Reconstruction

Based on the above algorithms in section 3, please refer to Appendix B for detailed codes in Matlab, The first 200 Nonzero Singular Values of the training data and the cumulative energy captured is plotted in Figure 1(a). Besides, the three dashed reference lines representing 85%(red), 90%(green), and 95%(blue) energy captured. We can see that the cumulative energy is gradually increasing as we consider more modes. Then, I also plotted the original(1st row) and reconstruction images of the first occurrence of digits 0-9 in the data set in Figure 1(b) to help us determine the modes required for good image reconstruction. Based on the plot, 85% energy capture (59 nodes) will present us with clear images of each digit except for digit 4, which is slightly unclear. Suppose we want to get more clear images for all digits, we can increase the modes to 87 modes(90% captured), 154 modes(95% energy captured), but this will also make the computation slower as more elements in SVD are taken into consideration. Overall, 85% energy capture with 59 nodes perform well and it is the number of feature we will use to classify digits later.



(a) Training-data: Singular Values and Energy Captured



(b) Images Reconstruction

Figure 1: Singular Value Spectrum and Images Reconstruction

4.2 Interpretation of U, Σ, and V matrices & Projection onto V-modes 2, 3, 5

- **U**: The Left Singular Vectors. It represents the principal components, which are the most important feature of all the digits in the data set.
- Σ: The singular values tell us how much weight each principal components carries, which is the strength of the projection. The first mode is the most dominant one.
- **V**: The Right Singular Vectors speaks for how each digit is represented in the PCA basis.

The 3D plot of Projection onto V-modes 2, 3, 5 is plotted in Figure 2. In Figure 2, we can see how each digit is projected into PCA space.

4.3 Linear Classifier: Linear Discriminant Analysis

Based on the above algorithms in section 3, please refer to Appendix B for detailed codes in Matlab, we are able to build a linear classifier that perform well on separating between all ten digits. Figure 3(a) top figure shows the one example of two digit identification. In here, digit 0 and 1 are chosen. Figure 3 bottom shows the one example of three digit identification, and digit 0, 1, and 2 are chosen.

We also calculated the misclassification error rate with first 59 features from Linear Discriminant Analysis, which is plotted in Figure 3(b). We can see that the most difficult to separate is digit 3 and 5, with approximately 4.3% error rate. This is something we are also expecting prior to the calculation because digit 3 and 5 may look very similar in handwriting, and since we only have 28x28 pixels for each image, it is hard for both human and machine to pick out the difference between these two digits. In contrast, digit 6 and 7 are most easy to separate, with approximately 0.173%) error rate. This is due to most of the writing for digit 6 is at the lower half part, and most distinct part of the writing for digit 7 is at the top half part. This makes the difference of two digits more outstanding compared with other pairs.

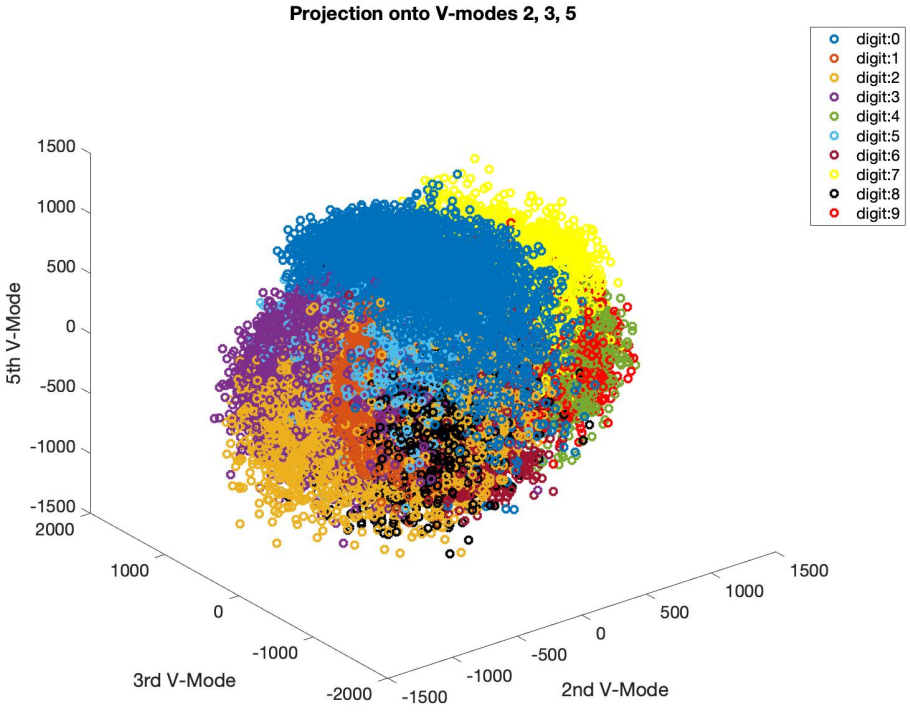
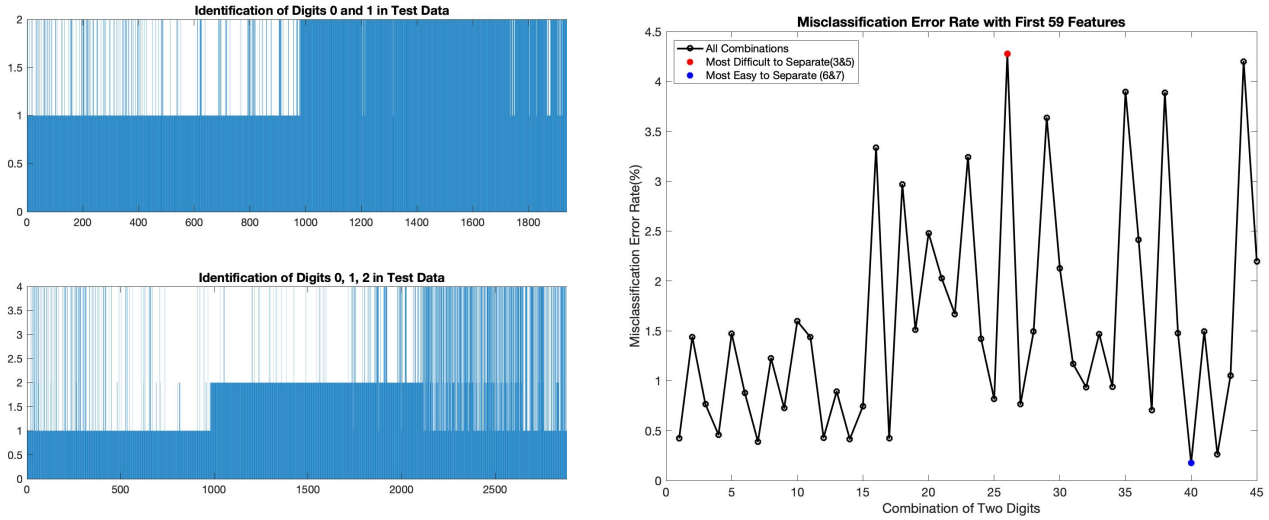


Figure 2: Projection onto V-modes 2, 3, 5



(a) Example of Two Digits and Three Digits Classification

(b) Images Reconstruction

Figure 3: Linear Discriminant Analysis: Misclassification Error Rate with First 59 Features

4.4 Support Vector Machines(SVM) and Decision Tree Classifiers

In the last part, we use another two types of classifiers, which are Support Vector Machines(SVM) and Decision Tree Classifiers to separate between all ten digits. We plotted in Figure 4 a demonstration of classification tree that has about 30 splits, but in calculation, we let the Matlab automatically decide the MaxNumSplits for optimize performance. Based on the above algorithms in section 3, we found that:

Using the Decision Tree Classifier, the accuracy for training data for classifying ten digits is about 98.36%; the accuracy for testing data for classifying ten digits is about 65.12%. Using the Support Vector Machines(SVM), the accuracy for training data for classifying ten digits is about 77.22%; the accuracy for testing data for classifying ten digits is about 71.46%. It turns out both of the method have lower accuracy compared with Linear Discriminant Analysis in the testing data, which LDA only misclassify about 2.18% of all the digits on average. For the training data set, decision tree classifier and LDA perform well with accuracy over 98%, but the SVM still doesn't seem to be as effective as the other two methods. This maybe caused by oversampling problem in the SVM method. We also get similar performance on separating the hardest and easiest pair of digits among three classifiers that SVM seems to not perform well as the other two.

	LDA	Decision Tree	SVM
Easiest Pair of digits to Separate Error Rate(%)	0.17	0.54	0.43
Hardest Pair of digits to Separate Error Rate(%)	4.28	5.6	14.3

5 Summary and Conclusions

In this problem, we first perform an SVD analysis of the training data set and testing data set, then we project the data matrix into PCA space. After that, we build a linear classifier (LDA) and compare it with SVM (support vector machines) and decision tree classifiers. At the end, we found that for this particular data set, the linear classifier (LDA) and decision tree classifier perform well with high accuracy, whereas the SVM doesn't perform as well as LDA and decision tree classifier, which could be caused by over-fitting problem and require further investigation.

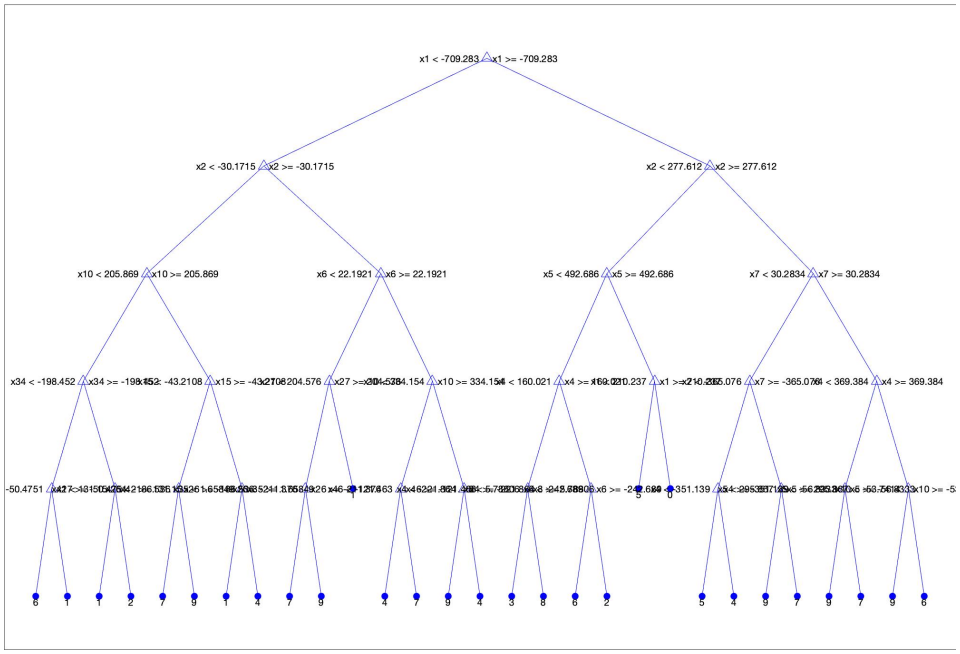


Figure 4: Demonstration of Classification Tree with 30 splits

References

[1] MathWorks, Support Vector Machine (SVM) URL: https://www.mathworks.com/discovery/support-vector-machine.html?s_tid=srchtitle

[2] MathWorks, Decision Trees URL:https://www.mathworks.com/help/stats/decision-trees.html

Appendix A MATLAB Functions

- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `B = reshape(A,sz)` reshapes A using the size vector, sz, to define size(B).
- `B = rescale(A)` scales the entries of an array to the interval [0,1]. The output array B is the same size as A.
- `combos = combntns(set,subset)` returns a matrix whose rows are the various combinations that can be taken of the elements of the vector set of length subset.
- `tree = fitctree(X,Y)` returns a fitted binary classification decision tree based on the input variables contained in matrix X and output Y. The returned binary tree splits branching nodes based on the values of a column of X.
- `ypred = predict(mdl,Xnew)` returns the predicted response values of the linear regression model mdl to the points in Xnew.
- `Mdl = fitcecoc(X,Y)` returns a trained ECOC model using the predictors X and the class labels Y.
- `[class,err]= classify(...)` classifies each row of the data in sample into one of the groups in training,also returns an estimate err of the misclassification error rate based on the training data. classify returns the apparent error rate,

Appendix B MATLAB Code

```
clear; close all; clc

[images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
% reshape each image
for i = 1:length(labels)
    imag(:,i) = reshape(images(:,:,i),784,1);
end
% subtracting the row wise mean
imag1 = double(imag);
for j = 1:size(imag1,1)
    imag1(j,:) = imag1(j,:) - mean(imag1(j,:));
end
%  SVD analysis of the digit images
[U,S,V] = svd(imag1,'econ');
sig1 = diag(S);

%% Plot singular value spectrum
figure(1)
subplot(2,1,1)
plot(diag(S),'ko','Linewidth',1.5)
title("Training-data: First 200 Nonzero Singular Values");
set(gca,'Fontsize',12,'Xlim',[0 200])
```

```
xlabel("Singular Values"); ylabel("Energy");
subplot(2,1,2)
plot(cumsum(sig1.^2/sum(sig1.^2)),'ko','Linewidth',1.5)
title("Training-data: Cumulative Energy Captured");
set(gca,'Fontsize',12,'Xlim',[0 800])
xlabel("Singular Values"); ylabel("Cumulative Energy");
hold on
plot([0 800], [0.85 0.85], '--r', 'Linewidth',1.5)
plot([0 800], [0.90 0.90], '--g', 'Linewidth',1.5)
plot([0 800], [0.95 0.95], '--b', 'Linewidth',1.5)
legend('Cum Energy Captured','85% Energy Captured','90% Energy Captured','95% Energy Captured')

%% Finding # of modes necessary for good image reconstruction
% Capture at least 85% Engery
capture_percent = 0.85;
energy1 = 0;
nodes1 = 0;
while energy1 < capture_percent
    nodes1 = nodes1 + 1;
    sig1 = diag(S);
    % get each singular value
    current_var1 = sig1(nodes1);
    % energy captured by each nonzero singular value
    each_energy1 = current_var1.^2/sum(sig1.^2);
    % find cumulative energy for first n-modes
    energy1 = energy1 + each_energy1;
end
disp(nodes1)
% Capture at least 90% Engery
capture_percent = 0.90;
energy2 = 0;
nodes2 = 0;
while energy2 < capture_percent
    nodes2 = nodes2 + 1;
    sig2 = diag(S);
    current_var2 = sig2(nodes2);
    each_energy2 = current_var2.^2/sum(sig2.^2);
    energy2 = energy2 + each_energy2;
end
disp(nodes2)
% Capture at least 95% Engery
capture_percent = 0.95;
energy3 = 0;
nodes3 = 0;
while energy3 < capture_percent
    nodes3 = nodes3 + 1;
    sig3 = diag(S);
    % get each singular value
    current_var3 = sig3(nodes3);
    % energy captured by each nonzero singular value
    each_energy3 = current_var3.^2/sum(sig3.^2);
    % find cumulative energy for first n-modes
    energy3 = energy3 + each_energy3;
end
```

```
disp(nodes3)

%% Plot first image Digit 0-9's reconstruction image and with 85,90,95% Energy
figure(2)
for k = 1:10
    % plot original digit 0-9
    subplot(4,10,k)
    label_index = find(labels == k-1,1);
    imshow(images(:,:,label_index))
    sgtitle({'Row1: First Original 10 Digits Images from Train-Images Data',...
        'Row2: Reconstruction Images with 85% Energy Capture',...
        'Row3: Reconstruction Images with 90% Energy Capture',...
        'Row4: Reconstruction Images with 95% Energy Capture'},'fontweight','bold','FontSize',16)
    % plot original digit 0-9 with 85% energy capture
    imag_est_85 = U(:,1:nodes1)*S(1:nodes1,1:nodes1)*V(:,1:nodes1)';
    subplot(4,10,10+k)
    imag_est_85 = reshape(imag_est_85(:,label_index),28,28);
    imshow(imag_est_85)
    % plot original digit 0-9 with 90% energy capture
    imag_est_90 = U(:,1:nodes2)*S(1:nodes2,1:nodes2)*V(:,1:nodes2)';
    subplot(4,10,20+k)
    imag_est_90 = reshape(imag_est_90(:,label_index),28,28);
    imshow(imag_est_90)
    % plot original digit 0-9 with 95% energy capture
    imag_est_95 = U(:,1:nodes3)*S(1:nodes3,1:nodes3)*V(:,1:nodes3)';
    subplot(4,10,30+k)
    imag_est_95 = reshape(imag_est_95(:,label_index),28,28);
    imshow(imag_est_95)
end

%% Interpretation of the U a matrices
% Interpretation of the U
figure(3)
for k = 1:9
   subplot(3,3,k)
   ut1 = reshape(U(:,k),28,28);
   ut2 = rescale(ut1);
   imshow(ut2)
   sgtitle('First Nine Principal Components','fontweight','bold')
end

%%  Projection onto three selected V-modes colored by their digit label
figure(5)
proj = S*V';  %project onto three selected V-modes
for i=1:7
    digit = i-1;
    str ='digit:' + string(digit);
    label_indices = find(labels == digit);
    plot3(proj(2, label_indices), proj(3, label_indices), proj(5, label_indices),'o', ...
    'DisplayName', str, 'Linewidth', 2)
    hold on
end
% Manullay plot last 3 digits due to there's only 7 default color for plot3
label_indices = find(labels == 7);
```

```
plot3(proj(2, label_indices), proj(3, label_indices), proj(5, label_indices),...
'o', 'DisplayName', 'digit:7', 'Linewidth', 2, 'Color',[1,1,0])
label_indices = find(labels == 8);
plot3(proj(2, label_indices), proj(3, label_indices), proj(5, label_indices),...
     'o', 'DisplayName', 'digit:8', 'Linewidth', 2, 'Color',[0 0 0])
label_indices = find(labels == 9);
plot3(proj(2, label_indices), proj(3, label_indices), proj(5, label_indices),...
     'o', 'DisplayName', 'digit:9', 'Linewidth', 2, 'Color',[1 0 0])
xlabel('2nd V-Mode'), ylabel('3rd V-Mode'), zlabel('5th V-Mode')
title('Projection onto V-modes 2, 3, 5')
legend
set(gca,'Fontsize', 14)

%% linear classifier (LDA) that identify 2 digits
[labels,I] = sort(labels);
imag1 = imag1(:,I');

%  SVD analysis of the digit images
[U,S,V] = svd(imag1,'econ');
sig1 = diag(S);
feature = nodes3;     % use first 59 nodes capture 85% energy
imag_proj = S(1:feature,1:feature)*V(:,1:feature)';
error_list = [];

% load test data: linear classifier (LDA) that identify 2 digits
[test_images, test_labels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');
% reshape each image
for i = 1:length(test_labels)
    imag_test(:,i) = reshape(test_images(:,:,i),784,1);
end
% subtracting the row wise mean
imag1_test = double(imag_test);
for j = 1:size(imag1_test,1)
    imag1_test(j,:) = imag1_test(j,:) - mean(imag1_test(j,:));
end
[test_labels,Ind] = sort(test_labels);
imag1_test= imag1_test(:,Ind');

%  SVD analysis of the test digit images
[U_test,S_test,V_test] = svd(imag1_test,'econ');
imag_proj_test = S_test(1:feature,1:feature)*V_test(:,1:feature)';

%% One Example for 2 digits and 3 digits
figure(6)
% 2 Digits Example: digit 0 and 1
label_first_indices = find(labels == 0);  % find all first digit images in train data
digit_first = imag_proj(:,label_first_indices);
label_second_indices = find(labels == 1);  % find all second digit images in train data
digit_second = imag_proj(:,label_second_indices);
num_first = length(label_first_indices);  % # num of first digit images in train data
num_second = length(label_second_indices);  % # num of second digit images in train data
xtrain = [digit_first'; digit_second'];
ctrain = [2*ones(num_first,1); ones(num_second,1)];
test_label_first_indices = find(test_labels == 0);  % find all first digit images in test data
```

```
test_digit_first = imag_proj_test(:,test_label_first_indices);
test_label_second_indices = find(test_labels == 1);  % find all second digit images in test data
test_digit_second = imag_proj_test(:,test_label_second_indices);
xtest = [test_digit_first'; test_digit_second'];
[prediction] = classify(xtest,xtrain,ctrain);
subplot(2,1,1)
bar(prediction)
set(gca,'Fontsize',12)
title('Identification of Digits 0 and 1 in Test Data','fontweight','bold','FontSize',14)

% 3 Digits Example: digit 0, 1 and 2
label_first_indices = find(labels == 0);  % find all first digit images in train data
digit_first = imag_proj(:,label_first_indices);
label_second_indices = find(labels == 1);  % find all second digit images in train data
digit_second = imag_proj(:,label_second_indices);
label_third_indices = find(labels == 2);  % find all second digit images in train data
digit_third = imag_proj(:,label_third_indices);
num_first = length(label_first_indices);  % # num of first digit images in train data
num_second = length(label_second_indices);  % # num of second digit images in train data
num_third = length(label_third_indices);  % # num of second digit images in train data
xtrain = [digit_first'; digit_second'; digit_third'];
ctrain = [2*ones(num_first,1); ones(num_second,1); 4*ones(num_third,1)];
test_label_first_indices = find(test_labels == 0);  % find all first digit images in test data
test_digit_first = imag_proj_test(:,test_label_first_indices);
test_label_second_indices = find(test_labels == 1);  % find all second digit images in test data
test_digit_second = imag_proj_test(:,test_label_second_indices);
test_label_third_indices = find(test_labels == 2);  % find all second digit images in test data
test_digit_third = imag_proj_test(:,test_label_third_indices);
xtest = [test_digit_first'; test_digit_second'; test_digit_third'];
[prediction] = classify(xtest,xtrain,ctrain);
subplot(2,1,2)
bar(prediction)
set(gca,'Fontsize',12)
title('Identification of Digits 0, 1, 2 in Test Data','fontweight','bold','FontSize',14)

%% Run through all combination of two digits
comb_2_digits = combntns(0:9,2);
for i = 1:45
    % Train data
    digit_1st = comb_2_digits(i,1);
    digit_2nd = comb_2_digits(i,2);
    label_first_indices = find(labels == digit_1st);
    digit_first = imag_proj(:,label_first_indices);
    label_second_indices = find(labels == digit_2nd);
    digit_second = imag_proj(:,label_second_indices);
    num_first = length(label_first_indices);
    num_second = length(label_second_indices);
    xtrain = [digit_first'; digit_second'];
    ctrain = [ones(num_first,1); 2*ones(num_second,1)];
    % Test data
    test_label_first_indices = find(test_labels == digit_1st);
    test_digit_first = imag_proj_test(:,test_label_first_indices);
    test_label_second_indices = find(test_labels == digit_2nd);
    test_digit_second = imag_proj_test(:,test_label_second_indices);
```

```
    xtest = [test_digit_first'; test_digit_second'];
    [prediction,err] = classify(xtest,xtrain,ctrain);
    error_list(1,i) = err;
end

%% Plot all All Combinations Error Rate
[Max_Err,I1] = max(error_list); % 0.04275
[Min_Err,I2] = min(error_list); % 0.00173
figure(7)
plot(1:45,100.*error_list,'-ko', 'Linewidth', 2)
ylabel('Misclassification Error Rate(%)')
xlabel('Combination of Two Digits')
hold on
plot(I1,100.*Max_Err,'r*','Linewidth', 4)
plot(I2,100.*Min_Err,'b*','Linewidth', 4)
legend('All Combinations','Most Difficult to Separate(3&5)','Most Easy to Separate (6&7)')
set(gca,'Fontsize',14)
title('Misclassification Error Rate with First 59 Features','fontweight','bold','FontSize',16)

%%  Decision Tree Classifier
data = imag_proj';
tree = fitctree(data,labels);
% Demo of tree plot for 30 MaxNumSplits
tree1=fitctree(data,labels,'MaxNumSplits',30,'CrossVal','on');
view(tree1.Trained{1},'Mode','graph');

%% Decision Tree-Traning Data Accuracy between all ten digits 0.9836
train_labels_tree = predict(tree,data);
counter = 0;
train_labels = labels;
train_labels_tree = sort(train_labels_tree);
for i = 1:length(train_labels_tree)
    pre = train_labels_tree(i,1);
    actual = train_labels(i,1);
    if pre == actual
        counter = counter + 1;
    end
end

%% most easy + most hard
easy_digit_6_num = length(find(train_labels_tree == 6));
easy_digit_7_num = length(find(train_labels_tree == 7));
total_num_easy = easy_digit_6_num + easy_digit_7_num;
data_total_num_easy = length(find(labels == 6)) + length(find(labels == 7));
easy_acc = abs(total_num_easy-data_total_num_easy) / data_total_num_easy;
hard_digit_3_num = length(find(train_labels_tree == 3));
hard_digit_5_num = length(find(train_labels_tree == 5));
total_num_hard = hard_digit_3_num + hard_digit_5_num;
data_total_num_hard = length(find(labels == 3)) + length(find(labels == 5));
hard_acc = abs(total_num_hard-data_total_num_hard) / data_total_num_hard;

%% Decision Tree-Testing Data Accuracy
test_data = imag_proj_test';
test_labels_tree = predict(tree,test_data);
```

```
counter_test = 0;
test_labels_tree = sort(test_labels_tree);
for i = 1:length(test_labels_tree)
    pre = test_labels_tree(i,1);
    actual = test_labels(i,1);
    if pre == actual
        counter_test = counter_test + 1;
    end
end

%% SVM Classifier
% keep data small for computing efficienty, by the inverse of the largest singular value.
data_1 = imag_proj' ./ max(diag(S));
Mdl = fitcecoc(data_1,labels);
% SVM-Traning Data Accuracy between all ten digits
train_labels_SVM = predict(Mdl,data_1);
counter = 0;
train_labels = labels;
train_labels_SVM = sort(train_labels_SVM);
for i = 1:length(train_labels_SVM)
    pre = train_labels_SVM(i,1);
    actual = train_labels(i,1);
    if pre == actual
        counter = counter + 1;
    end
end
% svm-Testing Data Accuracy
test_labels_SVM = predict(Mdl,test_data);
counter_test = 0;
test_labels_SVM = sort(test_labels_SVM);
for i = 1:length(test_labels_SVM)
    pre = test_labels_SVM(i,1);
    actual = test_labels(i,1);
    if pre == actual
        counter_test = counter_test + 1;
    end
end
easy_digit_6_num = length(find(train_labels_SVM == 6));
easy_digit_7_num = length(find(train_labels_SVM == 7));
total_num_easy = easy_digit_6_num + easy_digit_7_num;
data_total_num_easy = length(find(labels == 6)) + length(find(labels == 7));
easy_acc = abs(total_num_easy-data_total_num_easy) / data_total_num_easy;
hard_digit_3_num = length(find(train_labels_SVM == 3));
hard_digit_5_num = length(find(train_labels_SVM == 5));
total_num_hard = hard_digit_3_num + hard_digit_5_num;
data_total_num_hard = length(find(labels == 3)) + length(find(labels == 5));
hard_acc = abs(total_num_hard-data_total_num_hard) / data_total_num_hard;
```