AMATH 482 Assignment 4

Megan Freer

March 10, 2021

Abstract

Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA), a sub-concept of Singular Value Decomposition (SVD), are powerful mathematical tools, particularly for image recognition. These tools allow us to classify images, such as images of digits 0 to 9, based on their principal components, and therefore analyze the digits that are the most difficult and easy to separate. This provides an interesting test case for the application of LDA, and also allows us to compare the image recognition accuracy of LDA with other classification methods, such as SVM (support vector machines) and decision tree classifiers.

1 Introduction and Overview

Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) are useful mathematical tools for image recognition. LDA is that idea that two data sets can be projected onto new bases, while Principal Component Analysis (PCA) stems from the idea of Singular Value Decomposition (SVD) and is a way of finding out information about a matrix that represents a group of images. For example, we can use PCA to find a system's rank, or how many principal components are needed to represent the majority of what is being visually represented in an image. SVM (support vector machines) and decision tree classifiers are two more methods of classifying and distinguishing between images that were considered state-of-the-art until about 2014 that will also be explored in this assignment.

As an example of using these methods (LDA, SVM, and decision tree classifiers), we will be looking at distinguishing between images of digits 0 to 9. We are provided with 60,000 images of digits and their correct labels as part of the training data set, as well as 10,000 images of digits and their correct labels as part of the test data set. Every image is 28 pixels by 28 pixels in size, meaning very low resolution. These data sets allow us to train our models on how to identify between particular digits and then test our model to find its accuracy.

We will be using MATLAB to perform the Singular Value Decomposition function on the combined test image data. This then allows us to compare between LDA, decision trees, and SVM in terms of the accuracies of the predicted digit image labels. We will be exploring these methods and their accuracies when distinguishing between two, three, or more digits, as well as when testing on our test and training data separately.

2 Theoretical Background

Principal Component Analysis (PCA), a sub-concept of Singular Value Decomposition (SVD), and Linear Discriminant Analysis (LDA) are powerful mathematical tools, particularly for image recognition. Image recognition provides an example of when data-driven approaches (machine learning) are useful, so we will be working with training data that is images with labels. This model can then be used to predict some test data (images with labels), although the labels are not put into the machine learning algorithm, but rather compared with its output. This algorithm works by finding the principal components of the images to see how digits differ in the principal component basis. The next step is to use Linear Discriminant Analysis (LDA) to determine the thresholds that separate digits, then test the algorithm on the test images to see the accuracy.

Going into more detail about Linear Discriminant Analysis (LDA), the idea behind LDA is that two data sets (for example, two digits) are projected onto new bases. The ideal structure for LDA allows data/images to be completely separated. In particular, this means when the means μ_1 and μ_2 of the two data sets are significantly apart when projected onto the chosen subspace. In other words, the goal of LDA is to find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data. Going into the math behind LDA, in order to find the right subspace to project onto, we first calculate the means for each of our groups for each feature. These means will be called μ_1 and μ_2. These means are column vectors since they are means across each row. We can then define the between-class scatter matrix:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{1}$$

This is a measure of the variance between the groups, meaning between the means. Next we can define the within-class scatter matrix:

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

This is a measure of the variance within each group. We then want to find a vector w such that:

$$w = argmax \frac{w^T S_B w}{w^T S_w w} \tag{3}$$

The vector w that maximizes the above quotient is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem:

$$S_B w = \lambda S_w w \tag{4}$$

This can be solved in MATLAB using the "eig" command. After finding w using the previous equations, we can find a cutoff between the two labels/digits to be used as a threshold for decision making. When classifying between more than two groups (such as between three or more digits), we can then define the between-class scatter matrix as:

$$S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T \tag{5}$$

where μ is the overall mean and μ_j is the mean of each of the $N \geq 3$ classes. In this case, the within-class scatter matrix then becomes:

$$S_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{6}$$

w is then found as described above. This math behind LDA is incorporated into the MATLAB `classify` command. This command performs linear discriminant analysis when given inputs of training images, training labels, and test images.

It is important to note that Singular Value Decomposition (SVD) is also very useful in this example as a method to get our principal components. Performing Principal Component Analysis (PCA) tells us about the ways in which we have the most variation in our data, which will help us pick out some of the most important features of our data. The equation for the SVD in matrix form is:

$$AV = U\Sigma \tag{7}$$

where V is a unitary matrix with v_1 and v_2 as its columns, U is a unitary matrix with u_1 and u_2 as its columns, and Σ is a diagonal matrix with σ_j on its diagonals. Since V is unitary and therefore invertible, we can isolate for A to get:

$$A = U\Sigma V^* \tag{8}$$

The values σ_n on the diagonal of Σ are called the singular values of the matrix A, the vectors u_n that make up the columns of U are called the left singular vectors of A, and the vectors v_n that make up the columns of V are called the right singular vectors of A. In this digit classification example, the different principal components/modes (left singular vectors, or the columns of U) give different features of the digits that are present in varying weights in each image. The singular values are the weights that each principal component carries. The right singular vectors, or columns of V, represent how each of the digit images is represented in the PCA basis. Each column is a projection onto that number PCA mode. These values differ between different figures, as can be seen in plots of the projections onto the principal components such as in Figure 3, where different digits are present in different areas of space.

Principal Component Analysis, or PCA, follows the same idea of the SVD, but has to do more with low-rank approximations. 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. We would also want to know which variables have the largest variance because these contain the most important information about the data. Therefore, we want to diagonalize the variances/covariances matrix so that all off-diagonal elements (covariances) are zero:

$$C_X = V\Lambda V^{-1} \tag{9}$$

The basis of eigenvectors contained in V are called the principal components and are uncorrelated since they are orthogonal. The diagonal entries of Λ, the eigenvalues of C_x, are the variances of the new variables. This equation is very similar in form to that of the SVD.

One way of analyzing how "good" a certain rank approximation is, is to determine how much energy from the full system is contained in each mode. Assuming the full matrix is rank r, we measure the energy contained in the rank-N approximation by:

$$energy_N = \frac{\sigma_1^2 + \sigma_2^2 + ...\sigma_N^2}{\sigma_1^2 + \sigma_2^2 + ...\sigma_r^2} = \frac{||X_N||_F^2}{||X||_F^2} \tag{10}$$

We are then able to calculate the energies for each of the rank approximations to see how the energies change with the rank approximations. In our example of digit classification, we will be using lower rank approximations to analyze the images, because using a specified number of principal components can still provide useful information for classification, without an information overload. Overall, the SVD/PCA and LDA are powerful tools that will be useful when implementing our algorithm to distinguish between digit images.

3 Algorithm Implementation and Development

Algorithms involving LDA, SVM, and decision tree classifiers were used to distinguish between images of digits 0 through 9. To start, the code loads in the training and test image data sets and their labels, then reshapes the images into a matrix where each column vector represents a different image. This matrix is then converted to double precision. This data matrix is then demeaned by subtracting the row-wise mean from both the training images matrix and the test images matrix. Next, an SVD is performed and the algorithm normalizes both the training and test data by the largest singular value. The code then creates a scatter plot representing the singular value spectrum, a bar chart representing the rank energies, and a figure displaying the first 9 principal components. The code then creates a figure with a 3D plot of the projections onto principal components 2, 3, and 5 for the 60,000 training images, with a color representing each digit.

Moving into the section of the code that performs two digit LDA, the code defines that 50 modes will be used for future image classification, as well as what the two digits are that are currently being classified. Next, the algorithm finds the locations of all the examples of these two numbers in the test and training data sets, then narrows down these image data sets and their labels to only include the digits currently being sorted. The code then projects onto the principal components, as defined by SV*, due to the fact that X=USV* gives that U*X=SV*. This projection is then limited to only the first 50 modes and the images that are of the digits currently being classified. This projection is then transposed so that it is the correct shape for future steps. These projection steps are done for both the training and test data. The code then uses the MATLAB command `classify` to use linear discriminant analysis (LDA) to predict the labels of the test data based on the projections of the training data and its correct labels. The algorithm then computes the accuracy of these labels. These `classify` steps are then repeated to predict labels for the same training data set that was used to train the model and to find the associated accuracy.

The next major part of the code is the algorithm for the decision tree classifier. A decision tree is created by feeding in the training data projections, as well as the correct training data labels, to the MATLAB command `fitctree`. This tree can then be used to predict the labels of the test data by feeding in the tree and test data projections into the MATLAB command `predict`. The accuracy of these labels can then be computed. These decision tree classifier steps can then be repeated to predict the labels for the same training data set that was used to train the tree and to find the associated accuracy.

The next major part of the code is the algorithm for SVM (support vector machines). The SVM model is created by feeding in the training data projections and the correct training data labels into the MATLAB command `fitcsvm`. This SVM model can then be fed with the test data projections into the MATLAB command `predict` to get the predicted labels of the test data. The accuracy of this SVM model can then be calculated based on the differences between the predicted and correct labels. Next, these SVM steps can be repeated to predict the labels for the same training data set that was used to train the SVM and to find the associated accuracy.

If instead the user wants to perform LDA with three digits or more, a similar pattern is followed to the two digit LDA described above, with the only difference being that the projection data for both the training and test data set is narrowed to the locations of all three (or more) digits being tested. Next, a similar set of steps follows to create the projections and use the command `classify` to find the predicted labels and find the associated accuracy for both the test and training data sets.

4 Computational Results

We first performed an SVD analysis on the set of 60,000 training images of the digits. Looking at the singular value spectrum in Figure 1, there is not a sharp drop-off in singular values, which means we have a heavy-tail distribution and there is still important information in the later modes. Looking at the energies in Figure 1, the energy values represents how much energy from the full system is contained in the n-th mode/the energy contained in the rank-n approximation. It looks like about 50 modes is necessary for good image reconstruction, as this is when the energy value reaches approximately 83% of the image's energy being captured. In other words, the rank r of the digit space can be approximated to be about 50. Going into more information about the matrices U, Σ/S, and V, the vectors/columns that make up U are the left singular vectors, so the U matrix can be thought of as as 784 pixels x 784 principal components matrix that therefore contains the image information about the principal components. Σ/S is a 784x784 matrix with the values on the diagonal being the singular values. V is a 60,000 images x 784 matrix, and the vectors/columns of V are the right singular vectors of the input matrix.

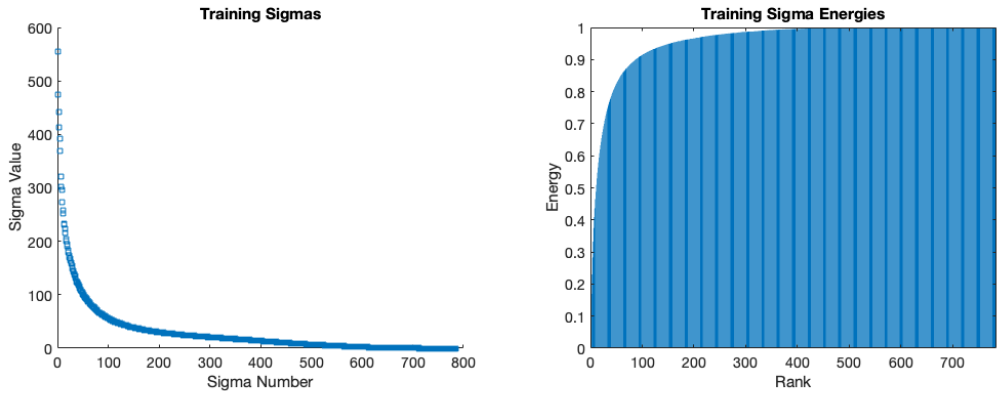

Figure 1: The figure on the left is the σ values (nonzero singular values) for each σ_n. Note the heavy-tail distribution, indicating the important information in later modes. The figure on the right is the cumulative energy accounted for by each rank. The higher the energy, the more of the energy of the system that is accounted for by that rank approximation. We are estimating the digit space to be approximately rank 50.

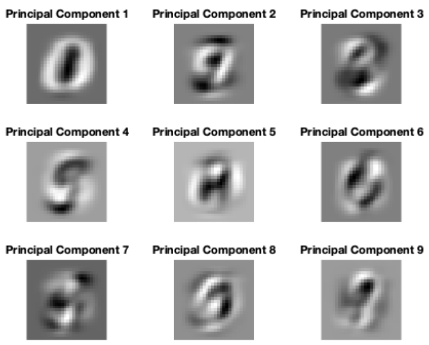

Figure 2: Image representations of the first nine principal components, as calculated from the 60,000 image training set of data.

Figure 2 shows what the first nine principal components look like. We can see certain shapes and features that are known to be more or less common in different digits. Seeing the principal components visually also supports that we will need a larger number of modes (about 50) in order to reconstruct the images well and perform the image recognition. Using a lower number of modes will cause the main features of the digit images to not be completely reconstructed, so it would be even more difficult to distinguish between the different digits.

Next, on a 3D plot we projected onto three selected V-modes (columns 2, 3, and 5) colored by their digit label, as can be seen in Figure 3. We can see that there are distinct areas of the figure where each digit is located, meaning that the digit images have different weights of the principal components present and will therefore be able to be distinguished based on the projections.

After performing this analysis, we once again projected the image data into the PCA space and built a linear classifier (LDA) to distinguish between two digits. The two digits in the data set that appear to be the most difficult to separate are 6 and 7, with an accuracy of 49.1%. The two digits in the data set that are the most easy to separate are 3 and 9, with an accuracy of 90.4%. These values are from classifying the images in the test data set, as is common practice after training the model with the training data set. If we instead classify the images in the training data set using LDA (after training the model with the same training data), we get much higher accuracies. For example, the accuracy when separating between 3 and 9 goes up to 97.4%, and the accuracy when separating between 6 and 7 goes up to 99.8%. Therefore the classifier performs better on the training data sets, which is to be expected because this data was explicitly used to create the classifier.

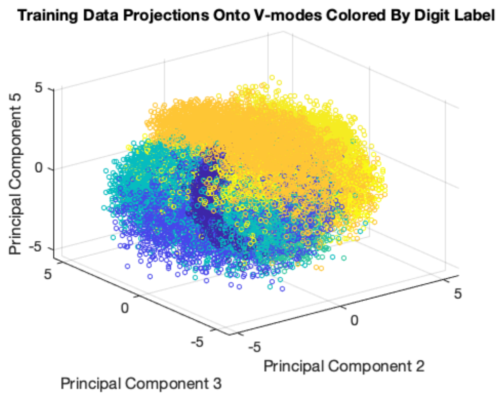

Figure 3: Projections of the training data onto the V-modes (columns) number 2, 3, and 5. Each color represents a different digit from 0 to 9. Note the colored clusters in the 3D space.

Next, we built a linear classifier to identify between three digits. This had a lower accuracy than when distinguishing between just two digits. For example, distinguishing between the digits 3, 4, and 9 gave an accuracy of 54.7%, which is lower than the previous accuracies for distinguishing between two digits (90.4% between 3 and 9 and 81.0% between 3 and 4). This is to be expected, as there are more digits to be distinguished between, but this accuracy level is still relatively low for if the algorithm were to be included in any product going to market. These values are from distinguishing the digits on the test data set after training the model with the training data set. If we instead classify the images in the training data set using LDA (after training the model with the same training data), we once again get much higher accuracies. For example, distinguishing between the digits 3, 4, and 9 now gives an accuracy of 94.1%. This is significantly higher than when distinguishing on the test data set, with an accuracy of 54.7%. This is the same pattern that was observed when distinguishing between two digits. When making predictions using the same data that was used to train the model, the results are much more accurate. This could possibly suggest that our model is overfit to the specific training data, as there is a significant decrease in accuracy when testing on the new test data.

SVM (support vector machines) and decision tree classifiers were the state-of-the-art until about 2014, but have since been replaced by neural networks. These methods do moderately well at separating between digits and are interesting methods to compare with our LDA classifier. On the hardest pair of digits to separate (6 and 7), the LDA gave an accuracy of 49.1%, the decision tree gave 54.9%, and SVM gave 37.9%. On the easiest pair of digits to separate (3 and 9), the LDA gave an accuracy of 90.4%, the decision tree gave 78.7%, and SVM gave 85.8%. It appears that there is some variation in which method gives the highest accuracy, as the LDA is sometimes highest, while the decision tree is also sometimes highest. There is also variation in which method gives the lowest accuracy between the three methods, as the decision tree is sometimes the least accuracy, while SVM is also sometimes the least accurate. When these methods are instead tested on the same training data that was used to train the models, the accuracies increase significantly. For example, when separating digits 6 and 7, the LDA now gives an accuracy of 99.8%, the decision tree gives 99.9%, and SVM gives 100%. Furthermore, when separating digits 3 and 9, the LDA now gives an accuracy of 97.4%, the decision tree gives 99.4%, and SVM gives 98.5%. This is to be expected because this same data was used to train each of these three models, but suggests that our models are possibly overfit due to the differences in accuracies when testing on the training versus test data. Overall, LDA, SVM, and decision tree classifiers all do moderately well at classifying between different digits in images.

5 Summary and Conclusions

In conclusion, Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), decision tree classifiers, and SVM (support vector machines) are useful tools, particularly for distinguishing between images digits 0 through 9, but are no longer the state-of-the-art methods for image recognition. These mathematical tools were able to help us approximate the rank of the digit space and accurately distinguish the digit being represented in different images. We were also able to demonstrate how MATLAB can be used as a way to implement these algorithms. The ideas of PCA, LDA, decision tree classifiers, and SVM can be applied to many other applications in order to distinguish between images of different objects or symbols with different visual features, unlocking the possibility for image recognition in many future applications.

Appendix A MATLAB Functions

The following MATLAB functions are implemented in this code:

- `[images, labels] = mnist_parse(images_name, images_labels_name)` loads the MNIST data set into MATLAB.
- `B = reshape(A, [sz1, sz2, sz3])` reshapes A into a sz1-by-sz2-by-sz3 array where szN indicates the size of each dimension.
- `szdim = size(A,dim)` returns the length of dimension dim in the matrix A.

- `I2 = im2double(I)` converts the image I to double precision.
- `M = mean(A, dim)` returns the mean along dimension dim of matrix A.
- `[U,S,V] = svd(X, 'econ')` produces an economy-size singular value decomposition of matrix X in order to find the matrices that evaluate to X=U*S*V'.
- `sig = diag(S)` returns a column vector of the main diagonal elements of S.
- `scatter(x,y)` creates a scatter plot with circles at the locations specified by the vectors x and y. The markers can also be modified to be different shapes and sizes.
- `length(obj)` returns the length of an object, such as the length of an array.
- `X = zeros(sz1,...,szN)` returns an sz1-by-...-by-szN matrix of zeros.
- `sum(A)` returns the sum of the elements of A along the first array dimension whose size does not equal 1.
- `bar(x,y)` creates a bar graph with one bar for each element in y, with the bars drawn at the locations specified by x.
- `B = rescale(A)` scales the color values of A to be between 0 and 1 in order to visualize the results properly.
- `imshow(I)` displays the grayscale image I in a figure.
- `colormap = brewermap(N,scheme)` returns a colormap based on digits 0 to 9 with scheme 'Set1'.
- `scatter3(X,Y,Z,S,C)` displays circles with area S and the color specified by C at the locations specified by the vectors X, Y, and Z.
- `k = find(X)` returns a vector k containing the linear indices of each element in X that evaluates to True from the expression in parentheses.
- `B = sort(A)` sorts the elements of A in ascending order.
- `class = classify(sample,training,group)` classifies each row of the data in sample into one of the groups in training using linear discriminant analysis. `group` is the labels for the training data set.
- `tree=fitctree(X,Y)` returns a fitted binary classification decision tree based on the input variables contained in matrix/training data X and output/labels Y. The returned binary tree splits branching nodes based on the values of a column of X.
- `label = predict(model,X)` returns a vector of predicted class labels for the predictor data in the matrix X, based on the trained model `model`.
- `model = fitcsvm(X,Y)` trains a support vector machine (SVM) model for two-class classification on a training data set X with labels Y.

Appendix B MATLAB Code

```matlab
% Megan Freer
% AMATH 482
% Assignment 4

clear all; close all; clc;

%% Load in and reshape data
% load in the MNIST training data
```

```
[training_images, training_labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');

% load in the MNIST test data
[test_images, test_labels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');

% reshape each image into a column vector and each column of data matrix is
% a different image
training_images_col = reshape(training_images, [size(training_images, 1)*...
                              size(training_images, 2), size(training_images, 3)]);
training_images_col = im2double(training_images_col);

test_images_col = reshape(test_images, [size(test_images, 1)*size(test_images, 2),...
                          size(test_images, 3)]);
test_images_col = im2double(test_images_col);

%% 1) De-mean
training_mean = mean(training_images_col, 2); % column vector w/ mean of each row (784x1)
training_images_col = training_images_col - training_mean;
test_images_col = test_images_col - training_mean;

%% 2) Divide by largest singular value
[U_train,S_train,V_train] = svd(training_images_col, 'econ');
[U_test,S_test,V_test] = svd(test_images_col, 'econ');
largest_singular_value = S_train(1,1);
training_images_col = training_images_col / largest_singular_value;
test_images_col = test_images_col / largest_singular_value;

%% Figure 1: Sigmas

% Finding sigma values from S matrix (with training data)
sigmas_train = diag(S_train);

% Plot sigmas (with training data)
figure()
scatter(1:length(S_train), sigmas_train, 's')
title('Training Sigmas', 'FontSize', 18)
xlabel('Sigma Number', 'FontSize', 18)
ylabel('Sigma Value', 'FontSize', 18)
set(gca, 'FontSize', 14)

%% Figure 2: Sigma Energies

% Calculate energy for each rank/sigma (with training data)
energies_train = zeros(1, length(sigmas_train));
for i=1:length(sigmas_train)
    energies_train(i) = sum(sigmas_train(1:i).^2)/sum(sigmas_train.^2);
end

% Plot energies (with training data)
figure()
bar(1:length(energies_train), energies_train)
ylim([0 1])
title('Training Sigma Energies', 'FontSize', 18)
xlabel('Rank', 'FontSize', 18)
```

```
ylabel('Energy', 'FontSize', 18)
set(gca, 'FontSize', 14)

%% Figure 3: First 9 Principal Components

figure()
for i = 1:9
    subplot(3,3,i)
    ut1 = reshape(U_train(:,i), size(training_images, 1), size(training_images, 2));
    ut2 = rescale(ut1);
    imshow(ut2)
    title('Principal Component ' + string(i))
end

%% Figure 4: Projections Onto Principal Components 2, 3, and 5

projection = S_train*V_train'; % projection onto principal components: X = USV' --> U'X = SV'
projection = projection';
projection = projection(:, [2 3 5]); % 60,000 images x 3 modes
% x = column 2, y = column 3, z = column 5

figure()
colormap = brewermap(max(training_labels+1),'Set1'); % Creating the colormap
markersize = 20;
colors = colormap(training_labels+1); % maps each label to the corresponding color
scatter3(projection(:,1), projection(:,2), projection(:,3), markersize, colors)
title('Training Data Projections Onto V-modes Colored By Digit Label', 'FontSize', 16)
xlabel('Principal Component 2', 'FontSize', 16)
ylabel('Principal Component 3', 'FontSize', 16)
zlabel('Principal Component 5', 'FontSize', 16)
set(gca, 'FontSize', 14)

%% Deciding if separating 2 or 3 digits
num_digits_separating = 2;

if num_digits_separating == 2
    %% Two Digit Recognition: narrowing down data (training data)

    % Choosing numbers using
    num_modes = 50;
    num_a = 6;
    num_b = 7;

    % narrow down to only values testing (training data)
    indexes_a = find(training_labels == num_a);
    indexes_b = find(training_labels == num_b);
    combined = [indexes_a; indexes_b];
    indexes_train = sort(combined);

    % narrow labels to only include two numbers looking at (training data)
    training_labels = training_labels(indexes_train);

    % narrow pictures to only include two numbers looking at (training data)
    training_images_col = training_images_col(:, indexes_train);
```

```
% narrow down to only values testing (test data)
indexes_a = find(test_labels == num_a);
indexes_b = find(test_labels == num_b);
combined = [indexes_a; indexes_b];
indexes_test = sort(combined);

% narrow labels to only include two numbers looking at (test data)
test_labels = test_labels(indexes_test);

% narrow labels to only include two numbers looking at (test data)
test_images_col = test_images_col(:, indexes_test);

%% 3) Project onto PCA modes (V-modes) (training data)
% do projection with training data
projection_train = S_train*V_train'; % projection onto principal components: X = USV' --> U'X = SV'

% narrow training data to first 50 modes and images that are 0 or 1
projection_train = projection_train([1:num_modes], indexes_train);
projection_train = projection_train';

% do projection with test data
projection_test = S_test*V_test';

% narrow test data to first 50 modes and images that are 0 or 1
projection_test = projection_test([1:num_modes], indexes_test);
projection_test = projection_test';

%% 4) Use classify (for LDA) (on test data)
class = classify(projection_test, projection_train, training_labels);

wrong_count = 0;
for i = 1:length(class)
    if class(i) ~= test_labels(i)
        wrong_count = wrong_count + 1;
    end
end

accuracy_lda_2_digits = (1-(wrong_count/length(class)))*100 % in percents

%% Use classify (for LDA) (on training data)
class = classify(projection_train, projection_train, training_labels);

wrong_count = 0;
for i = 1:length(class)
    if class(i) ~= training_labels(i)
        wrong_count = wrong_count + 1;
    end
end

accuracy_lda_2_digits_train = (1-(wrong_count/length(class)))*100 % in percents

%% Classification tree on fisheriris data (on test data)
load fisheriris;
```

```
    tree=fitctree(projection_train,training_labels);
    predicted_labels = predict(tree,projection_test);

    wrong_count = 0;
    for i = 1:length(predicted_labels)
        if predicted_labels(i) ~= test_labels(i)
            wrong_count = wrong_count + 1;
        end
    end

    accuracy_tree = (1-(wrong_count/length(predicted_labels)))*100 % in percents

    %% Classification tree on fisheriris data (on training data)
    load fisheriris;

    tree=fitctree(projection_train,training_labels);
    predicted_labels = predict(tree,projection_train);

    wrong_count = 0;
    for i = 1:length(predicted_labels)
        if predicted_labels(i) ~= training_labels(i)
            wrong_count = wrong_count + 1;
        end
    end

    accuracy_tree_train = (1-(wrong_count/length(predicted_labels)))*100 % in percents

    %% SVM classifier (for test data)
    svm_model = fitcsvm(projection_train,training_labels);
    testlabels_svm = predict(svm_model,projection_test);

    wrong_count = 0;
    for i = 1:length(testlabels_svm)
        if testlabels_svm(i) ~= test_labels(i)
            wrong_count = wrong_count + 1;
        end
    end

    accuracy_svm = (1-(wrong_count/length(testlabels_svm)))*100 % in percents

    %% SVM classifier (for training data)
    svm_model = fitcsvm(projection_train,training_labels);
    testlabels_svm = predict(svm_model,projection_train);

    wrong_count = 0;
    for i = 1:length(testlabels_svm)
        if testlabels_svm(i) ~= training_labels(i)
            wrong_count = wrong_count + 1;
        end
    end

    accuracy_svm_train = (1-(wrong_count/length(testlabels_svm)))*100 % in percents
elseif num_digits_separating == 3
```

```
%% Three Digit Recognition: narrowing down data (training data)

% Choosing numbers using
num_modes = 50;
num_a = 3;
num_b = 4;
num_c = 9;

% narrow down to only values testing (training data)
indexes_a = find(training_labels == num_a);
indexes_b = find(training_labels == num_b);
indexes_c = find(training_labels == num_c);
combined = [indexes_a; indexes_b; indexes_c];
indexes_train = sort(combined);

% narrow labels to only include 3 numbers looking at (training data)
training_labels = training_labels(indexes_train);

% narrow pictures to only include 3 numbers looking at (training data)
training_images_col = training_images_col(:, indexes_train);

% narrow down to only include 3 numbers looking at (test data)
indexes_a = find(test_labels == num_a);
indexes_b = find(test_labels == num_b);
indexes_c = find(test_labels == num_c);
combined = [indexes_a; indexes_b; indexes_c];
indexes_test = sort(combined);

% narrow labels to only include 3 numbers looking at (test data)
test_labels = test_labels(indexes_test);

% narrow pictures to only include 3 numbers looking at (test data)
test_images_col = test_images_col(:, indexes_test);

%% 3) Project onto PCA modes (V-modes) (training data)
% do projection with training data
projection_train = S_train*V_train'; % projection onto principal components: X = USV' --> U'X = SV'

% narrow training data to correct number modes and images that are 3 numbers
projection_train = projection_train([1:num_modes], indexes_train);
projection_train = projection_train'; % now 12,665x50

% do projection with test data
projection_test = S_test*V_test';

% narrow test data to correct number modes and images that are 3 numbers
projection_test = projection_test([1:num_modes], indexes_test);
projection_test = projection_test';

%% 4) Use classify (for LDA) (for test data)
class = classify(projection_test, projection_train, training_labels);

wrong_count = 0;
for i = 1:length(class)
```

```
        if class(i) ~= test_labels(i)
            wrong_count = wrong_count + 1;
        end
    end

    accuracy_lda_3_digits = (1-(wrong_count/length(class)))*100 % in percents

    %% Use classify (for LDA) (for training data)
    class = classify(projection_train, projection_train, training_labels);

    wrong_count = 0;
    for i = 1:length(class)
        if class(i) ~= training_labels(i)
            wrong_count = wrong_count + 1;
        end
    end

    accuracy_lda_3_digits_train = (1-(wrong_count/length(class)))*100 % in percents
end
```