AMATH 482 Homework 4

Haley Riggs

March 9, 2021

Abstract

Machine learning is a popular and largely beneficial topic in data science. In this report, three different machine learning techniques, linear discriminant analysis (LDA), support-vector machines (SVM), and decision tree classification, were utilized to train a program to differentiate between digit images from the MNIST data set.

1 Introduction and Overview

In this assignment, we used the singular value decomposition (SVD), principal component analysis (PCA), along with linear discriminant analysis (LDA), support vector machines (SVM), and classification trees to classify digits. In order to do this, the MNIST data set, a set of images of numerical digits, were utilized. The overall goal was to train the machine, using a set of training images and labels, to be able to recognize which digits correspond to which labels of a testing set. The efficacy of this, for each method, was calculated and compared.

2 Theoretical Background

This project relied upon the use of the SVD and PCA. As discussed in Assignment 3, any matrix can be decomposed into the project of three characteristic matrices, that is

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

where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{U}$ is a real, unitary (i.e. an orthogonal matrix whose transpose is equal to its inverse) matrix of dimensions $m \times m$, $\mathbf{V}$ is an $n \times n$ real, unitary matrix, and $\mathbf{\Sigma}$ is a diagonal matrix of size $m \times n$ whose diagonal entries are the singular values of matrix $\mathbf{A}$. The representation of $\mathbf{A}$ depicted by the SVD can be thought of as a combination of rotation, scaling, and another rotation.

It is possible to write $\mathbf{A}$ as a sum of rank-one matrices. If $\mathbf{A}$ is rank r and only the first N, where $1 \geq N < r$, singular values of $\mathbf{A}$ are calculated in the sum, we have created the best approximation of $\mathbf{A}$ of all matrices of rank N or less. In mathematical form, this sum is equal to the following:

$$\mathbf{A_N} = \sum_{j=1}^{N} \sigma_j \mathbf{u_j} \mathbf{v_j^*} \tag{2}$$

In Equation 2, $\mathbf{u_j}$ represent the columns of $\mathbf{U}$, $\mathbf{v_j}$ are the columns of $\mathbf{V}$, σ_j are the singular values of $\mathbf{A}$ (the diagonal entries of $\mathbf{\Sigma}$), and $\mathbf{A_N}$ is the resulting low-rank approximation of $\mathbf{A}$.

We can calculate the corresponding energy of $\mathbf{A}$ in the rank-N approximation by using the following formula:

$$\text{energy}_N = \frac{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_N^2}{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2} \tag{3}$$

Principal component analysis (PCA) is a method using the SVD to understand trends in the data. After subtracting off the mean, the SVD of the data matrix can be calculated. If the data was stored as columns in the original data matrix, the columns of $\mathbf{U}$ are the principal components. They depict the trends in the data. In order to transform this data matrix into the basis of principal components, we must multiply it by $\mathbf{U^T}$ on the left.

The SVD and PCA are useful tools when it comes to machine learning. We can give the computer training data, comprised of different classes we wish to differentiate between (such as dogs and cats), transform this data into a useful format, use PCA to identify the principal components, find a threshold through the use of linear discriminant analysis that distinguishes between the classes, and finally use this algorithm to see how well it performs on test data.

The goal of Linear Discriminant Analysis (LDA) is to project the data in such a way as to maximize the distance between inter-class data but minimize the distance between intra-class data. First, we will consider the implementation of LDA for two datasets. In order to achieve our goal, we must first calculate the means for each group, for each feature. Using the calculated means, column vectors called μ_1, μ_2, we can create a between-class scatter matrix, measuring the variance between the groups, with the following formula:

$$\mathbf{S_B} = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{4}$$

The within-class scatter matrix, measuring the variance within each group, can be defined with the following equation:

$$\mathbf{S_w} = \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T \tag{5}$$

Next, we wish to find a vector $\mathbf{w}$ such that the following is true:

$$\mathbf{w} = \text{argmax} \frac{\mathbf{w^T S_B w}}{\mathbf{w^T S_w w}} \tag{6}$$

It has been previously discovered that this vector $\mathbf{w}$ is the eigenvector that corresponds to the largest eigenvalue of this equation:

$$\mathbf{S_B w} = \lambda \mathbf{S_w w} \tag{7}$$

After obtaining this, it is relatively easy to create a threshold differentiating between the classes. We can choose this threshold value to be the midpoint of the data projected onto $\mathbf{w}$.

Now, considering LDA for more than two groups, we can define the between-class scatter matrix in the following way, defining μ to be the overall mean and μ_j to be the mean corresponding to each of $N \geq 3$ classes.

$$\mathbf{S_B} = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T \tag{8}$$

$\mathbf{w}$ stays the same as before and the within-class scatter matrix becomes

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

After using LDA to train the machine, we can determine the accuracy of our program using test data. After demeaning the test data and putting it into the same format that our training data was in, we can project the data onto the PCA space by multiplying it by $\mathbf{U^T}$ on the left. Then, we can multiply this result by the transpose of the $\mathbf{w}$ vector previously found through LDA. We can then use the discovered threshold to determine which class the entries in the resulting vector belong to, according to our algorithm. We can then calculate the resulting error and formulate a success rate.

Other machine learning methods can also be used. One such method is known as SVM (support-vector machine). The SVM algorithm aims to find an N-dimensional hyperplane (where N is the number of features) that has the maximum margin or, in other words, maximizes the distance between data points of the classes [2]. Another method is the decision tree classifier, which splits data in half in an iterative fashion according to some parameter [1]. Starting at the root of the decision tree, the data is divided based upon the feature that gives the largest information gain [1].

3 Algorithm Implementation and Development

In this assignment, the computer was presented with training data, comprised of images of digits and associated labels from the MNIST data set. Programs, created through the utilization of different machine learning algorithms (LDA, SVM, and decision tree classifiers), were then created to differentiate between these digits.

First, information from the MNIST data set was loaded into MATLAB using a function called `mnist_parse` [3]. This function outputted the images and labels for the dataset. The dimensions of the images tensor was the number of rows by the number of columns by the number of images. Then, the images tensors for the training and testing data was formatted. First, the data was transformed from the uint-8 data type into doubles. Then, each each slice of the training images tensor (representing one image each) was extracted, reshaped into a vector, and stored as a column in a training data matrix. This was repeated on the test images tensor to create the test data matrix. Next, the mean of each row of the training data matrix was calculated and subtracted from all elements in the row. The training data mean was also subtracted from each row of the test data matrix. Afterwards, the SVD of the training data matrix was computed.

The next step was to calculate the sigma energies of the resulting SVD to approximate the rank of the data. After creating a plot, the rank was determined to be approximately 50 through visual analysis. Following this, the first six principal components, which were the columns of $\mathbf{U}$ were plotted by reshaping each into a square, rescaling using the `rescale` function, and using `imshow` to display the results.

At this point, a color-coded plot of the PCA projections was created. In order to do this, the first three columns of $\mathbf{U}^{\mathbf{T}}$, representing the three most important principal components of the data, was multiplied by the data matrix on the left. This projected the data matrix into the PCA space. Then, the `find` function was utilized to create a vectors of indices corresponding to each digit. The three coordinates of the data projection matrix were plotted, color-coded according to the digits they represented. This plot allowed for a visual deduction of what the easiest and most difficult digits to separate appeared to be.

Next, an LDA was constructed to differentiate between 0s and 1s, which by visual analysis of the aforementioned projection plot appeared to be easiest to separate. The number of ones and zeros was calculated. Then, ones and zeros matrices were created from the first 50 rows, representing the low-rank approximation of the data, of the full projection matrix (resulting from multiplying the training data matrix by $\mathbf{U}^{\mathbf{T}}$ on the left), as well as the columns in this matrix corresponding to ones and zeros, separately.

After this was completed, the scatter matrices were created. First, the mean of the rows for each matrix was stored in a column vector. The within-class scatter matrix was calculated by summing the product of the differences between each column of the ones matrix and the vector of means and the transpose of this. This was repeated to create the zeros within-class matrix. The between-class scatter matrix was calculated by taking the product of the difference of the vector of means for the ones and the vector of the means for the zeros and the transpose of this.

Then, $\mathbf{w}$ was calculated by solving the corresponding eigenvalue equation, finding the eigenvector that goes with the largest eigenvalue, and normalizing the result. Then, the ones and zeros matrices were projected onto $\mathbf{w}$ by multiplication. Ones were arbitrarily chosen to be below the threshold, which was then enforced with an if statement.

Afterwards, the threshold value was determined. After sorting the values in the projection vectors for ones and zeros in ascending order, a while loop was created to cycle through the ones and zeros elements, starting at the smallest zeros value and the largest ones value and working inwards. The midpoint of the center values was calculated; this is the threshold. Finally, a histogram of the results, with the threshold line superimposed on top, was plotted.

Next, a three-way LDA classification, between 0s, 1s, and 3s was created. The data was essentially grouped into two categories for the first step of this process, threes and not-threes (ones and zeros). The number of threes and the number of not-threes was calculated for this subset of the data. Then, threes and not-threes matrices were created by taking a low-rank approximation of the full projection matrix and locating the indices of threes and not-threes, separately. Scatter matrices were created in a similar fashion

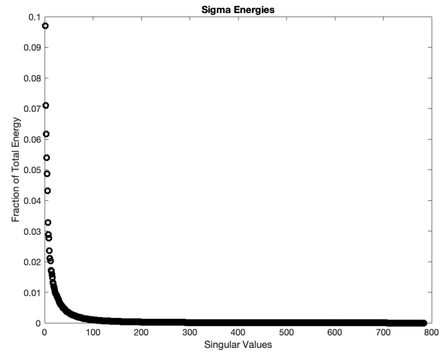

Figure 1: Energy associated with each singular value

as before, the new **w** vector was calculated, the data was projected onto this **w**, threes were set to be above the threshold, and the threshold was calculated.

Afterwards, the new indices for zeros and ones, using the data in the ones and zeros projection that was below the threshold for threes, was calculated. The corresponding matrices, comprised of the low-rank approximations of the projection matrix were created and then projected onto the **w** from the LDA for the ones and zeros classification. The result was sorted and then plotted in the form of a histogram along with the threshold from the ones and zeros classification.

The two-digit classification process for the LDA was repeated for fours and nines, which visually seemed to be the most difficult digits to separate based upon the previous color-coded projection plot.

Then, it was time to quantify the accuracy of the LDA on test data. This was first done for fours and nines and then later for zeros and ones. The previously formatted test data matrix was multiplied by the $\mathbf{U^T}$ from the training data to project it onto the PCA space. Then, the indices for the fours and nines were found and concatenated into a single vector. The **w** vector from the fours and nines training LDA was used to project a low-rank approximation with entries corresponding to fours and nines onto the corresponding line. A results vector, comprised of ones representing nines and zeros representing fours, that compared the one-dimensional projection to the threshold found previously was created. In order to find the actual labels for the test data, another vector was created, with entries corresponding to fours set to zero and corresponding to nines set to one. An error vector, comprised of the absolute value of the difference between the results vector and vector containing the true labels, was calculated and its entries were summed together. Subtracting the resulting error number divided by the total number of test images from one gave the success rate. This was repeated for the ones and zeros.

The next goal was to create a classification tree for all ten digits. A low-rank approximation each of the training data and test data matrices, projected into the PCA space by multiplying on the left by $\mathbf{U^T}$ was calculated. The transpose of the training data approximation and corresponding training data labels were fed into the `fitctree` function. Then, the resulting tree, as well as the transpose of the test data, were passed into the function `predict`, which outputted the guessed test labels from the classification tree.

The accuracy was calculated using a similar method as before. First, an error vector was created by taking the absolute value of the difference between the predicted labels and actual labels. Then, the error vector was put into binary terms. Since an accurate labeling resulted in an entry of zero in the error vector, and therefore an incorrect labeling would have an entry that was not zero, the entries with nonzero values were all set to be one. Essentially, correct labelings had value zero and incorrect labelings had value one. As previously, these entries were all summed and then divided by the total number of test images. This was subtracted from one to get the success rate.

For the SVM classifier for all ten digits, the transpose of the projected training data approximation and corresponding training data labels were inputted into the `fitcecoc` function. Once again, `predict` was used, with the output of the `fitcecoc` function and the transpose of the test data as arguments. The accuracy was computed in the exact same way as for the classification tree.

A classification tree was also created for the fours and nines. The indices for the fours and nines were

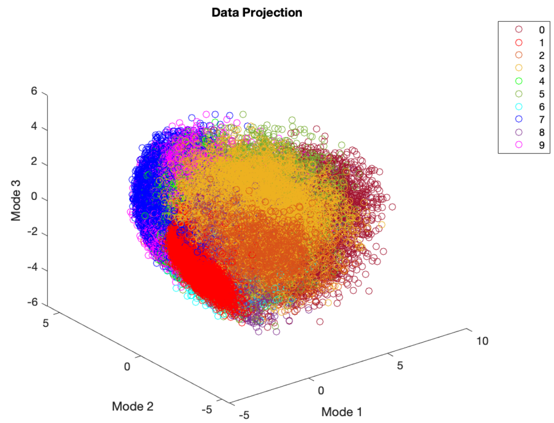

Figure 2: Color-coded projection of the training digit data onto a low-dimensional approximation of the PCA space

computed and a fours and nines training data matrix was created using a low-rank approximation of the projected data matrix and these indices. Additionally, a vector of labels, labeled with a four for the fours and a nine for each nine was created. A similar process was executed for the testing data. From this, a classification tree was created, with the transpose of the projected fours and nines training data and the corresponding labels as arguments. The result of this, along with the transpose of the fours and nines test data were passed into the `predict` function to create the label predictions. The accuracy was computed exactly as before. An SVM classifier, this time using the `fitcsvm` was created. The inputs to this function, and the steps afterwards, are the same as what was discussed previously. A classification tree and an SVM classifier for ones and zeros were created as well.

4 Computational Results

As aforementioned, the goal of this project was to train the computer to be able to distinguish between different digits. In order to do this, three different machine learning algorithms, namely LDA, SVM, and decision tree classifiers were implemented. First, though, it was important to perform an SVD analysis of the data and examine the singular value spectrum to determine the rank. The following plot, Figure 1 shows a visual depiction of the energies associated with the singular value spectrum.

From this plot, it can be observed that, out of the 784 singular values, only the first fifty or so really affect the data. The energy levels associated with the rest are approximately equal to zero. Therefore, the rank of the data is about fifty, which is the number of rows we will take to create a low-rank approximation of the data.

After taking the SVD of the data matrix, we can observe what the interpretation of the $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ matrices are. Since the image data has been stored as columns in the data matrix, the columns of $\mathbf{U}$ will be the principal components of the data set. In other words, these columns can be thought of as "eigendigits." They can be reshaped into square images, shown in Figure 4 in Appendix C. The first column of $\mathbf{U}$ represents the number one most important feature of the images, the dominant trait that they all (or at least most) have in common. In Figure 4, this is shown to be a zero-like shape. The next eigendigit, displaying another major pattern in the data, appears to be a combination of a nine, and eight, and a one. The next digits also show patterns in the data, though as we progress through the eigendigits, each displays a less "important" trend. The $\mathbf{\Sigma}$ matrix contains a relative importance for each eigendigit in the data and the $\mathbf{V^T}$ matrix has the linear coefficients needed to fully reconstruct the training data.

After this, the data was projected onto the first three modes, stored as columns of $\mathbf{U}$ and color-coded (in rainbow order) by digit in order to observe the clumps of digit data. The resulting plot is given in Figure 2. In this plot, we can observe that data corresponding to digits one and zero appear to be clumped together

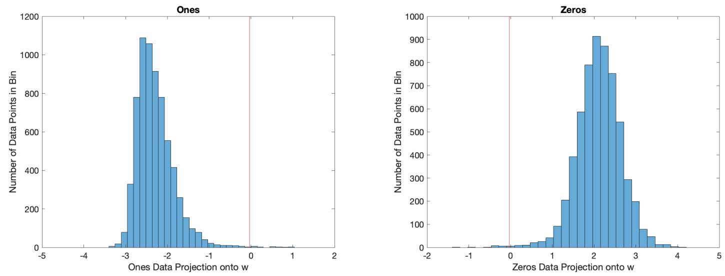

Figure 3: Results of LDA on ones and zeros data

well and relatively far apart, thus meaning that these digits should be fairly easy to separate. The threes data also appears to be fairly easy to separate from the ones and zeros. However, the fours and nines data looks scattered and mixed, therefore implying that fours and nines should be challenging to separate. This analysis was used later when LDA, SVM, and decision tree classifiers were implemented.

As previously mentioned, the two digits that appeared to be the easiest to separate were the ones and zeros. So, this is where the machine learning algorithm implementation began. First, an LDA classifier was built to separate the two digits. The result of this, where the plotted red line represents the threshold, is depicted in Figure 3. For the ones plot, all of the values to the left of the threshold line represent ones that were classified correctly. The values to the right of the line are values that ended up being classified as zeros instead. For the zeros plot, all of the values to the right of the threshold were classified correctly and those to the left were classified incorrectly, as ones. It can be observed from this that LDA created a fairly accurate way to distinguish between the ones and zeros in the training data.

Next, the test data needed to be analyzed for accuracy. The LDA algorithm for ones and zeros was found to be 99.96% accurate, using the method discussed in Section 3. On ones and zeros testing data, SVM was found to be 90.20% accurate and the classification tree was 98.80% accurate. Based upon this, it appeared that data that was considered easy to separate was distinguished excellently using LDA and still very well using the classification tree. SVM performed the worst but still ended up with a highly accurate result.

In addition to the two-way separation, a three-digit linear classifier was implemented to distinguish between zeros, ones and threes. The resulting plot which, at least according to the training data, appeared to be fairly accurate is located in Appendix C, called Figure 5.

The digits that seemed most difficult to separate were fours and nines. The LDA algorithm was implemented for these digits as well. The result is shown in Appendix C, Figure 6. After training the model, the test data was given to it. For LDA, this resulted in a success rate of 98.95%. SVM had a success rate of 99.38%, and the decision tree classifier has a rate of 96.25%. Surprisingly, SVM performed significantly better on the data that was apparently difficult to separate than it did for the data considered easy to separate. For this data, SVM outperformed both of the other methods, creating an almost perfect result.

SVM and decision tree classifiers were both implemented to distinguish between all ten digits as well. This resulted in a success rate of 94.12% for SVM and a success rate of 84.97% for the decision tree. Both results were impressive but the SVM performed significantly better than the decision tree classifier in this case.

5 Summary and Conclusions

In this project, after computing the SVD and performing PCA on the MNIST dataset, different machine learning algorithms (LDA, SVM, and decision tree classifiers) were utilized to train the computer to differentiate between digits. Although further analysis would be required to confirm this, SVM seemed to outperform the other methods on data that, through visual analysis, appeared to be difficult to separate between, such as fours and nines. In this case, LDA performed the worst. However, on data that was seemingly easy to distinguish between, SVM performed the worst while LDA performed the best. In general, the accuracy of the decision tree classifier landed somewhere in the middle. Depending upon the dataset, I would venture to suppose that all three methods can be useful.

References

[1] Afroz Chakure. *Decision Tree Classification: An Introduction to Decision Tree Classifier*. July 2019. URL: `https://medium.com/swlh/decision-tree-classification-de64fc4d5aac`.

[2] Rohith Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms*. June 2018. URL: `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`.

[3] rayryeng. Sept. 2016. URL: `https://stackoverflow.com/questions/39580926/how-do-i-load-in-the-mnist-digits-and-label-data-in-matlab`.

Appendix A MATLAB Functions

- `Mdl = fitcecoc(X,Y)` accepts a matrix `X` of training data organized as rows and a vector `Y` containing associated labels, outputting a an ECOC model, the more-than-two-class equivalent to `fitcsvm`.
- `Mdl = fitcsvm(X,Y)` accepts a matrix `X` of training data organized as rows and a vector `Y` containing associated labels, outputting a an SVM classifier.
- `tree = fitctree(X,Y)` accepts a matrix `X` of training data organized as rows and a vector `Y` containing associated labels, outputting a binary classification decision tree.
- `M = mean(A,dim)` outputs the mean of the tensor `A` along the specified dimension. For example, if `A` is a two-dimensional matrix, `mean(A,2)` returns a column vector of the means of each row of `A`.
- `label = predict(Mdl,Z)` accepts a classification tree/SVM classfier/ECOC model, `Mdl`, and an array of test data organized as rows, `Z`, outputting predicted labels for the test data.
- `B = rescale(A)` returns an array of the same dimensions as `A`, with all entries scaled to the interval $[0, 1]$.
- `B = sort(A)` returns the elements of `A`, sorted in ascending order.

Appendix B MATLAB Code

```
clear; close all; clc

%% Read in data
[trnImages, trnLabels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
[testImages, testLabels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');

%% Create data matrices and compute SVD of training data
% Convert from uint8 to double
trnImages = im2double(trnImages);
testImages = im2double(testImages);

% Create training data matrix
dim = size(trnImages,1)*size(trnImages,2);
numImages = size(trnImages,3);
dataMat = zeros(dim,numImages);
for j = 1:numImages
    img = trnImages(:,:,j);
    imgCol = reshape(img,[dim,1]);
```

```
    dataMat(:,j) = imgCol;
end

% Create test data matrix
numTestImages = size(testImages,3);
testDataMat = zeros(dim,numTestImages);
for j = 1:numTestImages
    img = testImages(:,:,j);
    imgCol = reshape(img,[dim,1]);
    testDataMat(:,j) = imgCol;
end

% Demean data
for k = 1:dim
   feature = dataMat(k,:);
   dataMat(k,:) = feature - mean(feature);
   testDataMat(k,:) = testDataMat(k,:) - mean(feature);
end

% Compute SVD
[U,S,V] = svd(dataMat, 'econ');

%% Calculate and plot sigma energies
% Calculate sigma energies
sig = diag(S);
energies = sig.^2/sum(sig.^2);

% Create plot
plot(energies,'ko','Linewidth',2)
title('Sigma Energies')
xlabel('Singular Values')
ylabel('Fraction of Total Energy')

%% Plot first six principal components
figure
for k = 1:6
   subplot(1,6,k)
   ut1 = reshape(U(:,k),28,28);
   ut2 = rescale(ut1);
   imshow(ut2)
   title(['Eigendigit ',num2str(k)])
end
sgtitle('Principal Components of Training Data')

%% Create color-coded projection plot of data
% Calculate data projections
UT = U';
proj = UT(1:3,:)*dataMat;

% Find which data points correspond to which digits
zerosIdx = find(trnLabels == 0);
onesIdx = find(trnLabels == 1);
twosIdx = find(trnLabels == 2);
threesIdx = find(trnLabels == 3);
```

```matlab
foursIdx = find(trnLabels == 4);
fivesIdx = find(trnLabels == 5);
sixesIdx = find(trnLabels == 6);
sevensIdx = find(trnLabels == 7);
eightsIdx = find(trnLabels == 8);
ninesIdx = find(trnLabels == 9);

% Plot color-coded projection, rainbow order
figure
plot3(proj(1,zerosIdx),proj(2,zerosIdx),proj(3,zerosIdx),'Marker','o','LineStyle','none','Color','#A2142F');
hold on
plot3(proj(1,onesIdx),proj(2,onesIdx),proj(3,onesIdx),'ro');
plot3(proj(1,twosIdx),proj(2,twosIdx),proj(3,twosIdx),'Marker','o','LineStyle','none','Color','#D95319');
plot3(proj(1,threesIdx),proj(2,threesIdx),proj(3,threesIdx),'Marker','o','LineStyle','none','Color','#EDB120');
plot3(proj(1,foursIdx),proj(2,foursIdx),proj(3,foursIdx),'go');
plot3(proj(1,fivesIdx),proj(2,fivesIdx),proj(3,fivesIdx),'Marker','o','LineStyle','none','Color','#77AC30');
plot3(proj(1,sixesIdx),proj(2,sixesIdx),proj(3,sixesIdx),'co');
plot3(proj(1,sevensIdx),proj(2,sevensIdx),proj(3,sevensIdx),'bo');
plot3(proj(1,eightsIdx),proj(2,eightsIdx),proj(3,eightsIdx),'Marker','o','LineStyle','none','Color','#7E2F8E');
plot3(proj(1,ninesIdx),proj(2,ninesIdx),proj(3,ninesIdx),'mo');
title('Data Projection')
xlabel('Mode 1')
ylabel('Mode 2')
zlabel('Mode 3')
legend('0','1','2','3','4','5','6','7','8','9')

%% Calculate variables to be used later
projFull = UT*dataMat; % Calculate full projection
rank = 50; % Use rank observed from sigma energies

%% Build LDA to classify 0s and 1s
numOnes = size(onesIdx,1); % Find number of ones
numZeros = size(zerosIdx,1); % Find number of sevens
ones = projFull(1:rank,onesIdx);
zeros = projFull(1:rank,zerosIdx);

% Calculate scatter matrices
meanOnes = mean(ones,2);
meanZeros = mean(zeros,2);

wnClassVar = 0; % within class variances
for k = 1:numOnes
    wnClassVar = wnClassVar + (ones(:,k) - meanOnes)*(ones(:,k) - meanOnes)';
end

for k = 1:numZeros
   wnClassVar =  wnClassVar + (zeros(:,k) - meanZeros)*(zeros(:,k) - meanZeros)';
end
```

```
bwClassVar = (meanOnes-meanZeros)*(meanOnes-meanZeros)'; % between class
    variance

% Find best projection line
[V2, D] = eig(bwClassVar,wnClassVar); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w1 = V2(:,ind);
w1 = w1/norm(w1,2);

% Project onto w
vOnes = w1'*ones;
vZeros = w1'*zeros;

% Make ones below the threshold
if mean(vOnes) > mean(vZeros)
    w1 = -w1;
    vOnes = -vOnes;
    vZeros = -vZeros;
end

% Find the threshold value
sortOnes = sort(vOnes);
sortZeros = sort(vZeros);

t1 = length(sortOnes);
t2 = 1;
while sortOnes(t1) > sortZeros(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
thresholdOnesZeros = (sortOnes(t1) + sortZeros(t2))/2;

% Plot histogram of results
figure
subplot(1,2,1)
histogram(sortOnes,30); hold on, plot([thresholdOnesZeros thresholdOnesZeros
    ],[0 1200],'r')
set(gca,'Xlim',[-5 2],'Fontsize',14)
title('Ones')
xlabel('Ones Data Projection onto w')
ylabel('Number of Data Points in Bin')
subplot(1,2,2)
histogram(sortZeros,30); hold on, plot([thresholdOnesZeros thresholdOnesZeros
    ],[0 1000],'r')
set(gca,'Xlim',[-2 5],'Fontsize',14)
title('Zeros')
xlabel('Zeros Data Projection onto w')
ylabel('Number of Data Points in Bin')

%% Build LDA to classify (0s and 1s) and 3s
onesZerosIdx = find(trnLabels <= 1);
numOnesZeros = size(onesZerosIdx,1); % Find number of (ones and zeros)
numThrees = size(threesIdx,1); % Find number of threes
```

```
onesZeros = projFull(1:rank,onesZerosIdx);
threes = projFull(1:rank,threesIdx);

% Calculate scatter matrices
meanOnesZeros = mean(onesZeros,2);
meanThrees = mean(threes,2);

wnClassVar = 0; % within class variances
for k = 1:numOnesZeros
    wnClassVar = wnClassVar + (onesZeros(:,k) - meanOnesZeros)*(onesZeros(:,k)
        - meanOnesZeros)';
end

for k = 1:numThrees
   wnClassVar =  wnClassVar + (threes(:,k) - meanThrees)*(threes(:,k) -
      meanThrees)';
end

bwClassVar = (meanOnesZeros-meanThrees)*(meanOnesZeros-meanThrees)'; % between
    class variance

% Find best projection line
[V2, D] = eig(bwClassVar,wnClassVar); % linear discriminant analysis
[lambda, ind] = max(abs(diag(D)));
w2 = V2(:,ind);
w2 = w2/norm(w2,2);

% Project onto w
vOnesZeros = w2'*onesZeros;
vThrees = w2'*threes;

% Make threes above the threshold
if mean(vOnesZeros) > mean(vThrees)
    w2 = -w2;
    vOnesZeros = -vOnesZeros;
    vThrees = -vThrees;
end

% Find the threshold value
sortOnesZeros = sort(vOnesZeros);
sortThrees = sort(vThrees);

t1 = length(sortOnesZeros);
t2 = 1;
while sortOnesZeros(t1) > sortThrees(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
thresholdThrees = (sortOnesZeros(t1) + sortThrees(t2))/2;

% Plot histogram of results
figure
subplot(2,2,1)
histogram(sortOnesZeros,30); hold on, plot([thresholdThrees thresholdThrees
```

```
    ],[0 3300],'r')
set(gca,'Xlim',[-4 4],'Fontsize',14)
title('Ones and Zeros')
xlabel('Ones and Zeros Data Projection onto w')
ylabel('Number of Data Points in Bin')
ylim([0 3300])
subplot(2,2,2)
histogram(sortThrees,30); hold on, plot([thresholdThrees thresholdThrees],[0
    800],'r')
set(gca,'Xlim',[-3 5],'Fontsize',14)
title('Threes')
xlabel('Threes Data Projection onto w')
ylabel('Number of Data Points in Bin')

zerosNewIdx = [];
onesNewIdx = [];
for j = 1:numOnesZeros
    %if below threshold -> ones and zeros
    if vOnesZeros(j) < thresholdThrees
        % relate to index for ones/zeros to determine if one or zero
        foundInZerosIdx = find(zerosIdx==onesZerosIdx(j));
        % if empty, it's a one; otherwise, it's a zero
        % add index of zero or one to array storing indices for each
        if isempty(foundInZerosIdx) == 1
            foundInOnesIdx = find(onesIdx==onesZerosIdx(j));
            onesNewIdx(end+1) = onesIdx(foundInOnesIdx);
        else
            zerosNewIdx(end+1) = zerosIdx(foundInZerosIdx);
        end
    end
end

% find ones and zeros using indices
onesNew = projFull(1:rank,onesNewIdx);
zerosNew = projFull(1:rank,zerosNewIdx);

% project onto w from ones and zeros classifier
vOnesNew = w1'*onesNew;
vZerosNew = w1'*zerosNew;

% sort result
sortOnesNew = sort(vOnesNew);
sortZerosNew = sort(vZerosNew);

% Plot histogram of results
subplot(2,2,3)
histogram(sortOnesNew,30); hold on, plot([thresholdOnesZeros
    thresholdOnesZeros],[0 1200],'r')
set(gca,'Xlim',[-4 3],'Fontsize',14)
ylim([0 1200])
title('Ones')
xlabel('Ones Data Projection onto w')
ylabel('Number of Data Points in Bin')
subplot(2,2,4)
```

```
histogram(sortZerosNew,30); hold on, plot([thresholdOnesZeros thresholdOnesZeros],[0 1000],'r')
set(gca,'Xlim',[-2 5],'Fontsize',14)
title('Zeros')
xlabel('Zeros Data Projection onto w')
ylabel('Number of Data Points in Bin')

%% Build LDA to classify 4s and 9s, potentially most difficult to separate
numFours = size(foursIdx,1); % Find number of ones
numNines = size(ninesIdx,1); % Find number of sevens
fours = projFull(1:rank,foursIdx);
nines = projFull(1:rank,ninesIdx);

% Calculate scatter matrices
meanFours = mean(fours,2);
meanNines = mean(nines,2);

wnClassVar = 0; % within class variances
for k = 1:numFours
    wnClassVar = wnClassVar + (fours(:,k) - meanFours)*(fours(:,k) - meanFours)';
end

for k = 1:numNines
   wnClassVar =  wnClassVar + (nines(:,k) - meanNines)*(nines(:,k) - meanNines)';
end

bwClassVar = (meanFours-meanNines)*(meanFours-meanNines)'; % between class variance

% Find best projection line
[V2, D] = eig(bwClassVar,wnClassVar); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w3 = V2(:,ind);
w3 = w3/norm(w3,2);

% Project onto w
vFours = w3'*fours;
vNines = w3'*nines;

% Make fours below the threshold
if mean(vFours) > mean(vNines)
    w3 = -w3;
    vFours = -vFours;
    vNines = -vNines;
end

% Find the threshold value
sortFours = sort(vFours);
sortNines = sort(vNines);

t1 = length(sortFours);
t2 = 1;
```

```
while sortFours(t1) > sortNines(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
thresholdFoursNines = (sortFours(t1) + sortNines(t2))/2;

% Plot histogram of results
figure
subplot(1,2,1)
histogram(sortFours,30); hold on, plot([thresholdFoursNines
    thresholdFoursNines],[0 800],'r')
set(gca,'Xlim',[-4 3],'Fontsize',14)
title('Fours')
xlabel('Fours Data Projection onto w')
ylabel('Number of Data Points in Bin')
subplot(1,2,2)
histogram(sortNines,30); hold on, plot([thresholdFoursNines
    thresholdFoursNines],[0 1000],'r')
set(gca,'Xlim',[-3 4],'Fontsize',14)
title('Nines')
xlabel('Nines Data Projection onto w')
ylabel('Number of Data Points in Bin')

%% Quantify accuracy of LDA on test data for 4s and 9s
% Classify test data
testMat = UT*testDataMat; % PCA projection
foursTestIdx = find(testLabels == 4);
ninesTestIdx = find(testLabels == 9);
foursNinesTestIdx = cat(1,foursTestIdx,ninesTestIdx);
pval = w3'*testMat(1:rank,foursNinesTestIdx);

% Check pval against threshold
% nine = 1, four = 0
resVec = (pval > thresholdFoursNines);

% Checking performance
foursNinesLabels = 0*foursNinesTestIdx';
foursNinesLabels(length(foursTestIdx)+1:end) = 1;

% 0s are correct and 1s are incorrect
err = abs(resVec - foursNinesLabels);
errNum = sum(err);
foursNinesSucRate = 1 - errNum/numTestImages;

%% Quantify accuracy of LDA on test data for 0s and 1s
% Classify test data
zerosTestIdx = find(testLabels == 0);
onesTestIdx = find(testLabels == 1);
onesZerosTestIdx = cat(1,onesTestIdx,zerosTestIdx);
pval = w1'*testMat(1:rank,onesZerosTestIdx); % w1, testMat from before

% Check pval against threshold
% zero = 1, one = 0
resVec = (pval > thresholdOnesZeros);
```

```
% Checking performance
onesZerosLabels = 0*onesZerosTestIdx';
onesZerosLabels(length(onesTestIdx)+1:end) = 1;

% 0s are correct and 1s are incorrect
err = abs(resVec - onesZerosLabels);
errNum = sum(err);
onesZerosSucRate = 1 - errNum/numTestImages;

%% Create classification tree for all 10 digits
allTrainData = projFull(1:rank,:);
allTestData = testMat(1:rank,:);
tree = fitctree(allTrainData',trnLabels);
predTestLabels = predict(tree,allTestData');

% Calculate accuracy
err = abs(predTestLabels - testLabels);
falseIdx = find(err ~= 0);
err(falseIdx) = 1;
errNum = sum(err);
allTreeSucRate = 1 - errNum/numTestImages;

%% Create SVM classifier for all 10 digits
svm = fitcecoc(allTrainData',trnLabels);
predTestLabels = predict(svm,allTestData');

% Calculate accuracy
err = abs(predTestLabels - testLabels);
falseIdx = find(err ~= 0);
err(falseIdx) = 1;
errNum = sum(err);
allSVMSucRate = 1 - errNum/numTestImages;

%% Create classification tree for 4's and 9's
foursNinesIdx = cat(1,foursIdx,ninesIdx);
foursNinesTrain = projFull(1:rank,foursNinesIdx);
foursNinesTrainLabs = 0*foursNinesIdx';
foursNinesTrainLabs(1:length(foursIdx)) = 4;
foursNinesTrainLabs(length(foursIdx)+1:end) = 9;

foursNinesTest = testMat(1:rank,foursNinesTestIdx);
foursNinesTestLabs = 0*foursNinesTestIdx';
foursNinesTestLabs(1:length(foursTestIdx)) = 4;
foursNinesTestLabs(length(foursTestIdx)+1:end) = 9;

% Create tree
foursNinesTree = fitctree(foursNinesTrain',foursNinesTrainLabs);
predFoursNinesTestLabs = predict(tree,foursNinesTest');

% Calculate accuracy
err = abs(predFoursNinesTestLabs' - foursNinesTestLabs);
falseIdx = find(err ~= 0);
err(falseIdx) = 1;
```

```
errNum = sum(err);
foursNinesTreeSucRate = 1 - errNum/numTestImages;

%% Create SVM classifier for 4's and 9's
foursNinesSVM = fitcsvm(foursNinesTrain',foursNinesTrainLabs);
predFoursNinesTestLabs = predict(foursNinesSVM,foursNinesTest');

% Calculate accuracy
err = abs(predFoursNinesTestLabs' - foursNinesTestLabs);
falseIdx = find(err ~= 0);
err(falseIdx) = 1;
errNum = sum(err);
foursNinesSVMSucRate = 1 - errNum/numTestImages;

%% Create classification tree for 1's and 0's
onesZerosTrain = projFull(1:rank,onesZerosIdx);
onesZerosTrainLabs = 0*onesZerosIdx';
onesZerosTrainLabs(1:length(onesIdx)) = 1;
onesZerosTrainLabs(length(onesIdx)+1:end) = 0;

onesZerosTest = testMat(1:rank,onesZerosTestIdx);
onesZerosTestLabs = 0*onesZerosTestIdx';
onesZerosTestLabs(1:length(onesTestIdx)) = 1;
onesZerosTestLabs(length(onesTestIdx)+1:end) = 0;

% Create tree
onesZerosTree = fitctree(onesZerosTrain',onesZerosTrainLabs);
predOnesZerosTestLabs = predict(tree,onesZerosTest');

% Calculate accuracy
err = abs(predOnesZerosTestLabs' - onesZerosTestLabs);
falseIdx = find(err ~= 0);
err(falseIdx) = 1;
errNum = sum(err);
onesZerosTreeSucRate = 1 - errNum/numTestImages;

%% Create SVM classifier for 1's and 0's
onesZerosSVM = fitcsvm(onesZerosTrain',onesZerosTrainLabs);
predOnesZerosTestLabs = predict(onesZerosSVM,onesZerosTest');

% Calculate accuracy
err = abs(predOnesZerosTestLabs' - onesZerosTestLabs);
falseIdx = find(err ~= 0);
err(falseIdx) = 1;
errNum = sum(err);
onesZerosSVMSucRate = 1 - errNum/numTestImages;
```

Listing 1: This is the code used for training and testing the machine learning algorithms discussed in this paper on the MNIST dataset.

Appendix C Additional Figures

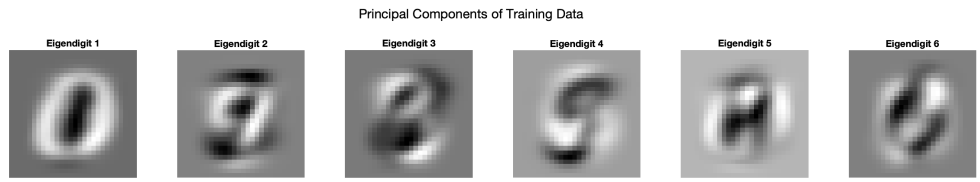

Figure 4: Principal components/"eigendigits" of the training data

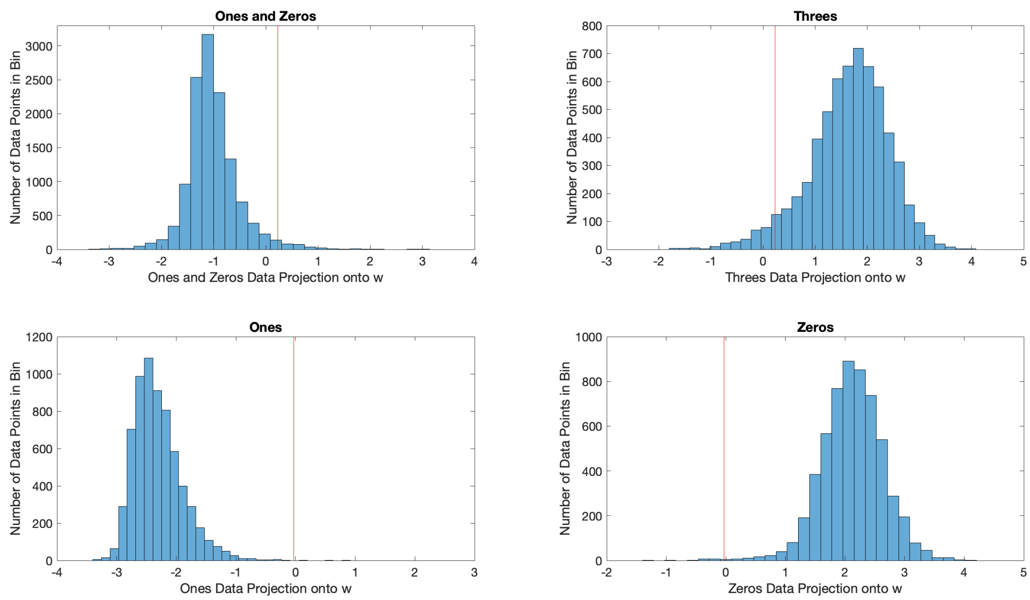

Figure 5: Results of the three-way LDA for ones, zeros, and threes

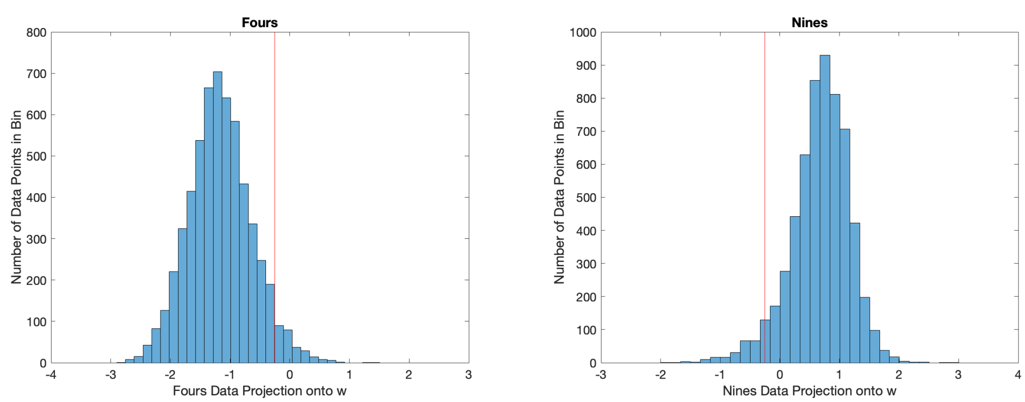

Figure 6: Results of the two-way LDA for fours and nines