AMATH 482 Homework 4

Sophia Jannetty

March 10, 2021

Abstract

Here, Linear Discriminant Analysis (LDA), Decision Tree Classification, and Support Vector Machine (SVM) algorithms were used to analyze the MNIST dataset. Initially SVD was performed on MNIST training data to investigate the singular value spectrum and determine the number of modes needed to accurately represent the data (344). Subsequent analysis was performed considering 344 features. Each digit pair was classified using LDA, decision tree classification, and SVM algorithms. The three digits 1, 2, and 3 were classified using LDA. Last all 10 digits were simultaneously classified using decision tree classification and SVM. Training and test accuracies were calculated for each analysis. Decision Tree classification was the least successful of the algorithms used.

1 Introduction and Overview

The MNIST dataset is a set of 60,000 training and 10,000 test images of the hand written digits 0 - 9. Each image is 28x28 pixels and the center of mass of each digit is centered in the image [5]. Here I perform SVD on the MNIST training data to determine how many dimensions I can eliminate while maintaining an accurate representation of the data. I performed this dimensionality reduction before using the classification algorithms. During initial exploration data were projected onto three V modes to see which digits were closer together and farther apart when projected onto those three vectors. Each digit pair was classified using LDA, decision tree classification, and SVM algorithms and both training and test accuracies were calculated. Then three digit classification was performed on the digits 1, 2, and 3 using LDA. Last 10 digit classification was performed using decision tree classification and SVM.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

Assume we have a large data set $X \in \mathbb{C}^{nxm}$ in which each column x_k contains pixel data from a distinct image in the MNIST data set. The SVD expresses this matrix as the product of three constituent matrices as shown in equation 1.

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

The economy SVD is a version of the SVD that minimizes the dimensions of these matrices to eliminate the zero entries in Σ (this reduced Σ is now called $\hat{\Sigma}$) and to eliminate the corresponding columns of U (this reduced U is now called $\hat{U}$. Equation 2 shows to reduced SVD [3]. I used the reduced SVD for this project so for the remainder of this document I will use the terminology associated with the reduced SVD.

$$X = \hat{U}\hat{\Sigma}V^* \tag{2}$$

The values σ_n along the diagonal of $\hat{\Sigma}$ are called the singular values of matrix A and are ordered from smallest to largest. The vectors u_n that make up the columns of $\hat{U}$ are called the left singular values of X and the vectors v_n that make up the columns of V are called the right singular values of X (note that the MATLAB implementation of the econ SVD returns matrices $\hat{U}$, $\hat{\Sigma}$, and V, NOT the complex conjugate transpose V^* in Equation 2. In this document I will describe V, not V^*).

2.1.1 Interpretation of $\hat{U}$, $\hat{\Sigma}$, and V Matrices in context of MNIST Dataset

The columns of $\hat{U}$ will have the same dimensions as the columns of X. Each column contains a principal component, or eigendigit, which is a basis vector that describes some amount of variance present in the rows of X. These basis vectors are orthonormal and hierarchically arranged such that the principal component in the first column is in the direction of more of the variance in the rows of X than the basis vector in the second column, and so on. The number of columns will be equal to the number of rows in X. One can construct each digit present in the dataset via a weighted sum of each of these eigendigits. The first 10 principal components of the MNIST training data can be seen in Figure 3 in Appendix C.

$\hat{\Sigma}$ holds the singular values of the dataset. All are all non-negative and represent the energy in the dataset captured by the corresponding basis vector, or the relative amount of variance in the cols of X that is in the direction of the corresponding basis vector. These values are also hierarchically arranged, such that the first value is the largest and the value in each subsequent row/col is less than the value in the previous row/col [4]. The value in each row/col m corresponds to the variance in the dataset in the direction of the basis vector in $\hat{U}[:, m]$.

Each column of V also corresponds to a basis vector (the m column corresponds to the basis vector in the m col of $\hat{U}$), however each row n corresponds to a column n of X (or a particular digit image in the dataset). Each element [n,m] of V is the projection of the mth standard basis vector onto the nth column of X. Conceptually, this tells me how much of variance in the nth column (or image) of X is in the direction of the mth standard basis vector. If talking about the conjugate transpose of V V^*, each entry [n,m] of V^* tells gives the inner product between the nth column of X and the mth column of $\hat{U}$, or the mth basis vector. These are the weights that must be applied to each of the principal components (or basis vectors) in a weighted sum in order to recreate each image.

2.1.2 Use for Matrix Approximation

The SVD provides a hierarchy of optimal low-rank approximation of matrix X. Matrix X is the sum of r rank-one matrices calculated in Equation 3 [4].

$$X = \sum_{j=1}^{r} \sigma_j u_j v_j^* \tag{3}$$

The SVD can be used to see how many independent basis vectors are present in the dataset and will order them from most to least important. By determining the proportion of the overall energy in the matrix each basis vector describes (found by dividing each singular value by the sum of all the singular values) you can set a threshold for how much of the energy you want represented in your matrix approximation and determine how many singular values (or ranks) you need to keep to reach that threshold. By setting r to this number of ranks in Equation 3, you can create a low-rank approximation of your input Matrix and can eliminate all information related to extraneous singular values. This allows for accurate and efficient matrix reduction.

2.1.3 Use for Determining Features to Consider for Classification (Dimension Reduction)

Classification can be computationally challenging if considering all principal components (or features) of a dataset. To minimize runtime/computational complexity while maintaining accuracy, a subset of principal components can be considered. This can be determined by identifying the k rank-approximation of data you need to capture a certain amount of energy in the dataset (as described above) and projecting your data onto k principal components. Equation 2 can be rearranged such that $\hat{U}^* X = \hat{\Sigma} V^*$, thus this projection can be constructed via finding the inner product of the input data matrix X and (the first k columns of $\hat{U}$) transposed, or via subsetting the first k rows of the inner product of $\hat{\Sigma}$ and V^*.

2.2 Linear Discriminant Analysis (LDA)

The LDA uses statistical information about the SVD decomposition of the digit images to make a decision about what digit is represented in each image. In LDA, data sets are projected onto a new line that maximizes the distance between inter-class data while minimizing the distance between intra-class data. Mathematically, we are trying to construct a projection w as outlined in Equation 4.

$$w = argmax \frac{w^T S_b w}{w^T S_W w} \tag{4}$$

where the scatter matrices for between-class S_b and within-class S_w data are given by Equations 5 and 6

$$S_b = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{5}$$

$$S_W = \sum_{j=1}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{6}$$

2.3 Decision Tree Classifier

The Decision Tree algorithm is a supervised machine learning algorithm used in this exercise for classification. During training, the algorithm learns simple decision rules about the training data's features to differentiate between categories [1]. These criteria are then applied to test data, resulting in classification predictions for the test data.

2.4 Support Vector Machines (SVM)

The Support Vector Machine is also a supervised machine learning algorithm used in this exercise for classification. An SVM maps training data in high-dimensional space such that a gap exists between different categories. It then maps test data into this same space and assigns categories based on where the data fall in relation to the gaps [2].

3 Algorithm Implementation and Development

3.1 SVD Analysis of Training Digit Images and V Mode Plotting

MNIST data were imported using the mnist_parse function and images were reshaped from 28x28 matrices to 784x1 column vectors. These vectors were stored in a matrix and the econ svd of this matrix was found using MATLAB's svd function, resulting in the creation of matrices U, S, and V. The first ten principal components (stored in the first ten columns of U) were reshaped back into 28x28 matrices and displayed as images (see Figure 3 in Appendix C). Singular values were plotted (see Figure 4 in Appendix C) and low rank approximations of the data were created using Equation 3. Columns of these approximations were reshaped into 28x28 pixel images and inspected to determine how much many singular values to use as features during analysis (see Figure 5 in Appendix C). The number of singular values needed to capture 90% of the energy in the system was calculated by adding each successive value stored in S until the sum was greater than .9 (see Figure 6 in Appendix C). This value was determined to be 334 and all subsequent data analysis was performed considering this many features (the rank r of the digit space was determined to be 334). The second, third, and fourth columns of V were plotted and each number was labeled using a different color (see Figure 1) . The plot was iteratively populated with data from rows corresponding on label at a time, so first all the rows (of the three specified columns) corresponding to images of 0's were plotted, then 1's, etc. Other V modes were plotted visually inspected but no discernable clustering improvement was noted.

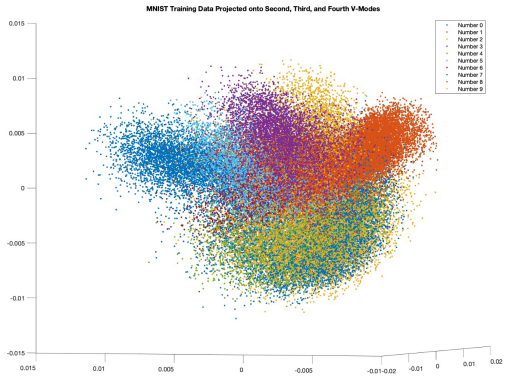

Figure 1: MNIST Training data projected onto second, third, and fourth V Modes. Each digit is labeled with a different color.

3.2 LDA For Classifying Two Digits

The LDA was run on each pair of numbers in the dataset. For each pair the following four vectors were constructed: a training data vector containing all training images of both digits to be compared, a training label vector containing all labels of images in the training data vector, a test data vector containing all test images of both digits to be compared, and a test label vector containing all labels of images in the test data vector. The econ SVD was run on the test data matrix using MATLAB's svd function. The number of principal components to project the data onto can be calculated for each digit pair or hard coded to a value (for the purposes of comparison I have hard coded the algorithm to incorporate 334 principal components as this was the number of modes required to capture 90% of the energy of the entire dataset during the SVD analysis described in the above section). The data are projected onto these modes by multiplying S*V' and keeping the first 334 rows of the resulting matrix.

This new data matrix is separated into one matrix $first$ containing all data for the first digit and one matrix *second* containing all data for the second digit. The row-wise means of these data are calculated and used to calculate the within and between class variance scatter matrices as described in equations 6 and 5. MATLAB's eig function is used to find the basis vectors in matrix V2 and corresponding eigenvalues in matrix D of the scatter matrices. The basis vector corresponding with the largest eigenvalue is the basis on which the data should be projected. The row of V2 corresponding to the largest element of D is extracted as its own vector and normalized to its own 2-norm and saved in vector w. The training data are projected onto this basis by multiplying $first$ by w resulting in vector $vfirst$, and multiplying *second* by w resulting in vector *vsecond*.

$vfirst$ and *vsecond* are vectors containing one value per input training image. A threshold that corresponds to the point on the basis projection line that differentiates between one digit and another is then determined. The average of the first digit is made to be less than the average of the second digit such firstDigitVals ¡ threshold ¡ secondDigitVals. The threshold is then determined by counting up from the smallest first digit value and down from the largest second digit value and averaging the values the first time the second digit value is less than the first digit value. See Figure 7 in Appendix C for visualizations of these projections and thresholds in the best and worst case digit pair scenarios.

The test data are projected onto the same modes the training data were projected onto by multiplying the test data by $U(:, 334)^T$. These data are then projected onto the determined LDA basis by multiplying by w. All images with values less than the threshold were assigned the first digit label and all others the second digit label. Percent accuracy was calculated for both the test and training data by comparing the assigned labels with the actual labels.

3.3 LDA For Classifying Three Digits

The four vectors described above were constructed containing train/test image and label data three digits instead of two (the digits 1, 2, and 3). The econ SVD of the training data matrix was found using MATLAB's svd function. The training and test data were projected onto 344 modes by multiplying each data's matrix by $U(:, 334)^T$. These matrices and the training labels were then fed into the classify function which returned predicted labels for the test data and the training error. The percent accuracy on the test data was calculated by comparing the predicted test labels to the actual test labels.

3.4 Decision Tree Classification

Each pair of digits was classified using the decision tree classifier and the full dataset of test and training images to see how well the tree could distinguish between each pair of two digits and to assess how well it could assign classifications to all 10 digits at once. The method for using the tree was the same in each case. The four vectors described in the LDA for two digits section above were constructed for the two digit comparisons (the full data and label datasets were used for the 10 digit classifier). Data were projected onto 344 modes by multiplying the data matrices by $U(:, 334)^T$. The training data and training labels were then passed to MATLAB's fitctree function, which returned a trained tree. This tree and the test data were then passed to MATLAB's predict function, which returned a vector of predicted labels for the test data. The test accuracy was calculated by comparing this vector to the test label vector and the accuracy on the training data was calculated by passing the tree to the MATLAB's cvloss function.

3.5 SVM Classification

As with the tree, SVM classification was used on each digit pair and on the full dataset. The four vectors described in the LDA for two digits section above were constructed for the two digit comparisons (the full data and label datasets were used for the 10 digit classifier). Data were projected onto 344 modes by multiplying the data matrices by $U(:, 334)^T$. Each data matrix was then divided by the maximum value in the training data matrix to put all of the training values between 0 and 1 (ans hopefully most if not all the test values in that range as well) in order to improve runtime. For the pairwise comparisons, the normalized training data and training labels were passed to MATLAB's fitcsvm function returning a SVM model. For the 10 digit classification the training data and labels were passed to MATLAB's fitcecoc function returning a model. After model construction in both cases, MATLAB's predict function was then given the model and the testing data as parameters, returning a vector of predicted labels. The test percent accuracy was calculated by comparing this vector to the actual test labels and the training data accuracy was calculated using MATLAB's loss function.

4 Computational Results

344 Modes were needed to capture 90% of the energy in the training dataset, so all classification algorithms were run considering 334 features.

4.1 Pairwise Digit Classification using LDA, Decision Tree, and SVM

See Figure 2 for plots of training and test accuracies for each algorithm run on each digit pair.

4.1.1 Training Data Accuracy

The lowest training accuracy rate for LDA was 96.04% when classifying 3's and 5's. The highest training accuracy rate for LDA was 99.88% when classifying 6's and 7's. The lowest training accuracy rate for the decision tree was 89.56% when classifying 4's and 9's. The highest training accuracy rate for the decision tree was 99.68% when classifying 1's and 0's. The lowest training accuracy rate for SVM was 96.10% when classifying 3's and 5's. The highest training accuracy rate for SVM was 99.91% when classifying 1's

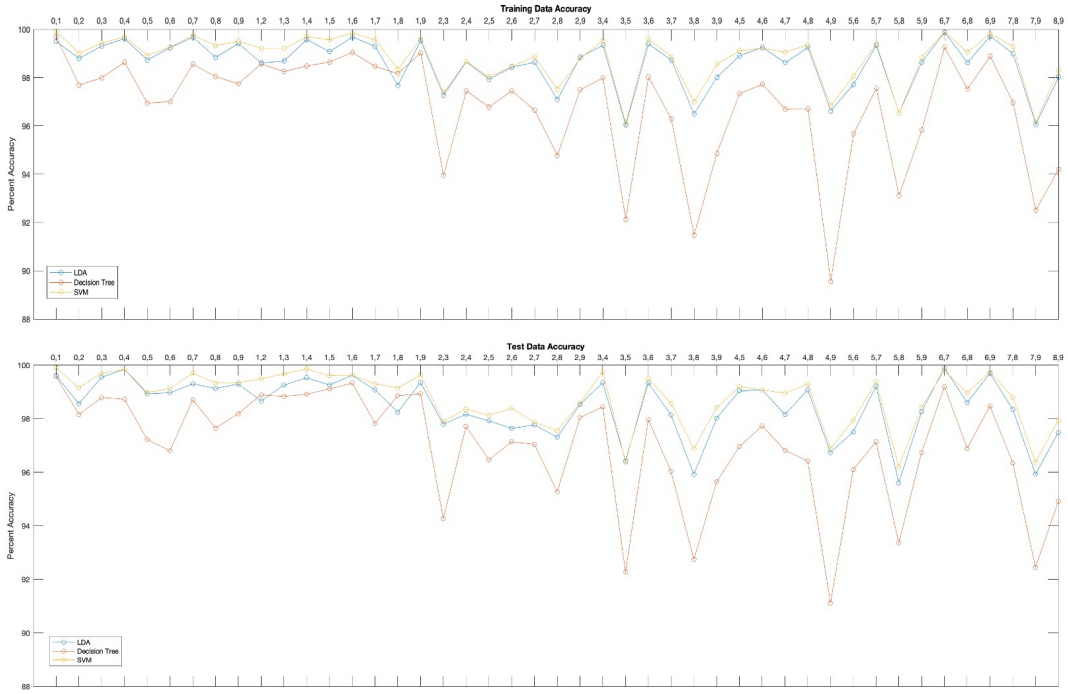

Figure 2: Percent accuracy of label prediction for each digit pair on training (top) and test (bottom) MNIST data for LDA, Decision Tree Classifier, and SVM Classification. All algorithms run considering 334 features.

and 0's. In general, the decision tree had a lower training accuracy rate than the LDA and the SVM. Trends concerning which digit pairs had higher/lower relative training accuracies were generally consistent throughout all algorithms (ex. 4 and 9 had a relatively low success rate in for all algorithms, 0 and 1 had a relatively high success rate for all algorithms, see Figure 2 top).

4.1.2 Test Data Accuracy

The lowest test accuracy rate for LDA was 95.61% when classifying 5's and 8's. The highest test accuracy rate for LDA was 99.90% when classifying 6's and 7's. The lowest test accuracy rate for the decision tree was 91.11% when classifying 4's and 9's. The highest training accuracy rate for the decision tree was 99.57% when classifying 1's and 0's. The lowest test accuracy rate for SVM was 96.10% when classifying 5's and 8's. The highest test accuracy rate for SVM was 99.91% when classifying 1's and 0's. In general, the decision tree had a lower test accuracy rate than the LDA and the SVM. Trends concerning which digit pairs had higher/lower relative test accuracies were generally consistent throughout all algorithms and digit pairs that had relatively high training accuracies for an algorithm also tended to have relatively high test accuracies for that algorithm and vice versa. (see Figure 2 bottom).

4.2 Three Digit LDA Classification

The training percent accuracy for the three digit LDA classifier run on the digits 1, 2, and 3 was 96.22%. The test percent accuracy was 96.82%.

4.3 10 Digit Classification using Decision Tree and SVM

The training percent accuracy for 10 digit classification with the decision tree was 82.95% and the test percent accuracy with the decision tree was 83.53%. The training percent accuracy for 10 digit classification with the SVM was 93.88% and the test percent accuracy with the SVM was 93.90%.

5 Summary and Conclusions

For pairwise classification of the digits, the lowest test data classification accuracy rate for the LDA (95.61%) was seen when classifying 5's and 8's. The decision tree had an even worse test data performance when classifying 5's and 8's with an accuracy rate of 93.35%. The SVM outperformed the LDA on this pair with an accuracy of 96.20%. The highest test data classification accuracy rate for the LDA (99.90%) was seen when classifying 6's and 7's. The decision tree had a slightly worse test data performance when classifying 5's and 8's with an accuracy rate of 99.17%. The SVM outperformed the LDA on this pair with an accuracy of 99.89%.

For pairwise classification, digit pairs tended to be either easier or harder to classify across all classification algorithms. I expected to be able to predict which digit pairs were easier and more difficult to separate based on how similar the digits look to my eye (for example I expected 7's and 1's to have a very low success rate) but my predictions were largely incorrect. I also expected that digits that were more mixed in the projection onto three V-modes seen in Figure 1 would have a lower classification success rate (like 4 and 7 which seem to be fairly well mixed in the lower part of the image). This was also not the case, likely because the base projected on for LDA is not a V-mode and thus will yield different clustering. The decision tree classifier tended to perform the worst out of the algorithms.

With a 96.82% test success rate the three digit LDA classification performed within the range of accuracy seen in the two digit classifications, but the accuracy was notably lower than the LDA accuracies observed when separating 1's and 2's (98.66%), 1's and 3's (99.25%), and 2's and 3's(97.80%). It makes sense that the upper limit of accuracy for a three digit LDA would be the lowest accuracy observed when separating any two of the considered digits, but I expected the three digit error rate to be equal to the sum of the error rates between each digit. This was not the case (the three digit error rate was 3.18% and the sum of the two digit error rates is 4.29%).The decision tree ten digit classifier had the lowest success rate of any classification attempted (82.95%). The SVM 10 digit classifier performed better with an accuracy of 93.90%. This was the lowest SVM accuracy rate seen in this project.

Here I see the trend that as the number of categories increased, the complexity of the problem increased and the accuracy of the resulting predictions decreased. If I had more computing power I would be interested in rerunning my code on the full dataset without any dimension reduction to see if including more modes would increase accuracy.

References

[1] *1.10. Decision Trees¶*. URL: https://scikit-learn.org/stable/modules/tree.html.

[2] *1.4. Support Vector Machines¶*. URL: https://scikit-learn.org/stable/modules/svm.html.

[3] Steven L. Brunton and Jose Nathan Kutz. *Data-driven science and Engineering: machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[4] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[5] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *THE MNIST DATABASE*. URL: http://yann.lecun.com/exdb/mnist/.

Appendix A MATLAB Functions

Table 1: Notable Matlab Functions

`[im, labs] = mnist_parse('imfile', 'labsfile')`	function for parsing MNIST data. Takes files downloaded from [5] and reads them into a data matrix of 28x28 matrices and a label vector holding all relevant digit labels
`B = reshape(A, sz)`	reshapes a given vector into a vector of specified size sz
`imagesc(C)`	displays data in array C as image that uses full range of colors in the colormap
`[U,S,V] = svd(AllNums, 'econ')`	performs single value decomposition on data in matrix AllNums. 'econ' flag eliminates zero rows/cols from U and S.
`cumEnergy = cumsum(eigvals)`	returns vector containing cumulative sums of elements of eigvals
`k = find(X == 1)`	returns a vector of the same length of X containing 1's in indices where the value equals 1 and 0's otherwise. Used to parse individual digits out of data.
`[V2, D] = eig(Sb, Sw)`	returns diagonal matrix D of generalized eigenvalues and full matrix V2 whose columns are the corresponding left eigenvectors, so that W'*Sb = D*W'*Sw
`n = norm(v, 2)`	returns the generalized vector v's 2-norm
`tree = fitctree(training', traininglabels)`	returns a fitted binary classification decision tree based on data in training' and labels in traininglabels
`tree = fitctree(training', traininglabels)`	returns a fitted binary classification decision tree based on data in training' and labels in traininglabels
`class = predict(tree, testdata')`	returns vector class of predicted labels for testdata', predicted using fitted binary classifier tree
`error = cvloss(tree)`	returns cross-validated classification error for classification tree tree
`Mdl = fitcsvm(training', traininglabels)`	trains or cross-validates a SVM model for one-class and two-class classification using data in matrix training and labels in vector traininglabels
`Mdl = fitcecoc(training', traininglabels)`	trains or cross-validates multiclass SVM model using matrix training and labels in vector traininglabels
`error = loss(Mdl, training', totallabels)`	returns classification error for model Mdl computed using data training' and true training' labels in vector totallabels

Appendix B MATLAB Code

B.1 Initial Data Exploration

```
%% Read in data
close all; clear all;
[trainimages, trainlabels] = mnist_parse('train-images-idx3-ubyte', 'train-
    labels-idx1-ubyte');

%Make matrix to hold all num info
picheight = 28;
picwidth = 28;
num_rows = picheight * picwidth;
num_cols = length(trainlabels);
allNums = zeros(num_rows, num_cols);

for i = 1:num_cols
    %reshape into col vector, store with each col = different image
    allNums(:,i) = reshape(trainimages(:,:,i), num_rows, 1);
end

figure()
plotnum(allNums(:,1))

%SVD
[U,S,V] = svd(allNums, 'econ');
eigvals = diag(S);

%% Determine number of modes to incorperate
energyThreshold = .9;
normalized_eigvals = eigvals / sum(eigvals);
cumEnergy = cumsum(eigvals)/sum(eigvals);
energyIndex = find(cumEnergy >energyThreshold,1);
%% Plot First 4 PCA Modes
close all;
figure()
for j = 1:10
    subplot(2,5,j)
    imagesc(reshape(U(:, j), picheight, picwidth))
end
sgtitle('First 10 Principal Components of MNIST Training Data')

%% Plot singular values
figure(2)
subplot(2,1,1)
plot(diag(S), 'ko', 'Linewidth', 2) %X, Y plot
set(gca, 'Fontsize', 16, 'Xlim', [0, 400])

subplot(2,1,2)
semilogy(diag(S), 'ko', 'Linewidth', 2) %log plot
set(gca, 'Fontsize', 16, 'Xlim', [0,400])
%slow decay, first is most important
sgtitle('Singular Values of MNIST Training Data')

%% Plot Normalized Singular Values And Threshold
```

```
figure()
sline = diag(S) / sum(diag(S));
plot(sline*100, 'ko', 'Linewidth', 2) %X, Y plot
set(gca, 'Fontsize', 16, 'Xlim', [0, 400])
hold on
cumline = cumEnergy*100;
plot(cumline, 'Linewidth', 1)
hold on
xline(energyIndex)
yline(90)
title("Normalized Singular Values of MNIST Training Data")
ylabel('Percent Energy Represented')
xlabel('Principal Component')
%% Plot first 3 V modes in 3D
close all;
%# find out how many clusters you have
uClusters = unique(string(trainlabels));
nClusters = length(uClusters);

%# create colormap
%# distinguishable_colormap from the File Exchange
%# is great for distinguishing groups instead of hsv
cmap = hsv(nClusters);

%# plot, set DisplayName so that the legend shows the right label
figure
for iCluster = 1:nClusters
    this_num = (iCluster-1);
    clustIdx = string(trainlabels)==uClusters(iCluster);
    plot3(V(clustIdx,4),V(clustIdx,2),...
        V(clustIdx,3), '.', 'MarkerSize', 5,...
       'DisplayName',sprintf('Number %i',(this_num)));
   hold on
end

legend('show');
title('MNIST Training Data Projected onto Second, Third, and Fourth V-Modes')

%% Make low rank approximations of data

%determine modes for 80 and 90 %
energyThreshold = .8;
modesfor80 = find(cumEnergy >energyThreshold,1);
modesfor90 = energyIndex;
modesfor100 = length(eigvals);


%make low rank approximations
rank80percent = U(:,1)*eigvals(1)*V(:,1)';
i = 1;
while i < modesfor80
    i = i + 1;
    rank80percent = rank80percent + (U(:,i)*eigvals(i)*V(:,i)');
end
```

```
105
106 rank90percent = rank80percent;
107 while i < modesfor90
108     disp(i)
109     i = i + 1;
110     rank90percent = rank90percent + (U(:,i)*eigvals(i)*V(:,i)');
111 end
112 %% Make figure comparing numbers
113 figure()
114 subplot(3,5,4)
115 plotnum(rank80percent(:,12))
116 subplot(3,5,1)
117 plotnum(rank80percent(:,2))
118 ylabel("80% Energy")
119 subplot(3,5,3)
120 plotnum(rank80percent(:,3))
121 subplot(3,5,2)
122 plotnum(rank80percent(:,4))
123 subplot(3,5,5)
124 plotnum(rank80percent(:,5))
125
126 subplot(3,5,9)
127 plotnum(rank90percent(:,12))
128 subplot(3,5,6)
129 plotnum(rank90percent(:,2))
130 ylabel("90% Energy")
131 subplot(3,5,8)
132 plotnum(rank90percent(:,3))
133 subplot(3,5,7)
134 plotnum(rank90percent(:,4))
135 subplot(3,5,10)
136 plotnum(rank90percent(:,5))
137
138 subplot(3,5,14)
139 plotnum(allNums(:,12))
140 subplot(3,5,11)
141 plotnum(allNums(:,2))
142 ylabel("100% Energy")
143 subplot(3,5,13)
144 plotnum(allNums(:,3))
145 subplot(3,5,12)
146 plotnum(allNums(:,4))
147 subplot(3,5,15)
148 plotnum(allNums(:,5))
149
150 sgtitle('Low-Rank Approximations of Data')
151
152 %% Functions
153 function [] = plotnum(numvec)
154     imagesc(reshape(numvec, 28, 28))
155 end
156
157 function [images, labels] = mnist_parse(path_to_digits, path_to_labels)
158
```

```
% The function is curtesy of stackoverflow user rayryeng from Sept. 20,
% 2016. Link: https://stackoverflow.com/questions/39580926/how-do-i-load-in-
    the-mnist-digits-and-label-data-in-matlab

% Open files
fid1 = fopen(path_to_digits, 'r');

% The labels file
fid2 = fopen(path_to_labels, 'r');

% Read in magic numbers for both files
A = fread(fid1, 1, 'uint32');
magicNumber1 = swapbytes(uint32(A)); % Should be 2051
fprintf('Magic Number - Images: %d\n', magicNumber1);

A = fread(fid2, 1, 'uint32');
magicNumber2 = swapbytes(uint32(A)); % Should be 2049
fprintf('Magic Number - Labels: %d\n', magicNumber2);

% Read in total number of images
% Ensure that this number matches with the labels file
A = fread(fid1, 1, 'uint32');
totalImages = swapbytes(uint32(A));
A = fread(fid2, 1, 'uint32');
if totalImages ~= swapbytes(uint32(A))
    error('Total number of images read from images and labels files are not
        the same');
end
fprintf('Total number of images: %d\n', totalImages);

% Read in number of rows
A = fread(fid1, 1, 'uint32');
numRows = swapbytes(uint32(A));

% Read in number of columns
A = fread(fid1, 1, 'uint32');
numCols = swapbytes(uint32(A));

fprintf('Dimensions of each digit: %d x %d\n', numRows, numCols);

% For each image, store into an individual slice
images = zeros(numRows, numCols, totalImages, 'uint8');
for k = 1 : totalImages
    % Read in numRows*numCols pixels at a time
    A = fread(fid1, numRows*numCols, 'uint8');

    % Reshape so that it becomes a matrix
    % We are actually reading this in column major format
    % so we need to transpose this at the end
    images(:,:,k) = reshape(uint8(A), numCols, numRows).';
end

% Read in the labels
labels = fread(fid2, totalImages, 'uint8');
```

```
211
212 % Close the files
213 fclose(fid1);
214 fclose(fid2);
215
216 end
```

B.2 All Pairwise Digit Classification

```
%% Read in data
close all; clear all;
[trainimages, trainlabels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');

%Make matrix to hold all num info
picheight = 28;
picwidth = 28;
num_rows = picheight * picwidth;
num_cols = length(trainlabels);
allNums = zeros(num_rows, num_cols);


for i = 1:num_cols
    %reshape into col vector, store with each col = different image
    allNums(:,i) = reshape(trainimages(:,:,i), num_rows, 1);
end

%% read in test data
[testimages, testlabels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');
%Make matrix to hold all test num info
test_num_cols = length(testlabels);
allTest = zeros(num_rows, test_num_cols);
for i = 1:test_num_cols
    %reshape into col vector, store with each col = different image
    allTest(:,i) = reshape(testimages(:,:,i), num_rows, 1);
end

%% Iterate through all the digits
accuracy_labels = ["0,1", "0,2", "0,3", "0,4", "0,5", "0,6", "0,7", "0,8",...
    "0,9", "1,2", "1,3", "1,4", "1,5", "1,6", "1,7", "1,8", "1,9", "2,3", ...
    "2,4", "2,5", "2,6", "2,7", "2,8", "2,9", "3,4", "3,5", "3,6", "3,7", ...
    "3,8", "3,9", "4,5", "4,6", "4,7", "4,8", "4,9", "5,6", "5,7", "5,8", ...
    "5,9", "6,7", "6,8", "6,9", "7,8", "7,9", "8,9"];

trainaccuracy_percents = zeros(length(accuracy_labels), 1);
trainaccuracy_percents_tree = zeros(length(accuracy_labels), 1);
trainaccuracy_percents_svm = zeros(length(accuracy_labels), 1);
testaccuracy_percents = zeros(length(accuracy_labels), 1);
testaccuracy_percents_tree = zeros(length(accuracy_labels), 1);
testaccuracy_percents_SVM = zeros(length(accuracy_labels), 1);
modes_used = zeros(length(accuracy_labels), 1);
accuracy_percent_index = 1;

%% Run on all number combinations
for i = 0:1:8
    for j = i+1:9
        %seperate out each type of number
        firstidx = find(trainlabels == i);
        firstlabels = trainlabels(firstidx);
        firstmat = allNums(:, firstidx);
        numfirst = length(firstlabels);
```

```
52
53          firsttestidx = find(testlabels == i);
54          firsttestlabels = testlabels(firsttestidx);
55          firsttestmat = allTest(:, firsttestidx);
56          numfirsttest = length(firsttestidx);
57
58          secondidx = find(trainlabels ==j);
59          secondlabels = trainlabels(secondidx);
60          secondmat = allNums(:, secondidx);
61          numsecond = length(secondlabels);
62
63          secondtestidx = find(testlabels == j);
64          secondtestlabels = testlabels(secondtestidx);
65          secondtestmat = allTest(:, secondtestidx);
66          numsecondtest = length(secondtestidx);
67
68          totalmat = [firstmat, secondmat];
69          totallabels = [firstlabels', secondlabels'];
70
71          totaltestmat = [firsttestmat, secondtestmat];
72          totaltestlabels = [firsttestlabels', secondtestlabels'];
73
74          [trainaccuracy_percents(accuracy_percent_index), ...
75              trainaccuracy_percents_tree(accuracy_percent_index), ...
76              trainaccuracy_percents_svm(accuracy_percent_index), ...
77              testaccuracy_percents(accuracy_percent_index),...
78              testaccuracy_percents_tree(accuracy_percent_index),...
79              testaccuracy_percents_SVM(accuracy_percent_index),...
80              modes_used(accuracy_percent_index)] = ...
81              twodigitLDA(totalmat, totallabels, numfirst, numsecond,...
82              totaltestmat, totaltestlabels,numfirsttest, numsecondtest, ...
83              picheight, picwidth, false);
84
85          accuracy_percent_index = accuracy_percent_index+1;
86      end
87  end
88
89  %% Plot percents train LDA
90  figure()
91  plot (trainaccuracy_percents, '-o');
92  set(gca,'xaxisLocation','top')
93  xticks(1:length(accuracy_labels));
94  xticklabels(accuracy_labels);
95  hold on
96  %%
97  %trainaccuracy_percents_tree = trainaccuracy_percents_tree* 100;
98  plot (trainaccuracy_percents_tree, '-o');
99  set(gca,'xaxisLocation','top')
100 xticks(1:length(accuracy_labels));
101 xticklabels(accuracy_labels);
102 hold on
103 %trainaccuracy_percents_svm = trainaccuracy_percents_svm * 100;
104 plot (trainaccuracy_percents_svm, '-o');
105 set(gca,'xaxisLocation','top')
```

```
106 xticks(1:length(accuracy_labels));
107 xticklabels(accuracy_labels);
108
109 ylabel('Percent Accuracy')
110 title("Training Data Accuracy")
111 legend('LDA','Decision Tree', 'SVM')
112
113 %% Plot percents test LDA
114  testaccuracy_percents = testaccuracy_percents * 100;
115  testaccuracy_percents_tree = testaccuracy_percents_tree * 100;
116  testaccuracy_percents_SVM = testaccuracy_percents_SVM * 100;
117
118 figure()
119 plot (testaccuracy_percents, '-o');
120 set(gca,'xaxisLocation','top')
121 xticks(1:length(accuracy_labels));
122 xticklabels(accuracy_labels);
123 hold on
124 %worst 5 and 8, best 6 and 7
125
126 % Plot percents test Tree
127 plot (testaccuracy_percents_tree, '-o');
128 set(gca,'xaxisLocation','top')
129 xticks(1:length(accuracy_labels));
130 xticklabels(accuracy_labels);
131 hold on
132 %worst 4 and 9, best 0 and 1
133 % plot percents test SVM
134
135 plot (testaccuracy_percents_SVM, '-o');
136 set(gca,'xaxisLocation','top')
137 xticks(1:length(accuracy_labels));
138 xticklabels(accuracy_labels);
139 title("Test Data Accuracy")
140 legend('LDA','Decision Tree', 'SVM')
141 ylim([88,100])
142 ylabel('Percent Accuracy')
143 %% Functions
144
145 function [trainpercentaccuracylda, trainpercentaccuracytree,...
146     trainpercentaccuracysvm, testpercentaccuracylda,...
147     testpercentaccuracytree, testpercentaccuracysvm, modesused] = twodigitLDA(
        totalmat,...
148     totallabels, numfirst, numsecond, testmat, testlabelmat, numfirsttest,
        numsecondtest,...
149     picheight, picwidth, plot)
150
151 trainpercentaccuracylda = 0;
152 trainpercentaccuracytree = 0;
153 trainpercentaccuracysvm = 0;
154
155 testpercentaccuracylda = 0;
156 testpercentaccuracytree = 0;
157 testpercentaccuracysvm = 0;
```

```
158
159 % SVD on this new matrix
160 [U,S,V] = svd(totalmat, 'econ');
161 eigvals = diag(S);
162
163
164 %% Plot first four principal components
165 if plot == true
166     for k = 1:4
167         subplot(2,2,k)
168         ut1 = reshape(U(:,k),picheight,picwidth);
169         ut2 = rescale(ut1);
170         imagesc(ut2)
171     end
172 end
173 %% Plot singular values
174 if plot == true
175     figure()
176     subplot(2,1,1)
177     plot(diag(S),'ko','Linewidth',2)
178     set(gca,'Fontsize',16,'Xlim',[0 80])
179     subplot(2,1,2)
180     semilogy(diag(S),'ko','Linewidth',2)
181     set(gca,'Fontsize',16,'Xlim',[0 80])
182 end
183 %% Plot right singular vectors
184 if plot == true
185     figure()
186     for k = 1:3
187         subplot(3,2,2*k-1)
188         plot(1:60,V(1:60,k),'ko-')
189         subplot(3,2,2*k)
190         plot(1:60,V(numfirst:(numfirst+59),k),'ko-')
191     end
192     subplot(3,2,1), set(gca,'Fontsize',12), title(num2str(totallabels(1)))
193     subplot(3,2,2), set(gca,'Fontsize',12), title(num2str(totallabels(numfirst
        +1)))
194     subplot(3,2,3), set(gca,'Fontsize',12)
195     subplot(3,2,4), set(gca,'Fontsize',12)
196     subplot(3,2,5), set(gca,'Fontsize',12)
197     subplot(3,2,6), set(gca,'Fontsize',12)
198 end
199
200
201 %% Determine number of modes to incorperate
202 energyThreshold = .9;
203 normalized_eigvals = eigvals / sum(eigvals);
204 cumEnergy = cumsum(eigvals)/sum(eigvals);
205 energyIndex = find(cumEnergy >energyThreshold,1);
206 if plot == true
207     figure()
208     subplot(1,2,1), plot(normalized_eigvals, '.')
209     %energySV = cumEnergy(energyIndex);
210     subplot(1,2,2), plot(cumEnergy, '.')
```

```
end
energyIndex = 344; %this was determined from running SVD on all test data,
                   % comment out this line to capture 90% of energy for
                   % difference between these two digits specifically
modesused = energyIndex;
%% Project onto PCA modes
feature = energyIndex;
nf = numfirst;
ns = numsecond;
numbers = S*V'; % projection onto principal components: X = USV' --> U'X = SV'
first = numbers(1:feature,1:nf); %Projection of first numbers
second = numbers(1:feature,nf+1:nf+ns); %Projection of second numbers

%% Calculate scatter matrices

mf = mean(first,2);
md = mean(second,2);

Sw = 0; % within class variances
for k = 1:nf
    Sw = Sw + (first(:,k) - mf)*(first(:,k) - mf)';
end
for k = 1:ns
    Sw =  Sw + (second(:,k) - md)*(second(:,k) - md)';
end

Sb = (mf-md)*(mf-md)'; % between class

%% Find the best projection line

[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

%% Project onto w

vfirst = w'*first;
vsecond = w'*second;

%% Make first below the threshold

if mean(vfirst) > mean(vsecond)
    w = -w;
    vfirst = -vfirst;
    vsecond = -vsecond;
end

%% Plot first/second projections (not for function)
if plot == true
    figure()
    plot(vfirst,zeros(nf),'ob','Linewidth',2)
    hold on
    plot(vsecond,ones(ns),'dr','Linewidth',2)
```

```
265     ylim([0 1.2])
266 end
267 %% Find the threshold value
268
269 sortfirst = sort(vfirst);
270 sortsecond = sort(vsecond);
271
272 t1 = length(sortfirst);
273 t2 = 1;
274 while sortfirst(t1) > sortsecond(t2)
275     t1 = t1 - 1;
276     t2 = t2 + 1;
277 end
278 threshold = (sortfirst(t1) + sortsecond(t2))/2;
279
280 %% Plot histogram of results
281
282 minxval = min([sortfirst, sortsecond]);
283 maxxval = max([sortfirst, sortsecond]);
284 if plot == true
285     figure(5)
286     subplot(2,1,1)
287     histogram(sortfirst,30); hold on, plot([threshold threshold], [0 1600],'r'
        )
288     set(gca,'Fontsize',14)
289     xlim([minxval maxxval]);
290     title(num2str(totallabels(1)))
291     subplot(2,1,2)
292     histogram(sortsecond,30); hold on, plot([threshold threshold], [0 1600],'r
        ')
293     set(gca,'Fontsize',14)
294     xlim([minxval maxxval]);
295     title(num2str(totallabels(numfirst+1)))
296 end
297
298 %% Calculate training data percent accuracy
299 wrong_firsts = sum(sortfirst > threshold);
300 wrong_seconds = sum(sortsecond < threshold);
301 wrongtotal = wrong_firsts+wrong_seconds;
302 trainpercentaccuracylda = (1 - wrongtotal/(nf+ns)) * 100;
303
304 %% run test data
305 modesofU = U(:, 1:energyIndex);
306 svdprojection = modesofU' * testmat; %SVD projection
307 pval = w' * svdprojection; %LDA projection
308 resvecs = (pval > threshold);
309 %first below threshold
310 % 0 if first num, 1 if second num
311 label2 = totallabels(numfirst+1);
312 label2idx = find(testlabelmat == label2);
313 hiddenlabels = zeros(length(testlabelmat), 1);
314 hiddenlabels(label2idx) = 1;
315 errnum = sum(abs(resvecs' - hiddenlabels));
316 testpercentaccuracylda = (1 - (errnum/(numfirsttest + numsecondtest)));
```

```
317
318 %%% Classify using Tree
319 training = [first, second]; %these are training data projected onto correct
      number of principal components
320 tree = fitctree(training', totallabels);
321 class = predict(tree, svdprojection');
322 %% Assess Train accuracy of Tree
323 trainpercentaccuracytree = (1 - cvloss(tree));
324 %% Assess Test accuracy of Tree
325 numwrong = sum(testlabelmat ~= class');
326 testpercentaccuracytree = 1 - (numwrong / (length(testlabelmat)));
327
328 %% Classify using SVM
329 sample = svdprojection / max(training(:));
330 training = training / max(training(:));
331 Mdl = fitcsvm(training',totallabels);
332 %% Assess Train accuracy of SVM
333 trainpercentaccuracysvm = 1 - (loss(Mdl, training', totallabels));
334 %% Assess Test accuracy of SVM
335 class = predict(Mdl,sample');
336 numwrong = sum(testlabelmat ~= class');
337 testpercentaccuracysvm = 1 - (numwrong / (length(testlabelmat)));
338
339 end
```

B.3 Three Digit Classification

```
%% Read in data
close all; clear all;
[trainimages, trainlabels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');

%Make matrix to hold all num info
picheight = 28;
picwidth = 28;
num_rows = picheight * picwidth;
num_cols = length(trainlabels);
allNums = zeros(num_rows, num_cols);


for i = 1:num_cols
    %reshape into col vector, store with each col = different image
    allNums(:,i) = reshape(trainimages(:,:,i), num_rows, 1);
end

%% read in test data
[testimages, testlabels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');
%Make matrix to hold all test num info
test_num_cols = length(testlabels);
allTest = zeros(num_rows, test_num_cols);
for i = 1:test_num_cols
    %reshape into col vector, store with each col = different image
    allTest(:,i) = reshape(testimages(:,:,i), num_rows, 1);
end

%% Parse out first, second, and third numbers from training and test data
firstnum = 1;
secondnum = 2;
thirdnum = 3;

firstidx = find(trainlabels == firstnum);
firstlabels = trainlabels(firstidx);
firstmat = allNums(:, firstidx);
numfirst = length(firstlabels);

firsttestidx = find(testlabels == firstnum);
firsttestlabels = testlabels(firsttestidx);
firsttestmat = allTest(:, firsttestidx);
numfirsttest = length(firsttestidx);

secondidx = find(trainlabels == secondnum);
secondlabels = trainlabels(secondidx);
secondmat = allNums(:, secondidx);
numsecond = length(secondlabels);

secondtestidx = find(testlabels == secondnum);
secondtestlabels = testlabels(secondtestidx);
secondtestmat = allTest(:, secondtestidx);
numsecondtest = length(secondtestidx);
```

```
52
53 thirdidx = find(trainlabels == thirdnum);
54 thirdlabels = trainlabels(thirdidx);
55 thirdmat = allNums(:, thirdidx);
56 numthird = length(thirdlabels);
57
58 thirdtestidx = find(testlabels == thirdnum);
59 thirdtestlabels = testlabels(thirdtestidx);
60 thirdtestmat = allTest(:, thirdtestidx);
61 numthirdtest = length(thirdtestidx);
62
63 %% Make total test and train matrices
64
65 totalmat = [firstmat, secondmat, thirdmat];
66 totallabels = [firstlabels', secondlabels', thirdlabels'];
67 totaltestmat = [firsttestmat, secondtestmat, thirdtestmat];
68 totaltestlabels = [firsttestlabels', secondtestlabels', thirdtestlabels'];
69
70 %% Find Singular Values
71 [U,S,V] = svd(totalmat, 'econ');
72 eigvals = diag(S);
73
74 %% Determine number of modes to incorperate
75 energyThreshold = .9;
76 normalized_eigvals = eigvals / sum(eigvals);
77 cumEnergy = cumsum(eigvals)/sum(eigvals);
78 energyIndex = find(cumEnergy >energyThreshold,1);
79 energyIndex = 344; %this was determined from running SVD on all test data,
80                    % comment out this line to capture 90% of energy for
81                    % difference between these two digits specifically
82 %% Project onto determined number of modes
83 U = U(:, 1:energyIndex);
84 sample = U' * totaltestmat;
85 training = U' * totalmat;
86 group = totallabels;
87
88 %% Classify using LDA
89 [class, error] = classify(sample', training', group);
90 %% Assess accuracy of LDA test
91 train_percent_accuracy = 1 - error;
92 numwrong = sum(totaltestlabels ~= class');
93 percent_accuracy = 1 - (numwrong / (length(totaltestlabels)));
94
95
96
97 %% EXTRAS
98 %% Classify using tree
99 tree = fitctree(training', totallabels);
100 class = predict(tree, sample');
101
102 %% Assess accuracy of tree test
103 numwrong = sum(totaltestlabels ~= class');
104 percent_accuracy = 1 - (numwrong / (length(totaltestlabels)));
105
```

```
106 %% Classify using SVM
107 sample = sample / max(eigvals);
108 training = training / max(eigvals);
109 Mdl = fitcecoc(training',totallabels);
110 %%
111 class = predict(Mdl,sample');
112 numwrong = sum(totaltestlabels ~= class');
113 percent_accuracy = 1 - (numwrong / (length(totaltestlabels)));
```

B.4 Ten Digit Classification

```
%% Read in data
close all; clear all;
[trainimages, trainlabels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');

%Make matrix to hold all num info
picheight = 28;
picwidth = 28;
num_rows = picheight * picwidth;
num_cols = length(trainlabels);
allNums = zeros(num_rows, num_cols);


for i = 1:num_cols
    %reshape into col vector, store with each col = different image
    allNums(:,i) = reshape(trainimages(:,:,i), num_rows, 1);
end

%% read in test data
[testimages, testlabels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');
%Make matrix to hold all test num info
test_num_cols = length(testlabels);
allTest = zeros(num_rows, test_num_cols);
for i = 1:test_num_cols
    %reshape into col vector, store with each col = different image
    allTest(:,i) = reshape(testimages(:,:,i), num_rows, 1);
end

%% Make total test and train matrices

totalmat = allNums;
totallabels = trainlabels;
totaltestmat = allTest;
totaltestlabels = testlabels;

%% Find Singular Values
[U,S,V] = svd(totalmat, 'econ');
eigvals = diag(S);

%% Determine number of modes to incorperate
energyThreshold = .9;
normalized_eigvals = eigvals / sum(eigvals);
cumEnergy = cumsum(eigvals)/sum(eigvals);
energyIndex = find(cumEnergy >energyThreshold,1);
energyIndex = 344;

%% Project onto determined number of modes
U = U(:, 1:energyIndex);
sample = U' * totaltestmat;
training = U' * totalmat;
group = totallabels;

```

```
%% Classify using tree
tree = fitctree(training', totallabels);
class = predict(tree, sample');

%% Assess training accuracy of tree train data
trainpercentaccuracytree = (1 - cvloss(tree));

%% Assess accuracy of test tree
numwrong = sum(totaltestlabels ~= class);
tree_percent_accuracy = 100*(1 - (numwrong / (length(totaltestlabels))));

%%% Determine number of modes to incorperate for LDA (fewer than tree)
% energyThreshold = .5;
% normalized_eigvals = eigvals / sum(eigvals);
% cumEnergy = cumsum(eigvals)/sum(eigvals);
% energyIndex = find(cumEnergy >energyThreshold,1);

%% Project onto determined number of modes
U = U(:, 1:energyIndex);
sample = U' * totaltestmat;
training = U' * totalmat;
group = totallabels;
%% Classify using SVM
sample = sample / max(training(:));
training = training / max(training(:));
Mdl = fitcecoc(training',totallabels);

%% Assess accuracy of training SVM
trainaccuracy_percents_SVM = 1 - (loss(Mdl, training', totallabels));

%% Assess accuracy of test SVM
class = predict(Mdl,sample');
numwrong = sum(totaltestlabels ~= class);
SVM_percent_accuracy = 1 - (numwrong / (length(totaltestlabels)));
```

Appendix C Additional Figures

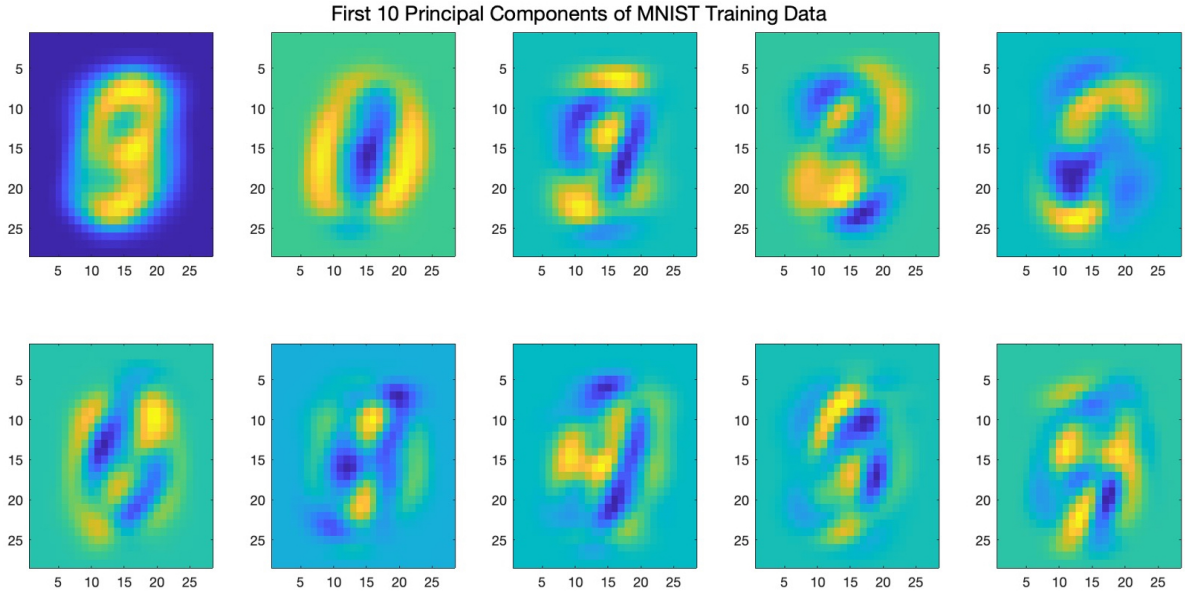

Figure 3: The first 10 columns of the U vector (following the SVD of the training MNIST data) reshaped into 28x28 matrices and displayed. The first row contains principal components 1 through 5 (from top left to top right), and the second row contains principal components 6 thought 10 (from bottom left to bottom right).

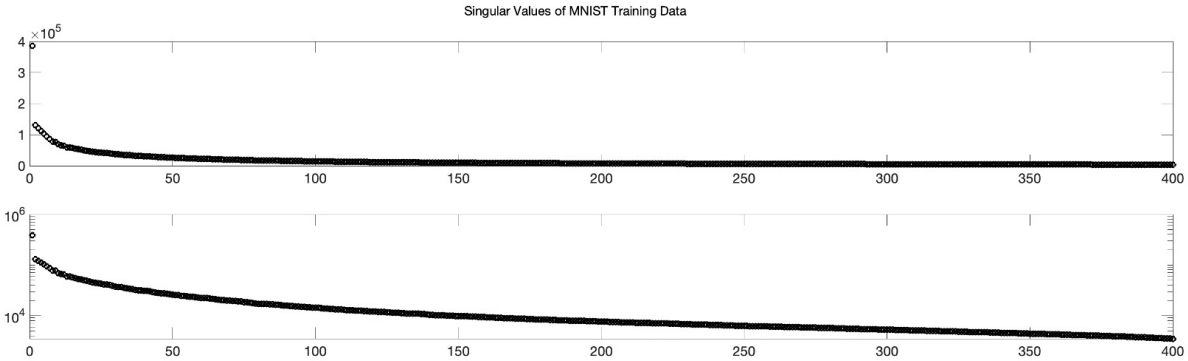

Figure 4: First 400 singular values of MNIST training data (lower plot Y axis is log scale). The first singular value captures much more energy than the rest, but subsequent singular values are not at or near zero, suggesting they also capture a lot of energy of the system.

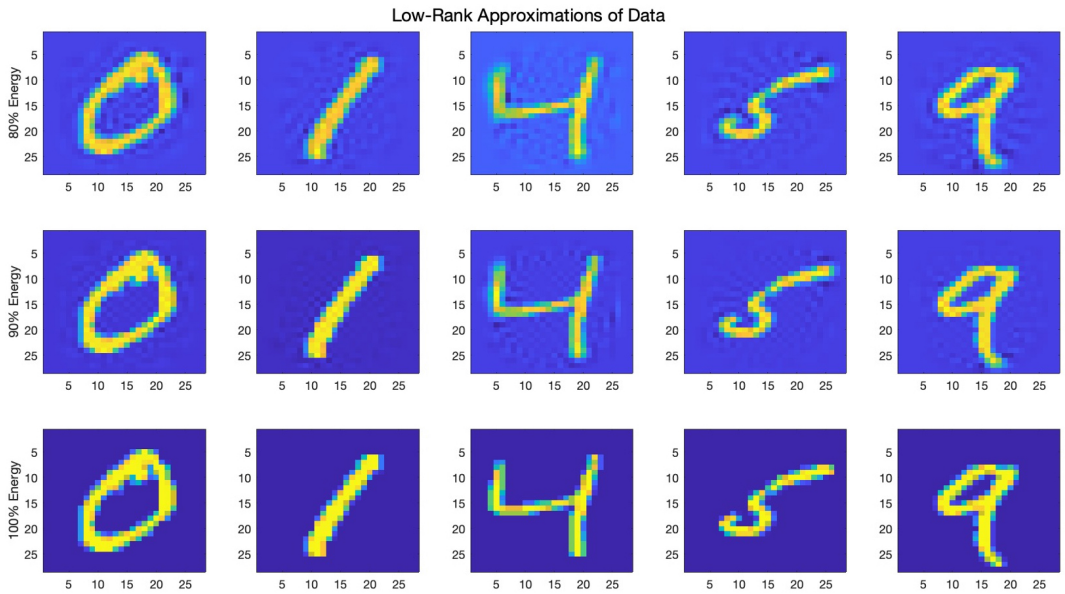

Figure 5: Example digits from low-rank approximations of MNIST training data. The number of modes used throughout this analysis was 344, which was the number of modes necessary to capture 90% of the energy of the full training MNIST dataset. Upon visual inspection 80% was deemed too blurry but some dimensionality reduction was required to improve runtime of analysis algorithms.

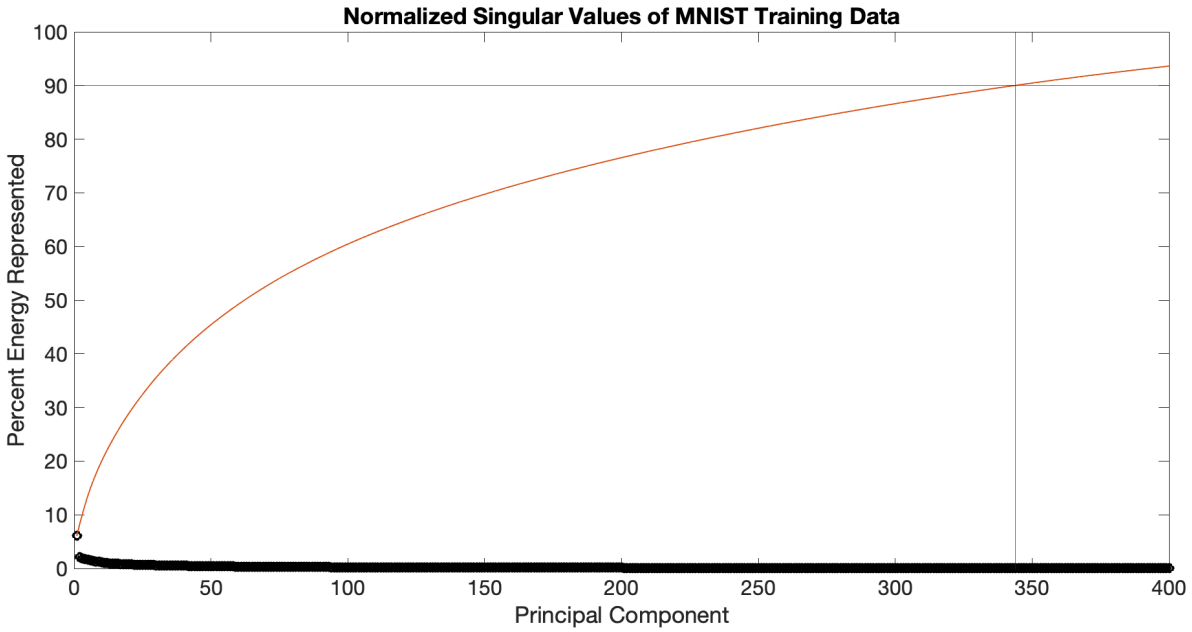

Figure 6: First 400 singular values of MNIST training data normalized to the sum of all singular values. Red line represents cumulative energy. Horizontal line at 90% energy intersects cumulative energy at principal component 344.

LDA Training Data Projections in Best and Worst Cases

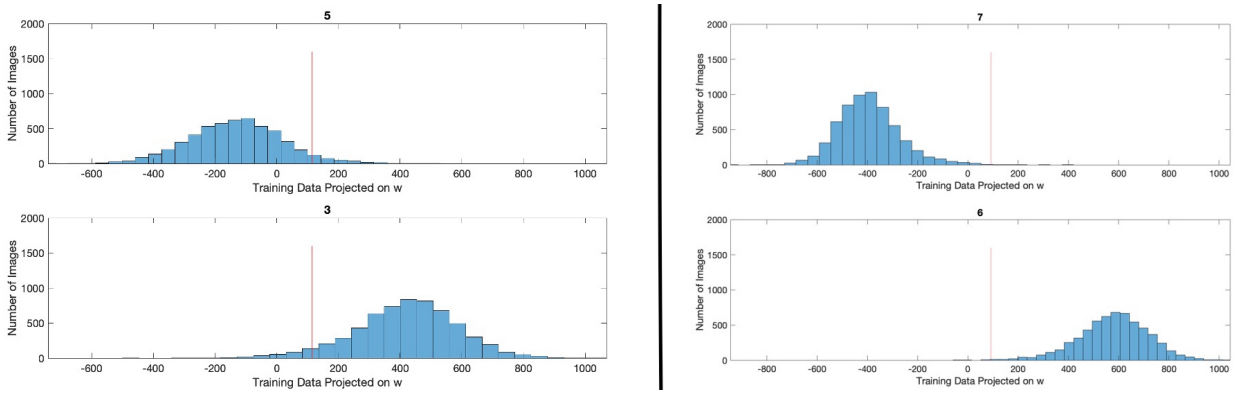

Figure 7: Left histograms are projections onto LDA basis for training 5's (top) and 3's (bottom). This digit pair yielded the worst LDA analysis. Right histograms are projections onto LDA basis for training 7's (top) and 6's (bottom). This digit pair yielded the best LDA analysis. Vertical red lines show the location of the classification threshold. Data in the best case scenario (right) are further from this threshold while data in the worst case scenario (left) cluster near the threshold.