Assignment 4: Classiforniacation

William Ojemann
AMATH 482
March 4th, 2021

Abstract

Machine learning, while a broad subject, is a powerful tool that applies statistical and mathematical reasoning to extrapolate meaning from data. This assignment explores the use of a common type of machine learning, classification, to identify the number in a series of handwritten images. This report analyzes linear discriminate analysis, binary classification trees, and support vector machines and their abilities to accurately tell different written numbers from various scribes apart. This assignment describes just one application of classification, but it explores a field that is the future of medicine, engineering, politics, and almost every other quantifiable interaction.

Introduction

One of the most commonly applied types of machine learning (ML) is classification, which applies mathematical and principals to separate a dataset into distinct groups. Classification itself as a tool has a myriad of uses ranging from deciding whether a picture of an animal is a dog or a cat, to deciding whether or not a criminal should be released on bail. While classification is a powerful tool, using ML and statistics in a way that can have profound impacts on real people requires careful thought and thorough understanding of the fundamentals that underscore how a classification algorithm makes decisions.

This report explores the details of how linear discriminate analysis (LDA), one of the simplest forms of classification, can be used to classify images of handwritten numbers by the number that was written. Not only does linear discriminate analysis find the best way to separate two different classes, but it can also look at new data and figure out which class it belongs to. The following protocol explores an in-depth analysis of both the applications of linear discriminate analysis and the mathematical operations that use singular value decomposition (SVD) and statistics to separate groups of seemingly similar data. While a simple phenomenon in application, LDA requires a strong mathematical background, which will be provided in full in the following report, to fully comprehend and use in a meaningful, responsible way.

Theoretical Background

LDA, developed in the mid 1930's, is a classification method that, like many other data science tools, relies heavily on singular value decomposition. SVD is a decomposition method that decomposes a matrix A into left and right orthogonal matrices, U and V that form an orthogonal basis for the data (1) and singular values S. The vectors in this orthogonal basis are often called the principal components (PCs), and they are organized such that the first PC contains the most variance of the matrix A, the second PC the second most variance, and so on.

$$A = USV^* \text{ or } A = \sum_{k=1}^{r} \sigma_k u_k v_k^* \qquad \text{eq. 1}$$

The process of decomposing a matrix into its right and left singular vectors and its singular values is so important for LDA because it allows for the distillation of the components of a matrix that separate different features.

Because LDA is an incredibly practical and applied algorithm, the theory can be best understood in the context of an example – in this case the classification of low and high pitch sounds. The matrix A (1) in the context of sound analysis will contain each separate n sound recording to be classified as the columns and the rows of the matrix are the individual m samples of each chunk at each point in relative time. Because an LDA algorithm requires training, each of these recordings is already classified as being low pitch or high pitch. Prior to any further data operations, the matrices are often demeaned and normalized across features.

Using the combined recordings from both classes, the A matrix is decomposed using SVD. Here the U matrix, or left singular vectors, represent the principal components of the sound recordings, and the V matrix, or right singular vectors, describes how much each principal component is present in each recording, while the S matrix of singular values weights each principal component according to how much variance it explains. Because LDA aims to identify the optimal one-dimensional hyperplane that best separates the different classes of data, each sample is projected onto the principal components (2) to order the features of each recording by explained variance.

$$U^{*}A = \mathrm{S}\mathrm{V}^{*} \qquad \text{eq. 2}$$

To identify the line that, when the two classes of data are projected onto it, separates between classes and condenses within classes, the within class variances (3) and between class variances (4) are calculated.

$$S_{between} = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \qquad \text{eq. 3}$$

$$S_{within} = \sum_{k=1}^{2} \sum_{x} (x - \mu_k)(x - \mu_k)^T \text{ eq. 4}$$

In the above equations, the variable x represents a sample of data with label *k.* The two variance matrices, also called scatter matrices, represent what we want to maximize, in the between class variance, and what we want to minimize, the within class variance. This objective is achieved by finding a vector, *w*, that represents the line upon which the classes can be optimally separated (5).

$$w = argmax \frac{w^T S^B w}{w^T S_w w} \qquad \text{eq. 5}$$

w can also be identified as the eigenvector with the largest corresponding eigenvalue determined from the eigenvalue problem presented in equation 6.

$$S_B \mathrm{w} = \lambda S_w \mathrm{w} \qquad \text{eq. 6}$$

The data is then projected onto w and these values can be used to determine the proper threshold for separating one class from another. While identifying the optimal threshold may seem like a relatively simple task, because of the variety of applications of a classification algorithm, there are similarly a variety of implications for any given threshold that need to be considered. For the purposes of this assignment the threshold will be determined as the mean value of the first overlapping points from each class.

When classifying between more than two classes, the algorithm is similar except the between and within class variances are calculated with equations 7 and 8 where μ is the average of the features across all samples.

$$S_{between} = \sum_{k=1}^{n} (\mu_k - \mu)(\mu_k - \mu)^T \qquad \text{eq. 3}$$

$$S_{within} = \sum_{k=1}^{n} \sum_{x} (x - \mu_k)(x - \mu_k)^T \text{ eq. 4}$$

While the project mainly focuses on using LDA for classification, there are other signal processing and machine learning tools that are relevant for a foundational understanding of the following work. In image processing specifically, a common tool for creating the best image for classification is convolution. For the purpose of this report, convolution is the application of a kernel, or window, that is multiplied iteratively across the image to alter it in some way. In this assignment and in much of image processing, a Haar wavelet (9) can be applied to extract the edges of an image. This is useful because edges are often the features that both humans and algorithms can most easily recognize.

$$\psi(x) = \begin{cases} 1,\ 0 \le x < \frac{1}{2} \\ -1,\ \frac{1}{2} \le x < 1 \\ 0,\ otherwise \end{cases} \qquad \text{eq. 9}$$

Beyond image processing, the report describes the computational implementation of a support vector machine (SVM) and a binary tree classifier, which are both out of the theoretical scope of the project.

Algorithm Implementation and Development

The followings steps are implemented in MATLAB. Like most machine learning projects, the first step in performing any analysis is to process the data. Using a provided function, the training and testing images of the handwritten numbers were unarchived from the publicly available image files. The images were then further manipulated to be taken out of a three-dimensional m x n x samples image format to a m*n x samples features and samples matrix for processing.

With the newly organized matrix, the economical SVD was calculated, and the singular values plotted and analyzed to identify the number of singular values needed to properly represent the data. To further analyze the data and decomposition, the data was projected onto three of the 784 possible principal components and then plotted using the right side of equation two. After analyzing the SVD, a Haar wavelet transform was applied to the data, but the resulting convolved features performed worse in the classification algorithms so no further analysis of the transforms will be discussed. It is normally standard to standardize the data across samples before implementing LDA, but in the case of this assignment, no standardization was applied because of sufficient classification without standard data.

The initial implementation of LDA was to classify between drawings of zeros and sevens, which were chosen based on intuition and used 40 principal components – this number is based on later analysis. The LDA algorithm was implemented by hand using the mathematical theory outlined in the theoretical background section. Test samples were classified by projecting onto the training data's principal components and then the classification vector. The threshold for separating the two classes based on their values on the classification axis was determined by iteratively comparing points until the classes overlapped. The average of those two class values was set to be the threshold. To understand the performance of the LDA algorithm, a confusion matrix for the predicted classes of the training data and the predicted classes of the testing data were created using the built in MATLAB confusion matrix plotting function. Validation of the optimal number of principal components used for classification was performed by iteratively classifying between drawn zeros and sevens with increasing numbers of modes.

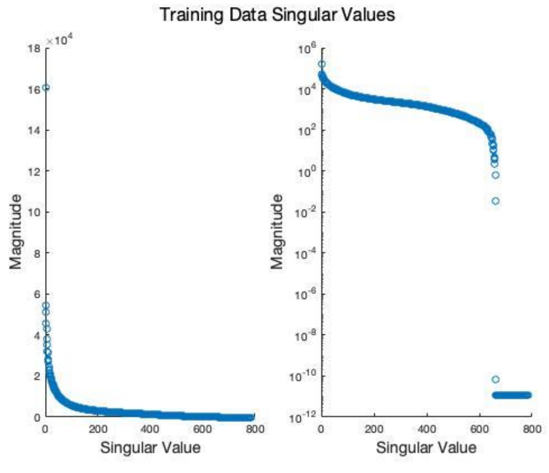
Because LDA has differing levels of success classifying between different written numbers, classification was performed comparing each of the digits from 0 to 9 and tested on the testing data. The results of that comparison were quantified by creating confusion matrices and using the average correct prediction probability as the score for that classification. LDA can also be used for multivariate classification, and this was implemented manually following the alternate steps provided in the theoretical background section. To analyze the results of this classification, histograms with the distributions of projected values were created for each class.

Exploring other classification methods started with implementing an SVM using MATLAB's 'fitecoc' function to generate an algorithm trained on all ten different numbers to classify any one of them. A confusion matrix was used to analyze the results of this multivariate classification. The next algorithm used to explore classification was a binary tree classifier. Because a binary tree can have a variable number of decision points, or splits, the optimal number of splits was plotted and evaluated using both cross-validated k-fold loss and correct prediction probability, which was again the mean of the correct prediction probabilities in the confusion matrix. Both an SVM and a binary tree were compared against LDA on the two digits that LDA could most easily classify and the two digits that LDA could least easily classify.

Computational Results

The results of the SVD decomposition of the data show that, while the first principal component contains the majority of the variance, there is still potentially non-negligible information contained in the first ~650 principal components (**figure 1**).

Figure 1: *Singular values extracted from the training dataset*



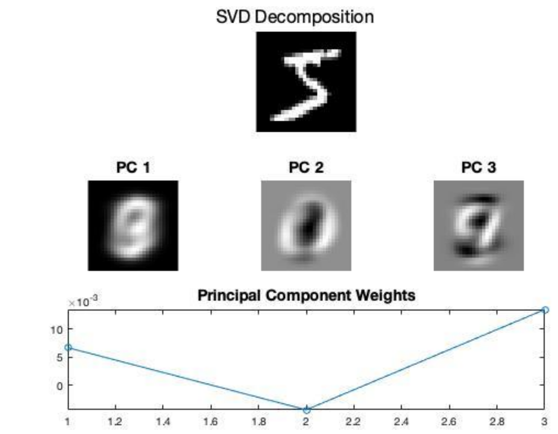
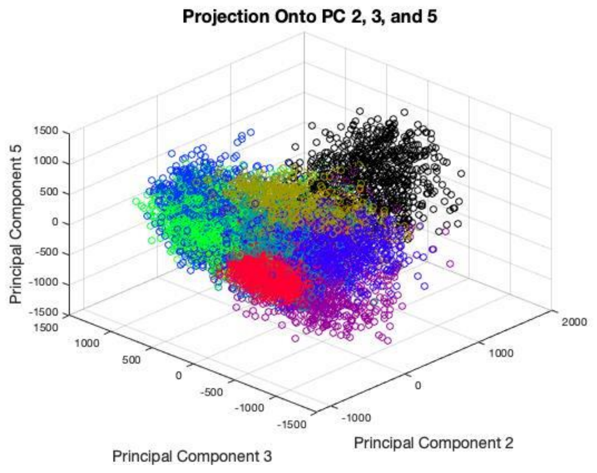


Figure 2: *Analysis of the SVD decomposition of the first sample, a hand-drawn five (top). (middle) first three principal components. (bottom) weight of the first three principal components on this sample.*

Further exploring the SVD through an analysis of how the first sample was represented in the principal component space helped show what parts of the written numbers were relevant for classification (**figure 2**). A more comprehensive collection of principal components is available in **Appendix B**. Diving deeper into the projections of the data onto three principal components reveals how the different numbers lie along the principal components and how they generally tend to clump (**figure 3**).

Figure 3: *Test data projected onto the second, third, and fifth principal components.*



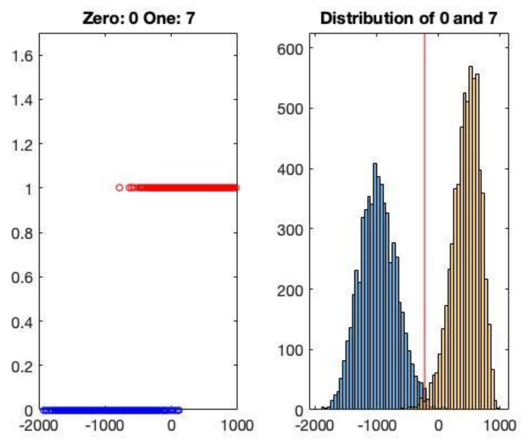
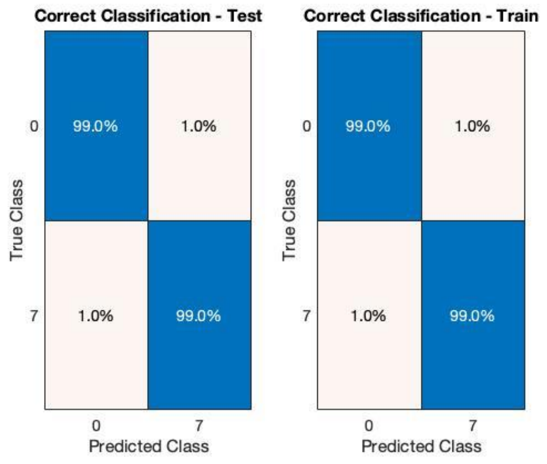


Figure 4: *(left) Samples classified as zeros and sevens projected onto the LDA vector. (right) distribution of zeros and sevens along the LDA vector with calculated threshold.*

The result of the first LDA algorithm with 40 principal components classifying between drawn zeros and sevens (**figure 4**) demonstrate a qualitatively successful algorithm. The classification score is quantitatively represented using a confusion matrix that shows the percent of the predictions that matched the true classification (**figure 5**). The confusion matrix shows how the classification algorithm performed highly on both training and testing sets.

Figure 5: *(left) Confusion matrix showing classification accuracy for each grouping in both the training and testing datasets.*



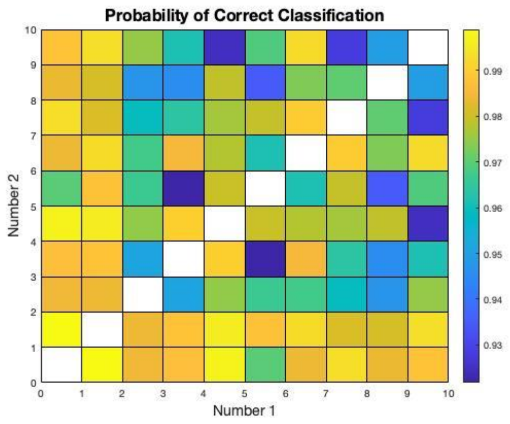
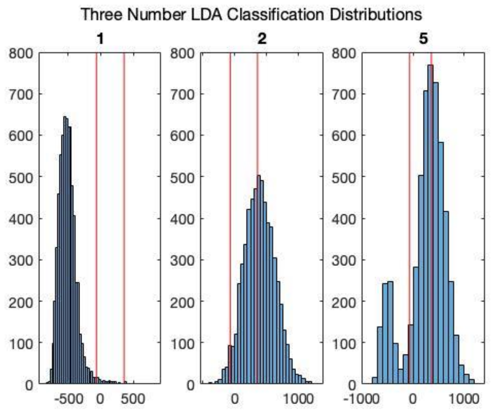


Figure 6: *Score of classification between two numbers. Left index corresponds to each box.*

To identify the optimal number of SVD modes to use for LDA, the LDA was again performed on the numbers zero and seven iteratively with increasing numbers of SVD modes (**Appendix B**). The optimal number of SVD modes was identified as 40 with an accuracy of 99.5% after an initial large jump from one to two modes from ~78% to ~97% accuracy. When two digit classification was applied to every combination of numbers (**figure 6**) it revealed that 0 and 1 were the easiest digits to classify and 3 and 5 were the hardest to classify. Extrapolating the LDA algorithm to classify between three different classes and plotting the resulting class distributions demonstrated the relative difficulty the algorithm had with this task because of large class overlaps (**figure 7**).

Figure 7: *Distributions of three different classes using three group LDA*



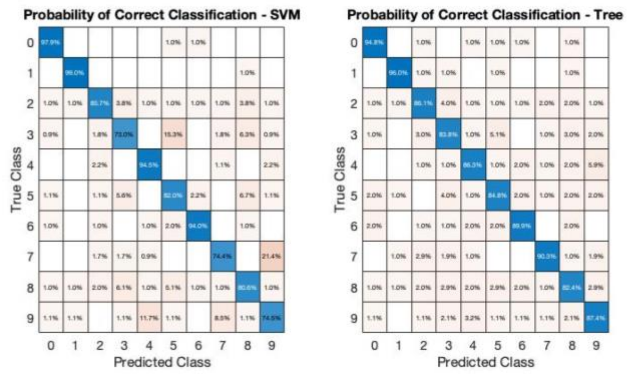


Figure 8: *confusion matrices for 10 class SVM and binary classification tree.*

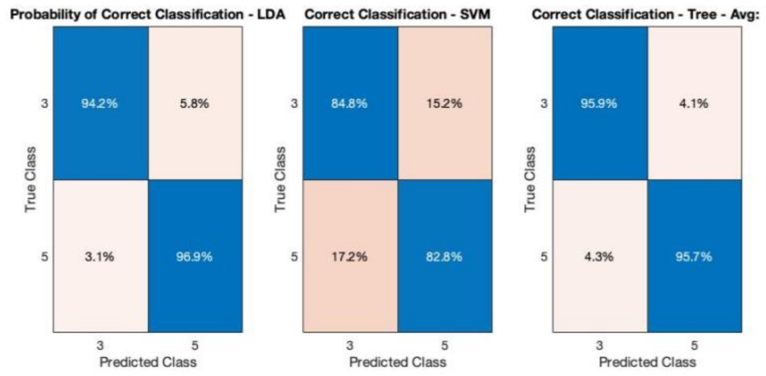
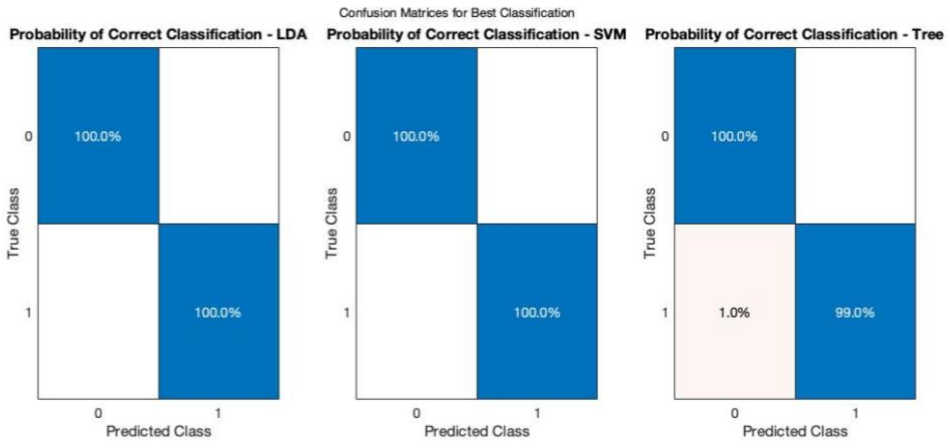


Figure 10: *Confusion matrices for LDA, SVM, and Binary Tree comparing three and five.*

When exploring other classification methods and implementing ten class SVMs and Binary trees, the resulting confusion matrices (**figure 8**) show that while both models were highly successful, the SVM had the highest individual number classification while the binary tree was the most consistent classifier. The number of splits in the tree – 1300 – is optimized according to **Appendix B**. The results of comparing a SVM, binary tree, and LDA classifier for the easiest numbers to discriminate (**figure 9**) shows that there is a negligible difference between the two algorithms. When comparing the results from the three algorithms for classifying the most challenging numbers, the LDA model and binary tree performed similarly while the SVM struggled to successfully separate the two classes with a high level of precision (**Figure 10**).

Figure 9: *Confusion matrices for LDA, SVM, and Binary Tree comparing zero and one.*



Conclusion and Summary

ML classification is a powerful tool, and when applied judiciously with a sufficient mathematical background it can be applied to successfully complete a complex task like identifying human handwriting. While LDA is very useful for discriminating between two distinct classes – that can be separated with one threshold – there are a myriad of increasingly complex classification algorithms that can accurately classify a large number of groups. ML is not only useful for this specific example of image analysis, but it can be used in every field and, for better or for worse, is pushing the cutting edge of research in almost every subject.

Appendix A – MATLAB Functions

x = cell(m,n) – Initializes an empty m x n cell, used to initialize all storage variables for the different tests and datum with varying sample numbers.
x = ones(m,n) – Initializes a m x n matrix of ones, used to initialize matrices.
x = zeros(m,n) – Initializes a m x n matrix of zeros, used to initialize matrices
plot(t,y) – plots the signal y against its time domain t, used to plot the location of the object using frames as a unit for time.
scatter3(x,y,z,[],c) – plots the individual data points as dots, used to plot the singular values, and projected data points on principal components with different colors for each different number.
round(x) – rounds the value x to the nearest integer, used to ensure values used as indices were integers.
histogram(x) – plots the distribution of the data x as a histogram, used to plot the distributions of data on the LDA vector.
linspace(start, end, int) – creates an array of variables between the start and end value with int data points, used to create axis ticks for plotting.
[U,S,V] = svd(x, 'econ') – returns the economical singular value decomposition of the matrix x.
mvlda – Internal function used to perform LDA using the math outlined in the theoretical background section when classifying between more than two different classes.
two_feature_lda – Internal function used to perform LDA using the math outlined in the theoretical background section when classifying between two different classes.
lda_predict – Internal function used to predict the class of features based on a previously trained LDA model for classifying two different groups.
make_confusion_matrix – Internal function used to create confusion matrices based on the true labels and the predicted labels.
num_wave – Internal function used to apply a haar wavelet transform
initial_processing(images,labels) – Internal function used to identify the labels and features for each different class and transform images from a pixelated image into a linear representation.

Appendix B – Additional Plots

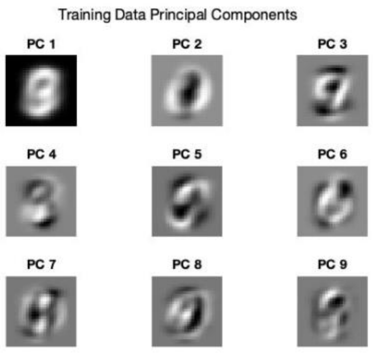


Figure 1: First 9 principal components of the training data

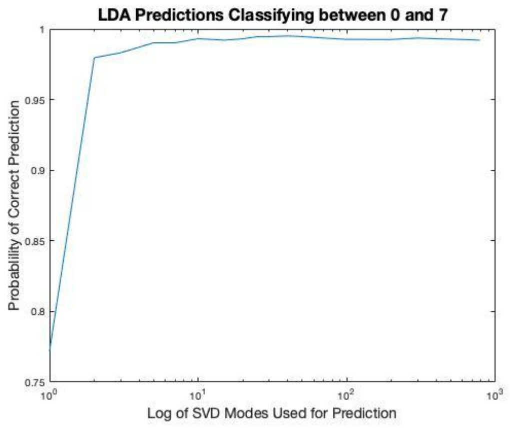


Figure 2: Prediction accuracy curve showing prediction accuracy for increasing numbers of SVD modes used in LDA.

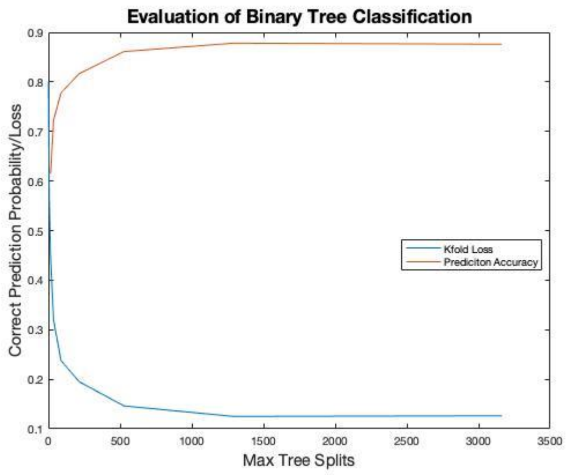


Figure 3: Prediction accuracy and cross-validated kfold loss for increasing number of maximum tree splits in the training of a binary classification tree. The ideal value was identified as 1300 splits.

Appendix C – MATLAB Code

```
% William Ojemann
% AMATH 482
% Assignment 4
% 27 FEB 2021
clear; close all; clc;
%% Data Loading and Processing
[images_train, labels_train] = mnist_parse('train-images.idx3-ubyte', 'train-
labels.idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images.idx3-ubyte', 't10k-
labels.idx1-ubyte');
[num_indices_train,features_train] =
initial_processing(images_train,labels_train);
[num_indices_test,features_test] =
initial_processing(images_test,labels_test);
%% SVD
[Utr,Str,Vtr] = svd(features_test,'econ');
m = size(images_test,1);
n = size(images_test,2);
figure
subplot(1,2,2)
scatter(1:length(diag(Str)),diag(Str))
set(gca,'yscale','log')
ylabel('Magnitude','Fontsize',14)
xlabel('Singular Value','Fontsize',14)
subplot(1,2,1)
scatter(1:length(diag(Str)),diag(Str))
ylabel('Magnitude','Fontsize',14)
xlabel('Singular Value','Fontsize',14)
sgtitle('Training Data Singular Values','Fontsize',16)
figure
for k = 1:9
    subplot(3,3,k)
    X = reshape(Utr(:,k),m,n);
    imshow(rescale(X))
    title(['PC ', num2str(k)],'Fontsize',14)
end
sgtitle('Training Data Principal Components','Fontsize',16)
%% Projection Color Scheme
colors = [0 0 0];
grad = linspace(0.2,1,3)';
gradalt = linspace(1,.2,3)';
temp = [0; 0; 0];
onethree = [gradalt temp grad];
foursix = [grad gradalt temp];
sevnine = [temp grad gradalt];
colors = [colors;
          onethree;
          foursix;
          sevnine];
colormaps = zeros(length(labels_test),3);
for i = 1:length(labels_test)
    label = labels_test(i);
    colormaps(i,:) = colors(label+1,:);
end
%% PC Projection Plotting
```

```
cols = [2,3,5];
scaled_modes = (Str*Vtr')';
projs = scaled_modes(:,cols);
figure
scatter3(projs(:,1),projs(:,2),projs(:,3),[],colormaps)
xlabel(['Principal Component ',num2str(cols(1))],'Fontsize',14)
ylabel(['Principal Component ',num2str(cols(2))],'Fontsize',14)
zlabel(['Principal Component ',num2str(cols(3))],'Fontsize',14)
title(['Projection Onto PC ',num2str(cols(1)),', ',num2str(cols(2)),...
    ', and ',num2str(cols(3))],'Fontsize',16)
%% Wavelet and SVD
% % X = num_wave(images_train);
% load('images_train_wave.mat')
% [mw,nw,~] = size(X);
% features_w = double(reshape(X,mw*nw,size(X,3)));
% [uw,sw,vw] = svd(features_w,'econ');
%% Wavelet and SVD plotting
% figure
% for k = 1:9
%     subplot(3,3,k)
%     X = reshape(uw(:,k),mw,nw);
%     imshow(rescale(X))
% end
%% One Sample Analysis
figure
subplot(3,3,1:3)
imshow(images_train(:,:,1))
for i = 1:3
    subplot(3,3,i+3)
    X = reshape(Utr(:,i),m,n);
    imshow(rescale(X))
    title(['PC ', num2str(i)],'Fontsize',14)
end
subplot(3,3,7:9)
Y = Vtr(1,1:3);
plot(Y,'-o')
title('Principal Component Weights','Fontsize',14)
sgtitle('SVD Decomposition','Fontsize',16)
%% Rank Evaluation
nums = [0,7];
ranks = [1 2 3 5 7 10 15 20 25 30 40 50 70 100 200 300 400 600 784];
%ranks = [2];
score = zeros(1,length(ranks));
features = features_train;
num_indices = num_indices_train;
for r = 1:length(ranks)
    num_components = ranks(r);
[alpha,beta,w,threshold,U] = two_feature_lda(nums(1),nums(2),features,...
                        num_indices,num_components);
    [labels,labels_pred] = lda_predict(nums(1),nums(2),num_indices_test,...
                            features_test,U,w,threshold);
    con_mat_test = make_confusion_matrix(labels,labels_pred,0);
    score(r) = mean(diag(con_mat_test));
end
%% Rank Evaluation Plotting
nums = [0,7];
ranks = [1 2 3 5 7 10 15 20 25 30 40 50 70 100 200 300 400 600 784];
```

```
load('rank_analysis.mat')
figure
semilogx(ranks,score)
xlabel('Log of SVD Modes Used for Prediction','Fontsize', 14)
ylabel('Probablility of Correct Prediction','Fontsize',14)
title(['LDA Predictions Classifying between ',num2str(nums(1)),' and
',num2str(nums(2))],'Fontsize',16)
%% Two Features - Prep
num1 = 0;
num2 = 7;
num_components = 10;
[vone,vtwo,w,threshold,U] =
two_feature_lda(num1,num2,features_train,num_indices_train,num_components);
figure(100)
subplot(1,2,1)
plot(vone,zeros(length(vone)),'ob')
hold on
plot(vtwo,ones(length(vtwo)),'or')
title(['Zero: ', num2str(num1), ' One: ', num2str(num2)])
set(gca,'Fontsize',14)
ylim([0 1.7]);
%%
subplot(1,2,2)
histogram(vone);
hold on, plot([threshold threshold], [0 650],'r')
set(gca,'Ylim',[0 625],'Fontsize',14)
subplot(1,2,2)
histogram(vtwo); %hold on, plot([threshold threshold], [0 650],'r')
title(['Distribution of ' num2str(num1) ' and ' num2str(num2)])
set(gca,'Ylim',[0 625],'Fontsize',14)
%% Confusion Matrices
% Confusion Matrix - Test
[labels,labels_pred] =
lda_predict(num1,num2,num_indices_test,features_test,U,w,threshold);
con_mat_test = make_confusion_matrix(labels,labels_pred);
figure
subplot(1,2,1)
cm = confusionchart(round(con_mat_test*100),[num1 num2]);
cm.Title = ['Correct Classification - Test'];
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
% Confusion Matrix - Train
[labels,labels_pred] =
lda_predict(num1,num2,num_indices_train,features_train,U,w,threshold);
con_mat_test = make_confusion_matrix(labels,labels_pred);
subplot(1,2,2)
cm = confusionchart(round(con_mat_test*100),[num1 num2]);
cm.Title = ['Correct Classification - Train'];
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
%% Digit Classification
% digits = 3:9;
% score = nan(length(digits));
% for digit = digits
%     for digit2 = digits
%         if digit == digit2
%             continue
%         end
%         num1 = digit;
%         num2 = digit2;
%         num_components = 10;
%         [vone,vtwo,w,threshold,U] =
two_feature_lda(num1,num2,features_train,num_indices_train,num_components);
%         [labels,labels_pred] =
lda_predict(num1,num2,num_indices_test,features_test,U,w,threshold);
%         con_mat_test = make_confusion_matrix(labels,labels_pred);
%         if digit == 3 && digit2 == 5
%             figure
%             confusionchart(round(con_mat_test*100))
%         end
%         score(digit+1,digit2+1) = mean(diag(con_mat_test));
%         disp([num1 num2 score(digit+1,digit2+1)])
%     end
% end
%%
clear score;
load 'two_digit_scores.mat'
score = [score; nan(1,length(score))];
score = [score nan(length(score),1)];
figure
pcolor(0:10,0:10,score)
colorbar
title('Probability of Correct Classification','Fontsize',16)
xlabel('Number 1','Fontsize', 14)
ylabel('Number 2','Fontsize',14)
[maxprob,maxidx] = max(score,[],'all','linear');
[X,Y] = ind2sub(size(score),maxidx);
%% MATLAB Multivariate Classification
num1 = 2; num2 = 1; num3 = 5;
nums = sort([num1 num2 num3]);
features = features_train;
num_indices = num_indices_train;
num_components = 20;
[values, w, ~, U] = mvlda(nums,features,...
                        num_indices,num_components);
meds = zeros(1,length(values));
for i = 1:length(values)
    meds(i) = median(values{i});
end
threshold = [(meds(1)+meds(2))/2,(meds(2)+meds(3))/2];
figure(101)
subplot(1,3,1)
histogram(values{1});
hold on, plot([threshold(1) threshold(1)], [0 800],'r')
plot([threshold(2) threshold(2)], [0 800],'r')
set(gca,'Fontsize',14);
title(num2str(nums(1)))
subplot(1,3,2)
histogram(values{2}); hold on, plot([threshold(1) threshold(1)], [0 800],'r')
plot([threshold(2) threshold(2)], [0 800],'r')
set(gca,'Fontsize',14);
title(num2str(nums(2)))
subplot(1,3,3)
histogram(values{3}); hold on, plot([threshold(1) threshold(1)], [0 800],'r')
```

```
plot([threshold(2) threshold(2)], [0 800],'r')
set(gca,'Fontsize',14);
title(num2str(nums(3)))
sgtitle('Three Number LDA Classification Distributions','Fontsize',16)
%% SVM
% % model = fitcecoc(features_train',labels_train);
load('svm_model.mat')
labels_pred = predict(model,features_test');
con_mat_test = make_confusion_matrix(labels_test,labels_pred,1);
figure
subplot(1,2,1)
cm = confusionchart(round(con_mat_test*100),0:9);
cm.Title = 'Probability of Correct Classification - SVM';
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
disp(['Average Score: ' num2str(mean(diag(con_mat_test)))])
%% Binary Tree Optimization
% clc
% splits = round(logspace(0,3.5,10));
% classErrors = zeros(2,length(splits));
% for i = 1:length(splits)
%     treecval=fitctree([features_train features_test]',[labels_train;
labels_test],'MaxNumSplits',splits(i),'Crossval','on');
%     tree =
fitctree([features_train]',[labels_train],'MaxNumSplits',splits(i));
%     %view(tree.Trained{1},'Mode','graph');
%     labels_pred_iter = predict(tree,features_test');
%     con_mat_iter = make_confusion_matrix(labels_test,labels_pred_iter,1);
%     classErrors(1,i) = kfoldLoss(treecval);
%     classErrors(2,i) = mean(diag(con_mat_iter));
%     disp(i);
% end
%% Optimization Plotting
load('binary_tree_results.mat')
splits = round(logspace(0,3.5,10));
figure
plot(splits,classErrors)
title('Evaluation of Binary Tree Classification','Fontsize',16)
legend('Kfold Loss','Prediciton Accuracy')
xlabel('Max Tree Splits','Fontsize',14)
ylabel('Correct Prediction Probability/Loss','Fontsize',14)
%% Optimized tree
tree = fitctree([features_train]',[labels_train],'MaxNumSplits',1300);
labels_pred_iter = predict(tree,features_test');
con_mat_iter = make_confusion_matrix(labels_test,labels_pred_iter,1);
subplot(1,2,2)
cm = confusionchart(round(con_mat_iter*100),0:9);
cm.Title = 'Probability of Correct Classification - Tree';
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
%% Best classification
num1 = 0;
num2 = 1;
num_components = 40;
[~,~,w,threshold,U] =
two_feature_lda(num1,num2,features_train,num_indices_train,num_components);
```

```
[labels,labels_pred] =
lda_predict(num1,num2,num_indices_test,features_test,U,w,threshold);
con_mat_test_lda = make_confusion_matrix(labels,labels_pred);
subplot(1,3,1)
cm = confusionchart(round(con_mat_test_lda*100),[0 1]);
cm.Title = 'Probability of Correct Classification - LDA';
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
disp('Performing SVM')
num1 = num1+1;
num2 = num2+1;
trainidxs = [num_indices_train{num1}; num_indices_train{num2}];
testidxs = [num_indices_test{num1}; num_indices_test{num2}];
features_train_1 = features_train(:,trainidxs); labels_train_1 =
labels_train(trainidxs);
features_test_1 = features_test(:,testidxs); labels_test_1 =
labels_test(testidxs);
model = fitcecoc(features_train_1',labels_train_1);
labels_pred = predict(model,features_test_1');
con_mat_test_svm = make_confusion_matrix(labels_test_1,labels_pred,1);
subplot(1,3,2)
cm = confusionchart(round(con_mat_test_svm*100),0:1);
cm.Title = 'Probability of Correct Classification - SVM';
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
disp('Performing Binary Tree')
tree = fitctree([features_train_1]',[labels_train_1],'MaxNumSplits',1300);
labels_pred_iter = predict(tree,features_test_1');
con_mat_iter = make_confusion_matrix(labels_test_1,labels_pred_iter,1);
subplot(1,3,3)
cm = confusionchart(round(con_mat_iter*100),0:1);
cm.Title = 'Probability of Correct Classification - Tree';
sgtitle('Confusion Matrices for Best Classification')
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
%% Worst Classification
num1 = 3;
num2 = 5;
num_components = 40;
[vone,vtwo,w,threshold,U] =
two_feature_lda(num1,num2,features_train,num_indices_train,num_components);
[labels,labels_pred] =
lda_predict(num1,num2,num_indices_test,features_test,U,w,threshold);
con_mat_test_lda = make_confusion_matrix(labels,labels_pred);
subplot(1,3,1)
cm = confusionchart(round(con_mat_test_lda'*100),[num1 num2]);
cm.Title = 'Probability of Correct Classification - LDA';
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
disp('Performing SVM')
num1 = num1+1;
num2 = num2+1;
trainidxs = [num_indices_train{num1}; num_indices_train{num2}];
testidxs = [num_indices_test{num1}; num_indices_test{num2}];
features_train_1 = features_train(:,trainidxs); labels_train_1 =
labels_train(trainidxs);
```

```
features_test_1 = features_test(:,testidxs); labels_test_1 =
labels_test(testidxs);
% model = fitcecoc(features_train_1',labels_train_1);
load('svm_model_twofeature.mat')
labels_pred = predict(model,features_test_1');
% con_mat_test_svm = make_confusion_matrix(labels_test_1,labels_pred);
subplot(1,3,2)
cm =
confusionchart(labels_test_1,labels_pred);%round(con_mat_test_svm*100),0:1);
cm.Title = 'Correct Classification - SVM';
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
disp('Performing Binary Tree')
tree = fitctree([features_train_1]',[labels_train_1],'MaxNumSplits',1300);
labels_pred_iter = predict(tree,features_test_1');
%con_mat_iter = make_confusion_matrix(labels_test_1,labels_pred_iter);
subplot(1,3,3)
cm =
confusionchart(labels_test_1,labels_pred_iter);%round(con_mat_iter*100),0:1);
cm.Title = ['Correct Classification - Tree - Avg: '];
cm.Normalization = 'row-normalized';
set(gca,'Fontsize',14)
%% Functions

% Multivariate Linear Discriminate Analysis
function [values, w, threshold, U] = mvlda(nums,features,...
                        num_indices,num_components)

    l = length(nums);
    values = cell(1,l);
    featuresets = [];
    featuresets_svd = cell(1,l);
    lengths = zeros(1,l+1);
    for i = 1:l
        num = nums(i) + 1;
        featuresets = [featuresets features(:,num_indices{num})];
        lengths(i+1) = length(features(:,num_indices{num}));
    end
    [U,S,V] = svd(featuresets,'econ');
    U = U(:,1:num_components);

    featuressvd =
S(1:num_components,1:num_components)*V(:,1:num_components)';
    means = nan(size(featuressvd,1),l);
    for i = 1:l
        featuresets_svd{i} =
featuressvd(:,lengths(i)+1:lengths(i)+lengths(i+1));
        means(:,i) =
mean(featuressvd(:,lengths(i)+1:lengths(i)+lengths(i+1)),2);
    end

    meantot = mean(means,2);
    Sw = 0;
    for i = 1:l
        for j = 1:lengths(i+1)
```

```
            Sw = Sw + (featuresets_svd{i}(:,j)-
means(:,i))*(featuresets_svd{i}(:,j)-means(:,i))';
        end
    end
    Sb = 0;
    for k = 1:l
        Sb = Sb + (means(:,k) - meantot)*(means(:,k) - meantot)';
    end
    [V2,D] = eig(Sb,Sw);
    [~, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    for i = 1:l
        values{i} = w'*featuresets_svd{i};
    end
    if l == 2
        i1 = length(values{1});
        i2 = 1;
        while values{1}(i1) > values{2}(i2)
           i1 = i1-1; i2 = i2 + 1;
        end
        threshold = (values{1}(i1) + values{2}(i2))/2;
    else
        threshold = 0;
    end
end

% Perform LDA on Two Class Classification
function [alpha,beta,w,threshold,U] =
two_feature_lda(num1,num2,features_train,...
                        num_indices_train,num_components)
    i1 = num1+1;
    i2 = num2+1;
    oneft = features_train(:,num_indices_train{i1});
    l1 = length(oneft);
    twoft = features_train(:,num_indices_train{i2});
    l2 = length(twoft);
    features12 = [oneft twoft];
    [U,S12,V12] = svd(features12,'econ');
    U = U(:,1:num_components);
    featuressvd = S12*V12';
    sing1 = featuressvd(1:num_components,1:l1);
    sing2 = featuressvd(1:num_components,l1+1:end);
    mean1 = mean(sing1,2); mean2 = mean(sing2,2);
    Sw = 0;
    for i = 1:l1
        Sw = Sw + (sing1(:,i) - mean1)*(sing1(:,i)-mean1)';
    end
    for i = 1:l2
        Sw = Sw + (sing2(:,i) - mean2)*(sing2(:,i)-mean2)';
    end
    Sb = (mean1-mean2)*(mean1-mean2)';
    [V2,D] = eig(Sb,Sw);
    [~, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    vone = w'*sing1;
    vtwo = w'*sing2;
    if mean(vone) > mean(vtwo)
        w = -w;
        vone = -vone;
        vtwo = -vtwo;
    end
    alpha = sort(vone);
    beta = sort(vtwo);

    % Thresholding
    i1 = length(alpha);
    i2 = 1;
    while alpha(i1) > beta(i2)
       i1 = i1-1; i2 = i2 + 1;
    end
    threshold = (alpha(i1) + beta(i2))/2;
end

% LDA Predict
function [labels,labels_pred] =
lda_predict(num1,num2,num_indices,features,U,w,threshold)
    % Only works for two variable LDA
    oneftest = features(:,num_indices{num1+1});
    l1 = length(oneftest);
    twoftest = features(:,num_indices{num2+1});
    l2 = length(twoftest);
    labels1 = ones(1,l1);
    labels2 = ones(1,l2)*2;
    labels = [labels1 labels2];
    features_num = [oneftest twoftest];
    features_pca = U'*features_num;
    pred_val = w'*features_pca;
    labels_pred = (pred_val >= threshold) + 1;
    end

% Make Confusion matrix
function con_mat = make_confusion_matrix(labels,labels_pred,check)
    if nargin<3
        check = 0;
    end
    label_list = unique(labels);
    num_categories = length(label_list);
    con_mat = zeros(num_categories);
    for j = 1:length(labels)
        con_mat(labels(j)+check,labels_pred(j)+check) =...
            con_mat(labels(j)+check,labels_pred(j)+check) + 1;
    end
    for n = label_list
        M = sum(con_mat(n+check,:));
        con_mat(n+check,:) = con_mat(n+check,:)./M;
    end
end

% Apply Haar Wavelet Transform
function wavedata = num_wave(imdata)
    [m,n,p] = size(imdata);
    wavedata = zeros(m/2,n/2,p);
    for k = 1:size(imdata,3)
        X = imdata(:,:,k);
        [~,h,v,~] = dwt2(X,'haar');
        wavedata(:,:,k) = rescale(abs(h)) + rescale(abs(v));
    end
end

% Intial Formatting of Data
function [indices, features] = initial_processing(images,labels)
    m = size(images,1); n = size(images,2);
    features = double(reshape(images,m*n,size(images,3)));
    indices = cell(1,length(unique(labels)));
    for i = 1:length(unique(labels))
        indices{i} = find(labels == i-1);
    end
end
```