AMATH 482 Homework 4

Rishabh Verma

March 10th, 2021

Abstract

Principal component analysis is used to reduce dimension and mine features from the MNIST database of handwritten digits. I then compare how three different classifiers perform when given these labeled features, i.e. linear discriminant analysis (LDA), decision tree classification, and support vector machines.

1 Introduction and Overview

The MNIST dataset contains labeled scans of handwritten digits. Each scan is a 28x28 pixel image, and each pixel is an 8-bit greyscale value. Each image contains a size-normalized centered digit. There are 60,000 scans in the training set and 10,000 scans in the testing set.

The first goal of this paper is to mine the training set for features which vary across the dataset and correspond with the digits. The second goal is to use these features to perform supervised digit classification.

2 Theoretical Background

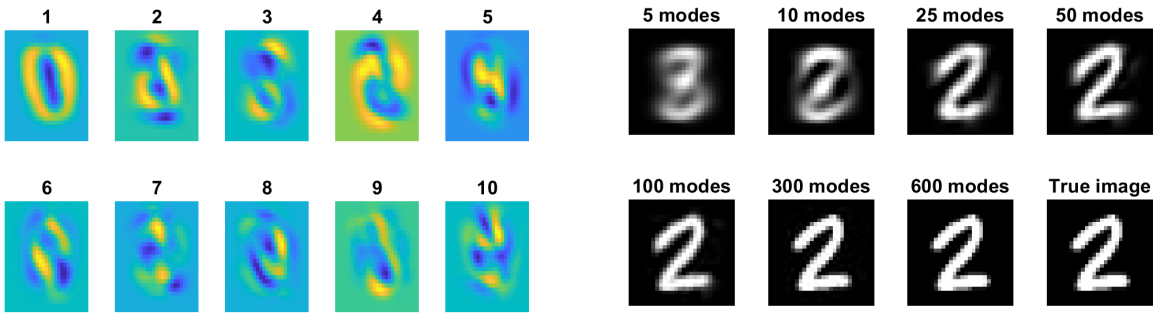
2.1 Dimensionality reduction and PCA

PCA is a powerful tool for dimensionality reduction. Consider a series of n observations represented as column vectors $\vec{x_i}$ with m rows. The $m \times n$ matrix $X = \begin{bmatrix} \vec{x_1} & \vec{x_2} & ... & \vec{x_n} \end{bmatrix}$ can be formed.

Let the mean column be $\vec{x}_\mu$. By computing $X' = X - \vec{x}_\mu \vec{1}^\top$, each column is subtracted from the mean. The resulting matrix has total mean zero, and so we can begin working with variance. Consider the singular value decomposition $X' = U\Sigma V^\top$. The left singular matrix, U, is $m \times m$, and contains a basis for the data-space. These basis vectors are the principal components, and are arranged in descending order of variance. The most significant basis vectors are the left-most columns of U, and the data can be projected into the U-basis via left-multiplication by $U^\top$. Note that this is equivalent to $\Sigma V^\top$. I conclude that the matrix $U^\top = \Sigma V^\top$ will contain in the ij-th entry the projection coefficient of the i-th data point along the j-th principal component.

The process of dimensionality reduction follows from picking a number of dimensions and selecting the corresponding number of singular vectors from the left-most columns of U. How do you choose right number of dimensions? One approach comes from the fact that each singular value is directly proportional to the variance along the corresponding singular vector, and so if the singular values drop suddenly in magnitude, the following dimensions do not characterize much of the variance and can be discarded. Another approach is visual; only use enough dimensions that the human eye is able to discern between digits. This latter approach is the one I take. From Figure 1b, it is apparent that no more than 50 dimensions should be necessary to reproduce the structure of the digit. Adding additional dimensions beyond this only seems to sharpen the image, which our classifiers shouldn't rely on.

The first two images in Figure 1b are linear combinations of the mean image and the eigendigits displayed in Figure 1a.



(a) The first 10 eigendigits

(b) Eigendigit reconstructions

Figure 1: Some of the eigendigits, and the approximations they form.

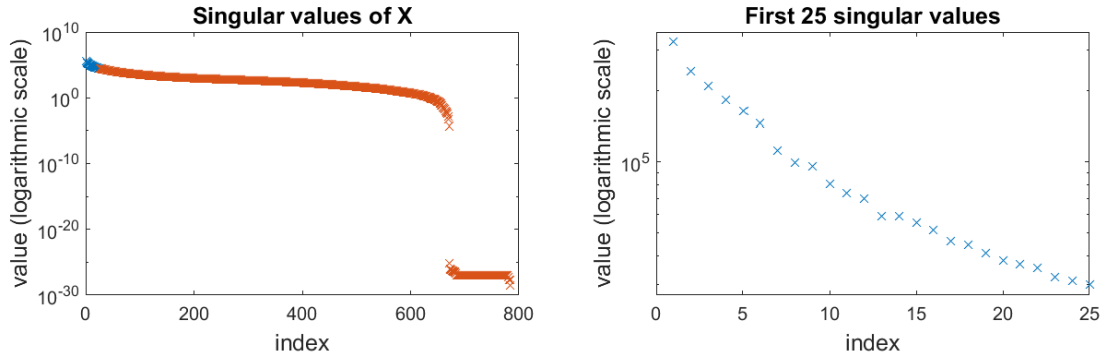


Figure 2: Singular values of X (after centering)

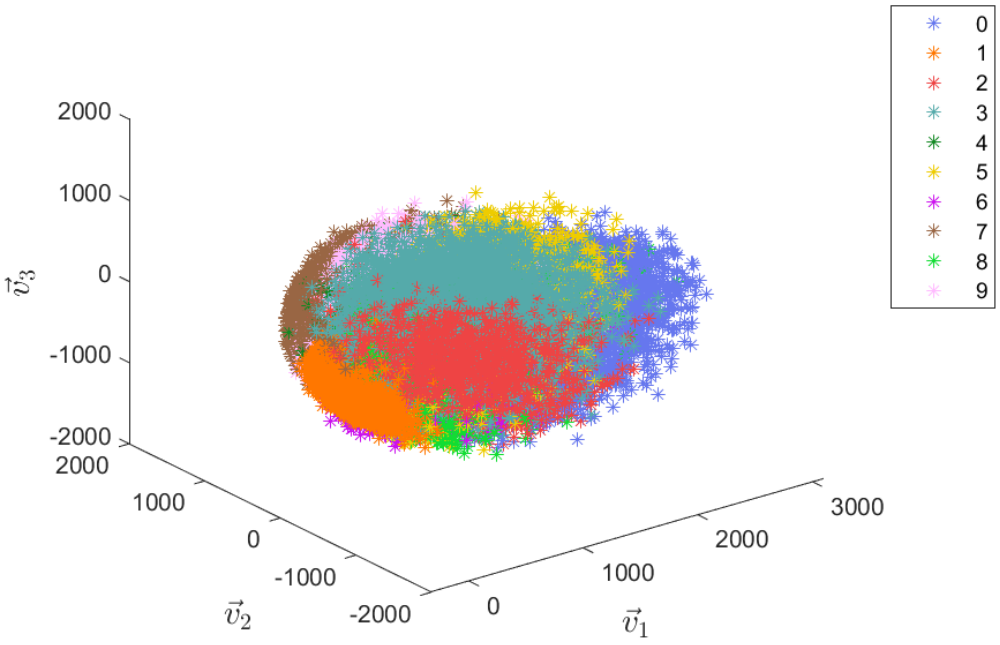


Figure 3: Projection coefficients along the first three singular vectors, colored by digit

3 Algorithm Implementation and Development

3.1 Linear classification

Once the data has been plotted into PCA-space, we can begin the process of classification. In linear classification, we seek to find a vector $\vec{w}$ such that when we project our data onto the line span$\{\vec{w}\}$, the resulting projection coefficients form clusters corresponding to the labels. We can then threshold an image's projection coefficient to classify it. This is called Linear Discriminant Analysis.

3.1.1 Finding the right projection image

Suppose we have N labeled groups of data, where the j-th group has mean μ_j. Let μ be the arithmetic mean of all groups. Define $S_B = \sum_{j=1}^{N}(\mu_j - \mu)(\mu_j - \mu)^\top$ to measure the covariance between groups, and define $S_W = \sum_{j=1}^{N}\sum_{\vec{x}_i}(\vec{x}_i - \mu_j)(\vec{x}_i - \mu_j)^\top$ to measure the covariance between means.

The scalar $\vec{w}^\top Aw$ characterizes the inner product of the vector $\vec{w}$ before and after transformation with A. Consider the quantity $\dfrac{\vec{w}^\top S_B \vec{w}}{\vec{w} S_W \vec{w}}$. Intuitively, this quantity will be maximized when $\vec{w}$ is close to an eigenvector of S_B (the between-class matrix) and far from the eigenvectors of S_W (the within-class matrix). When both of these things are true, I can expect lots of variance between classes and little variance within classes after projecting onto $\vec{w}$. This same vector is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem in Equation 1.

$$S_B \vec{w} = \lambda S_W \vec{w} \tag{1}$$

3.1.2 Thresholding

Let P_i be a sorted list containing the projection coefficients of group i onto span$\vec{w}$. Without loss of generality, suppose that if $i < j$, then the projections P_i should be less than the projections P_j.

Algorithm 1: Thresholding 2 groups after LDA

Input: P_1, P_2
  set $t_1 = \text{length}(P_1), t_2 = 0$
  while $P_1(t_1) > P_2(t_2)$ **do**
    decrement t_1
    increment t_2
  end while
  return the arithmetic mean of $\{t_1, t_2\}$

Algorithm 2: Thresholding 3 groups after LDA

Input: P_1, P_2, P_3
  Set $t_1 = \text{length}(P_1), t_2 = 0$
  Set $t_3 = \text{length}(P_2), t_4 = 0$
  while $P_1(t_1) > P_2(t_2)$ **do**
    Decrement t_1 by -2
    Increment t_2
  end while
  while $P_2(t_3) > P_3(t_4)$ **do**
    Decrement t_3
    Increment t_4 by 2
  end while
  return the arithmetic means of $\{t_1, t_2\}$ and of $\{t_3, t_4\}$

Algorithm 1 sets the threshold t so that within the training data, the cardinality of $P_1 > t$ approximately matches the cardinality of $P_2 < t$.

Algorithm 2 applies a similar approach to the case where three clusters are projected onto the same line, with slight modification. If Algorithm 1 were applied to P_1, P_2 and to P_2, P_3, then P_2 would be unfairly squished by thresholds on both sides. To counteract this, the algorithm is twice as lenient to P_2.

4 Computational Results

LDA performed very well for the digit combinations of $\{0, 1\}$ and $\{0, 4\}$, with error close to 0.2%. LDA performed very poorly for the digit combinations of $\{7, 9\}$ and $\{4, 9\}$, with error close to 6%. Combinations $\{0, 1\}$ and $\{7, 9\}$ are plotted in Figure 4

LDA overall had a tougher time separating three digits at a time. The combination $\{0, 1, 7\}$ performed the best with less than 9% error, and the combination $\{2, 3, 6\}$ performed the worst with 65%, which is about as bad as randomly guessing. These combinations are plotted in Figure 5.

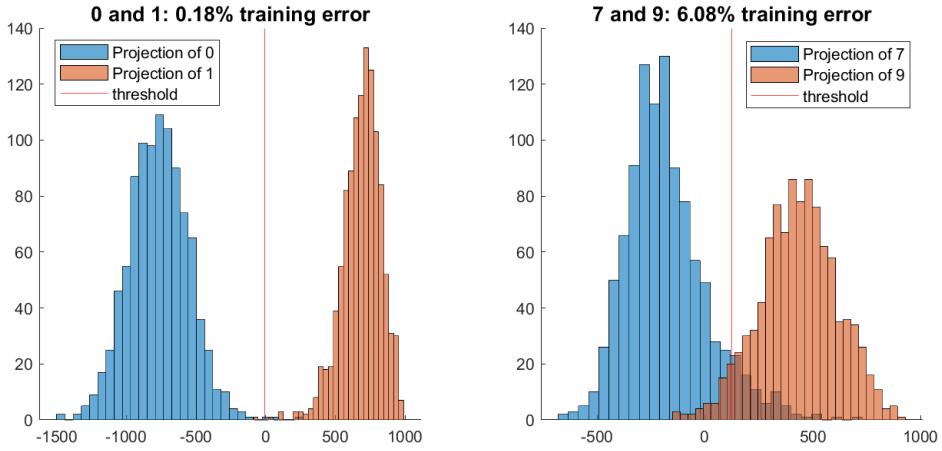


Figure 4: The best and worst 2-digit combinations for LDA

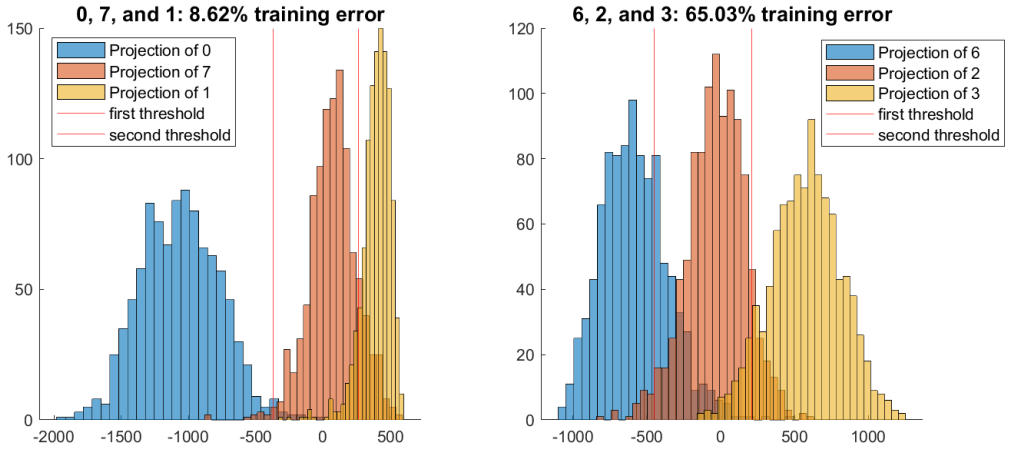


Figure 5: The best and worst 3-digit combinations for LDA

5 Summary and Conclusions

Principal Component Analysis performs exceedingly well to mine features in linearizable data. The clusters in Figure 3 show that many defining attributes of digits can be well-characterized by only three singular vectors. This analysis was conducted with a projection onto the first twenty singular vectors, which from Figure 1b does not seem like it would be optimal, and yet it performs quite well.

Linear Discriminant Analysis performs very well to separate easily linearizable data. This includes data with two clusters. Linear Discriminant Analysis also seems to be somewhat effective with data that forms three clusters, though some triples of digits may prove a challenge.

Appendix A MATLAB Functions

- `[performance, w, threshold] = lda2(a, b, img_train, label_train, img_test, label_test)` inputs two digits a, b and the associated image/label data sets. The image data should already be projected into PCA-space. The method builds an LDA model by computing the projection vector $\vec{w}$ and the threshold. Output variable `performance` stores the percentage of errors when testing.
- `[performance, w, threshold] = lda3(a, b, c, img_train, label_train, img_test, label_test)` is similar, except it inputs three digits a, b, c and builds a 3-class LDA model and returns two thresholds.
- `[U,S,V] = svd(X)` computes the singular value decomposition of X.

Appendix B MATLAB Code

B.1 main.m

```
close all; clear all;
[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
%%
len = size(images, 1) * size(images, 2); % # of pixels in an img
n = size(images, 3); % # of imgs

Xbig = double(reshape(images, len, n));
% each observation is a column in X, an element of R^{len}

% doing SVD would require 26.8 GB memory. Let's subsample.
% Let's make the sampling deterministic for easier replication.
rng(482); % param
entries = randperm(n);
pca_sample_factor = 6; % param
num_samples = floor(n/pca_sample_factor);
entries = entries(1:num_samples);

X = zeros(len, num_samples);
for j = 1:num_samples
    i = entries(j);
    X(:,j) = Xbig(:,i);
end

%%
% Let's try visually identifying how many modes it takes, without the PCA
%  method of zero-mean.
[U,S,V] = svd(X, 'econ');
```

```
figure(1)
subplot(2,4,8);
imshow(uint8(reshape(X(:,1), [28 28])));
title("True image", "FontSize", 16);

%modes = [inf 600 300 100 50 25 10 5];
modes = [5 10 25 50 100 300 600 inf];
for j = 1:7
    m = modes(j);
    subplot(2,4,j);
    approx = U(:,1:m) * S(1:m, 1:m) * V(:,1:m)';
    imshow(uint8(reshape(approx(:,1), [28 28])));
    title(strcat(int2str(m), " modes"), "FontSize", 16);
end

set(gcf, 'Position', [600 500 750 400]);
saveas(gcf, 'approximations.png');

clear approx;
%% Now let's do PCA
% Center the pixel values on each row
mn = mean(X,2);
X = X - repmat(mn, 1, num_samples);

[U,S,V] = svd(X'/sqrt(num_samples-1), 'econ');
lambda = diag(S).^2;
%clear U X; % we only need the projection vectors V and the singular values


figure(2)
% Let's also look at what the modes themselves are, the eigen-digits.
for i = 1:10
    subplot(2,5,i);
    digit = reshape(V(:,i), [28 28]);
    s = pcolor(digit);
    s.EdgeColor = 'none';
    title(int2str(i), 'FontSize', 16);
    ax = gca;
    ax.YTick = [];
    ax.XTick = [];
end

set(gcf, 'Position', [600 500 750 400]);
saveas(gcf, 'eigendigits.png');



% Plot the singular values
figure(3)
subplot(1,2,1);
semilogy(1:25, lambda(1:25), 'x', 26:len, lambda(26:end), 'x');
title("Singular values of X", 'FontSize', 12);
xlabel("index", 'FontSize', 12);
```

```
ylabel("value (logarithmic scale)", 'FontSize', 12);

subplot(1,2,2);
semilogy(1:25, lambda(1:25), 'x');
ylim([lambda(25)*0.9 lambda(1)*1.1]);
xlabel("index", 'FontSize', 12);
ylabel("value (logarithmic scale)", 'FontSize', 12);
title("First 25 singular values", 'FontSize', 12);

set(gcf, 'Position', [600 500 900 250]);
saveas(gcf, 'singular_values.png');

% There are 677 "non-zero" singular values in this set. That means the
% pixels are in a space of about R^674.

%% Let's try plotting all images onto three projection coefficients
cols = [1 2 3]; % param
coeffs = zeros(n, 4);  % three coeffs and the label
v1 = V(:,cols(1))';
v2 = V(:,cols(2))';
v3 = V(:,cols(3))';
for j=1:n
    img = Xbig(:,j);
    coeffs(j, 1) = v1*img;
    coeffs(j, 2) = v2*img;
    coeffs(j, 3) = v3*img;
    coeffs(j, 4) = labels(j);
end

X = sortrows(coeffs, 4);

first = zeros(10,1);
last = zeros(10,1);

for digit = 0:9
    rows = X(:,4) == digit;
    first(digit+1) = find(rows, 1, 'first');
    last(digit+1) = find(rows, 1, 'last');
end

figure(4);
newcolors = {'#67E','#F70','#E44','#5AA','#182','#EC0','#C1E','#964','#1D3','#FBF'};
colororder(newcolors)
symb = '*';
plot3(X(first(1):last(1),1), X(first(1):last(1),2), X(first(1):last(1),3), symb, ...
      X(first(2):last(2),1), X(first(2):last(2),2), X(first(2):last(2),3), symb, ...
      X(first(3):last(3),1), X(first(3):last(3),2), X(first(3):last(3),3), symb, ...
      X(first(4):last(4),1), X(first(4):last(4),2), X(first(4):last(4),3), symb, ...
      X(first(5):last(5),1), X(first(5):last(5),2), X(first(5):last(5),3), symb, ...
      X(first(6):last(6),1), X(first(6):last(6),2), X(first(6):last(6),3), symb, ...
      X(first(7):last(7),1), X(first(7):last(7),2), X(first(7):last(7),3), symb, ...
      X(first(8):last(8),1), X(first(8):last(8),2), X(first(8):last(8),3), symb, ...
      X(first(9):last(9),1), X(first(9):last(9),2), X(first(9):last(9),3), symb, ...
      X(first(10):last(10),1), X(first(10):last(10),2), X(first(10):last(10),3), symb);
```

```
%title(strcat("Projection onto first three right singular vectors"))
legend("0","1","2","3","4","5","6","7","8","9");

xlabel("$\vec{v}_1$", 'Interpreter', 'latex', 'FontSize', 14);
ylabel("$\vec{v}_2$", 'Interpreter', 'latex', 'FontSize', 14);
zlabel("$\vec{v}_3$", 'Interpreter', 'latex', 'FontSize', 14);

set(gcf, 'Position', [600 500 600 400]);
saveas(gcf, '10projections.png');

%% Prepare for ML methods
proj = V(:,1:20);

img_train = proj' * Xbig;
label_train = labels;
img_test = proj' * double(reshape(images_test, [784 10000]));
label_test = labels_test;

%% LDA2
errors = zeros(10);
for a = 0:8
    for b = (a+1):9
        [performance, w, threshold] = lda2(a, b, img_train, label_train, img_test, label_test, false);
        errors(a+1, b+1) = performance;
        errors(b+1, a+1) = performance;
    end
end

% Worst is 7 and 9, 6.09%
%          4 and 9, 6.07%
% Best is 0 and 1, 0.19%
%         0 and 4, 0.20%

%% LDA3
errors = zeros(10,10,10);
for a = 0:7
    for b = (a+1):8
        for c = (b+1):9
            % abc acb bac bca cab cba
            try
                [performance, w, threshold] = lda3(a, b, c, img_train, label_train, img_test, label_tes
            catch ME
                disp(strcat("Didn't work for", ...
                    int2str(a), ...
                    int2str(b), ...
                    int2str(c)))
            end
            errors(a+1, b+1, c+1) = performance;
        end
    end
end

% best is 0, 1, 7 with 8.62% error
```

```
%%
close all;
figure(5);

subplot(1,2,1);
a = 0; b = 1;
[performance, w, threshold] = lda2(a, b, img_train, label_train, img_test, label_test, true);
%title(strcat(int2str(a), " and ", int2str(b)), 'FontSize', 13);
ax = gca;
l = ax.Legend;
l.Position = [0.1454 0.7658 0.1422 0.1325];
l.FontSize = 10;

subplot(1,2,2);
a = 7; b = 9;
[performance, w, threshold] = lda2(a, b, img_train, label_train, img_test, label_test, true);
%title(strcat(int2str(a), " and ", int2str(b)), 'FontSize', 13);
ax = gca;
l = ax.Legend;
l.FontSize = 10;

set(gcf, 'Position', [600 500 950 400]);
saveas(gcf, 'bestworst.png');

%%
close all;
figure(6);

subplot(1,2,1);
a = 0; b = 1; c=7;
[performance, w, threshold] = lda3(a, b, c, img_train, label_train, img_test, label_test, true);
%title(strcat(int2str(a), " and ", int2str(b)), 'FontSize', 13);
ax = gca;
l = ax.Legend;
l.Position = [0.1454 0.7658 0.1422 0.1325];
l.FontSize = 10;

subplot(1,2,2);
a = 2; b = 3; c=6;
[performance, w, threshold] = lda3(a, b, c, img_train, label_train, img_test, label_test, true);
%title(strcat(int2str(a), " and ", int2str(b)), 'FontSize', 13);
ax = gca;
l = ax.Legend;
l.FontSize = 10;

set(gcf, 'Position', [600 500 950 400]);
saveas(gcf, 'bestworsttriple.png');



%%
tree = fitctree(img_train', label_train, 'MaxNumSplits', 10, 'CrossVal', 'on');
%view(tree.Trained{1},'Mode','graph');
%classError = kfoldLoss(tree);
```

```
error = evaluateClassifier(tree, img_test', label_test)
%%
n = 1000;
Mdl = fitcsvm(img_train(:,1:n)', label_train(1:n));
result = predict(Mdl,img_test');
score = sum(result == label_test);
error = score / n;
```

B.2 lda2.m

```
function [performance, w, threshold] = lda2(a, b, img_train, label_train, img_test, label_test, verbose)
% INPUT
% a, b are the digits to try and classify
% V are the singular projection vectors (pre-computed). Should be 784 x modes.
% img data should be reshaped to modes x n, so a column is a PCA'd image.
% if verbose, then it will place a double histogram in the current figure.
%
% OUTPUT
% performance is the percentage of images correctly classified.
% w spans the projection space
% threshold gives the value to project on

    % Make a < b
    if a > b
        temp = a;
        a = b;
        b = temp;
        clear temp
    end

    img_train_a = img_train(:,label_train==a);
    img_train_b = img_train(:,label_train==b);


    % Compute means of each group for variance calculation.
    ma = mean(img_train_a, 2);
    mb = mean(img_train_b, 2);

    % Sw: within-class scatter matrix. Get the variance within each group.
    % Sb: between-class scatter matrix. Get the variance between the two means.
    na = size(img_train_a, 2);
    nb = size(img_train_b, 2);

    Sb = (ma-mb)*(ma-mb)';
    Sw = zeros(size(img_train_a, 1));
    size(ma);
    size(img_train_a(:,1));

    for k = 1:na
        Sw = Sw + (img_train_a(:,k) - ma)*(img_train_a(:,k) - ma)';
    end
    for k = 1:nb
        Sw = Sw + (img_train_b(:,k) - mb)*(img_train_b(:,k) - mb)';
```

```
end

[V2, D] = eig(Sb, Sw);
[~, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

% Now let's project from modes subspace to w, and find a threshold.
lin_a = w'*img_train_a;
lin_b = w'*img_train_b;

% Let's make it so that digit a is below the threshold.
if mean(lin_a) > mean(lin_b)
    w = -w;
    lin_a = -lin_a;
    lin_b = -lin_b;
end
lin_a = sort(lin_a);
lin_b = sort(lin_b);

t1 = length(lin_a);
t2 = 1;
while (lin_a(t1) > lin_b(t2))
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = mean([lin_a(t1) lin_b(t2)]);

% Now let's classify each testing image.
img_test_a = img_test(:,label_test==a);
img_test_b = img_test(:,label_test==b);

a_projections = w' * img_test_a;
b_projections = w' * img_test_b;

errors = sum(a_projections > threshold) + sum(b_projections < threshold);

na = size(img_test_a, 2);
nb = size(img_test_b, 2);
performance = errors / (na+nb);

if verbose
    hold on;
    histogram(a_projections, 30);
    histogram(b_projections, 30);
    xline(threshold, 'r');

    percent = num2str(floor(performance*10000)/100);
    title(strcat(string(a), " and ", string(b), ": ", ...
        percent, "% training error"), "FontSize", 13);

    legend(strcat("Projection of ", int2str(a)), ...
        strcat("Projection of ", int2str(b)), ...
        "threshold");
```

```
        hold off;
    end
end
```

B.3 lda3.m

```
function [performance, w, threshold] = lda3(a, b, c, img_train, label_train, img_test, label_test, verbo
% INPUT
% a, b are the digits to try and classify
% V are the singular projection vectors (pre-computed). Should be 784 x modes.
% img data should be reshaped to modes x n, so a column is a PCA'd image.
% if verbose, then it will place a double histogram in the current figure.
%
% OUTPUT
% performance is the percentage of images correctly classified.
% w spans the projection space
% threshold gives the value to project on

    digits = [a b c];
    digits = sort(digits);
    a = digits(1);
    b = digits(2);
    c = digits(3);


    img_train_a = img_train(:,label_train==a);
    img_train_b = img_train(:,label_train==b);
    img_train_c = img_train(:,label_train==c);

    % Compute means of each group for variance calculation.
    ma = mean(img_train_a, 2);
    mb = mean(img_train_b, 2);
    mc = mean(img_train_c, 2);
    mu = mean([img_train_a img_train_b img_train_c], 2);

    % Sb: between-class scatter matrix. Get the variance between the two means.
    Sb = (ma-mu)*(ma-mu)' + ...
         (mb-mu)*(mb-mu)' + ...
         (mc-mu)*(mc-mu)';

    % Sw: within-class scatter matrix. Get the variance within each group.
    Sw = zeros(size(img_train_a, 1));

    na = size(img_train_a, 2);
    nb = size(img_train_b, 2);
    nc = size(img_train_c, 2);

    for k = 1:na
        Sw = Sw + (img_train_a(:,k) - ma)*(img_train_a(:,k) - ma)';
    end
    for k = 1:nb
        Sw = Sw + (img_train_b(:,k) - mb)*(img_train_b(:,k) - mb)';
    end
    for k = 1:nc
```

```
    Sw = Sw + (img_train_c(:,k) - mc)*(img_train_c(:,k) - mc)';
end

[V2, D] = eig(Sb, Sw);
[~, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

% Now let's project from modes subspace to w, and find thresholds.
lin_a = w'*img_train_a;
lin_b = w'*img_train_b;
lin_c = w'*img_train_c;

% order the values based on their order of projection.
order = [mean(lin_a) mean(lin_b) mean(lin_c); a b c]';
order = sortrows(order, 1);

switch(order(1,2))
    case a
        lin_1 = lin_a;
    case b
        lin_1 = lin_b;
    case c
        lin_1 = lin_c;
    otherwise
        disp("failed at reordering")
end
switch(order(2,2))
    case a
        lin_2 = lin_a;
    case b
        lin_2 = lin_b;
    case c
        lin_2 = lin_c;
    otherwise
        disp("failed at reordering")
end
switch(order(3,2))
    case a
        lin_3 = lin_a;
    case b
        lin_3 = lin_b;
    case c
        lin_3 = lin_c;
    otherwise
        disp("failed at reordering")
end

a = order(1,2);
b = order(2,2);
c = order(3,2);


% now lin_a < lin_b < lin_c
```

```
% time to find thresholds.

t1 = length(lin_a);
t2 = 1;

while (lin_a(t1) > lin_b(t2))
    t1 = t1 - 2;
    t2 = t2 + 1;
end

t3 = length(lin_b);
t4 = 1;
while (lin_b(t3) > lin_c(t4))
    t3 = t3 - 1;
    t4 = t4 + 2;
end

threshold = [mean([lin_a(t1) lin_b(t2)]) ...
             mean([lin_b(t3) lin_c(t4)])];

% Now let's classify each testing image.
img_test_a = img_test(:,label_test==a);
img_test_b = img_test(:,label_test==b);
img_test_c = img_test(:,label_test==c);

a_projections = w' * img_test_a;
b_projections = w' * img_test_b;
c_projections = w' * img_test_c;

errors = sum(a_projections > threshold(1)) ...
    + sum(b_projections < threshold(1)) ...
    + sum(b_projections > threshold(2)) ...
    + sum(c_projections < threshold(2));

na = size(img_test_a, 2);
nb = size(img_test_b, 2);
nc = size(img_test_c, 2);
performance = errors / (na+nb+nc);

if verbose
    hold on;
    histogram(a_projections, 30);
    histogram(b_projections, 30);
    histogram(c_projections, 30);
    xline(threshold(1), 'r');
    xline(threshold(2), 'r');

    percent = num2str(floor(performance*10000)/100);
    title(strcat(string(a), ", ", string(b), ", and ", string(c), ": ", ...
        percent, "% training error"), "FontSize", 13);

    legend(strcat("Projection of ", int2str(a)), ...
        strcat("Projection of ", int2str(b)), ...
        strcat("Projection of ", int2str(c)), ...
```

```
            "first threshold", ...
            "second threshold");

        hold off;
    end
end
```