AMATH 482 Homework 4

lewisxy

March 11, 2021

Abstract

In this paper, we demonstrated the possibility of classifying hand-written digits from MNIST dataset with Linear Discriminant Analysis (LDA), Support Vector Machine (SVM) and Decision Tree.

1 Introduction and Overview

In this paper, we explore the possibly of classifying hand-written digits from MNIST dataset with various techniques. In particular, we used Linear Discriminate Analysis in PCA Space with a subset of features. We also experimented with state-of-art classification algorithms such as SVM and decision trees (implementations are provided by MATLAB as is).

2 Theoretical Background

2.1 Principle Component Analysis (PCA)

At a high level, PCA allows us to reduce the dimension of data with minimum information loss (measured in L2 norm). The detail introduction of PCA is beyond the scope of this paper (as we already introduced it in the last assignment). In the context of this paper, we used PCA to extract a set of best features (features with maximum variances, a.k.a most separable) for LDA. We will use Singular Value Decomposition (SVD) to perform PCA.

2.2 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis is a classification technique. It involves projecting the data to a specific 1-dimensional space, spanned by single vector w, such that the data is most separable. Let S_B, S_W be the between class scatter matrix (measures the variance between features) and within class scatter matrix (measures the variance within features), mathematically, we can formulate w as the following.

$$w = \arg\max_w \frac{w^T S_B w}{w^T S_W w} \tag{1}$$

Intuitively, this means we want to find w to minimize the variance within features and maximizes the variance between features (analogy from ANOVA), and thus making the data most separable. Let μ be the mean for all data, μ_j be the mean for data of class j. Suppose we have N classes, and $C(j)$ denotes the set of data belong to class j, we can define S_B, S_W as the following.

$$S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T \tag{2}$$

$$S_W = \sum_{j=1}^{N} \sum_{x \in C(j)} (x - \mu_j)(x - \mu_j)^T \tag{3}$$

In order to find such w. We can take the generalized eigenvalue decomposition for matrix S_B and S_W, and find the eigenvector such with maximum eigenvalue (absolute value). A generalized eigenvalue decomposition given square matrices A, B of the same size seeks to find V, D (D is diagonal) such that $AV = BVD$.

With w, we can project training data to this space and find the threshold (the boundary between each clump of data points). In evaluation, we simply project test data to w, and use the threshold (intervals) of training data classify them.

2.3 Support Vector Machine (SVM)

Support Vector Machine seeks to find a hyperplane (high dimensional analogy of 2D plane) that separates the 2 class of data. Any hyperplane can be defined by equation $w^T x + b = 0$ where w is the normal of the plane. For any hyperplane defined by w and b, we can compute the signed distance of a data point x_i to the plane to be $(w^T x_i + b)/||w||_2$. For the best robustness of the model, we want to maximize the minimum distance for any vectors to the hyperplane (such vectors are known as "support vectors"). Given y values to be -1 and 1 for binary classification, the SVM problem can be formulated as the following.

$$\min_{w,b} ||w||_2^2 \text{such that } y_i(w^T x_i + b) \geq 1 \tag{4}$$

In reality, not all data is linearly separable. If that happens, we don't be able to find a solution. In these cases, we can either relax the contrarians or lifting the data to higher dimensions. We will talk about this in detail in later sections.

2.4 Decision Tree

Decision tree is another classification model. In the high level, it seeks to build a multi-level tree with each non-leaf node as a conditions on the input, and each leaf node of the tree is a classification result. For any input vector, it will goes deeper to the tree by sequentially evaluating conditions on the non-leaf nodes until it get classified to a leaf node. To grow such tree, we recursively partition the space in training set to minimize the impurity (which is a function measures the probability of a classification) until a depth limit is reached or no classifications occurs in the training set.

3 Algorithm Implementation and Development

In this section, we describes the different steps to analyze and classify data. Implementations are provided in the form of MATLAB code in Appendix B, explanations of selected MATLAB functions are provided in Appendix A.

3.1 Analysis

The firs step is to analyze the data. We perform a PCA on the data to find the dimension of the data, and extract corresponding features of it. Specifically, we use `svd` command to decompose the $d \times n$ column data matrix X (detail of this command can be found in Appendix A) into 3 matrices: U, Σ, V (Σ sometimes is written as S) such that $X = U\Sigma V^T$. The columns of U composed of the orthonormal eigenvectors of the matrix XX^T, which is the basis (features) extracted by PCA. Σ is a diagonal matrix containing the eigenvalues of for corresponding eigenvectors in U, measures the "strength" of each of those basis. Eigenvectors with small eigenvalues usually does not contain much information of the original matrix. V contains the normalized representation of data in the space spanned by U (Because $X = U\Sigma V^T$ and $UU^T = I$, we have relationship $U^T X = \Sigma V^T$, which means ΣV is the X changing the basis into U). Each column of V (known as V-mode) represents the relative strength of a particular principle component (need to be scaled by eigenvalue in Σ) in for all data points).

3.2 Classification

After analyzing the data, we run LDA with 20 features with largest singular value to classify both 2 digit and 3 digit cases. To compute the generalized eigenvalue, we use `eig` command in MATLAB (detail of this command can be found in Appendix A). To compute the threshold, we first sort the groups based on their mean in transformed space. Then for each pair consecutive groups, starting from the largest value of the larger group and smallest value of the smallest group, we find the 2 closest value by moving one index at a time, and take the average of them. This ensures that there is an approximately equal number of errors on both side of the threshold value. We also runs SVM and decision tree with implementations provided by MATLAB on 2 digits.

4 Computational Results

After performing PCA, we discovered the magnitude of singular values does not decrease abruptly in Figure 1. This means the data has a relatively high dimensions. We can visualize the first 10 principle components in Figure 2. A lot of them resembles some of the digits, which indicates these basis indeed represents some noticeable aspect of the data. In Figure 3, we project the data onto 3 V-modes. Without adjusting the angle, we can ready notice data for different digits starting to goes together. This is a good sign as it will make is easier for us to classify later. In the end, for computing efficiencies, we choose $k = 20$ features for LDA.

We performed LDA for 2 digits on each pair of digits. Most pairs get classified really well. Figure 4 shows the distribution of errors for pair of digits. The pair with lowest test success rate is 4 and 9, with success rate of 93.97%, another similarly difficult pair is 3 and 5, with success rate of 94.27%, and the highest is between 0 and 1, 99.95%. The success rate when evaluating on training data is very similar to the evaluation on test data (difference ¡ 1%).

We then performed the LDA on 3 digits. The success rate for classifying $1, 3, 5$ is 75.11% and for $1, 4, 9$ is 71.05%, which is still reasonable as it's much better than random guessing of $1/3 = 33.33\%$. We also tried to run LDA on the entire dataset with 10 digits. The success rate is 25.50%. While this success rate seems low, to put it into an context, the success rate for 10-class classification of random guessing is about $1/10 = 10\%$. From this perspective, LDA is clearly working as our result is 2.5 times better than random guessing.

We then performed the SVM and decison tree classification of on 2 of the hardest pairs of digits, with both implementation and default parameters provided by MATLAB. For 4 and 9, SVM gives a success rate of 87.95% and decision tree gives 95.23%. For 3 and 5, SVM gives success rate of 83.86% and decision tree gives 95.85%. Figure 5 is a visualization of a decision tree. It is expected that decision tree performs better than SVM as a tree-like structure allows much fine grain division of space comparing to linearly dividing the 2 region in SVM's approach, particularly when the 2 classes are not very linearly separable. To fix this problem for SVM, there is 2 major strategies, either using kernel trick (which is a fancy way of computing the dot product of 2 vectors, potentially lifting the data to higher dimensions or performing nonlinear transform in the process), or using soft boundaries by allowing some failures in the training set. Given limited time, unfortunately, we were not able to implement these improvements for SVM.

Comparing LDA and decision tree, it's surprising that LDA performs very close to decision tree, as LDA only seeks to differentiate data on 1 dimension. This, once again, demonstrate the power of PCA, which provides a set of very linearly separable features. A potential improvement for LDA is to find even better features. Since we are dealing with image data, we can also transform the image to Fourier or wavelet space and then perform PCA, which might gives interesting classification result.

5 Summary and Conclusions

In this paper, we demonstrated the possibility of classifying image of MNIST dataset with Linear Discriminant Analysis (LDA), Support Vector Machine (SVM) and Decision Tree. With proper features selected thanks to PCA, LDA performs suprisingly well in 2-digit classification. SVM does not perform well due to inability to linearly separate the data, and decision tree performs well due to its spatial partitioning capabilities. There are different trade-offs between these 3 methods in term of complexity and performance. In

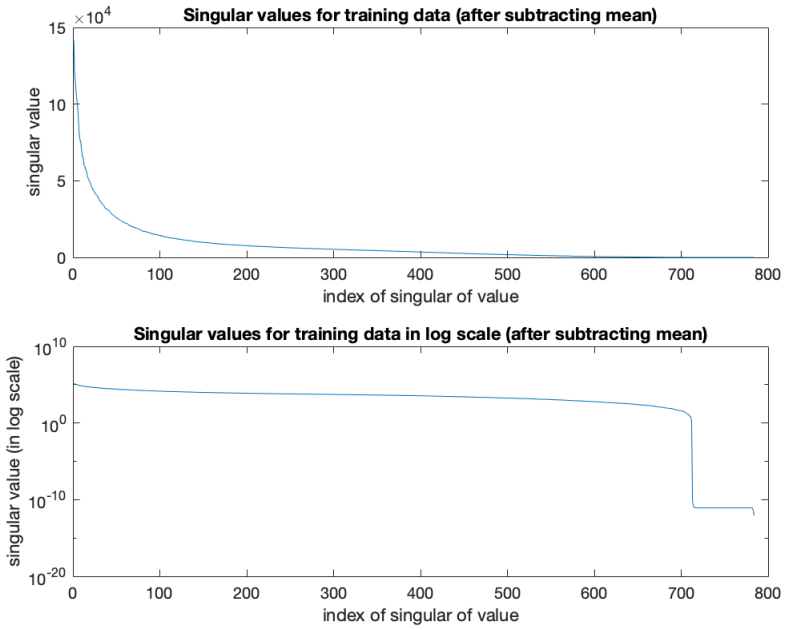

Figure 1: Singular values for the training data

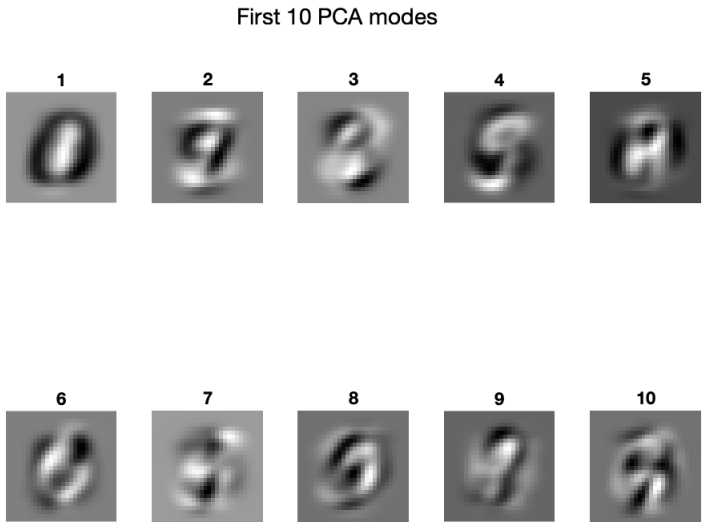

Figure 2: Visualizations of First 10 PCA modes for training data

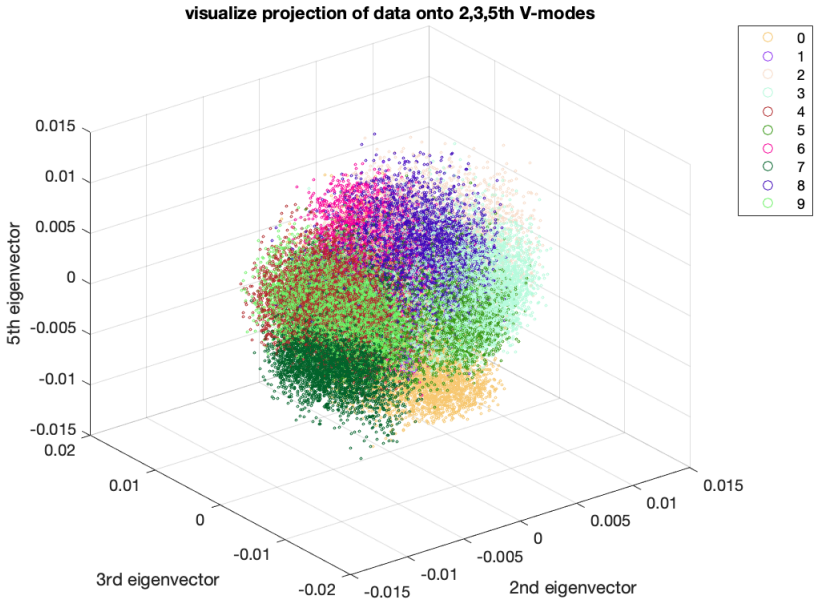

Figure 3: Visualizations of training data projecting onto 2,3,5th V-modes

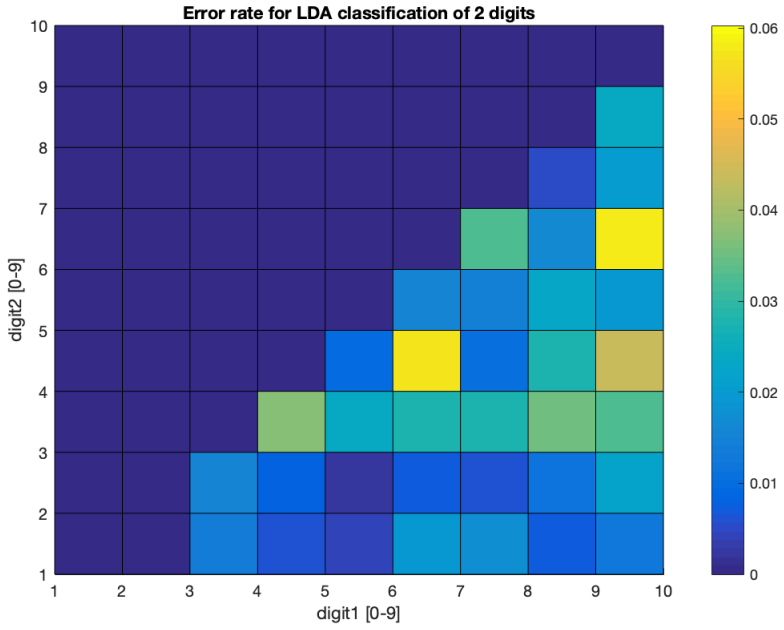

Figure 4: Error rate for LDA classification of 2 digits, only lower right triangle have data. The value next to bright yellow square has relatively high error rate

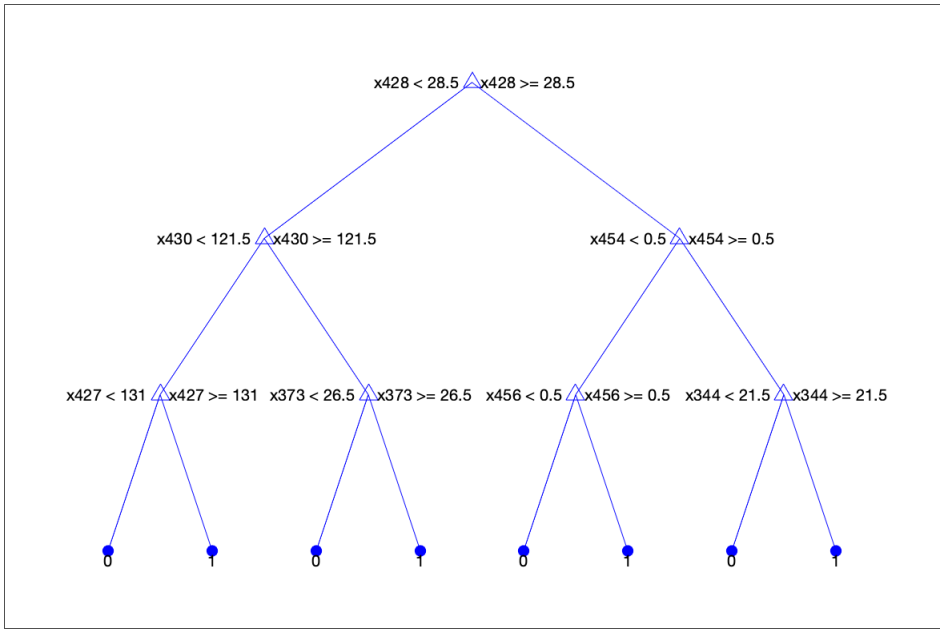

Figure 5: Visualizations of the decision tree with `MaxNumSplits` = 7 for demonstration. The actual tree is much deeper and thus unable to show on a figure

reality, we need to choose appropriate method depending the context.

Appendix A MATLAB Functions

- `Model = fitcsvm(X, Y)`: train an SVM classifier given training data `X` and training labels `Y`, returns a model object.
- `Model = fitctree(X, Y)`: train an decision tree classifier given training data `X` and training labels `Y`, returns a model object.
- `result = predict(Model, X)`: return the result of model prediction `result` using the given model `Model` to predict on given data `X`. The output format may depends on the `Model`.
- `[U,S,V] = svd(A, 'econ')`: perform reduced SVD on the given matrix A, return `U`, `S`, `V` of the decomposed matrix such that $A = USV^T$.
- `[V,D] = eig(A, B)`: perform generalized eigenvalue decomposition on the given matrices A, B, return `V`, `D`, such that $AV = BVD$.

Appendix B MATLAB Code

```
function [U,S,V,threshold,w,sort_v_data1,sort_v_data2] =
↪  lda_train(data1,data2,num_feature)
    n_data1 = size(data1,2);
    n_data2 = size(data2,2);
    [U,S,V] = svd([data1 data2],'econ');
    data = S*V';
    U = U(:,1:num_feature); % Add this in
    data1 = data(1:num_feature,1:n_data1);
    data2 = data(1:num_feature,n_data1+1:n_data1+n_data2);
    m_data1 = mean(data1,2);
    m_data2 = mean(data2,2);

    Sw = 0;
    for k=1:n_data1
        Sw = Sw + (data1(:,k)-m_data1)*(data1(:,k)-m_data1)';
    end
    for k=1:n_data2
        Sw = Sw + (data2(:,k)-m_data2)*(data2(:,k)-m_data2)';
    end
    Sb = (m_data1-m_data2)*(m_data1-m_data2)';

    [V2,D] = eig(Sb,Sw);
    [lambda,ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    v_data1 = w'*data1;
    v_data2 = w'*data2;

    if mean(v_data1) > mean(v_data2)
        w = -w;
        v_data1 = -v_data1;
        v_data2 = -v_data2;
    end

    % Don't need plotting here
    sort_v_data1 = sort(v_data1);
    sort_v_data2 = sort(v_data2);
    t1 = length(sort_v_data1);
    t2 = 1;
    while sort_v_data1(t1) > sort_v_data2(t2)
        t1 = t1-1;
        t2 = t2+1;
    end
    threshold = (sort_v_data1(t1)+sort_v_data2(t2))/2;

    % We don't need to plot results
end
```

Listing 1: Algorithm for training LDA classifier of 2 classes

```
function [res] = lda_classify(U, w, threshold, data)
    proj_data = U' * data;
```

```
    lda_value = w' * proj_data;
    res = (lda_value > threshold);
end
```

Listing 2: Algorithm for classifying 2 classes with LDA

```
function [U,S,V,threshold,w,labelset] = lda_multi_train(data,labels,labelset,num_feature)
    % labelset map index to labels
    % values in labelset must be unique
    % inverse_labelset map labels to index in labelset
    inverse_labelset = @(x) find(labelset==x);

    n_data = size(data,2);
    [U,S,V] = svd(data,'econ');
    data = S*V';
    data = data(1:num_feature, :);
    U = U(:,1:num_feature); % Add this in
    n_group = length(labelset);
    m_all = mean(data, 2); % overall mean

    % compute mean for each group
    m_data = zeros(size(data, 1), n_group);
    for i=1:n_group
        m_data(:, i) = mean(data(:, labels==labelset(i)), 2);
    end

    % compute SS_within and SS_between
    Sw = 0;
    for i=1:n_data
        Sw = Sw + (data(:,i)-m_data(inverse_labelset(labels(i))))*...
        (data(:,i)-m_data(inverse_labelset(labels(i))))';
    end

    Sb = 0;
    for i=1:n_group
        Sb = Sb + (m_data(i) - m_all)*(m_data(i) - m_all)';
    end

    [V2,D] = eig(Sb,Sw);
    [lambda,ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    v_data = w'*data;

    m_v_data = zeros(n_group, 1);
    for i=1:n_group
        m_v_data(i) = mean(v_data(:, labels==labelset(i)));
    end
    % sort labelset based on mean
    tmp_ = [m_v_data labelset'];
    tmp_ = sortrows(tmp_, 1);
    labelset = tmp_(:, 2)';
    % we don't care mean, but it's in tmp_(:, 1)
```

```
    % compute thresholds
    threshold_ = zeros(1, n_group+1);
    threshold_(1) = [-inf];
    for i=1:n_group-1
        % we can do this since labelset is sorted based on mean
        tmp1_ = sort(v_data(labels==labelset(i)));
        tmp2_ = sort(v_data(labels==labelset(i+1)));
        t1 = length(tmp1_);
        t2 = 1;
        while tmp1_(t1) > tmp2_(t2)
            t1 = t1-1;
            t2 = t2+1;
        end
        threshold_(i+1) = (tmp1_(t1)+tmp2_(t2))/2;
    end
    % threshold_ should contain n_group + 1 elements after this line
    threshold_(end) = inf;

    % threshold contains the range of values each group can take
    threshold = zeros(n_group, 2);
    for i=1:n_group
        threshold(i, 1) = threshold_(i);
        threshold(i, 2) = threshold_(i+1);
    end
end
```

Listing 3: Algorithm for training LDA classifier of N classes

```
function [res] = lda_multi_classify(U, w, threshold, labelset, data)
    % assume labelset is sorted
    proj_data = U' * data;
    size(w')
    size(proj_data)
    lda_value = w' * proj_data;
    res = zeros(1, size(data, 2));
    for i=1:length(labelset)
        lb = threshold(i, 1);
        ub = threshold(i, 2);
        tmp_ = (lda_value > lb & lda_value < ub);
        res(tmp_) = labelset(i);
    end
end
```

Listing 4: Algorithm for classifying N classes with LDA

```
%% setup
clear; close all; clc;

%% load data
[train_images, train_labels] = mnist_parse('data.nosync/train-images.idx3-ubyte',
↪  'data.nosync/train-labels.idx1-ubyte');
[test_images, test_labels] = mnist_parse('data.nosync/t10k-images.idx3-ubyte',
↪  'data.nosync/t10k-labels.idx1-ubyte');
```

```
%% flatten data: turn image to vectors
train_flatten = zeros(size(train_images, 1) * size(train_images, 2), size(train_images,
↪  3));
for i=1:size(train_images, 3)
    tmp_ = train_images(:, :, i);
    train_flatten(:, i) = tmp_(:);
end

test_flatten = zeros(size(test_images, 1) * size(test_images, 2), size(test_images, 3));
for i=1:size(test_images, 3)
    tmp_ = test_images(:, :, i);
    test_flatten(:, i) = tmp_(:);
end

%% train SVM
% digit1 = 4; % 3
% digit2 = 9; % 5
digit1 = 3; % 3
digit2 = 5; % 5
features = 784;

digit1_train_data = train_flatten(1:features, train_labels==digit1);
digit2_train_data = train_flatten(1:features, train_labels==digit2);

train_input = [digit1_train_data digit2_train_data]';
train_truth = [zeros(1, size(digit1_train_data, 2)) ones(1, size(digit2_train_data,
↪  2))]';

SVMModel = fitcsvm(train_input ,train_truth);

%% test results
digit1_test_data = test_flatten(1:features, test_labels==digit1);
digit2_test_data = test_flatten(1:features, test_labels==digit2);

test_input = [digit1_test_data digit2_test_data]';

[~, score] = predict(SVMModel, test_input);
truth = [zeros(1, size(digit1_test_data, 2)) ones(1, size(digit2_test_data, 2))]';

correct_count = sum((score(:, 2) > 0) == (truth == 1)); % 1751 outof 1991
correct_proportion = correct_count / size(test_input, 1); % 0.8795

% result: 4,9: 0.8795, 3,5: 0.8386
```

Listing 5: Algorithm for svm

```
%% setup
clear; close all; clc;

%% load data
[train_images, train_labels] = mnist_parse('data.nosync/train-images.idx3-ubyte',
↪  'data.nosync/train-labels.idx1-ubyte');
[test_images, test_labels] = mnist_parse('data.nosync/t10k-images.idx3-ubyte',
↪  'data.nosync/t10k-labels.idx1-ubyte');
```

```
%% flatten data: turn image to vectors
train_flatten = zeros(size(train_images, 1) * size(train_images, 2), size(train_images,
↪  3));
for i=1:size(train_images, 3)
    tmp_ = train_images(:, :, i);
    train_flatten(:, i) = tmp_(:);
end

test_flatten = zeros(size(test_images, 1) * size(test_images, 2), size(test_images, 3));
for i=1:size(test_images, 3)
    tmp_ = test_images(:, :, i);
    test_flatten(:, i) = tmp_(:);
end

%% train SVM
digit1 = 4; % 3
digit2 = 9; % 5
% digit1 = 3; % 3
% digit2 = 5; % 5
features = 784;

digit1_train_data = train_flatten(1:features, train_labels==digit1);
digit2_train_data = train_flatten(1:features, train_labels==digit2);

train_input = [digit1_train_data digit2_train_data]';
train_truth = [zeros(1, size(digit1_train_data, 2)) ones(1, size(digit2_train_data,
↪  2))]';

Tree = fitctree(train_input, train_truth, 'MaxNumSplits', 7);
view(Tree,'Mode','graph');

%% test results
digit1_test_data = test_flatten(1:features, test_labels==digit1);
digit2_test_data = test_flatten(1:features, test_labels==digit2);

test_input = [digit1_test_data digit2_test_data]';

score = predict(Tree, test_input);
truth = [zeros(1, size(digit1_test_data, 2)) ones(1, size(digit2_test_data, 2))]';

correct_count = sum(score == truth);
correct_proportion = correct_count / size(test_input, 1);

% result: 4,9: 0.9523, 3,5: 0.9585
```

Listing 6: Algorithm for Decision tree

```
%% setup
clear; close all; clc;

%% load data
[train_images, train_labels] = mnist_parse('data.nosync/train-images.idx3-ubyte',
↪  'data.nosync/train-labels.idx1-ubyte');
```

```
[test_images, test_labels] = mnist_parse('data.nosync/t10k-images.idx3-ubyte',
↪  'data.nosync/t10k-labels.idx1-ubyte');

%% flatten data: turn image to vectors
test_flatten = zeros(size(test_images, 1) * size(test_images, 2), size(test_images, 3));
for i=1:size(test_images, 3)
    tmp_ = test_images(:, :, i);
    test_flatten(:, i) = tmp_(:);
end

train_flatten = zeros(size(train_images, 1) * size(train_images, 2), size(train_images,
↪  3));
for i=1:size(train_images, 3)
    tmp_ = train_images(:, :, i);
    train_flatten(:, i) = tmp_(:);
end

%% remove mean of the data
test_mean = mean(test_flatten, 2);
test_demean = test_flatten - test_mean;
train_mean = mean(train_flatten, 2);
train_demean = train_flatten - train_mean;

%% perform SVD
% [U,S,V] = svd(test_demean,'econ');
[U,S,V] = svd(train_demean,'econ');
% U (784, 784) is eigenvectors
% S (784, 784) is diagonal matrix with eigenvalues
% V (10000, 784) is how data represented in eigen space (spaned by U)
% we can use V as training data for classification (LDA, ...)
% as SVD makes it is quite separatable

%% visualize 10 largest eigenvectors
for i=1:10
    tmp_ = U(:, i);
    tmp_ = reshape(tmp_, 28, 28);
    imshow(tmp_, []); % show with adaptative range
end

%% plot distribution of singular values
tmp_ = diag(S);
subplot(2, 1, 1);
plot(tmp_)
title('Singular values for training data (after subtracting mean)')
ylabel('singular value')
xlabel('index of singular of value')
subplot(2, 1, 2);
semilogy(tmp_)
title('Singular values for training data in log scale (after subtracting mean)')
ylabel('singular value (in log scale)')
xlabel('index of singular of value')

%% test reconstruction
% reconstruct image 1 from SVD result
```

```
tmp_ = U * S * V(1, :)';
imshow(reshape(tmp_, 28, 28), [])
% this is the same as
imshow(reshape(test_demean(:, 1), 28, 28), [])

% reconstruct with k modes
k = 100;
tmp_ = U(:, 1:k) * S(1:k, 1:k) * V(1, 1:k)';
imshow(reshape(tmp_, 28, 28), [])

% visualize difference
plot(tmp_)
hold on;
plot(test_demean(:, 1))

% reconstruction error (MSE)
mean((tmp_ - test_demean(:, 1)).^2)

%% visualize first 10 PCA modes
figure
suptitle('First 10 PCA modes');
for i=1:10
    subplot(2, 5, i);
    imshow(reshape(U(:, i), 28, 28), [])
    title(num2str(i));
end

%% visualize projection onto 3D
% set seed for color
% rng(123456);
rng(4);
colors = rand(10, 3);
% v_idx = [1, 3, 5];
v_idx = [2, 3, 5];
% v_idx = [2, 2, 2];

v_modes = V(:, v_idx);
% scatter3(v_modes(:, 1), v_modes(:, 2), v_modes(:, 3), 1, colors(test_labels + 1))

% plot 10 times to make it easier for legend
figure
for i=(1:10)
    scatter3(v_modes(train_labels==i-1, 1), v_modes(train_labels==i-1, 2),
    ↪  v_modes(train_labels==i-1, 3), 3, colors(i, :))
    if i == 1
        hold on;
    end
end
hold off;

tmp_ = num2str(((1:10)-1)');
% tmp_ = mat2cell(tmp_, ones(1,10), 1);
tmp_ = num2cell(tmp_);
legend(tmp_{:});
```

```
% to make it easier to view the resulted plot
rotate3d on;
title('visualize projection of data onto 2,3,5th V-modes')
xlabel('2nd eigenvector');
ylabel('3rd eigenvector');
zlabel('5th eigenvector');

%% LDA
digit1 = 0;
digit2 = 1;

% Project onto PCA modes
feature = 20;

proj_data = S*V'; % projection onto principal components: X = USV' --> U'X = SV'

digit1_data = proj_data(1:feature, test_labels==digit1);
digit2_data = proj_data(1:feature, test_labels==digit2);

% number of data vector for each catagories
n_digit1 = size(digit1_data, 2);
n_digit2 = size(digit2_data, 2);

% Calculate scatter matrices
m_digit1 = mean(digit1_data, 2);
m_digit2 = mean(digit2_data, 2);

Sw = 0; % within class variances
for k = 1:n_digit1
    Sw = Sw + (digit1_data(:,k) - m_digit1)*(digit1_data(:,k) - m_digit1)';
end
for k = 1:n_digit2
    Sw = Sw + (digit2_data(:,k) - m_digit2)*(digit2_data(:,k) - m_digit2)';
end

Sb = (m_digit1-m_digit2)*(m_digit2-m_digit2)'; % 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

v_digit1_data = w'*digit1_data;
v_digit2_data = w'*digit2_data;

%% Make digit1 below the threshold

if mean(v_digit1_data) > mean(v_digit2_data)
    w = -w;
    v_digit1_data = -v_digit1_data;
```

```
    v_digit2_data = -v_digit2_data;
end

%% Plot projections (not for function)

figure
plot(v_digit1_data,zeros(n_digit1),'ob','Linewidth',2)
hold on
plot(v_digit2_data,ones(n_digit2),'dr','Linewidth',2)
ylim([0 1.2])

%% Find the threshold value

sort_digit1 = sort(v_digit1_data);
sort_digit2 = sort(v_digit2_data);

t1 = length(v_digit1_data);
t2 = 1;
while sort_digit1(t1) > sort_digit2(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sort_digit1(t1) + sort_digit2(t2))/2;

%% Plot histogram of results

figure
subplot(1,2,1)
histogram(sort_digit1,30); hold on, plot([threshold threshold], [0 210],'r')
% set(gca,'Xlim',[-300 400],'Ylim',[0 300],'Fontsize',14)
title('digit1')
subplot(1,2,2)
histogram(sort_digit2,30); hold on, plot([threshold threshold], [0 210],'r')
% set(gca,'Xlim',[-300 400],'Ylim',[0 300],'Fontsize',14)
title('digit2')

%%
```

Listing 7: Algorithm for analyzing data

```
%% setup
clear; close all; clc;

%% load data
[train_images, train_labels] = mnist_parse('data.nosync/train-images.idx3-ubyte',
↪  'data.nosync/train-labels.idx1-ubyte');
[test_images, test_labels] = mnist_parse('data.nosync/t10k-images.idx3-ubyte',
↪  'data.nosync/t10k-labels.idx1-ubyte');

%% flatten data: turn image to vectors
train_flatten = zeros(size(train_images, 1) * size(train_images, 2), size(train_images,
↪  3));
for i=1:size(train_images, 3)
    tmp_ = train_images(:, :, i);
```

```
    train_flatten(:, i) = tmp_(:);
end

test_flatten = zeros(size(test_images, 1) * size(test_images, 2), size(test_images, 3));
for i=1:size(test_images, 3)
    tmp_ = test_images(:, :, i);
    test_flatten(:, i) = tmp_(:);
end

%% train LDA
digit1 = 5;
digit2 = 6;
features = 20;

digit1_train_data = train_flatten(:, train_labels==digit1);
digit2_train_data = train_flatten(:, train_labels==digit2);

[U,S,V,threshold,w,sort_v_data1,sort_v_data2] = lda_train(digit1_train_data,
↪  digit2_train_data, features);

%% test results
digit1_test_data = test_flatten(:, test_labels==digit1);
digit2_test_data = test_flatten(:, test_labels==digit2);
test_input = [digit1_test_data digit2_test_data];

res = lda_classify(U, w, threshold, test_input); % return 1 for digit2, 0 for digit1
truth = [zeros(1, size(digit1_test_data, 2)) ones(1, size(digit2_test_data, 2))];

correct_count = sum(res == truth);
correct_proportion = correct_count / size(test_input, 2);
```

Listing 8: Driver code for LDA of 2 classes

```
%% setup
clear; close all; clc;

%% load data
[train_images, train_labels] = mnist_parse('data.nosync/train-images.idx3-ubyte',
↪  'data.nosync/train-labels.idx1-ubyte');
[test_images, test_labels] = mnist_parse('data.nosync/t10k-images.idx3-ubyte',
↪  'data.nosync/t10k-labels.idx1-ubyte');

%% flatten data: turn image to vectors
train_flatten = zeros(size(train_images, 1) * size(train_images, 2), size(train_images,
↪  3));
for i=1:size(train_images, 3)
    tmp_ = train_images(:, :, i);
    train_flatten(:, i) = tmp_(:);
end

test_flatten = zeros(size(test_images, 1) * size(test_images, 2), size(test_images, 3));
for i=1:size(test_images, 3)
    tmp_ = test_images(:, :, i);
    test_flatten(:, i) = tmp_(:);
```

```
end

%% train LDA
%digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]; % 0.2550
% digits = [1, 3, 5]; % 0.7511
% digits = [1, 4, 9]; % 0.7105
n_group = length(digits);
features = 20;

train_data_size = zeros(1, n_group);

for i=1:n_group
    tmp_ = train_flatten(:, train_labels==digits(i));
    train_data_size(i) = size(tmp_, 2);
end

% allocate space for data
train_data_extracted = zeros(size(train_flatten, 1), sum(train_data_size));
tmp_ = 1;
% build training data (and labels)
for i=1:n_group
    train_data_extracted(:, tmp_:tmp_+train_data_size(i)-1) = train_flatten(:,
    ↪  train_labels==digits(i));
    train_labels_extracted(tmp_:tmp_+train_data_size(i)-1) = digits(i);
    tmp_ = tmp_ + train_data_size(i);
end

[U,S,V,threshold,w,labelset] = lda_multi_train(train_data_extracted,
↪  train_labels_extracted, digits, features);

%% test results
test_data_size = zeros(1, n_group);

for i=1:n_group
    tmp_ = test_flatten(:, test_labels==digits(i));
    test_data_size(i) = size(tmp_, 2);
end

% allocate space for data
test_data_extracted = zeros(size(test_flatten, 1), sum(test_data_size));
tmp_ = 1;
% build testing data (and labels)
for i=1:n_group
    test_data_extracted(:, tmp_:tmp_+test_data_size(i)-1) = test_flatten(:,
    ↪  test_labels==digits(i));
    test_labels_extracted(tmp_:tmp_+test_data_size(i)-1) = digits(i);
    tmp_ = tmp_ + test_data_size(i);
end

res = lda_multi_classify(U, w, threshold, labelset, test_data_extracted);

correct_count = sum(res == test_labels_extracted);
correct_proportion = correct_count / size(test_data_extracted, 2);
```

Listing 9: Driver code for LDA of N classes

```
%% setup
clear; close all; clc;

%% load data
[train_images, train_labels] = mnist_parse('data.nosync/train-images.idx3-ubyte',
↪  'data.nosync/train-labels.idx1-ubyte');
[test_images, test_labels] = mnist_parse('data.nosync/t10k-images.idx3-ubyte',
↪  'data.nosync/t10k-labels.idx1-ubyte');

%% flatten data: turn image to vectors
train_flatten = zeros(size(train_images, 1) * size(train_images, 2), size(train_images,
↪  3));
for i=1:size(train_images, 3)
    tmp_ = train_images(:, :, i);
    train_flatten(:, i) = tmp_(:);
end

test_flatten = zeros(size(test_images, 1) * size(test_images, 2), size(test_images, 3));
for i=1:size(test_images, 3)
    tmp_ = test_images(:, :, i);
    test_flatten(:, i) = tmp_(:);
end

%% train LDA
result = zeros(10, 10);
result_train = zeros(10, 10);
for i=1:9
    for j=(i+1):10
        digit1 = i-1;
        digit2 = j-1;
        features = 20;

        digit1_train_data = train_flatten(:, train_labels==digit1);
        digit2_train_data = train_flatten(:, train_labels==digit2);

        [U,S,V,threshold,w,sort_v_data1,sort_v_data2] = lda_train(digit1_train_data,
        ↪  digit2_train_data, features);

        % error on training set
        train_input = [digit1_train_data digit2_train_data];
        res = lda_classify(U, w, threshold, train_input); % return 1 for digit2, 0 for
        ↪  digit1
        truth = [zeros(1, size(digit1_train_data, 2)) ones(1, size(digit2_train_data,
        ↪  2))];

        correct_count = sum(res == truth);
        correct_proportion = correct_count / size(train_input, 2);
        result_train(i, j) = correct_proportion;

        % test results
        digit1_test_data = test_flatten(:, test_labels==digit1);
        digit2_test_data = test_flatten(:, test_labels==digit2);
```

```
        test_input = [digit1_test_data digit2_test_data];

        res = lda_classify(U, w, threshold, test_input); % return 1 for digit2, 0 for
        ↪  digit1
        truth = [zeros(1, size(digit1_test_data, 2)) ones(1, size(digit2_test_data, 2))];

        correct_count = sum(res == truth);
        correct_proportion = correct_count / size(test_input, 2);

        result(i, j) = correct_proportion;
    end
end

%%
tmp_ = result;
tmp_(tmp_ == 0) = 1;
figure
surf(1 - tmp_)
colorbar
rotate3d on;
title('Error rate for LDA classification of 2 digits');
xlabel('digit1 [0-9]');
ylabel('digit2 [0-9]');
zlabel('error'); % 4 and 9: 0.9397, 3 and 5: 0.9427
```

Listing 10: Driver code for LDA of 2 classes