AMATH 482: Homework 4

Alex Omusoru

Due: March 10, 2021

Abstract

Using the MNIST database, this report developed a linear classifier (Linear Discriminant Analysis - LDA) model to attempt digit classification. This model was applied for binary class identification of pairs of digits, and an attempt was made to apply it to a multi-class set as well. Support-Vector Machine (SVM) models and decision tree models were also applied to the data for multi-class as well as binary class identification, and compared both to each other, and to the LDA model developed.

1 Introduction and Overview

The intent of this report was multi-faceted. The first goal was to complete an analysis of the principal component analysis (PCA) of the training data. The second goal was to construct a binary class linear identifier model, using linear discriminant analysis (LDA). That model would then be applied to help distinguish between pairs of digits. The third goal was to train SVM and decision tree models for 10 digit classification, as well as binary identification (like the LDA model). The final goal was to extend the LDA model to three digit classification, however that was unsuccessful.

2 Theoretical Background

2.1 Principal Component Analysis

The goal of Principal Component Analysis is to produce low-rank approximations of data sets. It is an application of SVD to the covariance matrix

$$C_X = \frac{1}{n-1} X X^T$$

Given X is the data matrix.

PCA analysis is identical to SVD, and simply requires a transformation of the data matrix. If we consider the decomposition of C_X to be as follows,

$$C_X = U \Sigma V^T$$

Then we can learn alot about the images in an orthonormal basis. The columns of U (left singular vectors) represent the modes of the image, while the columns of V represent how the modes will change over time. Singular values represent the variance of the distribution of the function along the semimajor axes.

2.2 Linear Discriminant Analysis

The goal of LDA is to find a suitable projection which maximizes the inter-class data while minimizing the intra-class data. This essentially means the idea is to find a projection of the data which causes it to clump in seperate groups, in an identifiable manner.
LDA can be applied to multiple classes, and is as follows. We begin by defining two **scatter matrices** as follows:

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

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

Where S_B is the **between-class scatter matrix**, and S_w is the **within-class scatter matrix**.
Our goal is then to find the vector

$$w = argmax \frac{w^T S_B w}{w^T S_w w}$$

Which can be found by solving the eigenvalue problem

$$S_b w = \lambda S_w w$$

Once w is identified, it is a simple matter to determine the threshold values for each axis of projection.

3 Algorithm Implementation and Development

After setting up and importing the training and test data sets (see *Algorithm: Data Set-up*), the project was divided into two parts: LDA Model Development and Testing, and SVM/Decision Tree Model Development and Testing.

Algorithm: Data Set-up

```
given I = (n x m x c) training images, L = (1 x c) labels
D = reshape I to 2D array of dim (n x m) x (c)
M = row-wise means of D
D = D - M to center data
repeat for test images I_t and L_t to get D_t
center D_t using above M
```

3.1 LDA Model Development and Testing

Constructing and testing the LDA model was conducted for every unique pair of digits $a, b \in 0, 1, ..., 9$. For each digit pair, the first step was to train the model (see *Algorithm: LDA Model Training*). Following the training of the model, testing was conducted (see *Algorithm: LDA Model Testing*).

Algorithm: LDA Model Training

```
given data (D), labels (L), digits (a, b), feature
D_ab = filter D to include only a and b images
[U, S, V] = SVD of D_ab
U = rows 1:feature of U
digits = S * transpose of V
digits_a = filter columns by digit a, and take 1:feature rows
digits_b = filter columns by digit b, and take 1:feature rows
Find Sw, Sb
Solve eigenvalue problem for V2, D given Sb*V2 = D*Sw*V2
w = eigenvector of largest eigenvalue
Project digits_a and digits_b onto w (get 1D v_a, v_b)
t = threshold value between v_a and v_b
(note that t is the value that minimizes the number of
a and b values together on either side along the number line)=
```

Algorithm: LDA Model Testing

```
given test data (D_t), labels (L_t), digits (a, b), feature
D_ab = filter D to include only a and b images
filter D_t by digits a and b to get D_t_about
```

```
Project D_t_ab onto U (from LDA Model)
Project result onto w (from LDA Model) to get p-vector
values in p-vector > t are a's, and the rest are b's
calculate success by comparing to L_t
```

3.2 SVM/Decision Tree Model Development and Testing

The following was done seperately for each model (SVM and Decision Tree). The construction and testing of the model for all 10 digits was completed first (see *Algorithm: Built-In Model for 10 Digits*). Then, for the hardest and easiest digit pairs, a model was trained and tested (see *Algorithm: Built-In Model for 2 Digits*). Note that a seperate model was trained and tested for each pair.

Algorithm: Built-In Model for 10 Digits

```
given data (D), labels (L), test data (D_t), test labels (L_t)
find S, V from SVD of D
scale factor (SF) = max of S * transpose of V
normalizedD = S * transpose of V / SF
trainedModel = model(normalizedD, labels)
find St, Vt from SVD of D_t
normalizedD_t = St * transpose of V / SF
results = predict(trainedModel, normalizedD_t)
calculate success by comparing to L_t
```

Algorithm: Built-In Model for 2 Digits

```
given data (D), labels (L), test data (D_t), test labels (L_t), digits (a, b)
filter D and D_t by digits a and b to get D_ab and D_t_ab
find S, V from SVD of D_ab
scale factor (SF) = max of S * transpose of V
normalizedD_ab = S * transpose of V / SF
trainedModel = model(normalizedD_ab, labels)
find St, Vt from SVD of D_t_ab
normalizedD_t_ab = St * transpose of Vt / SF
results = predict(trainedModel, normalizedD_t_ab)
calculate success by comparing to L_t
```

4 Computational Results

4.1 SVD Analysis and Extensions

The first step of the computations after importing and setting up the training data, was to decompose the data matrix into U, Σ, and V matrices using SVD. I was then able to plot the singular value spectrum (see *Figure 1*), and the associated energies of each singular values (see *Figure 2*).

The intent of this analysis, was to determine the how many modes were appropriate for use in the LDA model. To that end, I reconstructed mode approximations of the first image in the set based on the 10%, 20%, ..., 100% energy modes (see *Figure 3*).

While the rank r was determined to be $r = 712$, based on the images I determined the 90% energy mode $m = 87$ to be a reasonable choice for the LDA model.To complete the analysis, I projected the digit images onto three choice V-modes (modes 4, 5, and 8) and then plotted each image as a (x, y, z) point, colored based on their digit label (see *Figure 4*).

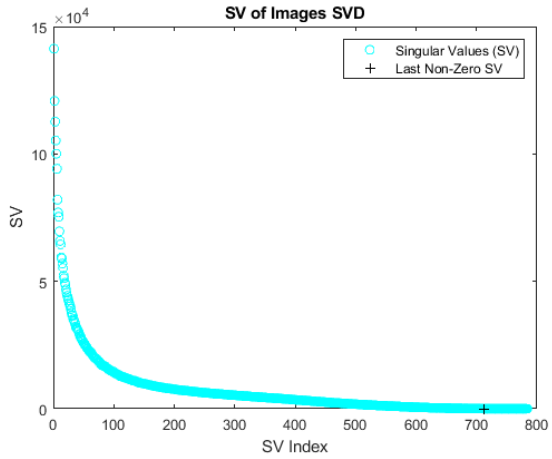

Figure 1: Graph of Singular Values from SVD of Training Data

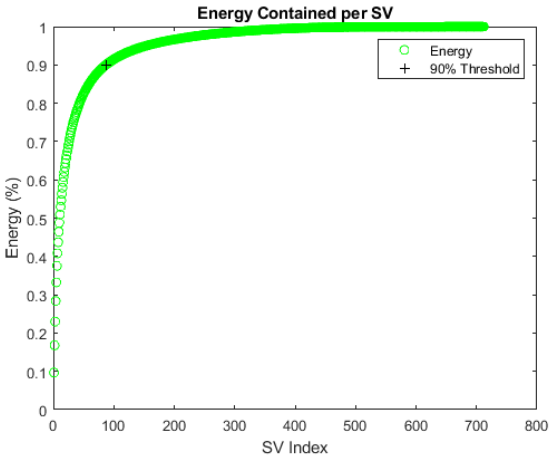

Figure 2: Graph of Energy by Singular Value

4.2 Training and Testing Models

The linear classifier (LDA) model was initially construced for digits $a = 6$ and $b = 7$, which resulted in a model with 99.65% accuracy for the test data, and 99.82% accuracy for the training data. These percent accuracies were deemed to be acceptable for this study, and therefore the same algorithm was generalized for all unique digit pairs a, b where $a, b \in 0, 1, ..., 9$. Their accuracies for the data are summarized in *Table 1* (for test data) and *Table 2* (for training data).

Table 1: Test Data Success Rate for LDA Model

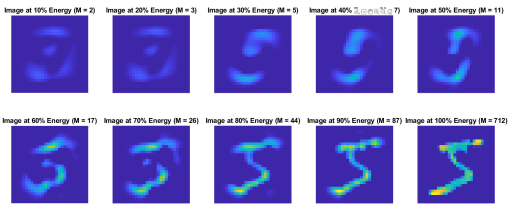

Figure 3: Graph of Mode Approximations by Energy

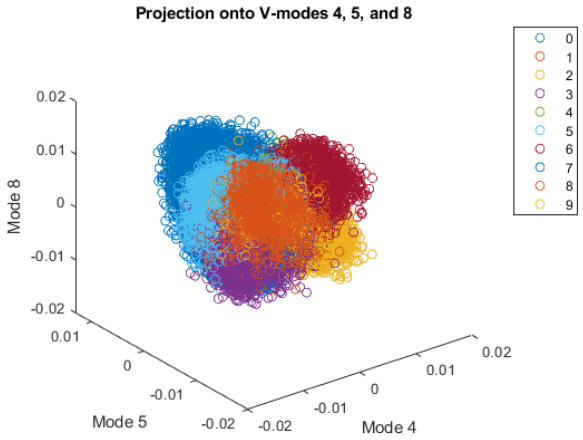

Figure 4: 3D Plot of Data Projected onto 3 V-Modes

digits	0	1	2	3	4	5	6	7	8	9
0		0.9976	0.9876	0.9955	0.9990	0.9877	0.9907	0.9930	0.9908	0.9915
1			0.9894	0.9916	0.9962	0.9926	0.9957	0.9917	0.9777	0.9939
2				0.9750	0.9796	0.9740	0.9759	0.9762	0.9721	0.9833
3					0.9930	0.9611	0.9934	0.9809	0.9632	0.9792
4						0.9877	0.9907	0.9856	0.9918	0.9513
5							0.9697	0.9911	0.9577	0.9842
6								0.9965	0.9845	0.9964
7									0.9820	0.9578
8										0.9733

Table 2: Training Data Success Rate for LDA Model

digits	0	1	2	3	4	5	6	7	8	9
0		0.9962	0.9850	0.9910	0.9947	0.9843	0.9900	0.9957	0.9868	0.9923
1			0.9849	0.9857	0.9943	0.9901	0.9961	0.9922	0.9727	0.9940
2				0.9682	0.9824	0.9719	0.9754	0.9823	0.9665	0.9834
3					0.9908	0.9538	0.9914	0.9852	0.9619	0.9773
4						0.9877	0.9905	0.9845	0.9901	0.9552
5							0.9735	0.9921	0.9585	0.9833
6								0.9982	0.9839	0.9973
7									0.9871	0.9555
8										0.9769

Based on the accuracies of the LDA model on the test data, I identified two pairs of digits. The easiest ones to distinguish (ones with the highest accuracy) and the hardest ones to distinguish (ones with lowest

accuracy). The hardest digit pair was $0, 4$ with 95.13% accuracy, while the easiest digit pair was $4, 9$ with 99.90% accuracy. I then constructed SVM and decision tree models for all 10 digits, the hardest digit pair, and the easiest digit pair.

4.2.1 10 Digit Comparison

For all 10 digits, I constructed a SVM model and decision tree model.
The accuracy of the SVM model for the test data was 9.04%, and for the training data it was 94.75%.
The accuracy of the decision tree model for the test data was 7.26%, and for the training data it was 96.18%.
Both models proved to struggle fitting test data, potentially due to overfitting to the training data.

4.2.2 2 Digit Comparison

For the easy and hard digits, I constructed a SVM model and decision tree model, and compared their results to the LDA model.

Easy Digits (0, 4): The accuracy of the SVM model for these digits on the test data was 72.94%, and for the training data it was 99.75%.
The accuracy of the decision tree model for these digits on the test data was 79.26%, and for the training data it was 99.86%.
The accuracy of the LDA model for these digits on the test data was 99.90%, and for the training data it was 95.52%

Hard Digits (4, 9): The accuracy of the SVM model for these digits on the test data was 30.14%, and for the training data it was 97.33%.
The accuracy of the decision tree model for these digits on the test data was 38.67%, and for the training data it was 98.83%.
The accuracy of the LDA model for these digits on the test data was 95.13%, and for the training data it was 95.52%

4.3 3 Digit Comparison

Constructing a linear classifier algorithm for an LDA model was attempted for three digit comparison, however I was unable to complete it, as I could not determine appropriate decision boundaries for the two-dimensional space of the three digits projected onto the first two eigenvectors of the solution to the Sb, Sw eigenvalue problem. (The attempted function is included below in *Appendix A* (see *Three Index Trainer*)).

5 Summary and Conclusions

Three types of models were constructed for the MNIST dataset. A SVM model, a decision tree model, and a linear classifier (LDA) model were used. The SVM and the decision tree models were first applied to the entire 10 digit set, however both models struggled to identify the test data. When compared to their accuracy on the training data, we can conclude that the models were likely overfitted, as the accuracy was much higher. We also find that the decision tree model performs worse than the SVM model for 10 digits.

I also applied the LDA model to all the unique pairs of data of digits 0 to 9 (excluding mirror pairs - i.e. 0 and 0). From the models accuracy on test data, I determined that the hardest digits to distinguish were 4 and 9, while the easiest were 0 and 4. I then compared the accuracy of the LDA model on those two pairs to the SVM and decision tree models (one model trained for each pair for each type). The results indicated that the LDA model was more appropriately suited to the classification of two digits. While they all had comparable success rates on the training data, when examing the test data, the SVM and decision tree models had a significantly lower (¿20% difference) accuracy for both sets of digits. While the decision tree model performed much worse than the LDA model, it was slightly better than the SVM model.

Appendix A MATLAB Functions

- `[images, labels] = mnist_parse(path_to_digits, path_to_labels)` converts the inputted MNIST files (for training or test data) into matrices representing the images and the corresponding labels. (see *Appendix A: Image Parser* for implementation details)
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `[U,S,V,threshold,w,sortb,sorta, accuracy] = two_index_trainer2(data, labels, a, b, feature)` produces the svd decomposition for the data filtered for digits a and b, and provides the LDA model parameters as well as the training data accuracy. (see *Appendix A: Two Index Trainer* for implementation details)
- `[V,D,W] = eig(A,B)` also returns full matrix W whose columns are the corresponding left eigenvectors, so that W'*A = D*W'*B.
- `Mdl = fitcecoc(Tbl,ResponseVarName)` returns a full, trained, multiclass, error-correcting output codes (ECOC) model using the predictors in table Tbl and the class labels in Tbl.ResponseVarName. fitcecoc uses K(K – 1)/2 binary support vector machine (SVM) models using the one-versus-one coding design, where K is the number of unique class labels (levels). Mdl is a ClassificationECOC model.
- `Mdl = fitcsvm(Tbl,ResponseVarName)` returns a support vector machine (SVM) classifier Mdl trained using the sample data contained in the table Tbl. ResponseVarName is the name of the variable in Tbl that contains the class labels for one-class or two-class classification.
- `tree = fitctree(Tbl,ResponseVarName)` returns a fitted binary classification decision tree based on the input variables (also known as predictors, features, or attributes) contained in the table Tbl and output (response or labels) contained in Tbl.ResponseVarName. The returned binary tree splits branching nodes based on the values of a column of Tbl.
- `label = predict(Model,X)` returns a vector of predicted class labels for the predictor data in the table or matrix X, based on the trained model Model.

Appendix B MATLAB Code

Main

```
%% Reset Workspace
close all; clear; clc;

%% Training Data Setup
path_to_digits = "train-images.idx3-ubyte";
path_to_labels = "train-labels.idx1-ubyte";

[images, labels] = mnist_parse(path_to_digits, path_to_labels);

data = zeros(28*28, 60000);
for i = 1:60000
    data(:, i) = reshape(images(:, :, i), [28*28, 1]);
end

avg_scaling = mean(data, 2);

for i = 1:728
```

```
    data(i,:) = data(i,:) - avg_scaling(i);
end

%% Test Data Setup
path_to_t_digits = "t10k-images.idx3-ubyte";
path_to_t_labels = "t10k-labels.idx1-ubyte";

[t_images, t_labels] = mnist_parse(path_to_t_digits, path_to_t_labels);


test_data = zeros(28*28, length(t_images));
for i = 1:length(t_images)
    test_data(:, i) = reshape(t_images(:, :, i), [28*28, 1]);
end

for i = 1:728
    test_data(i,:) = test_data(i,:) - avg_scaling(i);
end

%% Analysis of SVD

% rank of S matrix
[U, S, V] = svd(data, 'econ');
r_digit = rank(S);
sig = diag(S);

% singular value space
figure(1)
plot(1:length(sig), sig, 'co', r_digit, 0, 'k+');
legend('Singular Values (SV)', 'Last Non-Zero SV');
title('SV of Images SVD');
xlabel('SV Index')
ylabel('SV')

% energy
energy = zeros(1, r_digit);
for i = 1:r_digit
    energy(i) = sum(sig(1:i).^2)/sum(sig.^2);
end

%% feature determination:
figure(2)
sv_10s = zeros(1, 10);
for i = 1:10
    mode = find(energy >= i/10, 1, 'first');
    M_rank = U(:,1:mode)*S(1:mode,1:mode)*V(:,1:mode).';

    subplot(2, 5, i) % plot first image

    image(reshape(M_rank(:, 1), [28, 28]))

    axis image;
    title(['Image at ', num2str(i*10), '% Energy (M = ',num2str(mode),')'])
    set(gca, 'FontSize', 14)
```

```
    axis off;

    sv_10s(i) = mode;
end
feature = sv_10s(9); %87
%%
figure(3)
plot(1:length(energy), energy, 'go', feature, energy(feature), 'k+');
legend('Energy', '90% Threshold');
title('Energy Contained per SV')
xlabel('SV Index')
ylabel('Energy (%)')

%% Projection onto 3 V-modes
figure(4)
modes = [4 5 8];
for label = 0:9
    label_indices = find(labels == label);
    plot3(V(label_indices, modes(1)), V(label_indices, modes(2)),...
    V(label_indices, modes(3)),'o', 'DisplayName', sprintf('%i',label))
    hold on;
end
xlabel('Mode 4'), ylabel('Mode 5'), zlabel('Mode 8');
title('Projection onto V-modes 4, 5, and 8');
legend;
hold off;

%% LDA Model for two digits
[U, ~, ~, t, w, ~, ~, p] = two_index_trainer(data, labels, 6, 7, feature);

a = 6;
b = 7;
% refine test_data for digits 6 and 7
index_abt = find(t_labels == a | t_labels == b);
TestSet = test_data(:, index_abt);
labels_abt = t_labels(index_abt);
labels_abt = double(labels_abt == a);

% test LDA model on test data
TestNum = size(TestSet,2);
TestMat = U'*TestSet; % PCA projection
pval = w'*TestMat;

% calculate accuracy
results = (pval > t);
err = abs(results - labels_abt');
errNum = sum(err);

successRates_LDA_6_7 = 1 - errNum/TestNum;
successRates_LDA_6_7_TRAINING = p;


%% LDA Model Construction and Testing => successRate for all unique pairs of a and b (0:9)
```

```
successRates_LDA = zeros(10, 10);
successRates_LDA_TRAINING = zeros(10, 10);
for a = 0:9
    for b = a+1:9
        [U, ~, ~, t, w, ~, ~, p] = two_index_trainer(data, labels, a, b, feature);

        % refine test_data for digits a and b
        index_abt = find(t_labels == a | t_labels == b);
        TestSet = test_data(:, index_abt);
        labels_abt = t_labels(index_abt);
        labels_abt = double(labels_abt == a);


        % test LDA model on data
        TestNum = size(TestSet,2);
        TestMat = U'*TestSet; % PCA projection
        pval = w'*TestMat;

        % calculate accuracy
        results = (pval > t);
        err = abs(results - labels_abt');
        errNum = sum(err);

        successRates_LDA(a+1, b+1) = 1 - errNum/TestNum;

        successRates_LDA_TRAINING(a+1, b+1) = p;
    end
end

%% successRate Analysis
max_val = zeros(1, 3);
min_val = ones(1, 3);

for a = 0:9
    for b = a+1:9
        if max_val(1) < successRates_LDA(a+1, b+1)
            max_val = [successRates_LDA(a+1, b+1) a+1 b+1];
        end

        if min_val(1) > successRates_LDA(a+1, b+1)
            min_val = [successRates_LDA(a+1, b+1) a+1 b+1];
        end
    end
end

easy = max_val(2:3) - 1; % easiest to seperate
hard = min_val(2:3) - 1; % hardest to seperate

%% SVM - 10 Digits
[~, S, V] = svd(data, 'econ');
scale = max(S*V', [], 'all');
normalizedTrainingData = S*V' / scale;

% train model
```

```
SVMmodel = fitcecoc(normalizedTrainingData', labels');

% test on training data
labelPredictions = predict(SVMmodel, normalizedTrainingData');
results = labels == labelPredictions;
success_10digits_SVM_TRAINING = sum(results) / length(labels);

% test on test data
[~, St, Vt] = svd(test_data, 'econ');
normalizedTestData = St*Vt' / scale;

labelPredictions = predict(SVMmodel, normalizedTestData');

results = labelPredictions == t_labels;
success_10digits_SVM = sum(results) / length(t_labels);

%% SVM - 2 Digits
targets = [easy, hard];
success_2digits_SVM = zeros(1, 2);
success_2digits_SVM_TRAINING = zeros(1, 2);

for i = 1:2:4
    a = targets(i);
    b = targets(i+1);

    % Training Data Set Filtration
    ind_ab = find(labels == a | labels == b);
    data_ab = data(:, ind_ab);
    labels_ab = labels(ind_ab);

    % SVD of binary data
    [U,S,V] = svd(data_ab, 'econ');
    scale = max(S*V', [], 'all');
    normalizedTrainingData = S*V' / scale;

    % Train SVM Model
    SVMmodel = fitcsvm(normalizedTrainingData', labels_ab);

    % Test on Training Data
    labelPredictions = predict(SVMmodel, normalizedTrainingData');
    results = labels_ab == labelPredictions;
    success_2digits_SVM_TRAINING(ceil(i/2)) = sum(results) / length(labels_ab);

    % Test Data Set Filtration
    ind_ab_t = find(t_labels == a | t_labels == b);
    TestSet = test_data(:, ind_ab_t);
    labels_ab_t = t_labels(ind_ab_t);

    % SVD of binary test data
    [Ut, St, Vt] = svd(TestSet);
    normalizedTestData = St*Vt' / scale;

    % SVM Model Predictions and Success Rate
    labelPredictions = predict(SVMmodel, normalizedTestData');
```

```
    results = (labelPredictions == labels_ab_t);

    % first is easy, second is hard
    success_2digits_SVM(ceil(i/2)) = sum(results) / length(labels_ab_t);
end

%% Decision Trees - 10 digits
[~, S, V] = svd(data, 'econ');
scale = max(S*V', [], 'all');
normalizedTrainingData = S*V' / scale;

tree = fitctree(normalizedTrainingData', labels');

% test on training data
labelPredictions = predict(tree, normalizedTrainingData');
results = labels == labelPredictions;
success_10digits_tree_TRAINING = sum(results) / length(labels);

% test on test data
[~, St, Vt] = svd(test_data, 'econ');
normalizedTestData = St*Vt' / scale;

labelPredictions = predict(tree, normalizedTestData');

results = labelPredictions == t_labels;
success_10digits_tree = sum(results) / length(t_labels);



%% Decision Trees - 2 Digits
targets = [easy, hard];
success_2digits_tree = zeros(1, 2);

for i = 1:2:4
    a = targets(i);
    b = targets(i+1);

    % Training Data Set Filtration
    ind_ab = find(labels == a | labels == b);
    data_ab = data(:, ind_ab);
    labels_ab = labels(ind_ab);

    % SVD of binary data
    [U,S,V] = svd(data_ab, 'econ');
    scale = max(S*V', [], 'all');
    normalizedTrainingData = S*V' / scale;

    % Train SVM Model
    tree = fitctree(normalizedTrainingData', labels_ab);

    % Test on Training Data
    labelPredictions = predict(tree, normalizedTrainingData');
    results = labels_ab == labelPredictions;
```

```
    success_2digits_tree_TRAINING(ceil(i/2)) = sum(results) / length(labels_ab);

    % Test Data Set Filtration
    ind_ab_t = find(t_labels == a | t_labels == b);
    TestSet = test_data(:, ind_ab_t);
    labels_ab_t = t_labels(ind_ab_t);

    % SVD of binary test data
    [Ut, St, Vt] = svd(TestSet);
    normalizedTestData = St*Vt' / scale;

    % SVM Model Predictions and Success Rate
    labelPredictions = predict(tree, normalizedTestData');

    results = (labelPredictions == labels_ab_t);

    % first is easy, second is hard
    success_2digits_tree(ceil(i/2)) = sum(results) / length(labels_ab_t);
end
```

Three Index Trainer

```
function [U,S,V,threshold,w,sortb,sorta] = three_index_trainer(data, labels, a, b, c, feature)

    ind_abc = find(labels == b | labels == a | labels == c);
    data_abc = data(:, ind_abc);
    labels_abc = labels(ind_abc);

    na = length(find(labels == a));
    nb = length(find(labels == b));
    nc = length(find(labels == c));

    [U,S,V] = svd(data_abc,'econ');
    digits = S*V';
    U = U(:,1:feature);

    digits_a = digits(1:feature, find(labels_abc == a));
    digits_b = digits(1:feature, find(labels_abc == b));
    digits_c = digits(1:feature, find(labels_abc == c));

    ma = mean(digits_a,2);
    mb = mean(digits_b,2);
    mc = mean(digits_c,2);

    m_all = mean(digits);

    % scatter matrices
    Sw = 0;
    for i = 1:na
        Sw = Sw + (digits_a(:, i) - ma)*(digits_a(:, i) - ma)';
    end
    for i = 1:nb
        Sw = Sw + (digits_b(:, i) - mb)*(digits_b(:, i) - mb)';
```

```
end
for i = 1:nc
    Sw = Sw + (digits_c(:, i) - mc)*(digits_c(:, i) - mc)';
end

Sb = (ma-m_all)*(ma-m_all)'  + (mb-m_all)*(mb-m_all)'  + (mc-m_all)*(mc-m_all)';

% eigenvalues
[V2,D] = eig(Sb,Sw);
[~, ind] = max(abs(diag(D)));

temp = [abs(diag(D)) (1:length(abs(diag(D))))'];
temp = sortrows(temp, 'descend');
w1 = V2(:, temp(1, 2));
w2 = V2(:, temp(2, 2));
w3 = V2(:, temp(3, 2));

v1_a = w1'*digits_a;
v2_a = w2'*digits_a;
v3_a = w3'*digits_a;

v1_b = w1'*digits_b;
v2_b = w2'*digits_b;
v3_b = w3'*digits_b;

v1_c = w1'*digits_c;
v2_c = w2'*digits_c;
v3_c = w3'*digits_c;

% sorta_1 = sort(v1_a);
% sorta_2 = sort(v2_a);
% sortb_1 = sort(v1_b);
% sortb_2 = sort(v2_b);
% sortc_1 = sort(v1_c);
% sortc_2 = sort(v2_c);
%
% find 2 cutoff values for each dim
%
% dim 1
% v1_means = [mean(v1_a) mean(v1_b) mean(v1_c)];
% [v1_means, I] = sort(v1_means);
%
% t1 = (v1_means(1) + v1_means(2)) / 2;
% t2 = (v1_means(2) + v1_means(3)) / 2;
%
%
%
% %
% t1 = length(sortb);
% t2 = 1;
%
% while sortb(t1)>sorta(t2)
%     t1 = t1-1;
%     t2 = t2+1;
```

```
    % end
    % threshold = (sortb(t1)+sorta(t2))/2;

end
```

Two Index Trainer

```
function [U,S,V,threshold,w,sortb,sorta, accuracy] = two_index_trainer2(data, labels, a, b, feature)

    ind_ab = find(labels == a | labels == b);
    data_ab = data(:, ind_ab);
    labels_ab = labels(ind_ab);

    na = length(find(labels == a));
    nb = length(find(labels == b));

    [U,S,V] = svd(data_ab,'econ');

    digits = S*V';
    U = U(:,1:feature);

    digits_b = digits(1:feature,find(labels_ab == b));
    digits_a = digits(1:feature,find(labels_ab == a));

    mb = mean(digits_b,2);
    ma = mean(digits_a,2);

    Sw = 0;
    for k = 1:nb
        Sw = Sw + (digits_b(:,k)-mb)*(digits_b(:,k)-mb)';
    end
    for k = 1:na
        Sw = Sw + (digits_a(:,k)-ma)*(digits_a(:,k)-ma)';
    end
    Sb = (mb-ma)*(mb-ma)';

    [V2,D] = eig(Sb,Sw);
    [~, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    vb = w'*digits_b;
    va = w'*digits_a;

    if mean(vb)>mean(va) % so vb is always less than va
        w = -w;
        vb = -vb;
        va = -va;
    end

    sortb = sort(vb);
    sorta = sort(va);

    t1 = length(sortb);
```

```
    t2 = 1;

    while sortb(t1)>sorta(t2)
        t1 = t1-1;
        t2 = t2+1;
    end
    threshold = (sortb(t1)+sorta(t2))/2;

    accuracy = 1 - t2*2 / (na + nb);
end
```

Image Parser

```
function [images, labels] = mnist_parse(path_to_digits, path_to_labels)

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

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

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

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

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

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

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

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

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

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

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

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

% Close the files
fclose(fid1);
fclose(fid2);

end
```
