AMATH582 Hw4: Classifying Digits

Xiangyu Gao

Mar. 10, 2021

Abstract

In this report, we focused on the digit image classification problem and tried several state-of-art classifiers for identifying 10 different digits with their principal components representation. We operated extensive experiments and compared the performance between different algorithms.

1 Introduction and Overview

One of the most intriguing mathematical developments of the past decade concerns the ideas of image recognition. In particular, mathematically plausible methods are being rapidly developed to mimic real biological processes towards automating computers to recognize people, animals, and places [1]. The image recognition methods are firmly rooted in the basic mathematical techniques that are used for the analysis in this report.

Particularly, we focus on the digit image classification problem in this report. We perform the traditional LDA algorithm and the state-of-art SVM and decision tree algorithms towards identifying 10 different digits. To reduce input size and save computation complexity, we represent each digit image with its principal components (modes) calculated by SVD and PCA analysis.

2 Theoretical Background

2.1 PCA and SVM Analysis

The key to analyzing a PCA problem is to consider the covariance matrix:

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

where the matrix X contains the experimental data of the system. In particular, $\mathrm{X} \in \mathbb{C}^{m\times n}$ where m are the number of probes or measuring positions, and n is the number of experimental data points taken at each location. [1].

The most straightforward way to diagonalize the covariance matrix is by using eigenvectors and eigenvalues. We can observe that $\mathrm{X}\mathrm{X}^T$ is a square, symmetric $m \times m$ matrix, i.e. it is self-adjoint so that the m eigenvalues are real and distinct. Linear algebra provides theorems which state that such a matrix can be rewritten as

$$\mathbf{X}\mathbf{X}^T = \mathbf{S}\Lambda\mathbf{S}^{-1}$$

where the matrix $\mathbf{S}$ is a matrix of the eigenvectors of $\mathbf{X}\mathbf{X}^T$ arranged in columns. Since it is a symmetric matrix, these eigenvector columns are orthogonal so that ultimately the $\mathbf{S}$ can be written as a unitary matrix with $\mathrm{S}^{-1} = \mathbf{S}^T$. Recall that the matrix Λ is a diagonal matrix whose entries correspond to the m distinct eigenvalue of $\mathbf{X}\mathbf{X}^T$.

This suggests that instead of working directly with the matrix $\mathbf{X}$, we consider working with the transformed variable, or in the principal component basis,

$$\mathbf{Y} = \mathbf{S}^T\mathbf{X}$$

2.2 Linear Discrimination Analysis (LDA)

The goal of LDA is two-fold: find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data.

For a two-class LDA, the above idea results in consideration of the following mathematical formulation. Construct a projection w such that

$$\mathbf{w} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

where the scatter matrices for between-class $\mathbf{S}_B$ and within-class $\mathbf{S}_W$ data are given by

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T$$
$$\mathbf{S}_W = \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T$$

3 Algorithm Implementation and Development

We introduce the algorithm implementation details below and attach corresponding MATLAB codes in Appendix B.

The first step is to do an SVD analysis of all digit images. We read all images in the training set and test set, and reshape each image into a column vector. In this way, each column of our data matrix is a different image. We then subtract its row-wise mean from data matrix to remove the background component. The SVD is performed on the processed data matrix to obtain the U, Σ, and V matrix.

We then look for the rank of digit space k for good image reconstruction. We plot the $\|S\|^2$ as the singular value spectrum. From the spectrum, we find the number of singular values that take up to 70% variance (energy) and regard this number as the rank of digit space k. With the chosen rank, we tried to reconstruct several selected digit images for validation.

To interpret the U matrix, we visualize the first 9 columns of U by reshaping them into original image size. To interpret the V matrix, we project each image sample onto three selected V-modes (column 2, 3, and 5) on a 3D plot, and the samples are colored by their digit labels.

For the following classification tasks, we save the projection coefficient matrix V with its first k columns, where k is the number of principal components explained above.

We first pick 2 digits (0 and 1) and build a LDA classifier to identify them. The saved 26 V-modes for 2 digits and corresponding labels are concatenated along the rows. Then the concatenated vectors of train set and test set are input to a LDA classifier to get predictions. Similarly, we pick 3 digits (0 , 1, and 2) and build a LDA classifier to identify them.

To find the hardest and easiest pair of digits to separate with LDA classifier, we exhaustively find any pair of two digits from all combinations and quantify the separation accuracy on the test set.

We then adopt the state-of-art SVM and decision tree classifier to do the separation between all then digits. At the end, we compare the performance (training accuracy and teasing accuracy) between LDA, SVM and decision trees on the hardest and easiest pair of digits to separate.

4 Computational Results

We plot the first 50 singular values and show the obtained spectrum in Fig. 1. By counting the number of singular values that would take up to 70% energy in the spectrum, we determine that 26 modes is necessary for good image reconstruction. To validate above inference, we try to reconstruct one digit image with selected 26 modes. The reconstruction results compared to the original digit image are shown in Fig. 2.

The SVD analysis provides three metrics: U, Σ, and V. Particularly, columns of U represent different components/base of the digit space. We visualize the first 9 columns of U (the principal components) in Fig. 3. The diagonal elements of Σ are singular values, which can represent the importance of corresponding base. The i^{th} column of V represents the projection of digit images onto the i^{th} mode/basis (column of U). We plot the selected 3 columns of V (column 2, 3, and 5) in Fig. 4, and label the color of image samples by their digit labels.

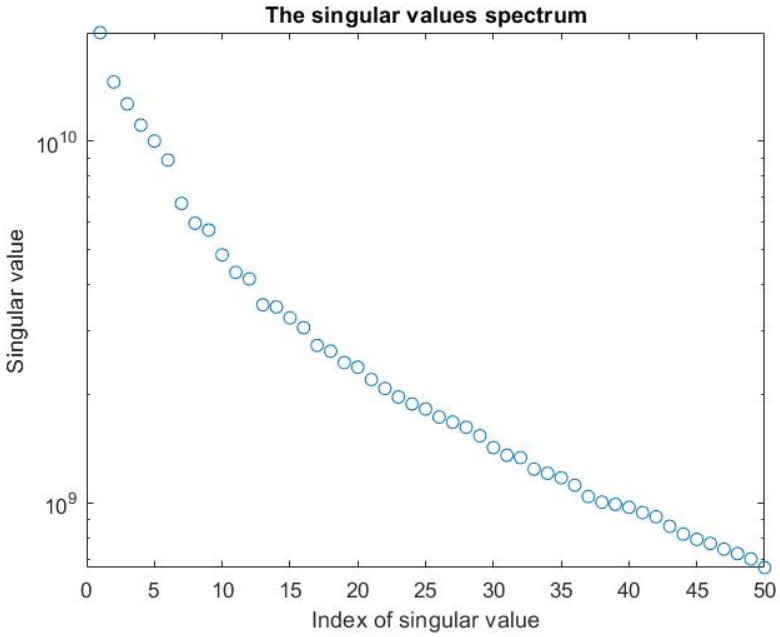

Figure 1: The singular value spectrum (first 50 singular values) of all digit images.

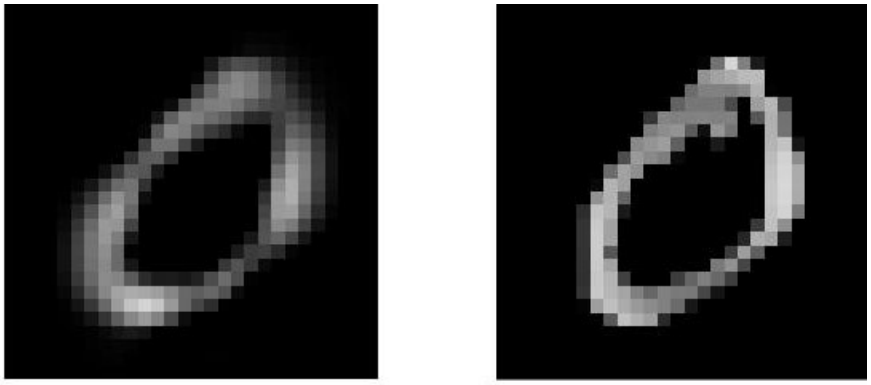

Figure 2: Left: the reconstructed digit image example with selected 26 modes; Right: the original digit image.

Figure 3: The visualization of the first 9 columns of U.

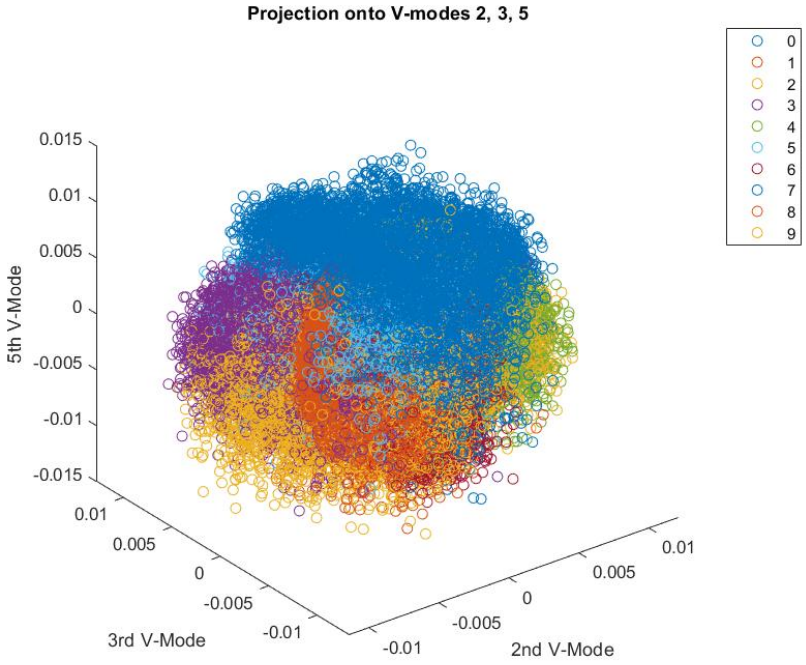

Figure 4: The projection of digit images onto three selected V-modes.

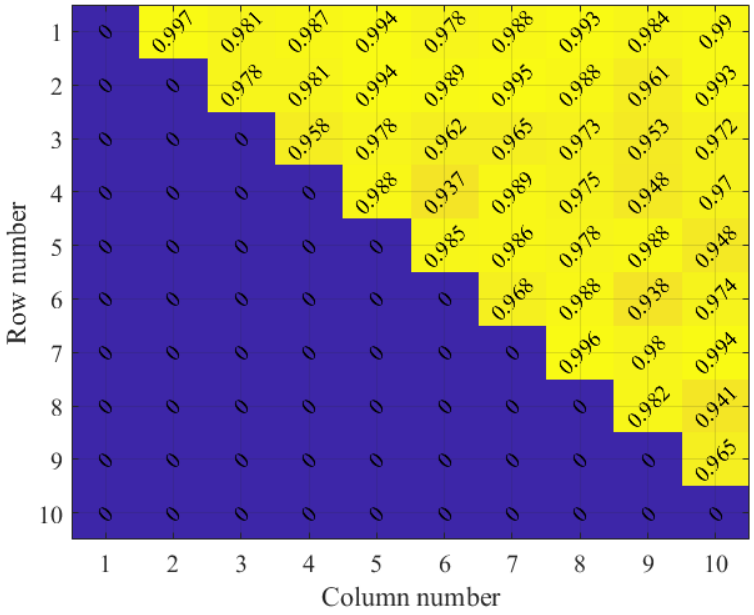

Figure 5: The visualization of training accuracy for all experiments. For example, the value 0.997 at row 1 and column 2 represent the training accuracy for classifying digit 0 and digit 1 with LDA, and so on.

	LDA	SVM	Decision Tree
Training accuracy	94.79%	50.67%	98.43%
Testing accuracy	94.12%	50.78%	89.65%

Table 1: Performance comparison between LDA, SVM and decision tree on the hardest pair of digits to separate.

For the classification of two digits (0 and 1), the implemented LDA classifier has 99.69% training accuracy and 99.86% testing accuracy. The results reveal that digit 0 and 1 are pretty easy to separate.

For the classification of three digits (0, 1, and 2), the implemented LDA classifier has 96.57% training accuracy and 96.41% testing accuracy. The results reveal that there is a little overfitting happening during the training process. Besides, digit 0, 1 and 2 are easy to separate.

We perform the LDA classifier for all pairs of two digits and try to find the hardest and easiest pair. The training accuracy and testing accuracy of all experiments are shown in Fig. 5 and Fig. 6. From the testing accuracy matrix, we can conclude that the easiest pair for separation are digit 0 and digit 1 with accuracy 0.999, while the hardest pair for separation are digit 4 and digit 9 with accuracy 0.941.

With the state-of-art SVM classifier, we get 47.24% training accuracy and 49.08% testing accuracy for identifying all 10 digits. Besides, with the state-of-art decision classifier, we get 95.88% training accuracy and 84.89% testing accuracy for identifying all 10 digits. Based on results, we know that the decision tree gets better performance than SVM though there is overftting during the training process. On the other hand, SVM has bad performance since it under-fits the model.

The comparison results between LDA, SVM, and decision tree classifier on the hardest pair (digit 4 and digit 9) are shown in Table. 1, and the same comparison on the easiest pair (digit 0 and digit 1) are presented in Table. 2. From the results of hardest pair separation, we know that LDA performs best on the testing set while Decision tree gets best accuracy on the training set. The performance of SVM is much lower than the other two. From the comparison result on easiest pair, we can tell that both LDA and decision tree classifier obtain very high training accuracy and testing accuracy. The performance of SVM is still worse than the other two, but improves a lot compared to that for the hardest pair separation.

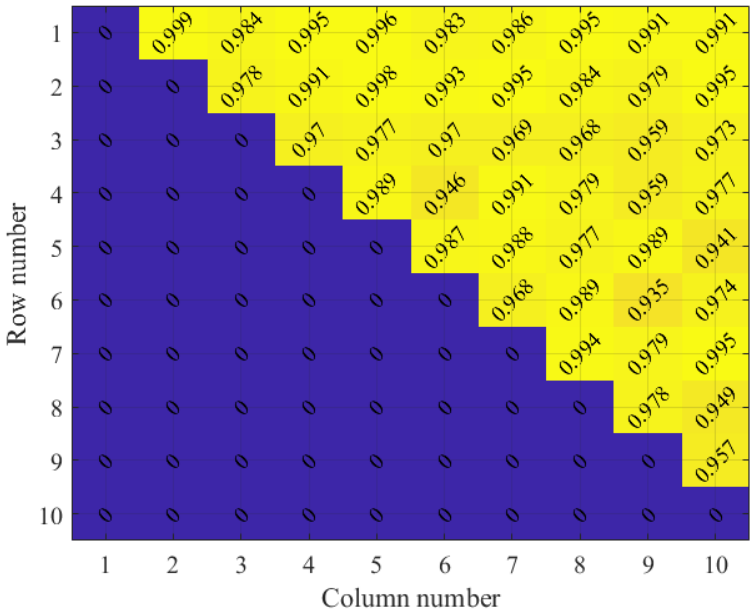

Figure 6: The visualization of testing accuracy for all experiments. For example, the value 0.999 at row 1 and column 2 represent the testing accuracy for classifying digit 0 and digit 1 with LDA, and so on.

	LDA	SVM	Decision Tree
Training accuracy	99.72%	89.27%	99.92%
Testing accuracy	99.91%	89.08%	99.81%

Table 2: Performance comparison between LDA, SVM and decision tree on the easiest pair of digits to separate.

5 Summary and Conclusions

In this report, we focused on the digit image classification problem and tried several state-of-art classifiers for identifying 10 different digits with their principal components representation. We operated extensive experiments and compared the performance between different algorithms.

References

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

Appendix A MATLAB Functions

The important MATLAB functions and thier implementation explanations are listed here:

- `CLASS = classify(SAMPLE,TRAINING,GROUP)` classifies each row of the data in SAMPLE into one of the groups in TRAINING. SAMPLE and TRAINING must be matrices with the same number of columns. GROUP is a grouping variable for TRAINING.
- `OBJ=fitcecoc(X,Y)` is an alternative syntax that accepts X as an N-by-P matrix of predictors with one row per observation and one column per predictor. Y is the response and is an array of N class labels.
- `TREE=fitctree(X,Y)` is an alternative syntax that accepts X as an N-by-P matrix of predictors with one row per observation and one column per predictor. Y is the response and is an array of N class labels.
- `MODEL=fitcsvm(X,Y)` is an alternative syntax that accepts X as an N-by-P matrix of predictors with one row per observation and one column per predictor. Y is the response and is an array of N class labels.

Appendix B MATLAB Code

The MATLAB codes for all problems are shown here. We also stored corresponding codes on Github under the repository (https://github.com/Xiangyu-Gao/AMATH-582-Computational-method-for-data-analysis/tree/main/Hw4).

```
clear all
close all
clc

%% Problem 1
% Load images
[images_train, labels_train] = mnist_parse('mnist\train-images.idx3-ubyte', ...
    'mnist\train-labels.idx1-ubyte');
[images_test, labels_test] = mnist_parse('mnist\t10k-images.idx3-ubyte', ...
    'mnist\t10k-labels.idx1-ubyte');

images = cat(3, images_train(:,:,1:50000), images_test);
labels = cat(1, labels_train(1:50000,:), labels_test);

% Size of each picture
m = 28;
n = 28;
% reshape images
A = double(reshape(images, [m*n, length(labels)]));

%% Problem 2
% remove the averaged image (row-wise)
mean_data = mean(A,2);
A = A - mean_data;

% Computing the SVD
[U,S,V] = svd(A, 0);

% PCA analysis and plot singular value spectrum
sigval_spct = diag(S).^2;
ALL_energy = sum(sigval_spct);
sum(sigval_spct(1:26)) / ALL_energy % Rank 26, %70 variance

figure(1)
plot(sigval_spct(1:50), 'o')
set(gca, 'YScale', 'log')
xlabel('Index of singular value')
ylabel('Singular value')
title('The singular values spectrum')

% Reconstruct images with PCA basis
num_PCAbasis = 26;
image_id = 2;
image1 = U(:, 1:num_PCAbasis)*S(1:num_PCAbasis, 1:num_PCAbasis)*V(image_id, 1:num_PCAbasis)';
figure(2)
subplot(1,2,1)
imshow(uint8(reshape(image1,m,n)))
subplot(1,2,2)
imshow(uint8(reshape(A(:, image_id),m,n)))
```

Listing 1: MATLAB codes for Problem 1,2.

```
%% Problem 3
% interpretate the U, S, V matrix
numImg = size(A, 2);
Phi = U;
Phi(:,1) = -1*Phi(:,1);
% plot the first 9 reshaped coumns of matrix U
figure(3)
count = 1;
for i=1:3
    for j=1:3
        subplot(3,3,count)
        imshow(uint8(25000*reshape(Phi(:,count),m,n)));
        count = count + 1;
    end
end

%% Problem 4
% Projection onto 3 V-modes
figure(4)
for label=0:9
    label_indices = find(labels == label);
    plot3(V(label_indices, 2), V(label_indices, 3), V(label_indices, 5),...
        'o', 'DisplayName', sprintf('%i',label), 'Linewidth', 0.5)
    hold on
end
xlabel('2nd V-Mode'), ylabel('3rd V-Mode'), zlabel('5th V-Mode')
title('Projection onto V-modes 2, 3, 5')
legend
set(gca,'Fontsize', 10)

%% Save projection onto V-modes for classification training, testing
V_save = V(:, 1:num_PCAbasis);
save('V_mode.mat', 'V_save')
save('labels.mat', 'labels')
```

Listing 2: MATLAB codes for Problem 3,4.

```
clc
clear all
close all

% read saved data
load('V_mode.mat');
load('labels.mat');
V_train = V_save(1:50000, :);
V_test = V_save(50001:end, :);
labels_train = labels(1:50000, :);
labels_test = labels(50001:end, :);

%% Linear Discriminate classifier (LDA) for classification of two digits
digit1 = 0;
digit2 = 1;

digit1_train_idx = find(labels_train == digit1);
digit2_train_idx = find(labels_train == digit2);
digit1_test_idx = find(labels_test == digit1);
digit2_test_idx = find(labels_test == digit2);

x_test = [V_test(digit1_test_idx, :); V_test(digit2_test_idx, :)];
x_train = [V_train(digit1_train_idx, :); V_train(digit2_train_idx, :)];
ctest = [labels_test(digit1_test_idx, :); labels_test(digit2_test_idx, :)];
ctrain = [labels_train(digit1_train_idx, :); labels_train(digit2_train_idx, :)];

% LDA classifier
pre = classify([x_test; x_train], x_train, ctrain);
% calculate accuracy for testing and training
accu_test = sum(pre(1:length(x_test)) == ctest)/length(x_test);
accu_train = sum(pre(length(x_test)+1:end) == ctrain)/length(x_train);

%% Linear Discriminate classifier (LDA) for classification of thress digits
digit1 = 0;
digit2 = 1;
digit3 = 2;

digit1_train_idx = find(labels_train == digit1);
digit2_train_idx = find(labels_train == digit2);
digit3_train_idx = find(labels_train == digit3);
digit1_test_idx = find(labels_test == digit1);
digit2_test_idx = find(labels_test == digit2);
digit3_test_idx = find(labels_test == digit3);

x_test = [V_test(digit1_test_idx, :); V_test(digit2_test_idx, :); ...
    V_test(digit3_test_idx, :)];
x_train = [V_train(digit1_train_idx, :); V_train(digit2_train_idx, :); ...
    V_train(digit3_train_idx, :)];
ctest = [labels_test(digit1_test_idx, :); labels_test(digit2_test_idx, :); ...
    labels_test(digit3_test_idx, :)];
ctrain = [labels_train(digit1_train_idx, :); labels_train(digit2_train_idx, :); ...
    labels_train(digit3_train_idx, :)];

% LDA classifier
pre = classify([x_test; x_train], x_train, ctrain);
% calculate accuracy for testing and training
accu_test = sum(pre(1:length(x_test)) == ctest)/length(x_test);
accu_train = sum(pre(length(x_test)+1:end) == ctrain)/length(x_train);
```

Listing 3: MATLAB codes for LDA classifier.

```
%% SVM classifier
% SVM classifier with training data, labels and test set
x_train = V_train;
ctrain = labels_train;
x_test = V_test;
ctest = labels_test;

Mdl = fitcecoc(x_train, ctrain);
test_pre = predict(Mdl, x_test);
train_pre = predict(Mdl, x_train);

accu_test = sum(test_pre == ctest)/length(x_test);
accu_train = sum(train_pre == ctrain)/length(x_train);

%% Decision tree classifier
% classification tree on fisheriris data
tree = fitctree(x_train, ctrain);
view(tree,'mode','graph');

test_pre = predict(tree, x_test);
train_pre = predict(tree, x_train);

accu_test = sum(test_pre == ctest)/length(x_test);
accu_train = sum(train_pre == ctrain)/length(x_train);
```

Listing 4: MATLAB codes for SVM and Decision tree classifier.