AMATH 482 Homework 4

Yiqing He

March 9th, 2021

Abstract

This report performs an analysis of two training data set. It starts with have the data projected into PCA space and then build a classifier to identify individual digits in the training set.

Introduction and Overview

At the beginning of the project, I did an SVD analysis of the digit images, including reshaping each image into a column vector and find the **U, Σ** and **V**. The next step involves the building of a classifier to identify individual digits in the training set. In this step, I pick different digits and build linear classifier to find which one is the most difficult to separate.

Theoretical Background

The general setting framework for Independent Component Analysis (ICA) is established in a similar manner to

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t)$$
$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t)$$

where $x_1(t)$ and $x_2(t)$ are mixed, recorded signals at each microphone.

Suppose N distinct linear combinations of N signals or data sets are given. Then, the original N signals or data sets could be determined.

$$x_j(t) = a_{j1}s_1(t) + a_{j2}s_2(t) + \cdots + a_{jN}s_N(t), \quad 1 \le j \le N.$$

Finally, the image could be reconstructed through the following linear algebra.

$$\begin{aligned}\mathbf{s} &= \mathbf{A}^{-1}\mathbf{x} = \mathbf{V}\Sigma^{-1}\mathbf{U}^{*}\mathbf{x} \\ &= \begin{bmatrix} \cos(\phi_0) & \sin(\phi_0) \\ -\sin(\phi_0) & \cos(\phi_0) \end{bmatrix} \begin{bmatrix} 1/\sqrt{\sigma_1} & 0 \\ 0 & 1/\sqrt{\sigma_2} \end{bmatrix} \begin{bmatrix} \cos(\theta_0) & \sin(\theta_0) \\ -\sin(\theta_0) & \cos(\theta_0) \end{bmatrix} \mathbf{x}.\end{aligned}$$

This project includes the implementation of LDA. The goal of LDA is to find a suitable projection that maximizes the distance between the interclass data while minimizing the intra-class data.

Implementing LDA including finding a vector **w** such that

$$\mathbf{w} = \operatorname{argmax} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}.$$

Algorithm Implementation and Development

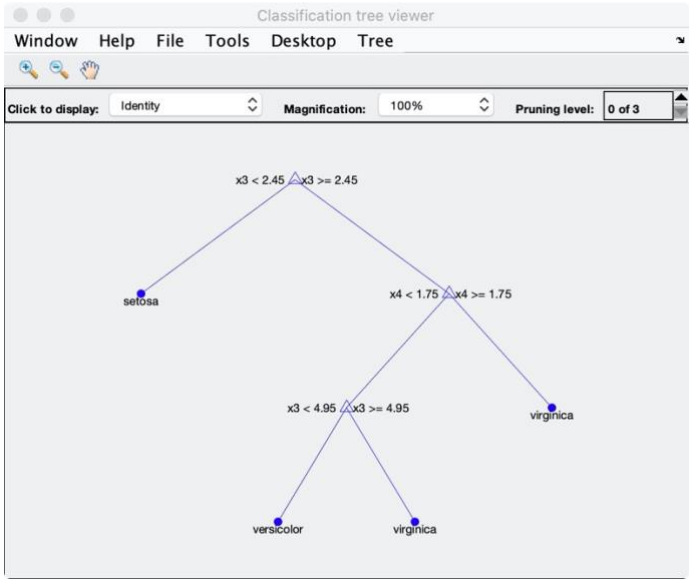
During the first part, I reshaped each image into a column vector and then constructed a singular value spectrum. Through this part, I determined the number of modes that are necessary for good image reconstruction. At the second part, I picked two and three digits each time to find out which two digits are most difficult and easiest to separate. After that, I compared the performance between LDA, SVM and decision trees on the hardest and eaiest pair of digits to separate.

Finally, I used the built-in function to implement SVM and decision tree classifiers.

Computational Results

The accuracy of the test result is 21.428571%. The class error is 6%.

The classification tree is displayed as below.



Summary and Conclusions

For plot3, I tried the following code from slack for V is 60000 by 784, but still have not figured out how to plot it.

```
for labels=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', 2)
    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', 14)
```

Σ are the singular values. **U** are the left singular values. **V** are the right singular vectors.

The accuracy of this test is not high enough. I can only reasonably identify about 21% of them.

Appendix A

I2 = im2double(I) converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a binary image. im2double rescales the output from integer data types to the range [0, 1].

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.

L = kfoldLoss(obj) returns loss obtained by cross-validated classification model obj. For every fold, this method computes classification loss for in-fold observations using a model trained on out-of-fold observations.

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.

p = randperm(n) returns a row vector containing a random permutation of the integers from 1 to n without repeating elements.

Appendix B

Training File

```
clear variables; clc; close all
[images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
images = im2double(images);
y = zeros(10,60000); %Correct outputs vector
for i = 1:60000
    y(labels(i)+1,i) = 1;
end

images = images(:,2:785)/255;

images = images'; %Input vectors

hn1 = 80; %Number of neurons in the first hidden layer
hn2 = 60; %Number of neurons in the second hidden layer

%Initializing weights and biases
w12 = randn(hn1,784)*sqrt(2/784);
w23 = randn(hn2,hn1)*sqrt(2/hn1);
w34 = randn(10,hn2)*sqrt(2/hn2);
b12 = randn(hn1,1);
```

```
b23 = randn(hn2,1);
b34 = randn(10,1);

%learning rate
eta = 0.0058;

%Initializing errors and gradients
error4 = zeros(10,1);
error3 = zeros(hn2,1);
error2 = zeros(hn1,1);
errortot4 = zeros(10,1);
errortot3 = zeros(hn2,1);
errortot2 = zeros(hn1,1);
grad4 = zeros(10,1);
grad3 = zeros(hn2,1);
grad2 = zeros(hn1,1);

epochs = 50;

m = 10; %Minibatch size

for k = 1:epochs %Outer epoch loop

    batches = 1;

    for j = 1:60000/m
        error4 = zeros(10,1);
        error3 = zeros(hn2,1);
        error2 = zeros(hn1,1);
        errortot4 = zeros(10,1);
        errortot3 = zeros(hn2,1);
        errortot2 = zeros(hn1,1);
        grad4 = zeros(10,1);
        grad3 = zeros(hn2,1);
        grad2 = zeros(hn1,1);
    for i = 28 %Loop over each minibatch

    %Feed forward
    a1 = images(:,i);
    z2 = w12*a1 + b12;
    a2 = elu(z2);
    z3 = w23*a2 + b23;
    a3 = elu(z3);
    z4 = w34*a3 + b34;
    a4 = elu(z4); %Output vector

    %backpropagation
    error4 = (a4-y(:,i)).*elup(z4);
    error3 = (w34'*error4).*elup(z3);
    error2 = (w23'*error3).*elup(z2);

    errortot4 = errortot4 + error4;
    errortot3 = errortot3 + error3;
    errortot2 = errortot2 + error2;
    grad4 = grad4 + error4*a3';
```

```
    grad3 = grad3 + error3*a2';
    grad2 = grad2 + error2*a1';

    end

    %Gradient descent
    w34 = w34 - eta/m*grad4;
    w23 = w23 - eta/m*grad3;
    w12 = w12 - eta/m*grad2;
    b34 = b34 - eta/m*errortot4;
    b23 = b23 - eta/m*errortot3;
    b12 = b12 - eta/m*errortot2;

    batches = batches + m;

    end
    fprintf('Epochs:');
    disp(k) %Track number of epochs
    [images,y] = shuffle(images,y); %Shuffles order of the images for next
epoch
end

disp('Training done!')
%Saves the parameters
save('wfour.mat','w34');
save('wthree.mat','w23');
save('wtwo.mat','w12');
save('bfour.mat','b34');
save('bthree.mat','b23');
save('btwo.mat','b12');

function fr = elu(x)
    f = zeros(length(x),1);
    for i = 1:length(x)
    if x(i)>=0
        f(i) = x(i);
    else
        f(i) = 0.2*(exp(x(i))-1);
    end
    end
    fr = f;
end

function fr = elup(x)
    f = zeros(length(x),1);
    for i = 1:length(x)
    if x(i)>=0
        f(i) = 1;
    else
        f(i) = 0.2*exp(x(i));
    end
    end
    fr = f;
end
```

```
function [B,v] = shuffle(A,y)
cols = size(A,2);
P = randperm(cols);
B = A(:,P);
v = y(:,P);
end
```

Test File

```
[images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-
ubyte');
images = im2double(images);
y = zeros(10,60000); %Correct outputs vector
for i = 1:60000
    y(labels(i)+1,i) = 1;
end

images = images(:,2:785)/255;

images = images'; %Input vectors

we34 = matfile('wfour.mat');
w4 = we34.w34;
we23 = matfile('wthree.mat');
w3 = we23.w23;
we12 = matfile('wtwo.mat');
w2 = we12.w12;
bi34 = matfile('bfour.mat');
b4 = bi34.b34;
bi23 = matfile('bthree.mat');
b3 = bi23.b23;
bi12 = matfile('btwo.mat');
b2 = bi12.b12;
success = 0;
n = 28;

for i = 1:n
out2 = elu(w2*images(:,i)+b2);
out3 = elu(w3*out2+b3);
out = elu(w4*out3+b4);
big = 0;
num = 0;
for k = 1:10
    if out(k) > big
        num = k-1;
        big = out(k);
    end
end

if labels(i) == num
    success = success + 1;
end
```

```
end

fprintf('Accuracy: ');
fprintf('%f',success/n*100);
disp(' %');

% classification tree on fisheriris data
load fisheriris;
tree=fitctree(meas,species,'MaxNumSplits',3,'CrossVal','on');
view(tree.Trained{1},'Mode','graph');
classError = kfoldLoss(tree)

% SVM classifier with training data, labels and test set
Mdl = fitcsvm(xtrain,label);
test labels = predict(Mdl,test);
```