AMATH 482 Homework 4

Jonah Wu

March 10, 2021

Abstract

We are going to use the MNIST data set, which is a collection of handwritten numerical digits (0 - 9), to perform machine learning training to classify different digits using Linear Discriminant Analyzer (LDA), Support Vector Machine (SVM) and Binary Decision Tree with MATLAB. In addition, we are going to find which two digits in the data set appear to be the most difficult to separate, and another pair that are most easy to separate.

1 Introduction and Overview

Machine Learning (ML) is one of the hottest topics in the world where there exist many applications of ML to complete tasks that could be accomplished by a human or not by human due to large size of data. In this assignment, Linear Discriminant Analysis (LDA) was used to accomplish the task of training a model that can classify different hand-written digits. Given pictures and labels, Machine Learning is unbiased where it only follows the pattern from the input data set. The two main types of machine learning models are supervised and unsupervised, and in this unbiased case, supervised learning was used to train on the pre-labelled data. In addition, we are going to use and compare 3 different classification models for training the model for classifying digits: linear discriminant analysis, support vector machines, and binary decision trees.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

The SVD is a form of factoring a real or complex matrix that generalizes the eigen decomposition of a square normal matrix to any $m \times n$ matrix. Singular value decomposition of the matrix $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V^*} \tag{1}$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are unitary matrices, and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal.
The values σ_n on the diagonal of Σ are called the singular values of the matrix $\mathbf{A}$. The number of non-zero singular values is equal to the rank of $\mathbf{A}$. The vectors u_n which make up the columns of $\mathbf{U}$ are called the left singular vectors of $\mathbf{A}$. The vectors v_n which make up the columns of $\mathbf{V}$ are called the right singular vectors of $\mathbf{A}$.

2.2 Linear Discriminant Analysis (LDA)

In order to classify different categories, we use LDA to find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data. We can first find the **between-class scatter matrix**:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{2}$$

where μ is the mean of each group. **Between-class scatter matrix** is a measure of the variance between the groups (between the means). Next, we can define the **within-class scatter matrix**:

$$S_w = \sum_{j=1}^{2} \sum_{x} (x - u_j)(x - u_j)^T \tag{3}$$

Within-class scatter matrix is a measure of the variance within each group. Finally, our end goal is to find the weight vector **w**:

$$w = \text{argmax} \frac{w^T S_B w}{w^T S_w w} \tag{4}$$

In order to distinguish different labels better, we want to maximize the distance between classes while minimizing distance within class. It turns out that the vector **w** that maximizes the above quotient is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem:

$$S_B w = \lambda S_w w \tag{5}$$

With this fact, we can easily perform this calculation in MATLAB.

3 Algorithm Implementation and Development

With the original file, we will first reshape the data into what we need and then perform the SVD in order to get information about the principal components. After we know which principal components are useful based on the singular value energies, we then use those components to train our model by calculating the between-class scatter matrix and the within-class scatter matrix in order to perform LDA on our data. Finally, with the weighted vectors we trained, we then can test the accuracy on the testing data set. Finally, we will perform SVM and Binary Decision Tree to compare the 3 different classifying methods.

Algorithm 1: Classifying

Import training and testing data
Reshape data
SVD on the training data
Calculate weighted vectors for the model
Use the model to classify the testing data
Compare results with testing labels to find accuracy

In the "Calculate weighted vectors for the model" part, LDA is used to compare all pairs among the digits one by one with the between-class scatter matrix and the within-class scatter matrix by using `digits-trainer()` in my MATLAB code. SVM and Binary Decision Tree methods are performed using MATLAB build in functions: `fitcecoc()` and `fitctree()`.

4 Computational Results

4.1 Singular Value Decomposition

After we perform SVD on our training data set, we will have 3 matrices: **U**, Σ, and **V**. **U** gives us the left singular vectors in which it represents the principal components. Σ gives us a diagonal with singular values. **V** gives us the right singular values that tells us how each of the images is represented in the Principal Component Analysis basis. Next, we plot the singular value spectrum to help us determine how many modes are necessary for good image reconstruction.

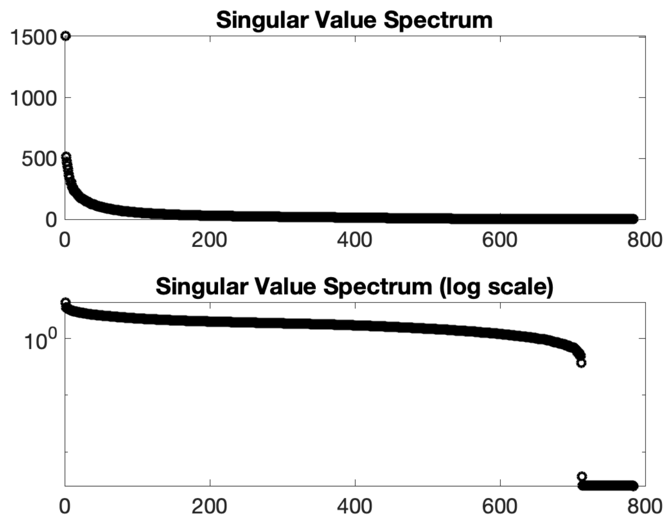

Figure 1: Singular Value Spectrum

As we can see from the top graph, the first principal component is dominating; however, we have almost 800 components to determine so we can scale it in log to know which ones matter. It turns out that after 700th component, there is a significant drop. Although the drop happens at 700, it does not necessarily mean all 700 components are needed. I calculated the energy with each components and found that the first 60 components cover more than 90% of the energy, so I choose 60 as the number of features that are necessary for good image reconstruction in this case.

4.2 Projection onto 3 selected V-modes

On a 3D plot, we project onto three selected V-modes (columns 2, 3, and 5) colored by their digit label.

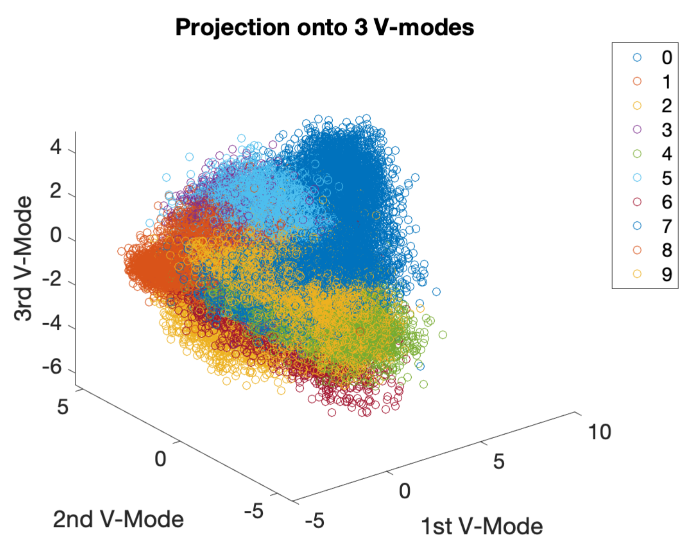

Figure 2: Projection onto 3 V-modes

4.3 Linear Discriminant Analysis (LDA)

In my MATLAB code in **Appendix B**, when I am performing LDA on the digits, I have a 10×10 matrix `accuracies` that stores the accuracy of each pairs from digits 0 to 9. There is no index 0 for MATLAB so index 1 means 0, and index 10 means 9, etc.

accuracies

10x10 double

	1	2	3	4	5	6	7	8	9	10
1	0	0.9981	0.9876	0.9930	0.9985	0.9882	0.9897	0.9935	0.9923	0.9930
2	0	0	0.9852	0.9907	0.9962	0.9921	0.9947	0.9921	0.9801	0.9935
3	0	0	0	0.9740	0.9772	0.9751	0.9739	0.9738	0.9716	0.9838
4	0	0	0	0	0.9925	0.9595	0.9919	0.9764	0.9647	0.9777
5	0	0	0	0	0	0.9883	0.9876	0.9871	0.9898	0.9528
6	0	0	0	0	0	0	0.9719	0.9885	0.9528	0.9826
7	0	0	0	0	0	0	0	0.9970	0.9855	0.9964
8	0	0	0	0	0	0	0	0	0.9820	0.9573
9	0	0	0	0	0	0	0	0	0	0.9702
10	0	0	0	0	0	0	0	0	0	0

Figure 3: Accuracy of each pair of digit

As we can see from the above table, row 1 and column 5 has the highest accuracy of 0.9985, and the lowest accuracy of 0.9528 at row 5 and index 10. This tells us that 0 and 4 appear to be the most easy to separate, and 4 and 9 are the most difficult to separate. Overall, LDA has an accuracy of 98.32% on the testing data set.

4.4 Support Vector Machines (SVM)

We first calculate the accuracy on the whole data set of 10 digits using SVM and found out that the accuracy is 93.4%. We then calculate its performance on the hardest pair 4 and 9 with an accuracy of 96.38%. while its performance on the easiest pair 0 and 4 is 99.85%

4.5 Decision Tree Classifiers

The classification tree has an accuracy of 84.87% when classifying between all 10 digits. For the hardest pair 4 and 9, the tree has an accuracy of 45.16% on the test data for 4 and 9, while having an accuracy of 98.98% on the easiest pair 0 and 4.

5 Summary and Conclusions

As we can see the table below, LDA and SVM generally have a better accuracy than the Decision Tree Classifier. In addition, when it comes to the most difficult pair to distinguish: 4 and 9, the Decision Tree Classifier has an accuracy below 50%. Through this assignment, we are able to see applications of machine learning using classifiers such as LDA, SVM, and decision trees. It is really easy for us human to recognize a digit from a hand-written digit image, but what if we need to do this for a large amount of images in a short period of time. We use computers to help us; however, computers do not have eyes and life experience like us to recognize hand-written digits. Machine Learning comes in to help computers to train models with whatever data and labels we give the computers to learn. Overall, we did 3 unbiased supervised machine learning classifier in order to classify different digits with a accuracy of 98.32% (LDA), 93.4% (SVM), and 84.87% (Decision Tree).

accuracy	LDA	SVM	Tree
between all digits	98.32%	93.4%	84.87%
between 0 and 4	99.85%	99.85%	98.98%
between 4 and 9	95.28%	96.38%	45.16%

Appendix A MATLAB Functions

- `size(A)` returns a row vector whose elements are the lengths of the corresponding dimensions of A.
- `[U, S, V]` = `svd(X)`: performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `B = repmat(A,n)` returns an array containing n copies of A in the row and column dimensions. The size of B is size(A) × n when A is a matrix.
- `abs(X)` returns the absolute value of a given input X.
- `diag(X)` returns the diagonal entries of a given a matrix X.
- `eig(X)` returns a column vector of eigenvalues corresponding to the matrix X.
- `fitecoc(X,Y)` returns a trained, multi-class support vector machine, given data X and labels Y.
- `fitctree(X,Y)` returns a trained, multi-class binary decision classification tree, given data X and labels Y.
- `im2double(X)` returns an image converted to double precision, given an image X.
- `mean(X, n)` returns a vector of mean along the n dimension of a given matrix X.
- `norm(X)` returns the Euclidean norm of a vector or matrix X.
- `sort(X)` returns a sorted array of input X, in ascending array.
- `mnist-parse(X)` returns given files downloaded from MNIST.

Appendix B MATLAB Code

```
%% Load data and reshape

[images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
[t_images, t_labels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');

training = zeros(784,60000);
for i = 1:60000
    training(:,i) = im2double(reshape(images(:,:,i),784,1));
end

testing = zeros(784,10000);
for i = 1:10000
    testing(:,i) = im2double(reshape(t_images(:,:,i),784,1));
end


%% Singular Value Decomposition

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

figure()
subplot(2,1,1)
plot(diag(S),'ko','Linewidth',2)
title("Singular Value Spectrum")
set(gca,'Fontsize',16)
subplot(2,1,2)
semilogy(diag(S),'ko','Linewidth',2)
title("Singular Value Spectrum (log scale)")
set(gca,'Fontsize',16)
saveas(gcf, 'spectrum.png')

```

```
%% Projection onto 3 V-modes

projections = U(:,[2,3,5])'*training;
for digit=0:9
    projection = projections(:,labels == digit);
    plot3(projection(1,:),projection(2,:),projection(3,:),'o', ...
        'DisplayName', sprintf('%i',digit))
    hold on
end
title('Projection onto 3 V-modes')
xlabel('1st V-Mode')
ylabel('2nd V-Mode')
zlabel('3rd V-Mode')
legend
set(gca,'Fontsize',16)
saveas(gcf, 'projection.png')


%% Running LDA on all pairs

feature = 60;
accuracies=zeros(10,10);
correct = 0;
count = 0;
for i=0:8
    for j=i+1:9
        digit1 = training(:,labels==i);
        digit2 = training(:,labels==j);
        [U,¬,¬,threshold,w,¬,¬] = digits_trainer(digit1,digit2,feature);

        test1 = testing(:,t_labels==i);
        match=0;
        length1=size(test1,2);
        for k=1:length1
            digit = test1(:,k);
            IMat = U' * digit;
            digitval = w' * IMat;
            if digitval < threshold
                match = match + 1;
            end
        end

        test2 = testing(:,t_labels==j);
        length2 = size(test2,2);
        for k=1:size(test2 ,2)
            digit = test2(:,k);
            IMat = U' * digit;
            digitval = w' * IMat;
            if digitval > threshold
                match = match + 1;
            end
        end

        accuracy = match/(length1+length2);
        accuracies(i+1,j+1) = accuracy;
        count = count+length1+length2;
        correct = correct+match;
    end
end
LDA_accuracy = correct / count


%% Other Classifiers

[U,S,V] = svd(training,'econ');
U=U(:,1:60);
projection = S*V';
```

```
train = (U'*training)'./max(projection(:));
test = (U'*testing)'./max(projection(:));

% SVM (support vector machines)
Mdl = fitcecoc(train,labels);
result = predict(Mdl,test);
match = result == t_labels;
SVM_accuracy = sum(match)/size(match,1)

% Decision Tree Classifiers
d_tree = fitctree(train,labels);
result = predict(d_tree,test);
match = result == t_labels;
tree_accuracy = sum(match)/size(match,1)


%% SVM (support vector machines)

train_data=train';
test_data=test';

train_0 = train_data(:,labels==0);
train_4 = train_data(:,labels==4);
train_9 = train_data(:,labels==9);

test_0 = test_data(:,t_labels==0);
test_4 = test_data(:,t_labels==4);
test_9 = test_data(:,t_labels==9);

label_0 = zeros(1,size(train_0,2));
label_4 = zeros(1,size(train_4,2)) + 4;
label_9 = zeros(1,size(train_9,2)) + 9;

t_label_0 = zeros(1,size(test_0 ,2));
t_label_4 = zeros(1,size(test_4 ,2)) + 4;
t_label_9 = zeros(1,size(test_9 ,2)) + 9;


train_04 = [train_0 train_4];
test_04 = [test_0 test_4];
label_04 = [label_0 label_4];
t_label_04 = [t_label_0 t_label_4];

Mdl_04 = fitcsvm(train_04',label_04);
results = predict(Mdl_04,test_04');
match = results == t_label_04';
SVM_easy_accuracy = sum(match)/size(match,1)


train_49 = [train_4 train_9];
test_49 = [test_4 test_9];
label_49 = [label_4 label_9];
t_label_49 = [t_label_4 t_label_9];

Mdl_49 = fitcsvm(train_49',label_49);
results = predict(Mdl_49,test_49');
match = results == t_label_49';
SVM_hard_accuracy =sum(match)/size(match,1)


%% Decision Tree Classifiers

d_tree1 = fitctree(train_04',label_04);
results = predict(d_tree1 ,test_04');
match = results == t_label_04';
tree_easy_accuracy = sum(match)/size(match,1)

d_tree2 = fitctree(train_49', label_49);
```

```
results = predict(d_tree2 ,test_04');
match = results == t_label_04';
tree_hard_accuracy = sum(match)/size(match,1)


%% digits_trainer function

function [U,S,V,threshold,w,sort1,sort2] = digits_trainer(d1,d2,feature)
    n1 = size(d1,2) ;
    n2 = size(d2,2) ;
    [U,S,V] = svd([d1 d2],'econ');
    digits = S*V';
    U = U(:,1:feature);
    digit1 = digits(1:feature,1:n1);
    digit2 = digits(1:feature,n1+1:n1+n2);
    ma = mean(digit1,2);
    mb = mean(digit2,2);

    Sw = 0;
    for k=1:n1
        Sw = Sw + (digit1(:,k)-ma)*(digit1(:,k)-ma)';
    end
    for k=1:n2
        Sw = Sw + (digit2(:,k)-mb)*(digit2(:,k)-mb)';
    end
    Sb = (ma-mb)*(ma-mb)';

    [V2,D] = eig(Sb,Sw);
    [¬,ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w / norm(w,2) ;
    v1 = w'*digit1;
    v2 = w'*digit2;

    if mean(v1) > mean(v2)
        w = -w;
        v1 = -v1;
        v2 = -v2;
    end

    sort1 = sort(v1);
    sort2 = sort(v2);
    t1 = length(sort1);
    t2 = 1;

    while sort1(t1) > sort2(t2)
        t1 = t1-1;
        t2 = t2+1;
    end
    threshold = (sort1(t1) + sort2(t2))/2;
end
```

Appendix C mnist-parse.m Code

```
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
```