AMATH 482 Homework 4

David Warne

March 10, 2021

Abstract

Continuing with the study of Singular Value Decomposition, this assignment delves more into the power that SVD has when it comes to analyzing images, and how it can be used to train software to recognize certain features of those images. We are also introduced to the process of Linear Discretion Analysis, which allows us to isolate certain threshholds to better identify objects of value.

1 Introduction and Overview

In this assignment, I am provided with data from the MNIST Database, and am tasked with implementing SVD and LDA to train my computer to recognize various handwritten digits and classify them within the test data that I am provided with. The SVD provides insight into the principal components of the training data, which in turn allows me to determine how many modes would be the proper amount to get accurate results. After performing LDA, I am able to determine which two digits are the most difficult for the computer to separate, and which two digits are the easiest to separate. I then implement a decision tree classifier, as well as a support vector machine, and compare the accuracy of all three methods (LDA, SVM and decision tree).

2 Theoretical Background

Similar to the last assignment, it is important to understand the Singular Value Decomposition. From the textbook, [1], we know that the SVD "gives a type of least-square fitting algorithm, allowing us to project a matrix onto low dimensional representations in a formal, algorithmic way". Something new to this assignment, however, is the introduction of Linear Discretion Analysis. In brief, LDA is used to separate classes of objects, which makes it fantastic for classification problems like this one. When we implement SVD in software and get the values U, S, and V, we can use these values to then implement LDA. For the sake of this assignment, we will be projecting the training and test data from the MNIST data set into PCA space so that we can then utilize the built in MATLAB function classify to help train our computer to recognize digits 0-9.

For context, the MNIST dataset provides us with the following:

- 28x28x60000 matrix containing the training images
- 28x28x10000 matrix containing the test images
- 60000x1 matrix containing the labels for the training data
- 10000x1 matrix containing the labels for the test data

Below is the equation that I use in order to project the training data into PCA space:

$$Xtrain = \Sigma * V' \tag{1}$$

Where Σ is the matrix containing the singular values of the training data on the diagonal and V contains the right-singular value vectors associated with those singular values in S.

Below is the equation that I use to project the test data into PCA space:

$$Xtest = U' * t \tag{2}$$

Where U is the matrix containing the principal components of the training data (note that we can call these principal components since we make sure that the row-mean of our training data is 0 before performing SVD), and t is the vectorized test data.

3 Algorithm Implementation and Development

There are several algorithms that will allow us to accurately perform the processes discussed above. The first of which is doing an SVD analysis of the data. See Algorithm 1 for this. The second is projecting into PCA space and performing LDA. See Algorithm 2 for this. The third is implementing SVM and the decision tree and comparing our results to the LDA process. See Algorithm 3 for this.

Algorithm 1: Performing an SVD Analysis of the Data

Load the MNIST data, should be in the shape discussed earlier.
Vectorize both the training data and the test data so that they have the shape 784x60000 and 784x10000, respectively (use reshape() for this).
Subtract the row-wise mean from the training data. Using repmat will make this process much easier.
Normalize the mean zero training data and perform SVD on it.
Visualize the first few principal components by plotting the first few columns of U.
Visualize the singular value spectrum to identify which projections are the most prominent in the data set. From these singular values you can calculate how many modes will be necessary to get accurate image reconstruction.
Visualize a projection onto 3 V-modes of your choice (I have chosen 2,3 and 5). This will give a better idea of how the digits overlap and can give an early indication of which values will be difficult to separate.
Report findings.

Algorithm 2: Implementing Linear Discretion Analysis

Project the training data and test data into PCA space using equations 1 and 2
Decide on a rank r that will provide accurate image reconstruction (I have chosen 50). Take the first r modes of your projected datasets and store them in new variables. This is what we will use when constructing the LDA.
Ideally, you will use a for loop to iterate through all possible digit combinations, but for the sake of this algorithm I will demonstrate how to do it for a single pair
In a new variable, sort your projected training and test data so that they only contain data for a specific digit. Do this again for a separate digit. Concatenate these values into a single matrix.
Do the same for the training and test labels, such that for comparing the values like 0 and 1 the respective matrices look like [0 0 0 0 ..... 1 1 1 1 ....].
Using the classify function, input your test, training and training labels matrices.
Compare the output of classify to the test labels data and record the accuracy.
Repeat for all possible digit combinations (there are 45 of them)

4 Computational Results

- From the computed singular value spectrum, we can see that there are a few singular values that have a prominent effect on the projection of the data. I was able to implement an energy equation that calculated how many modes it would take to reach perfect image reconstruction, which ended up being

Algorithm 3: Implementing SVM and decision tree

Input your projected training data and the training labels into the built-in MATLAB function fitcecoc().
Plug the output of that function into the predict() function along with the test labels data.
Compare the output to test data and report accuracy.
The decision tree is basically the same, but you are using the fitctree function instead.

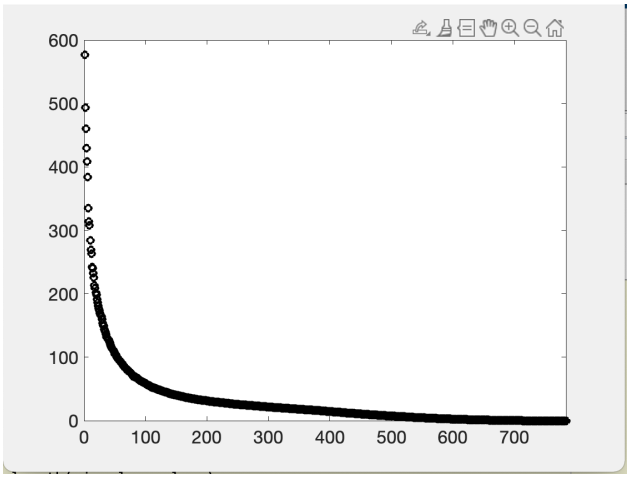

Figure 1: Singular Value Spectrum

712. This, however, is arbitrary in comparison to the human eye, and I found that about 50 modes was an accurate enough range to comprise an accurate representation of the data. See figure 1 for the full plot.

- The interpretation of U, Σ, and V is simple: U is a matrix containing the principal components of the training dataset, Σ contains the singular values on its diagonal, and V contains the right-singular values correlated to those singular values of Σ. I was able to plot the first 4 principal components of U and these can be seen in figure 2.
- Figure 3 illustrates the projection onto 3 V-modes, and as stated earlier is a decent indication of which numbers will be difficult to separate with LDA.
- In terms of the performance of the LDA that I built, I was able to discover that the most difficult digits to separate were 5 and 8, with an accuracy of only roughly 76.21 percent, and the two digits that were the easiest to separate were 0 and 1, with an accuracy of roughly 99.9 percent.
- In terms of the SVM method, I discovered that across the board, the SVM method had about 86.47 percent accuracy, which I would say is quite good considering that it had to classify all of the training data at once.
- In terms of the decision tree, I discovered that there was about 33 percent error with a max number of splits of 50. This seemed to be the worst method for classification out of the three, but with further exploration I believe a better result could be achieved.

5 Summary and Conclusions

In conclusion, I found the implementation of these various classification methods to be challenging but extremely rewarding. To know that I was able to take a massive dataset and use SVD to train my computer

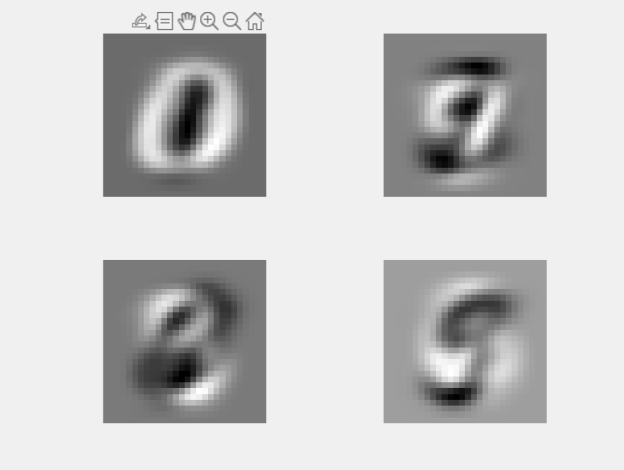

Figure 2: First 4 Principal Components

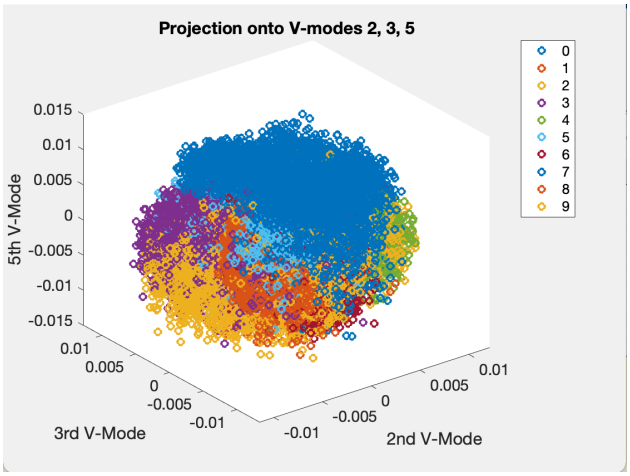

Figure 3: Projection onto 3 V-Modes (2,3, and 5)

to identify specific digits is something that I am really proud of. This, again, just goes to show how impressive these algorithms can be, and what they can do in terms of solving large-scale classification problems.

References

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

Appendix A MATLAB Functions

Here are the main functions that I used in this assignment:

- `[U,S,V] = svd()` allows us to compute singular value decomposition on our matrix of x and y components.
- `CLASS = classify(SAMPLE,TRAINING,GROUP)` classifies each row of the data in SAMPLE into one of the groups in TRAINING.
- `OBJ=fitcecoc(TBL,Y)` fits K*(K-1)/2 binary SVM models using the "one versus one" encoding scheme for data in the table TBL and response Y.
- `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.

Appendix B MATLAB Code

Here is the MATLAB code that was used to produce the information in this assignment.

```
% Assignment 4 Script
%%
clear all; close all; clc;

%% Loading the Data

[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');

%% Setting up the Data and Performing SVD
% reshaping the train data into a column vector so that we can perform SVD on it
num_digits = size(images_train, 3);
train_images = zeros(784, num_digits);
for k = 1:num_digits
   train = reshape(images_train(:,:, k), 784, 1);
   train_images(:, k) = train;
end

% do the same for the test data
num_test = size(images_test, 3);
test_images = zeros(784, num_test);
for k = 1:num_test
   test = reshape(images_test(:,:, k), 784, 1);
   test_images(:, k) = test;
end
```

```
% store size of training data
[m_train, n_train] = size(train_images);

% get the mean of the dataset
mtr = mean(train_images, 2);

% adjust the rows of each matrices by subtracting the mean
train_mean = train_images - repmat(mtr, 1, n_train);

% perform SVD on each set of data
[U,S,V] = svd(train_mean/sqrt(n_train-1),'econ');


%% Looking at the Principal Components
% taking a look at the first couple principal components
for k = 1:4
   subplot(2,2,k)
   ut1 = reshape(U(:,k),28,28);
   ut2 = rescale(ut1);
   imshow(ut2)
end

%% Looking at the singular values
figure(2)
plot(diag(S),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 784])

% in order to figure out how many modes are necessary for good image
% reconstruction we can take a look at the energy that is produced by a
% certain amount of singular values.

singular_values = diag(S);
num_modes = length(singular_values);
energies = zeros(num_modes,1);
for i = 2:num_modes
    energy = sum(singular_values(1:i))/sum(singular_values);
    energies(i) = energy;
end

[M, I] = max(energies);

% From the resulting values of M and I, we can see that we would want 712
% modes for 100% energy of the system; however, we can probably get a
% decent amount of image reconstruction that is visible to the human eye
% with something around 75%, or

%% Projection onto 3 V-modes
for label=0:9
    label_indices = find(labels_train == 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)
```

Code for set up and SVD analysis

```
%% Projecting into PCA space so that we can perform LDA

% set up values to put into classify
r = 50;

train_projection = (S*V')';
train_r = train_projection(:,1:r);

test_projection = (U'*test_images)';
test_r = test_projection(:, 1:r);

%% Testing all digit pairs

best_accuracy = 0;
worst_accuracy = 1;
idx_best = [0 0];
idx_worst = [0 0];
accuracies = zeros(3, 73);
counter = 1;
for i = 1:9
    for j = 1:9
        num_correct = 0;
        if ((i-1) ~= j)
            % get test data from corresponding digit pair
            test_x_labels = find(labels_test == i-1);
            test_x = test_r(test_x_labels, :);

            test_y_labels = find(labels_test == j);
            test_y = test_r(test_y_labels, :);

            test_xy = [test_x; test_y];

            % same for the train_projection
            train_x_labels = find(labels_train == i-1);
            train_x = train_r(train_x_labels, :);

            train_y_labels = find(labels_train == j);
            train_y = train_r(train_y_labels, :);

            train_xy = [train_x; train_y];

            % get associated labels and concatenate
            test_sorted = sort(labels_test);
            train_sorted = sort(labels_train);

            train_labels_x = train_sorted(train_sorted == i-1)';
            train_labels_y = train_sorted(train_sorted == j)';
            test_labels_x = test_sorted(test_sorted == i-1)';
```

```
            test_labels_y = test_sorted(test_sorted == j)';

            train_labels_xy = [train_labels_x train_labels_y]';
            test_labels_xy = [test_labels_x test_labels_y]';

            pre_xy = classify(test_xy, train_xy, train_labels_xy);

            length_data = length(test_labels_xy);
            for k=1:length_data
                if pre_xy(k) == test_labels_xy(k)
                    num_correct = num_correct + 1;
                end
            end
            accuracy_xy = num_correct/length_data;

            if (accuracy_xy > best_accuracy)
                best_accuracy = accuracy_xy;
                idx_best(1) = i-1;
                idx_best(2) = j;
            end

            if (accuracy_xy < worst_accuracy)
                worst_accuracy = accuracy_xy;
                idx_worst(1) = i-1;
                idx_worst(2) = j;
            end
            accuracies(1, counter) = accuracy_xy;
            accuracies(2, counter) = i-1;
            accuracies(3, counter) = j;
            counter = counter + 1;
        end
    end
end

% best is 0 and 1

% worst is 5 and 8
```

Code for linear discretion analysis

```
%% Checking the accuracy of SVM Method
% using digit space of r = 50
% MATLAB error tells us to use fitcecoc instead of fitcsvm for models with
% more than 2 classes

Mdl_svm = fitcecoc(train_r, labels_train);
pre_svm = predict(Mdl_svm, test_r);

num_correct = 0;
for i = 1:length(labels_test)
    if (pre_svm(i) == labels_test(i))
        num_correct = num_correct + 1;
    end
end
accuracy = num_correct/length(labels_test);
```

```
% accuracy is about 86% across the board -> not too bad!

%% Decision Tree
 % classification tree on fisheriris data
load fisheriris;
tree=fitctree(train_r,labels_train,'MaxNumSplits',50,'CrossVal','on');
view(tree.Trained{1},'Mode','graph');
classError = kfoldLoss(tree)
```

Code For SVM and decision tree