AMATH 482 Homework 4

Shane Fretwell

March 6, 2021

Abstract

As the world moves toward data-driven technologies, it is more and more important to understand some of the differences between methods of machine learning. Here, we explore computer vision through the lens of Singular Value Decomposition and Principal Component Analysis. We will compare several classification methods and their abilities to classify a specific dataset. Along the way, we will gain a better understanding of the capabilities of computer vision.

1 Introduction and Overview

Recognition of handwriting can be difficult, as anyone who has been to a doctor's office could tell you. We will be attempting to relegate this task to computer vision. Specifically, we will be using Principal Component Analysis to obtain feature data about a famous dataset of handwritten digits, which we will then classify with three methods: Linear Discriminant Analysis, Decision Trees, and Support Vector Machines.

2 Theoretical Background

2.1 Principal Component Analysis

To review, PCA is the use of SVD to change the base of a data matrix such that principal components can be inspected individually.

2.1.1 Singular Value Decomposition

For clarity, it is now useful to define some terms. If we have measurements of the height and weight of several people, we can put this data into a matrix with one row for each person. Then, the rows of this matrix are the *observations* of the height and weight data. Naturally, a person's height and weight are correlated, because as someone gets taller, they tend to weigh more as well. One could say that one's height and weight are both, in part, determined by a third factor – imagine that this third factor is the gene or collection of genes that determine someone's height. We will call third factors like this, that explain correlation in the data, *concepts*. With this vocabulary defined, we can discuss the meaning of the decomposed parts of a data matrix, seen in equation 1.

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

2.1.1.1 The matrix V

For our purposes, it will be useful to think of the matrix V as relations between observations of data and underlying concepts of the data (2). That is, each observation will be related to the concepts in varying amounts. In particular, the first column of V will be the amounts of each concept that the first data observation has. If the matrix A is the height-weight data matrix described before, then the first entry of the first column of V represents the height genes of the first person in the dataset. The second entry would be this person's relation to another concept; a concept which explains why a scatter plot of the height-weight data is not a simple line, determined solely by each person's height genetics. For simplicity of explanation, we will say this other concept is nutrition.

2.1.1.2 The Matrix Σ

The matrix Σ is a diagonal matrix containing the strengths of the influence of each concept on the data. The first entry on the diagonal would be the amount of spread of the data that can be explained by height genes (2). This entry consequently reveals how well we can predict a person's weight and height from their height genes alone.

2.1.1.3 The Matrix U

The columns of U are the principal components of the matrix A, but they can also be thought of as the strength of each concept's influence over the different measurement columns in A (2). One would expect that the first column of U has a larger value for the height measurements than for weights, because height genes would influence height more strongly than they would influence weight. Note that this matrix is capturing information about the relationship between weight-height measurements and height genetics, irrespective of the people themselves. This information is imperfect, because the matrix A does not contain measurements from every person in the population, but nonetheless, this information is an estimate of the whole population's relationship between weight-height measurements and height genetics.

2.1.2 Change of Basis for Use in Predictive Models

Imagine we have a second weight-height matrix X of measurements of different people from the population. If we decomposed X into its U_X, Σ_X, and V_X matrices, then we would expect that U would be very similar to U_X, since they are both approximating the same relationships between weight-height data and genetics and nutrition; this relationship will not change drastically between different samples of people.

Since the matrix U approximates the matrix U_X, we can estimate the height genetics and nutrition of the people in matrix X by left-multiplying by U*. This performs a change of basis, such that the result is a matrix of estimates of the height genes and nutrition of each person whose measurements are found in X, given what we learned about the population from the measurements in A.

Once we have these estimates of people's height genes and nutrition, we can use them as features in predictive models just as if we had measured these things directly. That is the power of PCA, as applied to machine learning. In total, one can imagine being given height and weight data about a person and being able to discern their likelihood of heart disease, because we can estimate the nutrition of their diet and how that will in turn affect their risk of cardiac events.

2.1.3 Dimensionality Reduction

From lecture 11, we have the theorem of equation 2 where r is the rank of A. This theorem also states that the partial sum up to i is the best i-rank approximation of A. In the vocabulary we have established, this means that the first i concepts are the best set of i concepts with which we can describe the data in A.

$$A = \sum_{k=1}^{r} \sigma_k u_k v_k * \tag{2}$$

2.2 Linear Discriminant Analysis

The first of three methods for classification modelling, LDA is the idea of reducing the dimensionality of the data to one, while ensuring that categorical groups are projected onto distinct portions along this single dimension. Good projection of two groups of data onto a line is determined by two factors; group-wise spread and inter-mean distance. We want the spread of each projected group to be as small as possible, while inter-mean distance should be maximized. This allows for categorization of data points into one group or the other based on their position on the projection line, as can be seen in figure 1 in appendix B, taken from lecture 19 notes. Specifically, a point along the line is chosen so that if an observation's projection onto w is below that point, it is classified as one group, and otherwise as the other group.

This generalizes to any number of groups by using more points to separate groups, so long as the projections of each group are sufficiently separated, as in part b of figure 1. In lecture 19, we learned that the optimal vector w onto which we should project data for linear discrimination is given by the maximizer of equation 3. Where μ is the overall mean measurements vector and μ_j is the mean measurements vector of each group, S_B is a measure of the inter-mean distance and S_w is a measure of the spread within each group. Maximizing the ratio in equation 3 is thus the same as finding the basis vector w that maximizes inter-mean distance and minimizes the spread in each group.

$$w = argmax \frac{w^T S_B w}{w^T S_w w}, \qquad S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T, \qquad S_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{3}$$

Finding this maximizer, as also discussed in lecture 19, is done by solving the eigenvalue problem in equation 4.

$$S_B w = \lambda S_w w \tag{4}$$

2.3 Decision Tree Classification

Given a set of features for an observation, a decision tree will navigate down a tree of decisions, traversing to the node indicated by the truth value of the decision at the current node until a leaf node is reached. An example of a decision is comparing measurement 3 of the observation against a constant like 5. If measurement 3 of this observation is less than 5, then the next decision is the left child of the current node, otherwise it is the right. All leaf nodes are labelled as belonging to one of the groups of the data, so when a leaf is reached, the label of that leaf is the predicted label of the observation. Following this process, each leaf node is reached only by data observations that fit every decision required to reach that leaf. These trees can contain as many decisions as are required to obtain perfect accuracy on a training set, but this would clearly overfit the data. For this reason, a hyperparameter is used in practice to limit the number of branches (decisions) in the tree.

2.4 Support Vector Machine

Very similar to LDA, a SVM finds separations in the data that divide groups. There are two main differences, however. LDA reduces the data to one dimension, so the separations between groups are single points on a line. A SVM does not reduce the dimension of the data, and so the separations are hyperplanes that divide the data space.

Where LDA considers every data point in choosing where to divide the line for classification, a SVM will instead take only the most difficult points to differentiate between. This means that the hyperplanes that separate the groups are chosen to maximize the minimum distance between points in different groups and the hyperplane. In other words, the hyperplane is chosen so that, while still being in between the groups, the perpendicular distance to all nearest points is maximized.

3 Algorithm Implementation and Development

The images of handwritten digits are all 28x28 pixel grayscale images, with 60000 in the training set and 10000 in the test set. To prep these for SVD and PCA, we reshape these images into column vectors of pixels and concatenate them into the 784x60000 matrix A and the 784x10000 matrix X and subtract the mean values of each pixel. Subtracting the mean focuses the analysis on the differences between the numbers, since we are looking to identify specific numbers and not whether each image is a number. We know they are all numbers, after all.

We also have labels of the digit contained in each image, for each of the training set and the test set.

3.1 Concepts in the Context of Digit Data

When it comes to handwritten digits, the concepts as defined in section 2.1.1 will be referred to as strokes. These strokes are groups of pixels that correlate strongly, and whose linear combination can form any of the numbers in the training set. This gives rise to an interpretation of the U, Σ, and V matrices where U is the relation of each pixel to the strokes. Each column of V is a linear combination of strokes to form the image corresponding to that column. The i^{th} value in the diagonal of Σ is the importance of the i^{th} stroke in forming the images in A.

3.2 Exploring the Data with the SVD Decomposition of A

Once we have decomposed A, there are several exploratory procedures available to us to better understand the digit data. Unless otherwise mentioned, the visualizations and data that arise from these procedures will be shown in section 4.

3.2.1 Low Rank Approximation and Viewing Strokes

First, we should get an idea of the number of strokes it takes to form each number. There are 784 strokes that result from the SVD decomposition of A, and almost none of these are necessary for differentiating between digits. We can get an idea as to how many strokes are necessary by reconstructing images of the digits with varying dimensionality. This is done by taking the partial sum of equation 2 up to k terms, yielding the k dimensional approximation of A, and reshaping this back into a 28x28x60000 matrix of images.

To view the strokes that make up the digits, one needs only to scale the columns of U by their singular values and reshape into images. The first 15 of strokes are shown in appendix B, figure 4.

3.2.2 Project Digits onto Select Strokes

To give context to the classification we will be implementing later, we plot digits in a 3-d space comprised of the 2nd, 3rd, and 4th strokes. That is, for each image, we plot the amount of the 2nd stroke in the image vs the amount of the 3rd stroke vs that of the 4th stroke. Coloring by digit label, we can see the grouping of each digit within this projection space. With this, the hope is to better understand how classification models can differentiate between digits using these stroke projections. Two such plots can be found in appendix B, figure 5.

3.3 Classifying the Data

Now, we implement the classification models that we will compare in section 4. The features these models will use to predict will be the projections of the data onto the principal components of A. These are obtained by left-multiplying both X and A by U' to perform a change of basis, as discussed in section 2.1.2.

3.3.1 LDA Classifiers

3.3.1.1 Differentiating Between Two and Three Digits

From section 3.2.1 we will have an idea as to how many strokes are needed to differentiate between digits, and we will use projections onto exactly that many strokes as our features for training two models. The first will differentiate between two digits, and the second between three. The only difference between the implementation of the first model and the second is in the first two lines of Algorithm 1. This algorithm describes the implementation of the two-digit classifier and the evaluation of its accuracy. The model is being implemented by the built-in matlab function *classify*. Accuracy is determined by the number of correct classifications divided by the number of total observations.

Algorithm 1: LDA 2-digit Classifier

```
SdigitMask = find((Slabels == 1) | (Slabels == 4)); % get indices of images containing 1s and 4s
TdigitMask = find((Tlabels == 1) | (Tlabels == 4)); % the 3-digit classifier has one more Boolean term for 3s
% Test data in the Principal Component Basis of Training Data
PCBX = U'*X; PCBA = U'*A;
% First k V-modes
testfeatures = transpose(PCBX(1:k,TdigitMask)); trainfeatures = transpose(PCBA(1:k,SdigitMask));
class = classify(trainfeatures, trainfeatures, Slabels(SdigitMask), 'linear');
trainaccuracy3digit = length(find(Slabels(SdigitMask) == class))/length(class)
class = classify(testfeatures, trainfeatures, Slabels(SdigitMask), 'linear');
testaccuracy3digit = length(find(Tlabels(TdigitMask) == class))/length(class)
```

3.3.1.2 Determining the Most and Least Distinct Digits

To determine the most and least difficult digits to distinguish between, we can simply apply algorithm 1 to every pair of digits, replacing 1 and 4 in the first two lines each time, saving each test accuracy in a 2d array of test accuracies.

3.3.2 Decision Tree Classifier

We use the same features for fitting and testing this classifier, except scaled by the max training projection to lie between -1 and 1 for computational speed in fitting the data.

Algorithm 2: Decision Tree Training

```
maxtrainfeatures = max(trainfeatures, [], 'all');
scaledtrainfeatures = trainfeatures / maxtrainfeatures;  scaledtestfeatures = testfeatures / maxtrainfeatures;
tree=fitctree(scaledtrainfeatures,Slabels,'MaxNumSplits',1000);
trainClass = predict(tree, scaledtrainfeatures); testClass = predict(tree, scaledtestfeatures);
```

3.3.3 Support Vector Machine Classifier

We again use the scaled features for the same reason as with the decision tree, and train several SVMs at the same time using the function *fitcecoc*.

Algorithm 3: SVM Training

```
Mdl =fitcecoc (scaledtrainfeatures,Slabels);
trainClass = predict(Mdl, scaledtrainfeatures);  testClass = predict(Mdl, scaledtestfeatures);
```

4 Computational Results

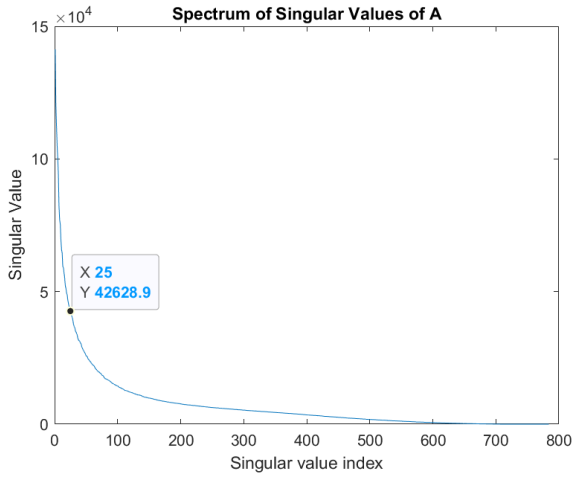

Figure 2: Spectral decomposition of A

4.1 Singular Value Spectrum and Reconstruction

In figure 2, we have the distribution of the singular values of the matrix A. These are the strengths of the concepts, or strokes, of A, and they trail off quite sharply. This means that with as little as 25 of the strokes, we can discern by eye what digit is in each reconstruction – though with some difficulty when it comes to the fourth column. This is seen in figure 3.

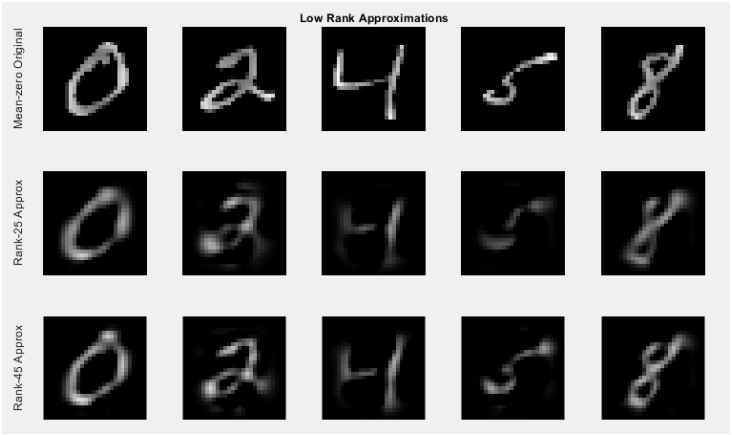

Figure 3: Low-Rank Approximations of five different digits.

4.2 LDA Performance

The two-digit model for differentiating between a 1 and a 4 had a training set accuracy of 0.9936 and a test set accuracy of 0.9981. This is an impressive level of accuracy in the test set, which is to be expected when one sees the level of separation between ones and fours in the projection space shown in figure 5, appendix B. Adding another digit, 3, slightly reduces the training and test set accuracies to 0.9748 and 0.9818, respectively.

Generalizing to any two pairs of digits, we find that the most difficult digits to separate are 5 and 8, which still retains a testing accuracy of 0.9325. It makes sense, due to their similar forms, that 5 and 8 would be the two digits most difficult to differentiate. Figure 5 shows how mixed fives and eights are in the projection space of only the 2nd, 3rd, and 4th V-Modes. The easiest to separate are 1 and 4, whose accuracies have already been discussed.

4.3 Decision Tree Performance

In differentiating between all digits at once, the Decision Tree model predicts on training data with an accuracy of 0.8906, and on test data with an accuracy of 0.8365.

4.4 SVM Performance

In differentiating between all digits at once, the SVM model can accurately predict 0.9143 of training data and 0.9184 of test data. This is significantly better than the accuracies of the decision tree model, and with more consistency across the training and test sets, which likely indicates that this model is overfit to neither the training nor the test set.

4.5 Performance Comparisons

The table below summarizes the test set accuracies of each model when differentiating between the easiest and most difficult pairs of digits. Overall, it seems That the SVM model is slightly more accurate than the other models, but all models perform quite well.

	LDA	Decision Tree	SVM
1,4 accuracy	0.9981	0.9943	0.9991
5,8 accuracy	0.9325	0.9389	0.9480

5 Summary and Conclusions

We went in depth into the meaning behind the principal components of our data and utilized this understanding to train several accurate predictive models. In essence, these methods of modelling all partition space to separate out the different groups within our data, and this is shown by the similar accuracy levels between the methods. There is one noticeable difference, however, between the decision tree model and the other models, because the decision tree performed poorly on the task of classifying all digits at once, as compared to the other models. This method of modelling also seems to be more sensitive to over fitting, as is indicated by the poor testing accuracy as compared to the training accuracy. All other models maintained similar testing and training accuracies using 25 V-mode projections, but for the decision tree this seems to have been an over fit.

With this analysis, we are in good company in utilizing this iconic dataset to better understand the concepts of computer vision.

References

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

[2] Rajaraman, A., & Ullman, J. (Writers). (2016, April 13). *Lecture 47 – singular value decomposition | tanford university* [Video file]. Retrieved March 7, 2021, from https://www.youtube.com/watch?v=P5mlg91as1c&ab_channel=ArtificialIntelligence-AllinOne

Appendix A MATLAB Functions

- [images, labels] = mnist_parse(path_to_digits, path_to_labels) parses the digit data files at the locations specified by the given paths, and returns the data.
- class = classify(SAMPLE,TRAINING,GROUP) classifies each row of the data in SAMPLE into one of the groups in TRAINING. GROUP is a grouping variable for TRAINING.
- Tree = fitctree(X,Y) fits a decision tree model based on the data in X and labels in Y.
- Class = predict(model, features) predicts the labels of the given feature data according to the given model's fit, and returns the classifications.
- Model = fitcecoc(X,Y) fits a multi-group SVM model based on the data in X and labels in Y.
- Model = fitcsvm(X,Y) fits a two-group SVM model based on the data in X and labels in Y.
- [U, S, V] = svd(A) returns the matrices U, Σ, and V from the SVD decomposition of A.

Appendix B Additional Figures

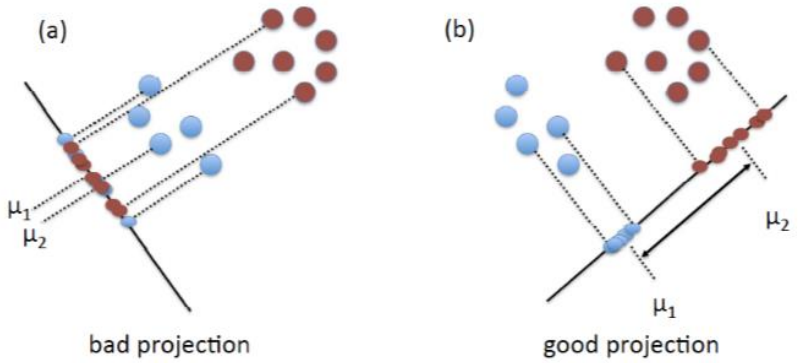

Figure 1: Good (b) and bad (a) projection of data onto a line for the purpose of linear discrimination of groups

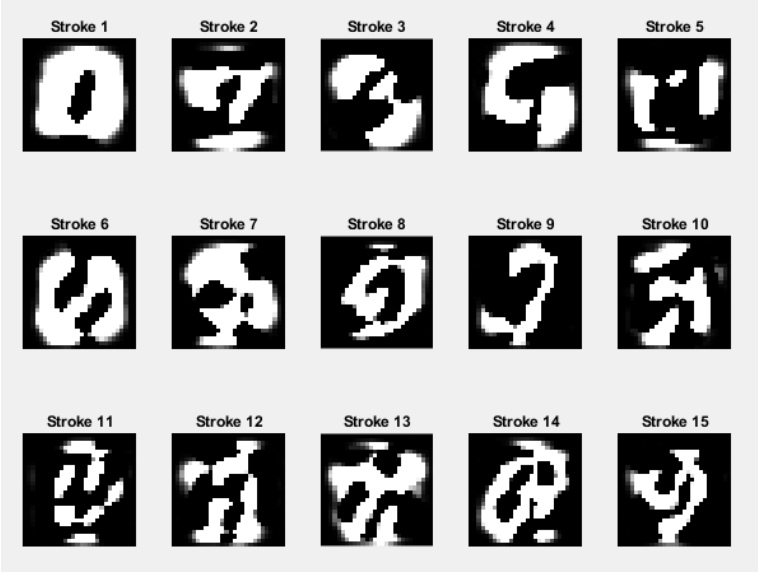

Figure 4: The first 15 strokes, or principle components, of A

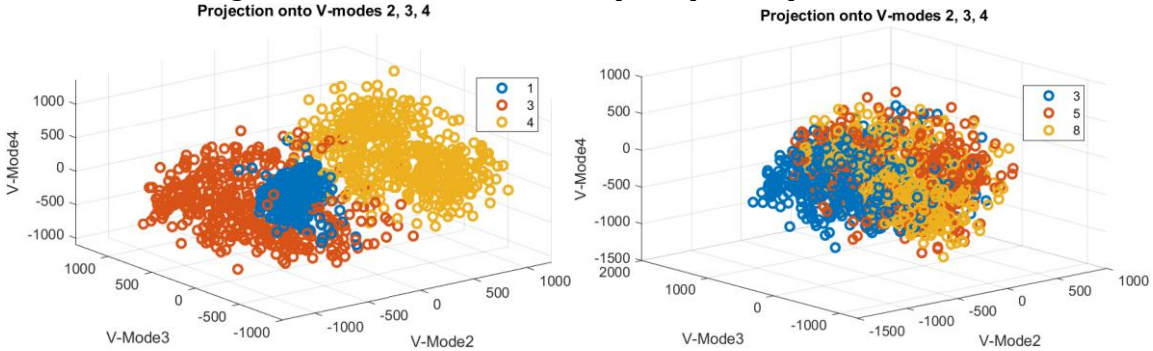

Figure 5: Plots of the projections of digits onto the same V-modes, demonstrating how some digits appear to be very similar in the projection space and others quite distinct.

Appendix C MATLAB Code

```
%%
%{
03-06-2021
Shane Fretwell
AMATH 482 Assignment 4, Hand-Written Digit Recognition
%}
%%
clear; close all;
%%
[Simages, Slabels] = mnist_parse(...
    'C:/Users/sear/Documents/AMATH482_data/Ass4/train-images-idx3-ubyte', ...
    'C:/Users/sear/Documents/AMATH482_data/Ass4/train-labels-idx1-ubyte');
[Timages, Tlabels] = mnist_parse(...
    'C:/Users/sear/Documents/AMATH482_data/Ass4/t10k-images-idx3-ubyte', ...
    'C:/Users/sear/Documents/AMATH482_data/Ass4/t10k-labels-idx1-ubyte');

%% SVD
A = reshape(Simages, [28^2, 60000]);
X = reshape(Timages, [28^2, 10000]);
m = mean(A,2);
A = double(A) - m;
X = double(X) - mean(X,2);
[U,S,V] = svd(A, 'econ');
s = diag(S);

%% Singular Value Spectrum
figure(1)
plot(s);
title('Spectrum of Singular Values of A')
xlabel('Singular value index'); ylabel('Singular Value')

%%
imgnum = 3;
img = reshape(uint8(A(:,imgnum)), [28, 28]);
ranks = [784, 25, 65, 105];
figure(2)

subplot(2,2,1)
imshow(img)
title("Original Image Minus Pixel Mean")

for digit=2:4
    subplot(2,2,digit)
    % LRA = Low Rank Approximation
    LRA = U(:, 1:ranks(digit))...
        * diag(s(1:ranks(digit)))...
        * transpose(V(:, 1:ranks(digit)));
    LRA = uint8(reshape(LRA,[28,28,60000]));
    imshow(LRA(:,:,imgnum))
    title(strcat("Rank-", string(ranks(digit)), " Approximation"))
end

%% Low Rank Approximation of Several Digits
imgnums = [2, 6, 3, 12, 18];
figure(3)
for digit=1:length(imgnums)
    subplot(3, length(imgnums), digit)
```

```
    imshow(reshape(uint8(A(:,imgnums(digit))), [28, 28]))
    title("Original Image Minus Pixel Mean")

    subplot(3, length(imgnums), digit + length(imgnums))
    LRA = U(:, 1:25) * diag(s(1:25)) * V(:, 1:25)';
    LRA = uint8(reshape(LRA,[28,28,60000]));
    imshow(LRA(:,:,imgnums(digit)))
    title("Rank-25 Approximation")

    subplot(3, length(imgnums), digit + 2*length(imgnums))
    LRA = U(:, 1:45) * diag(s(1:45)) * V(:, 1:45)';
    LRA = uint8(reshape(LRA,[28,28,60000]));
    imshow(LRA(:,:,imgnums(digit)))
    title("Rank-45 Approximation")
end

%% View strokes
L = 3;
W = 5;
start = 1;
strokes = start:(start + L*W - 1);
figure(4)
for j=1:L
    for digit=1:W
        subplot(L, W, digit + W*(j-1))

        strokeMask = zeros(1,size(V,2));
        strokeMask(strokes(digit + W*(j-1))) = 1;
        strokeimg = U(:, strokes(digit + W*(j-1)))...
            * s(strokes(digit + W*(j-1)));
        strokeimg = uint8(reshape(strokeimg,[28,28]));
        imshow(strokeimg)
        title(strcat("Stroke ", string(strokes(digit + W*(j-1)))))
    end
end

%% Project Digits onto V-Modes
figure(5)
strokes = [2, 3, 4];
n = 5000;
hold off
for digit=[1,3,4]
    SdigitMask = find(Slabels(1:n) == digit);
    scatter3(s(strokes(1))*V(SdigitMask,strokes(1)),...
        s(strokes(2))*V(SdigitMask,strokes(2)),...
        s(strokes(3))*V(SdigitMask,strokes(3)), ...
         'DisplayName', sprintf('%i',digit), 'Linewidth', 2)
    xlabel(strcat('V-Mode ', string(strokes(1))))
    ylabel(strcat('V-Mode ', string(strokes(2))))
    zlabel(strcat('V-Mode ', string(strokes(3))))
    title('Projection onto V-modes 2, 3, 4')
    hold on
end
legend

%% Two Digit Classifier
SdigitMask = find((Slabels == 1) | (Slabels == 4));
```

```
TdigitMask = find((Tlabels == 1) | (Tlabels == 4));

% Test data in the Principal Component Basis of Training Data
PCBX = U'*X; % U'*A = S*V'
PCBA = U'*A;
n = 25;
strokes = 1:n;
% V-modes specified by strokes
testfeatures = transpose(PCBX(strokes,TdigitMask));
trainfeatures = transpose(PCBA(strokes,SdigitMask));

class = classify(trainfeatures, trainfeatures, Slabels(SdigitMask), 'linear');
trainaccuracy2digit = length(find(Slabels(SdigitMask) == class))/length(class)
class = classify(testfeatures, trainfeatures, Slabels(SdigitMask), 'linear');
testaccuracy2digit = length(find(Tlabels(TdigitMask) == class))/length(class)
clear trainaccuracy2digit testaccuracy2digit;

%% Three Digit Classifier
SdigitMask = find((Slabels == 1) | Slabels == 3 | (Slabels == 4));
TdigitMask = find((Tlabels == 1) | Tlabels == 3 | (Tlabels == 4));

% Test data in the Principal Component Basis of Training Data
PCBX = U'*X; % U'*A = S*V'
PCBA = U'*A;
n = 25;
strokes = 1:n;
% V-modes specified by strokes
testfeatures = transpose(PCBX(strokes,TdigitMask));
trainfeatures = transpose(PCBA(strokes,SdigitMask));

class = classify(trainfeatures, trainfeatures, Slabels(SdigitMask), 'linear');
trainaccuracy3digit = length(find(Slabels(SdigitMask) == class))/length(class)
class = classify(testfeatures, trainfeatures, Slabels(SdigitMask), 'linear');
testaccuracy3digit = length(find(Tlabels(TdigitMask) == class))/length(class)
clear trainaccuracy3digit testaccuracy3digit;

%% Find most and least difficult digits to classify
trainaccuracies = zeros(10, 10);
testaccuracies = zeros(10, 10);
% Test data in the Principal Component Basis of Training Data
PCBX = U'*X; % U'*A = S*V'
PCBA = U'*A;
n = 25;
strokes = 1:n;

for i=0:8
    for j=i+1:9
        SdigitMask = find((Slabels == i) | (Slabels == j));
        TdigitMask = find((Tlabels == i) | (Tlabels == j));

        % V-modes specified by strokes
        testfeatures = transpose(PCBX(strokes,TdigitMask));
        trainfeatures = transpose(PCBA(strokes,SdigitMask));

        class = classify(trainfeatures, trainfeatures, Slabels(SdigitMask),
'linear');
```

```
        trainaccuracies(i+1,j+1) = length(find(Slabels(SdigitMask) ==
class))/length(class);
        class = classify(testfeatures, trainfeatures, Slabels(SdigitMask),
'linear');
        testaccuracies(i+1,j+1) = length(find(Tlabels(TdigitMask) ==
class))/length(class);
    end
end
% Subtract one from i,j to get the actual digits, due to matlab indexing
[i,j]=find(testaccuracies == min(testaccuracies(testaccuracies ~= 0),[],'all'));
min_test_accuracy = [i-1,j-1]
[i,j]=find(testaccuracies == max(testaccuracies,[],'all'));
max_test_accuracy = [i-1,j-1]
clear min_test_accuracy max_test_accuracy

%% Decision Tree Analysis
n = 25;
strokes = 1:n;

PCBX = U'*X; % U'*A = S*V'
LPCBX = PCBX(strokes,:); % Projections onto V-modes specified by strokes
PCBA = U'*A;
LPCBA = PCBA(strokes,:);

maxLPCBA = max(LPCBA, [], 'all');
features = transpose(LPCBA) / maxLPCBA;
tree=fitctree(features,Slabels,'MaxNumSplits',1000);

trainClass = predict(tree, features);
testClass = predict(tree, transpose(LPCBX) / maxLPCBA);

trainAccuracy = length(find(Slabels == trainClass))/length(trainClass)
testAccuracy = length(find(Tlabels == testClass))/length(testClass)

%% Get training and test accuracy for several different ranks and max splits
PCBX = U'*X; % U'*A = S*V'
PCBA = U'*A;
N = [4, 25, 50, 80, 100];
B = [100, 500, 1000, 1500];
treeAccuracies = zeros(length(N),length(B),2);
for i = 1:length(N)
    for j = 1:length(B)
        n=N(i);
        b=B(j);
        strokes = 1:n;

        LPCBX = PCBX(strokes,:);
        LPCBA = PCBA(strokes,:);

        maxLPCBA = max(LPCBA, [], 'all');
        features = transpose(LPCBA) / maxLPCBA;
        tree=fitctree(features,Slabels,'MaxNumSplits',b);

        trainClass = predict(tree, features);
        testClass = predict(tree, transpose(LPCBX) / maxLPCBA);
```

```
        treeAccuracies(i,j,1) = length(find(Slabels ==
trainClass))/length(trainClass);
        treeAccuracies(i,j,2) = length(find(Tlabels ==
testClass))/length(testClass);
    end
end

%% SVM classifier with all digits
PCBX = U'*X; % U'*A = S*V'
PCBA = U'*A;
n = 25;
strokes = 1:n;
% Projections onto V-modes specified by strokes
LPCBX = PCBX(strokes,:);
LPCBA = PCBA(strokes,:);

maxLPCBA = max(LPCBA, [], 'all');
features = transpose(LPCBA) / maxLPCBA;
Mdl = fitcecoc(features,Slabels);

trainClass = predict(Mdl, features);
testClass = predict(Mdl, transpose(LPCBX) / maxLPCBA);

trainAccuracy = length(find(Slabels == trainClass))/length(trainClass)
testAccuracy = length(find(Tlabels == testClass))/length(testClass)

%% Decision Tree Analysis with easy and difficult digits
SdigitMask = find((Slabels == 1) | (Slabels == 4));
TdigitMask = find((Tlabels == 1) | (Tlabels == 4));

n = 25;
strokes = 1:n;

PCBX = U'*X; % U'*A = S*V'
LPCBX = PCBX(strokes,TdigitMask); % Projections onto V-modes specified by strokes
PCBA = U'*A;
LPCBA = PCBA(strokes,SdigitMask);

maxLPCBA = max(LPCBA, [], 'all');
features = transpose(LPCBA) / maxLPCBA;
tree=fitctree(features,Slabels(SdigitMask),'MaxNumSplits',1000);

trainClass = predict(tree, features);
testClass = predict(tree, transpose(LPCBX) / maxLPCBA);

trainAccuracy = length(find(Slabels(SdigitMask) == trainClass))/length(trainClass)
testAccuracy = length(find(Tlabels(TdigitMask) == testClass))/length(testClass)

%% SVM classifier with easy and difficult digits
SdigitMask = find((Slabels == 5) | (Slabels == 8));
TdigitMask = find((Tlabels == 5) | (Tlabels == 8));

PCBX = U'*X; % U'*A = S*V'
PCBA = U'*A;
n = 25;
strokes = 1:n;
% Projections onto V-modes specified by strokes
```

```
LPCBX = PCBX(strokes,TdigitMask);
LPCBA = PCBA(strokes,SdigitMask);

maxLPCBA = max(LPCBA, [], 'all');
features = transpose(LPCBA) / maxLPCBA;
Mdl = fitcsvm(features,Slabels(SdigitMask));

trainClass = predict(Mdl, features);
testClass = predict(Mdl, transpose(LPCBX) / maxLPCBA);

trainAccuracy = length(find(Slabels(SdigitMask) == trainClass))/length(trainClass)
testAccuracy = length(find(Tlabels(TdigitMask) == testClass))/length(testClass)
```