Amath 482 Homework 4

USEFULNESS OF SINGULAR VALUE DECOMPOSITION IN CLASSIFICATION

MICHAEL GABALIS

Abstract: This paper aims to demonstrate the power and limitations of the SVD in extracting fundamental information from data. To do so, we will attempt to use singular value decomposition and principal component analysis to form a low-rank approximation of digits from the MNIST data set. Furthermore, we will use these ideas to develop a coordinate system ideal for classification via Linear Discriminant Analysis (LDA), Support Vector Machines (SVM), and Decision Tree Learning (DTL). As a case study, we will run these algorithms on different pairs of digits obtained from the MNIST data set, or the entire data set in whole in order to analyze the performance of each classifier.

University of Washington

March 2021

1 Introduction and Overview

The SVD is a powerful tool for understanding the fundamental structure of data. Large inputs of data usually needed in machine learning can be broken down into fundamental features to decrease the size of the data. However, some limitations of the SVD are that it does not handle translated data well because it relies solely on simple linear combinations of its principal modes to capture features in the data. To counteract this, we mean centered the data and normalized it to make the SVD more effective. Building on the SVD, we can perform Principal Component Analysis (PCA) in order to obtain a reduced projection of the centered data depending on how many principal components we want to keep. Once we have developed a coordinate system for our data, we can use different classifiers such as Linear Discriminant Analysis (LDA), Support Vector Machines (SVM), and Decision Tree Learning (DCL) to perform the classification of our training data. Finally, after building our classifier, we can test it on testing data to judge the accuracy of each classifying method and the limitations seen within our analysis.

2 Theoretical Background

2.1 Singular Value Decomposition

The Singular Value Decomposition is a diagonalization of a given matrix A that focuses on the rotations or reflections and stretching a vector undergoes when it is transformed by A. For any $A\backslash \mathrm{epsilonC}^{m\backslash \mathrm{timesn}}$ there exists a diagonalization of the form:

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

Where $U \in C^{m\times n}$ and $V \in C^{n\times n}$ are unitary matrices, $\Sigma \in R_{\geq 0}^{m\times n}$ is a diagonal matrix, and V^* is the complex conjugate transpose of V. As U and V are unitary matrices, their columns form an orthonormal basis for their respective space and the matrices have the properties of $UU^* = I_m$ and $VV^* = V^*V = I_n$. The diagonal entries of Σ are called the singular values of A. These singular values and their corresponding columns in U are the left singular vectors and the columns in V are the right singular vectors. These are often arranged within Σ such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. This breaks down the transformation of AX into three different fundamental transformations: a rotation or reflection of V transpose within the domain of X, a scaling of the components the space X by Σ , and another rotation or reflection of the space within the codomain of A.

2.2 Principal Component Analysis

The SVD is used within a data analysis technique called the principal component analysis (PCA). Given some data matrix X whose rows are statistically independent, PCA aims to determine the dimension of X or the least number of components that can sufficiently summarize the data. To find this we must consider the covariance matrix:

$$C_X = \frac{1}{n-1} XX^T \tag{2}$$

In this matrix, entry (i,j) corresponds to the covariance between row i and row j of X. If I does not equal to j then a high value in entry (i,j) suggests the two data measurements vary in similar ways and are likely to be redundant measurements of the same feature in the data. On the other hand, a low value at (i,j) suggests that the two data measurements are largely independent. If i is identical to j, the diagonal of C_x then the entry describes the variance in the data measurement i. Generally, data measurements with a large variance capture important features of the data, but those with low variance do not. To see the important of the SVD, consider the covariance matrix for $Y = U^*X$ where U is from the SVD of X:

$$C_Y = \frac{1}{n-1} YY^T = \frac{1}{n-1} \Sigma^2 \tag{3}$$

As $C_y$is the diagonal, all of its diagonal entries are zero; projecting the data X onto U^* produces a transformed data set with completely independent measurements, taking away the redundancy in the data. Thus, the columns of U represent the ideal coordinate system to use for our data and the columns of V are the coordinates of our data within this system. Also, the corresponding singular value σin the diagonal of Σ ranks the importance of each direction in this system based on the data's variance along that direction. For any given matrix X we can reconstruct it given these coordinates:

$$X = \sum_{i=1}^{min(m,n)} u_j \sigma_j v_j^* \tag{4}$$

where u_j , σ_j , v_j^* are the jth column of each SVD component. Since the singular values are a non-increasing sequence, we see that each successive term is increasingly less important to the structure of X. Thus, if the singular values decrease rapidly, we can get an accurate representation of X with only the first few terms of the sum. This is a low-rank approximation of the matrix X. The idea of PCA is that these first few terms form a good approximation to the data set because they represent the fundamental dynamics of the system. These principal components are the columns of U with the largest singular value and the direction U in which our data has the highest variance. This is likely to still capture the dynamics of our system while leaving out the noise or redundancy.

Another valuable tool to look at is the energy of our system. When finding the principal components, we can measure how much energy our low-rank approximation yields. This can measure whether or not we are accurately approximating the data of dimension n with a rank r matrix:

$$energy = \frac{\sigma_1^2 + \cdots + \sigma_n^2}{\sigma_1^2 + \cdots + \sigma_r^2} \tag{5}$$

2.3 LINEAR DISCRIMINANT ANALYSIS

Once the PCA is used to project the reduced dimensionality on our original data, we can now use LDA to classify the digits. The goal of LDA is to find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data. This can be formulated by:

$$w = \arg\max_{w} \frac{w^T S_B w}{w^T S_W w} \tag{6}$$

With the between-class scatter matrices S_B and within-class scatter matrix S_W written as

$$S_B = \sum_{j=1}^{n} m_j \left(\bar{x_j} - \bar{x}\right)\left(\bar{x_j} - \bar{x}\right)^T \tag{7}$$

$$S_W = \sum_{j=1}^{n} \sum_{i=1}^{N_j} \left(x_{i,j} - \bar{x_j}\right)\left(x_{i,j} - \bar{x_j}\right)^T \tag{8}$$

Where m_j is the number of samples in class j, and $\bar{x}$ is the mean of all the data and $\bar{x}_j$ is the mean of all the data in class j. Using these matrices, we can find the solution to the equation (6) with the generalized eigenvalue problem.

$$S_B w = \lambda S_W w \tag{9}$$

The information provided by LDA and solving the eigenvalue problem gives us the projection basis so we can separate different classes.

2.4 OTHER CLASSIFIERS

The other classifiers used are SVM and decision tree algorithms. Supervised learning follows a simple idea that given a simple set of labeled data measurements from a set of class, learn to classify future data measurements into those classes. SVM is essentially an optimization problem that tries to find the optimal hyperplane $\vec{w}x + b = 0$ where all points x satisfying $\vec{w}x + b < 0$ are labeled as class one and all points satisfying $\vec{w}x + b > 0$ are labeled class two. In this case the optimal hyperplane is defined as the one in which there are as few errors as possible in classification on the training data and the distance separating the classes is as large as possible. The Decision Tree follows a simple binary approach. For each variable in our system, scan over all possible values of that variable to be used as a separator of that data. Find the variable and its associated value that best divides the current collection of data into their respective groups. Divide the data according to this variable/value and repeat the process for both halves of the data, continuing for a set number of iterations or until some tolerance is

reached. In the end, the algorithm yields a series of conditions that can be used to classify the data by following the path down the decision tree.

3 Algorithm Implementation and Development

The code is broken up into two major parts: applying the SVD and analyzing the principal components, then using different classifying techniques on the test data.

3.1 SVD and PCA

1. First we have to load the digit data using our MNISTload function to convert the files given by LeCun et al. This function also converts the matrices into 784 by 60000, so that each column of data is a different image (Appendix B: MNISTload.m 0-57).
2. From here we have the option to extract certain digits. This is important later for classification, but only matters when we are trying to compare a certain pair or triplet of digits (Appendix B, Main_code : 7-19).
3. PCA process begins by normalizing the data: first taking the mean of each pixel across all images and subtracting from each pixel (Appendix B, Main_code : 20-26).
4. SVD is performed on normalized data and eigenvalues are acquired for variance calculations (Appendix B, Main_code : 29-31).
5. Singular values and variance is plotted to get a sense of how many singular values are needed for good representation. The energy and rank are then found to capture 90% of the systems energy (Appendix B, Main_code : 32-61).
6. The PCA projection is visualized in 3D by projecting **U** of a certain rank onto three modes colored by their digit label. (Appendix B, Main_code : 71-88).

3.2 LDA, SVM, and DTC

7. After we have these projections, we can use LDA to classify the digits. We can pull out images for two digits and use LDA_trainer.m to find the projection for the test images and the threshold value (Appendix B, Main_code : 90-92).
8. The within-class and between-class scatter matrices are found to be used in the eigenvalue problem. The eigenvalues and vectors are then found (Appendix B, LDA_Trainer : 14-24).
9. The projection of the training data is found, along with the threshold value for the classification (Appendix B, LDA_Trainer: 25-45).
10. The projection from the LDA trainer is used on the test images, with the threshold used to find the error rate in classifying the images (Appendix B, Main_code : 94-101).
11. Using the “fitcdiscr” command, LDA can be used to classify more than 2 digits at a time, with a confusion chart used to showcase error (Appendix B, Main_code : 102-108).
12. SVM is performed on the reduced images from PCA to create another confusion chart (Appendix B, Main_code : 109-111).
13. The Decision Tree algorithm is also used on the reduced images from PCA to find the error and make a confusion chart (Appendix B, Main_code : 113-117).
14. Repeat the entire process for different digits.

4 COMPUTATIONAL RESULTS

4.1 ANALYSIS OF THE DATA SET

Since we stored each image as a column in our data matrix, the SVD has a standard interpretation. As U forms an orthonormal basis for the codomain of our data matrix and the codomain is the space where each digit image exists, the U matrix is our coordinate system for digit-space. We can see this when we plot the columns of U which make of the basis of digit-space. Each principal component captures some of the general features of the digits. Similarly, just by considering the matrix multiplication in $A = U\Sigma V^*$, we see that after scaling the principal components by Σ, each column of V^*gives the coordinates of each image, a column in our original data matrix. In this digit-space defined by U: each digit in A is a linear combination of the columns of U weighted by the columns of ΣV^*

When working with the digits, we see that the averaging and normalizing works quite well. We see when taking the SVD, that we are able to drastically reduce the size of our system. As seen in **Figure 1,** it takes about 192 modes to capture about 90% of the energy in the system or a rank 192. Of course, this varies when looking at between different classes of digits and the entire range of digits. By reducing the dimension of our system, we can greatly improve the amount of computational work we can accomplish with it while also reducing any overfitting that might come from trying to extend this system, important when it comes to classifying the digits.

The benefit of PCA comes in analyzing the principal components and thus being able to analyze the important features of the system. We can get a better look at the digits in relation to each other by projected onto three selected modes colored by their digit label. With the first three principal components we can analyze how close the features of some digits are to each other. As seen in **Figure 2,** we are able to get a good idea of the features of each digit with only three principal components. While there is some blurring between the digits and not all of them are well separated, with only three principal components, it does a remarkable job at analyzing the system. We see that digits equal to 2 might be a little easier to separate from digits equal to 5, than digits equal to 3 and 5. Now that we have taken the PCA of our system it is time to look at the classification.

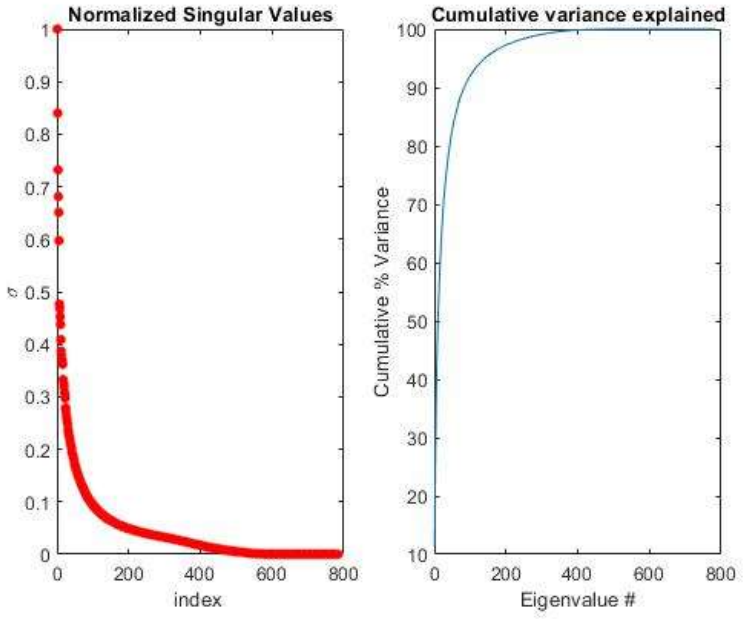

Figure 1: Normalized Singular Values and Variance obtained from taking SVD

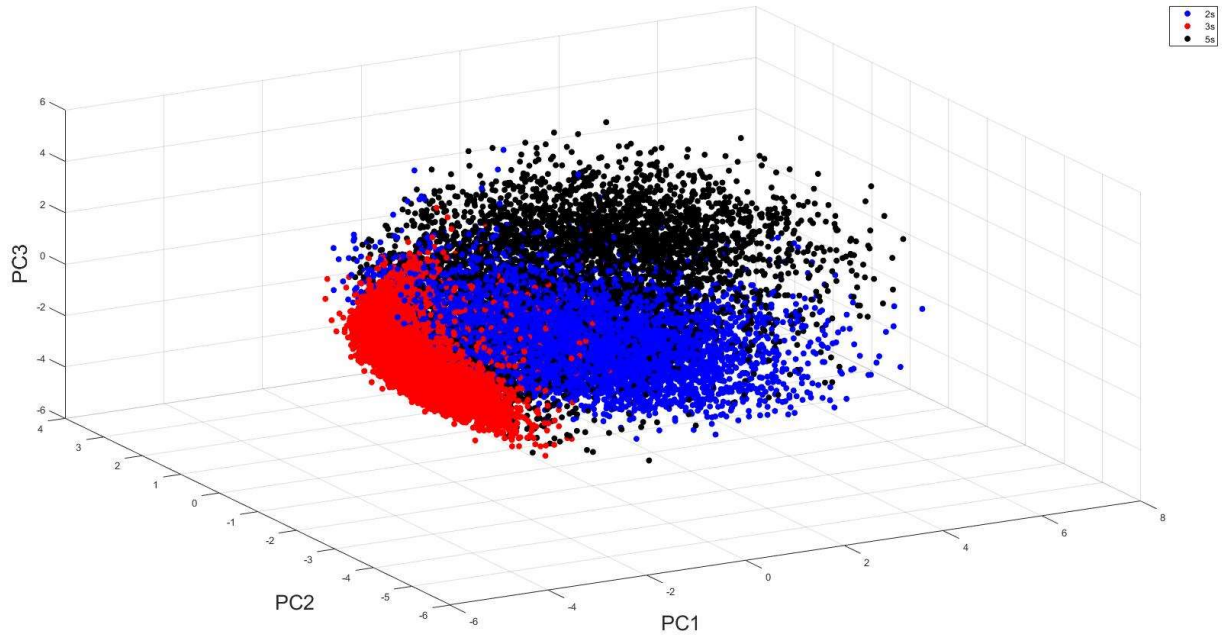

Figure 2: PCA projection for the digits of 2 (blue), 3 (red), and 5 (black).

4.2 LDA CLASSIFIER

Using SVD we can break down the patterns of each digit into a coordinate system that is more centered around their fundamental structures. Rather than using the entire data given for each image, we can train our classifiers using the coordinate system given by this system. This will help develop our classifiers by providing a low-dimensional space for them to train in.

In the first test case, we build a model to differentiate the digits of 0 from the digits of 1. After running the model we saw that Linear Discriminant Analysis was able to successfully differentiate the two digits quite well. As seen in **Figure 3a,** for about 2000 total digit images, our LDA model only misclassified 9 total images for a total error rate of less than 0.4%. As

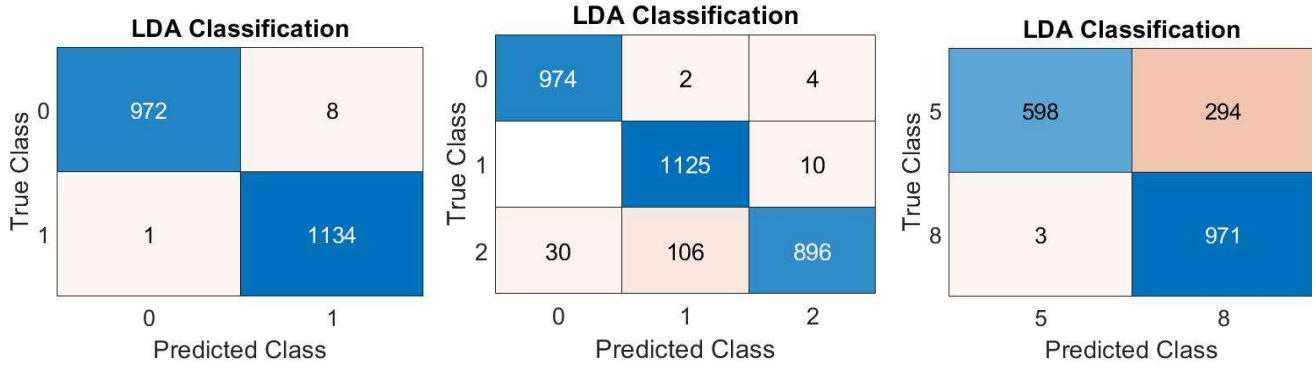

Figure 3:(Left): a) Confusion chart for LDA on digits of 0 and 1. (Middle): b) Confusion chart for LDA on digits of 0,1, and 2. Right: c) LDA, 5, 8 digits

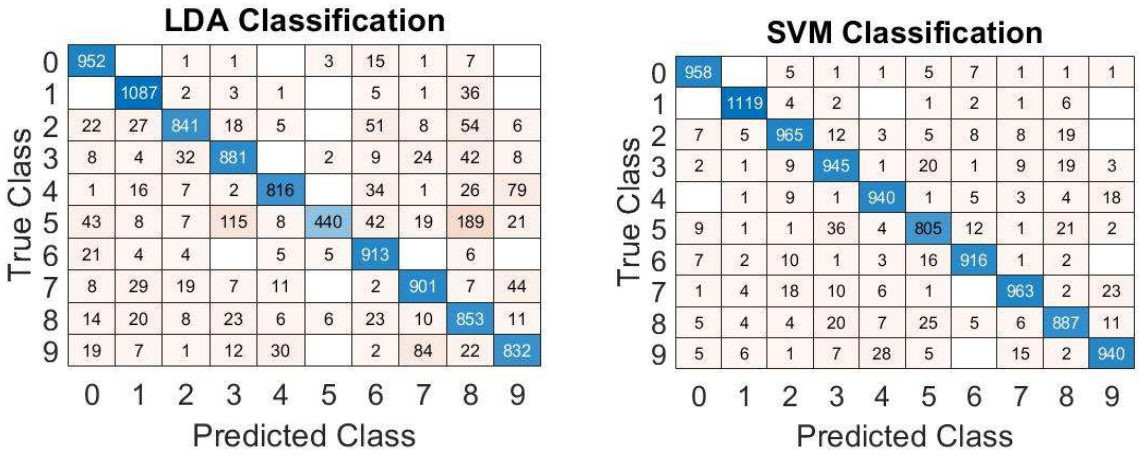

Figure 4: (Left) a) Confusion chart for LDA on all the digits in the data set. (Right) b) Confusion chart for SVM on all digits in the data set.

seen in **Figure 3b,** for about 3000 total digit images, our LDA model only misclassified 150 total images for an error rate less than 5%. There were a few pairs of digits that were easy to separate such as 5 and 4, and 0 and 1. However, I thought that the most consistently easy to separate digit was 0 and 1, because the error was less than a half of a percent for every iteration. As seen in **Figure 3c,** for a little less than 2000 total digit images of 5 and 8, our LDA model misclassified about 300 total images for an error of about 16%. While this error is not terrible, it is significantly more than other pairs of digits. The only other digits that were difficult to distinguish were 5 and 3, and maybe 9 and 7. Another interesting aspect is that the classifier was able to distinguish 8s and 8s very well, but it kept predicting an 8 when the digit was actually a 5. Overall, it seemed that the classifier did not separate 5s very well from the rest of the images. This may have to do with the fact that the digit 5 looks very similar to many other digits, especially in handwriting.

4.3 Comparison Between Classifiers

The LDA classifier did a superb job at classifying the entire range of digits. As seen in **Figure 4a,** almost every single digit was easy to classify except maybe the digit 5. When comparing this to Support Vector Machines (SVM) and Decision Tree Classifiers (DTC), we are seeing very similar results. Interestingly enough, for SVM it seemed that the classifier did a much poorer job when the images were reduced in rank versus when the images were not projected into a lower dimension using PCA. As seen in **Figure 4b,** it seems that SVM did about the same in classifying as LDA does. The error is kept below 5% and almost all digits are easily classified. Unfortunately I was unable to make a confusion chart for DTC**,** but we see that the error rate for the DTC also seems close to the error rates in other classifiers. Although, it may be a little lower than the error rate for the LDA on average, as it seems that DTC does a better job at differentiating 5 from other digits.

Error Table			
Digits	LDA	SVM	DCL
0,1	0.0043	0.00095	0.0042
5,8	0.159	0.0418	0.109

Figure 5: Error rate for the different classifiers on the hardest and easiest pair of digits to separate.

The error rates between LDA, SVM, and decision trees does not change drastically between different classifiers as seen in **Figure 5.** They all have very low rates of error so it is difficult to judge them with absolute certainty. However, the error rates start to differ when looking at the most difficult to separate digits. LDA was the worst at separating 5 and 8, SVM was the best at separating them, and DCL was right in between. This may be because LDA relies on the difference between the prominent features in the numbers, so when the shape of numbers look similar it can struggle.

5 Summary and Conclusions

We saw that while SVD was not as efficient at breaking down the dimensions of the system as in other examples, reducing the size of the image data from 784 to the rank of 192 is still a remarkable reduction. The only difficulty is the amount of preprocessing needed in order to take the SVD the most efficiently then analyze the system using PCA. It was remarkable at how much the data was differentiable only using three PCA modes. From the low-rank approximations we can pick out only the principal components that best separate the data. Rather than working with the entire data set we only have to work with the coordinates of our data in this reduced - dimension space. This allows us to train on larger datasets and improve the quality of our prediction in the Classification section. As we saw in the classification section, when we apply this SVD decomposed system to a classification problem, we are able to increase the runtime without sacrificing accuracy. There seemed to be a large difference in the effectiveness of our classifier based on the digits we were trying to separate. While LDA had the fastest run time, this sometimes sacrificed accuracy when it came to separating digits like 5 and 8. Compare this to the methods of SVM and DCL that took a while longer to run, and sometimes they performed a lot better. Overall, using PCA with LDA did a remarkable job at lowering the dimensionality of the system and providing accurate results.

6 Appendix A

Below are the following MATLAB functions I used and a brief description of their functions:

Confusionchart(test labels, predicted labels): Creates a confusion chart based on the predicted labels and the given test labels.

fitcdiscr(training data, labels): Generates a predictive model using LDA, training it with the given data and associated labels.

Fitecoc(training data, labels): Generates a predictive model using multi-class SVM, training it with the given data and associated lables.

Fitctree(training data, labels): Generates a predictive model using Decision Tree learning, training it with the given data and its associated labels.

LDA_Trainer((first_number,second_number,feature): Generates a threshold value and the projection model for LDA based on the training data and the rank preferred.

loadMNISTImages(filename): Loads the MNIST data from the downloaded files into a n by m matrix.

predict(Mdl,test data): Generates the predicted labels for any of the classifiers models using the test data.

SVD(X): Calculates the Singular Value Decomposition of the given matrix X, returning U, Σ, V .

7 APPENDIX B

MATLAB Code

Main Code:

```
clear,clc
%Michael Gabalis
% PCA for clustering MNIST Images
%% Loading in the data
[testimages,testlabels] = MNISTload('t10k-images.idx3-ubyte','t10k-labels.idx1-ubyte');
[images,labels] = MNISTload('train-images.idx3-ubyte','train-labels.idx1-ubyte');
%% Extracting different digits8 a = 5;
b = 8;
c = 5;
images = images(:,labels == a |labels ==b |labels==c);
%images = images(:,labels == a |labels ==b);
labels = labels(labels == a | labels ==b |labels ==c,:);
%labels = labels(labels == a | labels ==b,:);
testimages = testimages(:, testlabels==a |testlabels ==b |testlabels ==c);
testlabels = testlabels(testlabels==a | testlabels == b |testlabels ==c,:);
%testimages = testimages(:, testlabels==a |testlabels ==b);
%testlabels = testlabels(testlabels==a | testlabels == b,:);19
%% Mean centering and pixe l covariance
[numpixels,numimages] = size(images);
%mean of each pixel
mu = mean(images,2);
%Normalization
X = images - repmat(mu,1,numimages);
images_centered = X/sqrt(numimages-1);27

%% SVD and reconstructing the covariance
[U,S,V] = svd(images_centered,'econ');
eigs = diag(S*S'/(numimages-1))';32
%% Variance explained per Principal Component

var = 100* cumsum(eigs)./sum(eigs);
figure(1)
subplot(1,2,1)
plot(diag(S)./ max(diag(S)),'r.','markersize',15);
title('Normalized Singular Values')
xlabel('index ')
ylabel('\sigma')42
subplot(1,2,2)
plot(1:numpixels, var);
title('Cumulative variance explained')
xlabel('Eigenvalue #')
ylabel('Cumulative % Variance')48
%% Finding the energy of the system
```

```
energy = 0;
total = sum(diag(S));
% how much energy we want our modes to capture.
% 75% does alright; 90% does very good.
threshold = 0.90;55 r =
0;
while energy < threshold
    r = r + 1;
    energy = energy + S(r,r)/total;
end
rank = r;
Proj = U(:,1:rank)'*images_centered;
%% Vizualizing the eigenvectors as images63
figure(3)
for k = 1:9
    subplot(3,3,k)
    imagesc(reshape(U(:,k),28,28));
    colormap gray;
end

%% Rotate mean subtracted data and vizualize projections

figure

scatter3(Proj(1,labels==0), Proj(2,labels==0),Proj(3,labels==0), 'b','filled')
hold on77
scatter3(Proj(1,labels==1), Proj(2,labels==1),Proj(3,labels==1), 'r','filled')
hold on80
 scatter3(Proj(1,labels==2), Proj(2,labels==2),Proj(3,labels==2), 'k','filled')
legend('2s', '3s', '5s')
xlabel('PC1', 'fontsize', 20)
ylabel('PC2', 'fontsize', 20)
zlabel('PC3', 'fontsize', 20)
hold off
 box off88
%% Use LDA on only two digit numbers
first = Proj(:,labels==1);
second = Proj(:,labels==0);
[U2,S2,V2,threshold,w,sortones,sorttwos]=LDA_trainer(first,second,rank); 93
Testnum = size(testimages,2);
Proj3 = U2(:,1:rank)'*(testimages(1:rank,:));
pval = w'*Proj3;97
Res = (pval>threshold);
err = abs(Res'-testlabels);
errNum = sum(err);
success = 1-errNum/Testnum;
```

```
%% Use LDA on two or more digits, plot error
Proj2 = U(:,1:rank)'*(testimages);
Mdl = fitcdiscr(Proj',labels, 'DiscrimType','pseudolinear');
labeled = predict(Mdl,Proj2');
confusionchart(testlabels,labeled)
title('LDA Classification')
set(gca,'fontsize',20)
%% Perform SVM
SVM = fitcecoc(images', labels);
test_labels = predict(SVM,testimages');112
confusionchart(testlabels,test_labels)
title('SVM Classification')
set(gca,'fontsize',20)
%% Perform Decision Tree
Tree = fitctree(Proj', labels,'MaxNumSplits',3,'CrossVal','on');
%test_labels = predict(Tree,Proj2');
%confusionchart(testlabels,test_labels)
classError = kfoldLoss(Tree)121
122
```

LDA_Trainer

```
function [U,S,V,threshold,w,sortones,sorttwos] = LDA_trainer(first_number,
second_number,feature)

    nd = size(first_number,2);
    nc = size(second_number,2);5
    [U,S,V] = svd([first_number second_number],'econ');
    numbers = S*V';
    U = U(:,1:feature); % Add this in
    ones = numbers(1:feature,1:nd);
    twos = numbers(1:feature,nd+1:nd+nc);
    md = mean(ones,2);
    mc = mean(twos,2);13
Sw = 0;
for k=1:nd
Sw = Sw + (ones(:,k)-md)*(ones(:,k)-md)';
    end
    for k=1:nc
        Sw = Sw + (twos(:,k)-mc)*(twos(:,k)-mc)'; % within class matrix
    end
    Sb = (md-mc)*(md-mc)'; %between class scatter matrix22
    [V2,D] = eig(Sb,Sw);
    [lambda,ind] = max(abs(diag(D)));25      w = V2(:,ind);
    w = w/norm(w,2);
    vfnumber = w'*ones;
        vsnumber = w'*twos;29
if mean(vfnumber)>mean(vsnumber)31
    w = -w;
        vfnumber = -vfnumber;
        vsnumber = -vsnumber;
    end
```

```
 35
36      % Don't need plotting here
 37       sortones = sort(vfnumber);
 38       sorttwos = sort(vsnumber);
 39       t1 = length(sortones);40 t2 = 1;
 41  while  sortones(t1)>sorttwos(t2)42       t1 =
     t1-1;
 43  t2 = t2+1;
 44       end
 45       threshold = (sortones(t1)+sorttwos(t2))/2;46
 47       % We don't need to plot results
 48  end49
 50
```

MNISTLoad Code:

```
  1  %% MNIST Image Loading Functions
  2  % Not my Code, changed from Professor Bramburgers code
  3  function [images,labels] = MNISTload(images_location,labels_location)
  4  %loads mnist data from train,t10k files
  5  %load images
  6  images = loadMNISTImages(images_location);
  7  %load labels
  8  labels = loadMNISTLabels(labels_location);
  9  end10
 11  function images = loadMNISTImages(filename)
 12  %loadMNISTImages returns a 28x28x[number of MNIST images] matrix containing
 13  %the raw MNIST images14
 15  fp = fopen(filename, 'rb');
 16  assert(fp ~= -1, ['Could not open ', filename, '']);17
 18  magic = fread(fp, 1, 'int32', 0, 'ieee-be');
 19  assert(magic == 2051, ['Bad magic number in ', filename, '']);20
 21  numImages = fread(fp, 1, 'int32', 0, 'ieee-be');
 22  numRows = fread(fp, 1, 'int32', 0, 'ieee-be');
 23  numCols = fread(fp, 1, 'int32', 0, 'ieee-be');24
 25  images = fread(fp, inf, 'unsigned char');
 26  images = reshape(images, numCols, numRows, numImages);
 27  images = permute(images,[2 1 3]);28
 29  fclose(fp);30
 31  % Reshape to #pixels x #examples
 32  images = reshape(images, size(images, 1) * size(images, 2), size(images, 3));
 33  % Convert to double and rescale to [0,1]
 34  images = double(images) / 255;35
 36  end37
 38  function labels = loadMNISTLabels(filename)
 39  %loadMNISTLabels returns a [number of MNIST images]x1 matrix containing
 40  %the labels for the MNIST images41
 42  fp = fopen(filename, 'rb');
```

```
43 %assert(fp ~= -1, ['Could not open ', filename, '']);44
45 magic = fread(fp, 1, 'int32', 0, 'ieee-be');
46 assert(magic == 2049, ['Bad magic number in ', filename, '']);47
48 numLabels = fread(fp, 1, 'int32', 0, 'ieee-be');49
49     labels = fread(fp, inf, 'unsigned char');
50
51
52 assert(size(labels,1) == numLabels, 'Mismatch in label count');53
54 fclose(fp);55
56 end
```