AMATH 582 Homework 4

Tomas Perez

March 10, 2021

Abstract

In this paper, image recognition and simple machine learning techniques are utilized to classify images of hand-written digits. The analysis involves the application of singular value decomposition (SVD) and the use of classification models. Three classifiers are used; a Linear Discriminant Analysis classifier, a Support Vector Machines classifier, and a Classifiaction Tree. The accuracy of these models are then compared.

1 Introduction and Overview

The data comes from the MNIST data set of handwritten digits. Much of the reprocessing is already complete, with all digits being size-normalized and centered, and gray-scaled. The training data is composed of 60,000 $28x28$ images, with each digit having at least 5,000 images. The testing data has 10,000 images. In the first part of this paper, SVD analysis is performed. With the data projected into the PCA space, classifiers are used to identify the individual digits. The classifiers explored in this paper are linear discriminant analysis (LDA) classifiers, Support Vector Machine (SVM) classifiers, and classification trees.

2 Theoretical Background

2.1 Singular Value Decomposition

Singular Value Decomposition (SVD) is linear algebra method for factorization of a transformation matrix A by identifying orthonormal bases U and V and a diagonal component of singular values. Applying the transformation matrix A to a vector x changes the magnitude and direction of the vector. SVD decomposes the transformation matrix A into a two rotational matrices U and V, and a scaling matrix Σ.

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

Where U is a $m \times n$ orthonormal matrix representing the feature space, Σ is a $n \times n$ diagonal matrix of singular values that represents the strength or scaling of a projection, and V is a $n \times n$ unitary basis matrix representing the projection of column onto the principal modes. This is called the reduced-form SVD of A. By defining the transformed variable $Y = U^*X$, we can calculate the covariance of Y.

$$\begin{aligned} C_Y &= \frac{1}{n-1} YY^T \\ &= \frac{1}{n-1}(U^*X)(U^*X)^T \\ &= \frac{1}{n-1}U^*(XX^T)U \\ &= \frac{1}{n-1}U^*U\Sigma^2UU^* \\ &= \frac{1}{n-1}\Sigma^2 \end{aligned}$$

In comparison with normal eigendecomposition, we could see that Σ^2 is equivalent to λ. The covariance of Y is the diagonal matrix. By taking a standardized X and scaling it by $\frac{1}{n-1}$ we can get the reduced-form SVD of X^T. The columns of U are movement of the paint can along the PCA modes, and the columns of V are the horizontal and vertical directions of the PCA modes.

$$X^T = U\Sigma V^* \tag{2}$$

2.2 Classification

Linear Discriminant Analysis (LDA) is a statistical method used to find a linear combination of features that separates multiple classes of a data set. The idea is to find the optimal projection basis that maximizes the distance between the between-class data while minimizing the within-class data. The projection w is constructed such that,

$$\mathbf{w} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{3}$$

where $\mathbf{S}_B$ is the scatter matrix for between-class data and $\mathbf{S}_W$ is the scatter matrix for within-class data. $\mathbf{S}_B$ and $\mathbf{S}_W$ measure the variance within the data classes, as well as the variance of the difference in means between classes. are defined as follows,

$$\begin{aligned} \mathbf{S}_B &= (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \\ \mathbf{S}_W &= \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T \end{aligned} \tag{4}$$

Classification trees are models where classes can take a discrete set of values. Each node (feature) is evaluated to determine which class the value points to. Support Vector Machines (SVM) find an optimal way to separate data clusters by trying to maximize the distance between the best fit lines for each class projection.

3 Algorithm Implementation and Development

Algorithm 1: SVD

Load and reshape the data so that one column is an image
Select digits to analyze
Perform SVD and get the U,Σ, and V matrices
ΣxV gives the strength of the digit projections on the PCA modes

The general algorithm for LDA, classificaiton trees, and SVM are very similar thanks to the use of preexisting MATLAB funcitons. Though differing slightly for each classification method the general process is as follows:

Algorithm 2: Classification

Select digits to analyze
Build matrices for testing and training data, or build a matrix of the testing data projected onto the principle component basis for the training data
Specify the corresponding digit labels for each data set
Test model on based on the training data
Compute the error and overall accuracy

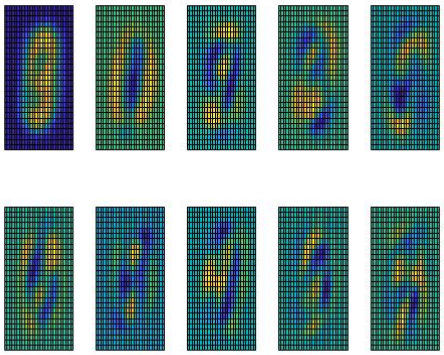

Figure 1: Principal Components

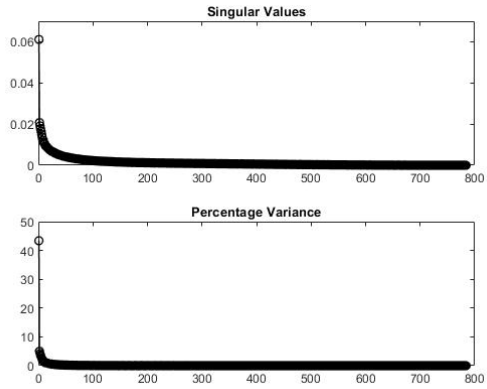

Figure 2: Singular Values

Figure 3: Digits with 50 Features

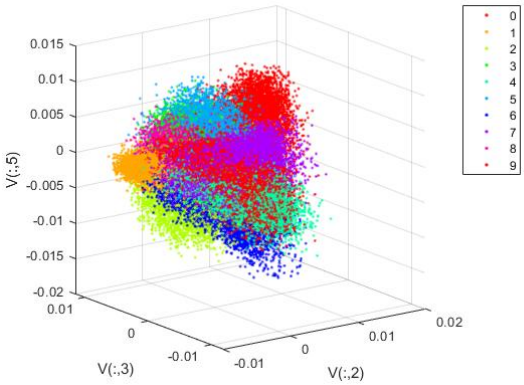

Figure 4: V-modes

4 Computational Results

4.1 SVD

Figure 1 shows a representation of the first 10 principal components of the handwritten digits. Some digits are more recognizable at certain modes. Figure 2 displays the singular values of the data, and the percentage variance explained by the PCA mode. The first singular value is clearly the dominant one. The features required for accurate evaluation can be between 1 and 784. When reconstructing the images from the SVD factorized matrices most of the digits become reasonably distinguishable at approximately 50 features. This is shown in Figure 3. Figure 4 presents the different digits projected onto three V-modes; 2, 3, and 5.

4.2 LDA

To illustrate LDA we first compare two digits that should be easily classifiable. For this example, digits 0 and 1 are used. Figure 5 shows a histogram representing the projection of the digit images onto the LDA basis. The red line is the decision threshold used to classify the digit. Figure 6 shows the projection of the first 100 images of digits 0 and 1 onto the first V-modes. When run through the LDA classifier using

Test 3

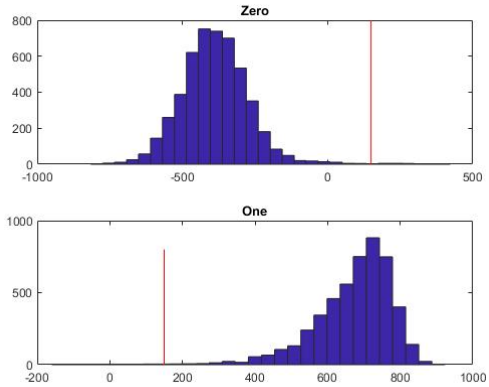

Figure 5: Decision Threshold

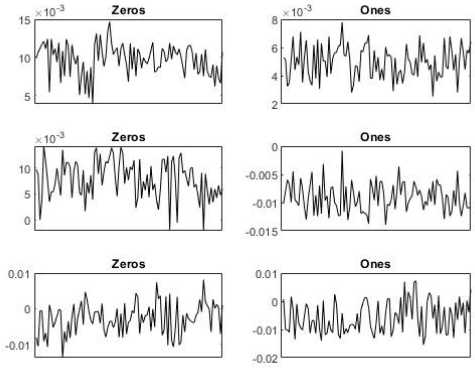

Figure 6: Sigma and Principal Component

Test 4

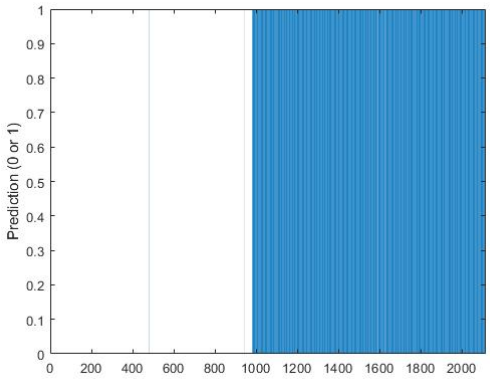

Figure 7: Prediction: Zero or One

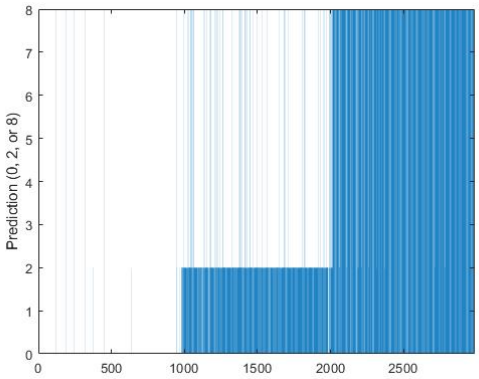

Figure 8: Prediction: Zero or Two or Eight

MATLAB's **classify** function, we achieve and accuracy rating of 99.4%. The results are shown in Figure 7. LDA can be extended to more than 2 classes as well. Figure 8 shows the prediction outcomes of the test data for digits 0, 2, and 8. In both Figure 7 and Figure 8 the results are displayed in a bar graph with each bar representing the predicted class of a digit. You can see in Figure 8, that the similarities between digits 2 and 8 resulted in more misclassifications. The overall accuracy for the LDA classifier with these three digits was roughly 96%.

In repeated runs of the 2 digit LDA classifier, the two easiest digits to classify were 0 and 1 with an accuracy score of 99.4% while the two most difficult digits to classify were 4 and 9 with an accuracy score of 81.7%.

4.3 Classification Tree and SVM

The implementation of the classification tree and SVM models are done with MATLAB's **fitctree** and **fitcsvm** functions respectively. Both classification models are run on all ten digits. The classification tree achieved an accuracy of approximately 84%, and the SVM classifier achieved an accuracy of approximately 93%.

Next we compare the easiest (0 and 1) and hardest (4 and 9) digits to classify using all three classifiers. As we mentioned before, LDA achieved accuracy scores of 99.4% and 81.7% for the easiest and hardest pairs respectively. The classification tree was 99.3% accurate for the easiest pair and 87.7% accurate for the hardest pair. SVM had accuracy ratings of 99.5% and 93.8% for the easiest and hardest pairs.

5 Summary and Conclusions

This study illustrates the application of SVD and various classification techniques on the MNIST data. The training data is first analyzed using SVD, then used to train the LDA, SVM, and classification tree models. Prediction are taken using the testing data and the accuracy is compared. The LDA model was explored first. The model proved to be more accurate when distinguishing two digits. The more digits you attempt to classify in one model, the less accurate the predictions. The model was very successful with digits that were more clearly distinct, such as 0 and 1, but suffered with more similarly shaped digits, mainly 4 and 9. The same digits are compared in the SVM model and classification tree. When classifying easily separable digits, all the models performed similarly with an accuracy of over 99%. There was a much more noticeable difference between the classifiers when testing difficult digit pairs. Each classifier had about a 6 percentage point increase in accuracy compared to the previous model with LDA performing the worst, then the classification tree, and then the SVM performing the best.

Appendix A MATLAB Functions

- `I = find(X)` returns a vector containing the linear indices of each nonzero element in array X
- `M = mean(A)` returns the mean of the elements of A along the first array dimension whose size does not equal 1
- `scatter3(X,Y,Z` displays circles at the locations specified by the vectors X, Y, and Z.
- `P = classify(Test,Train,Group)` classifies each row of the data in Test into one of the groups in Train
- `[M,I] = max(A)` returns the maximum element of an array and the index into the operating dimension that corresponds to the maximum value of `A`
- `[V,D]=eig(A)` returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that A*V = V*D
- `D=diag(v)` returns a square diagonal matrix with the elements of vector v on the main diagonal
- `tree = fitctree(Tbl,ResponseVarName)` returns a fitted binary classification decision tree based on the input variables (also known as predictors, features, or attributes) contained in the table Tbl and output (response or labels) contained in Tbl.ResponseVarName.
- `Mdl = fitcsvm(Tbl,ResponseVarName)` trains or cross-validates a support vector machine (SVM) model for one-class and two-class (binary) classification on a low-dimensional or moderate-dimensional predictor data set
- `[U,S,V]=svd(A)` performs singular value decomposition of matrix A, such that A=U*S*V

Appendix B MATLAB Code

For brevity, the code below is one cycle of the classifiers. The code is simply rerun again and again for when comparing each digit and minor changes were made to run the code for 3 digits.

```
[TrainImages, TrainLabels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
trainingData = double(reshape(TrainImages, size(TrainImages,1)*size(TrainImages,2), []));
trainingLab = double(TrainLabels);

%SVD over the whole training dataset
[U,S,V]=svd(trainingData,'econ');
lambda2=diag(S).^2;
sigma =diag(S);
PComp=U*trainingData;
digits=S*V';
%display the first 10 columns of U (PCA modes)
figure(1)
for j=1:10
    subplot(2,5,j)
    ut1=reshape(U(:,j),28,28);
    ut2=ut1(28:-1:1,:);
    pcolor(ut2)
    set(gca,'Xtick',[],'Ytick',[])
end
figure(2)
subplot(2,1,1), plot(sigma,'ko','Linewidth',[1])
axis([1 50 0 500000])
%xlim([1 10])
title('Singular Values')
subplot(2,1,2), plot(lambda2/sum(lambda2)*100,'ko','Linewidth',[1])
axis([1 50 0 100])
%xlim([0 10])
title('Percentage Variance')

Ind0=find(trainingLab==0);
Ind1=find(trainingLab==1);
Ind2=find(trainingLab==2);

for j=1:5000 %take the first 5000 images
    ind0=Ind0(j);
    Matrix0(:,j)=trainingData(:,ind0);
    ind1=Ind1(j);
    Matrix1(:,j)=trainingData(:,ind1);
    ind2=Ind2(j);
    Matrix2(:,j)=trainingData(:,ind2);
end

feature=50;
trainMat=[Matrix0,Matrix1,Matrix2];
%trainMat=[Matrix0,Matrix1];
[U,S,V]=svd(trainMat,'econ');
lambda2=diag(S).^2;
sigma=diag(S);
digits=S*V';
U=U(:,1:feature);
Digit0=digits(1:feature,1:5000);
Digit1=digits(1:feature,5001:10000);
Digit2=digits(1:feature,10001:end);
```

```
%display the first 10 columns of U (PCA modes)
for j=1:4
    subplot(2,2,j)
    ut1=reshape(U(:,j),28,28);
    ut2=ut1(28:-1:1,:);
    pcolor(ut2)
    set(gca,'Xtick',[],'Ytick',[])
end

figure(2)
subplot(2,1,1), plot(sigma,'ko','Linewidth',[1])
axis([1 50 0 200000])
title('Singular Values')
subplot(2,1,2), plot(lambda2/sum(lambda2)*100,'ko','Linewidth',[1])
axis([1 50 0 100])
title('Percentage Variance')

figure(3)
for j=1:3
    subplot(3,2,2*j-1)
    plot(1:100,V(1:100,j),'k-')
    title('Zeros')
    set(gca,'XTick',[])
    subplot(3,2,2*j)
    plot(5001:5100,V(5001:5100,j),'k-')
    title('Ones')
    set(gca,'XTick',[])
    %subplot(3,3,3*j)
    %plot(10001:10100,V(10001:10100,j),'k-')
    %title('Twos')
end
%LDA
m0=mean(Digit0,2);
m1=mean(Digit1,2);
%m2=mean(Digit2,2);
Sw=0; %within class variance
for i=1:5000
    Sw=Sw+(Digit0(:,i)-m0)*(Digit0(:,i)-m0)';
end
for i=1:5000
    Sw=Sw+(Digit1(:,i)-m1)*(Digit1(:,i)-m1)';
end
%for i=1:5000
%    Sw=Sw+(Digit2(:,i)-m2)*(Digit2(:,i)-m2)';
%end
%Sb=(m0-m1-m2)*(m0-m1-m2)'; %between clas varaince
Sb=(m0-m1)*(m0-m1)';
```

```
[V2,D] = eig(Sb,Sw); % linear discriminant analysis
[lambda,ind] = max(abs(diag(D)));
w = V2(:,ind); w = w/norm(w,2);
v0 = w'*Digit0; v1 = w'*Digit1; %v2=w'*Digit2
result = [v0,v1];
if mean(v0)>mean(v1)
    w=-w; v0=-v0; v1=-v1;
end
sort0=sort(v0);
sort1=sort(v1);

t1 = length(sort0);
t2 = 1;
while sort0(t1)>sort1(t2)
    t1 = t1-1;
    t2 = t2+1;
end
threshold = (sort0(t1)+sort1(t2))/2;

figure(4)
subplot(2,1,1)
hist(sort0,30); hold on, plot([threshold threshold],[0 800],'r')
title('Zero')
subplot(2,1,2)
hist(sort1,30,'r'); hold on, plot([threshold threshold],[0 800],'r')
title('One')
```

```
[TrainImgs, TrainLabels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
[TestImgs, TestLabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');

[training_rows,training_cols,training_images]=size(TrainImgs);
[testing_rows,testing_cols,testing_images]=size(TestImgs);
for i=1:training_images
    training_data(:,i)=double(reshape(TrainImgs(:,:,i),training_rows*training_cols,1));
end
for i=1:testing_images
    testing_data(:,i)=double(reshape(TestImgs(:,:,i),testing_rows*testing_cols,1));
end
% SVD
[U,S,V]=svd(training_data,'econ');
sigma =diag(S); lambda2=diag(S).^2;
digits=(S*V')';
figure(1)
for j=1:10
    subplot(2,5,j)
    ut1=reshape(U(:,j),28,28);
    ut2=ut1(28:-1:1,:);
    pcolor(ut2)
    set(gca,'Xtick',[],'Ytick',[])
end
figure(2) %singular values
subplot(2,1,1), plot(sigma/sum(sigma),'ko-','Linewidth',[1])
axis([0 800 0 0.07])
title('Singular Values')
subplot(2,1,2), plot(lambda2/sum(lambda2)*100,'ko-','Linewidth',[1])
axis([0 800 0 50])
title('Percentage Variance')
% V-modes
figure(3);
for i=0:9
    ind = find(TrainLabels==i);
    scatter3(V(ind,2),V(ind,3),V(ind,5),20,TrainLabels(ind),'.')
    hold on
end
xlabel('V(:,2)'),ylabel('V(:,3)'),zlabel('V(:,5)')
legend({'0','1','2','3','4','5','6','7','8','9'});
colormap hsv
```

```
% LDA
feature=50;
train0=digits(TrainLabels==0,1:feature);
train2=digits(TrainLabels==2,1:feature);
train8=digits(TrainLabels==8,1:feature);
[row0,col0]=size(train0);
[row2,col2]=size(train2);
[row8,col8]=size(train8);
train_vec=[train0; train2; train8];
group1=[0*ones(row0,1); 2*ones(row2,1); 8*ones(row8,1)];
PComp=(U'*testing_data)';
test0=PComp(TestLabels==0,1:feature);
test2=PComp(TestLabels==2,1:feature);
test8=PComp(TestLabels==8,1:feature);
[row0,col0]=size(test0);
[row2,col2]=size(test2);
[row8,col8]=size(test8);
test1=[test0; test2; test8];
group2=[0*ones(row0,1); 2*ones(row2,1); 8*ones(row8,1)];
prediction=classify(test1,train_vec,group1);
error=sum(abs(group2-prediction)>0);
accuracy=1-error/length(group2);
figure(5)
bar(prediction)
ylabel('Prediction (0, 2, or 8)')
% Classifier Tree
train1=digits(:,1:feature);
PComp=(U'*testing_data)';
test1=PComp(:,1:feature);
tree = fitctree(train1,TrainLabels,'OptimizeHyperparameters','auto');
predection=predict(tree,test1);
error=sum(abs(TestLabels-predection)>0);
accuracy=1-error/length(TestLabels);
% SVM
train1=digits(:,1:feature)/max(max(S));
test1=PComp(:,1:feature)/max(max(S));
for j = 0:9
    ind = TrainLabels==j;
    models{j+1} = fitcsvm(train1,ind,'ClassNames',[false true],'Standardize',true,...
        'KernelFunction','rbf','BoxConstraint',1);
end
for j = 0:9
    [~,pred] = predict(models{j},test1);
    acu(:,j) = pred(:,2);
end
[~,M] = max(acu,[],2);
error=sum(abs(TestLabels+1-M)>0);
accuracy=1-error/length(TestLabels)
```