AMATH 482 Assignment 4:

Classifying Digits

Yuqing Cui
March 10, 2021

Abstract

In this report, we are given four packs of the MNIST data set, which are sets and labels used for training and test. We are going to apply the singular value decomposition (SVD) on the data set and evaluate the performance of different classifiers accordingly. The classifiers we implemented in this report are the linear discriminant analysis (LDA), the support vector machines (SVM), and the decision tree classifier.

I. Introduction and Overview

As a part of heating topics in artificial intelligence, machine learning has been implemented by many experts combining knowledge from other science fields such as biology. In this report, we are going to classify handwritten digits from the MNIST data set to examine the accuracy of three classifiers: LDA, SVM, and the decision tree. The MNIST data set contains 60000 training images and 10000 test images which would be sufficient for our analysis. We will do two cases. The first case is making prediction with the training set and examine the result using the test set. On the other hand, we are going to using the training set to evaluate the performance of our classifiers trained with the test set. The purpose is to check the adequacy of our classifiers by analyzing if the results are stable or not. For LDA, we will classify over two digits, three digits, and ten digits at a time and identify the easiest and hardest pair to separate. For SVM and the decision tree, we will apply on ten digits at once and compare their accuracy on the two pairs as well.

II. Theoretical Background

(a) Singular Value Decomposition (SVD)

The singular value decomposition (SVD) reforms an $m \times n$ matrix $\mathbf{A}$ into the following format:

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

in which $\mathbf{U}$ is an $m \times m$ unitary matrix that contains principle components. $\mathbf{\Sigma}$ is an $m \times n$ diagonal matrix that contains singular values. $\mathbf{V}$ is an $n \times n$ unitary matrix with time information. $\mathbf{U}$ and $\mathbf{V}$ are orthonormal to each other. All diagonal elements of $\mathbf{\Sigma}$ are non-negative and are sorted from the largest (upper-left) to the smallest (lower-right). The number of nonzero singular values is the rank of matrix $\mathbf{A}$. The matrix multiplication will perform rotations and stretches. Besides, SVD enables every matrix to be diagonal with proper bases for the domain and the range.

(b) Linear Discriminant Analysis (LDA)

The linear discriminant analysis (LDA) is used to find a suitable projection that maximizes the distance between the inner-class while minimizing the intra-class data. That is, we would like to find a vector w, where

$$w = arg\,max \frac{w^T S_B w}{w^T S_W w}, \tag{2}$$

S_B is the between-class scatter matrix measuring variances between groups by:

$$S_B = \sum_{j=1}^{N} m_j (u_j - u)(u_j - u)^T, \tag{3}$$

Where m_j is the number of samples in each class, u_j is the mean of class j, and u is the mean of all the samples.
The within-class scatter matrix S_W is defined as:

$$S_W = \sum_{j=1}^{N} \sum_x (x - u_j)(x - u_j)^T, \tag{4}$$

With the two matrices, we can solve for w in equation (2) using the characteristics of eigenvalue:

$$S_B w = \lambda S_W w, \tag{5}$$

III. Algorithm Implementation and Development

We first loaded the MNIST data set. The training set was a $28 \times 28 \times 60000$ matrix with 60000 images of digits of size 28×28. Similarly, the test set was a $28 \times 28 \times 10000$ matrix with 10000 images of digits of size 28×28. We reshaped the two matrices into the size 784×60000 and 784×10000 respectively in order to apply SVD. Now each column represents a different image. After implementing SVD, we plotted the singular value spectrum and the 3D projection of V-modes using MATLAB built-in functions. We found out that 9 V-modes would create an acceptable accuracy overall, so we decided to take the 2 to 10 columns of the matrix to test our classifiers.

When examine the performance of the classifiers, we first made prediction with the training set and evaluated the accuracy with the test set. Then we repeated a similar process with predicting using the test set and calculating the accuracy using the training set to make sure our model is actually suitable.

For LDA, we used the built-in function `classify`. In the beginning, we take two digits at a time and calculated the accuracy by counting how many errors occurred in each prediction. Then we took 0,1, and 2 as an example to test how LDA did on classifying three digits at once. We tried LDA over 10 digits as well. For SVM and the decision tree classifiers, we directly used the built-in functions of MATLAB (see Appendix A. for detailed function explanation). We built the SVM with 10 digits classes respectively and individually examine each digit. The final prediction would be the digit with the highest score. The decision tree classified digits by finding the one with optimal hyper parameters.

IV. Computational Results

(a) SVD

Figure 1. shows the singular value spectrum of our SVD. We can see that the curve tend to turn horizontal when the singular value is around 50. Therefore, 50 should be satisfactory to be the number of modes, which is the rank of the digit space.

In our report, **U** builds our coordinate system for the digital space. **Σ** is the matrix containing our singular values. The columns of **V** gives us the coordinates of each image, classified by rows. **Figure 2**. is the 3D projection of V-modes in column 2, 3, and 5. We can see that some of them are clearly identifiable, while some are mixed with other modes, which would cause error in our classification.

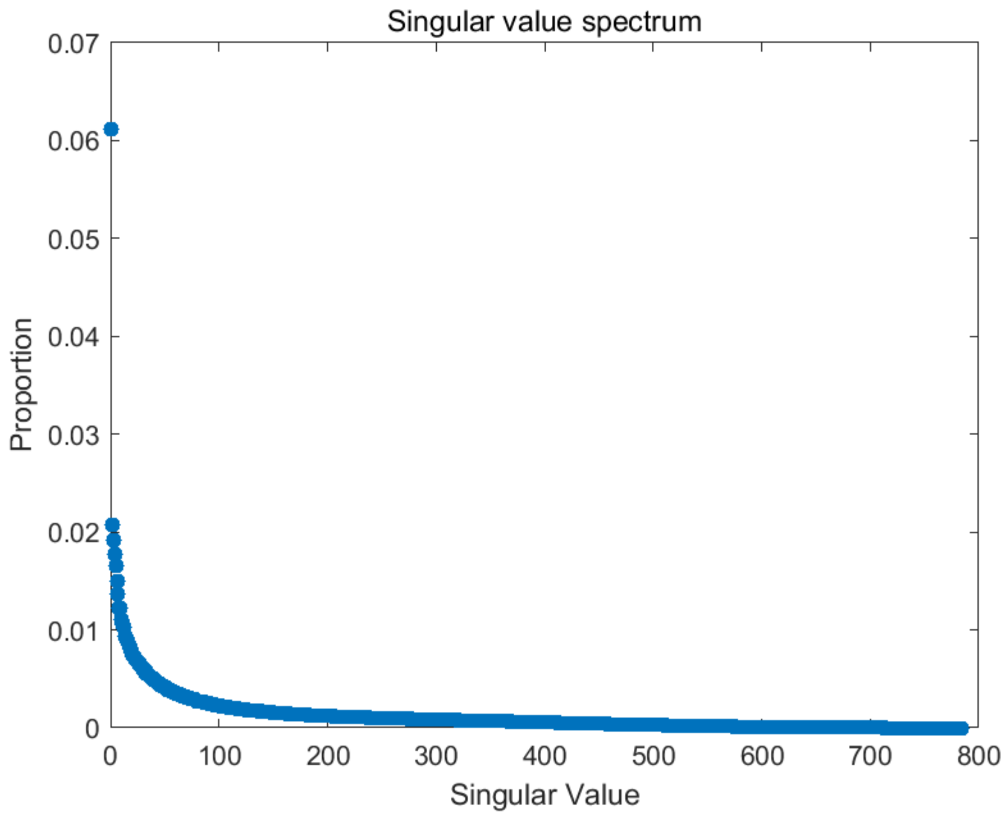

Figure 1. The singular value spectrum

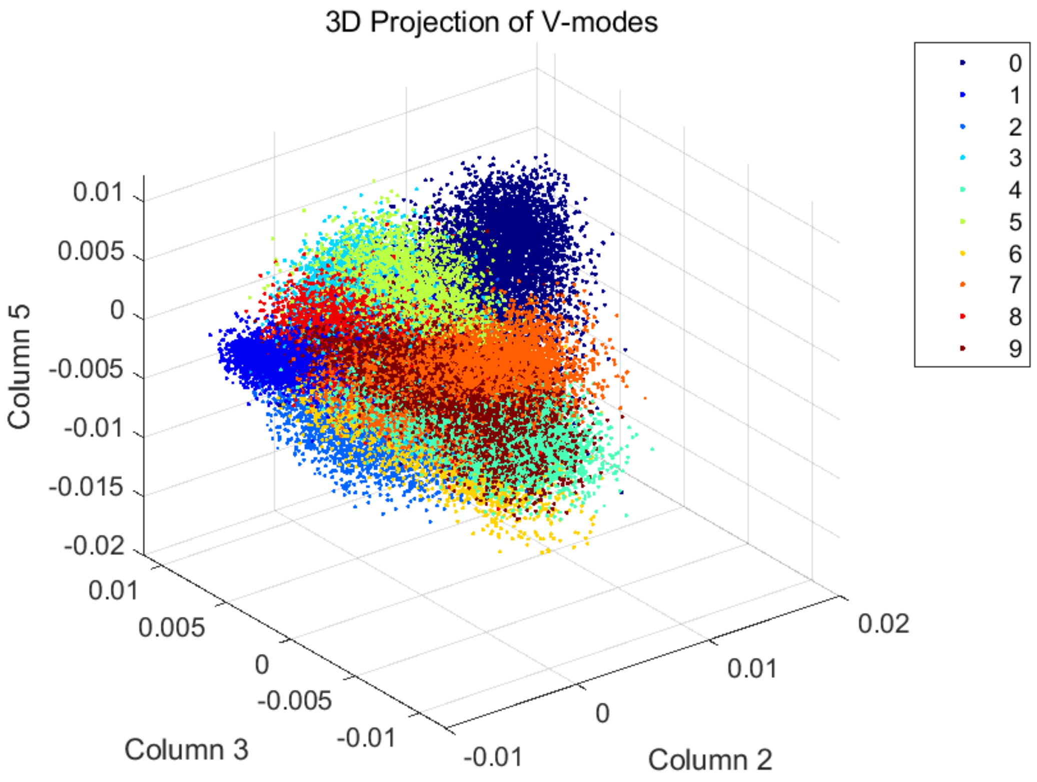

Figure 2. The 3D projection of V-modes in column 2, 3, and 5

(b) Linear Discriminant Analysis (LDA)

After projecting our data into PCA space, we tested our LDA on classifying two digits. **Table 1.** shows the accuracy of its performance. We can see that overall, LDA is good at classifying two digits with an accuracy above 95% for most digits. As highlighted in red in **Table 1.**, the pair of digit 0 and 1 is the easiest one to be classified by LDA with an accuracy of 99.81%, while the pair of digit 4 and 9 in blue is the hardest to be identified, which only has an accuracy of 81.61%.

Then we tried our LDA on classifying three digits. The 3 digits we chose are 0, 1, and 2. From **Table 1.** , if we just classify two of them at once, the accuracy must be above 97%. However, when we attempted to identify all three at once, the accuracy dropped to 94.79%. We also tried to use LDA to classify all digits at the same time, the performance was not ideal with a low accuracy of 75.59%.

Table 1. The accuracy of LDAs classifying two digits (Examine Training set)

	1	2	3	4	5	6	7	8	9
0	0.9981	0.9731	0.9793	0.9913	0.9497	0.9623	0.9895	0.9795	0.9813
1	-	0.9723	0.9827	0.9933	0.9901	0.9804	0.9736	0.9445	0.9864
2	-	-	0.9588	0.9712	0.9573	0.9266	0.9626	0.9327	0.9691
3	-	-	-	0.9864	0.8985	0.9817	0.9636	0.8986	0.9628
4	-	-	-	-	0.9706	0.9814	0.9636	0.9693	0.8161
5	-	-	-	-	-	0.9491	0.9723	0.8965	0.9547
6	-	-	-	-	-	-	0.9864	0.9730	0.9862
7	-	-	-	-	-	-	-	0.9400	0.9018
8	-	-	-	-	-	-	-	-	0.9319

Table 2. The accuracy of LDAs classifying two digits (Examine Test set)

	1	2	3	4	5	6	7	8	9
0	0.9947	0.9649	0.9652	0.9894	0.9435	0.9699	0.9818	0.9742	0.9802
1	-	0.9671	0.9682	0.9898	0.9808	0.9791	0.9798	0.9158	0.9830
2	-	-	0.9411	0.9622	0.9402	0.9156	0.9676	0.9312	0.9661
3	-	-	-	0.9758	0.8859	0.9778	0.9647	0.8913	0.9514
4	-	-	-	-	0.9649	0.9784	0.9585	0.9679	0.8141
5	-	-	-	-	-	0.9530	0.9624	0.8910	0.9485
6	-	-	-	-	-	-	0.9939	0.9704	0.9893
7	-	-	-	-	-	-	-	0.9513	0.8919
8	-	-	-	-	-	-	-	-	0.9264

Moreover, we trained our LDA with the test set and then evaluate the performance on classifying the train set. The accuracy are presented in **Table 2.** The overall accuracy dropped a little bit, but the 0-1 pair is still the easiest to separate (99.47%) while the 4-9 pair is the hardest (81.41%). When we classified 0,1, and 2 at once, the accuracy is 94.35%. The accuracy of LDA over all digits is 74.83%. It performs a little worse relatively when we trained with the test set.

(c) Support Vector Machines (SVM) and Decision Tree Classifiers

Using the built-in functions, we created SVM and decision tree classifiers and tested their performance on classifying 10 digits at once. Conclusively, SVM performs better with an accuracy of 93.32%, while the decision tree performs worse relatively with an accuracy of 84.05%. However, both of them are more accurate than LDA. After training using the test set, SVM gives an accuracy of 90.95% while the decision tree gives 77.74%. Again, both performs worse than the case where we examine on the training set. We can see that the decision tree is more negatively affected.

(d) Comparison on the Two Pairs

Table 3. The accuracy of three different classifiers (Examine Training set)

	LDA	SVM	Decision Tree
0-1 (easiest)	0.9981	0.9948	0.9972
4-9 (hardest)	0.8161	0.9387	0.8769

Table 4. The accuracy of three different classifiers (Examine Test set)

	LDA	SVM	Decision Tree
0-1 (easiest)	0.9947	0.9803	0.9874
4-9 (hardest)	0.8141	0.9107	0.8381

Finally, we applied SVM and decision tree classifiers on the 0-1 pair (the easiest pair to separate) and the 4-9 pair (the hardest pair to separate) respectively. **Table 3.** presents the accuracies of LDA, SVM, and decision tree. We can see that all three classifiers did well on identifying the 0-1 pair with an accuracy above 99%. As colored in red, LDA performs the best for the 0-1 pair with an accuracy of 99.81%, and SVM performs best on the 4-9 pair with an accuracy of 93.87%. For the poorest performance, as colored in blue, SVM is relatively weak on classifying 0 and 1 with an accuracy of 99.48%. When classifying the 4-9 pair, LDA is not a very suitable choice with a low accuracy of 81.61%.

Then we trained the classifiers with the test set and evaluated their performance. The result is in **Table 4.** Again, the overall accuracy is decreased, and SVM and the decision tree are affected more obviously than LDA. But their performance is similar to the case where we tested on the training set that LDA did best on the 0-1 pair (99.47%) and worst on the 4-9 pair (81.41%). SVM did best on the 4-9 pair (91.07%) and worst on the 0-1 pair (98.03%).

V. Summary and Conclusions

We did SVD on the MNIST data set and projected it into the PCA space. We have evaluated the performance of the three classifiers: LDA, SVM, and the decision tree, by training with the training set and examined the results with the test set first. We concluded that when classifying 10 digits, SVD performs the best while LDA is the worst. We also compared their accuracy on classifying the easiest pair to separate (0-1) and the hardest pair (4-9). We found out that the decision tree is moderate in both cases. LDA is the best option for classifying the 0-1 pair and SVM is the best choice for the 4-9 pair. We also did the opposite case in which we trained with the test set and examine the accuracy on the training set. The overall accuracy dropped a little bit, but the conclusion is not significantly influenced.

Appendix A. MATLAB functions[1]

`[images,labels] = mnist_parse('images','labels')` : loads data from the file. We used it to load our data.

`B = reshape(A,sz)` : reshapes `A` using the size vector, `sz`, to define `size(B)`. We used it to reshape each image into a column vector and each column of the data matrix is a different image.

`[u,s,v] = svd(A,'econ')`: Produces an economy-size decomposition of matrix `A`, such that `A = U*S*V'`. We used it to perform SVD.

`D = diag(v)` : returns a square diagonal matrix with the elements of vector `v` on the main diagonal. We used it to obtain the diagonal matrix to plot the singular value spectrum.

`k = find(X)` : returns the index `k` that satisfies the condition `X`. We used it to find each label for the 3D projection of V-modes.

`scatter3(X,Y,Z)`: displays circles at the locations specified by vectors `X,Y,Z`. We used it to draw the 3D projection of V-modes.

`class = classify(sample,training,group)`: classifies each row of the data in `sample` into one of the groups in `training`. We used it to build our LDAs.

`Mdl = fitcsvm(X,Y)`: returns an SVM classifier trained using the predictors in the matrix `X` and the class labels in vector `Y` for one-class or two-class classification. We used it to implement the SVM classifier.

`tree = fitctree(X,Y)` : returns a fitted binary classification decision tree based on the input variables contained in matrix `X` and output `Y`. We used it to implement the decision tree classifier.

`ypred = predict(Mdl,Xnew)` : returns the predicted response values of the linear regression model `Mdl` to the points in `Xnew`. We used it to make predictions for SVM and decision trees.

`[ypred,yci] = predict(Mdl,Xnew)`: also returns confidence intervals for the responses at `Xnew`. We used it to find the probability for a number coming from a certain class.

[1] Basic function descriptions from MathWorks Help Center.

Appendix B. MATLAB codes

```
clear all; close all; clc;
[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');

%% SVD
data_train = reshape(images_train, [784 60000]);
data_test = reshape(images_test, [784 10000]);

[u,s,v] = svd(double(data_train),'econ');
train = (s*v')';
test = (double(u)'*double(data_test))';
lambda = diag(s);

%% Singular value spectrum
figure()
plot(lambda/sum(lambda),'.','Markersize',20)
xlabel('Singular Value'),ylabel('Proportion')
title('Singular value spectrum')
saveas(gcf,'spectrum.png')

%% V-modes
figure()
colormap jet
for i = 0:9
    ind = find(labels_train == i);
    scatter3(v(ind,2),v(ind,3),v(ind,5),20,labels_train(ind),'.')
    hold on
end
xlabel('Column 2')
ylabel('Column 3')
zlabel('Column 5')
legend({'0','1','2','3','4','5','6','7','8','9'});
title('3D Projection of V-modes')
saveas(gcf,'vmode.png')

%% LDA for two digits
LDA_acc = zeros(10,10);
```

```
for i = 0:8
    for j = i+1:9
        x1_train = train(labels_train == i,2:10);
        x2_train = train(labels_train == j,2:10);
        xtrain = [x1_train; x2_train];
        x1_len = size(x1_train,1);
        x2_len = size(x2_train,1);
        numtrain = [i*ones(x1_len,1); j*ones(x2_len,1)];

        x1_test = test(labels_test == i,2:10);
        x2_test = test(labels_test == j,2:10);
        xtest = [x1_test; x2_test];
        x1_len = size(x1_test,1);
        x2_len = size(x2_test,1);
        numtest = [i*ones(x1_len,1); j*ones(x2_len,1)];
        prediction = classify(xtest,xtrain,numtrain);
        error = sum(numtest-prediction ~= 0);
        LDA_acc(i+1,j+1) = 1-error/length(numtest);
    end
end

%% LDA for 0,1,2
x1_train = train(labels_train == 0,2:10);
x2_train = train(labels_train == 1,2:10);
x3_train = train(labels_train == 2,2:10);
xtrain = [x1_train; x2_train; x3_train];
x1_len = size(x1_train,1);
x2_len = size(x2_train,1);
x3_len = size(x3_train,1);
numtrain = [0*ones(x1_len,1); 1*ones(x2_len,1);
2*ones(x3_len,1)];

x1_test = test(labels_test == 0,2:10);
x2_test = test(labels_test == 1,2:10);
x3_test = test(labels_test == 2,2:10);
xtest = [x1_test; x2_test; x3_test];
x1_len = size(x1_test,1);
x2_len = size(x2_test,1);
x3_len = size(x3_test,1);
numtest = [0*ones(x1_len,1); 1*ones(x2_len,1);
2*ones(x3_len,1)];
```

```
prediction = classify(xtest,xtrain,numtrain);
error = sum(numtest-prediction ~= 0);
acc = 1-error/length(numtest);

%% LDA for all
xtrain = train(:,2:10);
xtest = test(:,2:10);
prediction = classify(xtest,xtrain,labels_train);
error = sum(labels_test-prediction ~= 0);
LDA_acc_all = 1-error/length(labels_test);

%% SVM for all
xtrain = train(:,2:10)/max(max(s));
xtest = test(:,2:10)/max(max(s));

SVMModel = cell(10,1);
classes = 0:9;
rng(1);
for i = 1:numel(classes)
    ind = labels_train == classes(i);
    SVMModel{i} = fitcsvm(xtrain,ind,'ClassNames',[false
true],'Standardize',true,...
        'KernelFunction','rbf','BoxConstraint',1);
end
for i = 1:numel(classes)
    [~,score] = predict(SVMModel{i},xtest);
    Pos_score(:,i) = score(:,2);
end

[~,max] = max(Pos_score,[],2);
error = sum(labels_test+1-max ~= 0);
SVM_acc = 1-error/length(labels_test);

%% Decision Tree for all
xtrain = train(:,2:10);
xtest = test(:,2:10);
Mdl =
fitctree(xtrain,labels_train,'OptimizeHyperparameters','auto');
prediction = predict(Mdl,xtest);
error = sum(labels_test-prediction ~= 0);
DT_acc = 1-error/length(labels_test);
```

```
%% Compare performace on pairs
% easiest (0,1)
x1_train = train(labels_train == 0,2:10);
x2_train = train(labels_train == 1,2:10);
x1_len = size(x1_train,1);
x2_len = size(x2_train,1);
xtrain = [x1_train; x2_train];
numtrain = [0*ones(x1_len,1); 1*ones(x2_len,1)];

x1_test = test(labels_test == 0,2:10);
x2_test = test(labels_test == 1,2:10);
xtest = [x1_test; x2_test];
x1_len = size(x1_test,1);
x2_len = size(x2_test,1);
numtest = [0*ones(x1_len,1); 1*ones(x2_len,1)];

% SVM
rng default
Mdl_SVM = fitcsvm(xtrain,numtrain,'Standardize',true,...
       'KernelFunction','rbf','BoxConstraint',1);
prediction = predict(Mdl_SVM,xtest);
error = sum(numtest-prediction ~= 0);
SVM_acc_e = 1-error/length(numtest);

% Decision tree
Mdl_DT =
fitctree(xtrain,numtrain,'OptimizeHyperparameters','auto');
prediction = predict(Mdl_DT,xtest);
error = sum(numtest-prediction ~= 0);
DT_acc_e = 1-error/length(numtest);

% hardest (4,9)
x1_train = train(labels_train == 4,2:10);
x2_train = train(labels_train == 9,2:10);
xtrain = [x1_train; x2_train];
x1_len = size(x1_train,1);
x2_len = size(x2_train,1);
numtrain = [4*ones(x1_len,1); 9*ones(x2_len,1)];

x1_test = test(labels_test == 4,2:10);
```

```
x2_test = test(labels_test == 9,2:10);
xtest = [x1_test; x2_test];
x1_len = size(x1_test,1);
x2_len = size(x2_test,1);
numtest = [4*ones(x1_len,1); 9*ones(x2_len,1)];

% SVM
rng default
Mdl_SVM = fitcsvm(xtrain,numtrain,'Standardize',true,...
       'KernelFunction','rbf','BoxConstraint',1);
prediction = predict(Mdl_SVM,xtest);
error = sum(numtest-prediction ~= 0);
SVM_acc_h = 1-error/length(numtest);

% Decision tree
Mdl_DT =
fitctree(xtrain,numtrain,'OptimizeHyperparameters','auto');
prediction = predict(Mdl_DT,xtest);
error = sum(numtest-prediction ~= 0);
DT_acc_h = 1-error/length(numtest);
```