Classifying Digits in the MNIST Database

Yichen Shen

March 8, 2021

Abstract

In this paper is going to explore the practical usefulness of the Linear Discriminant Analysis by recognizing the digits from the MNIST Database using randomly selected digits. We are going to apply singular value decomposition to our data matrix in order to apply the principle component analysis, then essentially we are going to utilize the LDA analysis in teaching the machine to recognizing the digits and test its result. Finally, are going to compare the LDA with support vector machines (SVM) and decision trees to see how it perform in contrast to different algorithm.

1 Introduction and Overview

1.1 Background and Context

The MNIST database is a collective of handwritten digits with a training set of 60,000 examples and and test set of 10,000 examples, and is a subset of the larger number set NIST. All digits in the database has been size-normalized and centered in fixed size. In this paper, we are going to utilize LDA analysis along with the other two comparison analysis method, to learn the techniques and pattern recognition method in the big database using randomly selected digits.

1.2 Topic Importance

Data driven machine learning has become more and more prevalent in the real world as people are trying to understand it deeper. However, it is truly a black box for human to understand because the models are created directly from the data by the algorithm. The hidden combination of variables making the programmer incomprehensible in the decision making. However, we can utilize this black box along with the algorithm that we developed and understand to ask the computer to give us a prediction based on huge databases. This effective data process machine combine with the proper tools can help us processing massive of data and approach to our demand.

1.3 Objectives of Research

In this paper, we are going to first perform the singular value decomposition analysis and the principle component analysis in order for us to better spot our data. Then we are going to project our data onto the PCA space and build classifier to recognize digits in the training and testing sets. We are going to build a linear classifier for both two digits and three digits comparison, and then we are going to quantify the accuracy between the digits. Lastly, we are going to compare LDA to SVM and decision trees on the hardest and easiest pair of digits and separate them accordingly.

2 Theoretical Background

2.1 Linear Discriminant Analysis (LDA)

As two data sets are project onto a new bases, we want to completely separate them so that our data sets can be easily classified in different subspace. In order to do so, we want the mean of two data sets to be

apart, which is the goal of linear discriminant analysis. LDA is utilized to find the best projection which can maximize the inter-class distance and minimize the intra-class distance.

2.1.1 Implementing LDA for 2 Datasets

Given two data sets, in order to apply linear discriminant analysis, we first need to calculate the mean for each group and each feature. Note that the means are column vectors. We can define the intra class distance variance as a `between-class scatter matrix` as following:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T. \tag{1}$$

For the measurement of inter-class distance variance as a `within-class scatter matrix` as following:

$$S_\omega = \sum_{j=1}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T. \tag{2}$$

Now that we find the variance within the group, we can find the corresponding vector such that

$$w = argmax \frac{w^T S_B w}{w^T S_\omega w} \tag{3}$$

In this equation, the vector `w` find the max value of the above, is the eigenvector that matches the largest eigenvalue of the generalized eigenvalue problem which has form:

$$S_B w = \lambda S_\omega w. \tag{4}$$

Once we have this `w` after compute the operation, we will need to find a threshold value for the machine to make its decision.

2.1.2 LDA for more Datasets

In the real world we might be dealing with some datasets that is more than 2, therefore, we need to develop from the above and find a LDA operation for more data sets. In order to do so, we first make changes to the `between-class scatter matrix`:

$$S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T \tag{5}$$

within which the μ is the general mean and μ_j is the mean of each set for a N value greater or equal to 3. The `within-class scatter matrix` can be define as:

$$S_\omega = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{6}$$

after compute these two matrix, we can find the corresponding `w` vector.

3 Algorithm Implementation and Development

To perform the analysis of the data, we first need to perform singular value decomposition onto the digit images. By using the `mnist-parse.m` function, we can load the images and labels of the MNIST dataset for both training and testing. From then on, we are going to reshape the image and turns it into double digit using `im2double and reshape()` command, then we convert it into the column vector that each corresponding to a different image, also we want to subtract the mean from the matrix so that the black background would not be affecting the singular value. After decompose our matrix into the SVD components, we can perform our PCA analysis on our data set. The interpretation of `S, V, U` matrices is that: S contains the singular value on the diagonal with the information indicating the each principle components, the row in matrix V

contains information that represents each of the image, lastly, U contains the principle components of our of each column in the data set.

We then perform our algorithm to find the singular value spectrum of the training data in order to better assist us at choosing the proper value in the low rank of approximation for our feature that is the number of principle component that we are using in our data. Following that we choose to plot 3 columns: 2,3,5 of V-modes on the 3D plot and see their allocation in space.

3.1 Building LDA for 2 and 3 digits

With our feature value given by above algorithm, we first project the featured matrix of test and training data onto the PCA space by using U'*X(= S*V'). For the testing data we are using the principle component matrix from the training data since it contains the principle component information that is used to set the feature. We then store the testing and training matrix with distinctive digits in two data cell for further using. Then we are moving to calculate the scatter matrix. (We use the training data to find the projection line for our LDA analysis for projection.)

Algorithm 1: Building LDA for 2 and 3 digits

Find the mean of the two digit vectors then we calculate the `Sb and Sw` accordingly
Use `eig()` command to find the linear discriminant analysis components projection line `w`
Project our results onto `w` by multiplying `w'`
Set a fix threshold for the classifier for the two digit and use this trained threshold for further test analysis. However, for 3 digit, we move to set two threshold in between each pair of the data For conditional testing

3.1.1 Success Rate

After calculating our threshold, we want to see how successful we are in identifying the digits in both our testing and training data. Therefore, we take a look at the successful rate of our testing results.

Algorithm 2: Calculating Success Rate

We send our digit vector to compare to the threshold in a conditional testing, if the digit is within its region of threshold, it returns 0s, otherwise, it returns 1s.
By locating those 1s and find the total value of errors, we can compare this value to our total data number. The subtraction from 1 of this value is our success rate.
As for three digits, we are setting two threshold value to send each digit into one of the three region. By comparing their individual index in the vector, we are able to find the success rate.

4 Computational Results

4.1 SVD analysis and PCA

Based on the singular value spectrum in Figure 1 that we got, we can see that instead of a few greate value points, there are a lot of singular value that are above 300 and they all capture a great amount of information, consecutively, the first 60 singular values are having a value of above 100. In order to better have an idea of how much modes we need to use in order to process our classification work, we can take a look at the the low rank approximation result. In Figure 2 we can see that in a randomly selected image, with a rank of 25 approximation, we can recognize that it is a digit 3. Thus, we are using 25 as our feature in order to classified the other randomly selected digits.

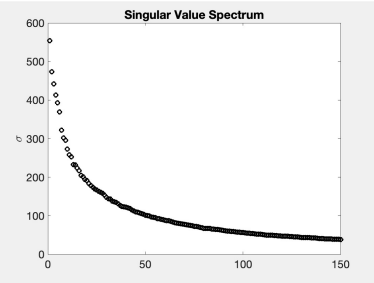

Figure 1: The singular value spectrum

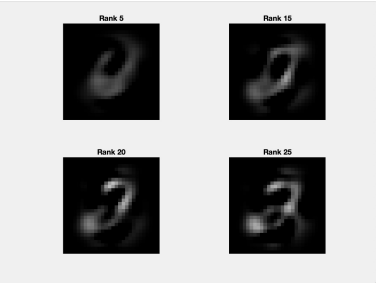

Figure 2: The plow rank approximation of selected image

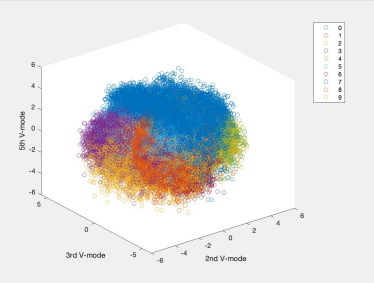

Figure 3: The selected V-modes project onto a 3D plot

4.1.1 3D Plot Project onto Selected V-modes

Based on Figure 3, the 3D projection onto three selected V-modes shows that these digits are project onto the 3D plot in a way that maximize their distance of mean value in between.

4.2 LDA Performance

4.2.1 2 Digits

We pick in all possible randomly picked pair of 2 digits into our algorithm 1 and 2, then we calculate the accuracy successful rate for each of the digit pairs. With a higher accuracy pair, they are easier to separate, conversely, for a low pair of accuracy value, the two digits are hard to separate since they all have several miss classified digits. Based on our trial, when we pick [6,9], our successful rate is the highest, which in the training data is [6: 99.44%, 9:99.50%], and in our testing data, it is [6: 99.48%, 9: 99.50%]. Therefore, this pair is the easiest to separate. Now for the most difficult to separate digit, we find [3,5]. In the training data is [3: 95.60%, 5:90.11%], and in our testing data, it is [3: 97.33%, 5: 89.57%]. Therefore, this pair is the hardest to separate.

4.2.2 3 Digits

In the 3 digits scenario, since there are more digits, it is harder for the digits to find their best position on the projection as appose to only two digits (one on each side). Hence, the successful rate in this case will be lower that that in the two digits case. To testify this idea, we randomly pick three digits [4,8,9], the successful rate for this set in the training data is: [4: 82.95%, 8: 84.79%, 9:88.66%], and in the testing data it is: [4: 81.89%, 8: 85.73%, 9: 89.32%].

4.3 Comparison with other classifiers

We again calculate the result for the difficult pair [3,5]. Based on our result from SVM, we get a 0.0058 error, and for tree, we got 0.0076 for the error. For the all 10 digits, we got an error for SVM with value of 0.0367, and for the tree classification we got a 0.0956. Based on this result, we can conclude that the SVM and the tree classification both perform better than the LDA classifier.

5 Summary and Conclusions

This research paper studies on the topic of utilizing LDA in MNIST digits recognition and its comparison with SVM classifier and classification tree. From our observation, we can see that the LDA method is a powerful tool to use in the digit recognition. However, as the number of digits increases, the accuracy of LDA decrease as their allocation on the projection line changes with increment of digits. Then when we

compare to the SVM and the tree classification methods, we can see that the LDA in the particular digit performance is less effective than the other two. However, the tree classification can have some over-fitting issue while the SVM can perform slightly worse in some cases. No matter how, it is important for us to understand the theoretical background and the algorithm of LDA in order to process the data.

Appendix A MATLAB Functions

Important MATLAB functions with a brief implementation explanation.

- `im2double` convrets the image to a double precision.
- `[U,S,V] = svd(A,'econ')` produces an economy sized singular value decomposition of a m by n matrix A.
- `[M,I] = max(A)` returns the maximum elements of an array A.
- `diag(A)` returns a column vector of the diagonal elements of matrix A.
- `reshape(A,sz)` reshape matrix A using size vector sz.
- `sort(A)` sort the element of A in ascending order
- `histogram(A)` plot a histogram of A.
- `cell` creates a cell arrays.
- `fitctree(A,B)` returns a fitted binary classification tree from the input of variables in matrix A and output B.
- `fitcsvm(A,B)` returns an SVM classifier trained using the predictors in matrix A and labels in matrix B.
- `fitcecoc(A,B)` returns a trained ECOC model using the predictors A and labels B.

Appendix B MATLAB Code

The following code is to perform the algorithm 1 2 and 3 on case 1. The code for the other cases are pretty similar to each other with only some modify on the area cover part in algorithm 1.

```
close all, clc

% Load MNIST data
[images, labels] = mnist_parse('train-images-idx3-ubyte','train-labels-idx1-ubyte');
[images_t, labels_t] = mnist_parse('t10k-images-idx3-ubyte','t10k-labels-idx1-ubyte');

% Perform analysis of data set
% SVD
vec = im2double(reshape(images,28*28,60000));
[m,n] = size(vec);
mn = mean(vec,2);
vec = vec - repmat(mn,1,n);
[U,S,V] = svd(vec,'econ');

vec_t = im2double(reshape(images_t,28*28,10000));
vec_t = vec_t - repmat(mn,1,10000);
[Ut,St,Vt] = svd(vec_t,'econ');

% Plot singular value spectrum
plot(diag(S),'ko','Linewidth',2)
set(gca,'Fontsize',14,'Xlim',[0,150])
ylabel('\sigma')
title('Singular Value Spectrum')

% Find the low-rank approximation
rank_5 = U(:,1:5)*S(1:5,1:5)*V(:,1:5)';
rank_15 = U(:,1:15)*S(1:15,1:15)*V(:,1:15)';
rank_20 = U(:,1:20)*S(1:20,1:20)*V(:,1:20)';
rank_25 = U(:,1:25)*S(1:25,1:25)*V(:,1:25)';

subplot(2,2,1)
plot_5 = reshape(rank_5(:,6),28,28);
imshow(plot_5)
title('Rank 5')
subplot(2,2,2)
plot_15 = reshape(rank_15(:,6),28,28);
imshow(plot_15)
title('Rank 15')
subplot(2,2,3)
plot_20 = reshape(rank_20(:,6),28,28);
imshow(plot_20)
title('Rank 20')
subplot(2,2,4)
plot_25 = reshape(rank_25(:,6),28,28);
imshow(plot_25)
title('Rank 25')

% Project three selected V-modes onto 3D
v_mode = U(:,[2,3,5])'*vec;

figure(2)
for j = 0:9
    loca = v_mode(:,find(labels == j));
    plot3(loca(1,:),loca(2,:),loca(3,:),'o'); hold on
end
%plot3(v_mode(1,:),v_mode(2,:),v_mode(3,:)); hold on
xlabel('2nd V-mode')
ylabel('3rd V-mode')
zlabel('5th V-mode')
legend('0','1','2','3','4','5','6','7','8','9')
```

```
%% Build a LDA for 2 digits
dig1 = 7;
dig2 = 9;

feature = 25;
num = U(:,1:feature)'*vec;
num_t = U(:,1:feature)'*vec_t;

digit0 = num(:,find(labels == 0));
digit1 = num(:,find(labels == 1));
digit2 = num(:,find(labels == 2));
digit3 = num(:,find(labels == 3));
digit4 = num(:,find(labels == 4));
digit5 = num(:,find(labels == 5));
digit6 = num(:,find(labels == 6));
digit7 = num(:,find(labels == 7));
digit8 = num(:,find(labels == 8));
digit9 = num(:,find(labels == 9));

digit_t0 = num_t(:,find(labels_t == 0));
digit_t1 = num_t(:,find(labels_t == 1));
digit_t2 = num_t(:,find(labels_t == 2));
digit_t3 = num_t(:,find(labels_t == 3));
digit_t4 = num_t(:,find(labels_t == 4));
digit_t5 = num_t(:,find(labels_t == 5));
digit_t6 = num_t(:,find(labels_t == 6));
digit_t7 = num_t(:,find(labels_t == 7));
digit_t8 = num_t(:,find(labels_t == 8));
digit_t9 = num_t(:,find(labels_t == 9));

%put into a cell array to store our data
cell = {digit0,digit1,digit2,digit3,digit4,digit5,digit6,digit7,digit8,digit9};
cell_t = {digit_t0,digit_t1,digit_t2,digit_t3,digit_t4,digit_t5,digit_t6, ...
    digit_t7,digit_t8,digit_t9};

train_dig1 = cell{dig1+1};
train_dig2 = cell{dig2+1};

test_dig1 = cell_t{dig1+1};
test_dig2 = cell_t{dig2+1};

size1 = size(train_dig1,2);
size2 = size(train_dig2,2);

% LDA calculation
%scatter matrix
m1 = mean(train_dig1,2);
m2 = mean(train_dig2,2);

Sw = 0; % within class
for k = 1:size1
    Sw = Sw + (train_dig1(:,k)-m1)*(train_dig1(:,k)-m1)';
end

for k = 1:size2
    Sw = Sw + (train_dig2(:,k)-m2)*(train_dig2(:,k)-m2)';
end
```


Listing 2: Code to perform algorithms 123

```
Sb = (m1-m2)*(m1-m2)'; % between class

[V2,D] = eig(Sb,Sw);
[lambda,ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

%project onto w
vdig1 = w'*train_dig1;
vdig2 = w'*train_dig2;
vdig_t1 = w'*test_dig1;
vdig_t2 = w'*test_dig2;

% find thershold for classifier in train and test
if mean(vdig1) > mean(vdig2)
    w = -w;
    vdig1 = -vdig1;
    vdig2 = -vdig2;
    vdig_t1 = -vdig_t1;
    vdig_t2 = -vdig_t2;
end

% set thershold value
sort1 = sort(vdig1);
sort2 = sort(vdig2);
sort3 = sort(vdig_t1);
sort4 = sort(vdig_t2);

t1 = length(sort1);
t2 = 1;
while sort1(t1) > sort(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sort1(t1) + sort2(t2))/2;

% plot histogram for two digit value in test
% figure(3)
% subplot(1,2,1)
% histogram(sort3,30); hold on, plot([threshold threshold],[0,800],'r')
% set(gca, 'Xlim',[-4 4],'Ylim',[0 800],'Fontsize',14)
% title('Selected digit 7')
% subplot(1,2,2)
% histogram(sort4,30); hold on, plot([threshold threshold], [0,800],'r')
% set(gca, 'Xlim',[-4 4],'Ylim',[0 800],'Fontsize',14)
% title('Selected digit 9')

% calculate the accuracy for test and train
%train
ResVec1 = (vdig1 > threshold);
err1 = ResVec1(:,find(ResVec1 == 1));
errNum1 = size(err1);
testNum1 = size(vdig1);
sucRate1 = 1-(errNum1/testNum1);
```

Listing 3: Code to perform algorithms 123

```
ResVec2 = (vdig2 < threshold);
err2 = ResVec2(:,find(ResVec2 == 1));
errNum2 = size(err2);
testNum2 = size(vdig2);
sucRate2 = 1-(errNum2/testNum2);

ResVec3 = (vdig_t1 > threshold);
err3 = ResVec3(:,find(ResVec3 == 1));
errNum3 = size(err3);
testNum3 = size(vdig_t1);
sucRate3 = 1-(errNum3/testNum3);
%test
ResVec4 = (vdig_t2 < threshold);
err4 = ResVec4(:,find(ResVec4 == 1));
errNum4 = size(err4);
testNum4 = size(vdig_t2);
sucRate4 = 1-(errNum4/testNum4);

% SVM classifier

normal1 = train_dig1./max(train_dig1(:));
normal2 = train_dig2./max(train_dig2(:));
normal_t = num_t/max(num_t(:));
dig_lab1 = dig1.*ones(length(sort1),1);
dig_lab2 = dig2.*ones(length(sort2),1);
xtrain = [normal1 normal2];
xtrain = xtrain';
label = [dig_lab1; dig_lab2];
Mdl = fitcecoc(xtrain,label);
accu = resubLoss(Mdl);
normal_t = normal_t';
test_labels = predict(Mdl,normal_t);
CVMdl = crossval(Mdl);
genError = kfoldLoss(CVMdl);

% classification tree

tree = fitctree(xtrain,label);
accu2 = predict(tree,normal_t);

xtrain = U(:,feature)'*vec;
xtrain = xtrain / max(xtrain(:));
xtrain = xtrain';
xlabel = labels;
Mdl = fitcecoc(xtrain,xlabel);
error = resubLoss(Mdl);
```

Listing 4: Code to perform algorithms 123

```
% Load MNIST data
[images, labels] = mnist_parse('train-images-idx3-ubyte','train-labels-idx1-ubyte');
[images_t, labels_t] = mnist_parse('t10k-images-idx3-ubyte','t10k-labels-idx1-ubyte');

% Perform analysis of data set
% SVD
vec = im2double(reshape(images,28*28,60000));
[m,n] = size(vec);
mn = mean(vec,2);
vec = vec - repmat(mn,1,n);
[U,S,V] = svd(vec,'econ');

vec_t = im2double(reshape(images_t,28*28,10000));
vec_t = vec_t - repmat(mn,1,10000);
[Ut,St,Vt] = svd(vec_t,'econ');

% Plot singular value spectrum
plot(diag(S),'ko','Linewidth',2)
set(gca,'Fontsize',14,'Xlim',[0,150])
ylabel('\sigma')
title('Singular Value Spectrum')

% Find the low-rank approximation
rank_5 = U(:,1:5)*S(1:5,1:5)*V(:,1:5)';
rank_15 = U(:,1:15)*S(1:15,1:15)*V(:,1:15)';
rank_20 = U(:,1:20)*S(1:20,1:20)*V(:,1:20)';
rank_25 = U(:,1:25)*S(1:25,1:25)*V(:,1:25)';

subplot(2,2,1)
plot_5 = reshape(rank_5(:,6),28,28);
imshow(plot_5)
title('Rank 5')
subplot(2,2,2)
plot_15 = reshape(rank_15(:,6),28,28);
imshow(plot_15)
title('Rank 15')
subplot(2,2,3)
plot_20 = reshape(rank_20(:,6),28,28);
imshow(plot_20)
title('Rank 20')
subplot(2,2,4)
plot_25 = reshape(rank_25(:,6),28,28);
imshow(plot_25)
title('Rank 25')

% Project three selected V-modes onto 3D
v_mode = U(:,[2,3,5])'*vec;

figure(2)
for j = 0:9
    loca = v_mode(:,find(labels == j));
    plot3(loca(1,:),loca(2,:),loca(3,:),'o'); hold on
end
%plot3(v_mode(1,:),v_mode(2,:),v_mode(3,:)); hold on
xlabel('2nd V-mode')
ylabel('3rd V-mode')
zlabel('5th V-mode')
legend('0','1','2','3','4','5','6','7','8','9')
```

Listing 5: Code to perform algorithms 123

```
%% Build a LDA for 2 digits
dig1 = 7;
dig2 = 9;
dig3 = 4;

feature = 25;
num = U(:,1:feature)'*vec;
num_t = U(:,1:feature)'*vec_t;
digit0 = num(:,find(labels == 0));
digit1 = num(:,find(labels == 1));
digit2 = num(:,find(labels == 2));
digit3 = num(:,find(labels == 3));
digit4 = num(:,find(labels == 4));
digit5 = num(:,find(labels == 5));
digit6 = num(:,find(labels == 6));
digit7 = num(:,find(labels == 7));
digit8 = num(:,find(labels == 8));
digit9 = num(:,find(labels == 9));

digit_t0 = num_t(:,find(labels_t == 0));
digit_t1 = num_t(:,find(labels_t == 1));
digit_t2 = num_t(:,find(labels_t == 2));
digit_t3 = num_t(:,find(labels_t == 3));
digit_t4 = num_t(:,find(labels_t == 4));
digit_t5 = num_t(:,find(labels_t == 5));
digit_t6 = num_t(:,find(labels_t == 6));
digit_t7 = num_t(:,find(labels_t == 7));
digit_t8 = num_t(:,find(labels_t == 8));
digit_t9 = num_t(:,find(labels_t == 9));

%put into a cell array to store our data
cell = {digit0,digit1,digit2,digit3,digit4,digit5,digit6,digit7,digit8,digit9};
cell_t = {digit_t0,digit_t1,digit_t2,digit_t3,digit_t4,digit_t5,digit_t6, ...
    digit_t7,digit_t8,digit_t9};
train_dig1 = cell{dig1+1};
train_dig2 = cell{dig2+1};
train_dig3 = cell{dig3+1};
test_dig1 = cell_t{dig1+1};
test_dig2 = cell_t{dig2+1};
test_dig3 = cell_t{dig3+1};

size1 = size(train_dig1,2);
size2 = size(train_dig2,2);
size3 = size(train_dig3,2);

% LDA calculation
%scatter matrix
m1 = mean(train_dig1,2);
m2 = mean(train_dig2,2);
m3 = mean(train_dig3,2);
Sw = 0; % within class
for k = 1:size1
    Sw = Sw + (train_dig1(:,k)-m1)*(train_dig1(:,k)-m1)';
end
for k = 1:size2
    Sw = Sw + (train_dig2(:,k)-m2)*(train_dig2(:,k)-m2)';
end
for k = 1:size3
    Sw = Sw + (train_dig3(:,k)-m3)*(train_dig3(:,k)-m3)';
end
```

```
m = (m1+m2+m3)/3;
Sb = (m1-m)*(m1-m)'+(m2-m)*(m2-m)'+(m3-m)*(m3-m)'; % between class

[V2,D] = eig(Sb,Sw);
[lambda,ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

%project onto w
vdig1 = w'*train_dig1;
vdig2 = w'*train_dig2;
vdig3 = w'*train_dig3;
vdig_t1 = w'*test_dig1;
vdig_t2 = w'*test_dig2;
vdig_t3 = w'*test_dig3;

% set thershold value
sort1 = sort(vdig1);
sort2 = sort(vdig2);
sort3 = sort(vdig3);

t1 = length(sort1);
t2 = 1;
while sort1(t1) > sort(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold12 = (sort1(t1) + sort2(t2))/2;

t2 = length(sort2);
t3 = 1;
while sort1(t2) > sort(t3)
    t2 = t2 - 1;
    t3 = t3 + 1;
end
threshold23 = (sort1(t2) + sort2(t3))/2;

% calculate the accuracy for test and train
%train
ResVec1 = (vdig1 > threshold12);
err1 = ResVec1(:,find(ResVec1 == 1));
errNum1 = size(err1);
testNum1 = size(vdig1);
sucRate1 = 1-(errNum1/testNum1);

ResVec2 = (vdig2 < threshold12 & vdig2 > threshold23);
err2 = ResVec2(:,find(ResVec2 == 1));
errNum2 = size(err2);
testNum2 = size(vdig2);
sucRate2 = 1-(errNum2/testNum2);

ResVec3 = (vdig3 < threshold23);
err3 = ResVec3(:,find(ResVec3 == 1));
errNum3 = size(err3);
testNum3 = size(vdig3);
sucRate3 = 1-(errNum3/testNum3);
```


Listing 7: Code to perform algorithms 123

```
%test
ResVec4 = (vdig_t1 > threshold12);
err4 = ResVec4(:,find(ResVec4 == 1));
errNum4 = size(err4);
testNum4 = size(vdig_t1);
sucRate4 = 1-(errNum4/testNum4);

ResVec5 = (vdig_t2 < threshold12 & vdig_t2 > threshold23);
err5 = ResVec5(:,find(ResVec5 == 1));
errNum5 = size(err5);
testNum5 = size(vdig_t2);
sucRate5 = 1-(errNum5/testNum5);

ResVec6 = (vdig_t3 < threshold23);
err6 = ResVec6(:,find(ResVec6 == 1));
errNum6 = size(err6);
testNum6 = size(vdig_t3);
sucRate6 = 1-(errNum6/testNum6);
```

Listing 8: Code to perform algorithms 123