Basic Machine Learning

Christine M. Baker, GitHub: `cmbaker94`

March 10, 2021

Abstract

Machine learning techniques are useful tools to classify images, such as handwritten numbers. Linear discrimination analysis and other state-of-the-art machine learning algorithms were applied to the images to classify two or more different values. Single value decomposition was used to reduce the dimensions of the data prior to the analysis. The machine learning algorithms are compared for the hardest and easiest number.

1 Introduction and Overview

Machine learning is applied to handwritten numbers to test the classification process of linear discrimination analysis (LDA) and other state-of-the-art machine learning algorithms. Singular value decomposition is applied to the images prior to the machine learning analysis to reduce the dimensions of the data. This method is a way to create the most accurate classification with a small set of observations. LDA is applied to classify two and three numbers for the test and training set. The hardest and easiest numbers to classify were identified. A support-vector machine and decision tree was applied to the dataset and the classification process was compared to LDA.

The theoretical background on LDA and basic machine learning algorithms are provided in Section 2. The algorithm implementation and development to resolve the principle components are presented in Section 3. The computational results for the four test cases are shown in Section 4. A conclusions are summarized in Section 5. Details about the MATLAB functions and the function written for this project are in Appendix A and the MATLAB code to run the analysis is in Appendix B.

2 Theoretical Background

Principal component analysis is applied to the dataset to reduce the number of dimensions of the data before applying machine learning. The full singular valued decomposition (SVD) decomposition is

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

where $\mathbf{U}$ is a $m \times m$ unitary matrix, $\mathbf{V}$ is a $n \times n$ unitary matrix, and Σ is a $m \times n$ diagonal matrix. SVD can diagonalize a matrix with equation 1 by transforming the variable as:

$$\mathbf{Y} = \mathbf{U}^*\mathbf{X} \tag{2}$$

The variance in $\mathbf{Y}$, the transformed variable, is:

$$\mathbf{C}_{\mathbf{Y}} = \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T = \frac{1}{n-1}\Sigma^2 \tag{3}$$

Linear discrimination analysis (LDA) is a method that generates a statistical decision based upon groups of data represented by principal components or by a proper orthogonal mode decomposition (Kutz, 2013). The goal of the two-class LDA is to project the data on a new basis function that produces well-separated statistical distribution between the data sets, maximizing the distance between inter-class data while minimizing the intra-class data.

For a two-class LDA, a projection $\mathbf{w}$ is constructed as:

$$\mathbf{w} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}} \tag{4}$$

The scatter matrices for between class ($\mathbf{S}_B$) and within-class ($\mathbf{S}_W$) data are defined as:

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{5}$$

$$\mathbf{S}_W = \Sigma_{j=1}^2 \Sigma_{\mathbf{x}}(\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T \tag{6}$$

which represent the variance within the data set and variance of differences in the means. The criterion established by equation 5 is the generalized Rayleigh quotient and has a generalized eigenvalue problem solution

$$\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_W\mathbf{w} \tag{7}$$

where λ is the maximum eigenvalue given as the quantity of interest and the projection basis.

The support-vector machine is a supervised learning model that classifies data and does a regression analysis. It constructs a hyperplane or set of them that can be used for classification, regression, or outlier detection (Wang, 2005). The decision tree machine learning technique predicts the value of a target variable based on several inputs and is a simple representation for classifying data (Somvanshi et al., 2016).

3 Algorithm Implementation and Development

The 28x28 dimension imagery data is read in for the training set of 6000 images and the test set of 1000 images by a `mnist_parse` function (see example images in Figure 1). The training and testing data are reshaped into a 748x6000 (training) and 784x1000 (testing) dataset, where each image is reshaped into a column vector and each column of data is a different image. For the training set, the mean of each column is removed and the singular valued decomposition (SVD) of the data is computed.

The variance of the data is computed as a percentage of the total variance. This is used to identify the number of modes which is necessary for further reduction of the data to apply LDA and other techniques. This is chosen as when the variance at a given mode a sufficiently low enough percentage of the total variance. The training and testing data projected with the transpose of the $\mathbf{U}$ basis vector from the training dataset and the dimensions are reduced based on the number of modes selected from the basis vector. These values, $\mathbf{XU}'$ for the training and testing set are then normalized by the maximum value from each set.

To run classifications for two and three digits, the $\mathbf{X}U'$ representative of selected numbers were identified based on the labels. The selected values and labels for the training data set where combined into one matrix and were used to compute the linear classifier (LDA) using the `fitcdiscr` function. Note that a linear classifier was build by hand following the dog/cat example (Kutz, 2013); however, due to the simplicity and robustness of the candid functions, the results are shown using the premade MATLAB function. The model developed by this function was used to predict the numbers based on the $\mathbf{XU}'$ values from the training and testing dataset. The error was tested by counting the number of incorrect values and them computing the percentage of correctly identified numbers from the sub-data set.

A similar processes was followed for the 3 number classification, except values of $\mathbf{XU}'$ and labels where selected for 3 numbers which were used to text and train the data set. A version of the dog/cat script for 3 selected numbers was attempted to create a linear classifier function, but the results were not close to the premade function and may have been incorrect. The same error metrics were computed.

To identify the hardest and easiest numbers to identify between, the LDA method was applied to each pairing of numbers and the percentage of correctly identified numbers for each pairing was stored for the training and testing dataset. The maximum and minimum errors were identified as the easiest and hardest pairs.

For the two-group classifier with SVM and decision tree, a similar method of the LDA was used except a different function was used: SVM was `fitcecoc` and `fitctree`. This was applied to the easiest and hardest pairs of numbers for the training and testing data. These methods were also used to classify the entire dataset. In these cases all the $\mathbf{XU}'$ values for all the images and labels were used to train the model. Then this model was applied to the training and testing data in the reduced dimension $\mathbf{XU}'$.

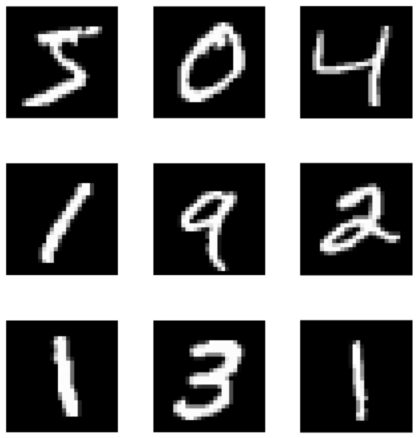

Figure 1: Nine example images of the handwritten numbers from the training dataset.

4 Computational Results

The SVD analysis was applied to the digit data that was reshaped. The singular value spectrum gradually decreases, where there is variance is many more modes than the previous assignment.

Table 1: The training and test set success rate for the linear discrimination analysis (LDA) with two and three classifiers and the support-vector machine (SVM) and decision tree machine learning techniques for all of the numbers.

	training	test
LDA [1, 2]	97.6	97.8
LDA [1, 2, 3]	95.1	94.4
SVM all numbers	90.2	90.8
decision tree all numbers	95.8	84.7

The choice of rank or necessary number of modes for a good image reconstruction was chosen based on when the percent of the total variance at a single mode became sufficiently small (around 1%), which was around 20 modes (Figure 3). Here, the analysis was performed with reduced dimensions down to 20 based on the variance drop in the singular value spectrum.

Here σ is a diagonal matrix with the energy values that can be used to scale the basis vectors, U is the basis/unitary vectors, and V is also a basis vector representing the opposite dimension of X than U does. Either $\mathbf{XU}'$ or $\mathbf{V}\Sigma^*$ can be used to help build the model. The projection onto three selected V-modes (mode 2, 3, and 5) show clusters based on the number as is important for the LDA and machine learning functions (Figure 4. The first four eigendigits for the SVD show the shape of the principal components (Figure 2.

The dimensions of the data values for the training and testing set were multiplied by the basis vector from the training set, $\mathbf{U}$ that has reduced dimensions based on the selected mode of 20. The prepared data was used to build a model which could then be used to test the training and testing data set. The success rate for the LDA for 2 and 3 digits have agreement above 97 and 94% respectively (Table 1 for digits 1,2, and 1,2,3). The LDA had less errors for the 2 digit classification than the 3 digit classification.

The easiest digits to separate using the LDA approach for the training data set were 7 and 6 and for the testing data set were 1 and 0. The hardest digits to separate using the LDA approach for the training data set were 5 and 3 and for the testing data set were 9 and 4. The LDA percent correct classification for the easiest numbers were $> 99\%$ and the hardest numbers where $< 94\%$ (Table 2). The percent correct for the SVM and decision tree for the easy and hard digit pairs in the training set were always greater than the LDA approach. However, the LDA actually performed very similar to the SVM and decision tree approach for the easiest values in the test data set. The LDA approach performed better for the hardest pair of numbers in the test data set, but significantly better than the decision tree approach. The SVM approach performed better than the decision tree for the test set, but not the training set.

The SVM and decision tree classifiers were able to separate between all ten digits with around 90% accuracy. The SVM approach performed worse than the decision tree for the training set, but better than the decision tree for the testing set. Overall, it seems like the decision tree is well suited for the training set, while the SVM approach is much better functioning for the test data set.

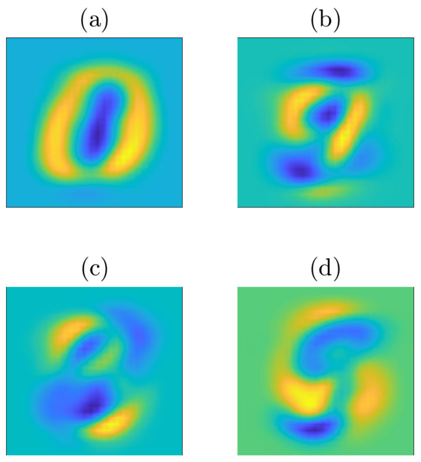

Figure 2: The first (a), second (b), third (c), and fourth (d) eigendigits from the singular valued decomposition of the test training set.

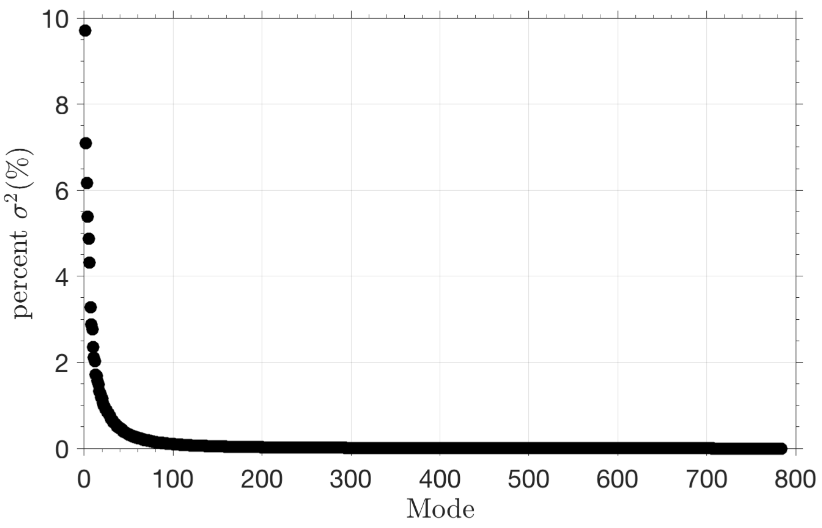

Figure 3: The percent of the total variance at each mode of the singular valued decomposition of the training dataset.

Table 2: Success rate for the linear discrimination analysis (LDA), support-vector machine (SVM), and decision tree machine learning techniques for the easiest and hardest pairs of numbers for the training and testing set of images.

	train		test	
	easiest [7,6]	hardest [5,3]	easiest [1,0]	hardest [9,4]
LDA	99.7	93.1	99.8	93.7
SVM	99.8	93.3	99.9	94.3
decision tree	99.9	98.6	99.7	89.3

5 Summary and Conclusions

Singular valued decomposition was used to reduced the dimensions in the data set of digits. The digits were classified using a linear discrimination algorithm, support vector machines (SVM), and decision tree classifiers. Overall, the LDA performed comparably well for the easiest digits, while the SVM and decision tree classifier performed better on the hard variables. The SVM worked better on the test data than the decision tree, while the opposite was true for the training data. The SVM and decision tree approach were used to classify the numbers. Generally, the success rate was not $> 90\%$. This approach exemplifies how different machine learning algorithms can be used to classify data, and using SVD can reduce dimensions and computational times.

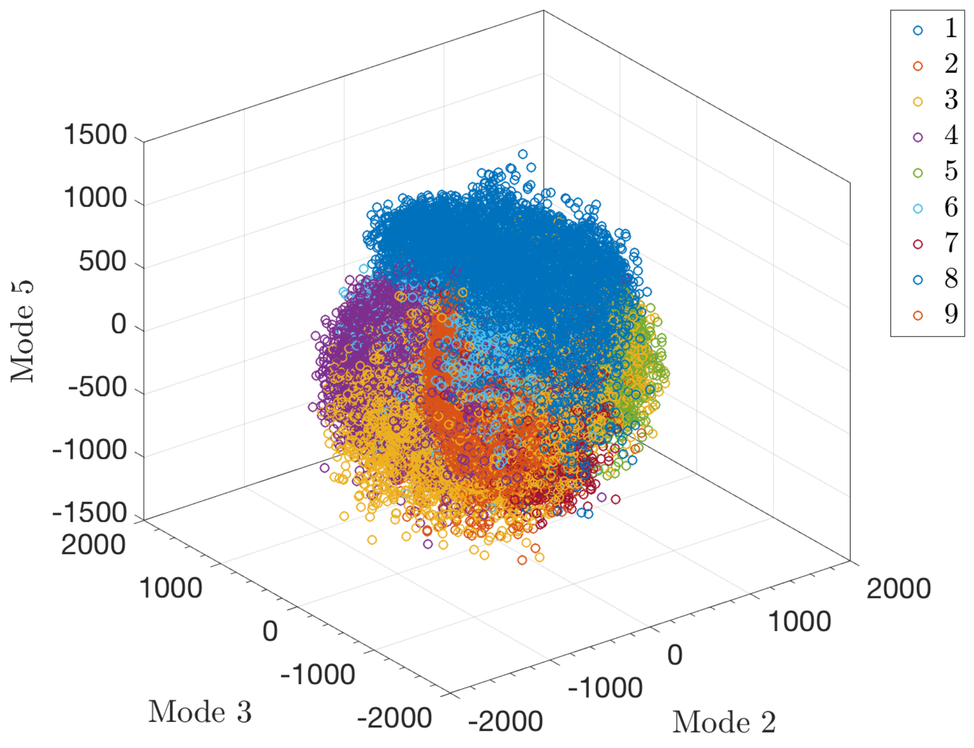

Figure 4: The projected 2nd, 3rd, and 5th **V**-modes from the singular valued decomposition colored by digit.

Appendix

A MATLAB functions used and brief implementation explanation

- `axis`: indicate the limits for the current plotted axes
- `calc_err`: function to compute the success rate
- `calc_svd`: function to compute the SVD
- `ind2sub`: find indexes of value
- `figure`: open new figure
- `find`: find values meeting some threshold
- `fitcdiscr`: classifier function for discrimination analysis that does a linear method
- `fitcecoc`: support vector machine model function for more than 2 groups
- `fitcsvc`: support vector machine model functino for 2 classifiers
- `fitctree`: decision tree classifier function
- `diag`: extract diagonals
- `double`: change value to type double
- `grid`: add grid lines to a figure
- `imshow`: plot image
- `*label`: label the x, y, z axes of plots
- `length`: find the length of the largest array dimension (grab the time-dimension in loops)
- `load`: load data from a .m file into workspace (load subdata.m)
- `max`: find the maximum value of an array or matrix
- `mean`: compute mean
- `min`: find the minimum value of an array or matrix
- `pcolor`: method to plot surfaces in 2d
- `plot3`: create 3d plot
- `predict`: function to use predicted class labels to identify misclassifications
- `print`: export and save figure

- `repmat`: created matrix with repeated values
- `scatter`: create scatter plot
- `set`: manipulate a figure
- `size`: grab size of a matrix
- `svd`: compute singular value decomposition values
- `zeros`: create a matrix of zeros
- `text`: define text for a figure

Function to extract the values of interest for the machine learning algorithms:

```
function [uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,
    Train_ux,Test_ux,labels_train ,labels_test);
% this function creates matrices of the digits for interest for the
   LDA and
% other machine learning algorithms

uxtrain= [];
uxtest = [];

labtrain = [];
labtest = [];


for i = 1:length(numchoice)
    idtrain = find(labels_train == numchoice(i));
    idtest = find(labels_test == numchoice(i));

    uxtrain = [uxtrain, Train_ux(:,idtrain)];
    uxtest = [uxtest, Test_ux(:,idtest)];

    labtrain = [labtrain; labels_train(idtrain)];
    labtest = [labtest; labels_test(idtest)];
end
```

Function to compute SVD:

```
function [s,u,v] = calc_svd(X)
% compute the principal component projection
% INPUT: X matrix of data valeus
% OUTPUT: outputs from the SVD function

[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean
```

```
[u,s,v]=svd(X,'econ'); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y=u'*X; % produce the principal components projection
svdout.u = u;
svdout.s = s;
svdout.v = v;
svdout.lambda = lambda;
```

Function to compute the success rate:

```
function [errNum,sucRate] = calc_err(result,labels)
% this function counts the number of erros in the results and
    calculates
% the success rate

% errors
% disp('Number of mistakes');
errNum = length(find(result ~= labels));

% errNum = sum(abs(ResVec - hiddenlabels))
% disp('Rate of success');
TestNum = length(labels);
sucRate = 1-errNum/TestNum;
```

B MATLAB code

Code available at: `https://github.com/cmbaker94/Baker`$_AMATH582$

```
clear all
close all
clc

addpath(genpath('/Users/cmbaker9/Documents/MTOOLS'))

%% STEP 0: Locate and load data

datapath    = '/Users/cmbaker9/Documents/UW_Classes/
    AMATH_Data_Analysis/HW4/data/';
figfolder   = '/Users/cmbaker9/Documents/UW_Classes/
    AMATH_Data_Analysis/HW4/figures/';

%% Load data

[images_train, labels_train] = mnist_parse([datapath,'train-images
    -idx3-ubyte'], [datapath,'train-labels-idx1-ubyte']);
```

```
[images_test , labels_test] = mnist_parse([datapath,'t10k-images-
    idx3-ubyte'], [datapath,'t10k-labels-idx1-ubyte']);

figure('units','inches','position',[1 1 6 6],'Color','w');
for j=1:9
    subplot(3,3,j)
    imshow(images_train(:,:,j))
end
Sname1 = [figfolder,'eximages'];
print(Sname1,'-dpng')

%% run svd on data

res = size(images_train,1);
ntrain = size(images_train,3);
ntest = size(images_test,3);

Xtrain = double(reshape(images_train,res*res,ntrain));
[S,U,V] = calc_svd(Xtrain);

Xtest = double(reshape(images_test,res*res,ntest));

%% inspect training set

sig=diag(S);
energy(1)=sig(1)/sum(sig);
for i = 2:length(sig)
    energy(i) = sum(sig(1:i))/sum(sig);
end

% pricipal components 1 , 2 , 3 ,4
figure('units','inches','position',[1 1 6 6],'Color','w');
titlenum = ['a','b','c','d'];
for j=1:4
    subplot(2,2,j)
    ut1=reshape(U(:,j),res,res);
    ut2=ut1(res:-1:1,:);
    pcolor(ut2); shading interp
    title(['(',titlenum(j),')'],'interpreter','latex','fontsize'
        ,20)
    set(gca,'Xtick',[],'Ytick',[])
end
Sname1 = [figfolder,'princ_components'];
print(Sname1,'-dpng')

% covariance
```

```matlab
lambda=diag(S).^2;

figure('units','inches','position',[1 1 10 6],'Color','w');
mode = 1:length(lambda);
scatter(mode,lambda/sum(lambda)*100,100,'k','fill')
hold on
xlabel('Mode','interpreter','latex','fontsize',20)
ylabel('percent $\sigma^2 (\%)$','interpreter','latex','fontsize'
    ,20)
grid on
box on
h1=gca;
set(h1,'fontsize',20);
set(h1,'tickdir','out','xminortick','on','yminortick','on');
set(h1,'ticklength',1*get(h1,'ticklength'));
Sname1 = [figfolder,'covariance'];
print(Sname1,'-dpng')


%% Projection onto 3 V-modes
SV = V*S;

figure('units','inches','position',[1 1 10 8],'Color','w');
for label=0:9
    label_indices = find(labels_train == label);
    plot3(SV(label_indices,2),SV(label_indices,3),SV(label_indices
        ,5),'o','Linewidth',1)
    hold on
end
xlabel('Mode 2','interpreter','latex','fontsize',20)
ylabel('Mode 3','interpreter','latex','fontsize',20)
zlabel('Mode 5','interpreter','latex','fontsize',20)
h2 = legend({'1','2','3','4','5','6','7','8','9'});
set(h2,'interpreter','latex','fontsize',20,'orientation','vertical
    ');
grid on
box on
h1=gca;
set(h1,'fontsize',20);
set(h1,'tickdir','out','xminortick','on','yminortick','on');
set(h1,'ticklength',1*get(h1,'ticklength'));
Sname1 = [figfolder,'3d'];
print(Sname1,'-dpng')

%% make matrixes

```

```
feature = 20;
Train_ux = U(:,1:feature)'*Xtrain;
Test_ux = U(:,1:feature)'*Xtest;

% Train_ux = Train_ux-mean(Train_ux,1);
% Test_ux = Test_ux-mean(Test_ux,1);

Train_ux = Train_ux/max(Train_ux(:));
Test_ux = Test_ux/max(Test_ux(:));

%% LDA: pick 2 numbers
% Pick two digits. See if you can build a linear classifier (LDA)
    that can reasonable identify them.

numchoice = [1 2];

[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);

Mdl = fitcdiscr(uxtrain',labtrain);
restrain = predict(Mdl,uxtrain');
restest = predict(Mdl,uxtest');
% train error
[errNum.LDA2pick_train,sucRate.LDA2pick_train] = calc_err(restrain
    ,labtrain);
% test error
[errNum.LDA2pick_test,sucRate.LDA2pick_test] = calc_err(restest,
    labtest);


%% LDA: pick 3 numbers
% Pick three digits. Try to build a linear classifier to identify
    these three now.

numchoice = [1 2 3];

[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);

Mdl = fitcdiscr(uxtrain',labtrain);
restrain = predict(Mdl,uxtrain');
restest = predict(Mdl,uxtest');
% train error
[errNum.LDA3pick_test,sucRate.LDA3pick_test] = calc_err(restrain,
    labtrain);
% test error
```

```
[errNum.LDA3pick_train,sucRate.LDA3pick_train] = calc_err(restest,
   labtest);


%% LDA Hardest / easiest
% Which two digits in the data set appear to be the most difficult
     to separate? Quantify the accuracy of the separation with LDA
   on the test data.
% Which two digits in the data set are most easy to separate?
   Quantify the accuracy of the separation with LDA on the test
   data

for i = 1:10
    for j = 1:10
        numchoice = [i j]-1;
        if i ~= j
            [uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice
               ,Train_ux,Test_ux,labels_train,labels_test);

            Mdl = fitcdiscr(uxtrain',labtrain);
            restrain = predict(Mdl,uxtrain');
            restest = predict(Mdl,uxtest');

            % train error
            [errNum_train(i,j),sucRate_train(i,j)] = calc_err(
               restrain,labtrain);

            % test error
            [errNum_test(i,j),sucRate_test(i,j)] = calc_err(
               restest,labtest);
        elseif i == j
            errNum_train(i,j) = NaN;
            sucRate_train(i,j) = NaN;

            errNum_test(i,j) = NaN;
            sucRate_test(i,j) = NaN;
        end
    end
end

% train
% find easiest
[sucRate.ldaeas_train,id] = max(sucRate_train(:)); % find max of
   spectra index in array
[num1,num2] = ind2sub(size(sucRate_train),id);
display('easiest to seperate')
```

```
easiest.train = [num1 num2]-1;

% find hardest
[sucRate.ldahard_train,id] = min(sucRate_train(:)); % find max of
   spectra index in array
[num1,num2] = ind2sub(size(sucRate_train),id);
display('easiest to seperate')
hardest.train = [num1 num2]-1;

% test
% find easiest
[sucRate.ldaeas_test,id] = max(sucRate_test(:)); % find max of
   spectra index in array
[num1,num2] = ind2sub(size(sucRate_test),id);
display('easiest to seperate')
easiest.test = [num1 num2]-1
% find hardest
[sucRate.ldahard_test,id] = min(sucRate_test(:)); % find max of
   spectra index in array
[num1,num2] = ind2sub(size(sucRate_test),id);
display('easiest to seperate')
hardest.test = [num1 num2]-1

%% SVM and decisiontree on all

% SVM
Mdl = fitcecoc(Train_ux',labels_train);
restrain = predict(Mdl,Train_ux');
restest = predict(Mdl,Test_ux');
% train error
[errNum.SVMall_train,sucRate.SVMall_train] = calc_err(restrain,
   labels_train);
% test error
[errNum.SVMall_test,sucRate.SVMall_test] = calc_err(restest,
   labels_test);
clear restrain retest

% Decision tree
% classification tree on fisheriris data
tree=fitctree(Train_ux',labels_train);
restrain = predict(tree,Train_ux');
restest = predict(tree,Test_ux');
% train error
[errNum.treeall_train,sucRate.treeall_train] = calc_err(restrain,
   labels_train);
% test error
```

```
[errNum.treeall_test,sucRate.treeall_test] = calc_err(restest,
    labels_test);
clear restrain retest

%% SVM on hardest and easiest
% hardest train
numchoice = hardest.train;
[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);
% SVM: 2 num
Mdl = fitcsvm(uxtrain',labtrain);
restrain = predict(Mdl,uxtrain');
% train error
[errNum.SVMhard_train,sucRate.SVMhard_train] = calc_err(restrain,
    labtrain);
clear restrain restest

% hardest test
numchoice = hardest.test;
[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);
% SVM: 2 num
Mdl = fitcsvm(uxtrain',labtrain);
restest = predict(Mdl,uxtest');
% train error
[errNum.SVMhard_test,sucRate.SVMhard_test] = calc_err(restest,
    labtest);
clear restrain restest

% easiest train
numchoice = easiest.train;
[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);
% SVM: 2 num
Mdl = fitcsvm(uxtrain',labtrain);
restrain = predict(Mdl,uxtrain');
% train error
[errNum.SVMeas_train,sucRate.SVMeas_train] = calc_err(restrain,
    labtrain);
clear restrain restest

% easiest test
numchoice = easiest.test;
[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);
% SVM: 2 num
```

```
Mdl = fitcsvm(uxtrain',labtrain);
restest = predict(Mdl,uxtest');
% train error
[errNum.SVMeas_test,sucRate.SVMeas_test] = calc_err(restest,
    labtest);
clear restrain restest

%% decision tress on hardest and easiest

% hardest train
numchoice = hardest.train;
[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);
% SVM: 2 num
Mdl = fitctree(uxtrain',labtrain);
restrain = predict(Mdl,uxtrain');
% train error
[errNum.treehard_train,sucRate.treehard_train] = calc_err(restrain
    ,labtrain);
clear restrain restest

% hardest test
numchoice = hardest.test;
[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);
% tree: 2 num
Mdl = fitctree(uxtrain',labtrain);
restest = predict(Mdl,uxtest');
% train error
[errNum.treehard_test,sucRate.treehard_test] = calc_err(restest,
    labtest);
clear restrain restest

% easiest train
numchoice = easiest.train;
[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
    Test_ux,labels_train,labels_test);
% tree: 2 num
Mdl = fitctree(uxtrain',labtrain);
restrain = predict(Mdl,uxtrain');
% train error
[errNum.treeeas_train,sucRate.treeeas_train] = calc_err(restrain,
    labtrain);
clear restrain restest

% easiest test
```

```
numchoice = easiest.test;
[uxtrain,uxtest,labtrain,labtest] = get_nums(numchoice,Train_ux,
   Test_ux,labels_train,labels_test);
% tree: 2 num
Mdl = fitctree(uxtrain',labtrain);
restest = predict(Mdl,uxtest');
% train error
[errNum.treeeas_test,sucRate.treeeas_test] = calc_err(restest,
   labtest);
clear restrain restest
```

References

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

Madan Somvanshi, Pranjali Chavan, Shital Tambade, and SV Shinde. A review of machine learning techniques using decision tree and support vector machine. In *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, pages 1--7. IEEE, 2016.

Lipo Wang. *Support vector machines: theory and applications*, volume 177. Springer Science & Business Media, 2005.