LDA and Digits Classification

Luotong Kang

Abstract

In this report, we will introduce Linear Discriminant Analysis (LDA). Using a set of training data from the MNIST data, we will develop classifiers of digits under different conditions and testing their accuracy.

1 Introduction

Image classifications usually don't bother human, however, computers may find them difficult. Machine learning is the field studying how to train the computer to perform things originally considered to be experience-based. In this report, we focus on a supervised machine learning approach called Linear Discriminant Analysis (LDA).

Like Principle Component Analysis (PCA) and Independent Component Analysis, LDA is also a powerful tool building on top of Singular Value Decomposition (SVD). This time, we will apply this data-driven approach to MNIST images and train a classifier to identify digits represented by certain image. We'll compare separation error in training set and success testing rate when different methods or groups are chosen.

2 Theoretical Background

We'll assume that the audience have basic knowledge about SVD. The result of SVD is three matrices: $\mathbf{U}, \mathbf{\Sigma}$ and $\mathbf{V}$. Columns of $\mathbf{U}$ are left singular vectors, representing the principal components. Columns in $\mathbf{V}$ are right singular vectors, representing how each layer of data is represented in the principle component basis. $\mathbf{\Sigma}$ contains singular values in descending mode tells how much energy is captured by each mode. They are the basis of PCA.

The essence of LDA is to extend the idea of projection from PCA. The goal of LDA is to find a suitable projection that maximizes the distance between inter-class data. In other words, we try to create distinct clusters of same-classed data using training data sets. Mathematically, we are dealing with

$$\mathbf{w} = \text{argmax}\frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_w\mathbf{w}},$$

where $\mathbf{S}_B$ is between-class scatter matrix, $\mathbf{S}_w$ is within-class scatter matrix.

Since strict optimization is tough, but we know that $\mathbf{w}$ that maximizes the above quotient is the eigenvector corresponding to the largest eigenvalue of the problem:

$$\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_w\mathbf{w}$$

For 2 datasets, we first calculate the means for each our groups for each feature μ_1, μ_2. Then we compute scatter matrices:

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T, \quad \mathbf{S}_w = \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T$$

For more than 2 datasets, we first calculate the overall mean μ and the mean of each groups μ_j. Then we compute scatter matrices:

$$\mathbf{S}_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T, \quad \mathbf{S}_w = \sum_{j=1}^{N} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T$$

3 Algorithm Implementation and Development

Lines 1-8 First we load, reshape and reorder the data, so that each column contains 742 pixels of a different picture and ordered in ascending digit classification. Then we perform SVD and obtain the projection on principle components `proj_on_PC = S*V'` .

Lines 9-34 Next, we will plot indications of SVD result (analysis in next section). By plotting the diagonal of `S`, we get singular value spectrum (Figure 1). Using the spectrum, we are able to pick out proper feature number. Then, we project the data onto **V**-modes 3,4,5 by calling the correct element in column 3,4,5 in **V**. We already know the order of layers is the same in `X` and `labels`. For each digit, locate the digit value in I using `lab_ind=find(labels==lab)` and use `lab_ind` to call correct elements.

Lines 35-44 Similar to Lines 1-8, we first obtain ordered testing data `te_X` and correct classification `hiddenlabels`. In addition, we initiate separation error list and success testing rate list, each have 45 elements (total number of 2 digits combination).

Lines 45-142 In the part, we perform LDA using a written function (see next paragraph) twice (first to 2 selected digit, then to 3 selected digits). Then, we create some illustrative plots. Using the same function, we then perform LDA to all 45 combinations of digits and compare the training error and success testing.

Function `two_digits_trainer_tester` We generalize the process of deriving separation error rate and testing success rate using this function. Firstly we locate the the selected digits in `proj_on_PC` with chosen features. Then we solve between-class and within-class scatter matrix to get the correct subspace `w`. To project onto it just multiply `v1 = w'*fir; v2 = w'*sec;`. Before finding the threshold, we make first input digit (`dig1`) always below to make easy decision for every digit. To pick the threshold, we start with the largest `dig1` and smallest `dig2` (first sorting them to be `t1`, `t2`). We go over the while loop until the relative relation changes and find threshold in the middle. Error rate is computed by dividing false classification with total number of images of two digits. Finally, we classify the testing data by projecting on to first feature numbers of columns in `U`. Success classification rate is calculated by `1-errNum/TestNum`.

Lines 143-155 We use built-in command `fitctree()` to classify all ten digit and make a decision tree plot. We also train a classifier using support vector machines and test it by computing success rate.

4 Computational Results

4.1 SVD Analysis

U is the left singular matrix. In this specific application, columns (each is a different principle component) can be plotted as images that carry characteristic information of the whole data set.

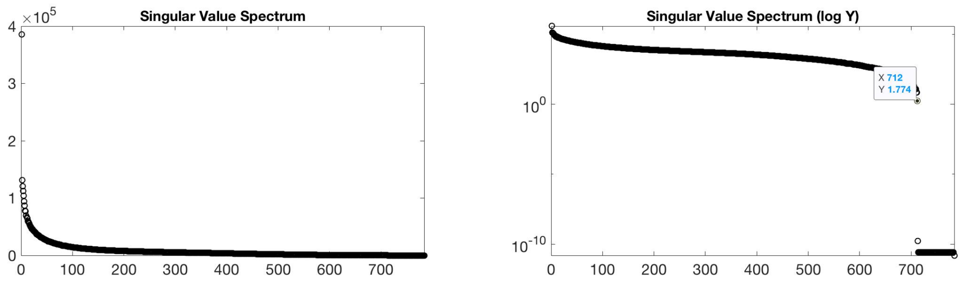

Figure 1: Singular value spectrum (`dig(S)`).

Σ is a matrix with singular values on the main diagonal. As said in introduction of SVD, the singular value are the scaling factor of the transform between principle component basis and current data. In other words, they indicate how much weight each principal component carries. From the normal scale graph in the left, we cannot find sharp decreases in energy captured. In logY scale, however, we can find a sharp decrease after mode 712. Therefore, we choose 712 features for LDA.

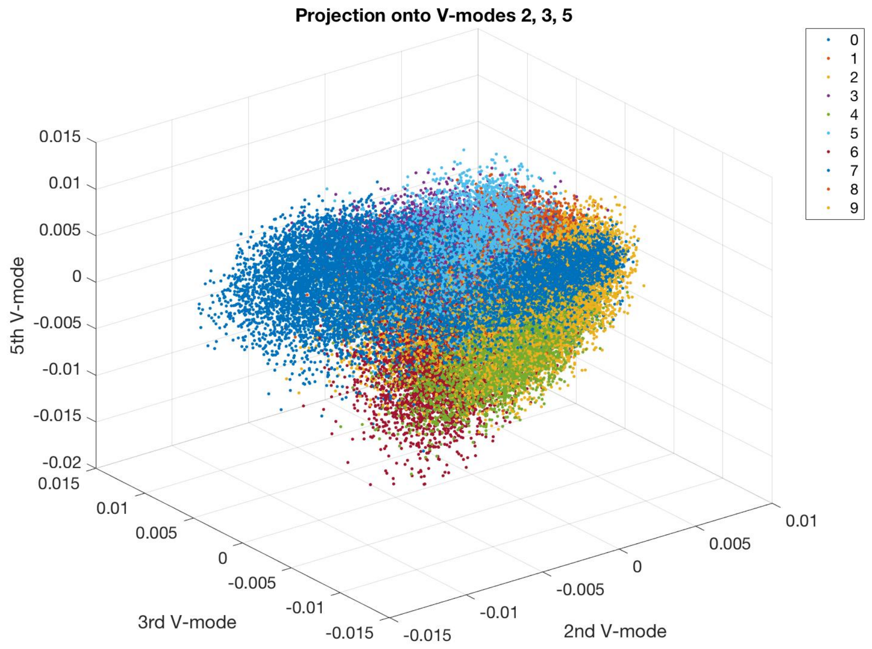

Figure 2: 3D projection onto V-modes 2,3 and 5.

V is the right singular matrix. Column vectors indicate how each of the digits is represented in PCA basis, in which column number is the mode we are projecting onto.

4.2 Linear Discriminant Analysis

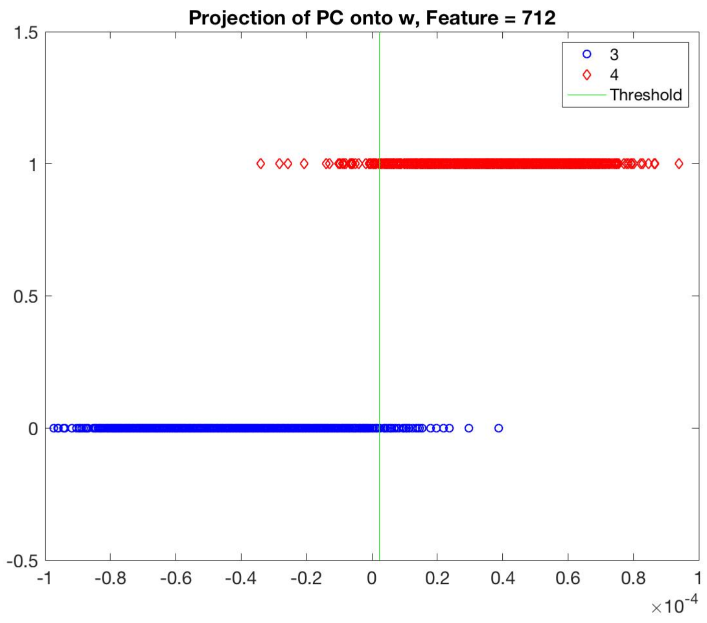

Figure 3: Projections on w when selecting 2 digits.

The horizontal axis is the projection coordinates on **w**. Difference vertically is created to make the threshold easier to be observed. Using while loop described in previous section, we found the threshold. By just checking with our eyes, we know that the threshold is reasonable.

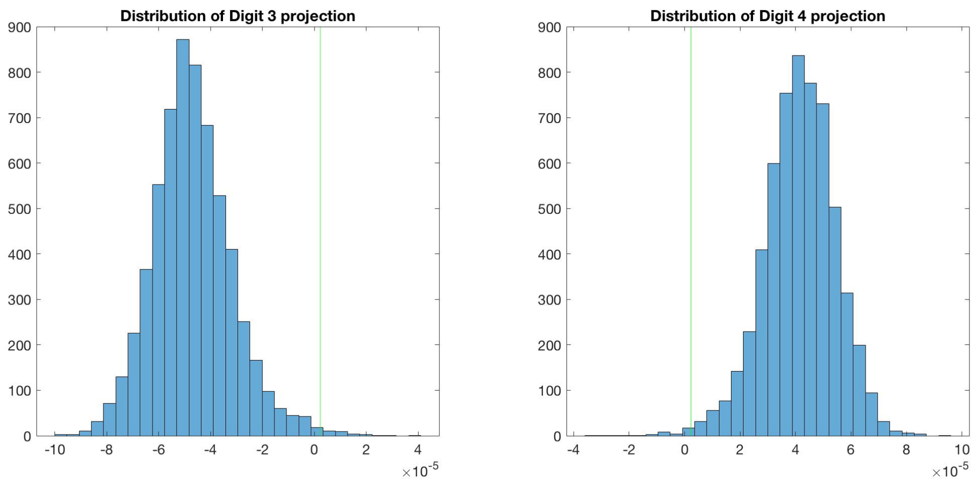

Figure 4: Distribution histogram of training data with selected threshold.

In distribution graph (Figure 4), each bin has height representing the count of images falling in that interval. We can see that for each of two digits, most images fall in one different side of threshold. Also, the direction of separation is also correct because first input (here is digit 3) is in lower (left) part. The error rate of training separation is calculated to be about 00.5261839%, which is small. The success rate of classifying testing data is about 99.47% (note we have not computed this by 1-error). Thus, it is easy to separate 3 and 4 in MNIST images.

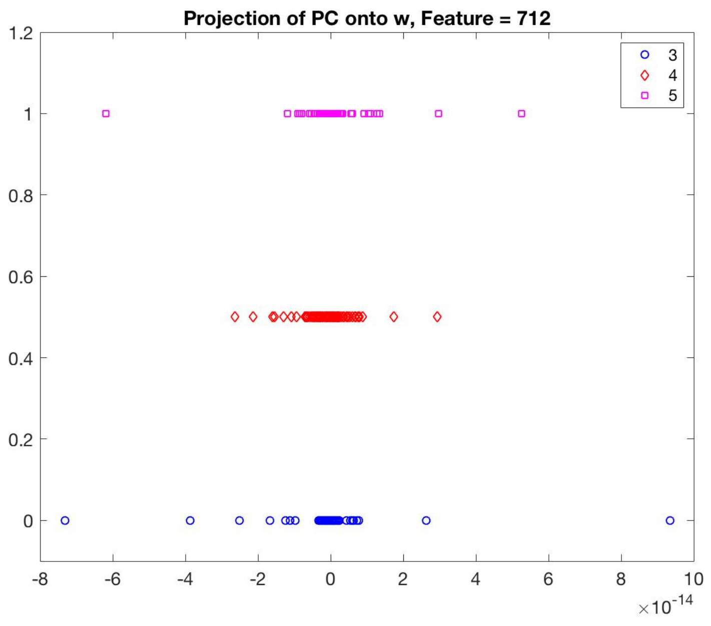

Figure 5: Projections on w when selecting 3 digits.

When it comes to classify 3 groups, there will not always exist a proper threshold. In Figure 5, three projections on **w** almost overlap each other, which means that it is difficult (or maybe impossible) to separate them. We may use some standard like mean to set threshold, but they will absolutely have a high error classification rate. Thus, we'd like to argue that we cannot find a classifier using LDA in this case.

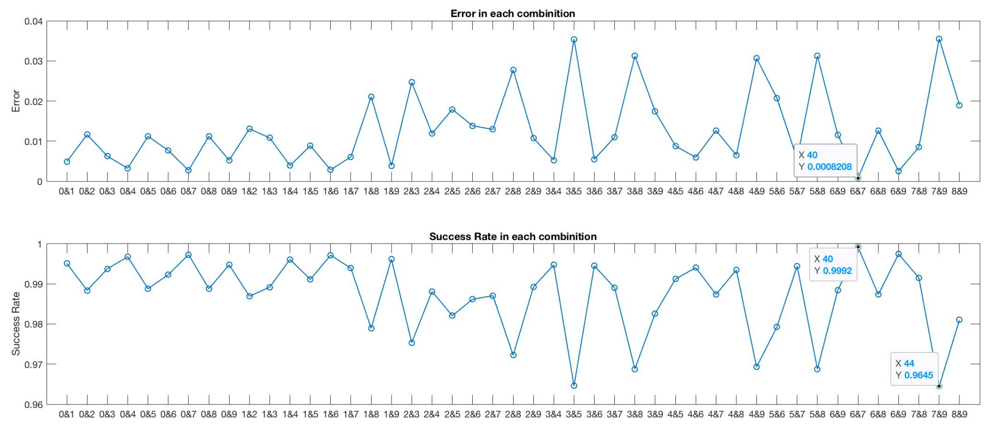

Figure 6: Error separation rate and success testing rate of all two digit combinitions.

After repeating LDA for all 45 combinations of 2 digits, we can evaluate their performance from plot of error rate in training data separation and plot of success rate in success rate. Group of 7 and 9 is the most difficult to separate (max error) and performs the worst on classifying testing data (min success). Group of 6 and 7 is the easiest to separate (min error) and performs the best on classifying testing data (max success).

Amazingly, they are symmetric with respect to y=0.5, which means *error rate + success rate = 1*, while using different methods (training and testing) and different dataset. This phenomenon is observed because the datasets are large enough so that the separation exactly predict the performance.

4.3 Comparison with SVM and decision tree

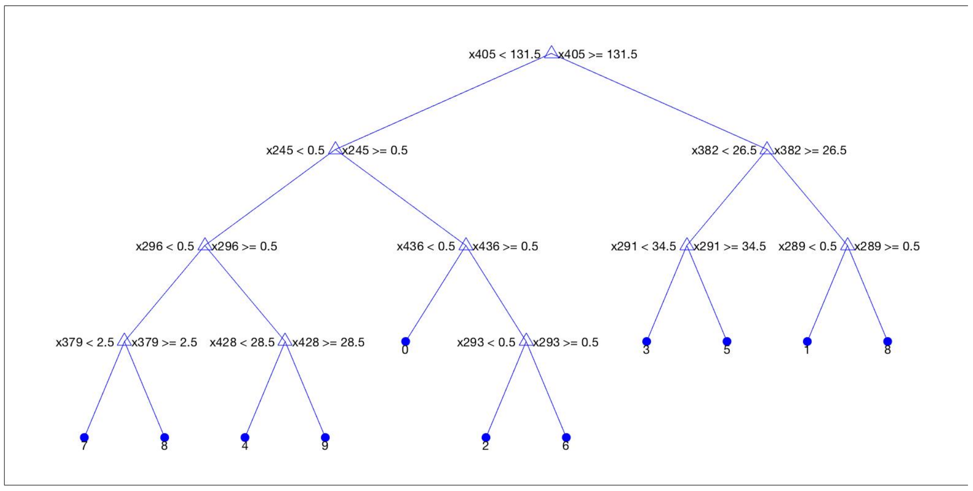

Figure 7: Classification tree with restricted split number.

When the `MaxNumSplits` is not specified, we can classify the 10 digits with a small classification error (13.03%). However, the decision tree has trivial value because there are so much branches. By setting the `MaxNumSplits`, we trade off low error (classification error = 43.46%)) with simply decision tree and faster process.

For Support Vector Machines (SVM), instead of projecting to a line, we separate data on a plane. This allow us to be more accurate when more groups are considered. We get a success rate of classifying all 10 digits in new testing data around 90.64%, which is pretty high. However, the cost of this accuracy is the low running speed.

5 Summary and Conclusions

Using Linear Discriminant Analysis, we've trained classifiers to predict the digits in MNIST data. The effectiveness of this supervised machine leaning depend on the variance with-in class and between-class. The separation will also become difficult if we increase the number of class. When dealing with classification of many groups (3 or more), we may consider to use other methods like Support Vector Machines, which though takes much longer time to train. This trade-off between accuracy and running time is a one of the main subjects to study in machine learning.

6 Appendices

6.1 MATLAB functions used

NOT listed chronologically
`load()`: loads the data stored in file.
`zeros(size)`,`ones(size)`: creates matrices with all entries being zero/one with indicated size.
`plot()`: creates a 2-D line plot of the data in Y versus the corresponding values in X.
`histogram(X,nbins)`: creates a histogram plot of X using a number of bins specified by nbins. It displays the bins as rectangles such that the height of each rectangle indicates the number of elements in the bin.
`set(gca,)`, `set(gcf,)`, `title()`, `xlabel()`, `ylabel()`, `yticks()`, `yticklabels()`, `subplot()`, `hold on`: plotting commands.
`find(X)`: returns a vector containing the linear indices of each nonzero element in X.
`[M,I]=max(A)`: returns the maximum elements of an array M and the index I.
`[M,I]=min(A)`: returns the minimum elements of an array M and the index I.
`size(A)`: returns a row vector whose elements are the lengths of the corresponding dimensions of A.
`length(A)`: returns the length of the largest array dimension in X.
`mean(V)`: V is a vector, returns the mean of the elements.
`sum(V)`: V is a vector, returns the sum of the elements.
`diag(A)`: returns a column vector of the main diagonal elements of A.
`[B,I] = sort(A)`: sorts the elements of A in ascending order. I is the arrangement of the elements (B = A(I)).
`[U,S,V] = svd(A)`: returns numeric unitary matrices U and V with the columns containing the singular vectors, and a diagonal matrix S containing the singular values. The matrices satisfy the condition A = U*S*V'.
`[V,D] = eig(A)`: The columns of V present eigenvectors of A. The diagonal matrix D contains eigenvalues.
`tree = fitctree(Tbl,Y)` : returns a fitted binary classification decision tree based on the input variables contained in the table Tbl and output in vector Y.
`view(tree)`: view graphical display of the classification tree.
`kfoldLoss(CVMdl)` returns the classification loss (classification error by default) obtained by the cross-validated kernel ECOC model CVMdl.
`Mdl = fitcecoc(Tbl,Y)`: returns an ECOC model using the predictors in table Tbl and the class labels in vector Y.
`predict(Mdl,X)`: returns predicted class labels for each observation in the predictor data X based on classification model Mdl.

6.2 MATLAB codes

```
clc;clear;close all;
[images, preordlabels] = ...
    mnist_parse('train-images-idx3-ubyte','train-labels-idx1-ubyte');
[labels,I]= sort(preordlabels);
preorder = double(reshape(images,size(images,1)*size(images,2),[]));
X = preorder(:,I);
[U,S,V] = svd(X,'econ');
proj_on_PC = S*V';

%% Plot singular values
figure(1)
subplot(121)
plot(diag(S),'ko','Linewidth',1)
set(gca,'Fontsize',16,'Xlim',[0 length(diag(S))])
title('Singular Value Spectrum','Fontsize',15)
subplot(122)
semilogy(diag(S),'ko','Linewidth',1)
set(gca,'Fontsize',16,'Xlim',[0 length(diag(S))])
title('Singular Value Spectrum (log Y)','Fontsize',15)
set(gcf,'Position',[0,0,1300,300])

%% Projection onto 3 V-modes
figure(2)
for lab=0:9
    lab_ind=find(labels==lab);
    plot3(V(lab_ind,2),V(lab_ind,3),V(lab_ind,5),'.','DisplayName', ...
        sprintf('%i',lab))
    hold on
end
xlabel('2nd V-mode'),ylabel('3rd V-mode'),zlabel('5th V-mode')
title('Projection onto V-modes 2, 3, 5','Fontsize',35)
legend()
grid on
set(gca,'Fontsize',14)
set(gcf,'Position',[0,0,800,600])

%% load test data
feature=712;
[te_images, te_preordlabels] = ...
    mnist_parse('train-images-idx3-ubyte','train-labels-idx1-ubyte');
[hiddenlabels,te_I]= sort(te_preordlabels);
te_preorder = ...
    double(reshape(te_images,size(te_images,1)*size(te_images,2),[]));
te_X = te_preorder(:,te_I);
Err_tr=zeros([1,45]);
SucRate=zeros([1,45]);
U2=U(:,1:feature);

%% Choose 2 digits: 3, 4
[Err_tr(25),SucRate(25),thre34,w34,vd3,vd4,sort3,sort4] = ...
    twodigits_trainer_tester(3,4,feature,proj_on_PC,labels,U2,hiddenlabels,te_X);
figure(3)
```

```
plot(vd3,zeros([1,length(vd3)]),'ob','Linewidth',1,'DisplayName','3')
hold on
plot(vd4,ones([1,length(vd4)]),'dr','Linewidth',1,'DisplayName','4')
plot([thre34,thre34],[-0.5,1.5],'g-','DisplayName','Threshold')
legend
title('Projection of PC onto w, Feature = 712','Fontsize',20)
set(gca,'Fontsize',14,'Ylim',[-.5 1.5])
set(gcf,'Position',[0,0,650,500])
figure(4)
subplot(121)
histogram(sort3,30); hold on, plot([thre34,thre34],[0,900],'g-')
title('Distribution of Digit 3 projection','Fontsize',17)
set(gca,'Fontsize',14)
subplot(122)
histogram(sort4,30); hold on, plot([thre34,thre34],[0,900],'g-')
title('Distribution of Digit 4 projection','Fontsize',17)
set(gca,'Fontsize',14)
set(gcf,'Position',[0,0,1200,500])

%% Choose 3 digits: 3,4,5
ind_for_3=find(labels==3);
ind_for_4=find(labels==4);
ind_for_5=find(labels==5);
d3 = proj_on_PC(1:feature,ind_for_3(1):ind_for_3(end));
d4 = proj_on_PC(1:feature,ind_for_4(1):ind_for_4(end));
d5 = proj_on_PC(1:feature,ind_for_5(1):ind_for_5(end));
n3 = size(d3,2);
n4 = size(d4,2);
n5 = size(d5,2);
m3 = mean(d3,2);
m4 = mean(d4,2);
m5 = mean(d5,2);
mall=mean([m3,m4,m5]);
Sw=0;
for k=1:n3
    Sw = Sw + (d3(:,k)-m3)*(d3(:,k)-m3)';
end
for k=1:n4
    Sw = Sw + (d4(:,k)-m4)*(d4(:,k)-m4)';
end
for k=1:n5
    Sw = Sw + (d5(:,k)-m5)*(d5(:,k)-m5)';
end
Sb = (m3-mall)*(m3-mall)'+(m5-mall)*(m5-mall)'+(m5-mall)*(m5-mall)';
[V2,D] = eig(Sb,Sw);
[¬,ind] = max(abs(diag(D)));
w345 = V2(:,ind);
w345 = w345/norm(w345,2);
v3 = w345'*d3;
v4 = w345'*d4;
v5 = w345'*d5;
figure(5)
plot(v3,zeros([1,length(v3)]),'ob','Linewidth',1,'DisplayName','3')
hold on
plot(v4,0.5*ones([1,length(v4)]),'dr','Linewidth',1,'DisplayName','4')
plot(v5,ones([1,length(v5)]),'sm','Linewidth',1,'DisplayName','5')
%plot([thre34,thre34],[-0.5,1.5],'g-','DisplayName','Threshold')
legend
title('Projection of PC onto w, Feature = 712','Fontsize',20)
set(gca,'Fontsize',14,'Ylim',[-.1 1.2])
set(gcf,'Position',[0,0,650,500])

%% All possible combinations of 2 digits
firdig=[zeros([1,9]) ones([1,8]) 2*ones([1,7]) 3*ones([1,6]) ...
    4*ones([1,5]) 5*ones([1,4]) 6*ones([1,3]) 7*ones([1,2]) [8]];
secdig=[1:9 2:9 3:9 4:9 5:9 6:9 7:9 8:9 9];
for h=1:45
     [Err_tr(h),SucRate(h)] = ...
         twodigits_trainer_tester(firdig(h),secdig(h),feature,proj_on_PC, ...
         labels,U2,hiddenlabels,te_X);
end
[¬,Itrmin]=min(Err_tr);
[¬,Itrmax]=max(Err_tr);
[¬,Itemin]=min(SucRate);
[¬,Itemax]=max(SucRate);
mintrerrNum = [firdig(Itrmin) secdig(Itrmin)]; % easiest to separate
maxtrerrNum = [firdig(Itrmax) secdig(Itrmax)]; % most difficult to ...
    separate
minsucNum = [firdig(Itemin) secdig(Itemin)]; % lowest sucess rate
maxsucNum = [firdig(Itemax) secdig(Itemax)]; % highest sucess rate

%%
figure(6)
subplot(211)
plot(1:length(Err_tr),Err_tr,'-o','Linewidth',1)
title('Error in each combinition','Fontsize',30)
ylabel('Error')
xticks(1:45)
xticklabels({'0&1','0&2','0&3','0&4','0&5','0&6','0&7','0&8','0&9','1&2', ...
    '1&3','1&4','1&5','1&6','1&7','1&8','1&9','2&3','2&4','2&5','2&6', ...
    '2&7','2&8','2&9','3&4','3&5','3&6','3&7','3&8','3&9','4&5','4&6', ...
    '4&7','4&8','4&9','5&6','5&7','5&8','6&9','6&7','6&8','6&9','7&8','7&9','8&9'})
set(gca,'Fontsize',10,'Xlim',[0,46])
subplot(212)
plot(1:length(SucRate),SucRate,'-o','Linewidth',1)
title('Success Rate in each combinition','Fontsize',30)
ylabel('Success Rate')
xticks(1:45)
xticklabels({'0&1','0&2','0&3','0&4','0&5','0&6','0&7','0&8','0&9','1&2', ...
    '1&3','1&4','1&5','1&6','1&7','1&8','1&9','2&3','2&4','2&5','2&6', ...
    '2&7','2&8','2&9','3&4','3&5','3&6','3&7','3&8','3&9','4&5','4&6', ...
    '4&7','4&8','4&9','5&6','5&7','5&8','6&9','6&7','6&8','6&9','7&8','7&9','8&9'})
set(gca,'Fontsize',10,'Xlim',[0,46])
set(gcf,'Position',[0,0,1500,500])

%% Classification Tree
% no MaxNumSplits
tree1=fitctree(X',labels,'CrossVal','on');
classError1 = kfoldLoss(tree1);
% have MaxNumSplits
```

```
tree2=fitctree(X',labels,'CrossVal','on','MaxNumSplits',10);
view(tree2.Trained{1},'Mode','graph');
classError2 = kfoldLoss(tree2);

%% SVM classifier
Mdl = fitcecoc(X',labels);
test_labels_svm = predict(Mdl,te_X');
svm_sucRate= sum(test_labels_svm==labels)/length(labels);
```

Function codes:

```
function [err_tr,sucRate,threshold,w,v1,v2,sort1,sort2] = ...
    twodigits_trainer_tester(dig1,dig2,feature,proj_on_PC,labels,U2, ...
    hiddenlabels,te_X)
     ind1=find(labels==dig1);
     ind2=find(labels==dig2);
     fir = proj_on_PC(1:feature,ind1(1):ind1(end));
     sec = proj_on_PC(1:feature,ind2(1):ind2(end));
     n1 = size(fir,2);
     n2 = size(sec,2);
     m1 = mean(fir,2);
     m2 = mean(sec,2);
     Sw=0;
     for k=1:n1
         Sw = Sw + (fir(:,k)-m1)*(fir(:,k)-m1)';
     end
     for k=1:n2
         Sw = Sw + (sec(:,k)-m2)*(sec(:,k)-m2)';
     end
     Sb = (m1-m2)*(m1-m2)';
     [V2,D] = eig(Sb,Sw);
     [¬,ind] = max(abs(diag(D)));
     w = V2(:,ind);
     w = w/norm(w,2);
     v1 = w'*fir;
     v2 = w'*sec;
     if mean(v1)>mean(v2)
         w = -w;
         v1 = -v1;
         v2 = -v2;
     end
     sort1 = sort(v1);
     sort2 = sort(v2);
     t1 = length(sort1);
     t2=1;
     while sort1(t1)>sort2(t2)
         t1 = t1-1;
         t2 = t2+1;
     end
     threshold = (sort1(t1)+sort2(t2))/2;
     err_tr=(sum(v1>threshold)+sum(v2<threshold))/(length(v1)+length(v2));
     % Testing
     te_ind1=find(hiddenlabels==dig1);
     te_ind2=find(hiddenlabels==dig2);
     te_fir = te_X(:,te_ind1(1):te_ind1(end));
     te_sec = te_X(:,te_ind2(1):te_ind2(end));
     TestNum = size(te_fir,2)+size(te_sec,2);
     TestWav = [te_fir te_sec];
     TestMat = U2'*TestWav;
     pval = w'*TestMat;
     ResVec = (pval>threshold);
     errNum = sum(abs(ResVec-[zeros([1,length(te_ind1)]) ...
        ones([1,length(te_ind2)])]));
     sucRate = 1-errNum/TestNum;
 end
```