AMATH 582 Homework 4

Qiangqiang Guo

Due March 10, 2021

Abstract

In this assignment, we will explore the singular value decomposition (SVD) techniques by analyzing the MNIST digit images. Based on the principle components generated by the SVD, we will project the data into the PCA space and build support vector machines and classification tree classifiers to identify individual digits in the MNIST data set.

1 Introduction and Overview

Recognizing handwritten digit numbers is an interesting topic for computer vision. The data set MNIST consists of 60000 training and 10000 testing digit images, which gives us great opportunity to analyze the properties of handwritten digits and explore the techniques of identifying different digit numbers from others. In this assignment, we will apply the singular value decomposition (SVD) method to analyze the principle modes of all digit images in the MNIST data set, and build classifiers to identify different digit numbers.

In the following of this report, we will first introduce the theoretical background used in this assignment (Section 2). Then, we show how to design and implement the analysis algorithm (Section 3). We show the computation results in Section 4 and conclude this report in Section 5.

2 Theoretical Background

The MNIST data set has 60000 training digit images, we want to use SVD to decompose such a high dimensional data matrix into low dimensional matrix. Meanwhile, we can analyze the principle modes behind the data to get better understanding of the data as well as to represent the data using dominant principle modes. In this way, the data can be compressed. The full SVD of any matrix $X \in \mathbb{R}^{m\times n}$ is

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

where $U \in \mathbb{C}^{m\times m}$ is unitary where each column represents one new basis, i.e., U is the new feature space. $V \in \mathbb{C}^{n\times n}$ is unitary, which represents the projection of each original column onto the principle modes. $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal where each diagonal element represents a singular value. Basically, the SVD shows that the operation of matrix X can be decomposed to three steps: first, apply a unitary transformation preserving the unit sphere via $V^{'}$; second, stretch and create an ellipse with principal semi-axes given by Σ; third, rotate the generated hyper-ellipse by the unitary transformation U.

We can arrange the diagonal elements in Σ such that the diagonal elements are non-negative and ordered from largest to smallest, and note the number of significant singular values as rank r. We can then reconstruct the image by using the r (smaller than the original full rank) values:

$$X_{recon} = U * \Sigma(:, 1:r) * V^{'}(:, 1:r) \tag{2}$$

Another key task of this assignment is to develop classifiers to identify different digits. Since the matrix V represents the projection of each digit image onto the principle modes, we can use some columns of $\Sigma * V^{'}$ as the classification features. When it comes to testing, we take the same indexed columns of $U^{'}X_{train}$ as the predictor, and compare the results with the labels of testing data. We will test three different classifiers: linear classifier (LDA), support vector machines (SVM), and classification trees (CT).

The key idea of LDA is to find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data. Please refer to Page 456 for the details of LDA. The key idea of CT is to generate a (binary or non-binary) tree to map the features to labels. Please refer to [1] for details. The key idea of SVM is to create a line or a hyperplane (based on the kernel) to separates the data into classes. Please refer to [2] for details. We do not focus on the mathematical details of these classifiers, and use the corresponding functions directly from MATLAB.

3 Algorithm Development and Implementation

3.1 Read the MNIST data and setup

We first read the MNIST data. The training images data is a $28 \times 28 \times 60000$ matrix, where wach 28×28 data represent a single digit image (from 0 to 9) and there are totally 60000 images. Correspondingly, the label data is a 60000×1 matrix. The test image data is a $28 \times 28 \times 10000$ matrix with a corresponding 10000×1 label matrix. In order to do the SVD analysis, we reshape each image into a column vector (784×1) and aggregate the whole 60000 images to one matrix as the training data, and aggregate another 10000 images to one matrix as the testing data. Each column of the data matrix thus is a different image.

3.2 SVD analysis

We then conduct the SVD using `svd`, and plot the singular value spectrum. Based on this spectrum and the experiment of re-constructing the images, we determine the number of modes that are necessary for good image reconstruction, i.e., the rank r of the digit space. In addition, we project onto three selected V-modes colored by their digit label to better understand the differences between different digit images.

3.3 Build LDA classifier to identify two different digit pairs

We use the second to $n, n = 2, 3, 4, 5, 10, 50$ columns of $\Sigma * V^{'}$ as features to design the LDA classifier and analyze the performance under different number of features. Then, we test all digit pairs to find the two digits that are the most difficult to separate, as well as the two digits that are the most easy to separate. In addition, we pick three digits and build a LDA classifier to identify them.

3.4 Build CT classifier to identify two different digit pairs

Similarly, we build a CT classifier using the training data to identify all 10 digits by letting the MATLAB function `fitctree` to find the optimal hyper parameters. Please refer to the codes appended and the MATLAB official documents for details of `fitctree`.

3.5 Build SVM classifier to identify two different digit pairs

We build a SVM classifier using the MATLAB function `fitcsvm` to identify all digits. `fitcsvm` is originally designed for binary classification, but we can extend such a binary SVM to find multiple class boundaries. The key idea of such a multiple class SVM is to build multiple binary SVMs to evaluate each digit individually, and take the one that has the highest score as the identification results. Please refer to the codes appended and the MATLAB official documents for details of `fitcsvm`. A good SVM implementation requires that the training and testing data are properly scaled. Considering that we use $\Sigma * V^{'}$ as the training data, the largest element of which may be as large as the first singular value. Therefore, we multiply the $\Sigma * V^{'}$ by the inverse of the largest singular value. When it comes to testing, we also multiply the $U^{'} X_{test}$ by the inverse of the largest singular value obtained previously.

4 Computation results

4.1 SVD analysis

The singular value spectrum of the MNIST data is shown in Figure 1. It is shown that the first singular value is the most dominant one, followed by about 100 "sub-dominant" singular values. It is calculated that the first 100 singular values account for 60.45% of the total singular values. To determine the rank r, we try different number of singular values for different images as shown in Figure 2 to Figure 5. It is shown that most digits become clear when $r = 50$. Therefore, we think $r = 50$ is a good choice for the rank.

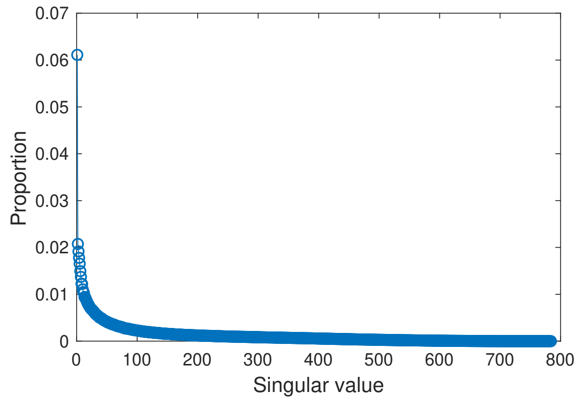

Figure 1: Singular value spectrum

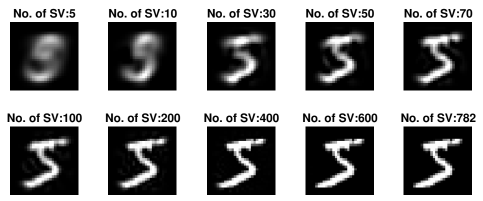

Figure 2: Reconstruct digit 5

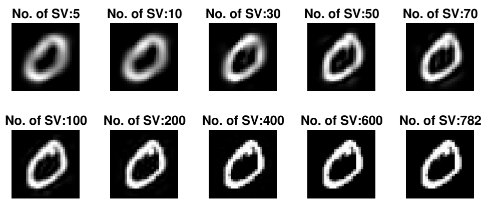

Figure 3: Reconstruct digit 0

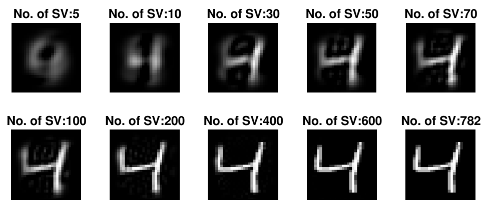

Figure 4: Reconstruct digit 4

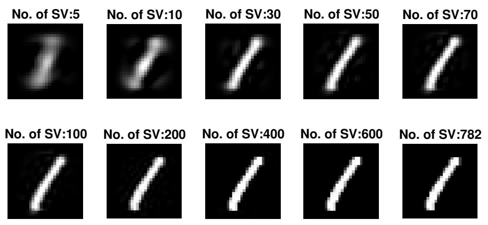

Figure 5: Reconstruct digit 1

The interpretation of the U, Σ and V matrices have been clearly explained in Section 2. To better understand the differences between different digit images, we use a 3D plot to project onto three V-modes (columns 2, 3, and 5) colored by their digit label, which is shown in Figure 6. We can see that different digits gather at different space domains, indicating that we can design classifiers based on the V-modes. However, some different digits are mixed with others, which will introduce difficulty to our classifiers.

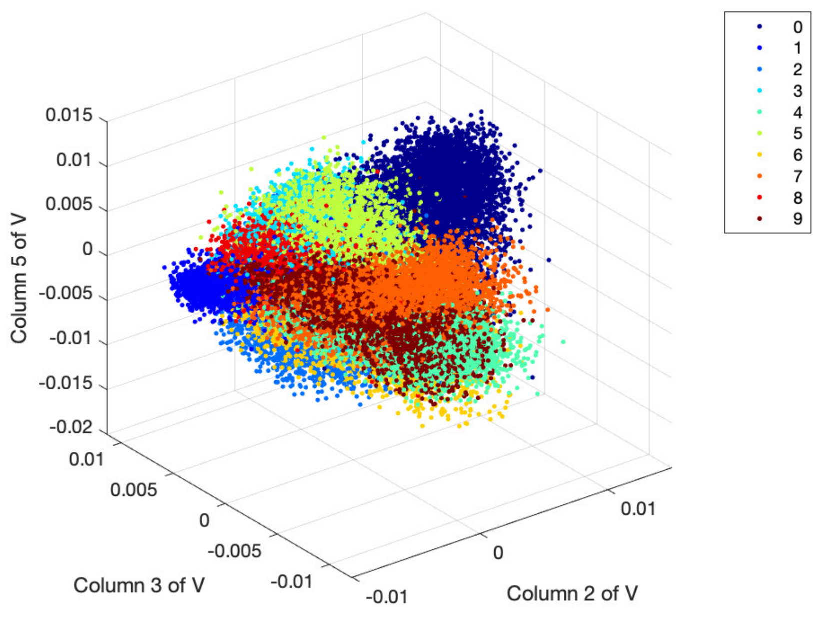

Figure 6: V-modes colored by labels

4.2 LDA classifier

We firstly test the impact of number of V-modes on the classification accuracy. Use number 0 and 4 as an example, for each number of V-modes, we train the classifier using the training data and test the classifier using the test data. The accuracy v.s. number of V-modes is shown in Figure 7. It is shown that when the number of V-modes is 9 (i.e., $n = 10$), the accuracy has reached a very high value (99.13%). Increasing n will increase the computation burden. Thus, we choose $n = 10$, i.e., take the 2,3,4,5,6,7,8,9,10 columns of $\Sigma * V'$ as the features input to the classifier in the following experiments.

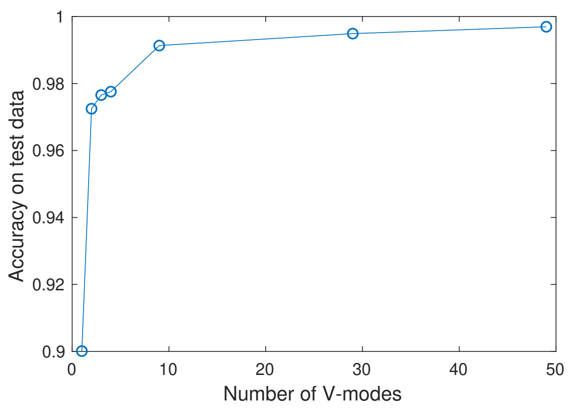

Figure 7: Accuracy under different number of V-modes

We then try to identify all other digit pairs using LDA. The result is shown in Table 1. Note that the

Digits	1	2	3	4	5	6	7	8	9
0	99.81%	97.32%	97.94%	99.13%	94.98%	96.23%	98.95%	97.95%	98.14%
1		97.23%	98.28%	99.34%	99.01%	98.04%	97.36%	94.45%	98.65%
2			95.89%	97.12%	95.74%	92.66%	96.26%	93.27%	96.91%
3				98.64%	89.85%	98.17%	96.37%	89.87%	96.29%
4					97.07%	98.14%	96.37%	96.93%	81.62%
5						94.92%	97.24%	89.66%	95.48%
6							98.64%	97.31%	98.63%
7								94.01%	90.18%
8									93.19%

Table 1: Accuracy for different digit pairs using LDA

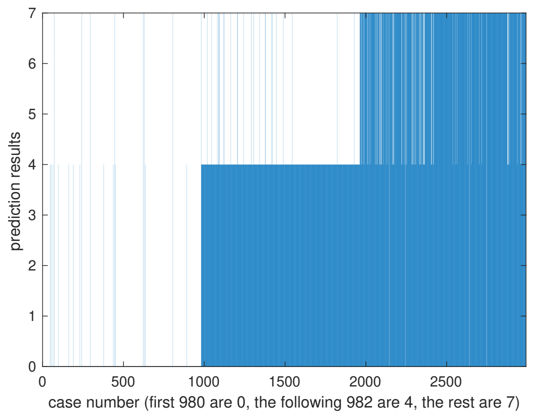

Figure 8: Prediction results on 0,4,7 using LDA

lower triangle is omitted since this table is symmetric. It is shown that the digit pair that is most difficult to separate is 4 and 9 with average accuracy 81.62%, while the digit pair that is most easy to separate is 0 and 1 with average accuracy 99.81%.

We finally train a LDA classifier to identify three digits: $0, 4, 7$, and the prediction result is shown in Figure 8. The accuracy is 96.02%. It indicates that LDA has an acceptable classification accuracy when 3 digits are combined. However, the accuracy slightly decreases compared with the 2 digit pairs.

4.3 CT and SVM classifiers

We train a CT classifier using the training data to identify all 10 digits by letting the MATLAB function `fitctree` to find the optimal hyper parameters. The accuracy on the test data is 83.98%. We also train a SVM classifier using the MATLAB function `fitcsvm` to identify all digits. `fitcsvm` is originally designed for binary classification, but we can extend such a binary SVM to find multiple class boundaries. The accuracy on the test data is 93.32%. (Please refer to the codes appended and the MATLAB official documents for details of `fitctree` and `fitcsvm`.) In addition, we train a LDA classifier to identify all digits. The accuracy on test data is 75.59%. It can be shown that, for the task of identifying all 10 digits, the SVM performs best while the LDA performs the worst.

Finally, we compare the performance between LDA, SVM and decision trees on the hardest (i.e., 4 and 9) and easiest pair of digits (i.e., 0 and 1) to separate. We build new binary CT classifiers and new binary SVM classifiers for those two pairs and show the results in Table 2. It is shown that there is no obvious differences of the performance of LDA, CT, and SVM considering the easiest pair (i.e., 0 and 1). The accuracies of these three methods are all over 99%. For the hardest pair (i.e., 4 and 9), the SVM (93.87%) performs significantly better than LDA (81.62%), while CT performs better than LDA but worse than SVM. Table 2 indicates that SVM is the best classifier considering classification accuracy. However, it should be noted

Pairs	Accuracy		
	LDA	CT	SVM
0 and 1	99.81%	99.76%	99.48%
4 and 9	81.62%	87.69%	93.87%

Table 2: Accuracy of LDA, CT and SVM for the easiest and hardest pairs

that it takes longer time to train the SVM classifier than others during the experiments.

5 Summary and Conclusions

In this assignment, we designed algorithms to explore the SVD and classification techniques on the MNIST data. We reshaped and analyzed the SVD of the training data. Based on the projection on the new basis, we designed LDA, CT and SVM classifiers to identify digits. For LDA, the accuracy of two-digit pair identification ranges from 81.62% to 99.81%, and the accuracy of 10 digits identification is 75.59%. For CT and SVM, the accuracies of 10 digits identification are 83.98% and 93.32%, respectively. For SVM, the accuracy of identifying the two-digit pair that is hardest for LDA is 93.87%, indicating that SVM performs significantly better than LDA. During this assignment, we have learned the power of SVD in identifying key features of data and have understood the LDA, CT and SVM for classification.

References

[1] Oded Z Maimon and Lior Rokach. *Data mining with decision trees: theory and applications.* Vol. 81. World scientific, 2014.

[2] William S Noble. "What is a support vector machine?" In: *Nature biotechnology* 24.12 (2006), pp. 1565–1567.

Appendix A GitHub repository

`https://github.com/Guoqq17/UW-AMATH-582`

Appendix B Key MATLAB Functions

- `B = reshape(A,sz1,...,szN)` reshapes `A` into a `sz1-by-...-by-szN` array where `sz1,...,szN` indicates the size of each dimension.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector `v` on the main diagonal.
- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of texttttm-by-n matrix textttA.
- `imshow(I)` displays the grayscale image `I` in a figure. `imshow` uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.
- `scatter3(X,Y,Z,S,C)` draws each circle with the color specified by `C`. If `C` is a vector with length equal to the length of `X, Y, and Z`, then the values in `C` are linearly mapped to the colors in the current colormap.
- `class = classify(sample,training,group)` classifies each row of the data in sample into one of the groups in training. sample and training must be matrices with the same number of columns. The default `linear` type fits a multivariate normal density to each group, with a pooled estimate of covariance.

- `tree = fitctree(X,Y)` returns a fitted binary classification decision tree based on the input variables contained in matrix `X` and output `Y`. The returned binary tree splits branching nodes based on the values of a column of `X`.
- `tree = fitctree(***,Name,Value)` fits a tree with additional options specified by one or more name-value pair arguments, using any of the previous syntaxes.
- `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.
- `Mdl = fitcsvm(***,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in previous syntaxes.
- `label = predict(Mdl,X)` returns a vector of predicted class labels for the predictor data in the table or matrix `X`, based on the trained, full or compact classification tree `Mdl`.
- `[label,score,node,cnum] = predict(***)` uses any of the input argument in the previous syntaxes and additionally returns: 1) A matrix of classification scores (score) indicating the likelihood that a label comes from a particular class. For classification trees, scores are posterior probabilities. For each observation in X, the predicted class label corresponds to the minimum expected misclassification cost among all classes. 2)A vector of predicted node numbers for the classification (node). 3)A vector of predicted class number for the classification (cnum).

Appendix C MATLAB Codes

```
close all; clear; clc
%% import data
[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');

%% %% part I
[a_train,b_train,c_train]=size(images_train);
[a_test,b_test,c_test]=size(images_test);
data_train = zeros(a_train*b_train, c_train);
data_test = zeros(a_test*b_test, c_test);
for i=1:1:c_train
    data_train(:,i)=reshape(images_train(:,:,i),a_train*b_train, 1);
end
for i=1:1:c_test
    data_test(:,i)=reshape(images_test(:,:,i),a_test*b_test, 1);
end
%% SVD
[U,S,V]=svd(double(data_train),'econ');
proj=(S*V')';
%% singular value spectrum
plot(diag(S)/sum(diag(S)), '-o')
xlabel('Singular value','Fontsize',12)
ylabel('Proportion','Fontsize',12)
%% reconstruct under different v-modes
figure(2)
num_recon = [5, 10, 30, 50, 70, 100, 200, 400, 600, 782];
for i=1:1:length(num_recon)
    img_recon = U*S(:,1:num_recon(i))*V(:,1:num_recon(i)).';
    img=uint8(reshape(img_recon(:,1), a_train, b_train));
    subplot(2,5,i)
    imshow(img)
    title(['No. of SV:', num2str(num_recon(i))])
end
%% V-modes visulization
figure(3);
colormap jet
```

```
for i=0:1:9
    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 of V')
ylabel('Column 3 of V')
zlabel('Column 5 of V')
legend({'0','1','2','3','4','5','6','7','8','9'});

%% %% Part II
%% LDA, 2 digits example, 0 and 4, find the best number of features
num_v=[2,3,4,5,10,30,50];
accuracy_v=zeros(1,length(num_v));
for i=1:1:length(num_v)
    x1_train=proj(labels_train==0,2:num_v(i));
    x2_train=proj(labels_train==4,2:num_v(i));
    [len1,temp]=size(x1_train);
    [len2,temp]=size(x2_train);
    xtrain=[x1_train; x2_train];
    ctrain=[0*ones(len1,1); 4*ones(len2,1)];

    xtest_temp=(U'*data_test)';
    x1_test=xtest_temp(labels_test==0,2:num_v(i));
    x2_test=xtest_temp(labels_test==4,2:num_v(i));
    [len1,temp]=size(x1_test);
    [len2,temp]=size(x2_test);
    xtest=[x1_test; x2_test];
    ctest=[0*ones(len1,1); 4*ones(len2,1)];

    pre=classify(xtest,xtrain,ctrain);

    errorNum=sum(abs(ctest-pre)>0);
    accuracy_v(i)=1-errorNum/length(ctest);
end
figure(4)
plot(num_v-1, accuracy_v, '-o')
xlabel('Number of V-modes','Fontsize',12)
ylabel('Accuracy on test data','Fontsize',12)

%% LDA, compare different digit pairs
accuracy_lda=zeros(10,10);
for i=0:1:8
    for j=i+1:1:9
        x1_train=proj(labels_train==i,2:10);
        x2_train=proj(labels_train==j,2:10);
        [len1,temp]=size(x1_train);
        [len2,temp]=size(x2_train);
        xtrain=[x1_train; x2_train];
        ctrain=[i*ones(len1,1); j*ones(len2,1)];

        xtest_temp=(U'*data_test)';
        x1_test=xtest_temp(labels_test==i,2:10);
        x2_test=xtest_temp(labels_test==j,2:10);
        [len1,temp]=size(x1_test);
        [len2,temp]=size(x2_test);
        xtest=[x1_test; x2_test];
        ctest=[i*ones(len1,1); j*ones(len2,1)];

        pre=classify(xtest,xtrain,ctrain);

        errorNum=sum(abs(ctest-pre)>0);
        accuracy_lda(i+1,j+1)=1-errorNum/length(ctest);
    end
end

%% LDA, 3 digits example, 0, 4 and 7
x1_train=proj(labels_train==0,2:10);
```

```
x2_train=proj(labels_train==4,2:10);
x3_train=proj(labels_train==7,2:10);
[len1,temp]=size(x1_train);
[len2,temp]=size(x2_train);
[len3,temp]=size(x3_train);
xtrain=[x1_train; x2_train; x3_train];
ctrain=[0*ones(len1,1); 4*ones(len2,1); 7*ones(len3,1)];

xtest_temp=(U'*data_test)';
x1_test=xtest_temp(labels_test==0,2:10);
x2_test=xtest_temp(labels_test==4,2:10);
x3_test=xtest_temp(labels_test==7,2:10);
[len1,temp]=size(x1_test);
[len2,temp]=size(x2_test);
[len3,temp]=size(x3_test);
xtest=[x1_test; x2_test; x3_test];
ctest=[0*ones(len1,1); 4*ones(len2,1); 7*ones(len3,1)];

pre=classify(xtest,xtrain,ctrain);

errorNum=sum(abs(ctest-pre)>0);
accuracy_047=1-errorNum/length(ctest)

figure(5)
bar(pre)
xlabel('case number (first 980 are 0, the following 982 are 4, the rest are 7)')
ylabel('prediction results')

%% LDA, all digits
xtrain=proj(:,2:10);
xtest_temp=(U'*data_test)';
xtest=xtest_temp(:,2:10);

pre=classify(xtest,xtrain,labels_train);

errorNum=sum(abs(labels_test-pre)>0);
accuracy_lda=1-errorNum/length(labels_test)

%% CT, all digits
xtrain=proj(:,2:10);
xtest_temp=(U'*data_test)';
xtest=xtest_temp(:,2:10);
Mdl = fitctree(xtrain,labels_train,'OptimizeHyperparameters','auto');

pre=predict(Mdl,xtest);

errorNum=sum(abs(labels_test-pre)>0);
accuracy_ct=1-errorNum/length(labels_test)

%% SVM, all digits
xtrain=proj(:,2:10)/max(max(S));
xtest=xtest_temp(:,2:10)/max(max(S));

SVMModels = cell(10,1);
classes = 0:1:9;
rng(1); % For reproducibility
for j = 1:numel(classes)
    indx = labels_train==classes(j); % Create binary classes for each classifier
    SVMModels{j} = fitcsvm(xtrain,indx,'ClassNames',[false true],'Standardize',true,...
        'KernelFunction','rbf','BoxConstraint',1);
end
for j = 1:numel(classes)
    [~,score] = predict(SVMModels{j},xtest);
    Scores(:,j) = score(:,2); % Second column contains positive-class scores
end

[~,maxScore] = max(Scores,[],2);
errorNum=sum(abs(labels_test+1-maxScore)>0);
```

```
accuracy_svm=1-errorNum/length(labels_test)

%% easiest and hardest pairs
% pair = [0,1]; % easiest
pair = [4,9]; % hardest

x1_train=proj(labels_train==pair(1),2:10);
x2_train=proj(labels_train==pair(2),2:10);
[len1,temp]=size(x1_train);
[len2,temp]=size(x2_train);
xtrain=[x1_train; x2_train];
ctrain=[pair(1)*ones(len1,1); pair(2)*ones(len2,1)];

xtest_temp=(U'*data_test)';
x1_test=xtest_temp(labels_test==pair(1),2:10);
x2_test=xtest_temp(labels_test==pair(2),2:10);
[len1,temp]=size(x1_test);
[len2,temp]=size(x2_test);
xtest=[x1_test; x2_test];
ctest=[pair(1)*ones(len1,1); pair(2)*ones(len2,1)];

% CT
Mdl_ct = fitctree(xtrain,ctrain,'OptimizeHyperparameters','auto');

pre=predict(Mdl_ct,xtest);

errorNum=sum(abs(ctest-pre)>0);
accuracy_ct_2=1-errorNum/length(ctest)

% SVM
rng default
Mdl_svm = fitcsvm(xtrain,ctrain,'Standardize',true,...
        'KernelFunction','rbf','BoxConstraint',1);

pre=predict(Mdl_svm,xtest);

errorNum=sum(abs(ctest-pre)>0);
accuracy_svm_2=1-errorNum/length(ctest)
```