AMATH 482 Homework 4

Tsz Wai Tsui

March 10, 2021

Abstract

Given a MNIST data set containing training and test sets and labels, we aim to first explore the data by perform a Singular Value Decomposition (SVD) and then build a linear classifier (LDA), Support Vector Machines (SVM), and a decision tree to test and compare their performance on training and test data sets.

1 Introduction and Overview

The training and test data of the MNIST data set consists of 60000 and 10000 images of digit 0 to 9 and the corresponding labels respectively. MNIST data is stored in .idx1-ubyte and .idx3-ubyte. We reshaped each image into a column vector, so each column of our data matrix is a different image. Through exploring the data using SVD, which can reduce the dimension of the data matrix, we can examine how many modes are necessary for good image reconstruction. Therefore, we can perform Principal Component Analysis (PCA) and project the training images to a new basis with fewer dimensions to speed up the process of building the classifiers in MATLAB. At last, we compare their success rates of predicting the test labels.

2 Theoretical background

PCA is based on the idea of Singular Value Decomposition (SVD), which is a way of factoring matrices:

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

where matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are unitary matrices and $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is diagonal. The values σ_n on the diagonal of $\mathbf{\Sigma}$ are called the singular values of the matrix $\mathbf{A}$. The vectors u_n which make up the columns of $\mathbf{U}$ are called the left singular vectors of $\mathbf{A}$. The vectors v_n which make up the columns of $\mathbf{V}$ are called the right singular vectors of $\mathbf{A}$.

The goal of our first classifier, LDA is to find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data. To find the right subspace to project onto. First, we calculate the means for each of our groups for each feature, which are denoted as μ_1 and μ_2. Note that these μ are column vectors since they are means across each row. We can then define the between-class scatter matrix:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T. \tag{2}$$

This is a measure of the variance between the groups (between the means). Then we can define the within-class scatter matrix.

$$S_w = \sum_{j=1}^{2}\sum_{x}(x-\mu_j)(x-\mu_j)^T. \tag{3}$$

This is a measure of the variance within each group. The goal is then to find a vector w such that

$$\text{w} = \text{argmax}\frac{w^T S_B w}{w^T S_w w}. \tag{4}$$

This is a tough problem to solve, but the vector **w** maximizes the above quotient is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem

$$S_B w = \lambda S_w w. \tag{5}$$

For classifying between more than two groups, we can make the following changes to the scatter matrices:

$$S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T \tag{6}$$

where μ is the overall mean and μ_j is the mean of each of the $N \geq 3$ groups/ classes. The within-class scatter matrix is

$$S_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T. \tag{7}$$

Then, **w** is found as it was above.

Our second classifier is SVM, which is a supervised learning model. When your data has exactly two classes, an SVM classifies data by finding the best hyperplane that separates all data points of one class from those of the other class by default. For multi-class classification, we trained an error-correcting output codes (ECOC) model.

Our last classifier is decision tree, which is a tree-like supervised learning model. It has two types of entities: nodes and leaves. Leaves represent the final outcomes while node represents the decisions. Each node can split up to two nodes/ leaves.

3 Algorithm Implementation and development

First, we load the data and reshape each image into a column vector so that we can have two image matrix: one for training and one for testing. Then, we perform SVD analysis on the training image. Note that although we are using PCA, we only subtract the matrix by its row mean, but we do not scale the variance of each row to unity since we expect the variance of the pixels in the middle of the frame to be higher than those at the edges. After calculating energies captured by different number of modes, we determine how many modes are necessary for good image reconstruction. We also project the data onto three V-modes on a 3D plot.

After that, we start building classifiers. We first pick two digits (1 and 3) and build a linear classifier (`number_trainer`) that can reasonable identify them. Next, we pick three digits (1, 3, and 5) and try to build a linear classifier (`number_trainer3`) to identify them. We refer to modes as 'features' in the code. Note that we always use the projected training data instead of the raw one when building all classifiers. Then, we plot histograms of the digits values and decisionmaking lines to visualize the data. At last, we calculate the success rates of identifying each pair on the test data to see which pair of the three we chose appear to be the easiest and the most difficult to separate.

Then, we use `fitcecoc` and `fitctree` to build the multi-class SVM and the decision tree model for separating between all ten digits respectively and use `predict` to get the labels predicted by the models on test data. To calculate the success rates, we obtain misclassification rates by `loss` and subtract it from 1.

Before passing projected training data (SV) to `fitcecoc`, we divide it by the max singular value. This bring the data interval between -1 and 1 so that we can keep the data small but efficient to run on the computer. As the model is trained using 60000x100, in which 100 is our chosen mode, we also need to resize the test data from 784 columns to 100 columns. To do that, we multiply the test data by U' on the left to convert the test data to the principal component basis because the matrix U contains the principal components of the training data matrix.

For the decision tree, we do not divide SV by the max singular value since we do not have computational speed problem like we did for SVM. Moreover, doing the division will greatly decrease the success rate.

4 Computational Results

4.1 SVD analysis

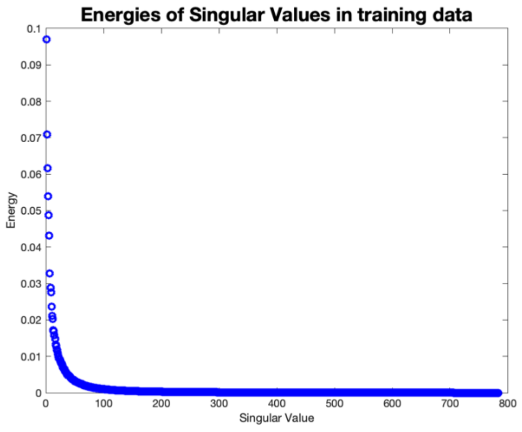


Fig. 1: Energies captured at each singular value

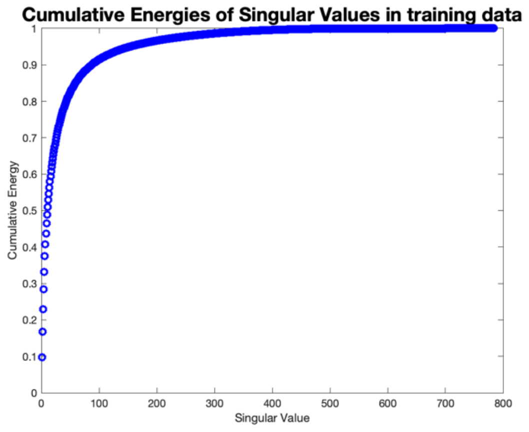


Fig. 2: Cumulative energies capture by singular values

As shown in Fig. 1, first ten singular values capture quite a lot of energies. However, Fig. 2 shows that they only contain around 50% of the total energies, which is not enough for us to construct a good low-rank approximation. Therefore, we calculate the energies captured by rank10, rank50, and rank100 and reconstruct the corresponding images. Fig.3 and Fig.4. show reconstructions of 10 image in rank10 and rank 100. Since the first 100 singular values capture 91.46% of the total energies, and 100 modes is a good reduction from 784 pixels, we set the feature as 100 for all classifiers.

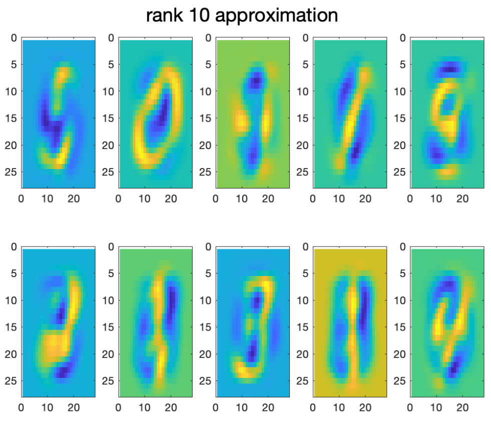


Fig. 3: rank10 approximation of 10 digit images in training data

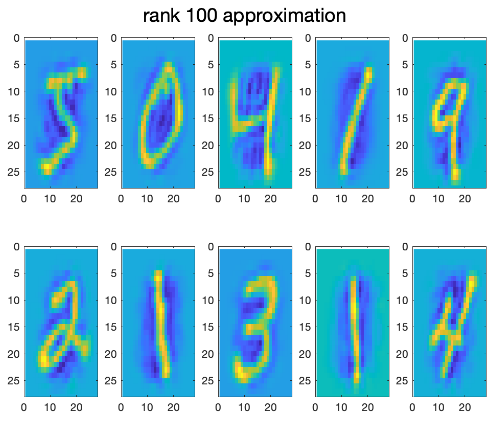


Fig. 4: rank100 approximation of 10 digit images in training data

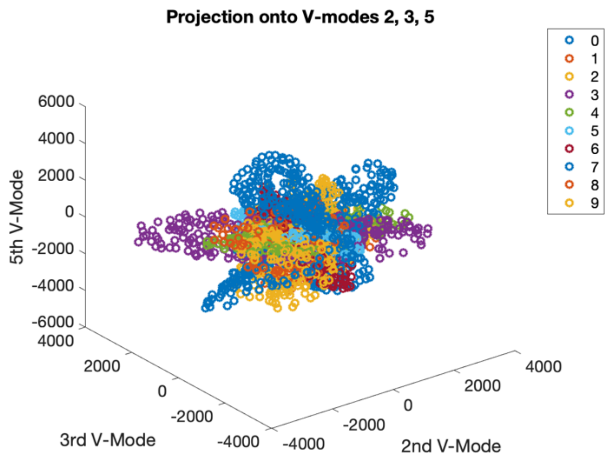


Fig. 5: Projection on V-modes 2, 3, and 5 in a 3D plot

As shown in Fig. 5, we can see some digits are clearly separated from each other while some overlap with each other. Therefore, some pairs of digits may have a higher success rate of separation compared to other pairs.

4.2 LDA

We choose 1 and 3 as our first pair of digits. As shown in the first subplot in Fig. 6, our linear classifier does pretty well on separating between them. After applying it on the test data, the success rate is 99.25%. Then, we apply the LDA for separating two groups on 3, 5 pair and 1, 5 pair, and their success rates are 92.43% and 99.26%. The success rate of LDA on training data is 98.64%, 95.38%, and 99.03% for 1, 3 pair, 3, 5 pair, and 1, 5 pair respectively. We can see that the results lead to the same conclusion that 3, 5 pair is the hardest to separate. It makes sense because their lower parts are very similar. Using the LDA for separating three groups, the distribution of 1, 3, 5 values is shown in Fig. 7, in which separating between 1 and 5 is the easiest, and separating between 3 and 5 is the hardest. The success rate of identifying 1, 3, and 5 is 95.98%.

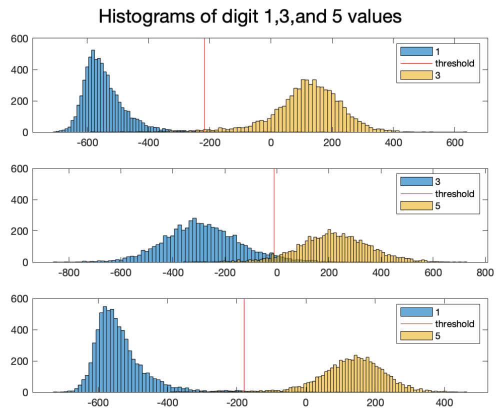


Fig. 6: histograms of digit 1, 3, 5 values using LDA of two groups; 1 vs 3 (Top); 3 vs 5 (Middle); 1 vs 5 (Bottom)

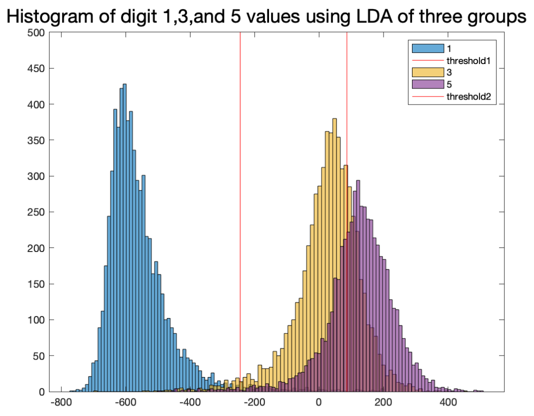
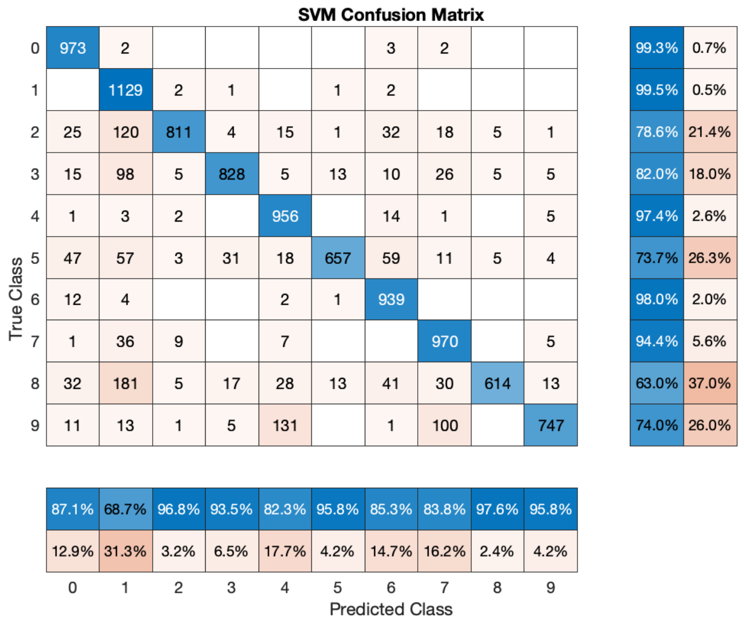


Fig. 7: histogram of digit 1, 3, 5 values using LDA of three groups

4.3 SVM



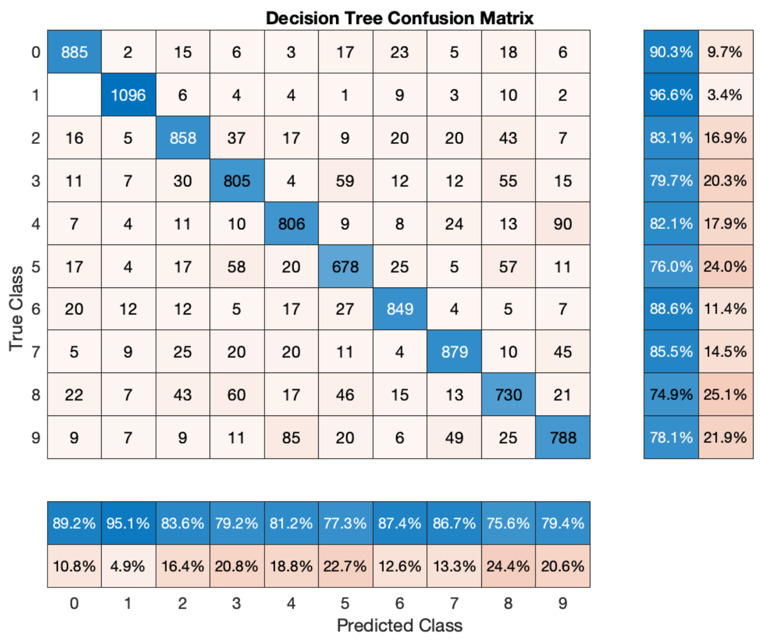
SVM Confusion Matrix

True Class \ Predicted Class	0	1	2	3	4	5	6	7	8	9		
0	973	2					3	2			99.3%	0.7%
1		1129	2	1		1	2				99.5%	0.5%
2	25	120	811	4	15	1	32	18	5	1	78.6%	21.4%
3	15	98	5	828	5	13	10	26	5	5	82.0%	18.0%
4	1	3	2		956		14	1		5	97.4%	2.6%
5	47	57	3	31	18	657	59	11	5	4	73.7%	26.3%
6	12	4			2	1	939				98.0%	2.0%
7	1	36	9		7			970		5	94.4%	5.6%
8	32	181	5	17	28	13	41	30	614	13	63.0%	37.0%
9	11	13	1	5	131		1	100		747	74.0%	26.0%
	87.1%	68.7%	96.8%	93.5%	82.3%	95.8%	85.3%	83.8%	97.6%	95.8%		
	12.9%	31.3%	3.2%	6.5%	17.7%	4.2%	14.7%	16.2%	2.4%	4.2%		

Fig. 8: confusion chart of the prediction of SVM on test data

As shown in Fig. 8, out of all digits, 8 appears to be the most difficult to identify, having 63% success rate, while 1 appears to be the easiest to identify, having 99.5% success rate. The other 2 digits we chose: 3 and 5 have 82% and 73.7% success rate respectively. Similar to the results in LDA, it is harder to separate 3 and 5. The loss/ error rate of SVM on training data is 6.27%, so the success rate is 93.73%. It is higher than the success rate of SVM on test data overall, which is 86.24%.

4.4 Decision tree



Decision Tree Confusion Matrix

True Class	0	1	2	3	4	5	6	7	8	9		
0	885	2	15	6	3	17	23	5	18	6	90.3%	9.7%
1		1096	6	4	4	1	9	3	10	2	96.6%	3.4%
2	16	5	858	37	17	9	20	20	43	7	83.1%	16.9%
3	11	7	30	805	4	59	12	12	55	15	79.7%	20.3%
4	7	4	11	10	806	9	8	24	13	90	82.1%	17.9%
5	17	4	17	58	20	678	25	5	57	11	76.0%	24.0%
6	20	12	12	5	17	27	849	4	5	7	88.6%	11.4%
7	5	9	25	20	20	11	4	879	10	45	85.5%	14.5%
8	22	7	43	60	17	46	15	13	730	21	74.9%	25.1%
9	9	7	9	11	85	20	6	49	25	788	78.1%	21.9%
	89.2%	95.1%	83.6%	79.2%	81.2%	77.3%	87.4%	86.7%	75.6%	79.4%		
	10.8%	4.9%	16.4%	20.8%	18.8%	22.7%	12.6%	13.3%	24.4%	20.6%		
	0	1	2	3	4	5	6	7	8	9		

Predicted Class

Fig. 9: confusion chart of the prediction of decision tree on test data

As shown in Fig. 9, out of all digits, 8 appears to be the most difficult to identify, having 75.6% success rate, while 1 appears to be the easiest to identify, having 96.6% success rate. The other 2 digits we chose: 3 and 5 have 83.1% and 76% success rate respectively. Similar to the results in LDA, it is harder to separate 3 and 5. The loss/ error rate of decision tree on training data is 16.24%, so the success rate is 83.76%. It is almost the same as the success rate of decision tree on test data overall, which is 83.75%.

5 Summary and Conclusions

In conclusion, we can determine the number of modes (100) needed to reconstruct good images by calculating cumulative energies capture by singular values, and proving that by showing the reconstructed images. After that, we can project the training data on the new basis so that it can have fewer dimensions, thus the computational speed of building classifier models is higher. Among the three pairs of digit we picked (1 vs 3, 3 vs 5, 1 vs 5), the pair of 3 and 5 appears to the most difficult to separate, while the pair of 1 and 5 appears to the easiest. Individually, 1 appears to be the easiest to identify in the results of SVM and the decision tree, and 8 appears to be the hardest to identify. All classifiers provide similar results for both the training and the test data, but the success rates of SVM and decision tree on the test data are lower than that on training data. The success rate of LDA also appears to decrease when there are more groups to classify.

Appendix A. MATLAB functions

- `X = zeros(sz1,...,szN)` returns an `sz1`-by-...-by-`szN` array of zeros where `sz1`, ... , `szN` indicate the size of each dimension. For example, `zeros(2,3)` returns a 2-by-3 matrix.
- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of m-by-n matrix `A`:
  - `m > n` — Only the first n columns of `U` are computed, and `S` is n-by-n.
  - `m = n` — `svd(A,'econ')` is equivalent to `svd(A)`.
  - `m < n` — Only the first m columns of `V` are computed, and `S` is m-by-m.
  - The economy-size decomposition removes extra rows or columns of zeros from the diagonal matrix of singular values, `S`, along with the columns in either `U` or `V` that multiply those zeros in the expression `A = U*S*V'`. Removing these zeros and columns can improve execution time and reduce storage requirements without compromising the accuracy of the decomposition.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector `v` on the main diagonal.
- `M = max(A)` returns the maximum elements of an array. If `A` is a vector, then `max(A)` returns the maximum of `A`. If `A` is a matrix, then `max(A)` is a row vector containing the maximum value of each column. If `A` is a multidimensional array, then `max(A)` operates along the first array dimension whose size does not equal 1, treating the elements as vectors. The size of this dimension becomes 1 while the sizes of all other dimensions remain the same. If `A` is an empty array whose first dimension has zero length, then `max(A)` returns an empty array with the same size as `A`.
- `[V,D] = eig(A)` returns diagonal matrix `D` of eigenvalues and matrix `V` whose columns are the corresponding right eigenvectors, so that `A*V = V*D`.
- `B = reshape(A,sz)` reshapes `A` using the size vector, `sz`, to define `size(B)`. For example, `reshape(A,[2,3])` reshapes `A` into a 2-by-3 matrix. `sz` must contain at least 2 elements, and `prod(sz)` must be the same as `numel(A)`.
- `imagesc(C)` displays the data in array `C` as an image that uses the full range of colors in the colormap. Each element of `C` specifies the color for one pixel of the image. The resulting image is an `m`-by-`n` grid of pixels where `m` is the number of rows and `n` is the number of columns in `C`. The row and column indices of the elements determine the centers of the corresponding pixels.
- `N = nnz(X)` returns the number of nonzero elements in matrix `X`.
- `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`.
- `L = kfoldLoss(obj)` returns loss obtained by cross-validated classification model `obj`. For every fold, this method computes classification loss for in-fold observations using a model trained on out-of-fold observations.
- `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`.

- `L = loss(ens,X,Y)` returns the classification error for ensemble `ens` computed using matrix of predictors `X` and true class labels `Y`.
- `Mdl = fitcecoc(X,Y)` returns a trained `ECOC` model using the predictors `X` and the class labels `Y`.
- `err = crossval(criterion,X,y,'Predfun',predfun)` returns a 10-fold cross-validation error estimate for the function `predfun` based on the specified `criterion`, either `'mse'` (mean squared error) or `'msc'` (misclassification rate). The rows of `X` and `y` correspond to observations, and the columns of `X` correspond to predictor variables.
- `confusionchart(trueLabels,predictedLabels)` creates a confusion matrix chart from true labels `trueLabels` and predicted labels `predictedLabels` and returns a `ConfusionMatrixChart` object. The rows of the confusion matrix correspond to the true class and the columns correspond to the predicted class. Diagonal and off-diagonal cells correspond to correctly and incorrectly classified observations, respectively. Use `cm` to modify the confusion matrix chart after it is created.

Appendix B. MATLAB codes

```
% Clean workspace
clear all; close all; clc
[images, labels] = mnist_parse('train-images.idx3-ubyte', ...
    'train-labels.idx1-ubyte');
for k = 1:60000
    im(:,k) = double(reshape(images(:,:,k), [], 1));
    %imshow(im)
end
[test_images, test_labels] = mnist_parse('t10k-images.idx3-ubyte', ...
    't10k-labels.idx1-ubyte');
for k = 1:10000
    test_im(:,k) = double(reshape(test_images(:,:,k), [], 1));
    %imshow(im)
end
[m,n] = size(im);
im_avg = mean(im,2);
im = im - im_avg*ones(1,60000);
[U,S,V] = svd(im, 'econ');
%%
sig = diag(S);
% figure(1)
% plot(sig,'ko','Linewidth',2);
figure(1)
plot(sig.^2/sum(sig.^2),'bo','Linewidth',2);
ylabel('Energy')
xlabel('Singular Value')
title('Energies of Singular Values in training data','Fontsize', 18)
print('Figure 1','-dpng');
figure(2)
plot(cumsum(sig.^2/sum(sig.^2)),'bo','Linewidth',2);
ylabel('Cumulative Energy')
xlabel('Singular Value')
title('Cumulative Energies of Singular Values in training data','Fontsize',
18)
print('Figure 2','-dpng');
```

```matlab
energy1 = sig(1)^2/sum(sig.^2);
energy2 = sum(sig(1:2).^2)/sum(sig.^2);
energy3 = sum(sig(1:3).^2)/sum(sig.^2);
energy5 = sum(sig(1:5).^2)/sum(sig.^2);
energy10 = sum(sig(1:10).^2)/sum(sig.^2);
energy30 = sum(sig(1:30).^2)/sum(sig.^2);
energy50 = sum(sig(1:50).^2)/sum(sig.^2);
energy100 = sum(sig(1:100).^2)/sum(sig.^2);
%%
rank_10 = U(:,1:10)*S(1:10,1:10)*V(:,1:10)';
figure(3)
for j=1:10
    ree = reshape(rank_10(:,j),28,28);
    subplot(2,5,j)
    imagesc(ree), axis([0 28 0 28])
end
sgtitle('rank 10 approximation','Fontsize', 18)
print('Figure 3','-dpng');
%%
% rank_50 = U(:,1:50)*S(1:50,1:50)*V(:,1:50)';
% figure(4)
% for j=1:10
%     ree = reshape(rank_50(:,j),28,28);
%     subplot(2,5,j)
%     imagesc(ree), axis([0 28 0 28])
% end
% sgtitle('rank 50 approximation','Fontsize', 18)
% print('Figure 4','-dpng');
rank_100 = U(:,1:100)*S(1:100,1:100)*V(:,1:100)';
figure(5)
for j=1:10
    ree = reshape(rank_100(:,j),28,28);
    subplot(2,5,j)
    imagesc(ree), axis([0 28 0 28])
end
sgtitle('rank 100 approximation','Fontsize', 18)
print('Figure 4','-dpng');
%% Projection onto 3 V-modes
figure(6)
for label=0:9
    label_indices = find(labels == label);
    plot3(im(:,label_indices)*V(label_indices, 2), ...
        im(:,label_indices)*V(label_indices, 3), ...
        im(:,label_indices)*V(label_indices, 5),...
        'o', 'DisplayName', sprintf('%i',label), 'Linewidth', 2)
    hold on
end
xlabel('2nd V-Mode'), ylabel('3rd V-Mode'), zlabel('5th V-Mode')
title('Projection onto V-modes 2, 3, 5','Fontsize', 18)
legend
set(gca,'Fontsize', 14)
print('Figure 5','-dpng');
%% LDA
feature = 100;
nOfNum = [];
test_nOfNum = [];
% count how many pictures of each number
for k = 1:10
```

```matlab
    nOfNum(k) = nnz(labels(:)==(k-1));
    im_{k} = zeros(784, nOfNum(k));
    im_{k} = im(:,find(labels==(k-1)));
    test_nOfNum(k) = nnz(test_labels(:)==(k-1));
    test_im_{k} = zeros(784, test_nOfNum(k));
    test_im_{k} = test_im(:,find(test_labels==(k-1)));
end
%% 1 vs 3
[threshold1,U1,S1,V1,w1,s1,s2] = number_trainer(nOfNum(2), nOfNum(4),
im_{1,2}, im_{1,4}, feature);
figure(6)
subplot(3,1,1)
histogram(s1,100); hold on, plot([threshold1 threshold1], [0 600],'r')
histogram(s2,100);
legend('1','threshold','3','Location','northeast');

% 3 vs 5
[threshold2,U2,S2,V2,w2,s3,s4] = number_trainer(nOfNum(4), nOfNum(6),
im_{1,4}, im_{1,6}, feature);
subplot(3,1,2)
histogram(s3,100); hold on, plot([threshold2 threshold2], [0 600],'r')
histogram(s4,100);
legend('3','threshold','5','Location','northeast');

% 1 vs 5
[threshold3,U3,S3,V3,w3,s5,s6] = number_trainer(nOfNum(2), nOfNum(6),
im_{1,2}, im_{1,6}, feature);
subplot(3,1,3)
histogram(s5,100); hold on, plot([threshold3 threshold3], [0 600],'r')
histogram(s6,100);
legend('1','threshold','5','Location','northeast');
sgtitle('Histograms of digit 1,3,and 5 values','Fontsize', 18);
print('Figure 6','-dpng');

%% training data
tr_Mat = [im_{1,2} im_{1,4}];
trMat = U1'*tr_Mat;
pval = w1'*trMat;
hiddenLabels = zeros(1, size(tr_Mat,2));
hiddenLabels(nOfNum(2)+1:nOfNum(2)+nOfNum(4)) = 1;
testNum = size(tr_Mat,2);

ResVec = (pval > threshold1);
err_lda = abs(ResVec-hiddenLabels);
errNum = sum(err_lda);
tr_sucRate_13 = 1 - errNum/testNum;

tr_Mat = [im_{1,4} im_{1,6}];
trMat = U2'*tr_Mat;
pval = w2'*trMat;
hiddenLabels = zeros(1, size(tr_Mat,2));
hiddenLabels(nOfNum(4)+1:nOfNum(4)+nOfNum(6)) = 1;
testNum = size(tr_Mat,2);

ResVec = (pval > threshold2);
err_lda = abs(ResVec-hiddenLabels);
errNum = sum(err_lda);
```

```
tr_sucRate_35 = 1 - errNum/testNum;

tr_Mat = [im_{1,2} im_{1,6}];
trMat = U3'*tr_Mat;
pval = w3'*trMat;
hiddenLabels = zeros(1, size(tr_Mat,2));
hiddenLabels(nOfNum(2)+1:nOfNum(2)+nOfNum(6)) = 1;
testNum = size(tr_Mat,2);

ResVec = (pval > threshold3);
err_lda = abs(ResVec-hiddenLabels);
errNum = sum(err_lda);
tr_sucRate_15 = 1 - errNum/testNum;
%% test data

Mat = [test_im_{1,2} test_im_{1,4}];
testMat = U1'* Mat;
pval = w1'*testMat;
hiddenLabels = zeros(1, size(Mat,2));
hiddenLabels(test_nOfNum(2)+1:test_nOfNum(2)+test_nOfNum(4)) = 1;
testNum = size(Mat,2);

ResVec = (pval > threshold1);
err_lda = abs(ResVec-hiddenLabels);
errNum = sum(err_lda);
sucRate_13 = 1 - errNum/testNum;

Mat = [test_im_{1,4} test_im_{1,6}];
testMat = U2'* Mat;
pval = w2'*testMat;
hiddenLabels = zeros(1, size(Mat,2));
hiddenLabels(test_nOfNum(4)+1:test_nOfNum(4)+test_nOfNum(6)) = 1;
testNum = size(Mat,2);

ResVec = (pval > threshold2);
err_lda = abs(ResVec-hiddenLabels);
errNum = sum(err_lda);
sucRate_35 = 1 - errNum/testNum;

Mat = [test_im_{1,2} test_im_{1,6}];
testMat = U3'* Mat;
pval = w3'*testMat;
hiddenLabels = zeros(1, size(Mat,2));
hiddenLabels(test_nOfNum(2)+1:test_nOfNum(2)+test_nOfNum(6)) = 1;
testNum = size(Mat,2);

ResVec = (pval > threshold3);
err_lda = abs(ResVec-hiddenLabels);
errNum = sum(err_lda);
sucRate_15 = 1 - errNum/testNum;

%% 1 vs 3 vs 5
[t1,t2,u,s,v,w,ss1,ss2,ss3] = number_trainer3(nOfNum(2), nOfNum(4),
nOfNum(6), ...
    im_{1,2}, im_{1,4},im_{1,6},feature);
figure(7)
```

```
histogram(ss1,100); hold on,plot([t1 t1], [0 500],'r')
histogram(ss2,100);
histogram(ss3,100); hold on,plot([t2 t2], [0 500],'r')
sgtitle('Histogram of digit 1,3,and 5 values using LDA of three groups',...
    'Fontsize', 18);
legend('1','threshold1','3','5','threshold2','Location','northeast');
print('Figure 7','-dpng');

Mat = [test_im_{1,2} test_im_{1,4} test_im_{1,6}];
testMat = u'* Mat;
pval = w'*testMat;
hiddenLabels = zeros(1, size(Mat,2));
hiddenLabels(test_nOfNum(2)+1:test_nOfNum(2)+test_nOfNum(4)) = 1;
hiddenLabels(test_nOfNum(2)+test_nOfNum(4)+1: ...
    test_nOfNum(2)+test_nOfNum(4)+test_nOfNum(6)) = 1;
testNum = size(Mat,2);

ResVec = (pval > (t1+t2)/2);
err_lda = abs(ResVec-hiddenLabels);
errNum = sum(err_lda);
sucRate_135 = 1 - errNum/testNum;

%% decision tree 1 vs 3
project1 = S1*V1';
labels1 = [ones(1,nOfNum(2)), ones(1,nOfNum(4))*3];
tree_13=fitctree(project1',labels1,'CrossVal','on');
view(tree_13.Trained{1},'Mode','graph');
classError_13 = kfoldLoss(tree_13, 'mode', 'individual');
[~, A] = min(classError_13);
predict_labels_13 = predict(tree_13.Trained{A},[test_im_{1,2}
test_im_{1,4}]');
expected_label_13 = [ones(1,test_nOfNum(2)) ones(1,test_nOfNum(4))*3];
T_13 = loss(tree_13.Trained{A},[test_im_{1,2}
test_im_{1,4}]',expected_label_13');

%% decision tree 3 vs 5
project2 = S2*V2';
labels2 = [ones(1,nOfNum(4))*3, ones(1,nOfNum(6))*5];
tree_35=fitctree(project2',labels2,'CrossVal','on');
view(tree_35.Trained{1},'Mode','graph');
classError_35 = kfoldLoss(tree_35, 'mode', 'individual');
[~, A] = min(classError_35);
predict_labels_35 = predict(tree_35.Trained{A},[test_im_{1,4}
test_im_{1,6}]');
expected_label_35 = [ones(1,test_nOfNum(4))*3 ones(1,test_nOfNum(6))*5];
T_35 = loss(tree_35.Trained{A},[test_im_{1,4}
test_im_{1,6}]',expected_label_35');

%% decision tree 1 vs 5
project3 = S3*V3';
labels3 = [ones(1,nOfNum(2)), ones(1,nOfNum(6))*5];
tree_15=fitctree(project3',labels3,'CrossVal','on');
view(tree_15.Trained{1},'Mode','graph');
classError_15 = kfoldLoss(tree_15, 'mode', 'individual');
[~, A] = min(classError_15);
predict_labels_15 = predict(tree_15.Trained{A},[test_im_{1,2}
test_im_{1,6}]');
```

```matlab
expected_label_15 = [ones(1,test_nOfNum(2)) ones(1,test_nOfNum(6))*5];
T_15 = loss(tree_15.Trained{A},[test_im_{1,2}
test_im_{1,6}]',expected_label_15');

%% SVM All digits
SV = S(1:100,1:100)*V(:,1:100)';
xtrain_all_scaled_ = SV./max(SV(:));
Mdl_all_scaled_ = fitcecoc(xtrain_all_scaled_',labels');

%% test
test_all_ = test_im - im_avg*ones(1,10000);
proj_test = U(:,1:100)'*test_all_;
predicted_labels_all_ = predict(Mdl_all_scaled_,proj_test');
%%
expected_labels_all_ = test_labels;
countErr_all_ = 0;
for k = 1:length(expected_labels_all_)
    if expected_labels_all_(k) ~= predicted_labels_all_(k)
        countErr_all_ = countErr_all_ + 1;
    end
end
SVM_sucRate = 1 - countErr_all_/length(test_labels);
%%
cross_Mdl_all_scaled_ = crossval(Mdl_all_scaled_);
CECOC_Loss = kfoldLoss(cross_Mdl_all_scaled_);

%%
figure(8)
cm = confusionchart(test_labels, predicted_labels_all_);
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
cm.Title = 'SVM Confusion Matrix';
print('Figure 8','-dpng');
%%
SVM_loss = loss(Mdl_all_scaled_,proj_test',expected_labels_all_');

%% Decision tree for all digits
tree_all=fitctree(SV',labels','CrossVal','on');
view(tree_all.Trained{1},'Mode','graph');
classError_all = kfoldLoss(tree_all, 'mode', 'individual');
[~, A] = min(classError_all);
t_predict_labels = predict(tree_all.Trained{A},proj_test');
t_expected_label = test_labels;
TL_all = loss(tree_all.Trained{A},proj_test',t_expected_label');
%%
figure(9)
cmtree = confusionchart(test_labels,t_predict_labels);
cmtree.ColumnSummary = 'column-normalized';
cmtree.RowSummary = 'row-normalized';
cmtree.Title = 'Decision Tree Confusion Matrix';
print('Figure 9','-dpng');
```