AMATH 482 Homework 4

Seoyoung Cho

March 10, 2021

Abstract

Classification is one method of image recognition in which the objects depicted in images are classified based on their distinct edge patterns. Among the wide variety of techniques of classification, we focus on edge detection using linear discrimination analysis (LDA) for this assignment. Given the handwritten digit images, we seek to build a linear discriminant classifier that detects the correct digit in the image after analyzing its edge patterns.

1 Introduction and Overview

In this assignment, we are given the MNIST data set of grayscale images depicting the handwritten single digits between 0 and 9. There is a total of 4 data: 60,000 of 28x28 pixel digit images for training classifiers, corresponding training digit labels, 10,000 images for the testing classifier, and the corresponding test digit labels. Given the data, the assignment is divided into two big parts:

1. Perform Principal Component Analysis (PCA) of the image data. After performing PCA, we discuss the nth mode approximation for image reconstruction and the interpretation of all the decomposed components U, Σ, and V.
2. Perform Linear Discriminant Analysis (LDA) for building classifiers for two and three digits. After, we discuss the performance of the two-digit classifier and compare it with different types called decision trees and Support Vector Machine (SVM).

Using the LDA, we will be catching the distinct patterns of each digit edge and be able to test the performance of the classifier with test image data. Then we will explore different types of classifiers and compare accuracy. Upon completing the given tasks, we seek to build a solid understanding of image recognition methods. This paper will guide through the background topics and the implementation details.

2 Theoretical Background

This section explains the mathematical concepts required to proceed with the task.

2.1 Edge Detection

Given the images of digits as shown in figure 6, how can we detect which digit is which? Since the shapes of digits are unique, the easiest way to detect them will be to trace the edges. Edge detection is simply a method to distinguish the images by analyzing the unique patterns of the edges. Hence, the main focus while performing the edge detection is the difference of the edges of all images or their variance. This essentially boils down to performing Principal Component Analysis (PCA). PCA is simply a way to re-express the original data matrix to a new basis based on the variance of the variables. PCA decomposes the data matrix as:

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

where the columns of U are the principal component modes based on the variances of variables, diagonals of Σ measure the relative importance of columns of U, V, and the columns of V represent the coefficients of the linear combination of U to reconstruct the original matrix. Recall our training data matrix X as represented in eq 2 in which each column containing the digit images. The row of the matrix will be the pixels of the images starting from the top-left corner to the bottom right.

$$X = \begin{bmatrix} | & | & & | \\ | & | & & | \\ x_1 & x_2 & \cdots & x_{60000} \\ | & | & & | \\ | & | & & | \end{bmatrix} \tag{2}$$

To proceed with PCA, we take an additional step of de-meaning our data matrix. We subtract the mean of each row to get zero centered rows. In this case, the mean of each row represents the average pixels across images. Taking out the means of each pixel help enhance the process of analyzing the variances across images. Henceforth, we consider our data matrix X as the matrix from eq 2 with being zero centered. Performing PCA on our data matrix ultimately gives the principal components that capture how each digit differs based on its edges. Then we can use Linear Discriminant Analysis (LDA) to set up a guideline that determines which digit is which.

2.2 Linear Discriminant Analysis (LDA)

As briefly mentioned above, the overall goal of performing LDA is to find a threshold value that distinguishes the digits. The general idea is depicted in the left of figure 7 in Appendix C where there is a linear line that separates two variables. Hence the linear line is the classifier. Another interpretation of LDA is illustrated in the right of figure 7. The figure shows two cases of projecting data onto new bases. On the left, the projection of two clusters is mixed whereas the projection on the right completely separates the clusters. Further, the mean μ_1 and μ_2 of two clusters are close to each other on the left and further apart on the right. Likewise, the goal of LDA is to find the right projection that maximizes the distance of between-class data while minimizing that of in-class. In this section, we specifically focus on two-class LDA. For two-class LDA, the "right" projection w is defined as below:

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

where S_B is the between-class scatter matrix measuring the variance between the means and S_w is the within-class scatter matrix measuring the variance of each class. They take the form as below:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{4}$$

$$S_w = \sum_{j=1}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{5}$$

It turns out that the above variables are related as:

$$S_B w = \lambda S_w w \tag{6}$$

$$\Rightarrow w = argmax\frac{w^T \lambda S_w w}{w^T S_w w} = argmax\lambda \tag{7}$$

where λ is the largest eigenvalue. Thus, w is the eigenvector corresponding to λ and is the projection that maximizes the distance between two groups. Then we can train the classifier by projecting the training data onto w subspace and determining the threshold value. After training, we are ready to test the performance with the testing data. Since decisions are decided in w subspace, project the test data onto the same subspace. Then we can classify the test data based on the decisions on training data.

2.3 Other classifiers

2.3.1 Decision Tree classifier

As can be inferred from the name "Decision Tree", it is a tree-structured classifier where the nodes represent the class and the branches represent the threshold values that lead to each class. An example is delivered in figure 8 in Appendix C. In the left figure, the first decision node represents the most prominent division, dividing the training data into two subgroups. The process continues recursively until we get leaf nodes that represent decisions. The figure on the right hand gives a run-through of the training and classifying process of a decision tree. As illustrated, generating a decision tree using training data decides the threshold values and classifies the data accordingly. Once the tree is generated, we can test using the test data which will give the decision outcomes based on the training data. In our case, the outcome would be the labels of the digits in test data decided after comparing with the training set.

2.3.2 Support Vector Machine (SVM)

Another classifying algorithm that we use is called the Support Vector Machine (SVM). SVM shares some similarity with linear discriminant analysis with how it's dividing the data using a linear classification in 2D space. However, unlike the LDA that linear division is applied in every division, SVM divides using the best hyperplane according to space dimension. In other words, a line in 2D space and a plane in 3D. Another difference is what defining the "best" hyperplane. Recall with LDA, the best decision was represented by the ratio of variances between and within the clusters. For SVM, the best decision is determined by its distance to the closest elements in each cluster. The decision is illustrated in the figure 9 in Appendix C for better understanding. Just like all the other classifying algorithms, the SVM determines the decision hyperplanes using the training data and classifies the test data according to the result class.

3 Algorithm Implementation and Development

This section explains the implementation process in MATLAB. For more details of MATLAB functions, refer to Appendix A. The initial steps are below:

1. Load and parse all training images, training labels, testing images, and testing labels using the given `mnist_parse` function. This loads in 28 pixels by 28 pixels by 60000 training digit images, by 10000 testing digit images, and string vectors of the digit labels corresponding to the order of the images.
2. Reshape the 3-dimensional image matrices into 2-dimensional data matrix with each column representing the images. Hence, we'll end up with 784 by 60000 training matrix and 784 by 10000 testing matrix.

3.1 SVD/PCA Analysis

1. Subtract the mean of each row from the training data matrix. This demeans the matrix so the mean of each pixel to zero.
2. Perform SVD on the zero-centered data matrix. This decomposes the training data matrix into $U * \Sigma * V^T$.
3. Plot the singular values and the cumulative energy. This specifies the range of modes that we can reduce the dimension of the original data matrix to.
4. Plot the projection of the data onto selected principal components. This is done by plotting the corresponding columns of V. This will show the clusters of each digit.

3.2 Linear Discriminant Analysis on two digits

1. Find all the images of two chosen digits in the SVD reduced training data and store them as separate vectors.
2. Calculate the within class variances, refer to equation 5.
3. Calculate the between class variance, refer to equation 4.
4. Find the projection w by performing eigen-decomposition to the within and between class variance matrices. Recall w is the eigenvector corresponding to the largest eigenvalue. We normalize the w by dividing by its 2nd norm, so the sum of all the elements equal to 1.

5. Project both digit vectors onto w. For consistency, compare the mean of projection of two digits and keep the first digit data to be always below the second digit. This enhances the process of determining the threshold.

6. Determine the threshold value that distinguishes the two digits. This is done by first sorting the digit projection vectors in ascending order. Then compare the largest of the first digit value and the smallest of the second digit value. If the first digit value is greater than the second digit value, move one in each and continue comparing until the first value is less than the second value. Threshold value is the mean of the two digit values.

7. Define the above process as a function for the future use. The function will take in the 2 digit train data and number of mode for dimensionality reduction. Then the function will return the U, Σ, V, threshold, w, and sorted projections of the two digits. This process essentially trains the classifier.

8. After training the classifier, we can test the performance using the test data. First demean the test data. Then choose two digits and project onto returned PCA modes, the U matrix. This reduces the dimension of the test data to capture only PCA modes. Since the threshold is determined in w subspace, multiply the test data with w to project the test data onto w subspace.

9. Distinguish the test digits according to the threshold value. If the digit value is less than the threshold, the digit is classified as the first training digit and vice versa.

10. For implementing three digits LDA classifier, the general approach is the same as the above steps 1-9. Only difference is that S_w and S_b are defined using the following equations:

$$S_w = \sum_{j=1}^{N} \sum_{x} (x - \mu_j)(x - \mu_j)^T \qquad S_b = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T$$

    where μ is the overall mean. For convenience, we use the function `classify` for three digits LDA.

3.3 Other classifiers

For the other classifiers (decision tree and SVM), we use the built-in MATLAB function `fitctree` and `fitcsvm` respectively. `fitcecoc` function is used for performing multi-class SVM classifier. These functions generally take the projection of training data, training labels, projection of test data onto training subspace, and return the trained models. We can then determine the accuracy by comparing it with the original test labels.

4 Computational Results

4.1 SVD/PCA Analysis

The first part of this assignment is to perform SVD or PCA analysis of the digit image data. The columns of U are the principal components of the data capturing the most prominent feature across all digits. The diagonals of Σ measure the importance of the corresponding columns of U to all digits. The columns of V show the projection of all the digit images on the corresponding columns of U. After performing SVD, we plot the singular value and cumulative energy to determine the modes for good image reconstruction. Since both the train and test data will be projected onto training principal components, we only show the plot for the training data. The figure 1 delivers the singular value and the cumulative energy spectrum. We determine the modes that their cumulative energy adds up to 90 percent. This is approximately 80 modes, but due to time restriction when performing SVM in MATLAB, we use 30 modes when comparing the accuracy of all classifiers. We still use 80 modes for testing two and three digits LDA classifiers. Figure 2 shows the projection of training digits onto the first three V-modes at different angles. All ten digits are labeled in different colors and we can see different clusters of the digits. According to the figure, digits 4 and 9 are particularly mixed. We anticipate that those digits may be hard to classify using LDA.

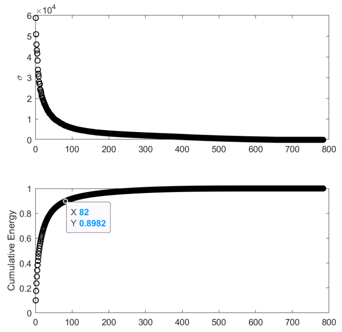


Figure 1: σ and cumulative energy spectrum of train data

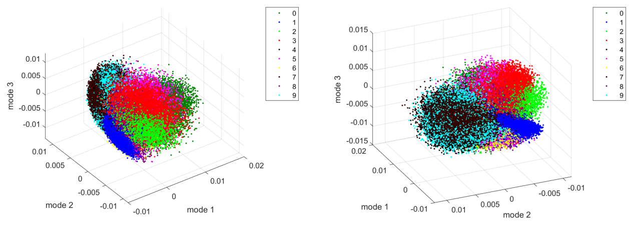


Figure 2: Projections of the first three V-modes

4.2 Two and Three digits LDA

First, we have chosen digits 0 and 1 to test the two-digit linear classifier. Using 60 principal component modes, the figure 3 shows the result of testing the linear classifier with 2 digits 0 and 1. The left figure shows the linear projection of the two training digits. 0s are the blue plots on the left side and 1s are the red plots on the right side. There are still some overlaps, but given the total number of images is 60000, the accuracy is still very high with 99.76 percent. The right figure shows the histogram of digits 0 and 1 with a red threshold line. We can again see that most of the digit 0s are on the negative side and most of the digit 1s are on the positive side based on the threshold line. These results convey that the two-digit linear classifier is working well with digits 0 and 1. For three-digit linear classifiers, we choose digits 0, 1, and 2. In this case, we test using the built-in `classify` function in MATLAB for classification accuracy. The function takes in the testing digits, training digits, training labels and returns the classified labels for the testing data. Again using 80 modes, the accuracy of the classifier was 99.55 percent. This is very high and conveys that using a built-in function also works well with digits 0, 1, and 2.

4.3 Decision tree and SVM classifiers

Continuing from LDA, we test the performance of decision tree and SVM classifiers, in particular on how well they separate between all ten digits. This specifically asks for the multi-class classification and we use built-in functions `fitctree` for decision tree and `fitcecoc` for SVM. Again, these functions take in the projected training data, training labels, and return the classified model of test data. Then we can use the built-in `predict` function to get the classified labels of the test data. Note that upon performing the SVM multi-class classifier, we encountered a run time issue. Hence, we reduce the dimension of data to the first 30 principal modes and the same with the decision tree for fairness. After comparing with the original test digit labels, we ended up with 85.22 percent accuracy with the decision tree and 44.65 percent accuracy with SVM. Due to the reduction of principal components, we see a general decrease in performance accuracy. According to the result, the accuracy of the multi-class SVM classifier is fairly low comparing to that of the decision tree.

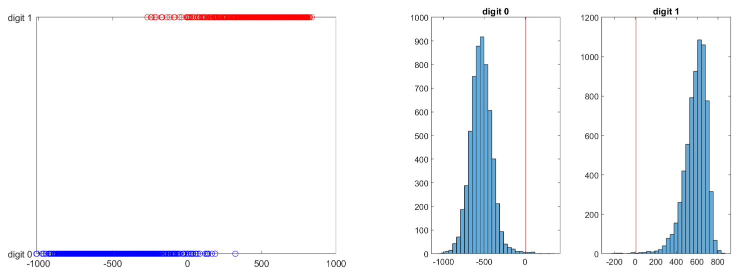
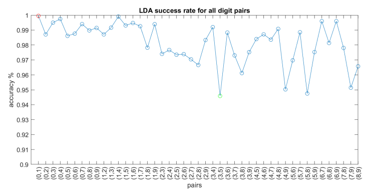
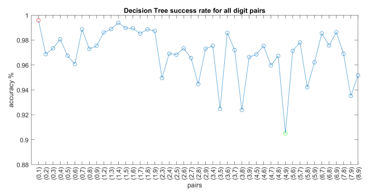


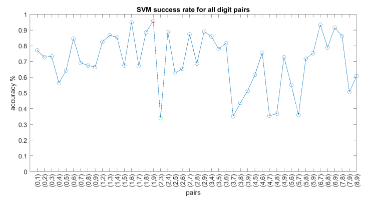
Figure 3: Result for testing 0 and 1 digit with linear classifier



(a) Accuracy rate of linear classifier



(b) Accuracy rate of DT classifier



(c) Accuracy rate of SVM classifier

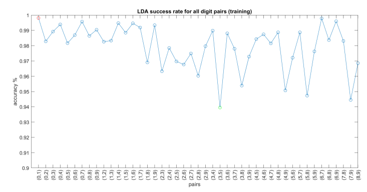
Figure 4: Plot of accuracy rate of every train digit pairs

4.4 Comparison of linear, decision tree, and SVM binary classifiers

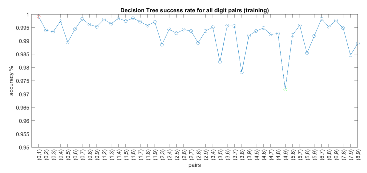
The last part of the assignment is to compare the performance of all classifiers on every possible digit pairs. There are total of 45 digit pairs starting from $(0, 1)$ to $(8, 9)$. Again due to run-time restriction with SVM implementation, we use the first 30 modes for all classifiers to consider fairness. Figure 4 show the plot of accuracy rate on all test digit pairs using linear, decision tree abbreviated as DT, and SVM classifiers. For all three plots, the red plots represent the pair with the highest classification accuracy and the green plots represent the pair with the lowest classification accuracy. In general, we see that the accuracy of the linear classifier is higher than others and that of SVM is the lowest with the greatest variance. For linear and decision tree classifiers, the digits 0 and 1 are classified most accurately. For the SVM classifier, the digits 1 and 9 are classified most accurately. Having the most accuracy essentially mean that the pairs (0, 1) for LDA, decision tree and (1, 9) for SVM were the easiest to classify. The pairs that were classified least accurately with linear classifier is (3, 5), with a decision tree is (4, 9), and with SVM is (2, 3). From the figure 2, we observed that digits 4 and 9 are particularly mixed and interestingly, its effect was only shown with the decision tree. To validate the derived test results, we compare the results after testing with training data. The figure 5 shows the accuracy rate of all training digit pairs. We see that in general, the accuracy rate of the decision tree and SVM has increased compared to that of test pairs. The linear classifier hasn't changed much either, also the minimum and maximum accuracy pairs remained. However, unlike the linear and decision tree classifiers that have the same minimum and maximum accuracy pairs, that of SVM has changed to (0, 1) and (3, 7) respectively. We cautiously anticipate the SVM may be more vulnerable with the dimensionality reduction. Overall, the decision tree and linear classifiers showed stability with their performance and all the classifiers performed well enough.

5 Summary and Conclusions

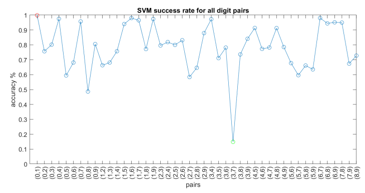
To summarize, we implemented and compared performances of different classification methods for classifying the given hand-written digit images in this assignment. Analyzing the computational results, we showed that the classification methods worked successfully even by only considering the edge of images. Further, the PCA implementation efficiently reduced the dimensionality of the image data matrices.



(a) Accuracy rate of linear classifier



(b) Accuracy rate of DT classifier



(c) Accuracy rate of SVM classifier

Figure 5: Plot of accuracy rate of every train digit pairs

References

[1] “Linear Methods for Classification.” Linear Methods for Classification - Mlpy v3.3.0 Documentation, Mlpy Developers, 2011, mlpy.sourceforge.net/docs/3.3/lin_class.html.

[2] “Machine Learning Decision Tree Classification Algorithm - Javatpoint.” Www.javatpoint.com, www.javatpoint.com/machine-learning-decision-tree-classification-algorithm.

[3] Navlani, Avinash. “Decision Tree Classification in Python.” DataCamp Community, 28 Dec. 2018, www.datacamp.com/community/tutorials/decision-tree-classification-python.

[4] “An Introduction to Support Vector Machines (SVM).” MonkeyLearn Blog, 22 June 2017, monkeylearn.com/blog/introduction-to-support-vector-machines-svm/.

[5] Schaeffer , Don, and Taylor Maddix. “Exploring Support Vector Machine Acceleration with Vitis.” Xilinx, 7 Jan. 2020, developer.xilinx.com/en/articles/exploring-support-vector-machine-acceleration-with-vitis.html.

Appendix A MATLAB Functions

This section lists all the important MATLAB Functions used in the process of submarine tracking.

- `abs(X)`: returns the absolute value of each element in array `X`.
- `class = classify(sample,training,group)`: performs linear classfication on each row of the `sample` into one of the groups in `training` based on the labels in `group`.
- `tree = fitctree(train, group)`: returns the fitted decision tree classifier based on the columns of `train` and the `group` label.
- `Md1 = fitcecoc(train, group)`: returns the fitted multi-class SVM classifier based on the columns of `train` and the `group` label.
- `Md1 = fitcsvm(train, group)`: returns the fitted binary-class SVM classifier based on the columns of `train` and the `group` label.
- `lael = predict(md, test)`: returns the vector of predicted class labels for test data based on the classifier model `md`.
- `diag(A)`: returns the diagonal entries in matrix `A`.
- `I = find(X)`: returns the indices vector of elements of array `X` that matches given logical argument.
- `plot3(X,Y,Z)`: plots coordinates X,Y,X in 3D space
- `histogram(X,nbins)`: plots histogram of X with number of bins `nbins`
- `length(X)`: returns the length of an array `X`.
- `max(X)`: returns the maximum value in array `X`.
- `sum(X)`: returns the sum of the elements in array `X`.
- `cumsum(X)`: returns a vector of the cumulative sum of the elements in array `X`.
- `mean(A,2)`: returns a column vector of the average values of each row of matrix `A`.
- `repmat(X,m,n)`: creates a `m` by `n` matrix with the values in array `X`. This is used to zero center the data matrix by subtracting the mean of each row.
- `size(A)`: returns the dimension of matrix `A`.
- `sort(X)`: sorts the elements of vector `X` in ascending order.
- `norm(v, 2)`: returns the 2-norm of vector `v`.
- `[U,S,V] = svd(A, 'econ')`: performs economy singular value decomposition on matrix `A`. Returns the U,Σ,V matrices.
- `[V, D] = eig(A, B)`: returns the D diagonal matrix of eigenvalues and V with columns of eigenvectors. The results satisfy the equation $A * V = B * V * D$.
- `xticks(ticks)` sets the x-axis tick values.
- `xtickangle(a)` rotates the x-axis tick values by angle `n`.
- `xticklabels(labels)` newly labels the x-axis tick values with the given label values.
- `yticks(ticks)` sets the y-axis tick values.
- `yticklabels(labels)` newly labels the y-axis tick values with the given label values.

Appendix B MATLAB Code

B.1 assignment4.m

```matlab
clear; close all; clc;
%% unzip gz files
gunzip('train-images-idx3-ubyte.gz');
gunzip('train-labels-idx1-ubyte.gz');
gunzip('t10k-images-idx3-ubyte.gz');
gunzip('t10k-labels-idx1-ubyte.gz');
%% parse
[trnimgs, trnlbls] = mnist_parse("train-images-idx3-ubyte", "train-labels-idx1-ubyte");

[testimgs, testlbls] = mnist_parse("t10k-images-idx3-ubyte", "t10k-labels-idx1-ubyte");

%% reshape
testdat=[]; trndat=[];
for k = 1:size(testimgs,3)
    img = reshape(testimgs(:,:,k),1,[]);
    img = img';
    testdat(:,k) = img;
end
for k = 1:size(trnimgs,3)
    img = reshape(trnimgs(:,:,k),1,[]);
    img = img';
    trndat(:,k) = img;
end

%% training
mtrn = mean(trndat,2);
trndat = trndat - repmat(mtrn,1,size(trndat,2));
[Utrn, Strn, Vtrn] = svd(trndat, 'econ');
% projection onto principal components
Ytrn = Strn*Vtrn';

%% test
testdat = testdat - repmat(mtrn,1,size(testdat,2));
[Utst, Stst, Vtst] = svd(testdat, 'econ');
% projection onto principal components
Ytst = Stst*Vtst';

%% Plotting singular values + cumulative energy
figure()
subplot(2,1,1)
plot(diag(Strn), 'ko', 'Linewidth', 1)
ylabel('\sigma')
subplot(2,1,2)
plot(cumsum(diag(Strn).^2)/sum(diag(Strn).^2),'ko','Linewidth',1)
ylabel('Cumulative Energy')

figure()
subplot(2,1,1)
plot(diag(Stst), 'ko', 'Linewidth', 1)
ylabel('\sigma')
subplot(2,1,2)
```

```
plot(cumsum(diag(Stst).^2)/sum(diag(Stst).^2),'ko','Linewidth',1)
ylabel('Cumulative Energy')

%% Plotting
figure()
plot3(Vtrn(find(trnlbls==0),1), Vtrn(find(trnlbls==0),2), Vtrn(find(trnlbls==0),3), '.', 'color', [0 0.5
hold on
plot3(Vtrn(find(trnlbls==1),1), Vtrn(find(trnlbls==1),2), Vtrn(find(trnlbls==1),3), '.', 'color', 'b')
hold on
plot3(Vtrn(find(trnlbls==2),1), Vtrn(find(trnlbls==2),2), Vtrn(find(trnlbls==2),3), '.', 'color', 'g')
hold on
plot3(Vtrn(find(trnlbls==3),1), Vtrn(find(trnlbls==3),2), Vtrn(find(trnlbls==3),3), '.', 'color', 'r')
hold on
plot3(Vtrn(find(trnlbls==4),1), Vtrn(find(trnlbls==4),2), Vtrn(find(trnlbls==4),3), '.', 'color', 'k')
hold on
plot3(Vtrn(find(trnlbls==5),1), Vtrn(find(trnlbls==5),2), Vtrn(find(trnlbls==5),3), '.', 'color', 'm')
hold on
plot3(Vtrn(find(trnlbls==6),1), Vtrn(find(trnlbls==6),2), Vtrn(find(trnlbls==6),3), '.', 'color', 'y')
hold on
plot3(Vtrn(find(trnlbls==7),1), Vtrn(find(trnlbls==7),2), Vtrn(find(trnlbls==7),3), '.', 'color', [0.2 0
hold on
plot3(Vtrn(find(trnlbls==8),1), Vtrn(find(trnlbls==8),2), Vtrn(find(trnlbls==8),3), '.', 'color', [0.5 0
hold on
plot3(Vtrn(find(trnlbls==9),1), Vtrn(find(trnlbls==9),2), Vtrn(find(trnlbls==9),3), '.', 'color', 'c')
grid on
legend(string([0:1:9]));
xlabel('mode 1'); ylabel('mode 2'); zlabel('mode 3');

%% LDA on two digits
feature = 80;
I1 = find(trnlbls == 1); I0 = find(trnlbls == 0);
D1 = Ytrn(1:feature,I1); D0 = Ytrn(1:feature,I0);
n1 = size(D1,2);
n0 = size(D0,2);

m1 = mean(D1,2);
m0 = mean(D0,2);

Sw = 0; % within class variances
for j = 1:n1
    Sw = Sw + (D1(:,j) - m1)*(D1(:,j) - m1)';
end
for j = 1:n0
    Sw = Sw + (D0(:,j) - m0)*(D0(:,j) - m0)';
end
Sb = (m1-m0)*(m1-m0)'; % between class

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


% project onto w
```

```
vd1 = w'*D1;
vd0 = w'*D0;

if mean(vd0) > mean(vd1)
    w = -w;
    vd1 = -vd1;
    vd0 = -vd0;
end

sort1 = sort(vd1);
sort0 = sort(vd0); t1 = 1; t0 = length(sort0);

while sort0(t0) > sort1(t1)
    t0 = t0 - 1;
    t1 = t1 + 1;
end
threshold = (sort0(t0) + sort1(t1))/2;

%%
figure()
plot(vd0,zeros(n0),'ob')
hold on
plot(vd1,ones(n1),'or')
yticks([0 1])
yticklabels({'digit 0', 'digit 1'})

figure()
subplot(1,2,1)
histogram(sort0,30); hold on, plot([threshold threshold], [0 1000], 'r')
title('digit 0')
subplot(1,2,2)
histogram(sort1,30); hold on, plot([threshold threshold], [0 1200], 'r')
title('digit 1')

%% 2 digits accuracy
feature = 30;
count = 1;
success_lda = [];
for i=1:10
    for j=i+1:10
        d1 = i-1; d2 = j-1;
        T1 = find(testlbls==d1); T2 = find(testlbls==d2);
        sample = [testdat(:,T1) testdat(:,T2)];
        I1 = find(trnlbls==d1); I2 = find(trnlbls==d2);
        [U,S,V,threshold,w,sortd1,sortd2] = digit_trainer(trndat(:,I1), trndat(:,I2),feature);
        testmat = U'*sample;
        pval = w'*testmat;
        ResVec = (pval > threshold);
        err = abs(ResVec - [zeros(1,length(T1)) ones(1,length(T2))]);
        success_lda(count) = 1-sum(err)/length(ResVec);
        count = count+1;
    end
end
%% plotting success rate
[m1 m1i] = max(success_lda); [m2 m2i] = min(success_lda);
```

```
figure()
plot(success_lda,'-o'); ylim([0.9, 1]);
hold on
plot(m1i, m1,'ro')
hold on
plot(m2i, m2,'go')
xticks(1:45)
xtickangle(90)
xticklabels(pairs)
xlabel('pairs'); ylabel('accuracy %')
title('LDA success rate for all digit pairs')

%% LDA on three digits; classification using classify function
feature = 80;
I0 = find(trnlbls == 0); I1 = find(trnlbls == 1); I2 = find(trnlbls == 4);
Utrn_r = Utrn(:,1:feature);
D0 = Utrn_r'*trndat(:,I0); D1 = Utrn_r'*trndat(:,I1); D2 = Utrn_r'*trndat(:,I2);
train = [D0 D1 D2];
group = [trnlbls(I0); trnlbls(I1); trnlbls(I2);];

It0 = find(testlbls == 0); It1 = find(testlbls == 1); It2 = find(testlbls == 4);
sampled0 = Utrn_r'*testdat(:,It0);
sampled1 = Utrn_r'*testdat(:,It1);
sampled2 = Utrn_r'*testdat(:,It2);
sample = [sampled0 sampled1 sampled2];
class = classify(sample', train', group);
err = class - [testlbls(It0);testlbls(It1);testlbls(It2)];
success_3d = length(find(err==0))/length(err); %0.9709

%% decision tree multiclass
feature = 30;
count = 1;
success_tree = [];
Utrn_r = Utrn(:,1:feature);
tree=fitctree((Utrn_r'*trndat)',trnlbls);
tree_labels1 = predict(tree, (Utrn_r'*testdat)');

err_tree = abs(tree_labels1-testlbls);
length(find(err_tree==0))/length(err_tree) % 0.8522

%% decision tree pair accuracy
feature = 30;
count = 1;
success_tree = [];
Utrn_r = Utrn(:,1:feature);
for i=1:10
    for j=i+1:10
        d1 = i-1; d2 = j-1;
        I1 = find(trnlbls == d1); I2 = find(trnlbls == d2);
        D1 = Utrn_r'*trndat(:,I1); D2 = Utrn_r'*trndat(:,I2);
        train = [D1 D2];
        group = [trnlbls(I1); trnlbls(I2)];
        tree=fitctree(train',group);
        T1 = find(testlbls==d1); T2 = find(testlbls==d2);
        test = [Utrn_r*testdat(:,T1) Utrn_r*testdat(:,T2)];
        label = predict(tree, test');
        err_tree = abs(label-[testlbls(T1); testlbls(T2)]);
        success_tree(count) = length(find(err_tree==0))/length(err_tree);
        count = count+1;
    end
end
%% decision tree accuracy plot
[m1 m1i] = max(success_tree); [m2 m2i] = min(success_tree);
figure()
plot(success_tree,'-o'); ylim([0.95, 1]);
hold on
plot(m1i, m1,'ro')
hold on
plot(m2i, m2,'go')
xticks(1:45)
xtickangle(90)
xticklabels(pairs)
xlabel('pairs'); ylabel('accuracy %')
title('Decision Tree success rate for all digit pairs')

%% multiclass svm
feature = 30;
Utrn_r = Utrn(:,1:feature);
Md1 = fitcecoc((Utrn_r'*trndat)',trnlbls);
svm_lables = predict(Md1, (Utrn_r'*testdat)');

%%
err_ecoc = abs(svm_lables-testlbls);
success_ecoc = length(find(err_ecoc==0))/length(err_ecoc) %0.4465
%% svm
feature = 30;
count = 1;
success_svm = [];
Utrn_r = Utrn(:,1:feature);
for i=1:10
    for j=i+1:10
        d1 = i-1; d2 = j-1;
        I1 = find(trnlbls == d1); I2 = find(trnlbls == d2);
        D1 = Utrn_r'*trndat(:,I1); D2 = Utrn_r'*trndat(:,I2);
        train = [D1 D2];
        group = [trnlbls(I1); trnlbls(I2)];
        Md1=fitcsvm(train',group);
        T1 = find(testlbls==d1); T2 = find(testlbls==d2);
        test = [Utrn_r*testdat(:,T1) Utrn_r*testdat(:,T2)];
        label = predict(Md1, test');
        err_svm = abs(label - [testlbls(T1); testlbls(T2)]);
        success_svm(count) = length(find(err_svm==0))/length(err_svm);
        count = count+1;
    end
end
%% svm accuracy
[m1 m1i] = max(success_svm); [m2 m2i] = min(success_svm);
figure()
plot(success_svm,'-o'); ylim([0, 1]);
hold on
```

```
plot(m1i, m1,'ro')
hold on
plot(m2i, m2,'go')
xticks(1:45)
xtickangle(90)
xticklabels(pairs)
xlabel('pairs'); ylabel('accuracy %')
title('SVM success rate for all digit pairs')
%%
pairs = ["(0,1)", "(0,2)", "(0,3)", "(0,4)", "(0,5)", "(0,6)", "(0,7)", "(0,8)", "(0,9)", "(1,2)", "(1,3)
```

B.2 digit_trainer.m

```
function [U_d,S_d,V_d,threshold,w,sort1,sort2] = digit_trainer(digit1,digit2,feature)

    nd = size(digit1,2);
    nc = size(digit2,2);
    [U_d,S_d,V_d] = svd([digit1 digit2],'econ');
    digits = S_d*V_d';
    U_d = U_d(:,1:feature); % Add this in
    d1 = digits(1:feature,1:nd);
    d2 = digits(1:feature,nd+1:nd+nc);
    m1 = mean(d1,2);
    m2 = mean(d2,2);

    S_dw = 0;
    for k=1:nd
        S_dw = S_dw + (d1(:,k)-m1)*(d1(:,k)-m1)';
    end
    for k=1:nc
        S_dw = S_dw + (d2(:,k)-m2)*(d2(:,k)-m2)';
    end
    S_db = (m1-m2)*(m1-m2)';

    [V_d2,D] = eig(S_db,S_dw);
    [lambda,ind] = max(abs(diag(D)));
    w = V_d2(:,ind);
    w = w/norm(w,2);
    vd1 = w'*d1;
    vd2 = w'*d2;

    if mean(vd1)>mean(vd2)
        w = -w;
        vd1 = -vd1;
        vd2 = -vd2;
    end

    % Don't need plotting here
    sort1 = sort(vd1);
    sort2 = sort(vd2);
    t1 = length(sort1);
    t2 = 1;
    while sort1(t1)>sort2(t2)
    t1 = t1-1;
    t2 = t2+1;
    end
    threshold = (sort1(t1)+sort2(t2))/2;

    % We don't need to plot results
end
```

Appendix C Supplementary Figures

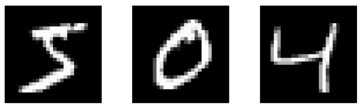


Figure 6: Sample digit images from MNIST data

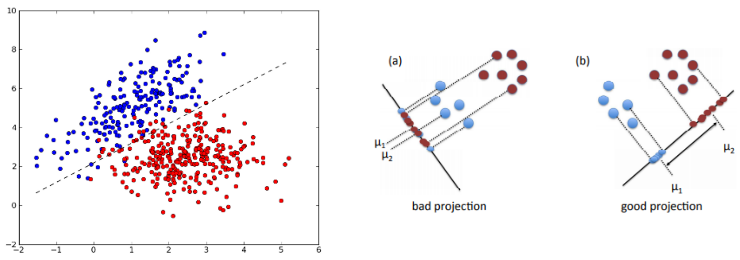


Figure 7: Visualization of LDA classifier (left: [1] right: [Kutz])

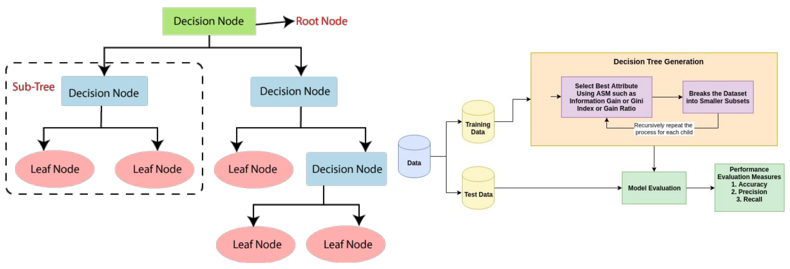


Figure 8: Visualization of Decision Tree classifier (left: [2] right: [3])

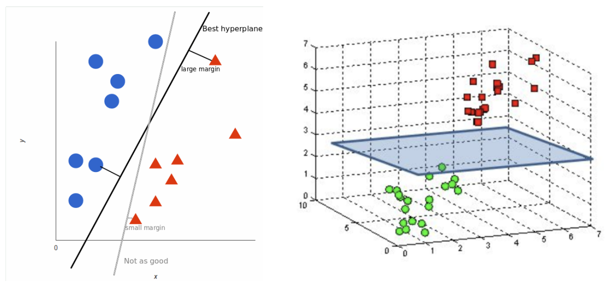


Figure 9: Visualization of SVM classifer in 2D and 3D (left: [4] right: [5])