CS303B Artificial Intelligence B

Southern University of Science and Technology

Assignment 2 - Report

Assignment 2 - Classification and Clustering

Name: Zhang Ce Student ID: 11910803

Contents

1 Introduction

In this assignment, I use PCA and LDA to implement dimension reduction on MNIST dataset, and also try different clustering methods. Additionally, I adopt SVM and simple feedforward neural network to complete the binary classification task, with cross-validation results and ROC curves attached. Here gives an general introduction for this report. In section 2, I will answer question 1 and 2,which is about dimensionality reduction and clustering. In section 3, I will use different classifiers to do the binary classification task, and also evaluate the results.

2 Part I: Dimension Reduction and Clustering

2.1 Principle Component Analysis (PCA) and Clustering

Principal component analysis is an unsupervised statistical procedure that allows us to summarize the information content in large data dimensions by means of a smaller set of "summary indices" that can be more easily visualized and analyzed. Its goal is to find several directions such that the variance of the projected data in the subspace is maximized. In this part, I use PCA to reduce dimensions of a partial MNIST dataset containing only digit 1,5,8, followed by adopting different clustering methods. I will guide you through how I adopt PCA and present you my results.

Firstly, since the matrix *images* is size 784×600, with 28 pixels a image in every column. We need to put the 784 dimension that we desire to reduce in the columns, so we transpose the data matrix initially, we mark this data table as A, which is an $m \times n$ matrix.

Then we do the data normalization, deleting the average value from each column (represented by a), followed by calculation of the covariance matrix R.

$$a = a - \frac{1}{m}\sum_{i=1}^{m} a_i \tag{1}$$

$$R = \frac{1}{m-1}A^T A \tag{2}$$

We do diagonalization (eigenvalue decomposition) for real symmetric positive semidefinite matrix R, getting a diagonal matrix with all real eigenvalues, and keep the first two leading eigenvectors with greatest eigenvalues as new basis, deleting the

rest part. The orthogonal diagonalization process is given by Equation 3, where Q is an orthogonal matrix with orthonormal eigenvectors in columns, while Σ is a diagonal matrix with eigenvalues on the main diagonal.

$$R = Q\Sigma Q^T \tag{3}$$

Finally, we can get a 2-dimension plot of the original 784-dimension pictures with lowest wastage. The related code is attached as follows.

```
% --- Principle Component Analysis ---
images = images.';
% Normalize the data (mean or zscore)
images_nor = images - mean(images);
% images_nor = zscore(images);
% Calculate the covariance matrix
r = cov(images_nor);
% Eigenvalue decomposition
[v, ~] = eigs(r);
% Calculate scores
score = images_nor * v(:,1:2);
% ------------------------------------
```

After executing the codes above, the final result 2-D plot after dimension reduction using PCA is shown in Figure 1.

Observing this result plot, we can find that the data points of digit 1 distribute on the left side, while digit 5 lies on the bottom-right, digit 8 on the top-right. We can see that the PCA process does separate those datapoints in the 2-dimension plot.

Next, we do clustering to divide the data points to 3 clusters. We firstly try to use K-means clustering, an unsupervised learning algorithm that divide unlabelled dataset into k different clusters. It is a centroid-based clustering method, which will choose a set of k centroids and keep updating to make the final result converge. Here I use the build-in function in MATLAB `kmeans()` to adopt clustering after PCA. The codes are shown as following and the clustering result is shown in Figure 2.

```
% ----- K-Means Clustering -----
[idx,C] = kmeans(score, 3);
x1 = min(score(:, 1)):0.01:max(score(:, 1));
x2 = min(score(:, 2)):0.01:max(score(:, 2));
[x1G,x2G] = meshgrid(x1, x2);
```

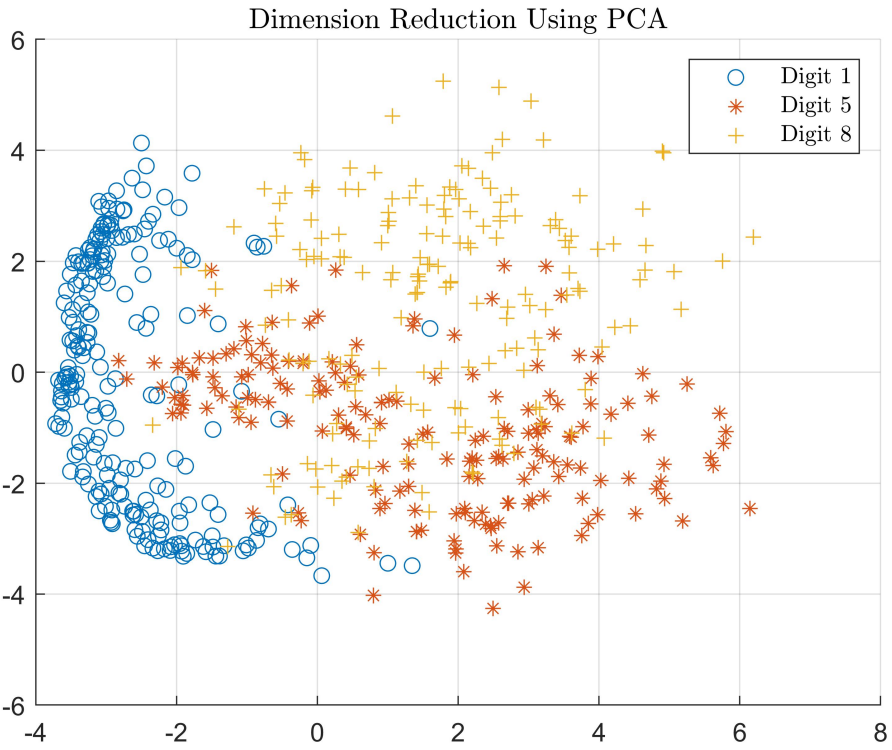

Figure 1: Dimension Reduction Using PCA

```
XGrid = [x1G(:), x2G(:)];
idx2Region = kmeans(XGrid, 3, 'MaxIter', 40, 'Start', C);
% ------------------------------
```

In Figure 2, I also mark the centroid of every cluster on the plot. We can observe that the clustering result fits our intuition: the 3 clusters are roughly digit 1, digit 5, digit 8. And as is obviously shown, the similar data points are clustered together, marked with the same color.

Here I also try hierarchical clustering, a connectively-based clustering method. Different from the centroid-based clustering method K-means, hierarchical clustering will try to calculate the distances between clusters and make a final decision. Here I use average linkage, where the distances between different clusters are determined by the average value of the distances between every data pairs. In mathematical language, distance between cluster r and s $L(r,s)$ is calculated by Equation 4.

$$L\left(r,s\right)=\frac{1}{n_r n_s}\sum_{i=1}^{n_r}\sum_{j=1}^{n_s}D\left(x_{ri},x_{sj}\right) \tag{4}$$

I adopt hierarchical clustering by `linkage()` and `cluster()` method, here as

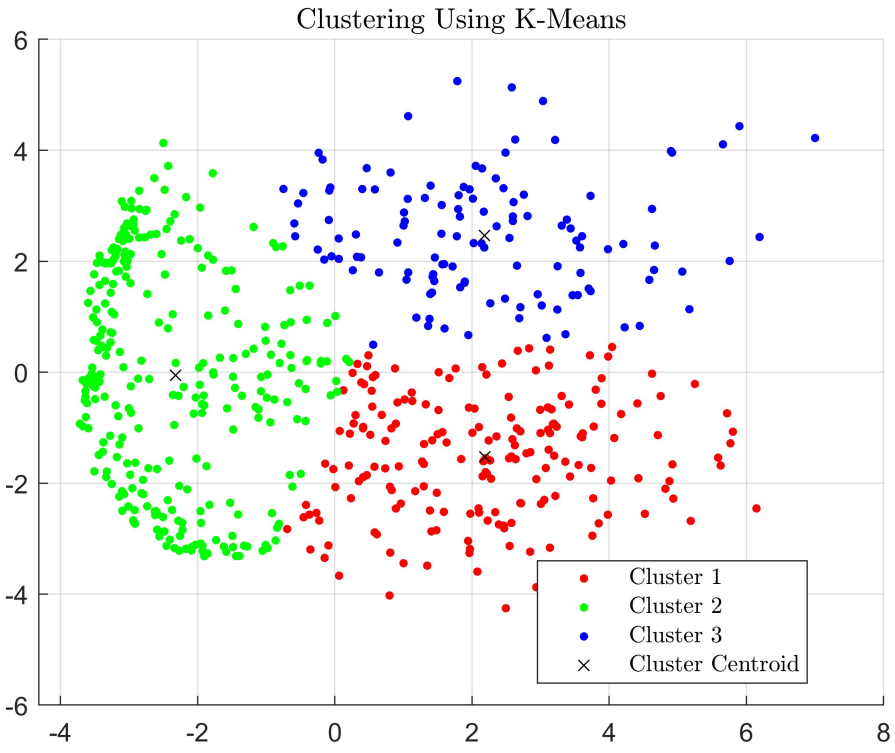

Figure 2: Clustering Result Using K-means after PCA

the following shows the codes related to hierarchical clustering. The dendrogram plot is shown in Figure 3 and the result is shown in Figure 4.

```
% ----- Hierarchical Clustering -----
%create the linkage tree using average-link
Z = linkage(score(:,1:2),'average','euclidean');
c = cluster(Z,'maxclust',3);
% -----------------------------------
```

Gaussian mixture model (GMM) clustering, a distribution based clustering method, is also adopted to complete the clustering task. As our options of covariance matrix type change, the clustering result also changes. Here as the following shows the codes related to GMM clustering. The result is shown in Figure 5.

```
% ----------------- GMM clustering -----------------
k = 3; % Number of GMM components
options = statset('MaxIter',1000);

% Options for covariance matrix type
Sigma = {'diagonal','full'};
nSigma = numel(Sigma);
```

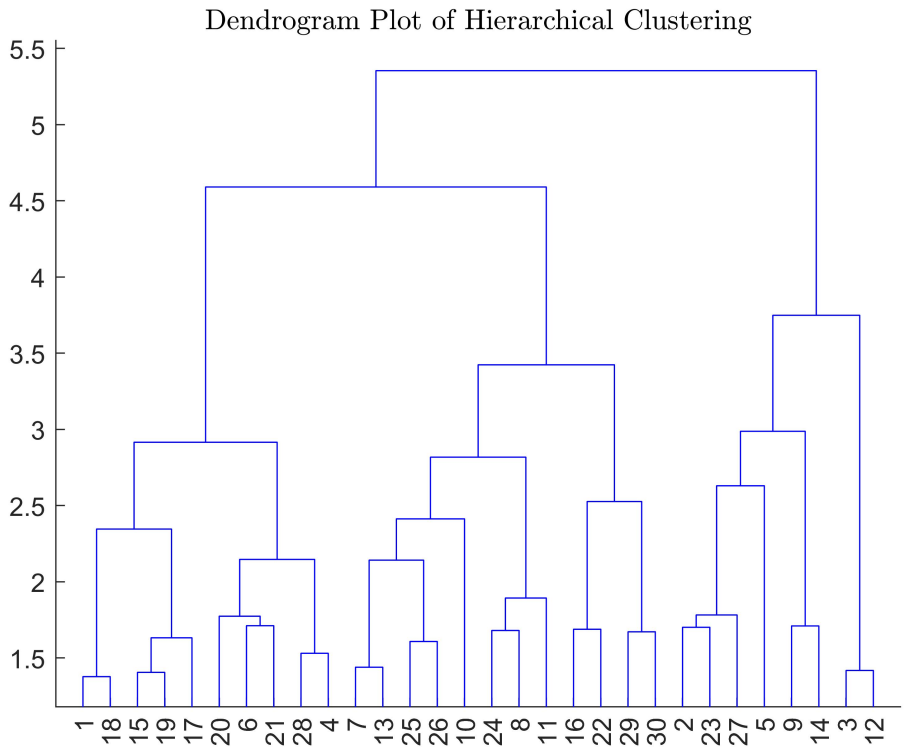

Figure 3: Dendrogram Plot of Hiererachical Clustering after PCA

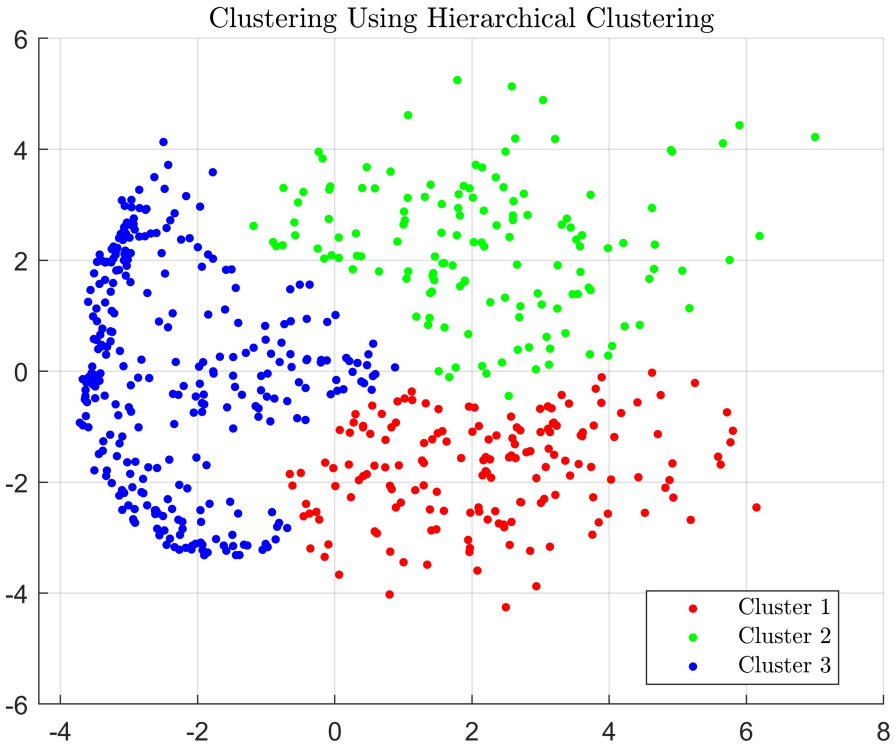

Figure 4: Clustering Result Using Hierarchical Clustering after PCA

```
8
9  % Indicator for identical or nonidentical covariance matrices
10 SharedCovariance = {true,false};
11 SCtext = {'true','false'};
12 nSC = numel(SharedCovariance);
13
14 d = 500; % Grid length
15 x1 = linspace(min(score(:,1))-2, max(score(:,1))+2, d);
16 x2 = linspace(min(score(:,2))-2, max(score(:,2))+2, d);
17 [x1grid,x2grid] = meshgrid(x1,x2);
18 X0 = [x1grid(:) x2grid(:)];
19
20 threshold = sqrt(chi2inv(0.99,2));
21 count = 1;
22 for i = 1:nSigma
23     for j = 1:nSC
24         gmfit = fitgmdist(score,k,'CovarianceType',Sigma{i}, ...
25             'SharedCovariance',SharedCovariance{j},'Options',options); %
   ↪  Fitted GMM
26         clusterX = cluster(gmfit,score); % Cluster index
27         mahalDist = mahal(gmfit,X0); % Distance from each grid point to each
   ↪  GMM component
28         % Draw ellipsoids over each GMM component and show clustering
   ↪  result.
29         subplot(2,2,count);
30         h1 = gscatter(score(:,1),score(:,2),clusterX);
31         hold on
32             for m = 1:k
33                 idx = mahalDist(:,m)<=threshold;
34                 Color = h1(m).Color*0.75 - 0.5*(h1(m).Color - 1);
35                 h2 =
   ↪  plot(X0(idx,1),X0(idx,2),'.','Color',Color,'MarkerSize',1);
36                 uistack(h2,'bottom');
37             end
38         plot(gmfit.mu(:,1),gmfit.mu(:,2),'kx','LineWidth',2,'MarkerSize',10)
39         title(sprintf('Sigma is %s\nSharedCovariance =
   ↪  %s',Sigma{i},SCtext{j}),'FontSize',8)
40         legend(h1,{'1','2','3'})
41         hold off
42         count = count + 1;
```

```
43      end
44  end
45  % ----------------------------------------------
```

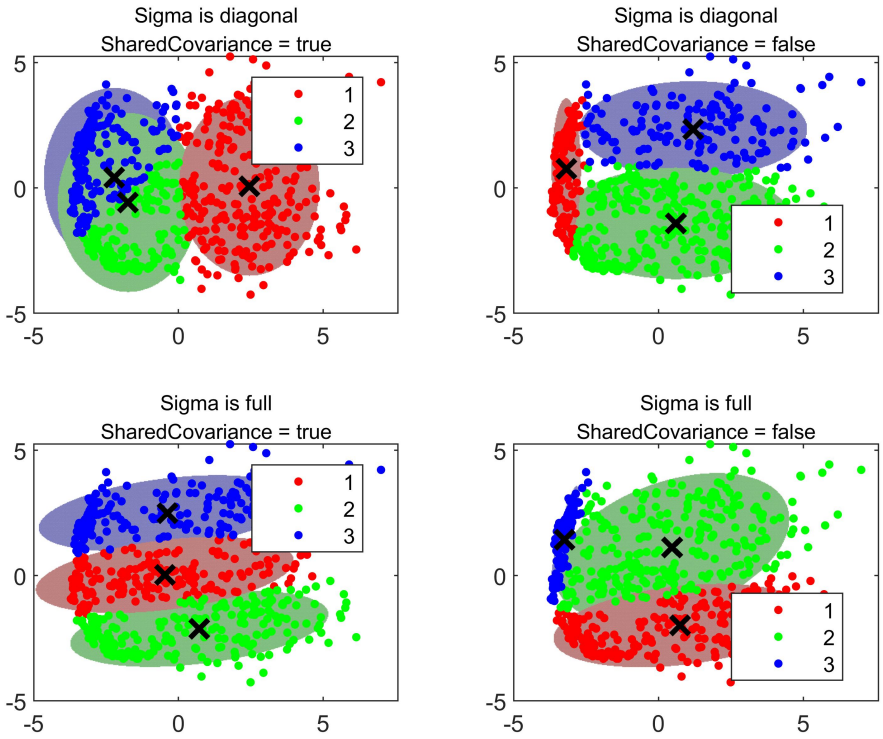

Figure 5: Clustering Result Using GMM Clustering After PCA

2.2 Linear Discriminant Analysis (LDA) and Clustering

Linear discriminant analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. PCA is used to maximize the variance of the projected data, while it may lead our data impossible to divide. Different with that, LDA finds most discriminant projection by maximizing between-class distance and minimizing within-class distance. In this part, I use LDA to reduce dimensions of the same partial MNIST dataset containing only digit 1,5,8, followed by adopting different clustering methods.

At first, we still need to pre-process the data matrix by deleting the mean value in each column. Then the between-class scatter S_b can be calculated by Equation 5, while within-class scatter S_w can be calculated by Equation 6, where S_1, S_2, S_3 are the

covariance matrix of data from digit 1, 5, 8.

$$S_b = \sum_{k=1}^{K} N_k \left(m_k - m\right)\left(m_k - m\right)^T \tag{5}$$

$$S_w = S_1 + S_2 + S_3 \tag{6}$$

The trace of those 2 matrices represent the between-class distance and the within-class distance. To maximize between-class distance and minimize within-class distance, we need to find the parameters that satisfies the following expression.

$$\arg\max_w \frac{trace\left(W^T S_b W\right)}{trace\left(W^T S_w W\right)}$$

Therefore, the optimal transformation is given by solving a generalized eigenvalue problem for $S_w^{-1} S_b$. To be noteworthy, the within-class scatter is singular for most of the cases, here we add an identity part kI with $k = 1e-10$ to S_w, making it has full rank. Finally we take the leading 2 eigenvectors as our projection directions to reduce the dimension from 784 to 2.

The whole implementation in MATLAB is given as following. The LDA result is shown in Figure 6.

```
images = images.';
class1 = images(labels==1,:);
class5 = images(labels==5,:);
class8 = images(labels==8,:);

% class means
m1 = mean(class1);
m5 = mean(class5);
m8 = mean(class8);
m = mean(images);
% class covariance matrix
s1 = cov(class1);
s5 = cov(class5);
s8 = cov(class8);
% within class scatter matrix
sw = s1 + s5 + s8;

% between class scatter matrix
```

```
mb = zeros(3, 784);
mb(1, :) =  m1 - m;
mb(2, :) =  m5 - m;
mb(3, :) =  m8 - m;
sb = mb.' * mb;

% computing the LDA projection vector
[v, d] = eigs((inv(sw + 1e-10 * eye(784))) * sb);

% computing the projection score:
score = images * v(:, 1:2);
```

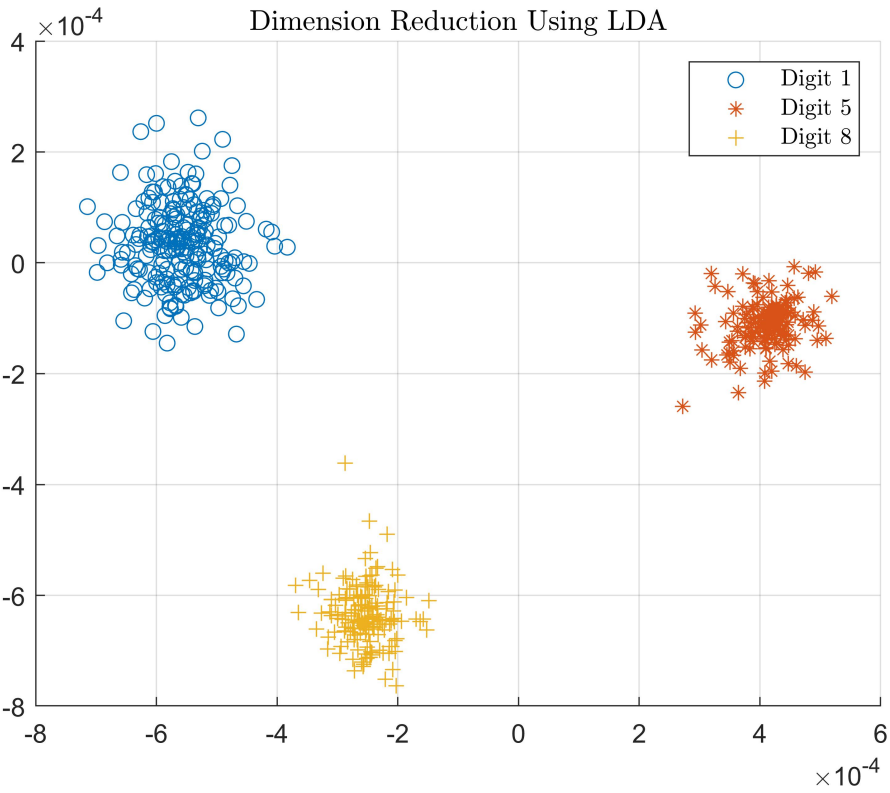

Figure 6: Dimension Reduction Using LDA

From the result, we can clearly seen that the different digits are divided into different clusters in the 2-D plot, with no overlapping area. Then we repeat the clustering process introduced in PCA part, The results are shown in Figure 7, 8, 9.

Since LDA has divided the class clearly, all the clustering methods perform well in this task, with different clusters exactly the different digits in MNIST dataset.

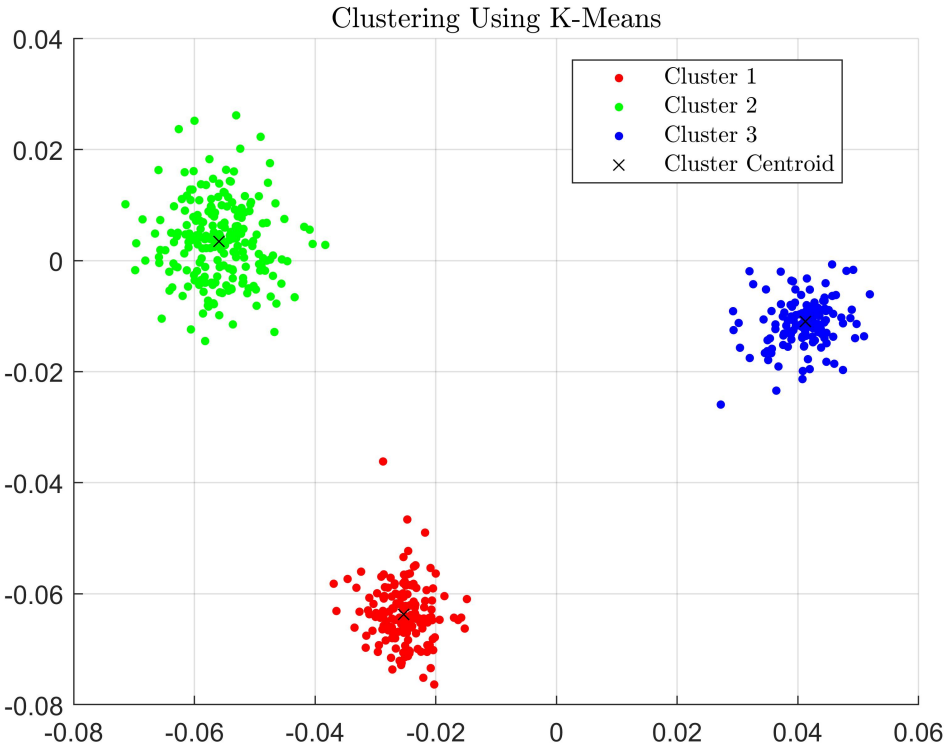

Figure 7: Clustering Result Using K-means after LDA

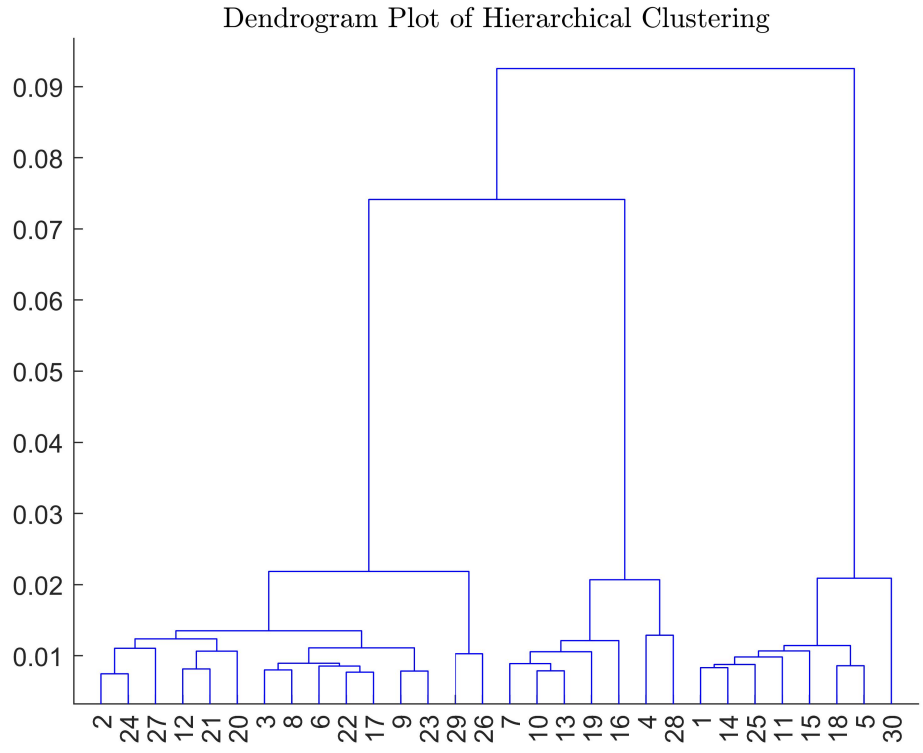

Figure 8: Dendrogram Plot of Hierarchical Clustering after LDA

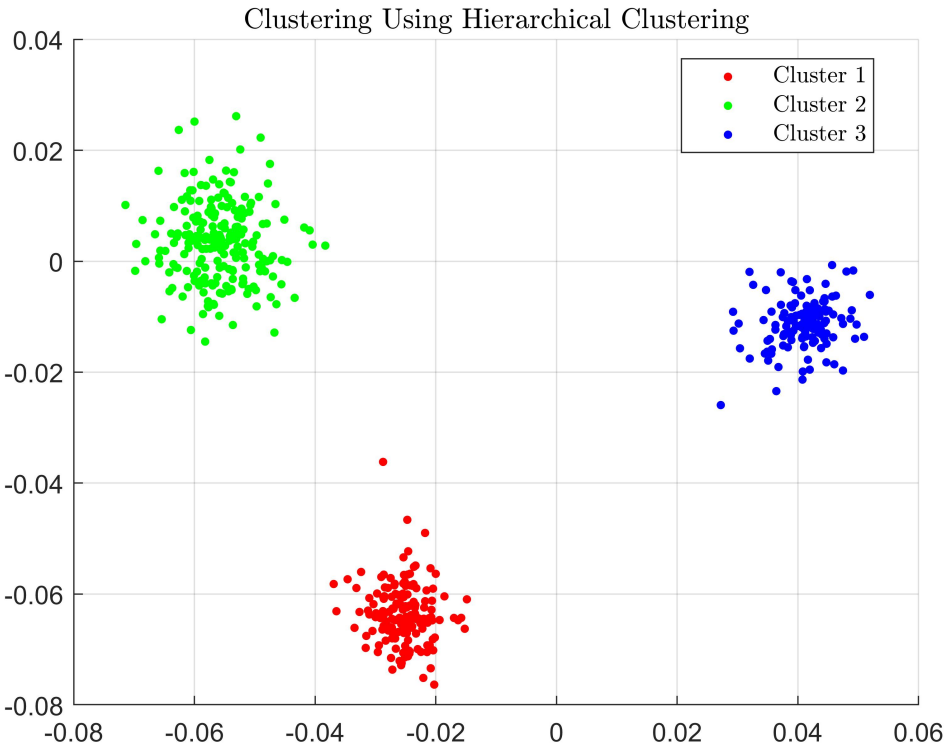

Figure 9: Clustering Result Using Hierarchical Clustering after LDA

2.3 PCA and LDA Using Larger MNIST Dataset

Here I download the whole MNIST datasets with $60,000$ pictures form digit 0 to digit 9, and repeat the LDA and PCA to reduce the dimension to 2. The following results in Figure 10 and 11 are the dimensionality reduction plot using $2,000$ random pictures from the whole dataset, containing 10 different digits.

We can actually observe some interesting points in these 2 plots. For example, in the PCA 2-D plot, digit 0 distributes on the bottom, while digit 1 distributes on the top-right, we can guess maybe the direction on y axis is finding curves and straight lines. However, in the center of PCA plot, there are many classes mixed together, making it difficult for us to do the classification, that is natural because PCA is an unsupervised algorithm. Comparatively, when we use LDA, which is a supervised algorithm, the data points of different classes are distributed in different regions in the final 2-D plot.

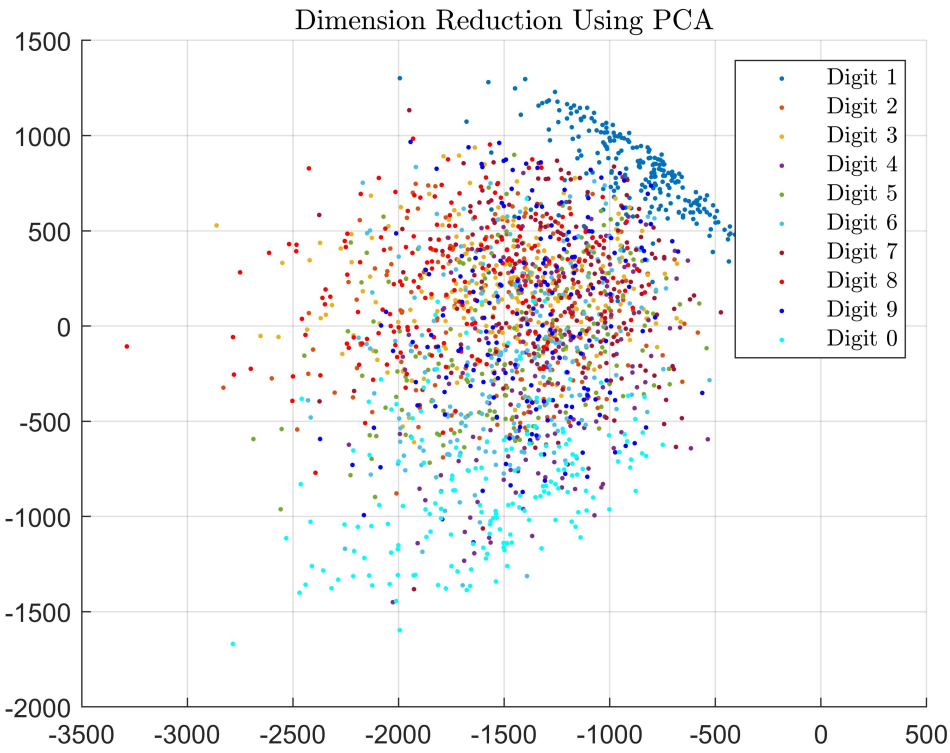

Figure 10: Dimension Reduction Using PCA (2000 data points)

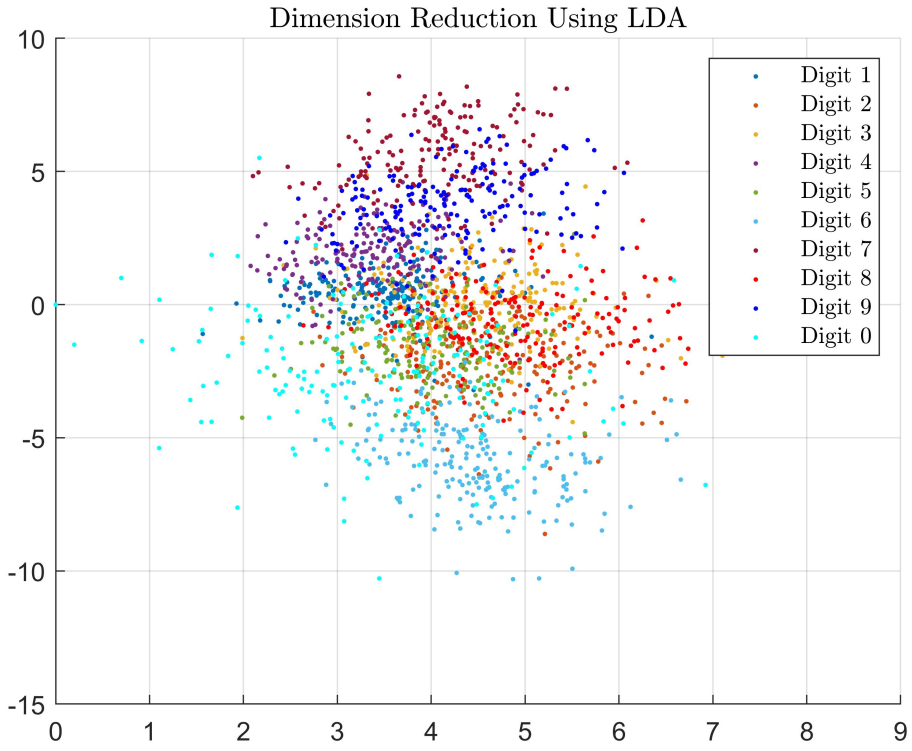

Figure 11: Dimension Reduction Using LDA (2000 data points)

3 Part II: Binary Classification

3.1 Preprocessing of Data, Cross-Validation Settings, Generating ROC Curves

Firstly, we divide the dataset by digit 5 against others. Here I set digit 5 with label 1 and others with label 0.

```
% preparing the vectors for digit 5 against the rest
a = zeros(size(labels));
a(labels==5) = 1;
```

I also use 5-fold cross-validation in this part to get a authentic accuracy to evaluate the model. The codes to implement the cross-validation process is given as below. To be noteworthy, we only divide the dataset once to guarantee the same division for different classifiers. That means, `cvpartition()` will only be executed once during this whole binary classification task.

```
% split the data into 5 fold.
cvo = cvpartition(a,'KFold',5);

predict_label_total = [];
test_label_vector_total = [];

% cross validation
accuracy_avg = 0;
for i = 1:5
trIdx = cvo.training(i); %% get the index of training samples
teIdx = cvo.test(i); %% get the index of the test samples
training_label_vector = a(trIdx); %% creating the training label
%ground truth
training_instance_matrix = images(trIdx,:); %% creating the training
%data matrix
test_label_vector = a(teIdx); %% creating the testing label
%ground truth
test_instance_matrix = images(teIdx,:); %% creating the test data

train...
predict...

predict_label_total = [predict_label_total; predict_label];
test_label_vector_total = [test_label_vector_total; test_label_vector];
```

```
end
% cross validation accuracy
accuracy_avg = accuracy_avg / 5;
```

I also try to plot the ROC curves and calculate the area under the curve (AUC) by myself using a self-written MATLAB function.

```
function  auc = plot_roc(predict, ground_truth)
% plot the ROC curve

x = 1.0;
y = 1.0;

%calculate numbers of positive and negative samples
pos_num = sum(ground_truth==1);
neg_num = sum(ground_truth==0);

%calculate step size for plot
x_step = 1.0/neg_num;
y_step = 1.0/pos_num;

%sort the output value
[~,index] = sort(predict);
ground_truth = ground_truth(index);

%plot the ROC curve
for i=1:length(ground_truth)
    if ground_truth(i) == 1
        y = y - y_step;
    else
        x = x - x_step;
    end
    X(i)=x;
    Y(i)=y;
end

hold on;
grid on;
set(gcf,'Position',[50/0.277 50/0.277 100/0.277 100/0.277]);
plot(X,Y,'-b','LineWidth',2,'MarkerSize',3);
plot(X,X,'--k','LineWidth',0.5)
```

```
xlim([0 1]); ylim([0 1]);

xlabel('False Positive Rate', 'Interpreter', 'latex');
ylabel('True Positive Rate', 'Interpreter', 'latex');
title('ROC Curve', 'Interpreter', 'latex');
%calculate the area under ROC curve, i.e. auc
auc = -trapz(X,Y);
end
```

3.2 SVM with RBF Kernal

I trained a SVM with RBF kernal ($\gamma = 0.07$) and evaluated the performance by using the `svmtrain()` and `svmpredict()`.

```
model = svmtrain(training_label_vector, training_instance_matrix, '-t 2 -g
↪  0.07');
[predict_label, accuracy, dec_values] = svmpredict(test_label_vector,
↪  test_instance_matrix, model);
```

The average accuracy after 5-fold cross-validation is **97.83%**. Area under the ROC curve AUC is **0.9710**. The ROC curve is shwn in Figure 12.

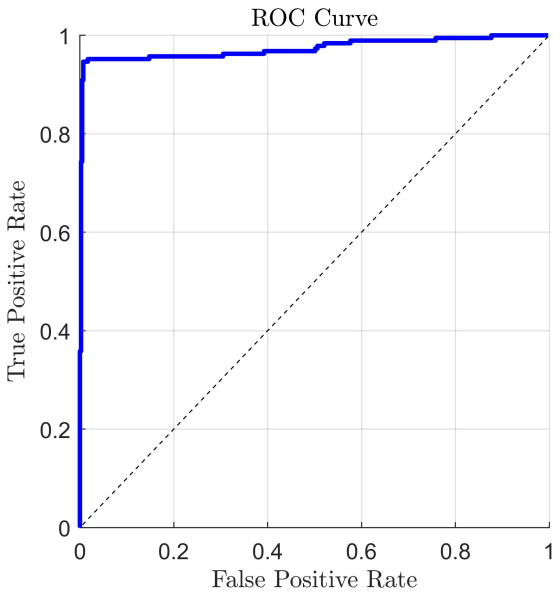

Figure 12: ROC Curve of SVM with RBF Kernal

3.3 SVM with Linear Kernal

In this part, I trained a SVM with linear kernal and evaluated the performance by using the `svmtrain()` and `svmpredict()`. The goal of SVM is to find a line that can

have the maximum spread with the data points. RBF kernal is common in defining SVM, it provides nonlinerity for the classification regions.

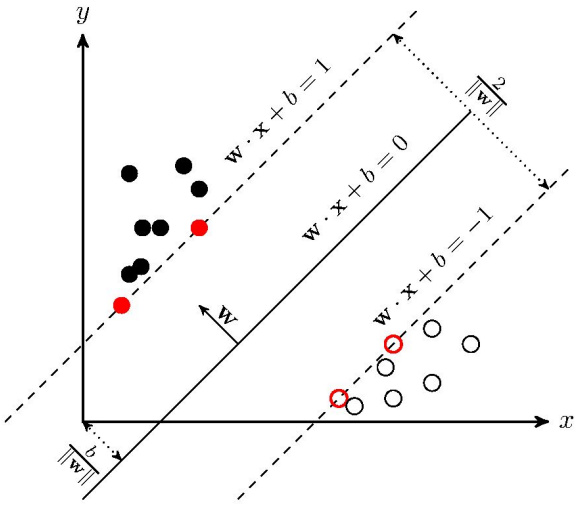

Figure 13: Schematic Plot of SVM

```
model = svmtrain(training_label_vector, training_instance_matrix, '-t 0');
[predict_label, accuracy, dec_values] = svmpredict(test_label_vector,
    test_instance_matrix, model);
```

The average accuracy after 5-fold cross-validation is **95.00%**. Area under the ROC curve AUC is **0.9420**. The ROC curve is shwn in Figure 14.

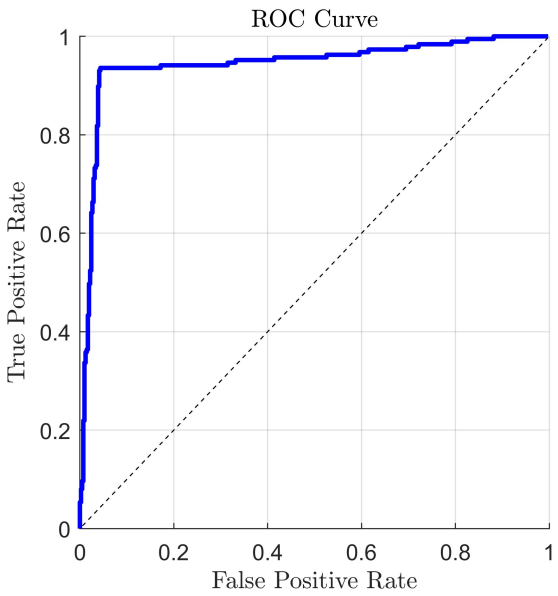

Figure 14: ROC Curve of SVM with Linear Kernal

3.4 Neural Network

In this part, I trained a neural network with 1 hidden layer with sigmoid activation function and SGD, evaluated the performance by using the `feedforwardnet()`. The related codes are attached.

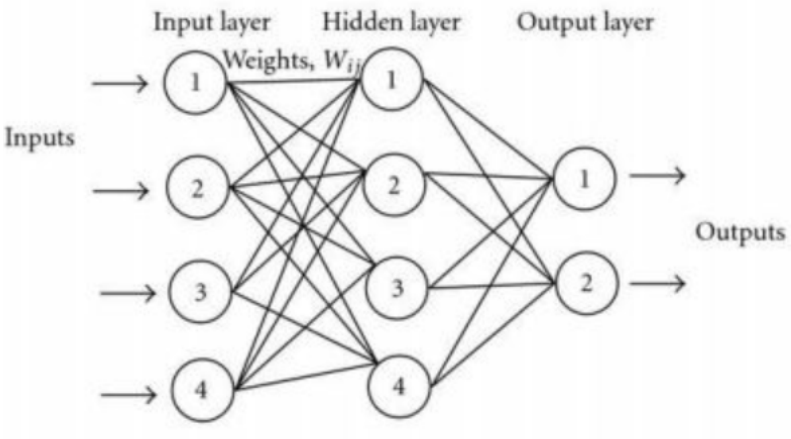

Figure 15: Schematic Plot of Neural Network

```
% Configure the net
net.divideParam.trainRatio = 1; % training set [%]
net.divideParam.valRatio = 0; % validation set [%]
net.divideParam.testRatio = 0; % test set [%]
net.inputs{1}.processFcns = {}; % modify the process function for inputs
net.outputs{2}.processFcns = {}; % modify the process function for outputs
net.layers{1}.transferFcn = 'logsig'; % the transfer function for the first
  ↪ layer
net.layers{2}.transferFcn = 'logsig'; % the transfer function for the
  ↪ second layer
net.trainParam.lr = 0.3; % learning rate. You may need to adjust it in the
  ↪ experiment.

% train the network
net = train(net, training_instance_matrix', training_label_vector');
```

The average accuracy after 5-fold cross-validation is **95.27%**. Area under the ROC curve AUC is **0.9737**. The ROC curve is shwn in Figure 16.

3.5 Summary

To summarize, SVM with RBF kernal performs the best in such division of dataset, while the neural network has the greatest AUC, which means it can do well in almost all

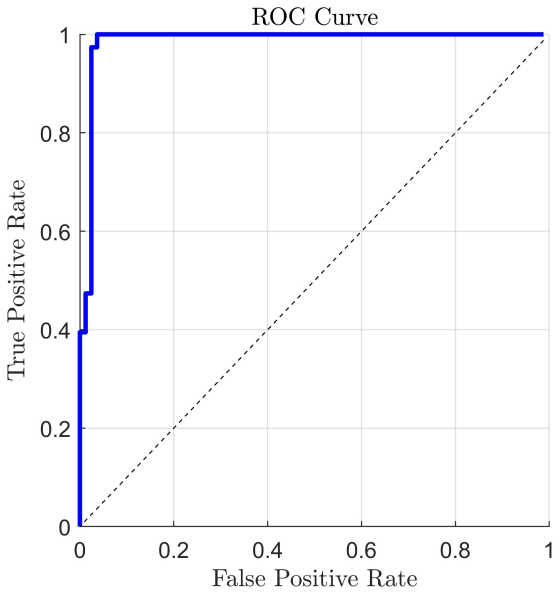

Figure 16: ROC Curve of Neural Network

the thresholds. SVM with linear kernal performs badly in this experiment. However, all these 3 models have a AUC much greater than 0.5, which is the AUC of random guess.

3.6 Fine-Tune the Parameters

Here we choose to fine-tune the hyperparameter γ in SVM with RBF kernal. We list the possible candidates for γ.

```
1 gamma_choices = [0.001, 0.007, 0.02, 0.035, 0.05, 0.07, 0.1];
```

For each candidate, we plot the ROC curves, get the AUC and calculate the average cross-validation accuracy in 3 different 5-fold cross validation settings. The experiment result is given in Figure 17 and 18.

As we can observe, the cross-validation accuracy and AUC will climb up initially as we increase the value of γ, followed a stable peak period, and finally, the cross-validation accuracy and AUC will drop dramatically. Therefore, the hyperparameter γ should be chosen between 0.02 and 0.05, which can make the SVM with RBF kernal performs the best on this partial MNIST dataset.

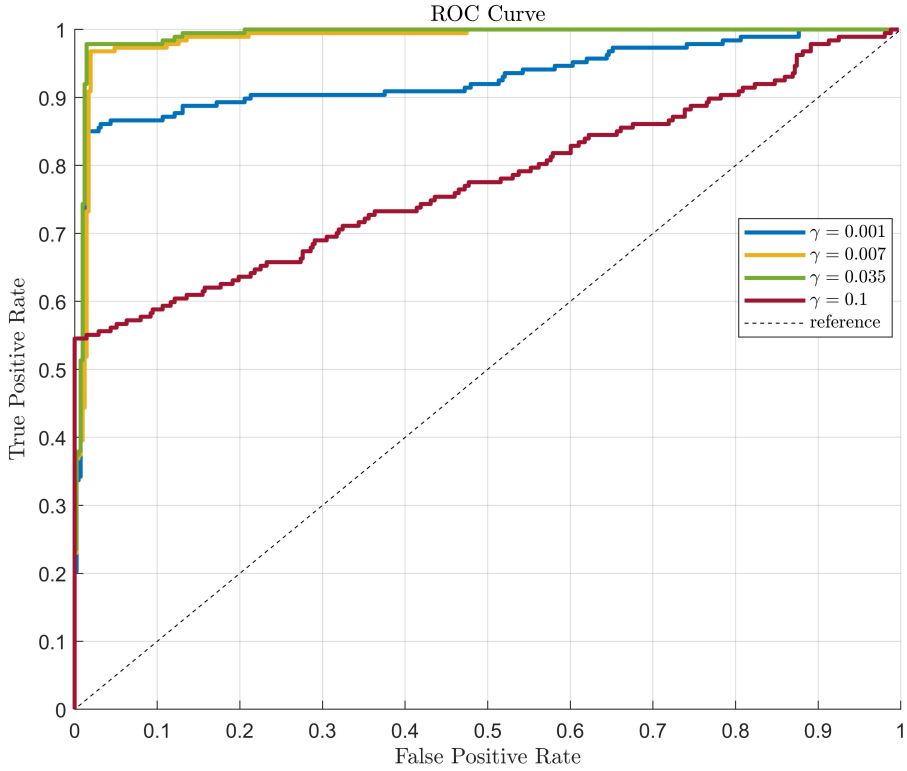

Figure 17: ROC Curves vs Different Choices of Gamma γ in SVM(RBF)

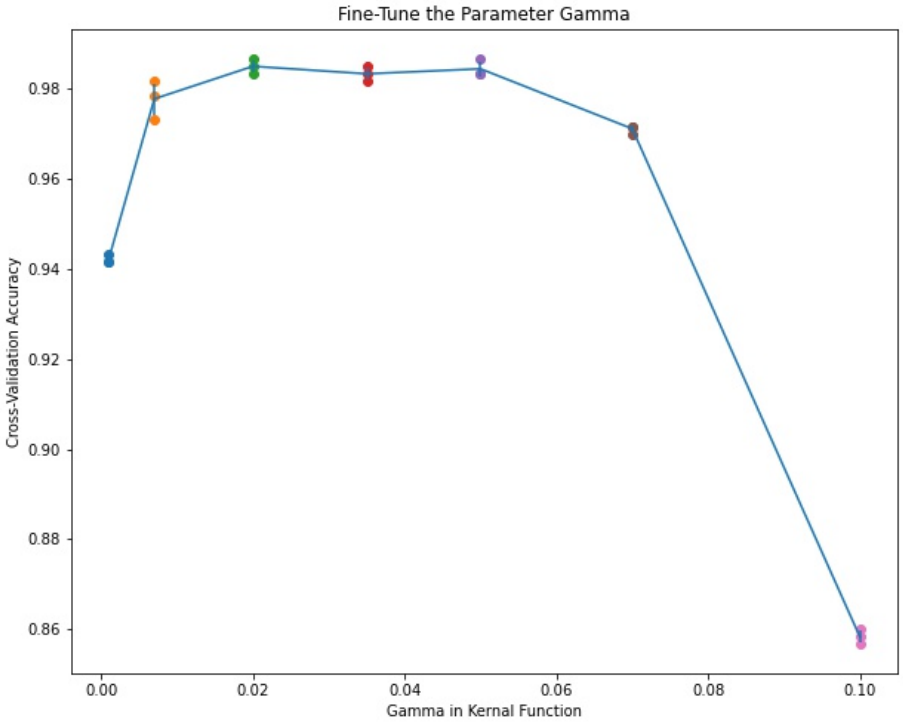

Figure 18: Cross-Validation Accuracy vs Different Choices of Gamma γ in SVM(RBF)