CS303B Artificial Intelligence B
Southern University of Science and Technology
Assignment 2 - Report

Assignment 2: Clustering and Classification on MNIST Dataset

Tian Feng 11910826

Contents

1 Introduction

In Assignment 2, I implemented dim-reduction and clustering to a subset of MNIST dataset with different methods, and compared the performance of two-class classification on this sub-set by SVM and Neural Networks. Here gives a general introduction of this assignment, giving the methods we used and the results obtained briefly.

MNIST dataset is from National Institute of Standards and Technology (NIST), which is an entry-level computer vision dataset. The dataset is composed of a large number of images of handwritten digits from 250 different people. So it can be used to do handwritten number recognition. This assignment is based on its sub-set: images of digits ′1′, ′5′ and ′8′. **Fig. 1** shows several images in this sub-set.

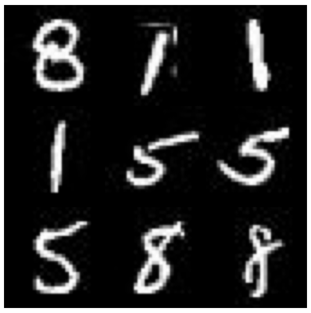


Figure 1: The images of digits in MNIST dataset.

In Part I, I first used **PCA** to reduce the dimensions of each image to **2**. Then I clustered the 2-dimension data points into 3 clusters by **K-Means** and **Hierarchical Clustering**. I rewrote the codes of K-Means method instead of using the built-in MATLAB function. Comparing the clustering results of the two different methods, the accuracy is highest when using cosine distance: **76.17%** (457 / 600) for K-Means with appropriate start points; **75.67%** (454 / 600) for Hierarchical Method using **average linkage**.

In Part II, I repeated the procedure in Part I using **LDA**. The data points after LDA are with larger between-class distance compared to those after PCA. Therefore, the accuracy of clustering can reach **100%** with both K-Means and Hierarchical Clustering.

In Part III, I separated the images of ′5′ from the rest and classified the sub-set with 3 classifiers: **SVM with Linear Kernel**, **SVM with RBF Kernel** and **Neural Network with one hidden layer**. The average accuracy of **5-fold** cross validation is **95.50%**, **98.17%** and **94.83%** respectively, and the average AUCs are **0.9456**, **0.9795** and **0.9849**. In addition, I tuned the RBF kernel parameter γ to improve the classifier performance. The appropriate range of γ is approximately **from 0.001 to no more than 0.1**.

2 Assignment Requirements

2.1 Part I: Dim-reduction by PCA and Clustering

Students are required to implement PCA method to the data and use methods learned in this semester to cluster the data after dim-reduction. The dimension of data should be reduced to 2 using the first two principal components. Students also need to show the clustering results by a scatter plot, and evaluate the results of clustering, i.e., whether images of the same digit are clustered into one category.

2.2 Part II: Dim-reduction by LDA and Clustering

The requirements of Part II are similar to Part I. The only difference is that PCA should be replaced by LDA. The dimension still needs to be reduced to two. Students also need to draw the same kind of scatter plot to compare the performance of the two different methods.

2.3 Part III: Binary Classification by Three Classifiers

In this part, the images of digit ′5′ should be separated from the images of ′1′ and ′8′. With the one-against-rest strategy, a two-class classification problem can be formed. Students should use SVM with linear kernel, SVM with RBF kernel and Neural Network classifier with one hidden layer to solve the problem, and compare the performance of the three classifiers with 5-fold cross validation test. The ROC curves and corresponding AUCs of the results from different classifiers should be provided to support the analysis. In addition, students need to choose at least one parameter of SVM and show how to tune it to obtain more accurate classification results.

3 Methodology

3.1 Dimension Reduction

Overview Dim-reduction is a method of pre-processing high-dimensional data. It is to retain some of the most important features of high dimensional data, and achieve the purpose of improving data processing efficiency.In computer science research, the data to be analyzed might contain high-dimensional variables. High-dimensional datasets will undoubtedly provide adequate information for research, but they also increase the complexity of data analysis to a certain extent. So it is often necessary to do dim-reduction to data. One option is analyze each variable separately. However, in this way, the correlation between the variables is ignored and a lot of useful information might be lost.

Therefore, it is necessary to find some reasonable ways to minimize the loss of original data information while reducing the dimensions. Since there are correlations between the variables, it is possible to transform closely correlated variables into new low-dimensional variables, so that these new variables are uncorrelated with each other, which is the basic principle of the dim-reduction methods used in this assignment: **PCA** and **LDA**. Detailed principles and implementations are discussed below.

3.1.1 PCA (Principal Components Analysis)

Principles PCA (Principal Component Analysis) is one of the most widely used data dim-reduction algorithms. The main concept of PCA is to **retain only the feature dimensions that generate most of the variance**, which is a new orthogonal axes set also known as principal components [1].

The principle of PCA is to sequentially find a set of orthogonal axes from the original space: The first new axis is the vector with the largest variance for the original data, the second is the vector with the largest variance in the space orthogonal to the first axis, and the rest can be done in the same manner to get n new axes. Then find out the first k axes with most of the variance, i.e., the axes after which generate almost 0 variance. These k axes are the principal components of the data. And the original data are replaced with their projections on the principal components [1].

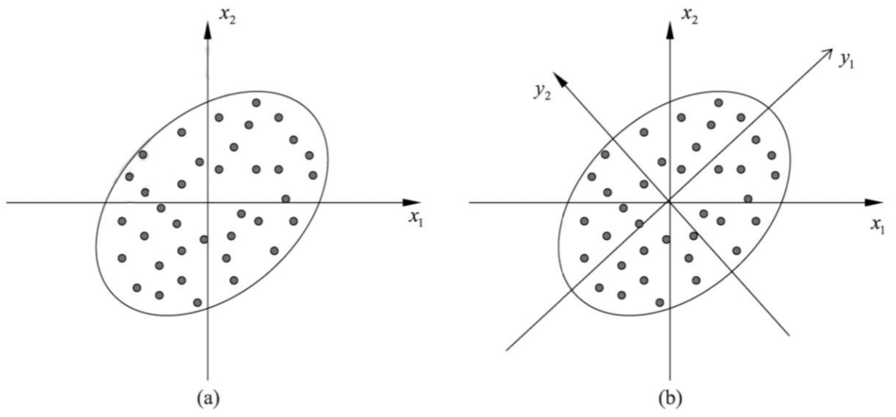


Figure 2: (Principal Component Analysis) Map the data points above from $x_1 - x_2$ space to $y_1 - y_2$ space so that data projections on y_1 axis has the largest variance.

To find the components resulting in maximum variability, we can compute the **covariance matrix** of the data after normalization as **Eq. 1**, and then compute the **eigenvalues and eigenvectors** of the covariance matrix.

$$Cov\left(X\right) = E\left[\left(X - E\left(X\right)\right)\left(X - E\left(X\right)\right)^T\right] \tag{1}$$

The k eigenvectors with the largest eigenvalues (i.e., the largest k variances) are the first k principal components, and we can compute the projections of data on these components. In this way, the data vectors can be mapped into a low-dimensional new space.

Therefore, the implementation of PCA on dataset $X = \{x_1, x_2, x_3, ..., x_n\}$ can be divided into following 6 steps:

* Normalize the data.

* Decentralization, i.e., each variable subtracts its mean value $X = X - E\left(X\right)$.

* Calculate the covariance matrix $C = \frac{1}{n-1}XX^T$.
* Find the eigenvalues and eigenvectors of the covariance matrix C.
* Select the k eigenvectors with the largest eigenvalues as the principal components.
* Compute data projections on these k eigenvectors as the dim-reduced data.

Codes The MATLAB codes to implement PCA are as following:

```matlab
%%Normalize
imgs = images;
for i=1:size(images,1)
    data = images(i,:);
%     data = (data-min(data(:)))./(max(data(:))-min(data(:))+0.001);
    [data,~] = mapminmax(data,0,1);
    imgs(i,:) = data;
end

%%Compute Mean Value Matrix
E = mean(imgs,2);
E = repmat(E, 1, size(images, 2));

%%Decentralize
imgs = imgs - E;

%%Compute Covariance Matrix
%C = cov(imgs.');
C = (1/size(images, 2)-1) * (imgs * imgs');

%%Compute Eigenvectors and Eigenvalues, i.e., principal components
[eigvcts, eigvals] = eigs(C);

%%Find the First Two Principal Components
eigmaxs = max(abs(eigvals));
[prinpval_1, prinploc_1] = max(eigmaxs);
eigmaxs(prinploc_1) = 0;
[prinpval_2, prinploc_2] = max(eigmaxs);
prinpvct_1 = eigvcts(:,prinploc_1);
prinpvct_2 = eigvcts(:,prinploc_2);

%%Compute Projections to the First Two Principal Components
proj_1 = prinpvct_1.' * imgs;
proj_2 = prinpvct_2.' * imgs;
```

Limitations In the process of reducing the correlations between variables, PCA assumes that the correlations are linear, so it cannot generate good results for nonlinear correlations [2]; If the researcher has some prior information about the data samples, for example, the labels of each sample are actually known in this assignment, but PCA cannot be intervened by the information, then the dim-reduction result might not be ideal, and the efficiency of data analysis will also be reduced.

3.1.2 LDA (Linear Discriminant Analysis)

Principles Different from PCA to maximize variance of the whole dim-reduced data, the concept of LDA (Linear Discriminant Analysis) is to map data into low-dimensional space so that data in the same class are as compact as possible, and different classes are as scattered as possible, i.e., **maximizing between-class distance and minimizing within-class distance**, as **Fig. 3** shows. And LDA is a supervised dim-reduction algorithm, while PCA is usually unsupervised [3].

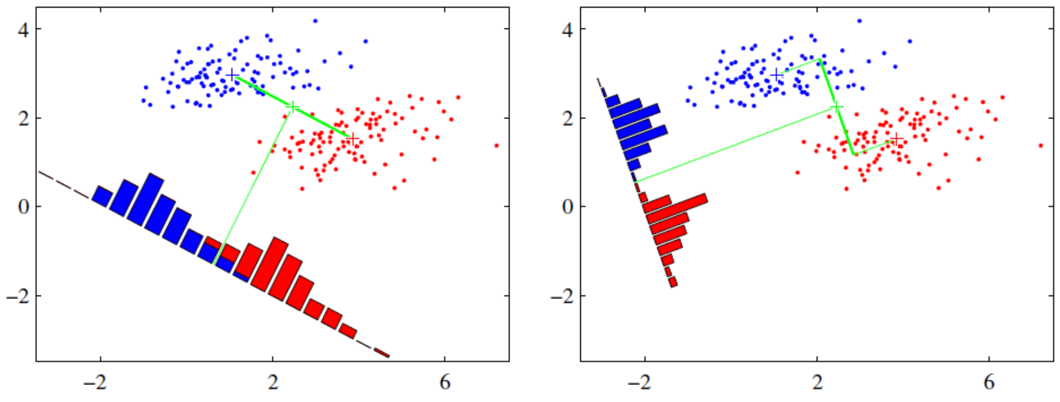


Figure 3: (Linear Discriminant Analysis) Map the data points to a line with maximum between-class distance and minimum within-class distance.

Denote that the dataset $D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n)\}$, where $y_i \in C_1, C_2, C_3, ..., C_k$ is the class of x_i. We define that X_j is the collection of data in class j, u_j is the average of class j, and u is the average of all classes. To derive the principle of LDA, we also define **between-class scatter matrix** S_b and **within-class scatter matrix** S_w as following. S_b is related to between-class distance, while S_w is related to within-class distance.

$$S_b = \sum_{j=1}^{k} (u_j - u)(u_j - u)^T$$

$$S_w = \sum_{j=1}^{k} \sum_{x \in X_j} (x - u_j)(x - u_j)^T$$

Denote that the dimension of the target low-dimensional space is d and corresponding basis vectors are $w_1, w_2, ..., w_d$ which constitute matrix W. To find the W that maximizes between-class distance and minimizes within-class distance in the meantime, we form an optimization problem [3]:

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

To solve this problem, we can assume that $W^T S_w W = 1$, then the optimization problem is transformed into **Eq. 2**.

$$\begin{aligned} arg\ \max\ W^T S_b W \\ s.t.\ \ W^T S_w W = 1 \end{aligned} \tag{2}$$

Then apply Lagrange Multiplier method:

$$\begin{aligned} L\left(W, \lambda\right) &= W^T S_b W + \lambda\left(1 - W^T S_w W\right) \\ \frac{\partial L}{\partial W} &= S_b W + {S_b}^T W - \lambda\left(S_w W + {S_w}^T W\right) \\ &= 2\left(S_b W - \lambda S_w W\right) \\ &= 0 \\ &\Rightarrow\ S_b W = \lambda S_w W \\ &\Rightarrow {S_w}^{-1} S_b W = \lambda W \end{aligned} \tag{3}$$

From **Eq. 3**, we can find that w is the eigenvector of ${S_w}^{-1} S_b$. However, mostly S_w is singular (i.e, having no inverse), for $n < d$. So we need to plus a βI (β is a small quantity almost 0) after S_w to make it non-singular.

To summarize, the procedure of LDA can be divided into 7 steps:

* Normalization
* Compute the average for each class and the whole data.
* Compute between-class scatter matrix S_b.
* Compute within-class scatter matrix S_w.
* Find the eigenvalues and eigenvectors of ${S_w}^{-1} S_b$ or $(S_w + \beta I)^{-1} S_b$.
* Select the d eigenvectors with the largest eigenvalues as basis vectors.
* Calculate data projections on the basis vectors.

Codes The MATLAB codes for LDA are as following:

```
%%Normalize
for i=1:size(images,1)
    data = images(i,:);
    [data,~] = mapminmax(data,0,1);
    images(i,:) = data;
end

%%Classify Images According to '1' '5' '8'
digit_one = []; count_one = 0;
digit_five = []; count_five = 0;
digit_eight = []; count_eight = 0;
for i=1:600
    if(labels(i) == 1)
        count_one = count_one+1;
        digit_one(:, count_one) = images(:, i);
    elseif(labels(i) == 5)
        count_five = count_five+1;
        digit_five(:, count_five) = images(:, i);
    elseif(labels(i) == 8)
        count_eight = count_eight+1;
        digit_eight(:, count_eight) = images(:, i);
    end
end

%%Compute Average of all Classes
C = mean(images,2);
%%Compute Average of Each Class
C1 = mean(digit_one,2);
C5 = mean(digit_five,2);
C8 = mean(digit_eight,2);

%%Compute between-class scatter
Mb = [C1-C C5-C C8-C];
Sb = Mb*(Mb.');

%%Compute between-class scatter
Mw = [digit_one-C1 digit_five-C5 digit_eight-C8];
Sw = Mw*(Mw.');

%%Solve eigenvalue problem of inv(Sw)*Sb
I = eye(size(Sw));
[eigvcts, eigvals] = eigs((Sw+0.0001*I)\Sb);

%%Find w1 & w2
eigmaxs = max(abs(eigvals));
[~, wloc_1] = max(eigmaxs);
eigmaxs(wloc_1) = 0;
[~, wloc_2] = max(eigmaxs);
w_1 = eigvcts(:,wloc_1);
w_2 = eigvcts(:,wloc_2);
```

```
%%Compute Projections to w1 & w2
E = repmat(C, 1, size(images, 2));
imgs = images - E;
proj_1 = w_1.' * imgs;
proj_2 = w_2.' * imgs;
```

Advantages & Limitations Because of the consideration of between-class distances, generally there are clear boundaries between different classes of dim-reduced data by LDA. It allows researchers to easily distinguish between different classes of samples. However, LDA requires that the label of samples are known, so not all data in science research can be processed by it. In addition, there are two assumptions for LDA [3]:

* The original data are classified according to their average.
* Different data classes have the same covariance matrix.

Therefore, when the difference between two different classes is mainly reflected in the variance rather than their average, the dim-reduction result of LDA will not be well, like shown in the following figure.

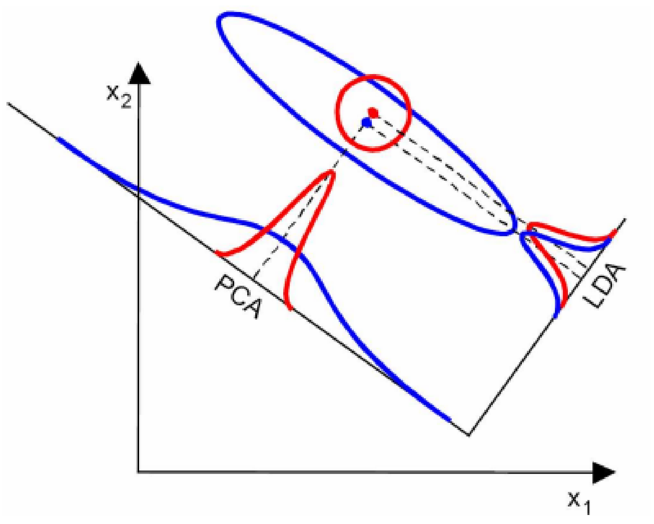


Figure 4: Bad performance of LDA when the means of two different classes are close.

3.2 Clustering

Overview In the field of Machine Learning, depending on whether the data contain labels, problems can be divided into supervised and unsupervised. Clustering is a typical unsupervised learning problems, for the samples contain the characteristics of different categories but do not have labels. Even if no label is provided, the data can be automatically divided into several clusters according to the regularity of their characteristics. In this assignment, we used two common clustering algorithms: **K-Means** and **Hierarchical Clustering**.

3.2.1 K-Means

Principles K-Means algorithm is a widely used clustering algorithm, which is an unsupervised algorithm that cluster data by similarity. It uses distance as the measurement criterion of similarity, i.e., the smaller distance between two data points, the more likely they are in the same cluster. The algorithm is called K-Means because it can divide the data into k different clusters, and the center of each cluster is calculated with the **mean** of contained data values. The main concept of K-Means is that it **continuously iterates the centers of clusters until they remain unchanged** [4]. **Fig. 5** shows the procedure of K-Means.

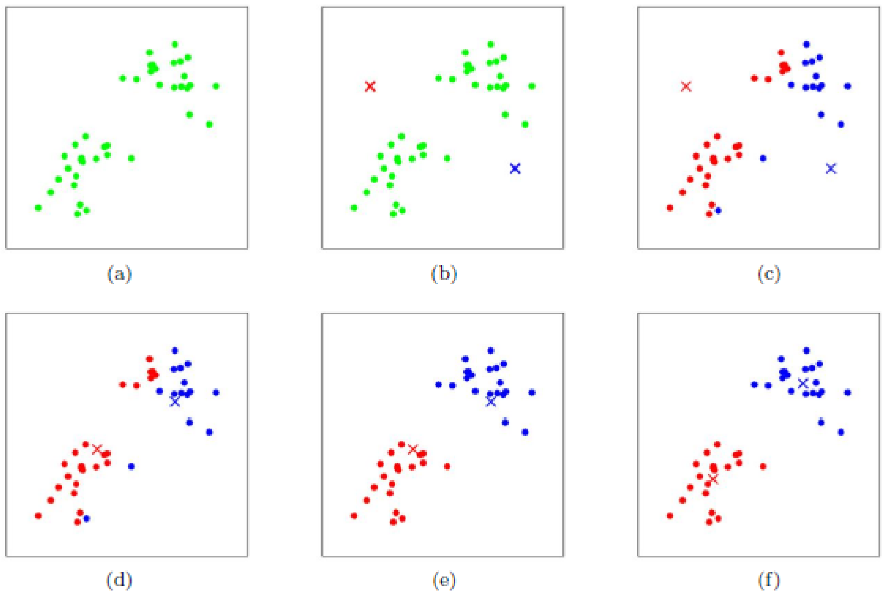


Figure 5: Keep iterating the centers to cluster the data until centers are unchanged anymore.

The steps of K-Means are as following:

* Randomly select k samples as the initial cluster centers.
* Compute the distances between samples and the k centers, and sort a sample into the cluster with nearest center.
* Calculate the new center for each cluster.
* Repeat steps 2 and 3 until the centers keep unchanged or the max iterations number is reached.

Codes Although there is built-in function to implement K-Means in MATLAB, I did not use it. I wrote the codes for K-Means by myself, which are shown below:

```
%%Randomly Set Initial Centers
clust1 = [];clust2 = [];clust3 = [];
```

```
centr = [data(unidrnd(size(data,1)),:); data(unidrnd(size(data,1)),:);
 ↪  data(unidrnd(size(data,1)),:)];
centr_new = [[]; []; []];

dist = pdist2(centr, data);
[~,class] = min(dist);
clust1 = data(class==1,:); clust2 = data(class==2,:); clust3 =
 ↪  data(class==3,:);
centr_new = [mean(clust1); mean(clust2); mean(clust3)];

%%Iteration (Stop When New Centers Unchange)
while(~isequal(centr, centr_new))
    centr = centr_new;
    dist = pdist2(centr, data,'cosine');
    [~,class] = min(dist);
    clust1 = data(class==1,:); clust2 = data(class==2,:); clust3 =
 ↪  data(class==3,:);
    centr_new = [mean(clust1); mean(clust2); mean(clust3)];
end
```

Advantages & Limitations From above introduction, we can see that K-Means is a clustering method very easy to understand and implement. However, since the start points are chosen randomly, the clustering result is not robust and sometimes the accuracy can be extremely low and the final centers even do not have different labels for some start points [4]. Therefore, it is important for K-Means to start from appropriate data points.

3.2.2 Hierarchical Clustering

Principles Hierarchical Clustering is a quite intuitive clustering algorithm. Its main idea is to first consider each sample as a cluster, then **find two clusters with the smallest linkage distance and merge them**, repeating until the expected number of clusters is obtained [5]. The methods of calculating the distance between two clusters are mainly 3 types.

* **Complete Linkage**: The minimum distance between two data points in two clusters is considered as the distance between the clusters.

* **Single Linkage**: The maximum distance between two data points in two clusters is considered as the distance between the clusters.

* **Average Linkage**: For each data in one cluster, calculate the distances from all data in another cluster. The average of all distances is the distance between two clusters.

Complete Linkage and Single Linkage are susceptible to extreme data (e.g., some data are too close or far), which are suitable for some special cases [5]. For general cases, Average Linkage can generate better results, while it requires a larger computation amount.

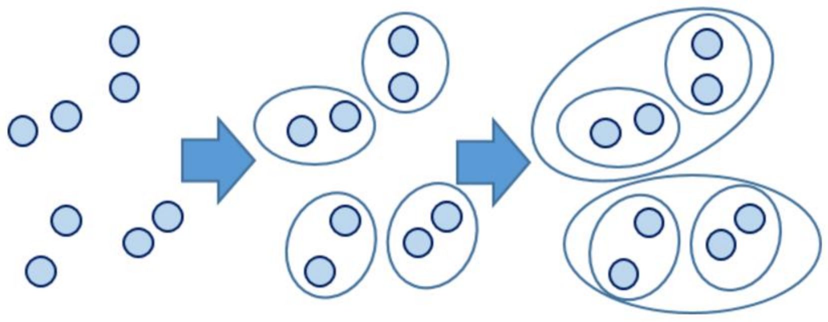


Figure 6: The nearest two clusters become one new cluster in each step.

Codes The codes of Hierarchical Clustering are as following:

```
data = [proj_1 proj_2]; clear proj_1 proj_2
clust = [];

%%Write Hierarchical Clustering Codes Myself (Though Failed)
% Dist = pdist2(data,data);
% dist = Dist;
% for i = 1:length
%     for j = i:length
%         dist(j,i) = Inf;
%     end
%     Dist(i,i) = Inf;
% end
%
% clust_num = length;
% while(~(clust_num == 3 || clust_num < 3))
%     [r,c] = find(dist == min(min(dist)));
%     r = r(1);c=c(1);
%     if(ismember(r,clust))
%         [label,~] = find(clust == r);
%         clust(label,end+1) = c;
%     elseif(ismember(c,clust))
%         [label,~] = find(clust == c);
%         clust(label,end+1) =  r;
%     else
%         clust = [clust;r c];
%     end
%
%     [label,~] = find(clust == r)
```

```
%
%     inter_Dist = Dist(clust(label,1),:);
%     for i = 2:size(clust(label,:),2)
%         inter_Dist = inter_Dist + Dist(clust(label,i),:);
%     end
%     inter_dist = inter_Dist ./ size(clust(label,:),2);
%
%     for i = 1:size(clust(label,:),2)
%         Dist(clust(label,i),:) = inter_dist;
%         Dist(:,clust(label,i)) = inter_dist.';
%     end
%
%     dist = Dist;
%     for i = 1:length
%         for j = i:length
%             dist(j,i) = Inf;
%         end
%     end
%     clust_num = clust_num-1;
% end

%%Use Built-in Function
dist = pdist(data,'cosine');
%%Use Average Linkage
process = linkage(dist,'average');
correlation = cophenet(process,dist);

%%Expected Cluster Number = 3
clust = cluster(process,'maxclust',3);
```

Advantages & Limitations Usually, for a dataset the results of Hierarchical Clustering are the same, which is quite stable. However, the method requires a large computation amount.

3.3 Classification

Overview Classification is one of the most common problems in machine learning research. It is a supervised learning problem. To solve this kind of problem, it is generally required to construct a model as a classifier. The main method to obtain the classifier is to train the model on a dataset with labels. Then the model can learn from the data and predict the class of new data. The performance of a classifier can be tested by **Cross Validation** and shown by certain ways, e.g., **ROC curve** and **AUC**. In this assignment, we use **SVM** and **Neural Network** as classifiers, both of which are very classic models in machine learning.

3.3.1 SVM with Linear Kernel

Principles SVM (Support Vector Machine) is a binary classification model whose basic model is a linear classifier defined with the largest interval in the feature space of the data. Actually, according to different kernel used, SVM models can be generally divided into two types: linear and nonlinear. Here first gives a brief introduction of linear SVM. The basic concept of linear SVM is to **find a separating hyperplane** that can correctly divide the training data and has the **largest interval** [6].

As shown in **Fig. 7**, $w^T x + b = 0$ is a separating hyperplane. What we need to do is to find the parameters w and b to maximize the distances from the data points in different classes to the hyperplane. Then more accurate results might be obtained when classifying the data based on the position of data coordinates relative to the hyperplane. When the training data is **linearly separable**, i.e., the data points can be completely separated by a hyperplane, we can accurately find such a hyperplane [6].

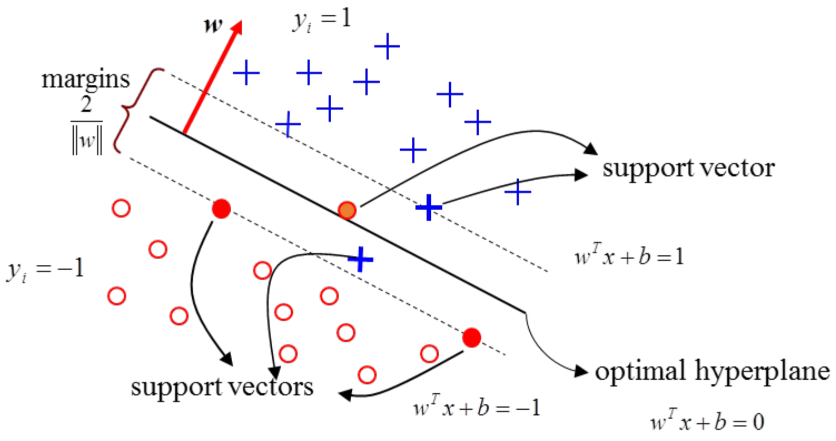


Figure 7: For the case in the figure, $w^T x + b = 0$ is the optimal separating hyperplane of the data; The parameters of hyperplane are related to support vectors; Margin is the distance of the SVM boundary constrained by support vectors.

It is not difficult to understand that we only need to consider the data points which are nearest to the separating hyperplane in **Fig. 7**. And these data are called **Support Vector**, from which the name of SVM comes.

To derive the optimization problem function, we assume the separating hyperplane is $w^T x + b = 0$. Then the distance from data points to the hyperplane is:

$$\frac{|w^T x + b|}{\|w\|}$$

Denote that the distance from the support vector to the hyperplane is d, and the class $y \in \{0, 1\}$. We can obtain the formula:

$$\begin{cases} \frac{\left|w^T x+b\right|}{\|w\|} \geqslant d \quad for\ y=1 \\ \frac{\left|w^T x+b\right|}{\|w\|} \leqslant -d \quad for\ y=0 \end{cases} \Rightarrow \begin{cases} \frac{\left|w^T x+b\right|}{\|w\| d} \geqslant 1 \quad for\ y=1 \\ \frac{\left|w^T x+b\right|}{\|w\| d} \leqslant -1 \quad for\ y=0 \end{cases}$$

If we let $\|w\|\, d = 1$ (without influence on the solution) and merge the two formulas. We can get **Eq. 4** [7]:

$$y\left(w^T x + b\right) \geqslant 1 \tag{4}$$

And for support vectors, we have:

$$y\left(w^T x + b\right) = 1$$

So the function of margin shown in **Fig. 7** can be derived:

$$margin = \frac{|1-(-1)|}{\|w\|} = \frac{2}{\|w\|}$$

To maximize the margin, the optimization problem can be formulated as [7]:

$$\begin{aligned} \max \frac{2}{\|w\|} &\Rightarrow \min \frac{\|w\|}{2} \\ &\Rightarrow \min \frac{1}{2}\|w\|^2 \\ s.t. \quad y_i\left(w^T x_i + b\right) &\geqslant 1 \end{aligned} \tag{5}$$

Apply Lagrange Multiplier method to solve the optimization problem in **Eq. 5**. The results are [6]:

$$\begin{aligned} w &= \sum_{i=1}^{n} \lambda_i y_i x_i \\ b &= \frac{1}{|S|}\sum_{s\in S}\left(y_s - w x_s\right) \end{aligned}$$

In the formulas above, n is the data number, S is the set of support vectors and $|S|$ is the number of support vectors. Only when x_i is a support vector, $\lambda_i > 0$; $\lambda_i = 0$ in the other case. The values of λ_is can be found by **Sequential Minimal Optimization (SMO)** algorithm. With w and b being known, the trained SVM model has been obtained [6].

Codes MATLAB provides built-in tools for implementing SVM. Here we choose to use *fitcsvm* and other supporting functions to train SVM model and do cross validation:

```
%%Prepare data
load('../mnist-1-5-8.mat');
[length,~] = size(images);
imgs = images.';
%%Normalization
for i=1:length
    data = images(i,:);
    [data,~] = mapminmax(data,0,1);
    imgs(:,i) = data;
end

%%Prepare the vectors for '5' against the rest
class = zeros(size(labels));
for i=1:size(class)
    class(i) = (labels(i)==5);
    %1 means '5'; '0' means others
end

%%Train SVM using a linear kernel
model = fitcsvm(imgs, class, 'KernelFunction','linear','KernelScale',1);
model.ScoreTransform = 'doublelogit';
```

Limitations Linear SVM is suitable for datasets where most samples are linearly separable. However, when the data are nonlinear, the accuracy of the classification can be extremely low. This problem can be solved by introducing kernel functions to construct nonlinear SVM [7]. A kind of nonlinear SVM with RBF kernel will be discussed in the next subsection.

3.3.2 SVM with RBF Kernel

Principles The main idea to classify nonlinear data using SVM is mapping them from the current space to a higher-dimensional linearly separable space as shown in **Fig. 8**. We let $\mathit{\Phi}\left(x\right)$ to represent the mapping of x in higher-dimensional space [8]. Then find the separating hyperplane:

$$f\left(x\right) = w\mathit{\Phi}\left(x\right) + b$$

Now we can solve the optimization problem using the same method as linear SVM, and the result is as [8]:

$$f\left(x\right) = \sum_{i=1}^{n} \lambda_i y_i \mathit{\Phi}\left(x\right)^T \mathit{\Phi}\left(x\right) + b$$

The next steps are to compute λ, then w and b can be found with the similar function to linear case. In the procedure to find solutions, we need to compute the dot product of $\mathit{\Phi}\left(x_i\right)$ and $\mathit{\Phi}\left(x_j\right)$. However, the computations of dot product in high-dimensional

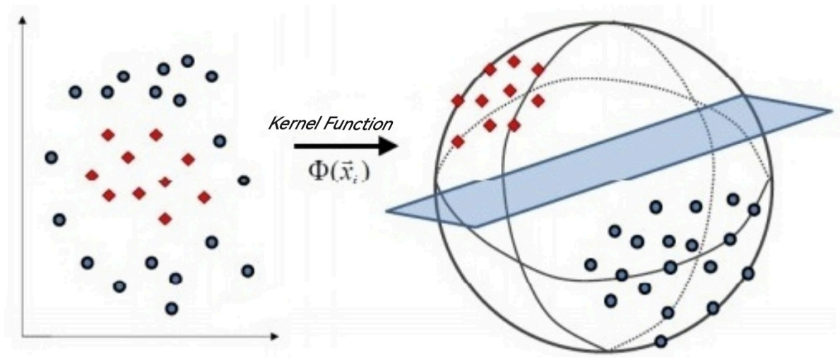


Figure 8: The nonlinear data can become linearly separable after being mapped to a higher-dimensional space by some kernel functions.

can be very complex. Therefore, **kernel function** is introduced [8]. The dot product of $\Phi(x_i)$ and $\Phi(x_j)$ equals to the result computed by the kernel function $\kappa(x_i, x_j)$:

$$\kappa(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$$

Therefore, we only need to compute $\kappa(x_i, x_j)$ instead of the dot product [8], which considerably reduces the computation complexity:

$$f(x) = \sum_{i=1}^{n} \lambda_i y_i \kappa(x_i, x_j) + b \tag{6}$$

It can be demonstrated that when the kernel matrix K (shown below) is always **positive semi-definite** for any x, $\kappa(x_i, x_j)$ can be used as a kernel function [8].

$$K = \begin{bmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \cdots & \kappa(x_1, x_n) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \cdots & \kappa(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(x_n, x_1) & \kappa(x_n, x_2) & \cdots & \kappa(x_n, x_n) \end{bmatrix}$$

In this assignment, the kernel function used is RBF (Radial Basis Function) kernel, which is also known as Gaussian kernel [9]:

$$\kappa = \exp\left(-\gamma \|x_i - x_j\|^2\right), \quad \gamma > 0 \tag{7}$$

It is obvious that the kernel matrix of it is always positive semi-definite, for it is an exponential function, i.e., it is a proper kernel. The basic principle of RBF kernel is to **interpolate the samples on a higher-dimensional Gaussian hyper-surface** [9]. By substituting the kernel function into **Eq. 6**, we can have a SVM model that can classify nonlinear data.

The parameter γ is important to the performance of SVM with RBF kernel. It determines the influence of each single support vector on the hyperplane: The smaller γ, the larger the influence. It can be considered as **the reciprocal of the influence radius** the sample selected as the support vector [9].

If γ is too small, the influence region of each support vector will contain almost the entire training dataset. The model will behave like a high-density linear hyperplane.

If γ is too large, the radius of the influence region of the support vector will be so small that it can only affect itself. In this case, the model will be over-fitting, and will get low accuracy when predicting for new data.

Therefore, it is important to find an appropriate γ. Generally, the γ value defaults to *1/the number of sample features*.

Codes The codes for SVM with RBF kernel are similar to linear SVM. We can change the kernel function by changing the corresponding parameter in *fitcsvm*:

```
%%Prepare data
load('../mnist-1-5-8.mat');
[length,~] = size(images);
imgs = images.';
%%Normalization
for i=1:length
    data = images(i,:);
    [data,~] = mapminmax(data,0,1);
    imgs(:,i) = data;
end

%%Prepare the vectors for '5' against the rest
class = zeros(size(labels));
for i=1:size(class)
    class(i) = (labels(i)==5);
    %1 means '5'; '0' means others
end

%%Train SVM using a linear kernel
gamma = 1/784;
model = fitcsvm(imgs, class,
↪  'KernelFunction','rbf','KernelScale',1/sqrt(gamma));
model.ScoreTransform = 'doublelogit';
```

3.3.3 Neural Network

Principles Neural Networks (NN) are computational models that mimic the structure and function of biological neural networks. The models do computations using a large number of artificial neurons, and they can change the internal parameters on the basis of external information [10]. The structure of a simple neural network with one hidden layer is shown in **Fig. 9**. In present machine learning research, neural networks are mainstream classifiers. Many powerful models, e.g., Convolutional Neural

Networks (CNN), have been developed based on them [10]. Here only gives an brief introduction of the basic neural networks.

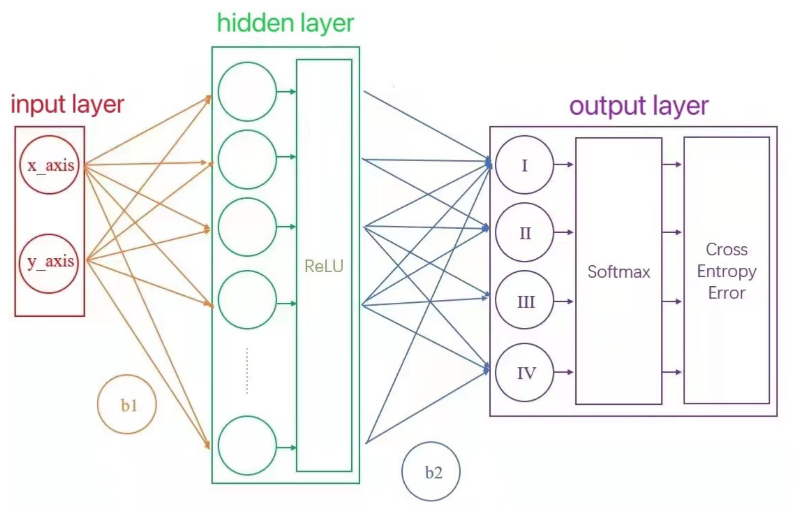


Figure 9: The computational process of a one hidden layer neural network on a 2D dataset.

The computational process of neural networks can be described with following steps [11].

* Normalize data values.
* Transfer the data from the input layer to the first hidden layer.
* Weight each input in the hidden layer, and then use the activation function to compute the output in the activation layer.
* Transfer the output to the next hidden layer and repeat step 3, or directly to the output layer.
* In the output layer, compute the probabilities belonging to each class and cross entropy error.
* Iteratively optimize parameters based on cross entropy error until the maximum number of iterations or expected error is reached.

The structure of a computational neuron in hidden layer is as following figure. We can see that the neuron computes the output of weighted inputs by activation function [11].

Activation functions are used to make nonlinear results for the network computations. A frequently used activation function is **ReLU**, which is [11]:

$$f(x) = \max(0, x)$$

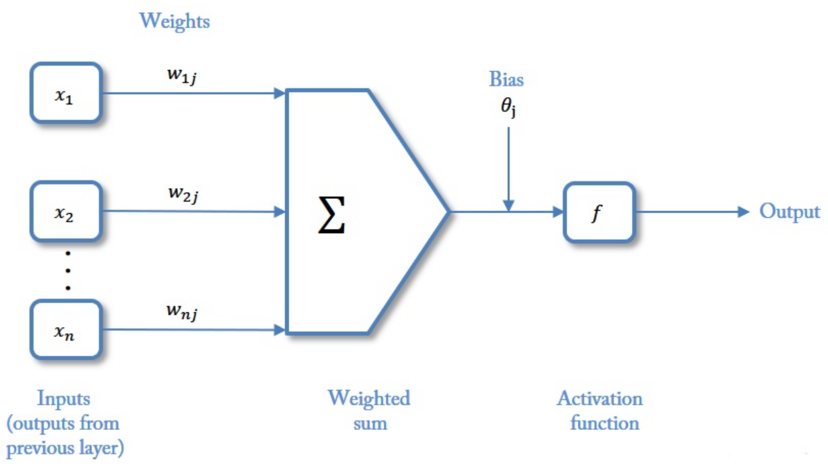


Figure 10: A neuron of hidden layer.

The probabilities or scores of a sample belonging to each class are computed in Softmax layer of output layer [12]. The function is:

$$S_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

(S_i is the score for i class; y_i is the output for i class)

Calculate the exponential powers of all outputs based on e and quotients of each exponential power to their sum to get the probability.

The cross entropy error is the opposite of the logarithm of corresponding probability [12]:

$$error = -\log S_i$$

Cross entropy error is the basis for optimizing parameters (mainly the weights of inputs). A commonly used method is **gradient descent**, which is to compute the gradient of cross entropy error and adjust the parameters in the direction that maximize the reduction of error [12].

Codes We can use the functions in the *Deep Learning Toolbox* to obtain a neural network classifier:

```
%% Configure Net
net = feedforwardnet(10, 'traingd');

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 = 'poslin'; % the transfer function for the first
↪   layer: ReLU
```

```
net.layers{2}.transferFcn = 'softmax'; % the transfer function for the
↪  second layer
net.performFcn = 'crossentropy'; % loss function
%%Adjust Training Parameters
net.trainParam.epochs = 10000;
net.trainParam.lr = 0.1; % learning rate.
%% 5-Fold Validation
for i = 1:5
    trainIdx = cvp.training(i); %% get the index of training samples
    testIdx = cvp.test(i); %% get the index of the test samples
    training_label = class(trainIdx); %% create the training label ground
↪   truth
    training_instance = imgs(trainIdx,:); %% create the training data
    test_label = class(testIdx); %% create the testing label ground truth
    test_instance = imgs(testIdx,:); %% create the test data

    %%Training
    model = train(net, training_instance.', training_label.');

    %%Test
    scores(testIdx,:) = sim(model, test_instance.');

    %%Obtain Labels
    thresh = 0.5;
    predict_labels(scores<thresh,:) = 0;
    predict_labels(scores>=thresh,:) = 1;

    %%Compute Accuracy
    correct_count(:,i) = (sum(predict_labels(testIdx,:)==class(testIdx)));
    accuracy(:,i) = correct_count(:,i) ./ size(test_label);
end
```

3.3.4 Cross Validation

Principles Generally, we cannot use all of data to train a model, otherwise we will not have available data to do validation to evaluate the performance of the model. However, if we divide a part of the data as a test set and do not use it to train the model, it will result in a waste of data information. In order to solve this problem, we can do cross validation. Here introduces the kind of cross validation methods used in this assignment, which is called **K-fold cross validation** [12].

The procedure of K-fold cross validation is that [12]: Divide the dataset into K sets; Each time take one of the sets as test set without repeating, train the model with the other four training sets, and calculate the loss or accuracy of the prediction on the test set; Calculate the average loss or accuracy of the K tests as an evaluation. Usually, K equals to 5 or 10.

Codes The K-fold cross validation can be done in MATLAB with *cvpartition* and other supporting functions:

```
%%Prepare the vectors for '5' against the rest
class = zeros(size(labels));
for i=1:size(class)
    class(i) = (labels(i)==5);
    %1 means '5'; '0' means others
end

cvp = cvpartition(class,'KFold',5);^^I
model = fitcsvm(imgs, class,
 ↪  'KernelFunction','linear','KernelScale',1,'CVPartition',cvp);
```

3.3.5 ROC Curve & AUC

Principles ROC (Receiver Operating Characteristic) curve is a curve used to evaluate the prediction performance of a classifier [13]. To explain its principle, we first need to introduce some basic concepts:

* In binary-classification problems, generally one class is called **positive** class and the other is called **negative** class.
* TP (True Positive): Positive samples that is predicted to be positive.
* FN (False Negative): Positive samples that is predicted to be negative.
* FP (False Positive): Negative samples that is predicted to be positive.
* TN (True Negative): Negative samples that is predicted to be negative.
* TPR (True Positive Rate, also called Sensitivity): TP/(TP+FN)
* FPR (False Positive Rate, also called Specificity): FP/(FP+TN)
* Score: The output of the classifier for a sample belonging to a class, which is equivalent to probability. The higher the score, the larger the probability of being in the class.
* Threshold: Samples with scores below threshold are classified as negative, and above threshold are positive.

ROC curve, with FPR as x-axis and TPR as y-axis, is drawn by going through all thresholds. FPR represents the degree of error of the model, while TPR represents the degree of coverage of the model. So the lower FPR, and the higher TPR (i.e. the closer ROC curve is to the upper left corner of the figure), the better the performance of the model. And because the x and y coordinates of ROC curve are calculated in the positive and negative sample sets respectively, ROC curve cannot be influenced by the imbalance of samples [13].

AUC (Area Under Curve) measures the effect of all possible thresholds synthetically. It can be seen as the probability that the score for a random positive sample is larger than the score of a random negative sample. It needs to notice that if we plot a diagonal between the lower left and the upper right, the AUC is equal to 0.5. The actual meaning of the diagonal is to randomly classify data, so the positive and negative accuracy should be both 50%. When AUC is larger than 0.5, the performance of the model is better than randomly classifying [14]. The criteria for evaluation using AUC are as following:

* 0.5-0.7: Poor performance.
* 0.7-0.85: Just okay performance.
* 0.85-0.95: Quite good performance.
* 0.95-1: Excellent performance.

Codes The function *roccurv* (modified on the basis of some codes from GitHub) is used to obtain ROC curve and AUC:

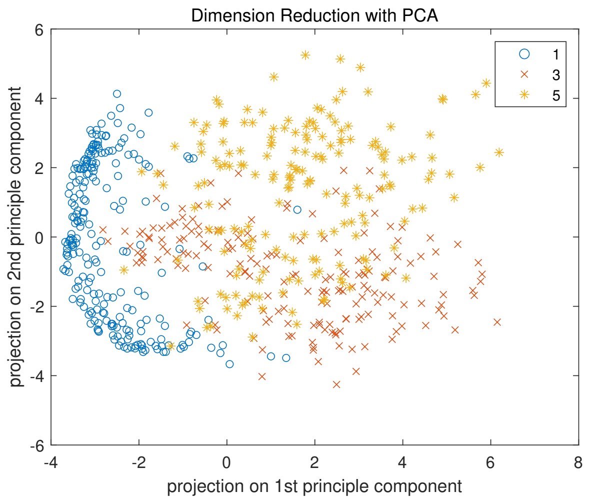
```
function [X, Y, AUC] = roccurv(ground_truth, predict)
    x = 1.0;
    y = 1.0;
    pos_num = sum(ground_truth==1);
    neg_num = sum(ground_truth==0);
    x_step = 1.0/neg_num;
    y_step = 1.0/pos_num;
    [predict,index] = sort(predict);
    ground_truth = ground_truth(index);
    for i=1:length(ground_truth)
        if ground_truth(i) == 1
            y = y - y_step;
        else
            x = x - x_step;
        end
        X(i)=x;
        Y(i)=y;
    end

    %%Return AUC
    AUC = -trapz(X,Y);
end
```

4 Results & Discussion

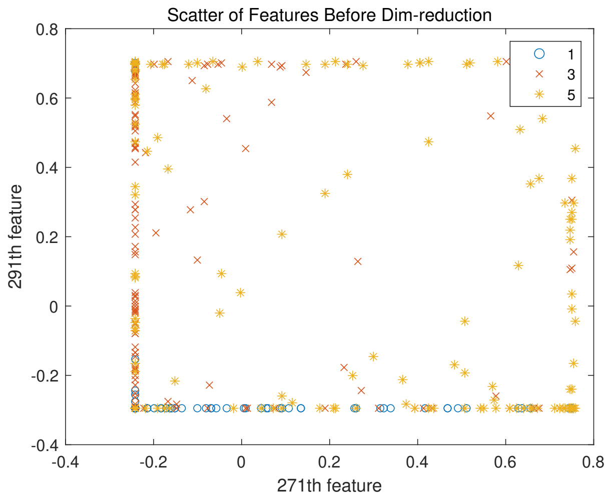
4.1 Part I: Dim-reduction by PCA and Clustering

4.1.1 Reduce Dimension by PCA

Results The 2D data points after PCA is shown in **Fig. 11**. To make comparison, I also show the scatter plot of 271th and 291th features (these two numbers were chosen randomly by the function *unidrnd*) of data before PCA.



(a) after PCA



(b) before PCA

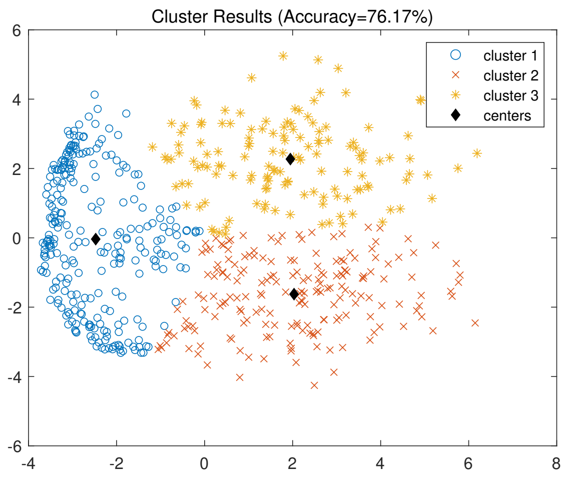
Figure 11: Comparison of the original data and dim-reduced data by PCA.

Discussion From the figure, we can find that the dimension of data has been reduced to 2. Compared with two **random** features of the original data, the data points after PCA have obvious differences in distribution. Although there are still some overlaps between different classes of data (e.g, the overlap of digits '3' and '5' is relatively serious), it is already possible to intuitively distinguish the regions occupied by different classes of data.

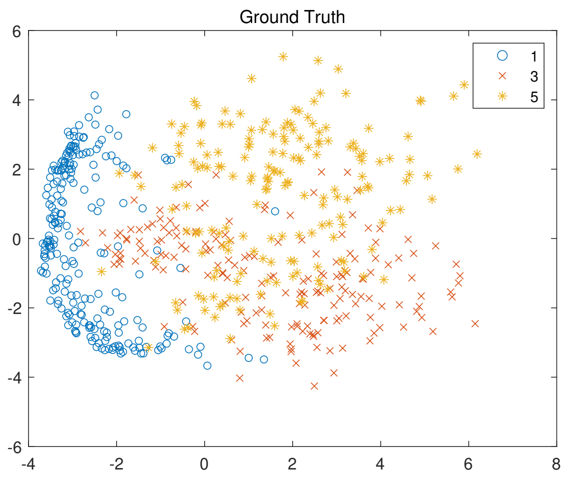
Therefore, two-dimensional features for the original data cannot reflect the differences among different samples, while the two-dimensional data after PCA can still contain most of the feature information and reflect the difference between different samples.

4.1.2 Cluster by K-Means

Results I first used K-Means method to cluster the data after PCA. The scatter of result is shown in **Fig. 12**. Here I used **cosine distance**, can reflect the difference in direction between vectors, to measure the distance between two data points. Because the data has been normalized, it is more reasonable to look for differences among samples by comparing directions.



(a) K-Means result



(b) Ground truth

Figure 12: Comparison of the clustering results by K-Means with appropriate start points and ground truth classes of data after PCA.

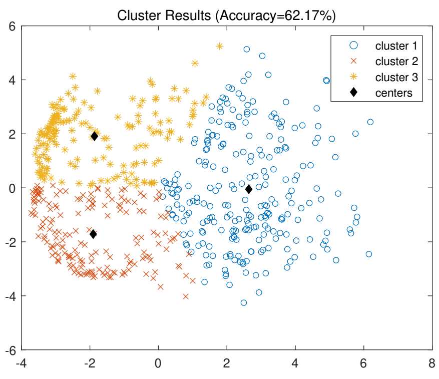
Discussion To quantitatively evaluate the results of clustering, I define the accuracy of clustering as:

$$ACC = \frac{\sum_{i=1}^{k} \max\left(number\ of\ samples\ clustered\ into\ the\ same\ region\ with\ i\ label\right)}{total\ number\ of\ samples}$$

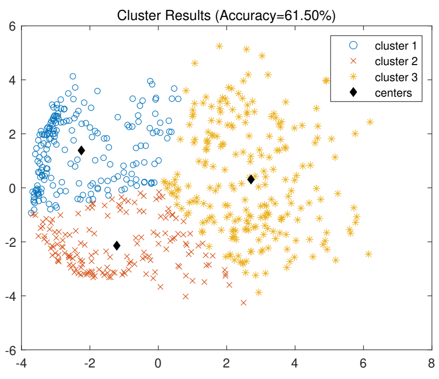
Though the definition of the accuracy is not rigorous and sometimes misleads my evaluation of the result (e.g., if the majority of samples with two labels are clustered into one region, the accuracy calculated with the above formula will be high, but it is obviously unreasonable), it can still reflect the clustering performance to some extent.

The scatter above shows the result when the start points of K-Means are appropriate. So the final centers nicely belongs to 3 classes, i.e., ′1′, ′5′ and ′8′. In this case, there are the most samples of the same class being in the same cluster. The accuracy is **76.17%**, i.e., **457** samples are correctly clustered together with the samples in the same class as them.

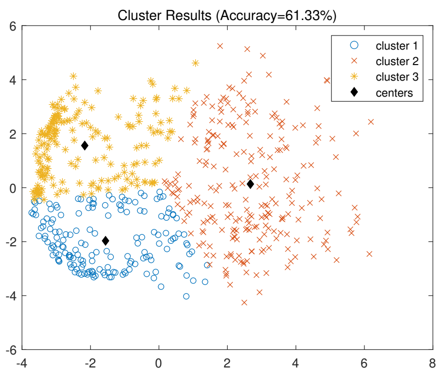
However, as we discuss in the last section, the clustering by K-Means is not robust. If the start points are not good enough, the final centers may not belongs to 3 classes separately and the accuracy will be lower, as the figures below show.



(a) accuracy=62.17%



(b) accuracy=61.50%



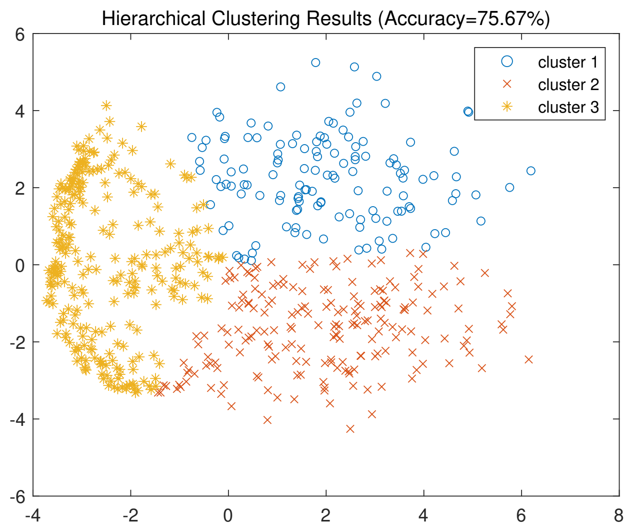
(c) accuracy=61.33%

Figure 13: The labels of final centers in (a) are [′5′,′1′,′1′]; The labels of final centers in (b) are [′1′,′8′,′8′]; The labels of final centers in (c) are [′8′,′5′,′5′].

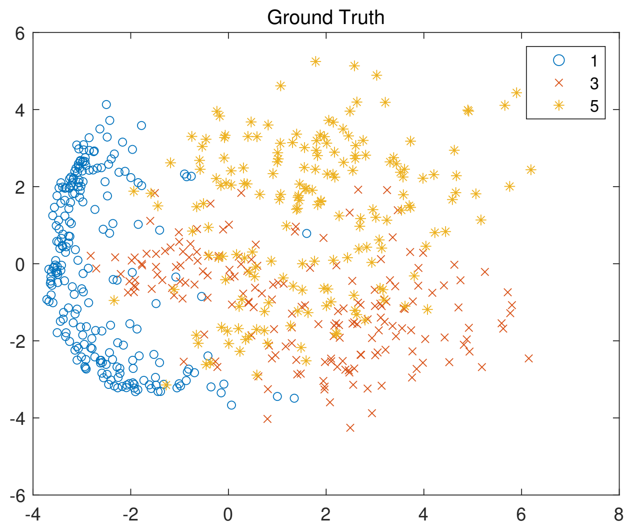
Therefore, I do clustering again using Hierarchical Clustering method.

4.1.3 Hierarchical Clustering

Results I used cosine distance and **average linkage** to compute the distance between two cluster. The results of Hierarchical Clustering are shown below.



(a) Hierarchical Clustering result



(b) Ground truth

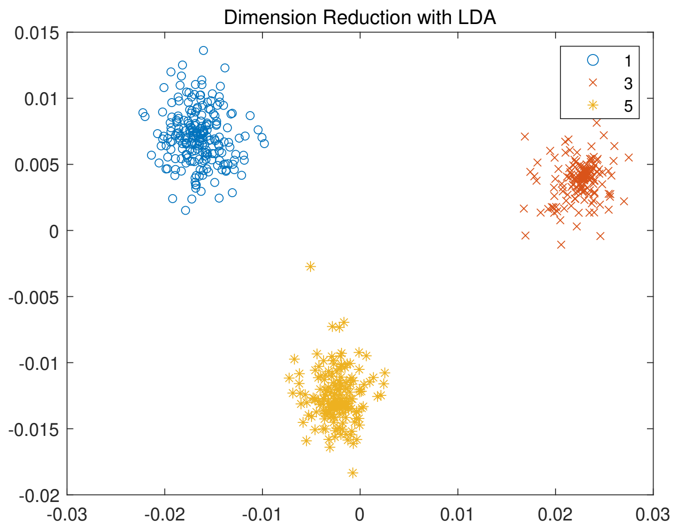
Figure 14: Comparison of the clustering results by Hierarchical Clustering and ground truth classes of data after PCA.

Discussion The accuracy of Hierarchical Clustering to the data after PCA is **75.67%**, i.e. **454** samples, which is quite close to the highest accuracy of K-Means results. And the clustering results are always the same. Therefore, Hierarchical Clustering might be better than K-Means when clustering the data after PCA in this assignment.

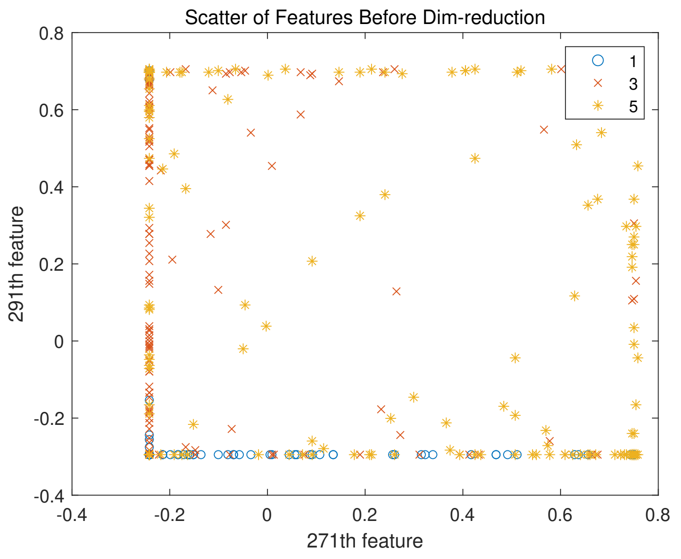
4.2 Part II: Dim-reduction by LDA and Clustering

4.2.1 Reduce Dimension by LDA

Results The dim-reduction result by LDA is shown below.



(a) after LDA



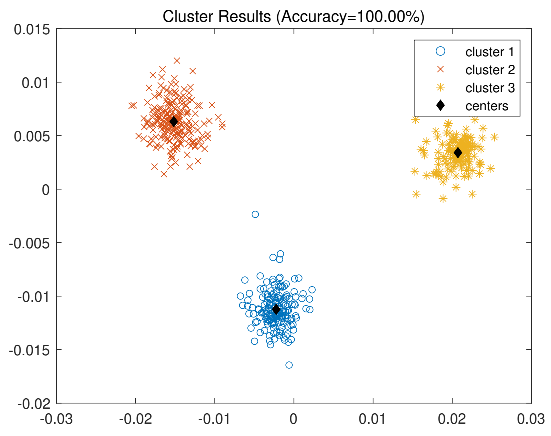
(b) before LDA

Figure 15: Comparison of the dim-reduced results by LDA and the original data.

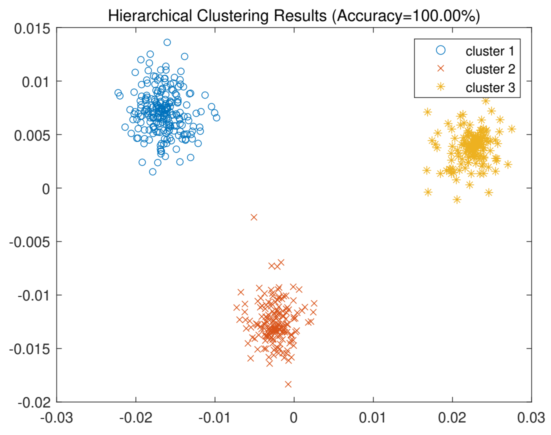
Discussion For the result of LDA, samples with the same label are concentrated together, and the intervals between different classes are large. It indicates that two-dimensional data after LDA retain a lot of useful feature information, which can well reflect the differences among different classes.

4.2.2 Cluster by K-Means & Hierarchical Clustering

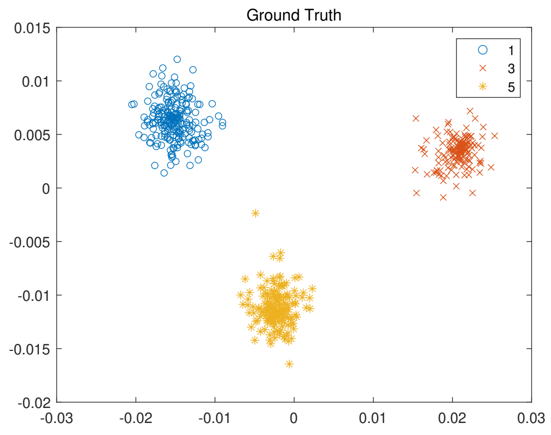
Results I also used both K-Means and Hierarchical Clustering to cluster the data after LDA. The results are in **Fig. 16**.



(a) K-Means result



(b) Hierarchical Clustering result



(c) Ground Truth

Figure 16: Comparison of the clustering results by K-Means, Hierarchical Clustering and ground truth of data after LDA.

Discussion The results of K-Means and Hierarchical Clustering are quite close, the accuracy of which can both reach **100%**. The reason might be that the distances between samples with different labels after LDA are so large.

4.2.3 Comparison with PCA

Comparison & Conclusion PCA is unsupervised, while LDA is supervised. When reduce the dimension to 2, LDA makes the data with **different labels more dispersed** and the data with the **same label more concentrated**. Therefore, We can more easily distinguish different classes of samples after LDA. The reason might be unlike PCA which only looks for directions that maximize variance among all data, LDA maximizes the between-class distance and minimizes the within-class distance.

Besides, compared to PCA, **the clustering results of data after LDA are better**. Any algorithm clusters the samples in the same class together more easily and obtains higher accuracy for LDA, because there are distinct intervals among the 3 classes in the plot of LDA result. **In conclusion, the performance of LDA is much better than PCA in this assignment**.

4.3 Part III: Binary Classification

The images of ’5’ are separated from the rest to construct a binary-classification problem. The cross validations of the 3 classifiers are accomplished on the **same 5-folds**, so that the evaluation can be more convincing.

4.3.1 SVM with Linear Kernel

Results The accuracy of the 5-fold cross validation is listed in **Tab. 1** and the corresponding confusion matrix is also shown below.

validation	1	2	3	4	5	average
accuracy	95.21%	94.79%	95.42%	95.83%	96.25%	95.50%

Table 1: Cross validation accuracy of SVM with linear kernel.

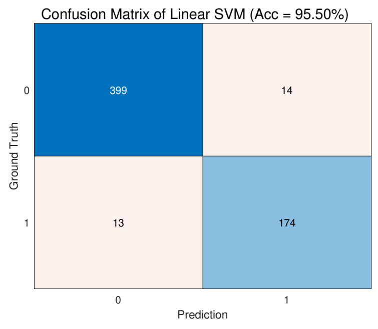


Figure 17: Confusion matrix of 5-fold cross validation for SVM with linear kernel.

Its ROC curve is shown in **Fig. 18**

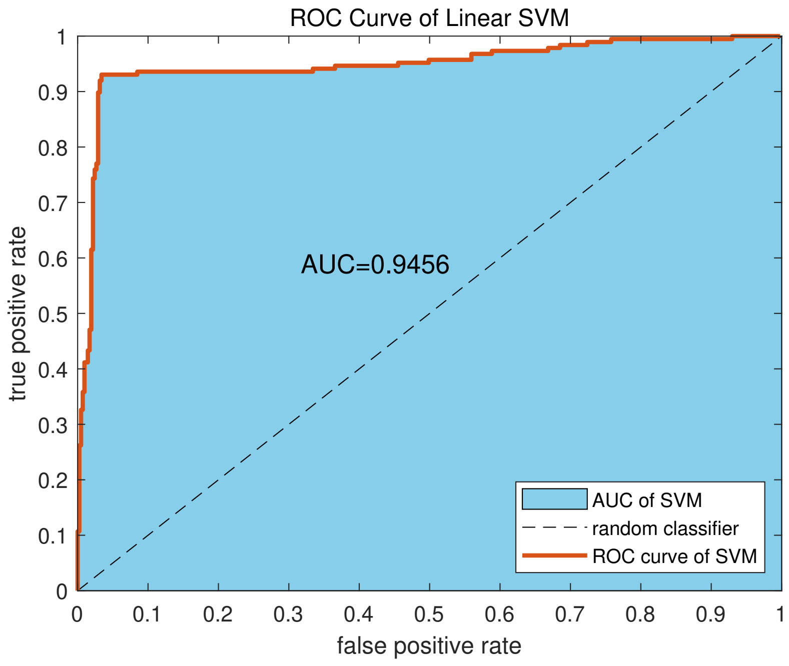


Figure 18: ROC curve of SVM with linear kernel.

Discussion The classification accuracy using SVM with linear kernel has been considerably high, which is **95.50%**. And the AUC of it is **0.9456**. Therefore, the performance of SVM with linear kernel is quite good on this dataset.

4.3.2 SVM with RBF Kernel

Results The accuracy of the cross validation is listed in **Tab. 2**.

validation	1	2	3	4	5	average
accuracy	98.33%	97.71%	98.13%	98.33%	98.33%	98.17%

Table 2: Cross validation accuracy of SVM with RBF kernel.

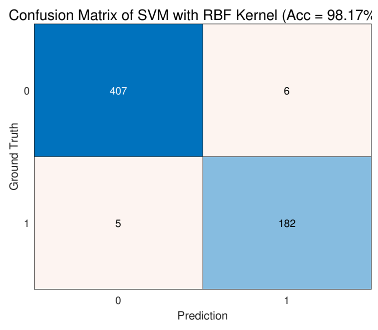


Figure 19: Confusion matrix of 5-fold cross validation for SVM with RBF kernel.

The ROC curve is shown in **Fig. 20**

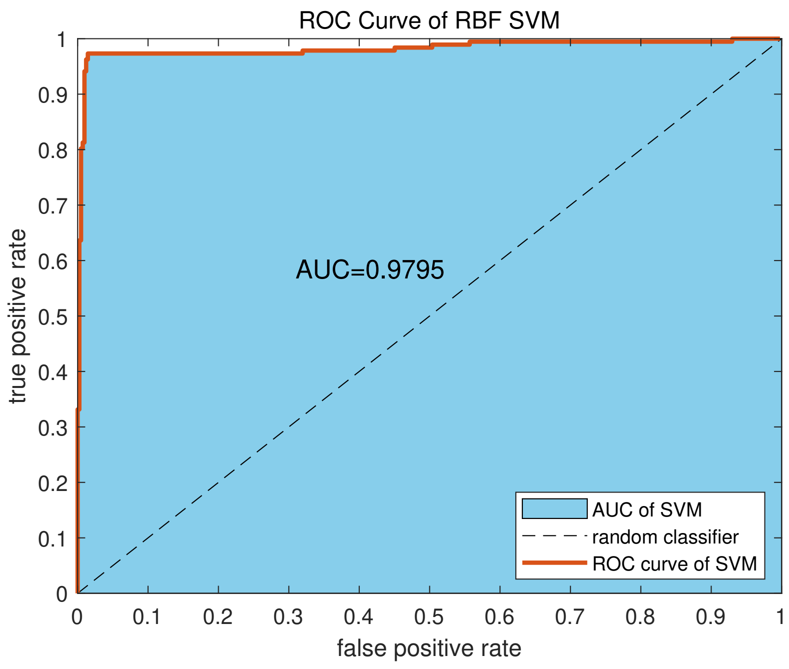


Figure 20: ROC curve of SVM with RBF kernel.

Discussion The classification accuracy using SVM with RBF kernel is even higher, which is **98.17%**. And the AUC is **0.9795**. So the classification performance of SVM with RBF kernel is excellent.

4.3.3 Neural Network with One Hidden Layer

Results I construct a neural network with one hidden layer, and size of the layer is 10. The activation function is **ReLU**. **Softmax** and **cross entropy** are also used in the output layer. The accuracy of the cross validation is listed in **Tab. 3**.

validation	1	2	3	4	5	average
accuracy	93.33%	93.33%	95.00%	98.33%	94.17%	94.83%

Table 3: Cross validation accuracy of neural network with one hidden layer.

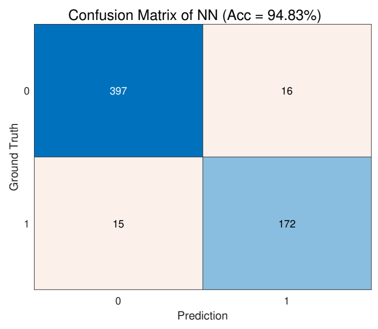


Figure 21: Confusion matrix of 5-fold cross validation for neural network.

The ROC curve is shown in **Fig. 22**

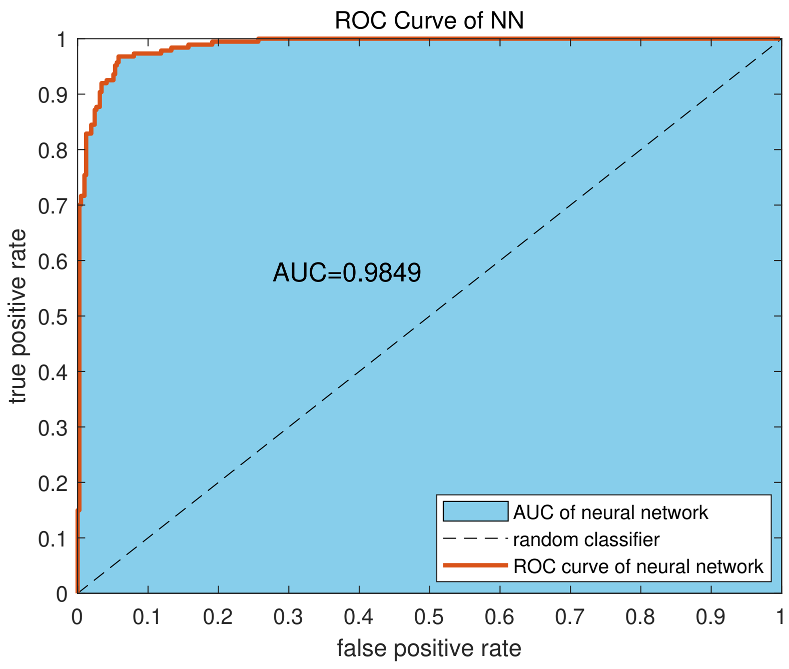


Figure 22: ROC curve of neural network.

Discussion The classification accuracy of neural network is **94.83%**. And the AUC is **0.9849**. So the classification performance of neural network with one hidden layer is also quite good.

4.3.4 Comparison

From the above analysis, we can find that the performances of the 3 classifiers are good but still have some differences.

Based on AUC I plot the ROC curves of the 3 classifiers together as **Fig. 23**. The AUC of neural network is 0.9849, which is much higher than that of SVM with linear kernel (0.9456) and close to SVM with RBF kernel (0.9795). So from the view of AUC, the neural network has the best performance.

Based on Accuracy Among the 3 classifiers, SVM with RBF kernel has the highest accuracy, which is 98.17%. The accuracy of SVM with linear kernel and neural network is both around 95%. So if based on accuracy, SVM with RBF kernel is the best classifier among the three.

Conclusion In conclusion, SVM with RBF kernel and neural network with one hidden layer are better than SVM with linear kernel in classifying the data of this assignment.

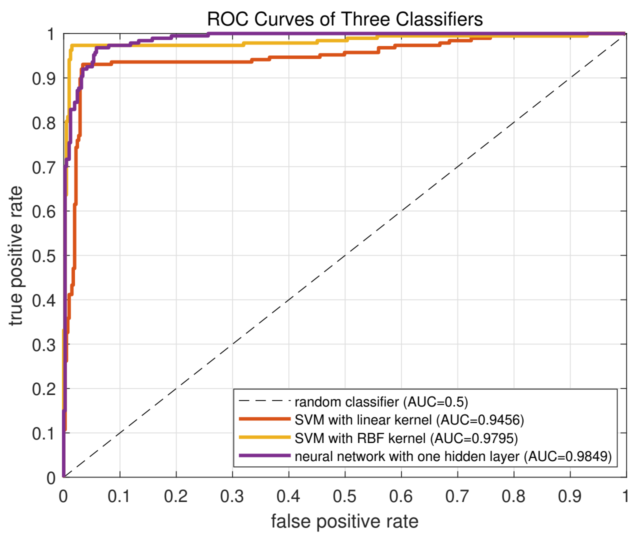


Figure 23: ROC curves of three classifiers.

4.3.5 Parameter Adjustment

Experiment & Results Here I choose to tune the RBF kernel parameter γ to adjust the performance of SVM. As introduced in the previous section, too large or too small γ can both result in bad classification performance. What I need to do is to find the appropriate range of γ. So I tune γ **from 0.0001 to 1000**, the corresponding accuracy and AUC of each γ are shown in **Tab. 4** and **Fig. 24**.

γ	**0.0001**	**0.0005**	**0.001**	**0.005**	**0.01**	**0.03**
accuracy	68.83%	90.00%	94.17%	97.50%	98.17%	98.50%
AUC	0.5203	0.8376	0.9192	0.9695	0.9795	0.9826
γ	**0.05**	**0.08**	**0.1**	**1**	**10**	**1000**
accuracy	98.17%	95.00%	85.67%	68.83%	68.83%	68.83%
AUC	0.9765	0.9200	0.7923	0.5203	0.5203	0.5203

Table 4: Accuracy for different γ.

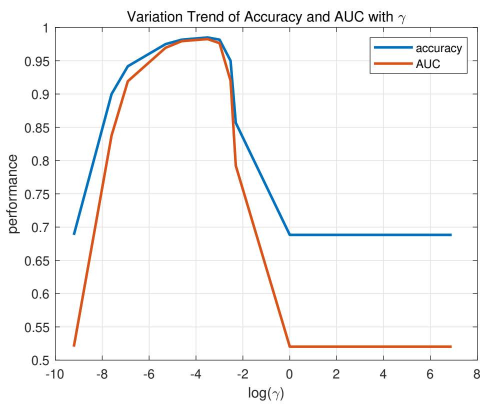


Figure 24: Changing trend of accuracy and AUC.

And the ROC curves are plotted in **Fig. 25**.

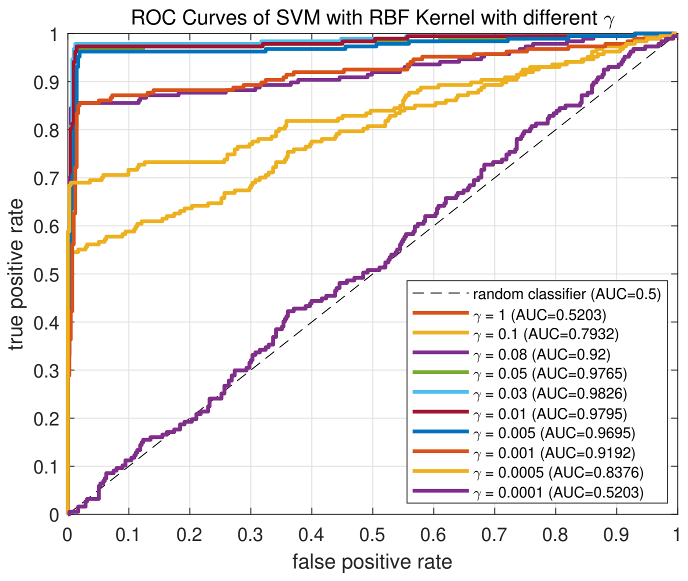
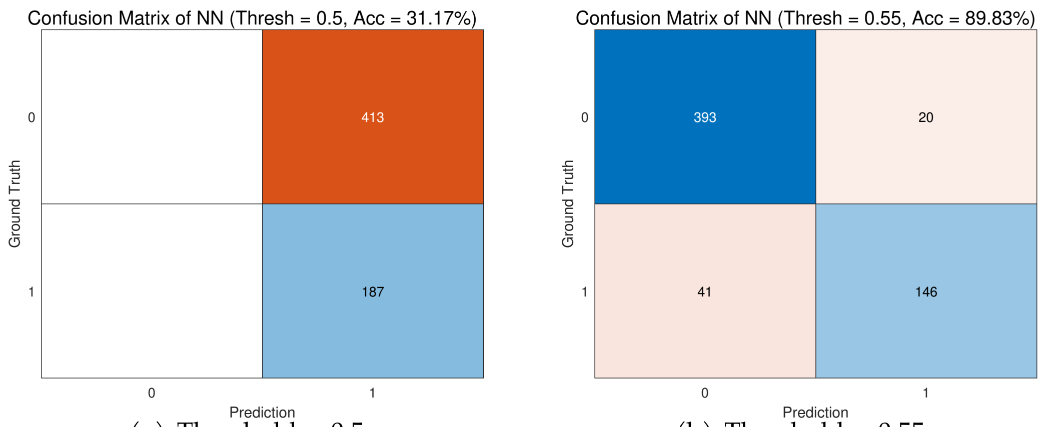


Figure 25: ROC curves of SVM with RBF kernel for different γ.

Discussion From the table and figures above, we can find that when γ changes from 0.0001 to 1000, the classification accuracy and AUC of SVM both firstly increase and then decrease. When $\gamma = 0.0001$ (too small) and $\gamma = 1, 10, 1000$ (too large), the performance of SVM is just like a random classifier. In conclusion, from the experiment results, the appropriate range of γ in this assignment might be **from 0.001 to less than 0.1**.

4.3.6 Further Exploration

Neural Network with Two Hidden Layers I also test the performance of neural network with two hidden layers on this dataset. To compare with my neural network with one hidden layer, I set the two hidden layers sizes both to 5. The results are as following:



(a) Threshold = 0.5 (b) Threshold = 0.55

Figure 26: The confusion matrices of neural network with two hidden layers with two thresholds.

From the above figure, we can find that when the threshold is set to 0.5, all samples are classified to class 1, i.e., digit '5'. I checked the output scores and found that the lowest score is 0.5. Then I tuned the threshold to 0.55, and the accuracy became 89.83%. I also plotted the ROC curve and its AUC is 0.9282:

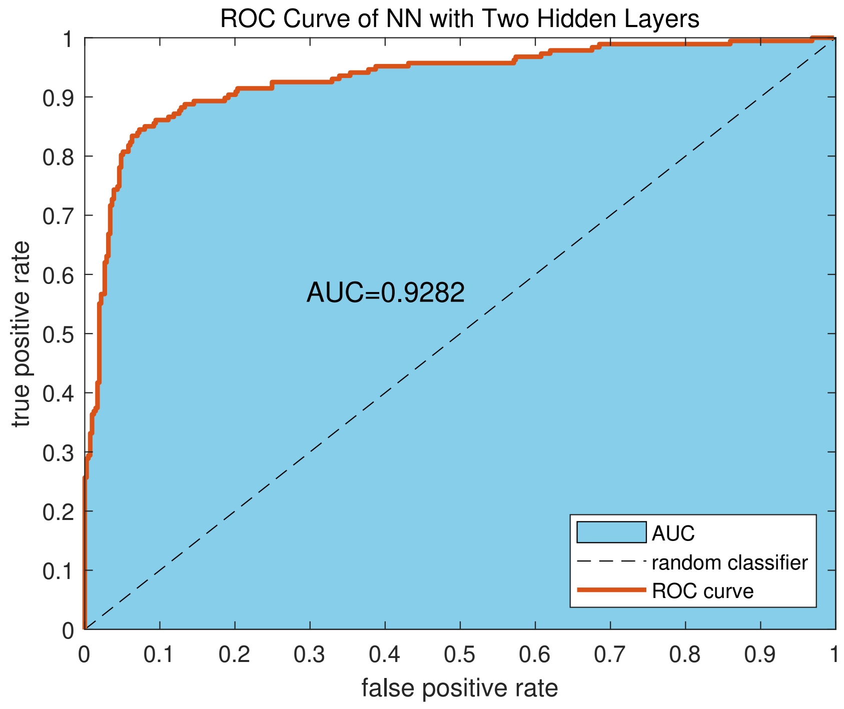


Figure 27: ROC curve and AUC of neural network with two hidden layers.

Compared with the neural network with one hidden layer, the accuracy and AUC of this neural network are both much lower. Therefore, the performance of the neural network with two hidden layers is worse. This result is kind of surprising, for generally the performance of two-layer neural network is better than that of one-layer neural network. The reason might be that the neural network with two hidden layers is overfitting, and more possible explanations need to be further studied.

References

[1] Shlens, Jonathon. ″A tutorial on principal component analysis.″ arXiv preprint arXiv:1404.1100 (2014).

[2] Karamizadeh, Sasan, et al. ″An overview of principal component analysis.″ Journal of Signal and Information Processing 4.3B (2013): 173.

[3] Sharma, Alok, and Kuldip K. Paliwal. ″Linear discriminant analysis for the small sample size problem: an overview.″ International Journal of Machine Learning and Cybernetics 6.3 (2015): 443-454.

[4] Bock, Hans-Hermann. ″Clustering methods: a history of k-means algorithms.″ Selected contributions in data analysis and classification (2007): 161-172.

[5] Murtagh, Fionn, and Pedro Contreras. ″Algorithms for hierarchical clustering: an overview.″ Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2.1 (2012): 86-97.

[6] Burges, Christopher JC. ″A tutorial on support vector machines for pattern recognition.″ Data mining and knowledge discovery 2.2 (1998): 121-167.

[7] Chauhan, Vinod Kumar, Kalpana Dahiya, and Anuj Sharma. ″Problem formulations and solvers in linear SVM: a review.″ Artificial Intelligence Review 52.2 (2019): 803-855.

[8] Pal, Mahesh. ″Kernel methods in remote sensing: a review.″ ISH Journal of Hydraulic Engineering 15.sup1 (2009): 194-215.

[9] Chandra, Mayank Arya, and S. S. Bedi. ″Survey on SVM and their application in image classification.″ International Journal of Information Technology 13.5 (2021): 1-11.

[10] Abiodun, Oludare Isaac, et al. ″Comprehensive review of artificial neural network applications to pattern recognition.″ IEEE Access 7 (2019): 158820-158846.

[11] Jain, Anil K., Jianchang Mao, and K. Moidin Mohiuddin. ″Artificial neural networks: A tutorial.″ Computer 29.3 (1996): 31-44.

[12] Basheer, Imad A., and Maha Hajmeer. ″Artificial neural networks: fundamentals, computing, design, and application.″ Journal of microbiological methods 43.1 (2000): 3-31.

[13] Gonçalves, Luzia, et al. ″ROC curve estimation: An overview.″ REVSTAT–Statistical Journal 12.1 (2014): 1-20.

[14] Park, Seong Ho, Jin Mo Goo, and Chan-Hee Jo. ”Receiver operating characteristic (ROC) curve: practical review for radiologists.” Korean journal of radiology 5.1 (2004): 11-18.