Classification: From Linear Solver to Neural Network

Xiangyu Gao, Qiangqiang Guo

March 19, 2021

Abstract

Classification is one of the most important problems in machine learning. In this project, we explore different classification techniques, from the basic linear regression, the classical support vector machine, to those methods emerging in recent years such as fully connected neural network and convolutional neural network. We test the classification accuracy, training time and classification running time of different methods and compare their performances. It is found that, for the MNIST data, convolutional neural network performs best considering the test accuracy while linear regression performs worst. However, when taking account of both accuracy and timing, the fully connected neural network is the best choice to achieve the balance. The experiment results also provide insights of how each method performs under different hyper-parameters and how we should select the best ones.

1 Introduction and Overview

Classifying different objects is an easy task for human beings. However, it is quite difficult for machines to correctly identify different objects. Building accurate classifiers is one of the most challenging tasks in machine learning. Researchers have put a lot of efforts on this topic and many classifiers have been developed. In this project, we will explore four representative methods during the development of classification algorithms: linear regression solver (LRS), support vector machine (SVM), fully connected neural networks (FCN), and convolutional neural networks (CNN). These methods were invented chronologically, each representing the best classifier at their age. We will introduce the backgrounds and mechanisms behind each method as well as comprehensively evaluate and compare their performances.

We use the MNIST data set as the training and testing data. MNIST consists of 60000 training and 10000 testing handwritten digit images from number 0 to 9, which gives us great opportunity to analyze the properties of handwritten digits and explore the techniques of identifying different digit numbers from others.

In the following of this report, we will first introduce the theoretical backgrounds of LRS, SVM, FCN, and CNN (Section 2). Then, we show how to design and implement those classification algorithmS in MATLAB (Section 3). We show the computation results in Section 4 and conclude this report in Section 5.

2 Theoretical Background

2.1 Linear regression solver (LRS)

Given a data set $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^{m}$ of m statistical units, a linear regression model assumes that the relationship between the dependent variable y and the p -vector of regressors $\mathbf{x}$ is linear. This relationship is modeled through a disturbance term or error variable ε - an unobserved random variable that adds "noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the form

$$y_i = \alpha_0 + \alpha_1 x_{i1} + \cdots + \alpha_p x_{ip} + \varepsilon_i = \mathbf{A}\mathbf{x}_i + \varepsilon_i, \quad i = 1, \ldots, m$$

For $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$, and $\mathbf{Y} = [y_1, y_2, \cdots, y_m]$, we can model above linear regression problem as

$$\mathrm{Y} = \mathrm{AX} + \varepsilon$$

The linear regression problem can be solved by adding the constraint to minimize the term ε.

2.2 Support vector machine (SVM)

Let $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...(\mathbf{x}_m, y_m)\}$ be the training data set, where $\mathbf{x}_i$ is a n-dimensional feature vector and y_i is the label. The key idea of SVM is to create a line or a hyperplane (based on the kernel) to separates the data into classes. The SVM is initially developed as a binary classifier to identify the positive and negative outputs, i.e., y_i is either 1 or -1. In the rest of this section, we will follow the flow of [3] to discuss the mechanism behind the binary SVM.

We firstly assume that the training data can be correctly separated by a linear function, which indicates there is a hard-margin that can perfectly identify the training data. Let the linear function be

$$F(\mathbf{x}) = \mathbf{w}\mathbf{x} - b \tag{1}$$

where $\mathbf{w}$ is a weight vector and b is the bias. To correctly classify the training set, we must have $F(\mathbf{x}_i) > 0$ if $y_i = 1$ and $F(\mathbf{x}_i) < 0$ if $y_i = -1$. This condition can be aggregated as

$$F(\mathbf{x}_i)y_i > 0 \quad \forall(\mathbf{x}_i, y_i) \in D \tag{2}$$

If Equation 2 is satisfied, $\mathbf{w}$ and b can always be re-scaled to

$$F(\mathbf{x}_i)y_i \geq 1 \quad \forall(\mathbf{x}_i, y_i) \in D \tag{3}$$

The distance between the hyperplane and a vector $\mathbf{x}_i$ is $\frac{|F(\mathbf{x}_i)|}{\|\mathbf{w}\|}$. Given Equation 3, the margin (i.e., the minimum distance between the hyper plane and all vectors) is

$$d = \frac{1}{\|\mathbf{w}\|} \tag{4}$$

The closest vectors are called support vectors. We want to maximize the margin to better separate the data. Given Equation 4, the problem of maximizing the margin becomes the problem of minimizing the weight vector $\|\mathbf{w}\|$:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & (\mathbf{w}\mathbf{x}_i - b)y_i \geq 1 \quad \forall(\mathbf{x}_i, y_i) \in D \end{aligned} \tag{5}$$

where $\frac{1}{2}$ is added for computation convenience. This is a constrained optimization problem with a convex objective function and linear constraints. We can solve this problem using the method of Lagrange multipliers. In this way, we can directly get $\mathbf{w}$. If a new query $\mathbf{x}_{new}$ comes in, we should explicitly calculate $\mathbf{w}\mathbf{x}_{new}$ to determine the classification. However, such a computation may be expensive if the dimension is large. One way to address this problem is to transform the prime problem (Equation 5) to its dual problem

$$\begin{aligned} \max_{\alpha_i} \quad & \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \mathbf{x}_i\mathbf{x}_j \\ \text{s.t.} \quad & \sum_i \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \end{aligned} \tag{6}$$

where α_i is the Lagrange multiplier. The optimal Lagrange multiplier α_i^* calculated from the dual problem indicate whether the vector $\mathbf{x}_i$ is one of the support vectors or not. To answer a new query $\mathbf{x}_{new}$, we only need to calculate $F(\mathbf{x}_{new}) = \sum_i \alpha_i y_i \mathbf{x}_i \mathbf{x}_{new} - b$, only the support vectors need to be calculated.

If D is not linearly separable, we can extend the hard-margin SVM to a soft-margin SVM by adding a slack variable ξ:

$$\begin{aligned} \min_{\mathbf{w},b,\xi_i} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & (\mathbf{w}\mathbf{x}_i - b)y_i \geq 1 - \xi_i \quad \forall(\mathbf{x}_i, y_i) \in D \\ & \xi_i \geq 0 \end{aligned} \tag{7}$$

where the slack variable ξ_i allows the data to be misclassified, C is a weighting parameter to determines the trade-off between the margin size and the amount of error. Similarly, this prime problem can be converted to the dual problem and be solved.

One can extend the binary SVM to multiple class classifiers by building multiple binary SVMs to evaluate each label individually, and take the one that has the highest score as the classification results.

2.3 Fully connected network (FCN)

A fully connected neural network consists of a series of fully connected layers. A fully connected layer is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$. Each output dimension depends on each input dimension.

Let's dig a little deeper into what the mathematical form of a fully connected network is. Let $\mathbf{x} \in \mathbb{R}^n$ represent the input to a fully connected layer. Let $\mathbf{z} \in \mathbb{R}^m$ be the m-dim output from the fully connected layer. Then $\mathbf{z}$ is computed with $\mathbf{W} \in \mathbb{R}^{n \times m}$ as follows:

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x})$$

Here, σ is a nonlinear function (think of σ as the sigmoid function for example), and the $\mathbf{W}$ are learnable parameters in the network.

It's directly possible to stack fully connected networks. A network with multiple fully connected networks is often called a *deep* network.

2.4 Convolutional neural network (CNN)

Similar to the FCN, the CNN is also a neural network with inputs and outputs. The input of CNN could be matrices while the input of FCN is usually vectors. Compared with FCN which only has fully connected hidden layers, there are different types of hidden layers in CNN to effectively learn the features hidden behind the data. These hidden layers enable the CNN learn complex features from very simple features to correctly identify the object. Therefore, CNN is more suitable to process image data.

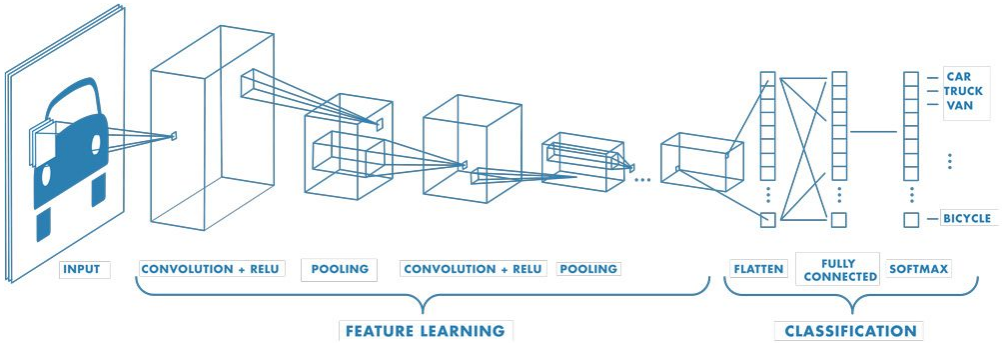

Figure 1: Illustration of CNN

Figure 1 shows a CNN example, which can be found at the MATLAB website [1]. There are four types of most commonly used hidden layers in CNN: convolution layer, rectified linear unit (ReLU) layer, pooling layer, and fully connected layer. The convolution layer use multiple convolutional filters to extract certain features from the images. The ReLU layer maps negative input values to zeros and maintain positive values to increase the training speed and avoid gradient explosion. The pooling layer apply nonlinear downsampling to reduce the number of parameters that the network needs to learn. The fully connected layer aggregates the features extracted by the former layers and output a vector indicating the classification results.

For a convolution layer, assume the input is $n_1 \times n_2 \times n_c$ where n_1 and n_2 are the height and width of a matrix and n_c is the number of matrices, also assume we use $f_1 \times f_2 \times f_c$ filters where f_1 and f_2 are the height and width of a filter and f_c is the number of filters that we want to use. Let p be the padding and s be the stride (see [2] for the detailed introduction of these parameters), the dimension of the output should be $[\frac{n_1+2p-f_1}{s} + 1] \times [\frac{n_2+2p-f_2}{s} + 1] \times f_c$. For a pooling layer, assume the same inputs are given, but now

$f_1 \times f_2$ represent the pooling size. The dimension of the output should be $[\frac{n_1-f_1}{s}+1] \times [\frac{n_2-f_2}{s}+1] \times n_c$. When the feature learning part is done, all the features are flattened to a vector as the first fully connected layer. Then, a deep network as discussed in Section 2.3 is connected to calculate the outputs.

3 Algorithm Development and Implementation

We first read the MNIST data. The training images data is a $28 \times 28 \times 60000$ matrix, where each 28×28 data represent a single digit image (from 0 to 9) and there are totally 60000 images. Correspondingly, the label data is a 60000×1 matrix. The test image data is a $28 \times 28 \times 10000$ matrix with a corresponding 10000×1 label matrix.

3.1 LRS

We first reshape the each training images into a column vector $\mathbf{x}_i$, thus the new training matrix is with size 784×60000. Then we represent each label as a column vector $\mathbf{b}_i$, which is given by

$$\text{"1"} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \text{"2"} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \cdots \quad, \text{"9"} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \text{"0"} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

Now let B be the set of all output vectors $\mathbf{b}_i$

$$\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 & \cdots & \mathbf{b}_m \end{bmatrix}$$

and let the matrix X be the corresponding reshaped (vectorized) MNIST images

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_m \end{bmatrix}$$

We subtract the row-wise mean from the training image matrix $\mathbf{X}$ to remove the effects of backround pixels. The mean-value vector is saved and then subtracted from the testing image matrix as well.

By using the $\mathbf{AX} = \mathbf{B}$ solver $\mathbf{A} = \mathbf{BX}^{-1}$, we can determine a mapping $\mathbf{A} \in \mathbb{R}^{10 \times 784}$ from the image space to the label space, where 10 is the size of label space and 784 is the size of image space.

With matrix $\mathbf{A}$, we predict the class of testing images by calculating $\mathbf{AX}_{\text{test}}$ and project the output vectors to exact labels.

3.2 SVM

3.2.1 SVD

In order to reduce the feature dimension and improve the classification efficiency, we first conduct singular value decomposition (SVD) on the MNIST data. In order to do the SVD analysis, we reshape each image into a column vector (784×1) and aggregate the whole 60000 images to one matrix as the training data $\mathbf{X}_{train}$, and aggregate another 10000 images to one matrix as the testing data $\mathbf{X}_{test}$. Each column of the data matrix thus is a different image. We then conduct the SVD on the aggregated training data $\mathbf{X}_{train} = \mathbf{U\Sigma V}^{'}$. Based on the spectrum of singular values (please check assignment 4 for details) and the experiment of reconstructing the images, we determine the number of modes that are necessary for good image reconstruction, i.e., the rank r of the digit space, as $r = 50$.

3.2.2 SVM

Since the matrix $\mathbf{V}$ represents the projection of each digit image onto the principle modes, we can use some columns of $\mathbf{\Sigma} * \mathbf{V}^{'}$ as the classification features. As discussed before, we use the first 10 columns as the

features. When it comes to testing, we take the same indexed columns of $\mathbf{U}^{'}\mathbf{X}_{test}$ as the predictor, and compare the results with the labels of testing data.

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

3.3 FCN

We reshape each image to a column vector and concatenate them along the columns, forming the training matrix $\mathbf{X}$ and testing matrix $\mathbf{X}_{\text{test}}$. The training matrix with corresponding labels are input a fully connected neural network, which is composed of 8 layers: fully connected layer with output size 1024, ReLU layer, fully connected layer with output size 512, ReLU layer, fully connected layer with output size 128, ReLU layer, fully connected layer with output size 10, softmax layer. This fully connected is implemented and trained on MATLAB Deep Learning Toolbox. For training details including parameters, please refer to the codes in appendix for details. We use the trained FCN model to test the images and calculated the accuracy of correctly classifying each digit.

3.4 CNN

We first transform the matrix-format MNIST data to images and store them under corresponding folders. Then, we load those image data (for both training data and test data) into MATLAB `imageDatastore` object, which consists of the file path of each image, and the labels of each image (which is extracted based on the parent folder's name). Then, we build the CNN using three convolution layers, each is followed by a batch normalization layer to standardize the inputs to the next layer and a ReLU layer to increase training speed. The filter size of each convolution layer is set to 3×3 and the numbers of filters are 8, 16, and 32, respectively. Between each adjacent convolution layer, we add a pooling layer with pooling size 2×2. Finally, we flatten the outputs of the last ReLU layer to a fully connected layer and add a softmax layer to generate the classification results. This structure is the most basic CNN structure and can be found at the MATLAB website [1]. Then, we set the training parameters using `trainingOptions`. Please refer to Appendix B for details. There are many hyper-parameters that can be tuned in CNN, e.g., solvers, the learning rate, number of epochs, filter size, pooling size, stride, padding size, activation function, etc. We use the hyper-parameters same as those at the MATLAB website [1]. In addition, we test different number of epochs to better understand the performance of CNN. Finally, we train the CNN and calculate the classification accuracy.

4 Computation results

Before discuss the detailed results of each classification method, we first introduce the properties of the computer that we used to do the numerical experiment. The operation system is 64-bit Windows 10, the memory (RAM) is 32GB, the processor is Intel Core i9-9900K CPU @ 3.6GHz.

4.1 LRS

We use the calculated LRS for classifying all 10 digits. The average accuracy on the test data is 86.12%. Meanwhile, we calculate the LRS training time and running time. The training time on the MNIST training dataset is $3.45s$ and the running time to go over all the MNIST test data is $0.029s$. The accuracy of LRS on different individual digits is shown in Table 1, different digits have different accuracies, ranging from 74.78% (digit 5) to 97.09% (digit 1).

Digits	0	1	2	3	4	5	6	7	8	9
Accuracy	96.22%	97.09%	79.07%	87.13%	89.92%	74.78%	91.23%	86.09%	77.93%	79.58%

Table 1: Accuracy for individual digit under LRS.

4.2 SVM

Considering that SVM is originally designed for binary classification, we first show the performance of SVM on different two-digit pairs. The results are shown in Table 2. Note that the lower triangle is omitted since this table is symmetric. It can be shown that SVM performs pretty good (the lowest accuracy is 93.87% for the digit pair 4 and 9) on identifying every two-digit pairs.

Digits	1	2	3	4	5	6	7	8	9
0	99.48%	99.35%	99.50%	99.75%	98.56%	98.66%	99.55%	99.23%	99.25%
1		99.35%	99.21%	99.34%	99.31%	99.19%	99.21%	99.05%	99.11%
2			98.87%	99.35%	99.06%	98.64%	98.74%	97.76%	98.97%
3				99.15%	97.48%	99.49%	98.72%	94.56%	97.72%
4					99.52%	98.87%	98.81%	99.03%	93.87%
5						98.65%	99.43%	96.25%	98.05%
6							99.80%	99.02%	99.14%
7								98.15%	97.45%
8									96.57%

Table 2: Accuracy for different digit pairs using SVM (Gaussian kernel)

We then build multiple binary SVMs and integrate them together to classify all 10 digits. The average accuracy on the test data is 93.32%. Meanwhile, we calculate the SVM training time and running time. The training time on the MNIST training dataset is $325.421s$ and the running time to go over all the MNIST test data is $2.106s$. We then show the accuracy of SVM on different individual digits. As shown in Table 3, different digits have different accuracies, ranging from 85.93% (digit 8) to 98.94% (digit 1).

Digits	0	1	2	3	4	5	6	7	8	9
Accuracy	97.14%	98.94%	95.74%	92.57%	91.65%	92.38%	95.93%	94.26%	85.93%	87.71%

Table 3: Accuracy for individual digit under SVM

4.3 FCN

We train the FCN for 1 epoch and the training loss and accuracy are shown in Figure 2. From that, we can tell that FCN is almost converged wit around 100 iterations.

We use the trained FCN for classifying all 10 digits. The average accuracy on all test data is 93.56%. Meanwhile, we calculate the FCN training time and running time. The training time on the MNIST training dataset is $9.94s$ and the running time to go over all the MNIST test data is $0.391s$. The accuracy of FCN on different individual digits is shown in Table 4, different digits have different accuracies, ranging from 87.98% (digit 4) to 98.50% (digit 1).

Figure 2: The training process of FCN for 1 epoch

Digits	0	1	2	3	4	5	6	7	8	9
Accuracy	97.86%	98.50%	92.25%	94.95%	87.98%	89.24%	96.24%	95.82%	88.30%	93.26%

Table 4: Accuracy for individual digit under FCN

4.4 CNN

We test the CNN performance under different number of epochs used in the training process. The results are shown in Table 5. It can be shown that the accuracy has reached to a high level (98.13%) when the CNN is trained for only 1 epoch. Further increasing the number of epochs will increase the training time (almost linearly) and the testing time used to go through all the test data. However, the accuracy increment brought by increasing epoch number is trivial. Therefore, we believe that 1 epoch is enough for current CNN setting and the MNIST data.

Epoch	Accuracy (%)	Training time (seconds)	Testing time (seconds)
1	98.13	53	1.646
2	98.54	102	1.651
4	98.96	197	2.168
10	98.85	493	2.215

Table 5: Performance of CNN under different epochs

Figure 3 shows the training process of CNN for 1 epoch. It is shown that the loss reduces dramatically at the early stage, and keeps reducing steadily. Finally, the loss approximates zero and maintain that level in a stable manner. This indicates the CNN is successfully converged and loss is minimized.

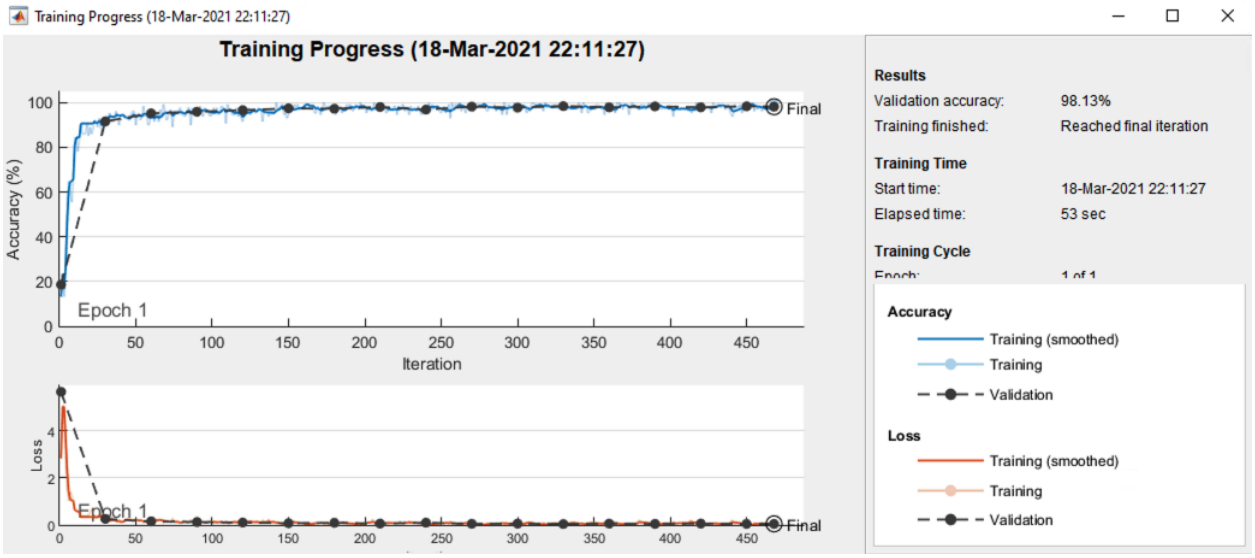

Figure 3: The training process of CNN for 1 epoch

We then show the accuracy of CNN (trained with 1 epoch) on different individual digits. As shown in Table 6, different digits have different accuracies, ranging from 95.34% (digit 9) to 99.91% (digit 1).

Digits	0	1	2	3	4	5	6	7	8	9
Accuracy	99.49%	99.91%	97.77%	98.71%	98.68%	98.21%	95.82%	97.96%	99.18%	95.34%

Table 6: Performance of CNN for individual digits

4.5 Comparison among the four classifiers

We summarize the performance of the four methods in Table 7. It can be seen that CNN achieves best testing accuracy 98.13% while LRS has the worst one 86.12%. The high performance of CNN comes at the cost of computation load and time. On the other hand, FCN achieves similar testing accuracy to SVM with much less training and testing time.

Method	Accuracy (%)	Training time (seconds)	Testing time (seconds)
LRS	86.12	3.45	0.029
SVM	93.32	325.42	2.106
FCN	93.56	9.94	0.391
CNN	98.13	53	1.646

Table 7: Performance comparison of different classifiers on the MNIST data

5 Summary and Conclusions

In this project, we explore different classification techniques, from the basic LRS, the classical SVM, to those methods emerging in recent years such as FCN and CNN. We test the classification accuracy, training time and classification running time of different methods and compare their performances. The experiment results give insights about how we should select the best ones under different situations.

References

[1] MATLAB. *CNN*. `https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html#how-they-work`.

[2] Jianxin Wu. "Introduction to convolutional neural networks". In: *National Key Lab for Novel Software Technology. Nanjing University. China* 5 (2017), p. 23.

[3] Hwanjo Yu and Sungchul Kim. "SVM Tutorial-Classification, Regression and Ranking." In: *Handbook of Natural computing* 1 (2012), pp. 479–506.

Appendix A GitHub repository

`https://github.com/Guoqq17/UW_AMATH582_Project`

Appendix B Key MATLAB Functions

- `B = reshape(A,sz1,...,szN)` reshapes `A` into a `sz1-by-...-by-szN` array where `sz1,...,szN` indicates the size of each dimension.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector `v` on the main diagonal.
- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of textttm-by-n matrix texttt$A$.
- `Mdl = fitcsvm(X,Y)` returns an SVM classifier trained using the predictors in the matrix `X` and the class labels in vector `Y` for one-class or two-class classification.
- `Mdl = fitcsvm(***,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in previous syntaxes.
- `label = predict(Mdl,X)` returns a vector of predicted class labels for the predictor data in the table or matrix `X`, based on the trained, full or compact classification tree `Mdl`.
- `[label,score,node,cnum] = predict(***)` uses any of the input argument in the previous syntaxes and additionally returns: 1) A matrix of classification scores (score) indicating the likelihood that a label comes from a particular class. For classification trees, scores are posterior probabilities. For each observation in X, the predicted class label corresponds to the minimum expected misclassification cost among all classes. 2)A vector of predicted node numbers for the classification (node). 3)A vector of predicted class number for the classification (cnum).
- `f = fullfile(filepart1,...,filepartN)` builds a full file specification from the specified folder and file names.
- `mkdir folderName` creates the folder folderName. If folderName exists, MATLAB issues a warning.
- `imwrite(A,filename)` writes image data `A` to the file specified by `filename`, inferring the file format from the extension. `imwrite` creates the new file in your current folder. The bit depth of the output image depends on the data type of `A` and the file format.
- `imds = imageDatastore(location)` creates a datastore `imds` from the collection of image data specified by location.
- `net = trainNetwork(images,layers,options)` trains the neural network specified by `layers` for `image` classification and regression tasks using the images and responses specified by images and the training options defined by `options`.
- `YPred = classify(net,imds)` predicts class labels for the images in the image datastore `imds` using the trained network `net`.

Appendix C MATLAB Codes

C.1 LRS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Regression and Sparsity for Digit Classification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all
close all

% read training data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-
    ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte')
    ;

% Size of each picture
m = 28;
n = 28;
% reshape images
X = double(reshape(images_train, [m*n, length(labels_train)]));

% remove the averaged image (row-wise)
mean_data = mean(X, 2);
X = X - mean_data;

B_label = zeros(10, length(labels_train));% matrix B
for i = 1:length(labels_train)
    if labels_train(i) > 0
        B_label(labels_train(i), i) = 1;
    else
        B_label(10, i) = 1;
    end
end

% Solve AX = B
A = B_label * pinv(X);

%% Test
% reshape  test images
X_test = double(reshape(images_test, [m*n, length(labels_test)]));

% remove the averaged image (row-wise)
X_test = X_test - mean_data;

% test
B_test = A * X_test;

% predict
test_soft = softmax(B_test);
[~, testI] = max(test_soft, [], 1);
testI = mod(testI, 10);

% calculate accuracy for all 10 digits
YPred = testI';
YValidation = labels_test;
accuracy = sum(YPred == YValidation)/numel(YValidation);

%% accuracy for individual digits
accuracy_ind=zeros(1,10);
for i=0:9
    idx = find(labels_test == i);
    YPred_digit = testI(idx)';
    YValidation_digit = labels_test(idx);
    accuracy_ind(i+1) = sum(YPred_digit == YValidation_digit)/numel(YValidation_digit);
end
```

C.2 SVM

```
close all; clear; clc
%% import data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-
    ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte')
    ;


%% %% part I
[a_train,b_train,c_train]=size(images_train);
[a_test,b_test,c_test]=size(images_test);
data_train = zeros(a_train*b_train, c_train);
data_test = zeros(a_test*b_test, c_test);
for i=1:1:c_train
    data_train(:,i)=reshape(images_train(:,:,i),a_train*b_train, 1);
end
for i=1:1:c_test
    data_test(:,i)=reshape(images_test(:,:,i),a_test*b_test, 1);
end
% SVD
[U,S,V]=svd(double(data_train),'econ');
proj=(S*V')';
% singular value spectrum
plot(diag(S)/sum(diag(S)), '-o')
xlabel('Singular value','Fontsize',12)
ylabel('Proportion','Fontsize',12)

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

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

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

        pre=predict(Mdl_svm,xtest);

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

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

SVMModels = cell(10,1);
classes = 0:1:9;
rng(1); % For reproducibility
for j = 1:numel(classes)
    indx = labels_train==classes(j); % Create binary classes for each classifier
```

```
    SVMModels{j} = fitcsvm(xtrain,indx,'ClassNames',[false true],'Standardize',true,...
        'KernelFunction','rbf','BoxConstraint',1);
end

for j = 1:numel(classes)
    [~,score] = predict(SVMModels{j},xtest);
    Scores(:,j) = score(:,2); % Second column contains positive-class scores
end

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

%% SVM, individual accuracy
accuracy_ind=zeros(1,10);
for i=0:9
    xtest=xtest_temp(labels_test==i,2:10)/max(max(S));
    [len1,temp]=size(xtest);
    ctest=i*ones(len1,1);
    Scores=[];
    for j = 1:numel(classes)
        [~,score] = predict(SVMModels{j},xtest);
        Scores(:,j) = score(:,2);
    end
    [~,maxScore] = max(Scores,[],2);
    errorNum=sum(abs(ctest+1-maxScore)>0);
    accuracy_ind(i+1)=1-errorNum/length(ctest);
end
```

C.3 FCN

```
close all; clear; clc
%% import data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-
    ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte')
    ;

% reshape images
images_train = double(reshape(images_train, [28*28, length(labels_train)]));
images_test = double(reshape(images_test, [28*28, length(labels_test)]));

% create categorical label
cat_label_train = categorical(labels_train);

%% FCN
% define the network
layers = [
    featureInputLayer(28*28, 'Name','input')
    fullyConnectedLayer(1024, 'Name','fc1')
    reluLayer('Name','relu1')
    fullyConnectedLayer(512, 'Name','fc2')
    reluLayer('Name','relu2')
    fullyConnectedLayer(128, 'Name','fc3')
    reluLayer('Name','relu3')
    fullyConnectedLayer(10, 'Name','fc4')
    softmaxLayer('Name','sm')
    classificationLayer('Name','classification')]

% training options
options = trainingOptions('adam', ...
    'InitialLearnRate',0.005, ...
    'MaxEpochs',1, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

```

```
% train the network
net = trainNetwork(images_train',cat_label_train,layers,options);

% calculate accuracy
YPred = single(classify(net, images_test'));
YPred = YPred - 1;
YValidation = labels_test;
accuracy = sum(YPred == YValidation)/numel(YValidation);

%% accuracy for individual digits
accuracy_ind=zeros(1,10);
for i=0:9
    idx = find(labels_test == i);
    YPred_digit = YPred(idx);
    YValidation_digit = labels_test(idx);
    accuracy_ind(i+1) = sum(YPred_digit == YValidation_digit)/numel(YValidation_digit);
end
```

C.4 CNN

```
close all; clear; clc
%% import data
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-
    ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte')
    ;

% tranform MNIST matlab data to image files
if exist('MNIST_images','dir')==0
    for i=0:9
         label = num2str(i);
         mkdir(fullfile(pwd,'MNIST_images', 'train',label));
         mkdir(fullfile(pwd,'MNIST_images', 'test',label));
    end
    num=0;
    for i=1:length(labels_train)
        num=num+1;
        label=num2str(labels_train(i));
        name=fullfile(pwd,'MNIST_images','train',label,['images_' label '_' num2str(num) '.
    png']);
        imwrite(images_train(:,:,i), name);
    end
    num=0;
    for i=1:length(labels_test)
        num=num+1;
        label=num2str(labels_test(i));
        name=fullfile(pwd,'MNIST_images','test',label,['images_' label '_' num2str(num) '.
    png']);
        imwrite(images_test(:,:,i), name);
    end
else
end

%% CNN
% create data store
train_db=imageDatastore(fullfile(pwd,'MNIST_images','train'), 'IncludeSubfolders',true,'
    FileExtensions','.png','LabelSource','foldernames');
test_db=imageDatastore(fullfile(pwd,'MNIST_images','test'), 'IncludeSubfolders',true,'
    FileExtensions','.png','LabelSource','foldernames');

% define the network
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

```

```
    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];

% training options
options = trainingOptions('adam', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',1, ...
    'Shuffle','every-epoch', ...
    'ValidationData',test_db, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

% train the network
net = trainNetwork(train_db,layers,options);

% calculate accuracy
YPred = classify(net,test_db);
YValidation = test_db.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation);

%% CNN on individual digits
accuracy_ind=zeros(1,10);
for i=0:9
    test_db=imageDatastore(fullfile(pwd,'MNIST_images','test', num2str(i)), '
    IncludeSubfolders',true,'FileExtensions','.png','LabelSource','foldernames');
    YPred = classify(net,test_db);
    YValidation = test_db.Labels;

    accuracy(i+1) = sum(YPred == YValidation)/numel(YValidation);
end
```