Project Report for Handwritten Digit Recognition

Xin Wang

1 Introduction

Handwritten digit recognition is quite common in many applications nowadays. And there are many such data sets with quite high quality. And this is not a too complex classification problem, which means it is appropriate for preliminary investigation of some common classification algorithms. This project explored the following three classification algorithms on the recognition of handwritten digits.:

- ridge regression
- LASSO
- neural network

And Results and code can be found

`https://github.com/XinWang2019/ECE-532-Project`

2 Data set

The data set that will be used is MINIST database of handwritten digits, the link is

`http://yann.lecun.com/exdb/mnist/`

This data set contains a training set of 60,000 examples, and a test set of 10,000 examples. Each sample is an image with a handwritten digit from 0 to 9.

The image has 28×28 pixels and is gray-scaled. The pixel value is from 0 (white) to 255 (black). The digits have been size-normalized and the center of the mass of the pixels has been centered in the image. The sets of writers of the training set and test set have been ensured to be disjoint.

The classification problem is trying to recognize the handwritten digits and classify the image to digits.

3 Results

3.1 Ridge Regression

The sample is a vector of pixels, denoted as $\boldsymbol{x}$. The weight matrix is denoted as $\boldsymbol{w}$. And the prediction is given by

$$\hat{d} = \boldsymbol{x}\boldsymbol{w}$$

$\boldsymbol{w}$ is computed by

$$\boldsymbol{w} = (\boldsymbol{X}^T X + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

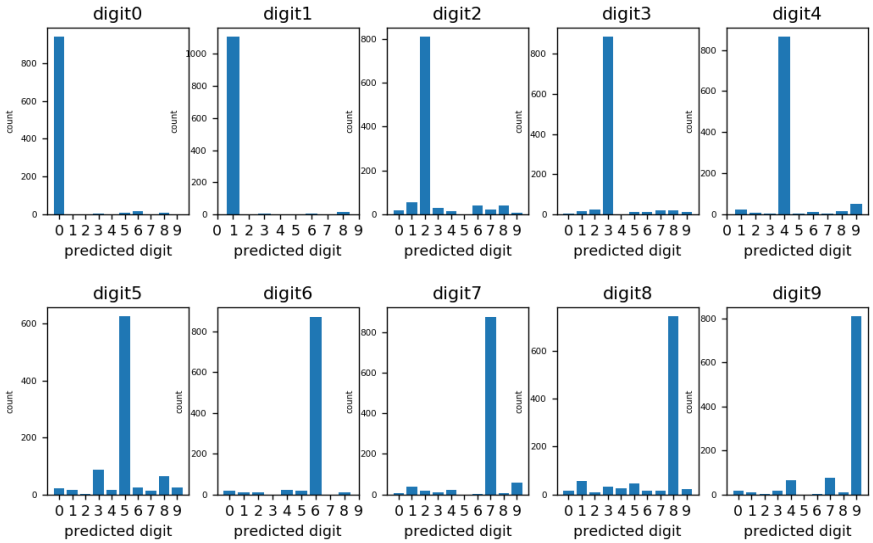

Figure 1: Error count distributions of each digit for ridge regression.

where $\boldsymbol{X}_{m,:} = \boldsymbol{x}_m^T$, which is the transpose of the m^{th} sample vector. $\boldsymbol{y}_m$ is the label for the m^{th} sample. λ is the tuning parameter, which will be determined by cross validation.

The experiment on ridge regression can be found in `ridge_regression.ipynb`.

This is a multi-class classification problem. How we define the label would determine the structure of the model.

3.1.1 Single model

Initially, $\hat{d} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. In this case, only one model needs to be trained.

The total number of singular values of training matrix is 784, of which about 100 values are most important. With only the 100 singular values preserved, a comparable accuracy can be achieved. But no improvement in accuracy is observed.

The overall error rate is very high, which is slightly better than jsut random guess. about 74% even for the best training result which I get. using training data directly or using a set that has been held out in advance to select the value of λ would give very different choice. This justifies the cross validation for choosing tuning parameters.

3.1.2 One model per digit

We will train a binary classifier model for each digit to predict whether the image belongs to this class. $+1$ means yes, -1 means no. In this case, $\hat{d} \in \{+1, -1\}$.

When doing the inference, there might be more than one model yield $+1$. In this case, only the prediction which is closest to $+1$ will be set as $+1$. All other prediction will be set as -1.

For this strategy, the overall error rate is reduced to 14.6%.

The detailed error count for each digit has been shown in Figure.1.

The error rate comparison among digits can be found in Figure.2.

3.2 LASSO

The gradient descent algorithm is used to solve LASSO. The experiment on LASSO can be found in `lasso.ipynb`.

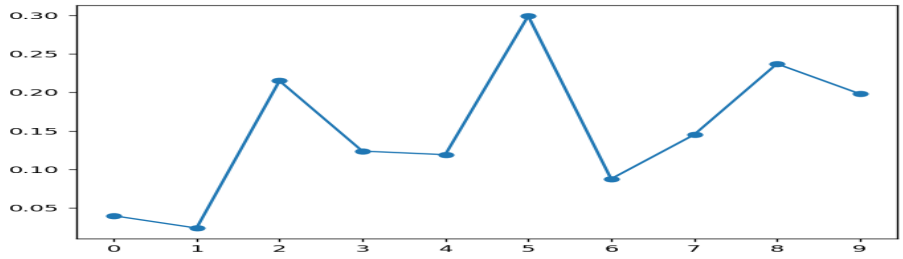

Figure 2: Error rate of each digit for ridge regression.

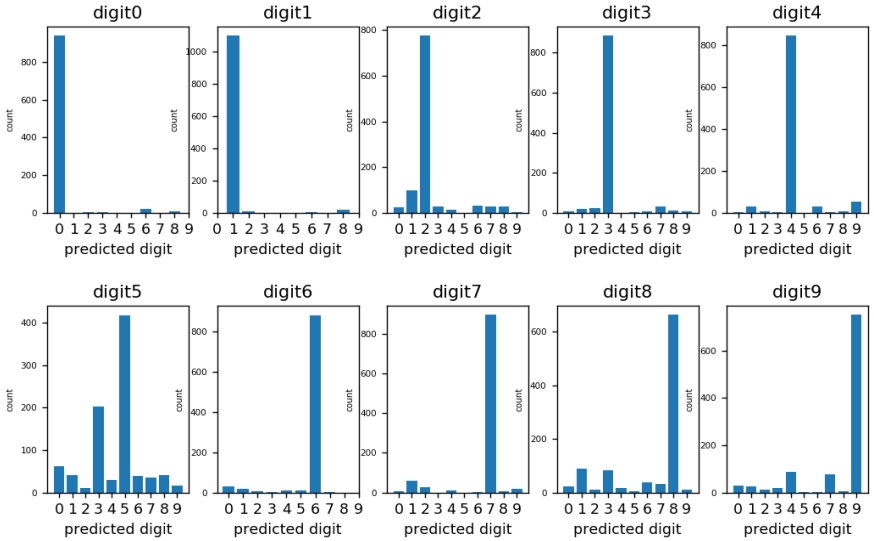

Figure 3: Error count distributions of each digit for LASSO.

With the lesson from ridge regression, we will train one binary classifier model for each digit for LASSO. The detailed error count for each digit has been shown in Figure.3.

The error rate comparison among digits can be found in Figure.4.

The overall error rate is 18.4%, which is slightly higher than ridge regression.

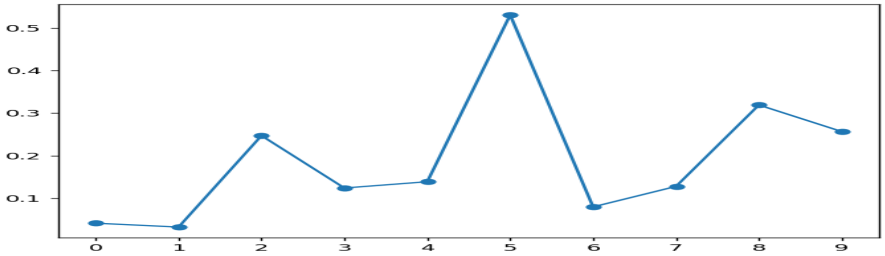

Figure 4: Error rate of each digit for LASSO.

3.3 Neural network

The experiment on the neural network can be found in `neural_network.ipynb`.

3.3.1 Structure of the neural network

The network has three hidden layers. The first layer has more nodes than the remaining two layers.

Three hidden layer, output is a 10-by-1 vector. For digit i, the i^{th} element of the label is 1. All other elements are 0. For example, the label for 2 is $[0010000000]^T$.

$$x \overset{w^{(1)}}{\Longrightarrow} h^{(1)} \overset{w^{(2)}}{\Longrightarrow} h^{(2)} \overset{w^{(3)}}{\Longrightarrow} h^{(3)} \overset{v}{\Longrightarrow} d$$

The activation function is the logistic function,

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$

3.3.2 Equation

First, the gradient equations are written into a more compact matrix form, which is more clear and easier to transform into code. Also, it is easier to infer the equations for deeper neural networks.

loss function is $f(w) = \frac{1}{2}(\hat{d}-d)^2$, where $\hat{d}$ is the prediction, d is the actual label.

- Gradient over v

$$\frac{\partial f}{\partial v_{ij}} = (\hat{d}_i - d_i)\hat{d}_i(1-\hat{d}_i)h_j^{(3)}$$

  with $\tilde{d}_i = (\hat{d}_i - d_i)\hat{d}_i(1-\hat{d}_i), \partial\tilde{v_{ij}} = \frac{\partial f}{\partial v_{ij}}$,

$$\partial\tilde{v} = \tilde{d}{h^{(3)}}^T$$

- Gradient over $w^{(3)}$

$$\frac{\partial f}{\partial w_{ab}^{(3)}} = \sum_i (\hat{d}_i - d_i)\hat{d}_i(1-\hat{d}_i)v_{ia} \\ h_a^{(3)}(1-h_a^{(3)})h_b^{(2)}$$

  with $\tilde{h}_a^{(3)} = h_a^{(3)}(1-h_a^{(3)}), \partial\tilde{w}_{ab}^{(3)} = \frac{\partial f}{\partial w_{ab}^{(3)}}$,

$$\partial\tilde{w}^{(3)} = (\tilde{d}^T v)^T \circ \tilde{h}^{(3)}{h^{(2)}}^T$$

  where $\circ$ represents element-wise multiplication. For $B_{n\times m} = A_{n\times m} \circ x_{1\times m}$ where dimensions do not match, $B_{ij} = A_{ij}x_j$.

- Gradient over $w^{(2)}$

$$\frac{\partial f}{\partial w_{cd}^{(2)}} = \sum_i (\hat{d}_i - d_i)\hat{d}_i(1-\hat{d}_i) \\ \sum_a v_{ia}h_a^{(3)}(1-h_a^{(3)})w_{ac}^{(3)} \\ h_c^{(2)}(1-h_c^{(2)})h_d^{(1)}$$

  with $\tilde{h}_c^{(2)} = h_a^{(2)}(1-h_c^{(2)}), \partial\tilde{w}_{cd}^{(2)} = \frac{\partial f}{\partial w_{cd}^{(2)}}$,

$$\partial\tilde{w}^{(2)} = (\tilde{d}^T v \circ (\tilde{h}^{(3)})^T w^{(3)})^T \circ \tilde{h}^{(2)}{h^{(1)}}^T$$

- Gradient over $w^{(1)}$

$$\frac{\partial f}{\partial w_{ef}^{(1)}} = \sum_i (\hat{d}_i - d_i)\hat{d}_i(1-\hat{d}_i) \sum_a v_{ia} h_a^{(3)}(1-h_a^{(3)}) \sum_c w_{ac}^{(n)} h_c^{(2)}(1-h_c^{(2)}) w_{ce}^{(2)} h_e^{(1)}(1-h_e^{(1)}) x_f$$

with $\tilde{h}_e^{(1)} = h_e^{(1)}(1-h_e^{(1)}), \partial\tilde{w}_{ef}^{(1)} = \frac{\partial f}{\partial w_{ef}^{(1)}}$,

$$\partial\tilde{w}^{(1)} = (\tilde{d}^T v \circ (\tilde{h}^{(3)})^T w^{(3)} \circ (\tilde{h}^{(2)})^T w^{(2)})^T \circ \tilde{h}^{(1)} x^T$$

3.3.3 Results

At first, the weights are all initialized to zero before training. But the training is unsuccessful. $w^{(1)}$ is always zeros.

Then the weights are initialized using Gauss distribution. And the training works fine.

1. Change the number of iterations

Iterations(10^4)	Nodes of layer 1	Nodes of layer 2	Nodes of layer 3	error rate
100	50	50	50	0.199
80	50	50	50	0.197
60	50	50	50	0.213
40	50	50	50	0.234
20	50	50	50	0.39
6	50	50	50	0.542

2. Change the number of nodes of the first layer

Iterations(10^4)	Nodes of layer 1	Nodes of layer 2	Nodes of layer 3	error rate
80	50	50	50	0.197
80	60	50	50	0.1915
80	70	50	50	0.188
80	80	50	50	0.171
80	90	50	50	0.187
80	100	50	50	0.170
80	150	50	50	0.155
80	300	50	50	0.140

3. Batch training. The gradient would be more stable using batch training.

Nodes of layer 1	Nodes of layer 2	Nodes of layer 3	Batch size	error rate
150	50	50	80	0.1519
150	50	50	128	0.137
150	50	50	256	0.124

4 Assessment and conclusion

For ridge regression and LASSO, their accuracy is quite similar. Both of them are very accurate when digit $\in \{0, 1, 3, 4, 6, 7\}$. The accuracy for digit $\in \{2, 8, 9\}$ is slightly worse. Both have very low accuracy in terms of $\{5\}$. This result is in accordance with their common idea, which is the tradeoff between accuracy and norm of the weights. And the performance of ridge regression is slightly better with overall error rate 14.6%compared with 18.4% of LASSO. This means that the weights are not very sparse. Otherwise, LASSO should yield better performance.

And the accuracy can be very low if we want to use a single classifier to deal with the multi-class problem. The best choice is to uses a series of simple linear classifiers for such problems.

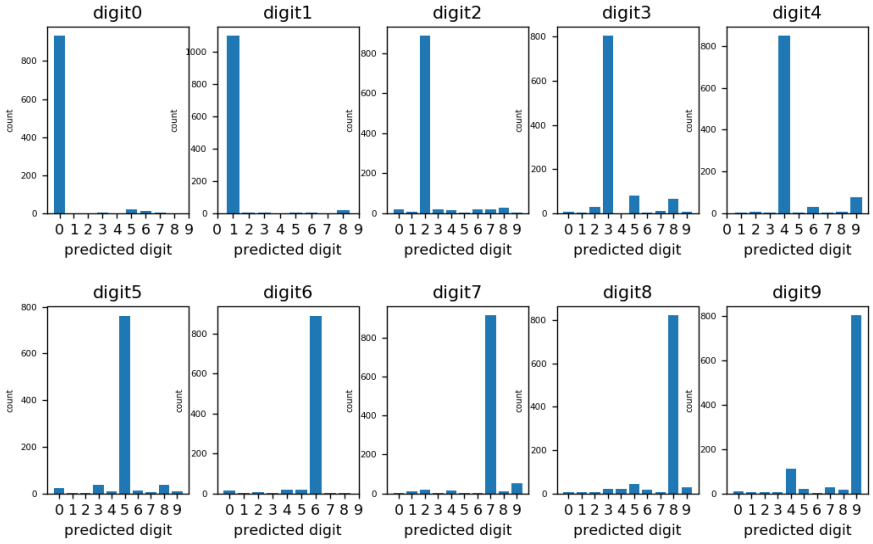

Figure 5: Error rate of each digit for neural network. The parameter is the 3rd row of batch training.

The neural network investigates achieved an error rate 12.4%, which is better than the two linear classifiers. There are far more tuning parameters to adjust compared with the two linear classifiers. For example, the number of nodes of each layer, number of iterations, and the choice of actuation functions. So I believe that there is more space for improvement than the two linear classifiers.

But the adjustment of λ is more intuitive since there is some geometrical explanation associated with λ. For neural networks, it is hard to know what impact the adjustment of one parameter would have. When I increase the number of nodes of the first layer, the error rate can be reduced. But when I increase the number of nodes of the second layer, the error rate increases. And there are multiple parameters for neural networks. It is very difficult to predict the combined effect of these parameters.

Another important point is that the same training sample can be used multiple times to improve the accuracy of the model for neural networks. I guess this resulted from the gradient descent of neural network to some extent. And the initialization of the weights are very important. When I initial all the weights to zero, the stochastic gradient descent algorithm does not work well. The weights of the first layer are always zero. The algorithm works much better when I use Gauss distribution for initialization.

In terms of efficiency, the training of the two linear classifiers are much faster than the neural networks. And even sometimes, more time is spent just to find that adjustment of the parameter of the neural networks will increase computation time but decrease accuracy. The adjustment of neural network can be very frustrating but has much more potential than the linear classifiers.

The batch training may be employed to accelerate the training process since this can save some computation time.