LogisticRegression and MLP for binary classification

Liu Chengtao 12212708

***Abstract*—This article mainly explains the effectiveness of LogisticRegression and MLP, written using Python and Numpy, in handling binary classification problems when adjusting its hyperparameters.**

***Index Terms*—Python, Numpy, Logistic Regression, MLP, Binary Classification, Machine Learning**

I. Introduction

In this article, I will mention the impact of using only Logistic Regression and continuously adjusting the number of neurons and hidden layers or change the tolerance in MLP on binary classification problems, and use the loss and advisor recall, precision, and F1 score to evaluate the performance of the binary classification model to evaluate its effectiveness.

II. Logistic regression

My LogisticRegression exists in "AI_project_LG" and set the initial learning rate to 0.01 and the number of iterations to 10000. Preprocess the data using the "__pretreatment" method and add a intercept term. Then, in the fit method, the initial weight is set to 0 and the "__sigmoid"" method is used as the activation function.(Which $\mathbf{z} = \mathbf{w}^T\mathbf{x} + \mathbf{w_0}$)

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

Loss is defined as the following equation, and is printed every 100 or 1000 iterations. (Which $h_i \cdot = sigmoid(z_i \cdot)$)

$$\text{loss} = -\frac{1}{m}\sum_{i=1}^{m}[y_i \cdot \log(h_i) + (1 - y_i) \cdot \log(1 - h_i)] \tag{2}$$

Use the following function to change the gradient:

$$\mathbf{dw} = \frac{1}{m}\mathbf{x}^T(\mathbf{h} - \mathbf{y}) \tag{3}$$

$$\mathbf{w} = \mathbf{w} - lr * \mathbf{dw} \tag{4}$$

After training, use function "predict_prob" to obtain output use the "predict" function to output predictions.

III. MLP base on logistic regression

My MLP exists in "AI_project_MLP" and set the initial learning rate to 0.01 , the number of iterations to 10000 and tolerance to 1e-5(10^-5).

You can manually input the number of hidden layers and input the number of neurons in each layers during training. Then randomly generate weights into arrays and layers corresponding to the size of the neuron.

Preprocess the data using the "__pretreatment" method and add a intercept term as what Logistic Regression does. Use the same sigmoid like Eq. (1) and defin its derivative in function "sigmoid_derivative" as:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \tag{5}$$

Let the sample set pass through multiple layers of neural networks in function "forward" to obtain output and we can obtain the error term through output:

$$\mathbf{error} = (\mathbf{h} - \mathbf{y})\sigma'(\mathbf{y}) \tag{6}$$

updateing the gradient for each layers using the following equation:

$$\mathbf{dw} = \mathbf{x}^T(\mathbf{error}) \tag{7}$$

$$\mathbf{w} = \mathbf{w} - lr * \mathbf{dw} \tag{8}$$

During training, record the difference between every two iterations, and stop training when the absolute value of this difference is greater than the tolerance level. When training is stopped due to loss convergence, the output will be the loss size and number of iterations at the time of stopping training under this tolerance. The function for calculating loss is as follows:

$$\mathbf{loss} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{h} - \mathbf{y})^2 \tag{9}$$

Finally, after the training is completed, the prediction is also obtained through the "predict" function

IV. Experiment results and analysis

A. Create data

Generate an array X with a shape of (n, 2) using the NumPy library, where each element is randomly selected from a standard normal distribution. And divide the dataset into two straight lines, $y = x$ and $y = -x$. Forming a dataset classified in an x-shape which you can see in the Fig. 1.

B. Only use Logistic Regression

From the chart in the Fig. 1.and data below, we can see that the fitting effect of logistic regression alone is relatively poor. The various data are Loss: 0.693, Accuracy: 0.596, Recall: 0.151, Precision: 0.875, F1 Score: 0.257

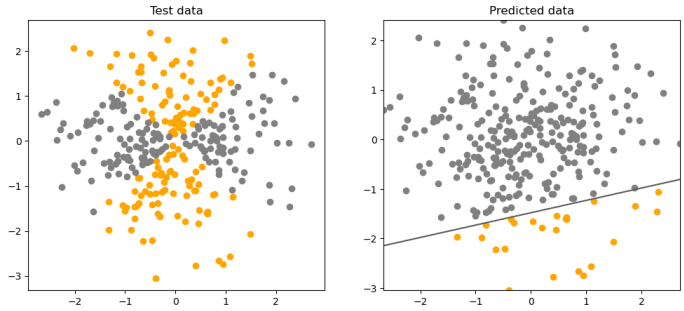

Fig. 1. Test data and predicted data from Logistic Regression

C. Use MLP for one hidden layers

In order to fit the data more accurately, we have decided to use MLP to fit the data. Firstly, we will only use one layers, and use 2, 4, 8, and 20 neurons to test their respective effects. When using the default tolerance in the neurons test, there was no convergence, so in the subsequent tests, the tolerance was changed to 1e-6 for testing. Learning rate and times set as default.

The data in the table consists of two, four, eight, and twenty neurons from top to bottom. From TABLE I, Fig. 2. and Fig. 3 , it can be seen that when there is only one hidden layers and two neurons, the convergence is faster, using only 1215 times. However, the fitting effect and loss are better than logistic regression, but the effect is still not good.

The convergence times of 4, 8, and 20 neurons are similar in 8000 times, and the loss gradually decreases during convergence. In one hidden layers, 4 neurons have been fitted more accurately, while 8 and 20 neurons have been fitted completely accurately, but they also require more time to calculate accordingly

Convergence times	Loss	Accuracy	Recall	Precision	F1 Score
1215	0.1523	0.7867	0.7410	0.7863	0.7630
8546	0.0253	0.9867	0.9856	0.9856	0.9856
8320	0.0132	1.0000	1.0000	1.0000	1.0000
8320	0.0104	1.0000	1.0000	1.0000	1.0000

TABLE I
2,4,8 AND 20 NEURONS IN ONE HIDDEN LAYERS

D. Use MLP for two hidden layers

In the previous section of the test, we tried the situation of one hidden layers. Now let's test the situation of two hidden layers. I tried to use different numbers of neurons in each layers to test, with specific data of 2 and 3 neurons, 3 and 6 neurons, 6 and 12 neurons ,12 and 20 neurons. Set the same learning rate, learning times and tolerance as one hidden layers.

Observing the loss curve in Fig. 4. , prediction results in Fig. 5. and Table II, we can see that when the number of neurons is set to 2 and 3, it does not converge in the end, but it has been fully fitted. However, when the number of neurons is set

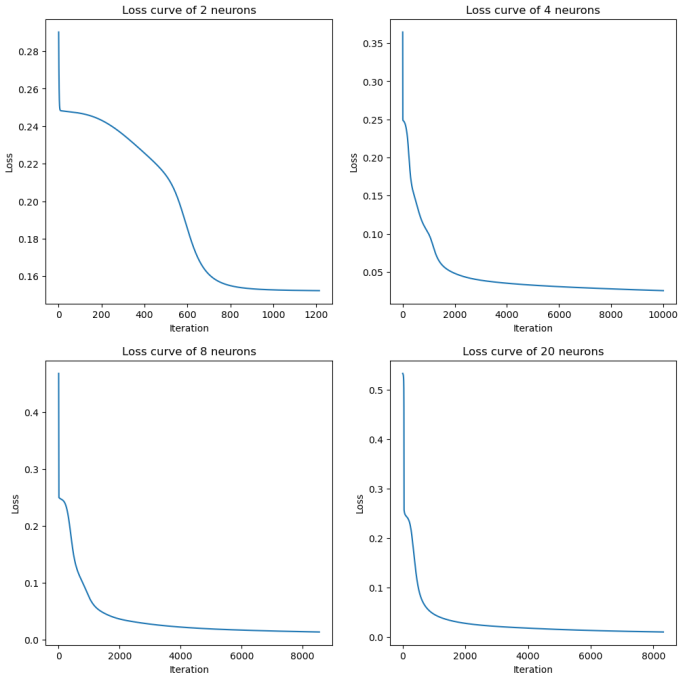

Fig. 2. one hidden layers loss curve

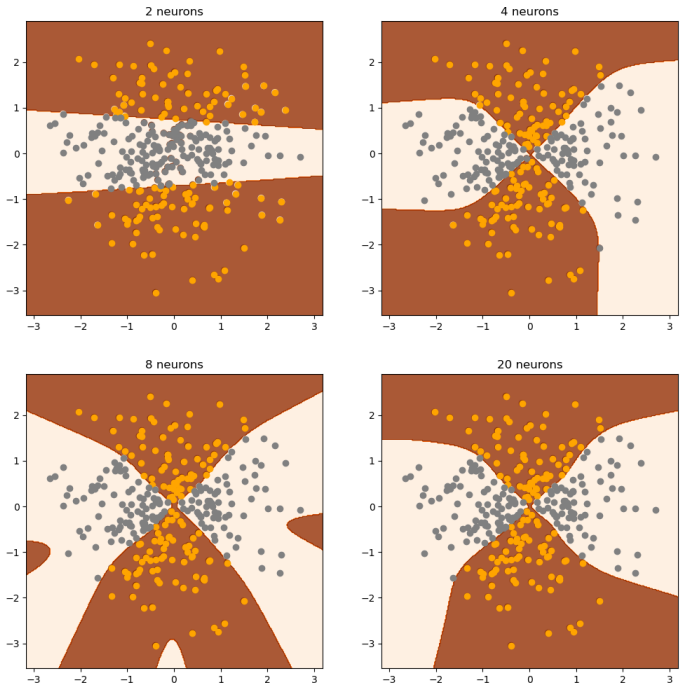

Fig. 3. one hidden with of differernt neurons

to 3 and 6, it converges at 7264 times, and the fitting effect is also equivalent to a complete fit. When the number of neurons is set to 6 and 12, it quickly converges at 43 times, but at this time, the loss value is still high, and the scores of Accuracy, Recall, Precision, and F1 are relatively low, reflecting poor fitting results at this time, which should be due to stucking in a local optimum. When the number of neurons is set to 12 and 20, convergence occurs only once. Observing the scores of each item, there should also be stucking in a local optimum.

Convergence times	Loss	Accuracy	Recall	Precision	F1 Score
-	0.0019	1.0	1.0	1.0	1.0
7264	0.0050	0.9967	0.9928	1.0	0.9963
43	0.2493	0.59	0.3597	0.5952	0.4484
1	0.5367	0.4633	1.0	0.4633	0.6332

TABLE II
2/3, 3/6, 6/12, AND 12/20 NEURONS IN TWO HIDDEN LAYERS

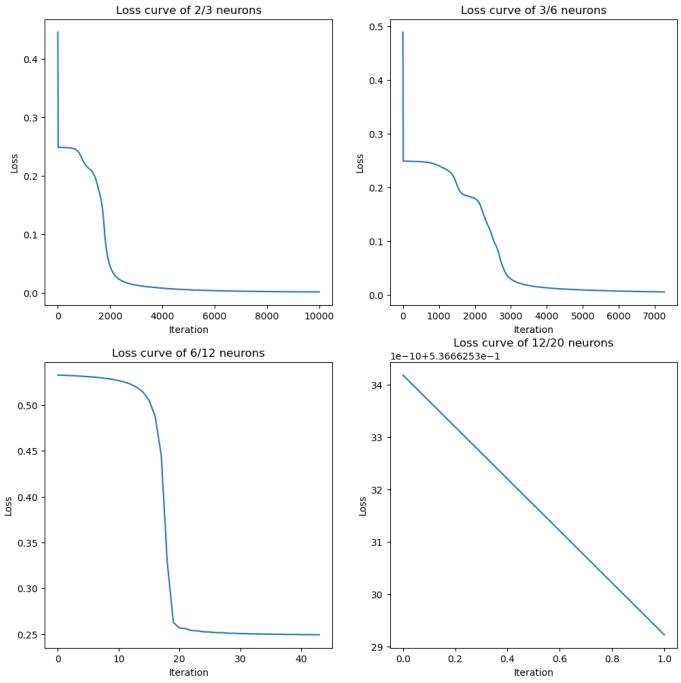

Fig. 4. two hidden layers loss curve

E. Use MLP for several hidden layers

After observing two hidden layers rusult, further testing is conducted on multiple hidden layers. As the prediction performance of the groups of 2 and 3 neurons in the two hidden layers is already good, 2 or 3 neurons were used in this test, respectively in 3 hidden layers and 6 hidden layers.

At first, I still used the same default parameters, but soon I found that at a tolerance of 1e-6, the loss quickly converged and did not achieve the training effect. Therefore, I adopted a tolerance of 1e-7 and obtained effective predictions

Observing Fig. 6. , Fig. 7. and Table III , we can see

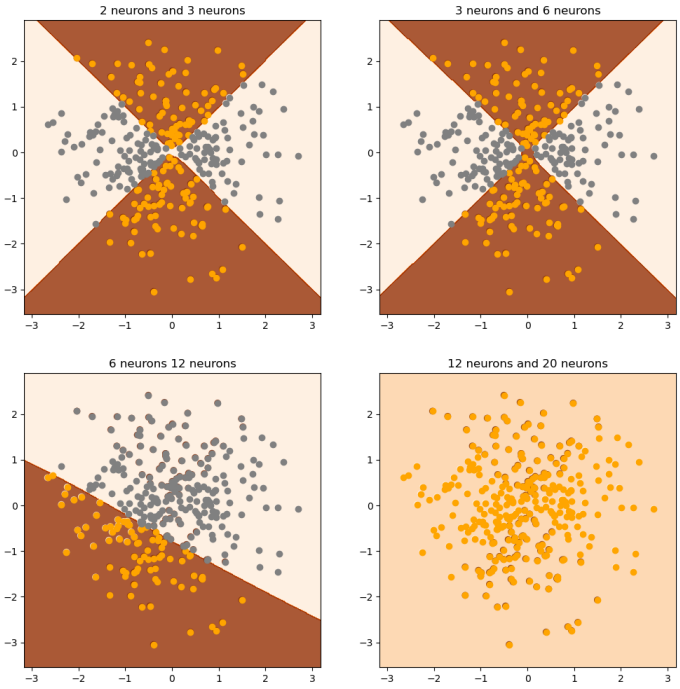

Fig. 5. two hidden layers

that when 2 neurons are in three hidden layers, the fitting effect is not good, and the obtained data is similar to that of using logistic regression alone. However, when the number of neurons in each layer increases to 3, although it does not converge before 10000 training sessions, the loss becomes very small and the fitting effect becomes very good. But for the 6 hidden layers, regardless of how many neurons there are, the loss converges early and the output prediction does not have any fitting effect. I guess it is because there are too many layers in the hidden layer, which leads to stucking in a local optimum.

Convergence times	Loss	Accuracy	Recall	Precision	F1 Score
-	0.1628	0.77	0.5036	1.0	0.6699
17	0.2487	0.5367	0.0	0.0	nan
-	0.0027	1.0	1.0	1.0	1.0
15	0.2487	0.5367	0.0	0.0	nan

TABLE III
2 OR 3 NEURONS IN SEV HIDDEN LAYERS

F. Use MLP for different tolerance

In our previous tests, we notice that different tolerances can also affect the test results. Considering that the fitting effect of 3 and 6 neurons on the two hidden layers was better in the previous tests, we gradually reduced the tolerance from 1e-5 to 1e-8 for the data under this setting to make predictions separately.

Observing Fig. 8. , Fig. 9. and Table IV, we can know that when the tolerance value is relatively small, such as 1e-5 or

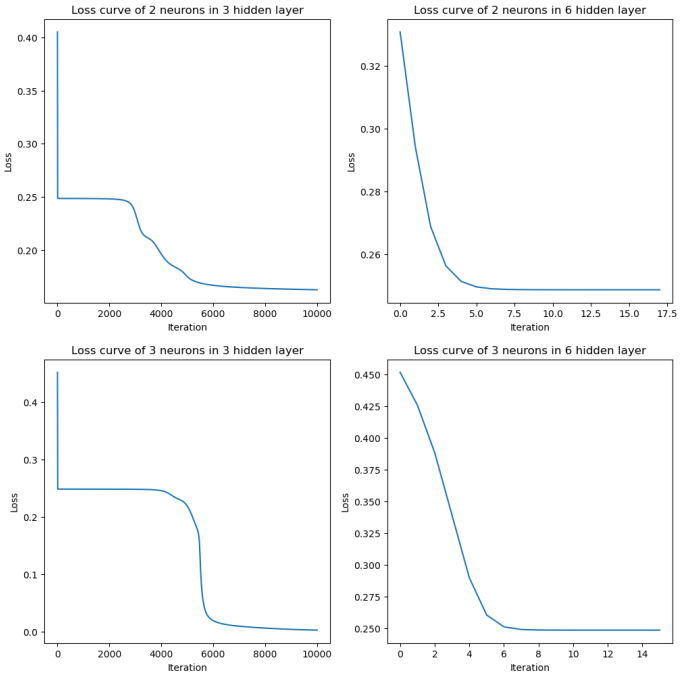

Fig. 6. several hidden layers loss curve

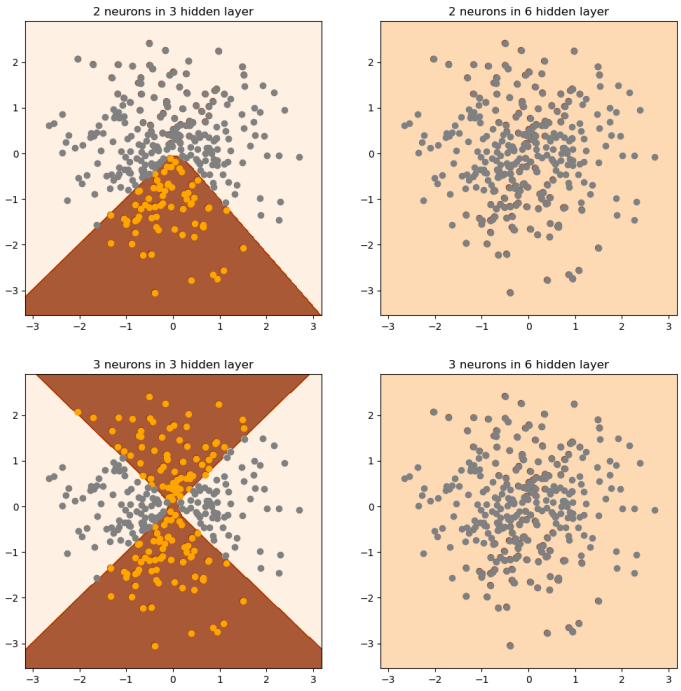

Fig. 7. several hidden layers

1e-6, the loss quickly converges and cannot achieve effective training results, resulting in incorrect prediction sets. When the tolerance value is further reduced, even though the loss does not converge, it still reaches a smaller value. When the tolerance is 1e-7 or 1e-8, trained model can predicte values output accurately fit.

However, additionally, we also noticed that there was a slight stucking in a local optimum phenomenon when the tolerance was set to a relatively small value of 1e-8, which reminded us not to set the tolerance too small.

Convergence times	Loss	Accuracy	Recall	Precision	F1 Score
7	0.2489	0.5367	0.0	0.0	nan
149	0.2488	0.5367	0.0	0.0	nan
-	0.0020	1.0	1.0	1.0	1.0
-	0.0601	0.9267	0.9496	0.8980	0.9231

TABLE IV
DIFFERENCE TOLERANCE FOR MLP

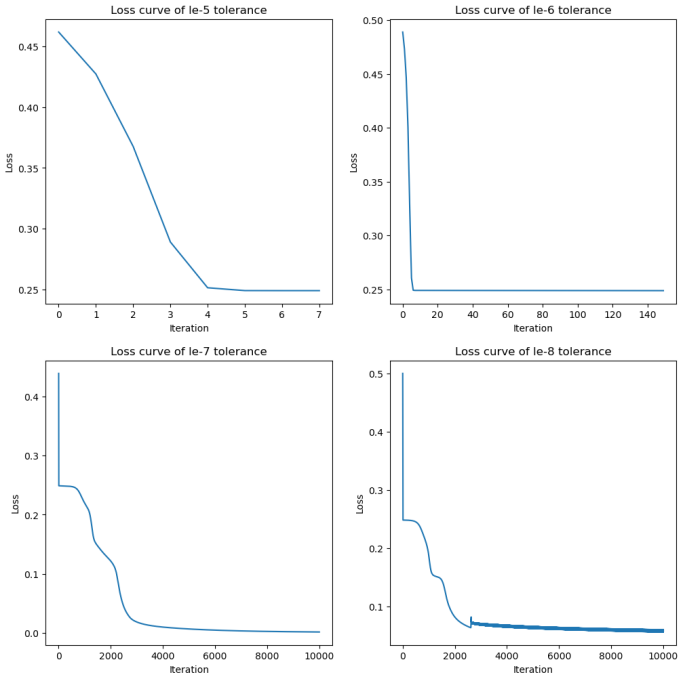

Fig. 8. difference tolerance loss curve

V. Conclusion and Future Problems, and Possibly References.

We tested the handling of binary classification problems using logistic regression and MLP in the above tests. We also tested the effects of changing the number of hidden layers and the number of neurons on the predicted output, and we also tested the effects of different tolerances on the predicted output.

In the above test, we found that the fitting effect of a simple logistic regression model is weak for relatively complex datasets. Furthermore, we attempted to use MLP to help us

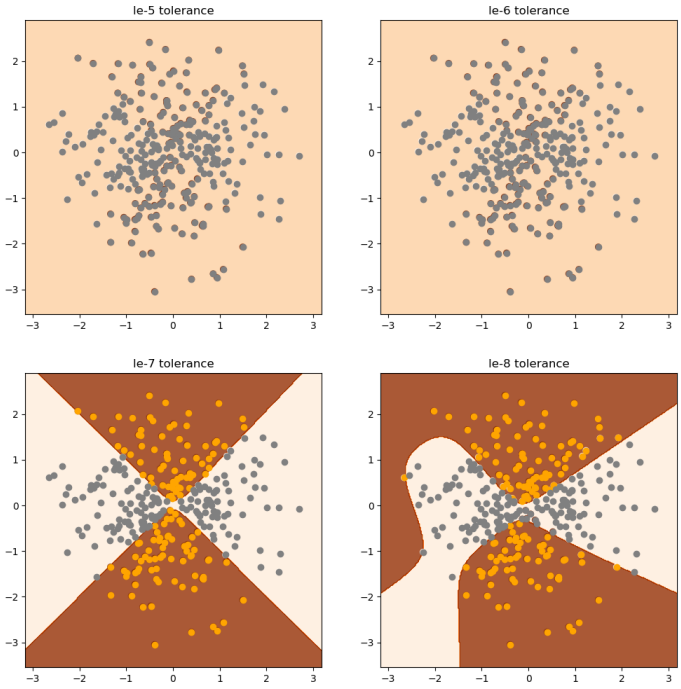

Fig. 9. different tolerance

better solve the problem. We also found that when the hidden layer is only one layer, too few neurons can lead to inaccurate fitting, while when there are many neurons, the fitting is still very accurate and there is no stucking in a local optimum phenomenon. However, when the hidden layer is increased to two layers, a small number of neurons (such as two or three) can also achieve good fitting results, When the number of neurons continues to increase by more than 8, stucking in a local optimum occurs. Next, we continue to test more situations in hidden layers. We found that too few neurons (such as 2 neurons) can achieve good fitting when there are more hidden layers, but when the number of hidden layers increases to 6, regardless of the number of neurons,stucking in a local optimum will occur and no good fitting output will be obtained.

In addition, we also tested the differences in prediction results under different tolerance levels and found that in cases with lower tolerance levels, it is easy to lead to early convergence of loss, resulting in poor training performance. So we need to choose a suitable tolerance for the current hyperparameters, which can prevent overfitting and ensure that the loss does not converge prematurely.

Finally, to summarize the conclusions of the previous test set, we need to choose the appropriate hidden layer and number of neurons. For example, for the less complex dataset used in this test, using a 2-3 layer hidden layer model with approximately 3 neurons per layer is sufficient to solve the problem and achieve good fitting results. At the same time, we also need to set appropriate learning rates, iterations, and tolerance levels.The learning rate set for this test is 0.01, with 10000 iterations and a tolerance of 1e-6 or 1e-7, which can already yield good prediction results.

Due to the fact that the dataset used in this test has never changed, it may be necessary to adjust the size of hyperparameters in the model when facing simpler or more complex datasets. At the same time, many of the data used in this test have undergone pre testing before proceeding to the next step of testing, in order to better illustrate the changes caused by hyperparameter changes. Due to limited space, the tested data was not presented, Please forgive me!