1. Problem Statement

1.1 Logistic Regression

In this question:

1.1.1 Find and select suitable data set from UCI machine learning database.

1.1.2 Implement logistic regression to selected training data, in using gradient descent to calculate best fitting weight theta. Here we could change the gradient function in order obtain global minimum.

1.1.3 Analysis the performance of the data.

1.2 Multilayer Perception

In this question:

1.2.1 Find and select suitable data set from UCI machine learning database.

1.2.2 Build a required neural network set weight or randomly set the weight to every layer of neural network.

1.2.3 Applying training data feedforward and using the output update the weight (BP).

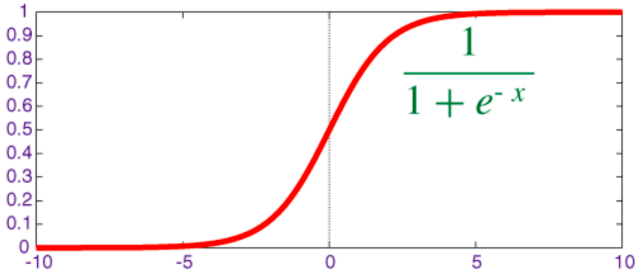
1.2.4 Implement trained NN test on testing set.

2 Proposed solution

2. 1 Logistic Regression is a classification method used to classify problems. You need to find a prediction function (h). The output of the function must be two values (respectively representing two categories), so use the Sigmoid function.

$$h_\theta(x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta^T x}}$$

$$P(y=1|x;\theta) = h_\theta(x)$$



$$P(y=0|x;\theta) = 1 - h_\theta(x)$$

The function output of sigmoid is between (0,1) and the intermediate value is 0.5, so the meaning of the previous formula hθ(x) is well understood, because the output of hθ(x) is between (0,1). Between them, it also indicates the probability that the data belongs to a certain category. hθ(x)<0.5 indicates that the current data belongs to class A, and hθ(x)>0.5 indicates that the current data belongs to class B. So, we can use the sigmoid function as the probability density function of the sample data.

The cost function:

$$\begin{cases} J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \cos t(h_\theta(x^{(i)}), y^{(i)}) \\ \cos t(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & y = 1 \\ -\log(1 - h_\theta(x)) & y = 0 \end{cases} \end{cases}$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)})]$$

To find the minimum value of J(θ), we can use the gradient descent method. According to the gradient descent method, the update process of θ can be obtained: (alpha: learning rate)

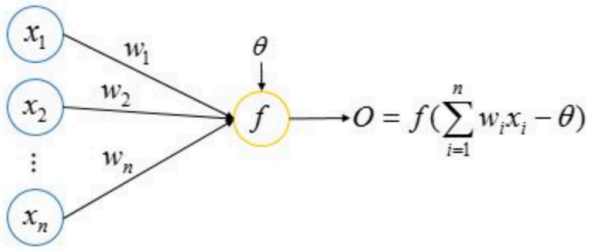
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Sometimes it is necessary to map the input values to polynomials, because the characteristics of the data may be small, and the response will lead to large deviations, so some combinations of features are created to enhance the data dimension.

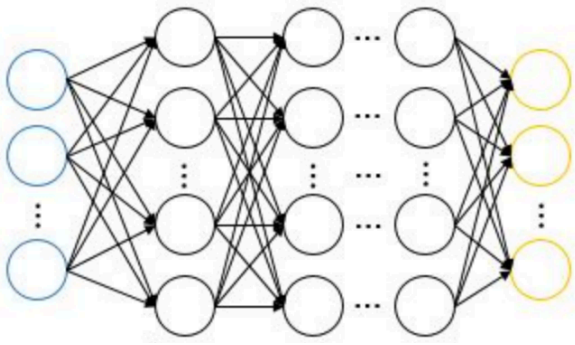
2. 2 The most basic component of a neural network is a neuron. In a neural network, each neuron is connected to other neurons, and the neurons send information to the next neuron through the connected path.

The perceptron consists of two layers of neurons, as shown in the following figure, where x_{i} represents the i-th input of the neuron, w_{i} represents the

weight of the i-th input link, and \theta represents the threshold of the neuron, The neuron that reaches the threshold, f represents the excitation function of the neuron, and O (Output) represents the output of the neuron.



The multi-layer neural network consists of the input layer, the hidden layer and the output layer. Each layer contains multiple neurons. The learning ability of the multi-layer neural network greatly exceeds the single-layer perceptron model, but it is necessary to train the multi-layer network. The learning rules of the perceptron are too simple to be used, so the classic error back propagation algorithm.



BP neural network is a kind of neural network model. It consists of an input layer, an output layer and one or more hidden layers. Its activation function uses sigmoid function and multi-layer feedforward neural network trained by BP algorithm. The BP algorithm is called the error back propagation algorithm. The basic idea of the algorithm is: In the feedforward network, the input signal is input through the input layer, output by the output layer through hidden layer calculation, and the output value is compared with the tag value. If there is an error, the error is reversed from the output layer to the input layer. In this process, the gradient weighting algorithm is used to adjust the neuron weights.

Training set:

$$D=\{(\boldsymbol{x}_1,\boldsymbol{y}_1),(\boldsymbol{x}_2,\boldsymbol{y}_2),\cdots,(\boldsymbol{x}_m,\boldsymbol{y}_m)\},\boldsymbol{x}_m\in\mathbf{R}^d,\boldsymbol{y}_m\in\mathbf{R}^l$$

d, the number of input neurons

l, the number of output neurons

q, the number of neurons in the hidden layer

θ_j, the threshold of the jth neuron in the output layer

γ_h, the threshold of the hth neuron in the hidden layer

b_h, the output of the hth neuron in the hidden layer

vih, the i-th neuron of the input layer and the link weight of the h-th neuron in the hidden layer

w_{hj} ,the weight of the h-th neuron in the hidden layer and the j-th neuron in the output layer

$\alpha_h=\sum_{i=1}^{d}v_{ih}x_i$, the input of the hth neuron in the hidden layer

$\beta_j=\sum_{h=1}^{q}w_{hj}b_h$, the input of the jth neuron in the output layer

Output:

$$\hat{y}_j^k=f(\beta_j-\theta_j)$$

Mean square error:

$$E_k=\frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k-y_j^k)^2$$

Gradient descent:

$$v=v+\Delta v$$

$$\Delta v=-\eta\frac{\partial E_k}{\partial v}$$

Chained rule:

$$\frac{\partial E_k}{\partial\theta_j}=\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\theta_j}\quad\frac{\partial E_k}{\partial\hat{y}_j^k}=\hat{y}_j^k-y_j^k\quad\frac{\partial\hat{y}_j^k}{\partial\theta_j}=-\hat{y}_j^k(1-\hat{y}_j^k)$$

$$\frac{\partial E_k}{\partial\theta_j}=\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\theta_j}=\hat{y}_j^k(1-\hat{y}_j^k)(y_j^k-\hat{y}_j^k)$$

$$g_j=\frac{\partial E_k}{\partial\theta_j}=\hat{y}_j^k(1-\hat{y}_j^k)(y_j^k-\hat{y}_j^k)$$

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \beta_j}{\partial w_{hj}} \quad \frac{\partial \hat{y}_j^k}{\partial \beta_j} = \hat{y}_j^k (1 - \hat{y}_j^k) \quad \frac{\partial \beta_j}{\partial w_{hj}} = b_h$$

$$\frac{\partial E_k}{\partial w_{hj}} = -g_j b_h$$

$$\frac{\partial E_k}{\partial \gamma_h} = \frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \gamma_h}$$

$$\frac{\partial E_k}{\partial b_h} = \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} = -\sum_{j=1}^{l} g_j w_{hj}$$

$$\frac{\partial b_h}{\partial \gamma_h} = \frac{\partial}{\partial \gamma_h} f(\alpha_h - \gamma_h) = -f'(\alpha_h - \gamma_h) = -b_h(1 - b_h)$$

$$\frac{\partial E_k}{\partial \gamma_h} = b_h(1 - b_h) \sum_{j=1}^{l} g_j w_{hj}$$

As long as you know the threshold gradient of the upper layer of neurons, you can calculate the threshold gradient and link weight of the current neuron.

$$g_h^{(m)} = b_h^{(m)}(1 - b_h^{(m)}) \sum_{j=1}^{l} w_{hj}^{(m+1)} g_j^{(m+1)}$$

$$p_{hj}^{(m)} = -g_j^{(m)} b_h^{(m-1)}$$

$$g_h^{(m)} = b_h^{(m)}(1 - b_h^{(m)}) \sum_{j=1}^{l} w_{hj}^{(m+1)} g_j^{(m+1)}$$

$$p_{hj}^{(m)} = -g_j^{(m)} b_h^{(m-1)}$$

According to the above formula, the threshold gradient of the output neurons can be calculated, so that the gradient of the neuron threshold and the connection weight of the entire network can be calculated, thereby completing the training network.

3 Implementation details

3.1 Logistic Regression

The data is first read from the data set and the data is split into training and test sets. The gradient descent algorithm is applied using the training set data, and the weights are iteratively updated to maximize the maximum likelihood probability. A non-linear gradient descent can also be used here where the input training data is mapped to a high dimension. Draw a data scatter plot by visualizing and plot the classification boundaries on the graph. Use the calculated weights to test the model using a split test set and output accuracy. Draw a test data scatter plot and mark the classification boundaries. Linear gradient descent and nonlinear gradient descent methods are implemented here.

The Horse Colic dataset was used to train and test the model, using the Stochastic Gradient Descent method and the random Stochastic Gradient Descent. I hope to avoid local minima. You can call multiple test data to find the average error rate.

Multi-class logistic regression is applied to handwritten digit recognition. In categorization, the pre-imitation method treats the current class as a class, and all other classes as one class, which translates into a two-category problem. The data set y gives the numbers 0, 1, 2...9, and the logical regression requires a label of 0/1, so the response y is processed when different classes are obtained. The theta is solved using the gradient descent method. The result of the prediction is a probability value, which is substituted into the predicted sigmoid function by using the learned theta. The maximum value of each row is the maximum probability of a certain number, and the column number is the true value of the predicted number, because in the classification When all 0 is mapped to y in the first column, 1 is mapped in the second column, and so on.

Problem:

When using the SGD method, if you do not use random SGD, there will be a local minimum, not a global minimum.

Over-fitting occurs when multi-classification problems are reduced using traditional gradients

3.2 Multilayer Perception

Read data from the data set and add the data to a dimension of all 1. The data was randomly divided into a training set and a test set. Define the parameters of the neural network. The number of input neurons is the same as the data dimension. Set a hyperparameter of the number of neurons in the hidden layer. The number of neurons in the output layer is the same as the dimension of the data label. The training model was set up 80 times. Each time the training data was used to train the model, the model was tested using the test data set after each training session to obtain the total error.

In the training process, using the expected output and actual output of the network, the partial derivative of the error function to each neuron in the output layer is calculated, and the connection weight of the hidden layer to the output layer, the output layer, and the output of the hidden layer are calculated. The partial derivative of the error function for each neuron in the hidden layer. The connection weights are corrected using the outputs of the neurons of the output layer and the neurons of the hidden layer. The connection weights are corrected using the input parameters of each neuron of the hidden layer and the neurons of the input layer. Calculate global error.

Problem:

When classifying, different error layers can be adjusted to different error results, and multiple experiments can find relatively good results.

The accuracy of multi-classification is not very high.

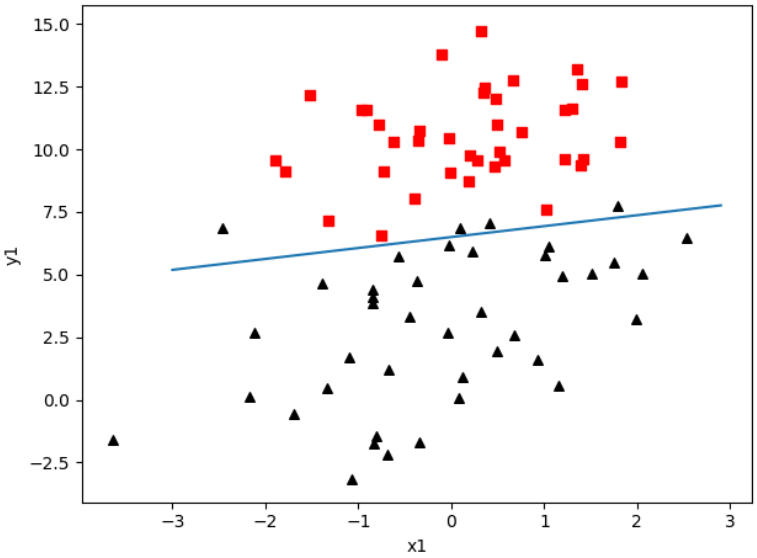
The main problem is that python is not skilled.

4 Result and Discussion

4.1 Logistics Regression

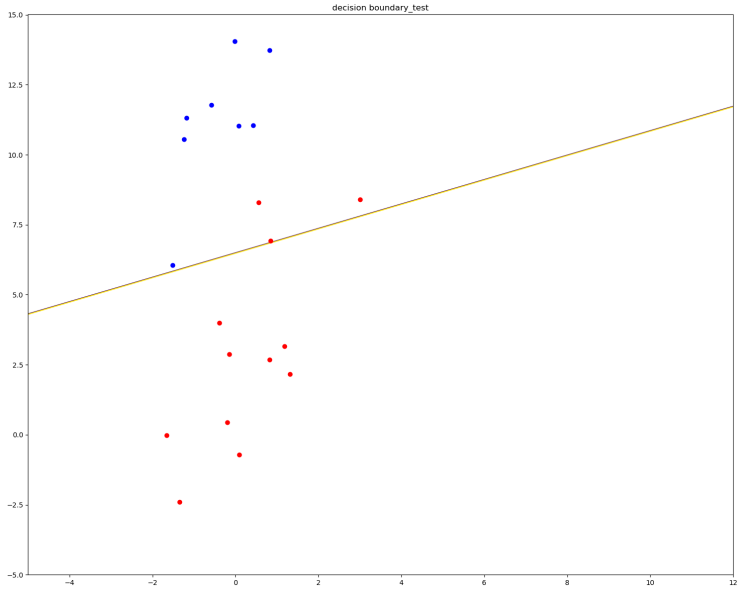
4.1.1

Training data set is plot as :



There are some data around decision boundary, thus there are some miss classfiy.

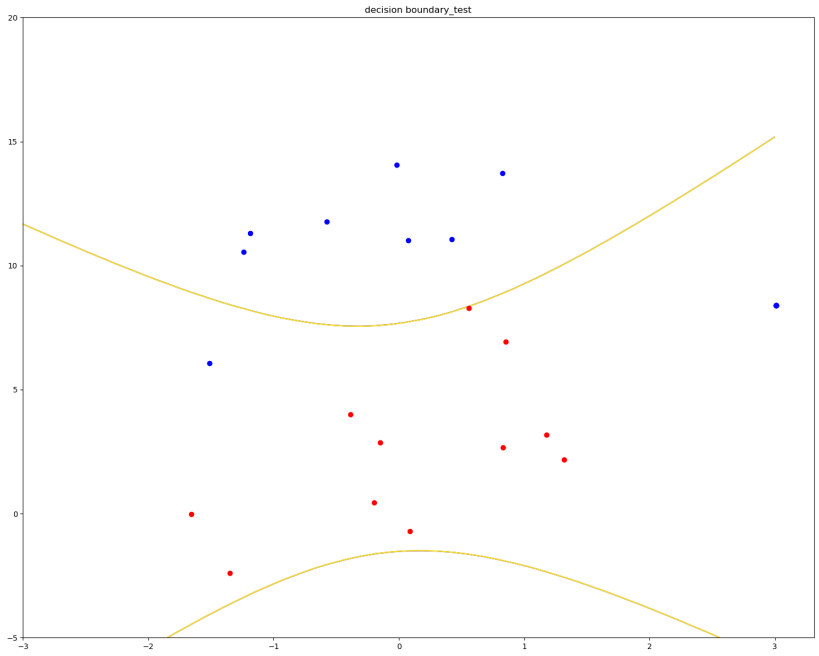
Testing data set is plot as:




```
linear function theta: [[ 3.77689334]
 [ 0.25354764]
 [-0.58140762]]
linear function on testing set accuracy: 85.000000%
```

Training data may be a little few thus the decision boundary should be relative higher. And the linear model may not fit well towards the data.

Testing data set in non-linear logistic regression plot as:




```
non-linear function theta: [[ 4.69322307]
 [-0.62292094]
 [ 2.46838894]
 [ 3.73980728]
 [ 0.40876584]
 [-0.40145618]]
non-linear function on testing set accuracy:95.000000%
```

Non-linear regression fits data well.

Using Horse Colic dataset:

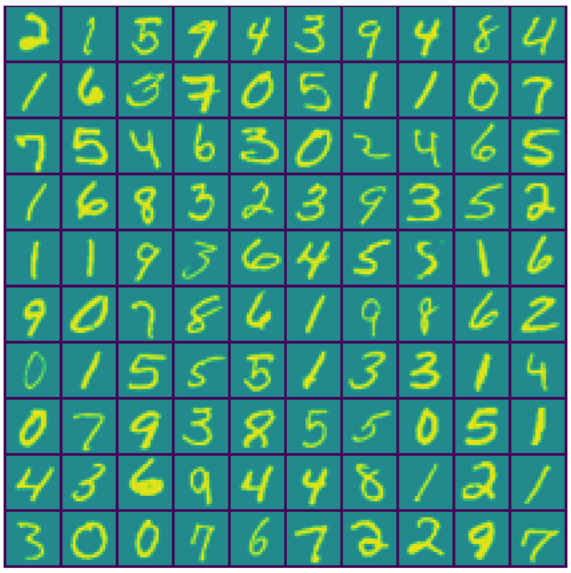
```
using separate dataset on horse-colic.data AND horse-colic.test
GD:
the error rate is 0.2835820895522388
SGD:
the error rate is 0.5970149253731343
random SGD:
the error rate is 0.31343283582089554
```

I think the reason why SGD performance not that well is because that it may reaches some local minimum. And the random SGD's output is not that stable. 20 times average performance is:

```
multi SGD (20 times avg):
the error rate is 0.29850746268656714
the error rate is 0.29850746268656714
the error rate is 0.2537313432835821
the error rate is 0.417910447761194
the error rate is 0.44776119402985076
the error rate is 0.29850746268656714
the error rate is 0.417910447761194
the error rate is 0.23880597014925373
the error rate is 0.2835820895522388
the error rate is 0.373134328358209
the error rate is 0.3880597014925373
the error rate is 0.31343283582089554
the error rate is 0.3283582089552239
the error rate is 0.23880597014925373
the error rate is 0.2537313432835821
the error rate is 0.2835820895522388
the error rate is 0.29850746268656714
the error rate is 0.29850746268656714
the error rate is 0.40298507462686567
the error rate is 0.29850746268656714
after 20 iteration the average error rate is 0.3216417910447761
```

4.1.2 K-class logistic regression

Origin handwritten digit is shown as:



```
predict accuracy is : 96.480000%
```

Using scikit-learn towards data set:

```
Using scikit-learn predict accuracy is : 94.380000%
```

4.2 BP

Hidden layer = 2

```
epoch:0, test error 0.097699
epoch:1, test error 0.103955
epoch:2, test error 0.111844
epoch:3, test error 0.068061
epoch:4, test error 0.034574
epoch:5, test error 0.018842
epoch:6, test error 0.012034
epoch:7, test error 0.008712
epoch:8, test error 0.006841
epoch:9, test error 0.005575
epoch:10, test error 0.004625
epoch:70, test error 0.000365
epoch:71, test error 0.000359
epoch:72, test error 0.000354
epoch:73, test error 0.000349
epoch:74, test error 0.000344
epoch:75, test error 0.000339
epoch:76, test error 0.000333
epoch:77, test error 0.000328
epoch:78, test error 0.000324
epoch:79, test error 0.000320
```

Hidden layer = 4

```
epoch:0, test error 0.116556
epoch:1, test error 0.116286
epoch:2, test error 0.111152
epoch:3, test error 0.117771
epoch:4, test error 0.117323
epoch:5, test error 0.113640
epoch:6, test error 0.112036
epoch:7, test error 0.112771
epoch:8, test error 0.114903
epoch:9, test error 0.111072
epoch:10, test error 0.111093
epoch:70, test error 0.000783
epoch:71, test error 0.000756
epoch:72, test error 0.000733
epoch:73, test error 0.000715
epoch:74, test error 0.000691
epoch:75, test error 0.000682
epoch:76, test error 0.000663
epoch:77, test error 0.000653
epoch:78, test error 0.000627
epoch:79, test error 0.000602
```

Hidden layer = 6

```
epoch:0, test error 0.114578
epoch:1, test error 0.092569
epoch:2, test error 0.044757
epoch:3, test error 0.021710
epoch:4, test error 0.012728
epoch:5, test error 0.008608
epoch:6, test error 0.006468
epoch:7, test error 0.005246
epoch:8, test error 0.004259
epoch:9, test error 0.003626
epoch:70, test error 0.000333
epoch:71, test error 0.000328
epoch:72, test error 0.000321
epoch:73, test error 0.000315
epoch:74, test error 0.000309
epoch:75, test error 0.000305
epoch:76, test error 0.000300
epoch:77, test error 0.000296
epoch:78, test error 0.000291
epoch:79, test error 0.000288
```

Hidden layer = 8

```
epoch:0, test error 0.102704
epoch:1, test error 0.092687
epoch:2, test error 0.054574
epoch:3, test error 0.021307
epoch:4, test error 0.012621
epoch:5, test error 0.008717
epoch:6, test error 0.006039
epoch:7, test error 0.004885
epoch:8, test error 0.004239
epoch:9, test error 0.003459
epoch:10, test error 0.002980
epoch:70, test error 0.000311
epoch:71, test error 0.000307
epoch:72, test error 0.000302
epoch:73, test error 0.000297
epoch:74, test error 0.000293
epoch:75, test error 0.000288
epoch:76, test error 0.000285
epoch:77, test error 0.000279
epoch:78, test error 0.000276
epoch:79, test error 0.000272
```

For multiclass BP

Hidden layer = 6

```
epoch:0, accuracy 0.377778
epoch:1, accuracy 0.666667
epoch:2, accuracy 0.622222
epoch:3, accuracy 0.644444
epoch:4, accuracy 0.622222
epoch:5, accuracy 0.644444
epoch:6, accuracy 0.622222
epoch:7, accuracy 0.622222
epoch:8, accuracy 0.977778
epoch:9, accuracy 0.622222
epoch:10, accuracy 0.977778
```

```
epoch:70, accuracy 0.866667
epoch:71, accuracy 0.933333
epoch:72, accuracy 0.955556
epoch:73, accuracy 1.000000
epoch:74, accuracy 0.977778
epoch:75, accuracy 0.933333
epoch:76, accuracy 0.955556
epoch:77, accuracy 1.000000
epoch:78, accuracy 1.000000
epoch:79, accuracy 0.955556
```

5 Reference