Web Economics Assignment (Part B)

Weijie Huang
Web Science and Big Data Analytics
Weijie.huang@ucl.ac.uk

Step1: Download the data. (initial work)

The data are put in /dataset folder,
the output figures are located in /Figure output folder,
the plot.py is used to plot the curve, figures,
the util.py contains all the functions with a detailed comment,
the Spambase dataset classification.py is the main functions.
In this assignment, I used both the "Seaborn" [1] and "Matplotlib" [4] library to plot the figure, and some of the functions such as log () or roc_auc_score was imported from "Numpy" and "Sklearn".
The complete project's source code is available to access from the GitHub page: "https://github.com/koalagreener/Spambase-dataset-classification."

Step2: Partition the data into 10 folds.

```
# 10-Fold
for i in range(10):
    count = 1
    temp_for_fold_test = []
    temp_for_fold_train = []
    with open(filename) as infile:
        for line in infile:
            if((count + i) % 10 == 0):
                temp_for_fold_test.append(line)
                count += 1
            else:
                temp_for_fold_train.append(line)
                count += 1
        # Z-score Format of both the training dataset and test dataset
    K_fold_training_dataset.append(temp_for_fold_train)
    K_fold_test_dataset.append(temp_for_fold_test)
```

While reading through the file, we separated the file into ten folds, and every time we use one of the folds to used for testing, the rest of them are used for training.

Step3: Precondition the data.

In this step, I calculated the sum, mean, standard value respectively, and then used the following formula to transform the training dataset into the z-score format. So all of the training datasets are now storing in "trainingDataset" list. And also, I had created the Theta List in this step, and all of the theta value are set to 0 at the beginning. Then we can use the shuffle function in random.py to shuffle the data.

$$\text{Z-score} = \frac{(\text{feature value X} - \text{mean value})}{(\text{Standard deviation})}$$

Step4 – Step6: Linear regression learner trained via stochastic gradient descent (SGD)

In this part, I used the Seaborn [1] and matplotlib package [4] to plot the curves. And the following three graphs are the result of three various learning curves that trained via SGD with different learning rates while the X axis is the iteration, and the Y axis is mean squared error (MSE). You should notice that the measurement of the axis is different in this three graphs.

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}$$

}

}

Formula.1 Theta list trained via stochastic gradient descent

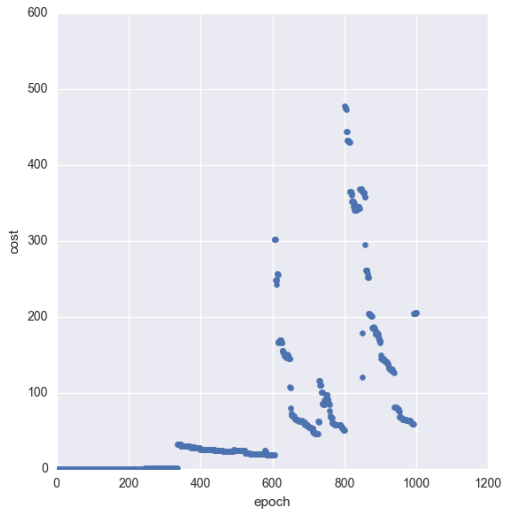

Figure.1 mean squared error vs iteration (SGD, learning rate = 0.01)

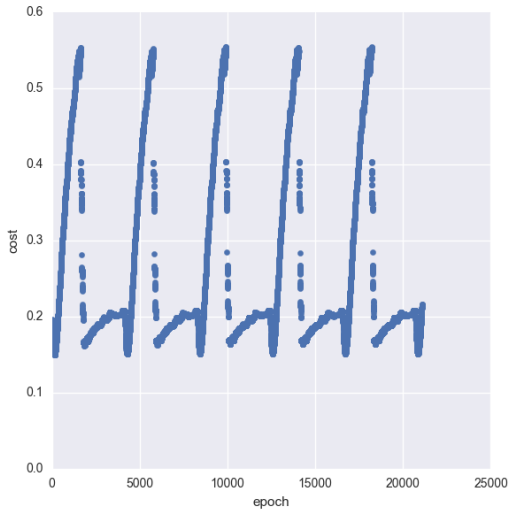

Figure.2 mean squared error vs iteration (SGD, learning rate = 0.001)

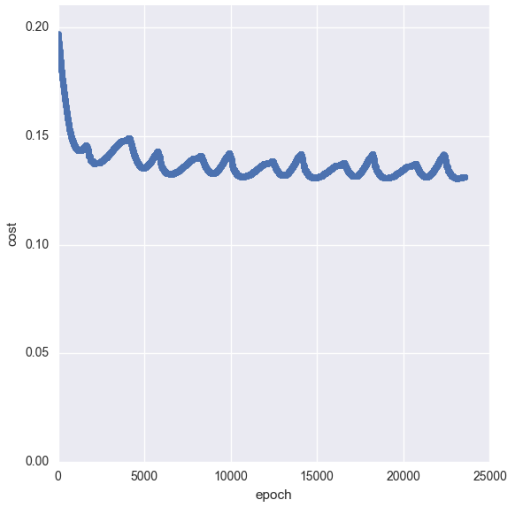

Figure.3 mean squared error vs iteration (SGD, learning rate = 0.0001)

	Learning rate	Min MSE
Figure1	0.01	0.161840911
Figure2	0.001	0.150537496
Figure3	0.0001	0.130540309

Table.1 Min mean squared error value of relevant learning rates

As we can observe from the figures and table above, the convergence rates are varied while the learning rates are different. And the best result comes from the smallest learning rate, but at this moment, the epoch of the training has reached more than 20000 times which means the gradient descent converge slowly. Finally, we choose 0.0001 as our parameter to draw the ROC curve.

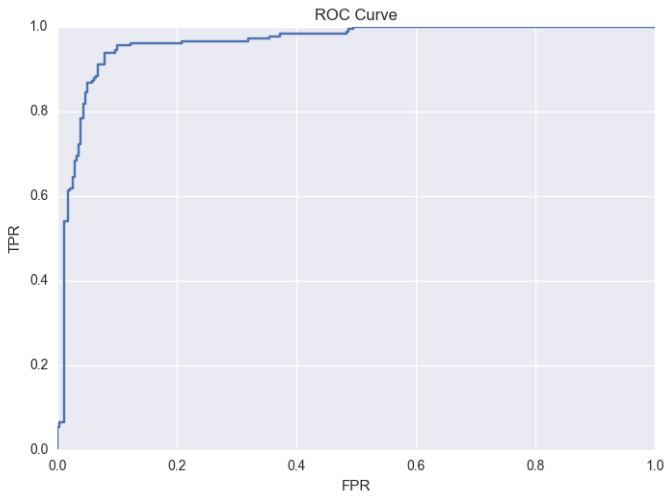

Figure.4 ROC curve of SGD (learning rate = 0.0001, AUC = 96.1%)

The AUC of the ROC curve can be calculated simply using the trapezoidal rule [2] by sklearn library [3], and it is 96.1% (0.96166260718).

Step4 – Step6: Linear regression learner trained via batch gradient descent (BGD)

$$\text{Repeat until convergence } \{$$

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad (\text{for every } j)$$

$$\}$$

Formula.2 Theta list trained via batch gradient descent

This times we will use the batch gradient descent (BGD) to train the linear regression model.

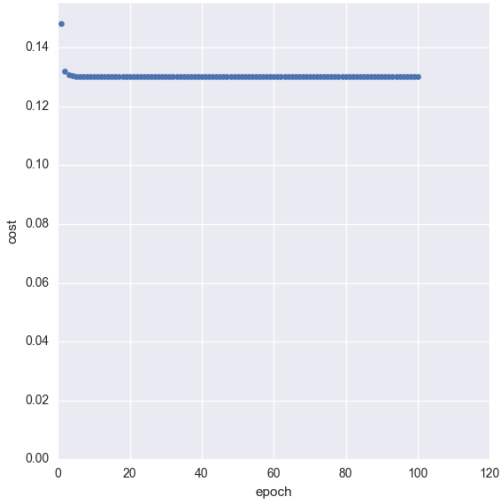

Figure.5 mean squared error vs iteration (BGD, learning rate = 0.0001)

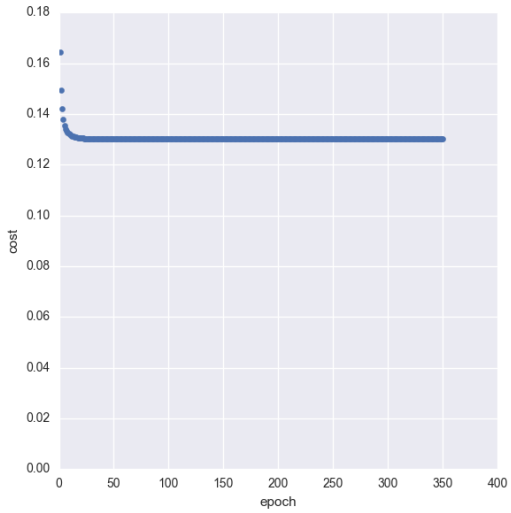

Figure.6 mean squared error vs iteration (BGD, learning rate = 0.00001)

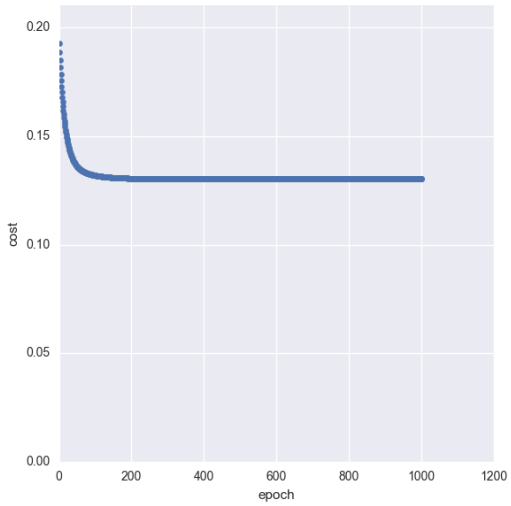

Figure.7 mean squared error vs iteration (BGD, learning rate = 0.000001)

	Learning rate	Min MSE
Figure5	0.0001	0.130181147
Figure6	0.00001	0.130181345
Figure7	0.000001	0.130181551

Table.2 Min mean squared error value of relevant learning rates

As we can see from the table.2, there is no big difference between different learning rate when calculated the MSE, so we just use the learning rate = 0.0001 to plot the ROC curve.

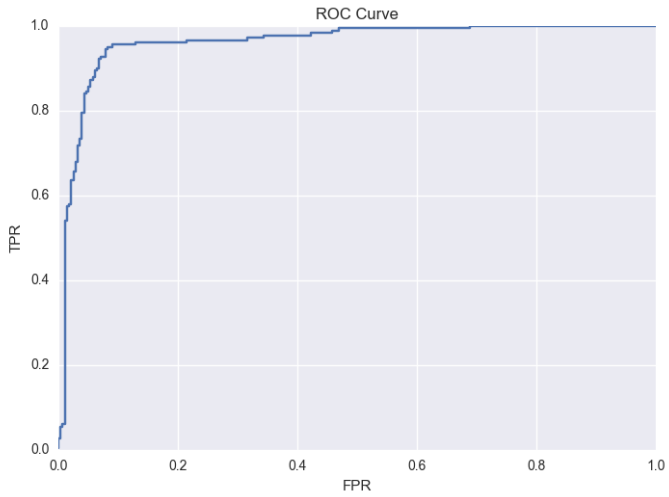

Figure.8 ROC curve of BGD (learning rate = 0.0001, AUC=96.12%)

The AUC of figure.8 can be calculated simply using the trapezoidal rule [2] and it is 96.12% (0.961246757362). The result is quite closed to the SGD one because both of their result of cost function are about 0.13.

What's more, if we compare those figures, we can observe that the convergence rate of SGD and BGD are getting stable after 6000 and 30 times respectively.

Step7: logistic regression learner trained via SGD and BGD

As the assignment said, there is only some minor change between linear one and logistic one.

In step5, we need to apply the sigmoid function and log when calculated the cost function.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Figure.9 The cost function in logistic regression is different

Want $0 \le h_\theta(x) \le 1$

$$h_\theta(x) = g(\theta^T x)$$

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

Sigmoid function

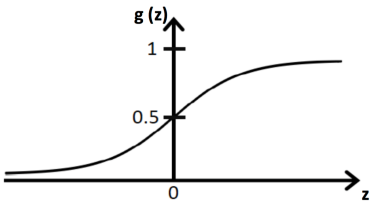

Figure.10 The sigmoid function being used in H(x)

But in this stage, I found that the accuracy of float in Python may affect the result. While the input X of the sigmoid function is quite large, the output value of the sigmoid function is approaching to 1. And if the y equals to 0 at this moment, the cost will be calculated as -log (1 - 1), while the log may encounter some problems here. So I made a trick here, if the input of the sigmoid function is larger than 10, I may directly set it to 10 because there is a minor difference between the result but it can avoid the log (0) problem.

Then, I set the learning rate 0.01, 0.001, 0.0001 using SGD, and here are the results.

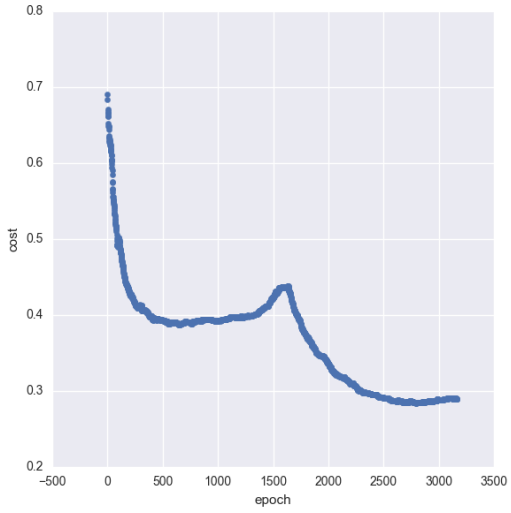

Figure.11 logistic, SGD, learning rate = 0.01

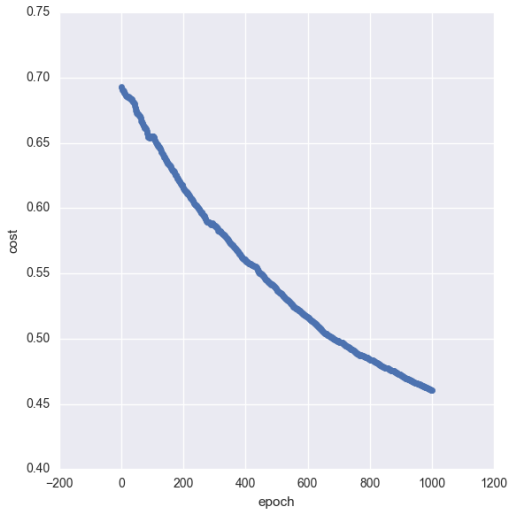

Figure.12 logistic, SGD, learning rate = 0.001

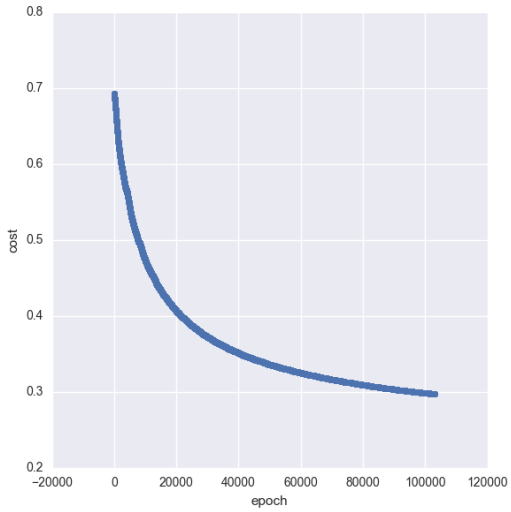

Figure.13 logistic, SGD, learning rate = 0.0001

And also, the same as SGD, the BGD one's learning rate is 0.001 and 0.0001 separately.

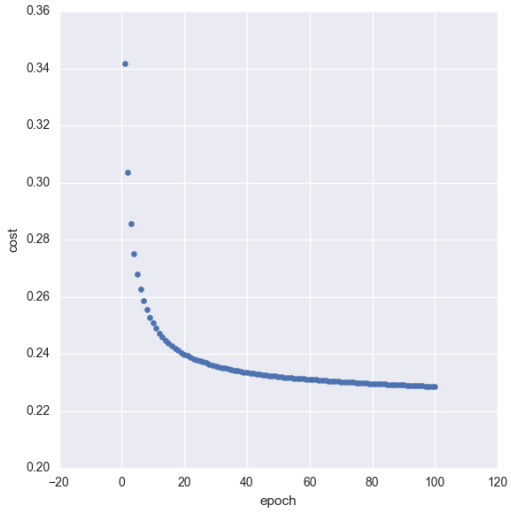

Figure.14 logistic, BGD, learning rate = 0.001

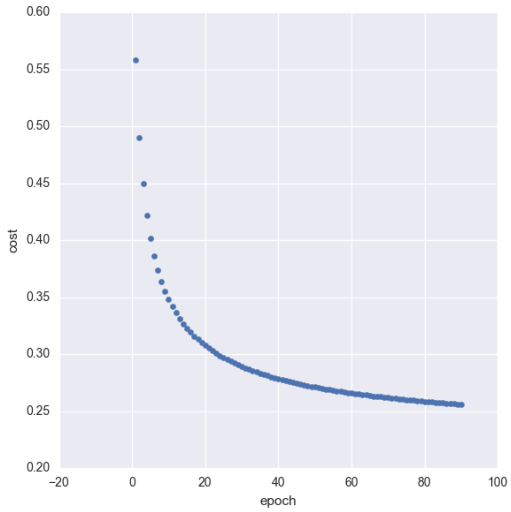

Figure.15 logistic, BGD, learning rate = 0.0001

In the logistic regression, the SGD and BGD gain a “good” result of cost after 100000, and 90 times respectively. So, we can now plot the Roc curve by choosing the best parameters.

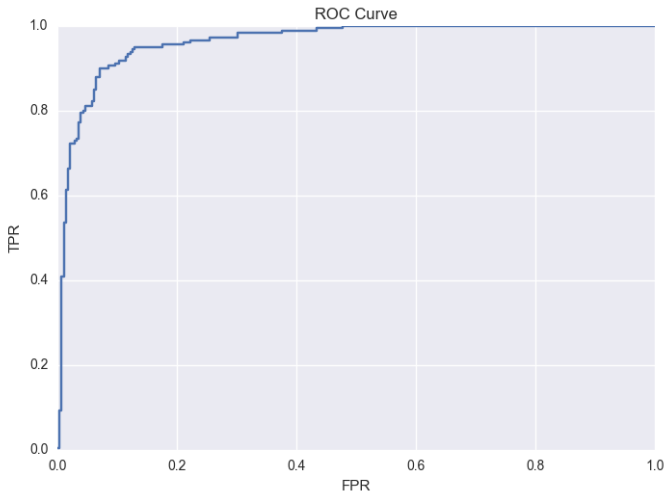

Figure.16 ROC curve of SGD (learning rate = 0.01, AUC=96.29%)

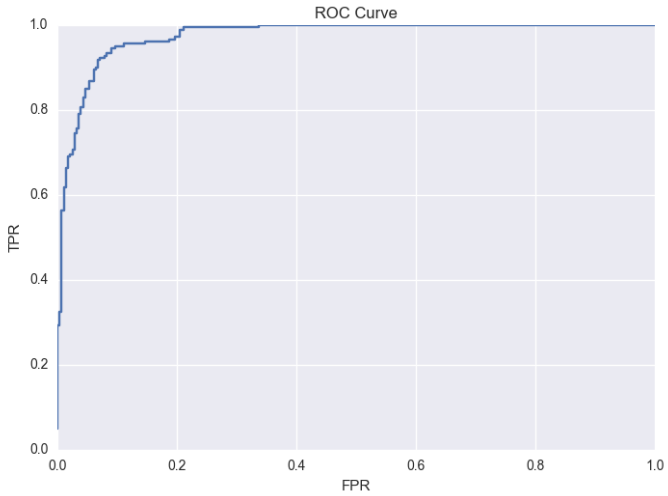

Figure.17 ROC curve of BGD (learning rate = 0.01, AUC=97.39%)

Summary

- In all of the cost function results, the values are no more than 0.2, so all of their AUC values are so close to 96%.
- If we print out the trained theta list, we can observe that some of the value are larger than others, so we can compare them to the attributes list to see which factors they are related. And that is why linear regression and logistic regression are being used widely because they are both easy to explain the weights of the factors.
- If we compare the epoch between SGD and BGD, we can find that the SGD one are much larger than the BGD's, which means the BGD convergence faster than the SGD. Because the BGD use the whole dataset to train the theta list, and 1 epoch in BGD are the same as 4100 epochs in SGD.

Reference:

[1] https://stanford.edu/~mwaskom/software/seaborn/#
[2] https://en.wikipedia.org/wiki/Trapezoidal_rule
[3] http://scikit-learn.org/stable/index.html
[4] http://matplotlib.org
[5] https://docs.eyesopen.com/toolkits/cookbook/python/plotting/roc.html
[6] http://alexkong.net/2013/06/introduction-to-auc-and-roc/