# Web Economics Assignment (Part B)

Weijie Huang

Web Science and Big Data Analytics

[Weijie.huang@ucl.ac.uk](mailto:Weijie.huang@ucl.ac.uk)

## Step1: Download the data.

The data are put in /dataset folder, and the whole project are available to access from the github page: “https://github.com/koalaGreener/Spambase-dataset-classification”

## Step2: Partition the data into 10 folds.

**with** open(filename) **as** infile:  
 **for** line **in** infile:  
 **if**(count % 10 == 0):  
 testDataset.append(line)  
 count += 1  
 **else**:  
 trainingDataset.append(line)  
 count += 1

While read through the file, we separated the file into 10 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 z-score format. So all of the training dataset are now storing in “trainingDatasetInZScoreFormat” 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.

Z-score =

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

In this part, I used the Seaborn [1] and matplotlib package [3] to plot the curves. And the following three graphs are the result of three different learning curves that trained via SGD with different learning rates, while the X axis is 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.

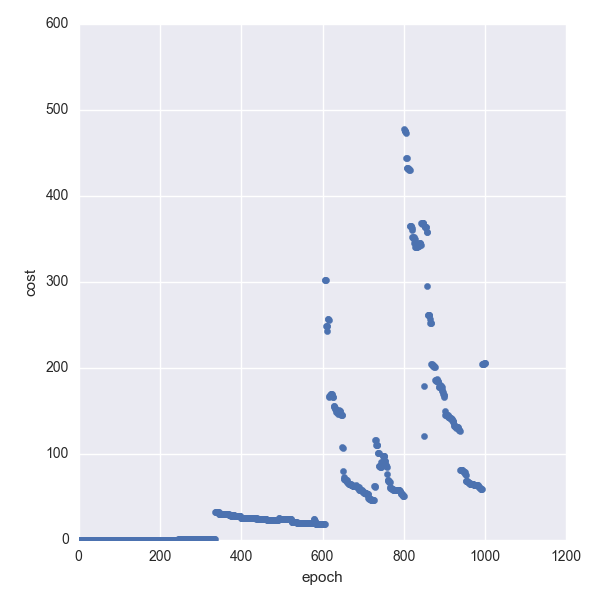


Figure.1 mean squared error vs iteration (SGD, learning rate = 0.01)

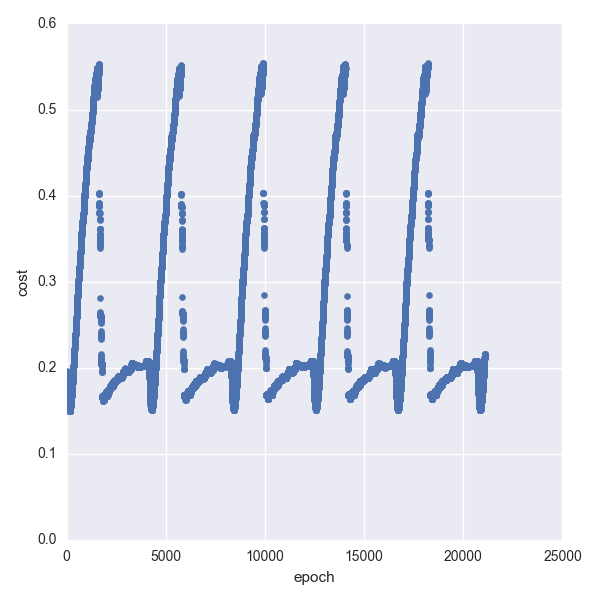


Figure.2 mean squared error vs iteration (SGD, learning rate = 0.001)

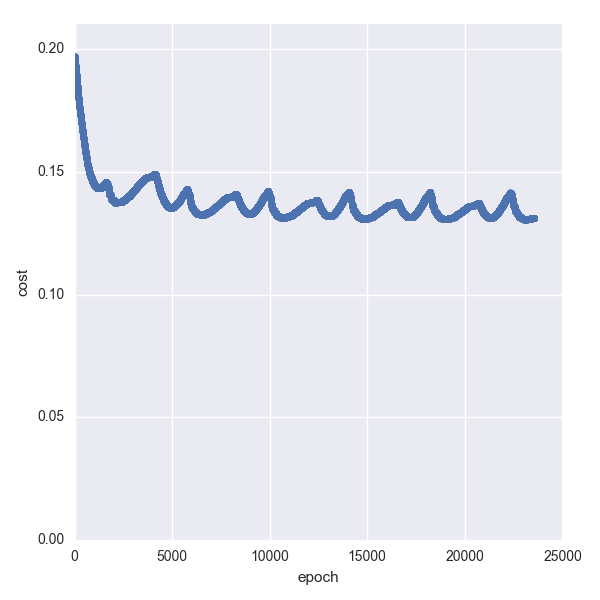


Figure.3 mean squared error vs iteration (SGD, learning rate = 0.0001)

|  |  |  |
| --- | --- | --- |
|  | 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 rate are different. And the best result come 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. Anyway, we choose 0.0001 as our parameter to draw the ROC curve.

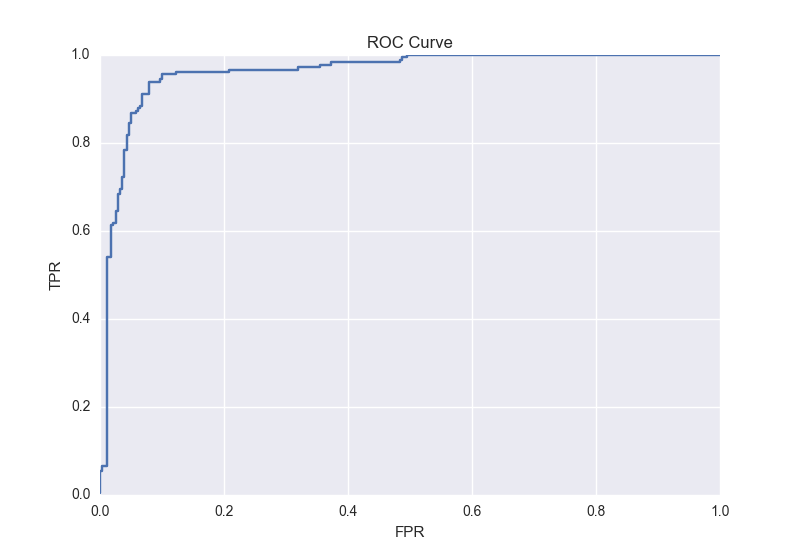


Figure.4 ROC curve of SGD (learning rate = 0.0001)

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)

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

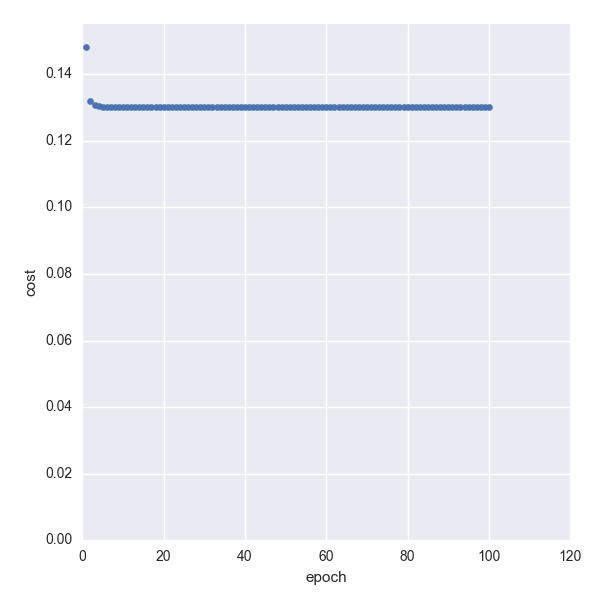


Figure.5 mean squared error vs iteration (BGD, learning rate = 1)

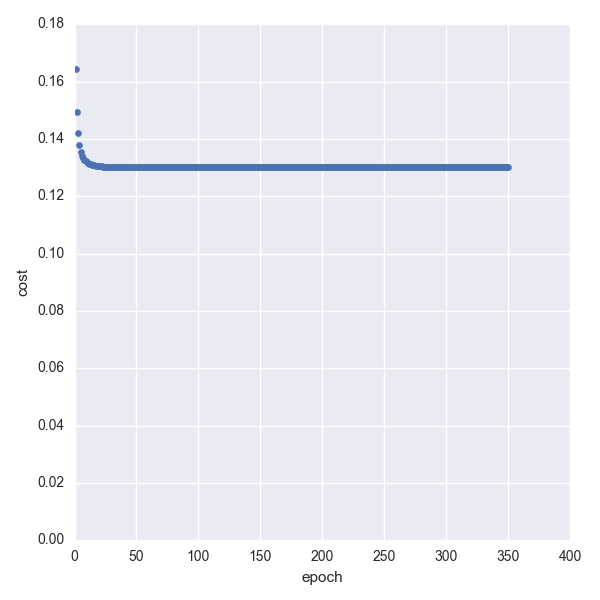


Figure.6 mean squared error vs iteration (BGD, learning rate = 0.1)

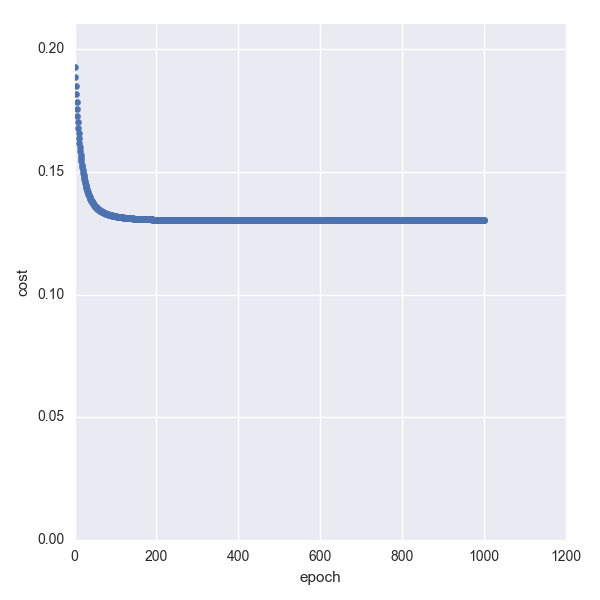


Figure.7 mean squared error vs iteration (BGD, learning rate = 0.01)

|  |  |  |
| --- | --- | --- |
|  | Learning rate | Min MSE |
| Figure5 | 1 | 0.130181147 |
| Figure6 | 0.1 | 0.130181345 |
| Figure7 | 0.01 | 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 = 1 to plot the ROC curve.

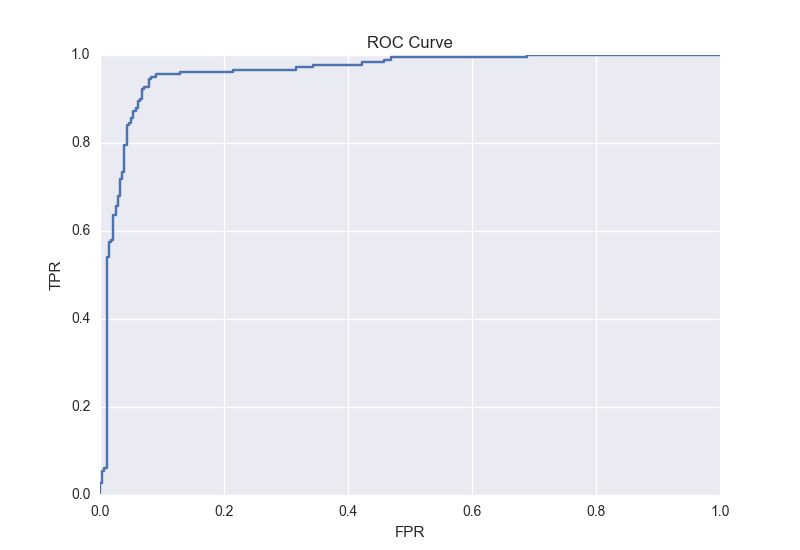


Figure.8 ROC curve of BGD (learning rate = 1)

The AUC of the 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.



Figure.9 The cost function in logistic regression

But in this stage, I found that the accuracy of float in Python may affect the result. While the input X of sigmoid function is quite large, the output value of 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 are 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.

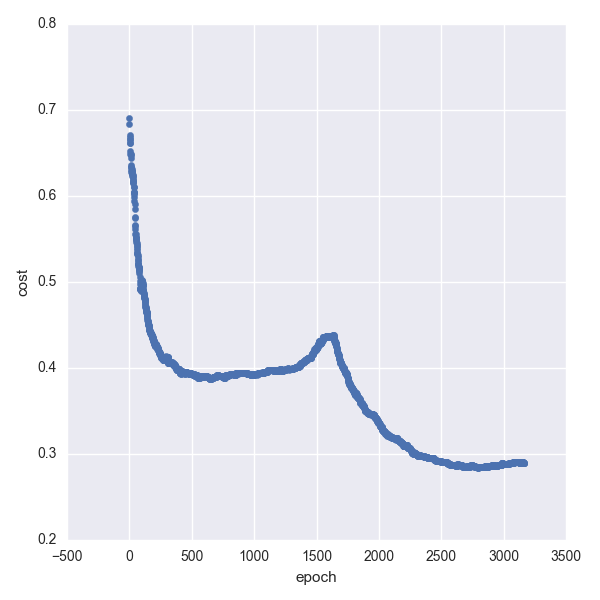


Figure.10 logistic, SGD, learning rate = 0.01

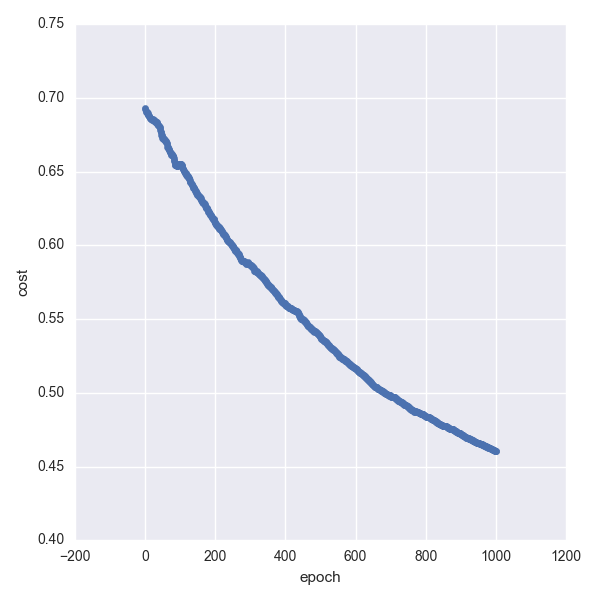


Figure.11 logistic, SGD, learning rate = 0.001

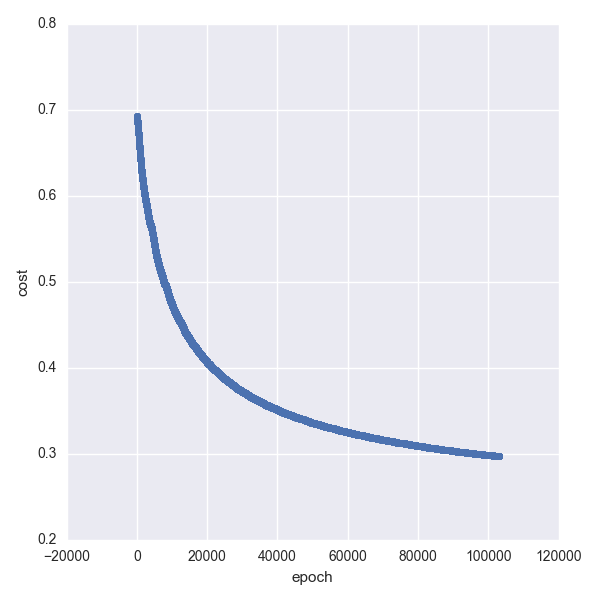


Figure.12 logistic, SGD, learning rate = 0.0001

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

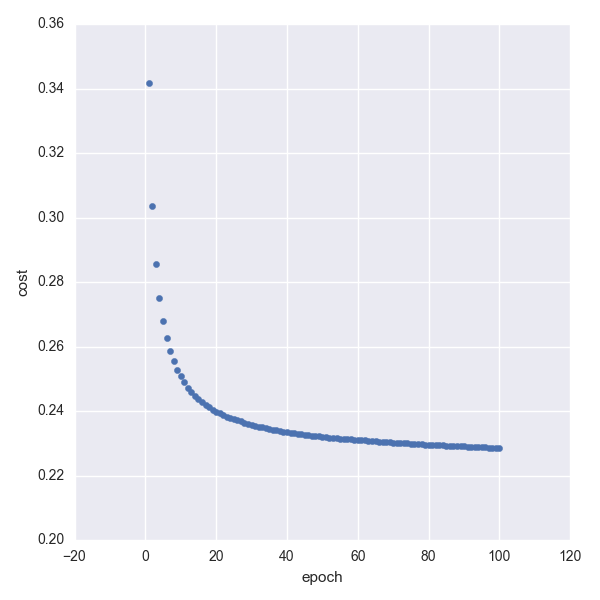


Figure.13 logistic, BGD, learning rate = 0.001

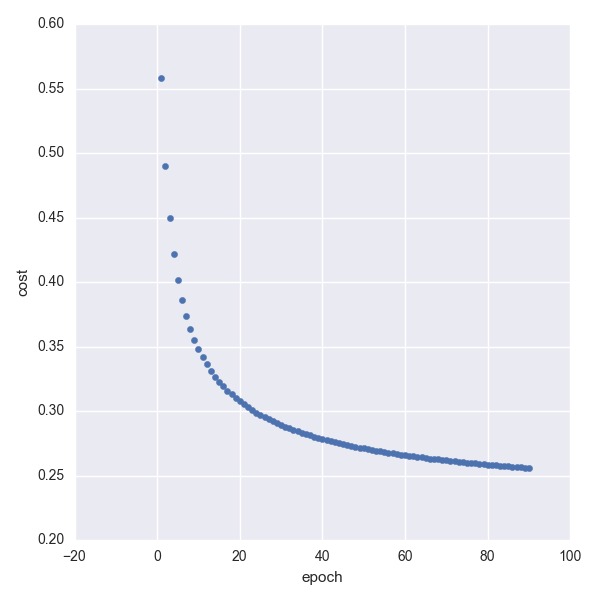


Figure.14 logistic, BGD, learning rate = 0.0001

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.

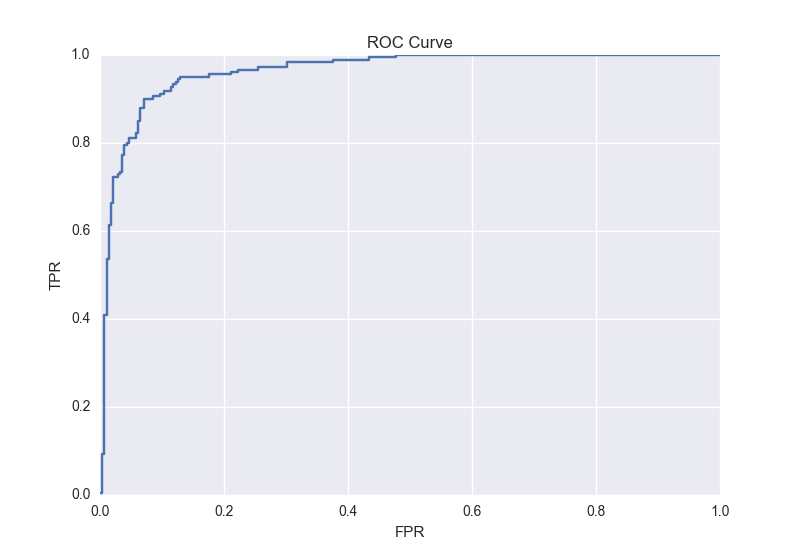


Figure.15 ROC curve of SGD (learning rate = 0.01, AUC=96.29%)

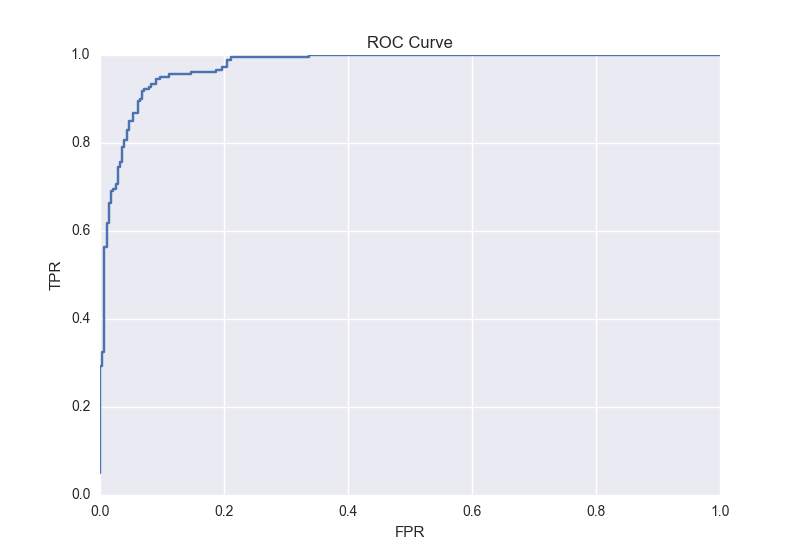


Figure.16 ROC curve of BGD (learning rate = 0.01, AUC=97.39%)

# Summery