Training Classifiers via Stochastic Gradient Descent

Qiu Hu
MG1633031
Master graduate, Nanjing University, PASA Lab
huqiu00@163.com

Abstract

This report states how I implements two algorithm: Logistic Regression and Ridge Regression(Classifier) which are tailored to large-scale classification. Given the datasets with a large number of examples, we will train classifiers via stochastic gradient descent(SGD) in order to make training phase more efficient. [1]

1 Introduction to Datasets

There are two datasets which come from LIBSVM[3], Dataset 1 (Training) contains 32561 rows and 124 columns, and dataset 1 (Testing) contains 16281 rows and 124 columns; Dataset 2 (Training) contains 290507 rows and 54 columns, and dataset 2 (Testing) contains 290505 rows and 54 columns. Each row represents an example. The last column represents the label of the corresponding example, and the remaining columns represent the features of the corresponding example. For each dataset, label $\mathcal{Y} = \{-1, +1\}$.
After carefully observing the datasets, I find Dataset 1 contains only 0 or 1, but Dataset 2 contains both float numbers between 0 and 1 and integer 0/1, indicating that Dataset 2 may not be scaled, so I scaled then to [0, 1], using Min-Max scaling, fortunately Dataset 1 was immune to the scaling because of above description.

2 Logistic Regression via SGD with L1-Regularization

Logistic Regression is one of most popular classification algorithm in our daily life and industrial application. In this section, I will show you a profile of Logistic Regression at first, and then give the pseudo code of my implementation, then I will show results in two datasets after run my code, and compare it to a python machine learning library Scikit-Learn's results, and at last show some figures of results for you understanding the whole process.

[1] Because suffering for confusing typography of Microsoft Word Software, I tried another typography system – LaTeX to report my experiment this time, by the way, I tried using English to organize my expression. Wish you can accept it and forgive my poor English.

2.1 What is Logistic Regression?

In linear regression, we note the model as

$$y = w^T x + b \tag{1}$$

but can we let predictions approach some thing derive from y ? Answer is absolutely yes, for example we can use log of y as objective that linear models should approach to, which is called log-linear regression.

$$\log y = w^T x + b \tag{2}$$

Generally, consider a differentiable monotone function $g(\cdot)$, let

$$y = g^{-1}(w^T x + b) \tag{3}$$

models like above form is called generalized linear model. Logistic regression can be seen as a special case of the generalized linear model and thus analogous to linear regression. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features),[4] it's $g^{-1}(\cdot)$ is

$$y = \frac{1}{1 + e^{-w^T x + b}} \tag{4}$$

above equation is actually using prediction results of linear regression model to approach logit probability of true label, thus, the corresponding model is called "Logistic Regression", although it's name contains "Regression", it is truly a classifier. Logistic regression have many advantages in classification task, including that it directly modeling on the real data, unnecessary to make hypothesis of data distribution, which avoided problems with inaccurate hypothesis of data distribution, moreover, it not only predict "categories", but approximation of categories' probability, last but not least, logistic function is a convex function which have many merits. We can do some conversion Eq.4 that:

$$\ln \frac{y}{1 - y} = w^T x + b \tag{5}$$

and we can see y as posterior probability estimates $p(y = 1|x)$, so Eq.5 can be rewrite as

$$\ln \frac{p(y = 1|x)}{p(y = -1|x)} = w^T x + b \tag{6}$$

absolutely we can infer following two equation:

$$p(y = 1|x) = \frac{e^{w^T x + b}}{1 + e^{w^T x + b}} = \frac{1}{1 + e^{-(w^T x + b)}} \tag{7}$$

$$p(y = 1|x) = \frac{1}{1 + e^{w^T x + b}} \tag{8}$$

thus we can use maximum likelihood method to estimate w and b, and for convenience, we note $\beta = (b; w)$ and $x = (1; x)$, and we add L1-regularization to our loss function, so our loss function can be written as

$$loss(\beta, x_i) = \frac{1}{N} \sum_{i=1}^{N} \log \left(1 + e^{-y_i \beta^T x_i}\right) + \lambda \|\beta\|_1 \tag{9}$$

after add L1-regularization, we note the final goal is to

$$min_\beta \frac{1}{N}\sum_{i=1}^{N}\log\left(1+e^{-y_i\beta^T x_i}\right)+\lambda\|\beta\|_1 \tag{10}$$

Stochastic Gradient Descent method usually used to solve the above optimization problem, which from Gradient Descent, because of GD's low efficiency in large scale dataset, people want to speed up the rate of convergence, at the same time without losing accuracy, so, they use a randomly selected every time to update parameters, that is to say, the batch size(the number of training examples) used for approximation of Eq.10 is 1.
Using SGD to solve Eq.10, we first calculate the gradient of Eq.10 for a sample x_i:

$$\frac{\partial loss(\beta, x_i)}{\partial \beta} = \frac{-y_i x_i}{1+e^{y_i\beta^T x_i}} + \lambda sign(\beta) \tag{11}$$

so we can update weights β with every iteration and every data sample using following equation:

$$\beta_{k+1} = \beta_k - \gamma\frac{\partial loss(\beta, x_i)}{\partial \beta} = \beta_k + \gamma\frac{y_i x_i}{1+e^{y_i\beta^T x_i}} - \gamma\lambda sign(\beta) \tag{12}$$

where $sign(x) = 1$ if x > 0, $sign(x) = -1$, and $sign(x) = 0$ if x $= 0$. This update equation was called "SGD-L1(Naive)"[2], and this naive method will cause two problems, one is that at each update, we need to perform the application of L1 penalty to all features, including the features that are not used in current training sample, and it does not produce a compact model, that is, when a dimension of β is very small and we can neglect it's impact, but we will update that dimension as 1 or -1 times regularization lambda and learning rate, regardless of it's size, at next update, this will cause appearance of many non-zero value in β, but we want that with the same effect, the less features the better.

定理 **2.1** *Occam's Razor: Entities should not be multiplied unnecessarily.*

For this reason, I tried some methods in Yoshimasa's paper, including Clipping, which clip the weight that crosses zero, and the update function can refer his paper, and finally I used L1 regularization with cumulative penalty to apply L1 penalty to object function, any details can be find in his paper.[2]

2.2 Implementation

After above analysis and derivation, we can easily write the algorithm using SGD to solve logistic regression with L1 regularization,(wh is a temporary variable), see Algorithm 1. In order to evaluate effectiveness of my algorithm implementation, I written a implementation using Scikit-Learn, a python machine learning library, and you can see the comparison in Table.2. After conduct a simple Cross Validation and adopted some tricks[1] on dataset, I set hyperparameters as following values:

2.3 Results of Experiment

After the programming, I evaluated the effectiveness and accuracy of my training algorithm on all 2 datasets, Figure.1 and Figure.2 show the change of key dependent variable including Train set error rate, Test error rate, Objective function and Sparsity in training process. Figure.1 shows results of logistic regression with L1-regularization on dataset 1 that with increasing

Table 1: Logistic Regression Hyperparameters

Hyperparameters	Values
Learning Rate γ	0.001
Regularization Lambda λ	0.0001
Max Iteration Times on Dataset	200(Dataset1), 30(Dataset2)

Algorithm 1 Logistic Regression with L1 regularization

procedure StochasticGradientDescent(D, $Labels$, $Iter$)
Input: $Dataset\ D$, $Labels\ of\ Dataset$, $Iteration\ num$
Output: optimal weight of logistic regression
$w \leftarrow [1, 1, ..., 1]$
Initialize q_i with zero for all i
for $k = 1 \rightarrow Iter$ **do**
  $chooseData = D$
  for $i = 1 \rightarrow m$ **do**
    $\gamma \leftarrow Learning\ Rate$
    $\lambda \leftarrow Regularization\ Lambda$
    $u = u + \gamma\lambda$
    Select a index of chooseData idx randomly
    $x \leftarrow chooseData[idx]$
    $del\ chooseData[idx]$
    for $i \in featuresinsamplex$ **do**
      $w_i = w_i - \gamma\frac{\partial loss(w,x_i)}{\partial w}$
      $wh \leftarrow w$
      if $w_i > 0$ **then**
        $w_i \leftarrow \max(0, w_i - (u + q_i))$
      else if $w_i < 0$ **then**
        $w_i \leftarrow \min(0, w_i + (u - q_i))$
      end if
      $q_i \leftarrow q_i + (w_i - wh)$
    end for
  end for
end for

value of iteration times of training, the training error rate and test set error rate decrease, like one line, after nearly 500000 times of iteration, the training error and test set error keep stable and smooth, only have some small fluctuations, indicating the loss function converged. Objective function has the same trend. See the red line indicating that with the increasing of training iteration, sparsity of weights vector increasing continually, final results after more than 6000000+ iterations, weights vector have sparsity of 31 dimension of 124 dimension, holding a quarter, that is to say, we just need use 3/4 of origin features to predict class labels of data sample, without loss of accuracy of prediction, see, we did a feature selection unconsciously.

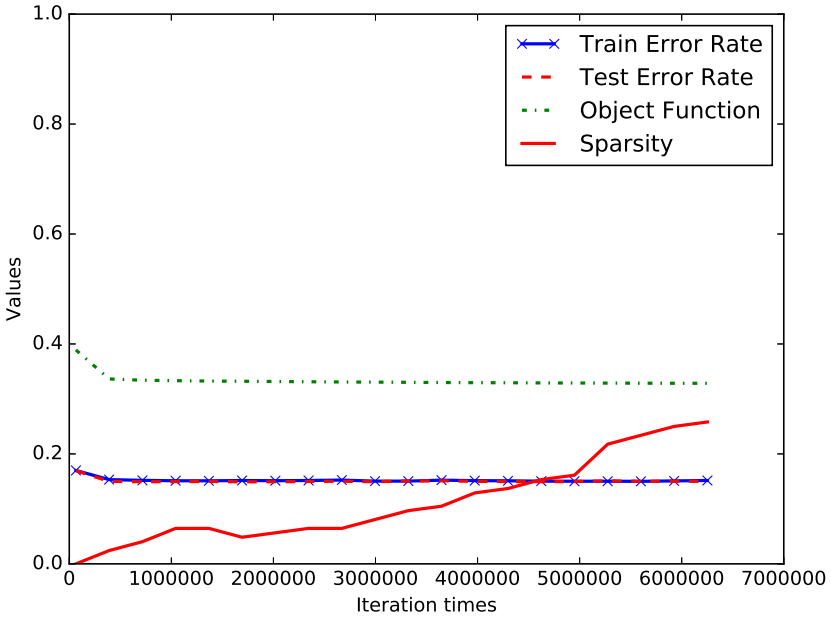

Figure 1: Logistic Regression with L1 regularization on Dataset 1

Figure.2 shows results of conducting algorithm on dataset 2, and we can see almost the same trend on Figure.1, training error rate, test set error rate and objective function decrease with increasing of iteration times, and sparsity increase. What different is that test set error rate on data set 2 is about 0.45, more than stable training error rate, 0.1879, and following form is the final results of logistic regression with L1 regularization on two Dataset.

Table 2: Logistic Regression with L1 regularization results

	Training error rate	Test error rate	Sparsity	Scikit-Learn Test error rate
Dataset 1	0.1504	0.1501	31/124	0.1500
Dataset 2	0.1879	0.4475	9/55	0.4100

2.4 Comparison of Naive and Cumulative Penalty L1 regularization in LR

As I stated, there are many approach to apply L1 penalty to our logistic regression's loss function, one of that is so-called "Naive" method[2], which directly use absolute value of $sign(\beta)$ as it's sub-gradient for weight update, but I used the cumulative penalty method, and following

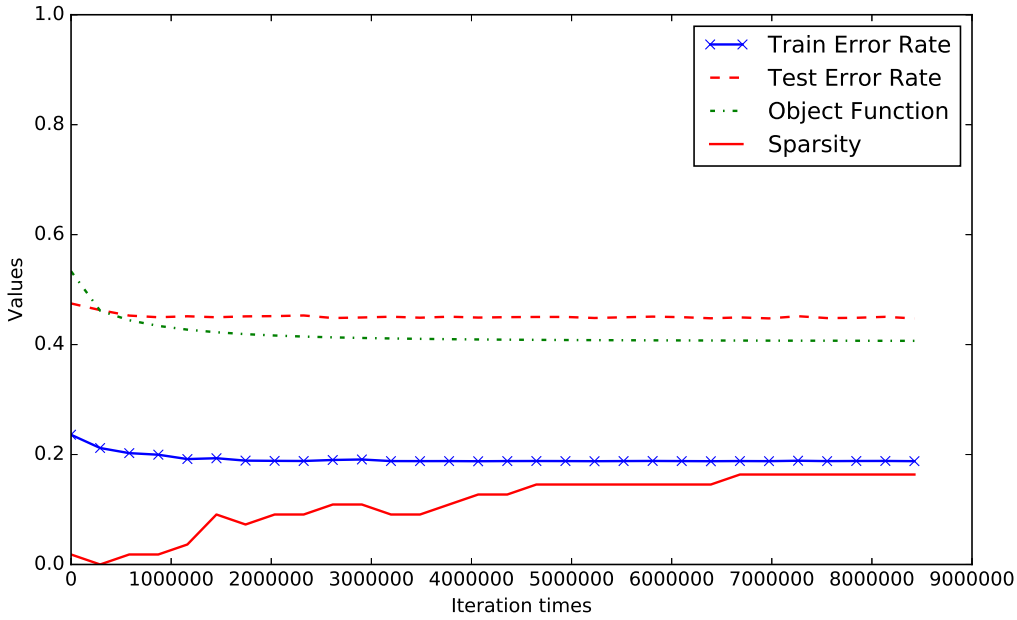

Figure 2: Logistic Regression with L1 regularization on Dataset 2

is the comparison of this two method. From Table.3 and Table.4, we can see that two methods have nearly the same performance, but cumulative penalty method has large sparsity and Naive method has 0 sparsity. See figures of the performance of Naive method in Appendix.

Table 3: Comparison of Naive and Cumulative method in Dataset 1

	Training error rate	Test error rate	Sparsity	Scikit-Learn Test error rate
Naive	0.1507	0.1493	0/124	0.1500
Cumulative	0.1504	0.1501	31/124	0.1500

Table 4: Comparison of Naive and Cumulative method in Dataset2

	Training error rate	Test error rate	Sparsity	Scikit-Learn Test error rate
Naive	0.1878	0.4467	0/55	0.4100
Cumulative	0.1879	0.4475	9/55	0.4100

3 Ridge Regression via SGD with L2-regularization

Ridge Regression is a variant of Linear Regression, as we all know, there are many different methods to fit the linear model to a set of training data, but by far the most popular is the method of *least squares*, in this approach, we pick the coefficients w to minimize the residual sum of squares[5]

$$RSS(w) = \sum_{i=1}^{N} (y_i - w^T x_i)^2 \tag{13}$$

because $RSS(w)$ is a quadratic function of the parameters, its' minimum always exists, we can use many numerical optimization algorithm to optimize the function, like SGD. But what's

ridge regression, actually it's very simple, ridge regression shrinks the regression coefficients by imposing a penalty on their size, the ridge coefficients minimize a penalized residual sum of squares

$$min_w \frac{1}{N} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2 \tag{14}$$

and that, is our objective.
Using SGD to solve this optimization problem, we just need to get the gradient of objective in one training sample, which is

$$\nabla loss(w, x_i) = -2x_i(y_i - w^T x_i) + 2\lambda w \tag{15}$$

So, according Eq.15, we can easily write update function of weights by one iteration and one training example as:

$$w_{k+1} = w_k - \gamma \frac{\partial loss(w, x_i)}{\partial w} = w_k + 2\gamma x_i(y_i - w^T x_i) - 2\gamma\lambda w \tag{16}$$

3.1 Implementation

As we have the update function of ridge coefficient, we write it's pseudo-code easily in Algorithm 2. In order to evaluate effectiveness of my algorithm implementation, I also written a implementation using Scikit-Learn, a python machine learning library, and you can see the comparison in Table.6. And after conduct a simple Cross Validation and use some tricks[1]on dataset, I set hyperparameters as following values: (t is iteration times)

Table 5: Ridge Regression Hyperparameters

Hyperparameters	Values
Learning Rate γ	0.001 / (1.0 + 0.001*t*0.003)
Regularization Lambda λ	0.0001
Max Iteration Times on Dataset	200(Dataset1), 30(Dataset2)

3.2 Results of Experiment

After the programming, I evaluated the effectiveness and accuracy of my training algorithm on all 2 datasets, Figure.3 and Figure.4 show the change of key dependent variable including Train set error rate, Test error rate, Objective function and Sparsity in training process. Figure.3 shows results of ridge regression with L2-regularization on dataset 1 that with increasing value of iteration times of training, the training error rate and test set error rate decrease, like one line, after nearly 350000 times of iteration, the training error and test set error keep stable and smooth, only have some small fluctuations, indicating the loss function converged. Objective function has the same trend. Because of algorithm applying L2 regularization on loss function, so there will be no sparsity, because of the property of L2 regularization. From here we can know that, L1 regularization will cause more sparsity of weights vector than L2 regularization, which is the main reason of L1 regularization has been adopted/used in many scenarios, no matter in industry or academia.

Figure.4 shows results of conducting algorithm on dataset 2, and we can see almost the same trend on Figure.1, training error rate, test set error rate and objective function decrease

Algorithm 2 Ridge Regression with L2 regularization

procedure StochasticGradientDescent(D, *Labels*, *Iter*)
Input: *Dataset D*, *Labels of Dataset*, *Iteration num*
Output: optimal weight of ridge regression
    $w \leftarrow [1, 1, ..., 1]$
    for $k = 1 \rightarrow Iter$ **do**
        $chooseData = D$
        for $i = 1 \rightarrow m$ **do**
            $\gamma \leftarrow$ *Learning Rate*
            $\lambda \leftarrow$ *Regularization Lambda*
            Select a index of chooseData *idx* randomly
            $x \leftarrow chooseData[idx]$
            del $chooseData[idx]$
            $w_i = w_i - \gamma \frac{\partial loss(w, x_i)}{\partial w}$
        end for
    end for

with increasing of iteration times, and sparsity increase. What different is that test set error rate on data set 2 is about 0.45, more than stable training error rate, 0.1921, and following form is the final results of ridge regression with L2 regularization on two Dataset.

References

[1] Léon Bottou. "Stochastic Gradient Tricks". In: *Neural Networks, Tricks of the Trade, Reloaded*. Vol. 7700. Springer, 2012, 430–445. URL: https://www.microsoft.com/en-us/research/publication/stochastic-gradient-tricks/.

[2] Yoshimasa Tsurouka. et.al. "Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty". In: *ACL and AFNLP* (2009).

[3] *LIBSVM Data: Classification (Binary Class)*. URL: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[4] *Logistic regression*. URL: https://en.wikipedia.org/wiki/Logistic_regression.

[5] Robert Tibshirani Trevor Hastie and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2008.

A Results Appendix

Table 6: Ridge Regression with L2 regularization results

	Training error rate	Test error rate	Scikit-Learn Test error rate
Dataset 1	0.1547	0.1548	0.1500
Dataset 2	0.1921	0.4449	0.4100

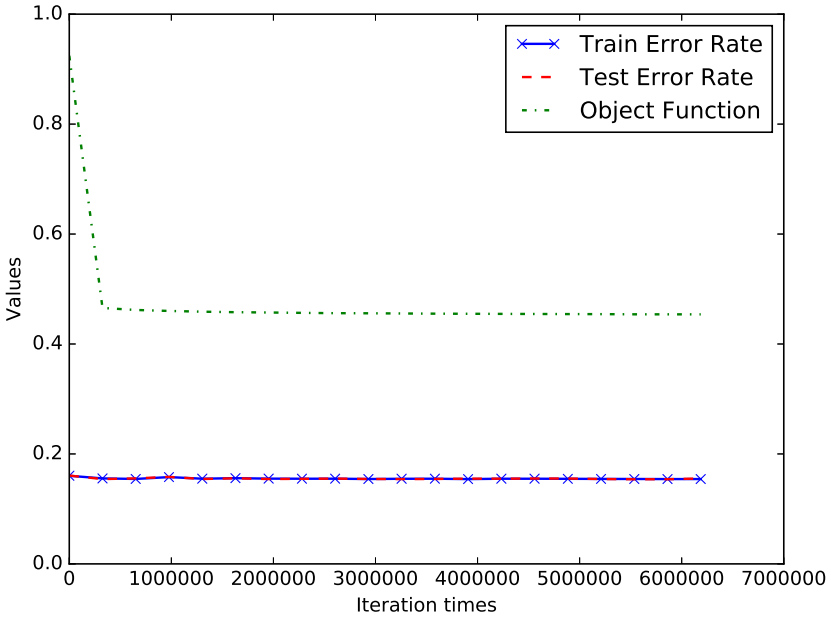

Figure 3: Ridge Regression with L2 regularization on Dataset 1

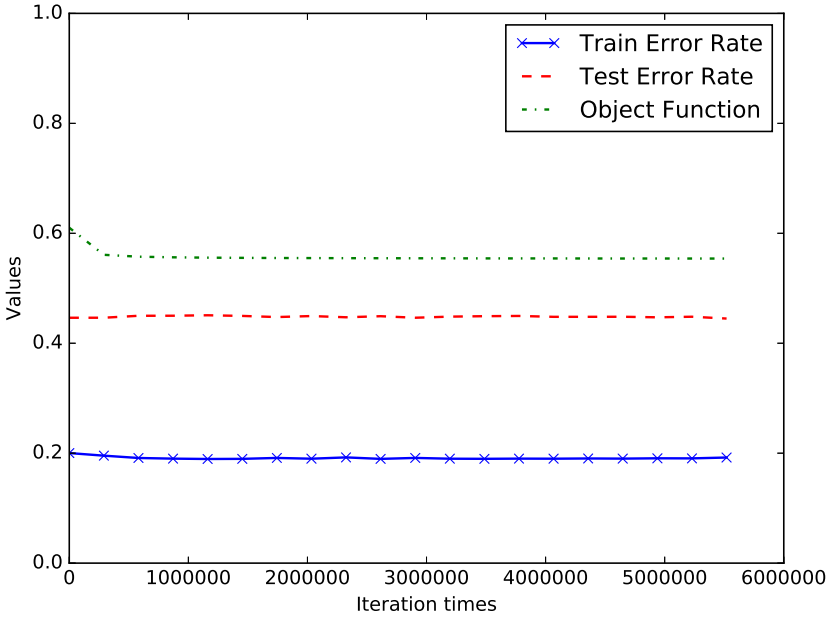

Figure 4: Ridge Regression with L2 regularization on Dataset 2

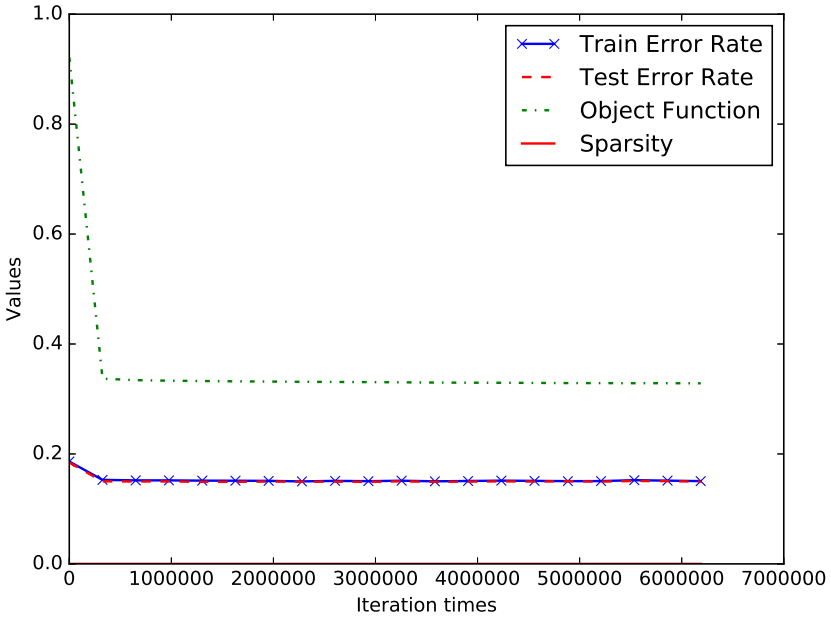

Figure 5: Logistic Regression with Naive L1 regularization on Dataset 1

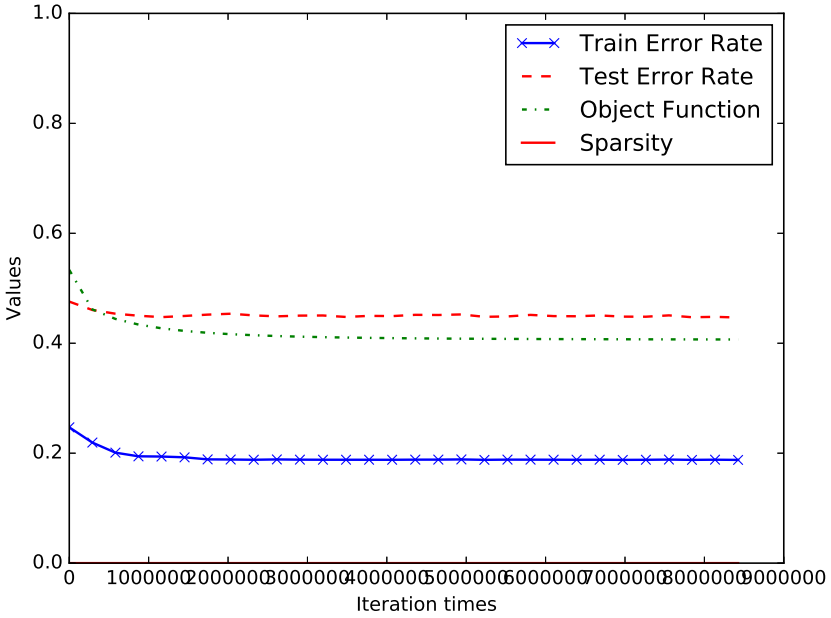

Figure 6: Logistic Regression with Naive L1 regularization on Dataset 2