MoSIG, Data Science

Convex and Distributed Optimization
Lab Report

Laleh ROSTAMI HOSOORI & Sreynoch SOUNG
December 28, 2016

I. Introduction

In this lab, we picked a supervised binary classification problem as an optimization problem, since in the binary classification we need to minimize the error function of the linear model. We worked specifically on Logistic Regression and tried to minimize Logistic Loss with gradient and proximal algorithms.

II. Read and Preprocess data

Using Spark MLlib, we read the whole data from the given file and converted it to an RDD type. We noticed that the given dataset was sparse, so we used LabeledPoint with SparseVector to store the given examples. There were 351 examples, 35 features and the density of the dataset was approximately 0.911.

For preprocessing the data, we need to take account of the fact that the classes {-1, +1} might be imbalanced. Therefore, we added an intercept with the value 1 to the feature vector of each example. Then to have a faster convergence, it is proper to normalize the dataset. Since our dataset was sparse, we normalized the dataset to have feature vectors with unit ℓ2 norm. Eventually, the density of normalized dataset slightly decreased to 0.887.

We randomly divided the dataset into two sets, 95% for *learning Set* and 5% for *testing Set*.

III. Gradient Algorithm

First, we implemented two functions one for calculating the Logistic Loss per example and another for computing the gradient of the Logistic Loss for each example. Second, we found the stepsize for the gradient algorithm, which was equal to the division of one by the Lipschitz constant of the gradient function. Then, we implemented a gradient algorithm and plotted the average of logistic loss values per iteration. The Figure 1 shows the results for the learning and testing sets.

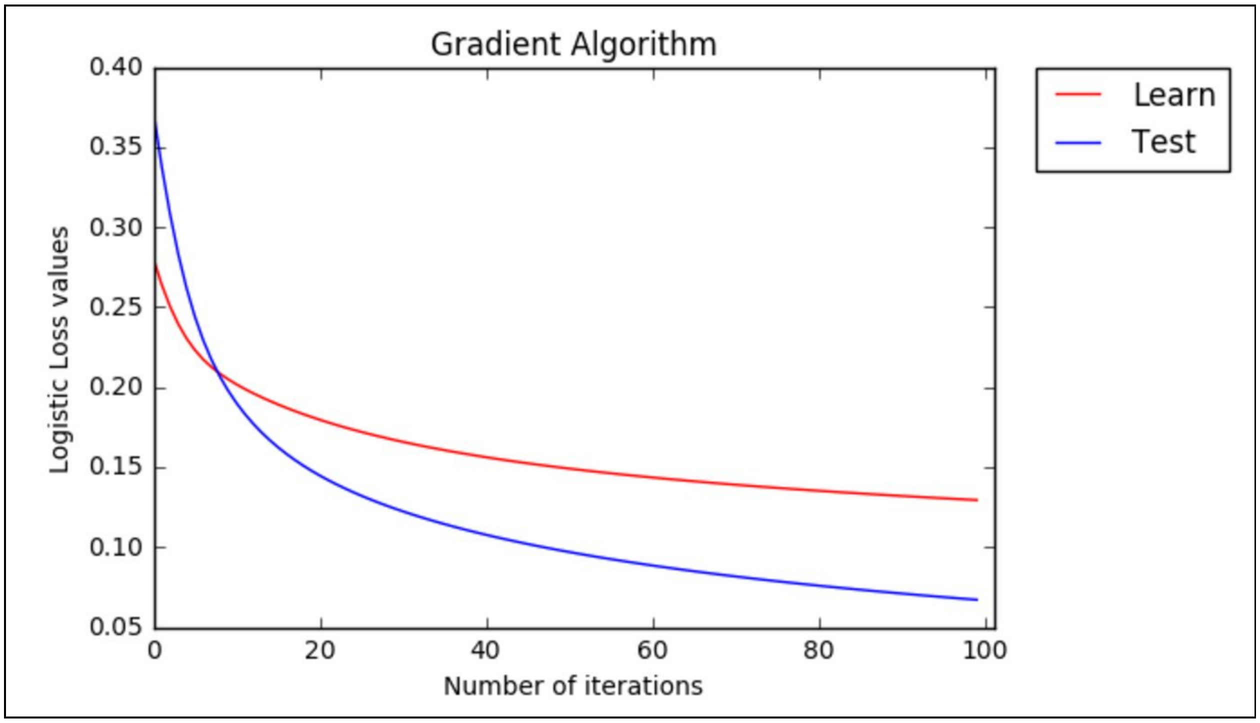

Figure 1. The results of Gradient algorithm

As illustrated in the chart, the value of Logistic loss was decreasing for both learning and testing sets. It seems the linear model works better for the testing set since the Logistic loss value is closer to zero after 100 iterations; however, it shows a proper result for learning set.

We observed that the gradient algorithm takes kind of a long time to finish, although it gives a good result. It might be because of overfitting the model. We used *%timeit* command to log the execution time of the algorithm. It took 9.88 s per loop for the *Learning* set and 14.8 s per loop for the *Testing* set.

IV. Proximal Algorithm

In order to prevent overfitting, we add a regularization term to the Logistic Loss function. This regularizer has a smooth and non-smooth parts. Therefore, we implemented a new gradient function for the smooth part, which calculates the gradient of the Logistic Loss per example. We also implemented another function for the Proximal operator of the non-smooth part. We adapted the stepsize to consider the coefficient of the smooth part (λ_2). Eventually, we wrote the proximal algorithm. Figure 2 plots the result of the running the algorithm on the learning and testing sets with zero values for λ_1 and λ_2.

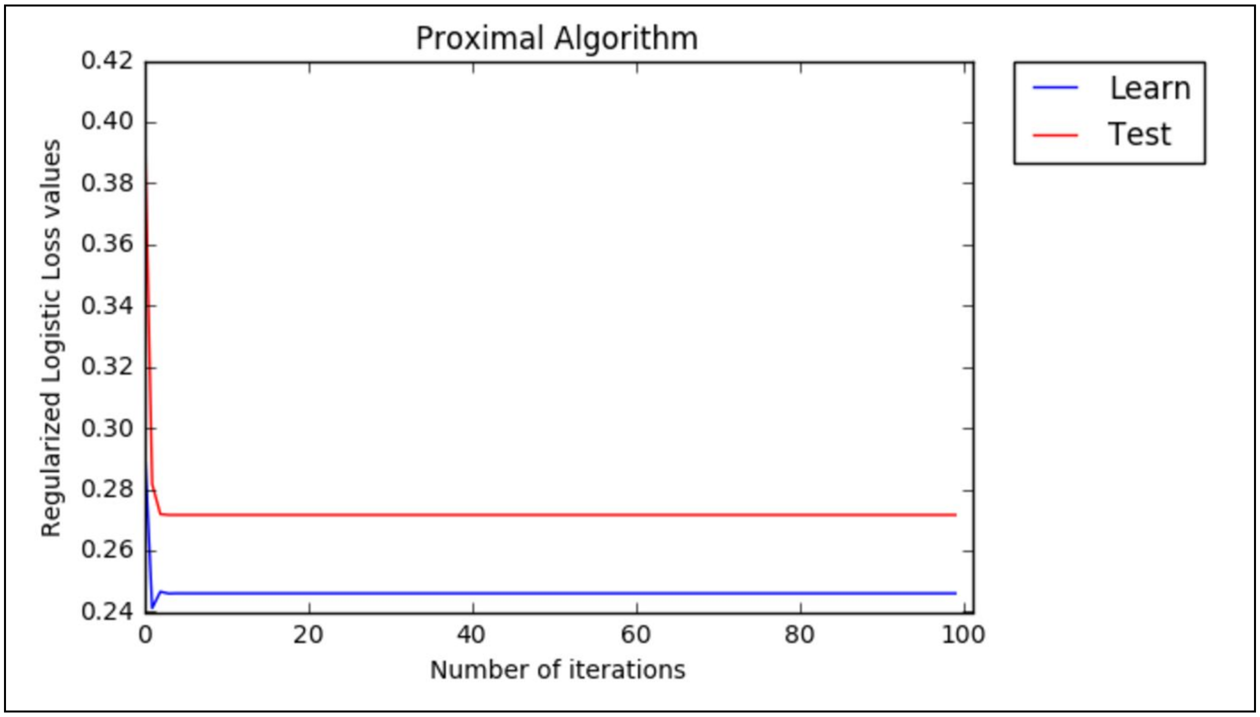

Figure 2. The results of Proximal algorithm for *learning* and *testing* sets

As you can see in the chart, the proximal algorithm performs slightly better for the *learning* set and the result stays the consistent after a few iterations. The execution times are 4.56 s and 7.12 s per loop for *learning* and *testing* sets respectively. Obviously, these times are almost half of the execution times captured for the gradient algorithm. So, it seems the proximal algorithm runs faster than the gradient algorithm.

Figure 3 compares the result of the proximal algorithm on the learning set for the different values of λ_1 and λ_2. According to the chart, it seems the change in the value of λ_1 is more effective than λ_2. The lower values of λ_1 and λ_2 shows better results. On the other hand, in comparison to the gradient algorithm, there are more errors but we can get the result after fewer iterations.

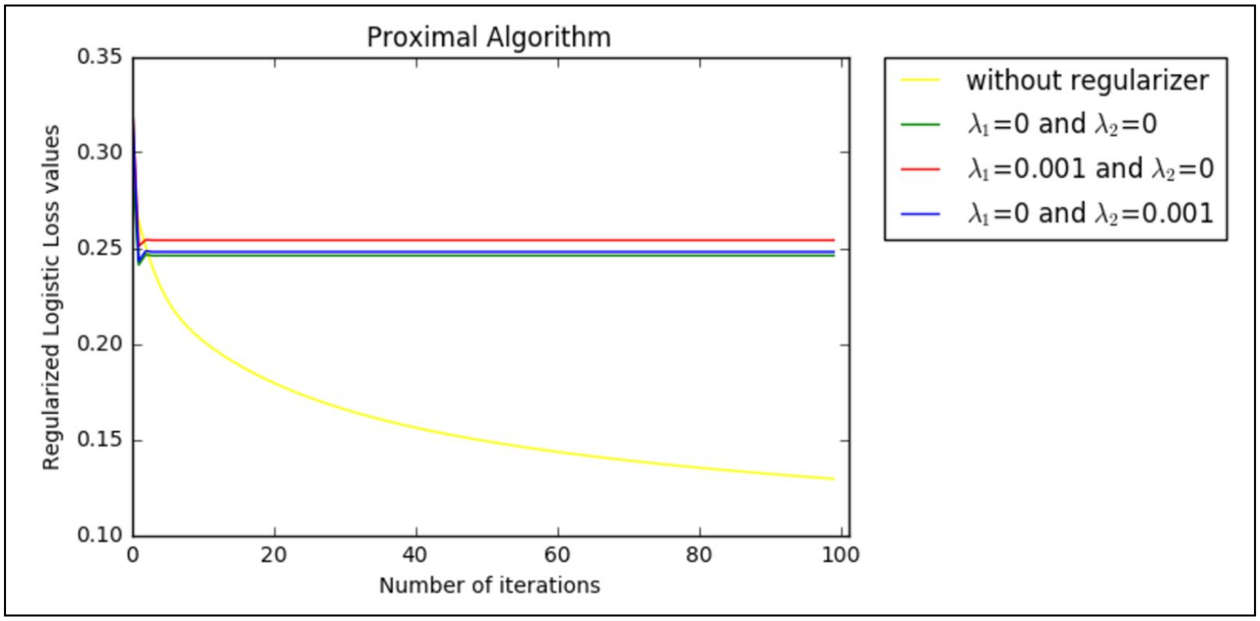

Figure 3. The results of Proximal algorithm for different values of λ_1 and λ_2 on *learning* set

V. Accuracy

To compute the accuracy of the proximal algorithm, we calculated two following metrics:

$$Accuracy = \frac{\#\ of\ True\ Positive\ Labels + \#\ of\ True\ Negative\ Labels}{\#\ of\ All\ the\ Labels}$$

$$F1 = \frac{2 * \#\ of\ True\ Positive\ Labels}{(2 * \#\ of\ True\ Positive\ Labels) + \#\ of\ False\ Positive\ Labels + \#\ of\ False\ Negative\ Labels}$$

The following table shows the results with the different values of λ_1 and λ_2.

	Accuracy	F1
$\lambda_1 = \lambda_2 = 0$	0.711	0.815
$\lambda_1 = 0.001$, $\lambda_2 = 0$	0.708	0.814
$\lambda_1 = 0$, $\lambda_2 = 0.001$	0.711	0.815

We got the similar values for accuracy and F1 for $\lambda_1 = \lambda_2 = 0$ and $\lambda_1 = 0$, $\lambda_2 = 0.001$ but the accuracy decreases when we increase λ_1. We can conclude that the proximal algorithm is more accurate with lower λ_1 and λ_2; however, the increase in the value of λ_1 has more negative side effect in the accuracy of results.