華南理工大學

South China University of Technology

The Experiment Report of *Machine Learning*

School: School of Software Engineering

Subject: Software Engineering

Author:
Lizhao Liu

Supervisor:
Mingkui Tan

Student ID:
201730683109

Grade:
Undergraduate

November 11, 2019

Logistic Regression and SVM

***Abstract*—In this report, we solve the binary classification problem in a9a dataset by Logistic Regression (LR) and Support Vector Machine (SVM).**
We perform experiments on four aspects:
1. Loss function comparision in SVM.
2. Hyper-parameter tuning in SVM.
3. Initilizer comparision in both LR and SVM.
4. Optimizer comparision in both LR and SVM.

I. Introduction

LOGISTIC Regression(LR) is an extension of linear regression, which use σ function as activation to map the output of linear regression into a interval (0, 1). So that LR can output the probability in binary classification. SVM is also an extension of linear regression. Based on the idea ”large margin classification”, SVM can perform very well in many classification task. Even now, LR and SVM are widely used in academic and industry community. These two algorithms are classic, so we fully explore them by solving a9a binary classification task. Beyond that, we will explore the initilizer and optimizer selection in both LR and SVM.

II. Methods and Theory

In this part, we first define logistic regression and SVM. Then we define LR’s loss function Binary Cross Entrophy (BCE) and SVM’s loss function Hinge Loss (HL). Mean Squared Error (MES) loss function. Then we solve the closed form solution for LR and its variant RR. At last we define GD algorithm, and its variants SGD and MBGD.

A. Logistic Regression

Given dataset $D = \{x_i, y_i\}_{i=1}^{m}$, where $x_i \in \mathbf{R}^n$ and $y_i \in 0, 1$. Mathematically, for a single training datapoint (x, y), LR assumes:

$$P(Y = 1|X = x) = \sigma(z) \tag{1}$$

$$P(Y = 0|X = x) = 1 - \sigma(z) \tag{2}$$

$$z = F(x; W) \tag{3}$$

where $F(x; W)$ is linear regression parameterized by W and $\sigma(.)$ is sigmoid function, where $\sigma(z) = \frac{1}{1+e^{-z}} \in (0, 1)$.

B. Binary Cross Entrophy

Logistic regression assume each datapoint is independent, so the likelihood of all data is:

$$\begin{aligned} L(W) &= \prod_{i=1}^{m} P(Y = y_i|X = x_i) \\ &= \prod_{i=1}^{m} \sigma(z)^{y_i}(1 - \sigma(z)^{1-y_i}) \end{aligned} \tag{4}$$

So our goal is to maximize the likelihood of data, in practice we maximize the log likelihood.

$$\begin{aligned} LL(W) &= log \prod_{i=1}^{m} \sigma(z)^{y_i}(1 - \sigma(z)^{1-y_i}) \\ &= \Sigma_{i=1}^{m} y_i \sigma(z) + (1 - y_i)(1 - \sigma(z)) \end{aligned} \tag{5}$$

C. Ridge Regression

In Equation **??**, when $n > m$, $X^T X$ is not invertible. Ridge Regression tackle this problem by adding a small term. So the CFS for RR is

$$W_{ridge}^{*} = (X^T X + \lambda \mathbf{I})^{-1} X^T y \tag{6}$$

Where $\mathbf{I}$ is identity matrix.
In this way, the loss function for Ridge Regression is

$$L_{ridge}(\hat{y}, y) = \frac{1}{2m}(\hat{y} - y)^T(\hat{y} - y) + \frac{1}{2} W^T W \tag{7}$$

In another point of view, RR can be seen as a regularized version of LR.

D. Gradient Descent

Not all problem can be derived a CFS, which is often the case in machine learning. So instead of getting the best parameters W^* immedeatly, we can update it incremently until W gets close enough to W^*. That is the basic idea of GD algorithm. GD algorithm can be found in Algorithm 1.

E. Stochastic Gradient Descent

Different from GD which updates after it computes the gradient of parameters over all examples, stochastic gradient descent updates immediately once it computes parameters’ gradient from only one example. SGD algorithm can be found in Algorithm 2. It can be seen in the algorithm that, there is another for loop inside the for loop of GD algorithm, which leads to time un-efficiency.

```
Algorithm 1: Gradient Descent
  Input: Training Dataset $D = \{x_i, y_i\}_{i=1}^m$, learning rate
         $\eta$, max iteration $N$, parameter $W$
  Output: Optimized parameter $\hat{W}$
  $t \leftarrow 1$
  for $t < N$ do
      $\hat{y} \leftarrow X^T W$
      $L(\hat{y}, y) \leftarrow \frac{1}{2m}(\hat{y} - y)^T(\hat{y} - y)$
      $\Delta W \leftarrow \frac{\partial L}{\partial W}$
      $W \leftarrow W - \eta * \Delta W$
  $\hat{W} \leftarrow W$
  return $\hat{W}$
```

```
Algorithm 2: Stochastic Gradient Descent
  Input: Training Dataset $D = \{x_i, y_i\}_{i=1}^m$, learning rate
         $\eta$, max iteration $N$, parameter $W$
  Output: Optimized parameter $\hat{W}$
  $t \leftarrow 1$
  for $t < N$ do
      $i \leftarrow 0$
      for $i < m$ do
          $\hat{y}_i \leftarrow X_i^T W$
          $L(\hat{y}_i, y_i) \leftarrow \frac{1}{2}(\hat{y}_i - y_i)^T(\hat{y}_i - y_i)$
          $\Delta W \leftarrow \frac{\partial L}{\partial W}$
          $W \leftarrow W - \eta * \Delta W$
  $\hat{W} \leftarrow W$
  return $\hat{W}$
```

F. Mini Batch Gradient Descent

Due to the fact that SGD updates the parameters based on only one examples, which it is time-comsuming, mini batch gradient descent updates the parameter based on a batch of examples, typically the batch size is bigger than 1 and smaller than the number of whole examples, e.g. 8, 16, 32... MBGD Algorithm can be found in Algorithm 3.

```
Algorithm 3: Mini Batch Gradient Descent
  Input: Training Dataset $D = \{x_i, y_i\}_{i=1}^m$, learning rate
         $\eta$, max iteration $N$, parameter $W$
  Output: Optimized parameter $\hat{W}$
  $t \leftarrow 1$
  for $t < N$ do
      $i \leftarrow 0$
      for $X_{batch}, y_{batch}$ in $D$ do
          $\hat{y}_{batch} \leftarrow X_{batch}^T W$
          $L(\hat{y}_{batch}, y_{batch}) \leftarrow$
              $\frac{1}{2}(\hat{y}_{batch} - y_{batch})^T(\hat{y}_{batch} - y_{batch})$
          $\Delta W \leftarrow \frac{\partial L}{\partial W}$
          $W \leftarrow W - \eta * \Delta W$
  $\hat{W} \leftarrow W$
  return $\hat{W}$
```

III. Experiments

A. Dataset

We conduct all the experiments on housing dataset, which has total 506 examples and each example's form is (x, y), where $x \in \mathbf{R}^{13}$ and $y \in \mathbf{R}$. We split the dataset into three part: 272 training examples, 67 validation examples and 167 test examples. X is already scaled between -1 and 1. And we normalized the $X_{train}, X_{val}, X_{test}$ by using the mean and variance of X_{train}.

B. Implementation

We implement the LR, RR and GD, GD, MBGD algorithm using python and mainly rely on the numpy package.

C. Linear Regression and Ridge Regression

In this section, we conduct the experiment of serveral magnitude of λ of RR in training set, which leads to serveral W_{ridge}. We use L_2norm to represent the magnitude of a vector. Given $v \in \mathbf{R}^n$

$$L_2norm(v) = \sqrt{\Sigma_{i=1}^{n} v_i} \tag{8}$$

From Table I, we can see that:

First, as the magnitude of lambda becomes bigger, both MSE_{train} and MSE_{val} become bigger.

Second, as the magnitude of lambda becomes bigger, the magnitude of W_{ridge} become smaller.

The first observation is not often the cases. But the second observation can be explained that as the regularizer λ become bigger, the W is regularized, so that it is smaller.

By comparing Table I and Table II, we can see that, the performance and the weight magnitude is almost the same as the λ is small, but the difference become larger as λ grows.

We also explore the weight mantitude between LR and RR under different λ magnitude setting. Specifically, we set log_{10}^{λ} in range from -7 to 3. We can see from Fig 1, as the magnitude of λ growing, the weight magnitude of ridge regression become much smaller than linear regression.

TABLE I
DIFFERENT LAMBDAS' IMPACT IN TERMS OF ERROR AND W MAGNITUDE. BEST RESULT ARE IN BOLD.

log_{10}^{λ}	-2	-1	0	1	2	3
MSE_{train}	**10.23**	**10.23**	10.24	10.63	30.37	181.35
MSE_{val}	**17.08**	17.09	17.16	18.20	43.92	214.55
$L_2norm(W_{ridge})$	0.83	0.82	0.81	0.72	0.50	0.38

TABLE II
LINEAR REGRESSION ERROR AND W MAGNITUDE.

MSE_{train}	10.23
MSE_{val}	17.08
$L_2norm(W)$	0.83

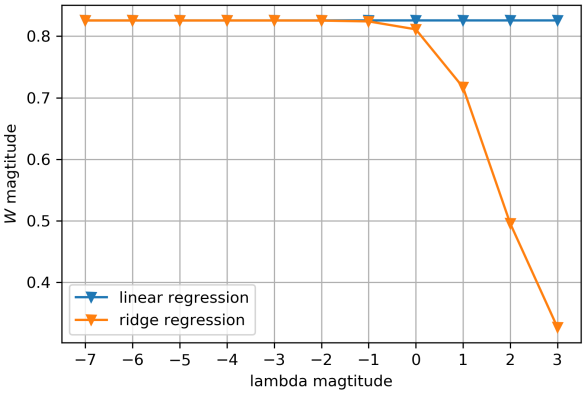

Fig. 1. The magnitude of W_{ridge} under different lambda mgnitude.

D. Gradient Descent, Stochastic Gradient Descent and Mini Batch Gradient Descent

In this section, we conduct experiments on optimizing the LR model by using GD, SGD, MBGD algorrithms, and compare them in terms of convergence and time cost.

We first evaluate the convergence of three algorithms, specificlly we set batch size in a range from 1 to number of full examples in training set. When the batch size is one, mini batch gradient descent becomes stochastic gradient descent and when the batch size equals to number of full examples, mini batch gradient descent becomes gradient descent. All experiments are with the same learning rate 0.01.

As we can see from Fig 2 and Fig 3, as the batch size become larger, the convergence of the linear regression model bocome slower. The reason is that, in each epoch, the smaller the batch size, the model have more update steps, so the convergence is faster. Especially, the stochastic gradient descent converges within 10 steps, while gradient descent requires nearly 400 steps, there is a 400x margin.

We then measure the time cost for each batch size.

As illustrated in Fig 4, the smaller the batch size, more time cost is required, because of the explicit for loop in the SGD and MBGD. Especially, the GD requires only about 1 seconds in one step, while SGD requires 60 seconds, there exist a 60x margin.

E. Impact of learning rate

In this section, we evaluate the impact of learning rate in MBGD, we set batch size to 64 and epochs to 200. We set learning rate in a range e.g. 1e-4, 1e-3, 1e-2, 1e-1, 1.

As presented in Fig 5 and Fig 6, small learning rate tends to make the convergence smoothly, but needs more epoches to fully converge. However, big learning rate make the convergence very fast, but will not guaratee that the convergence will be near the optimal(or local optimal) and makes the MSE shaking around after a specific epoch.

F. Weight magnitude between CSF and MBGD

In this section, we explore the weight magnitude($L_2 norm$) between CSF and MBGD optimized weight of linear regression.

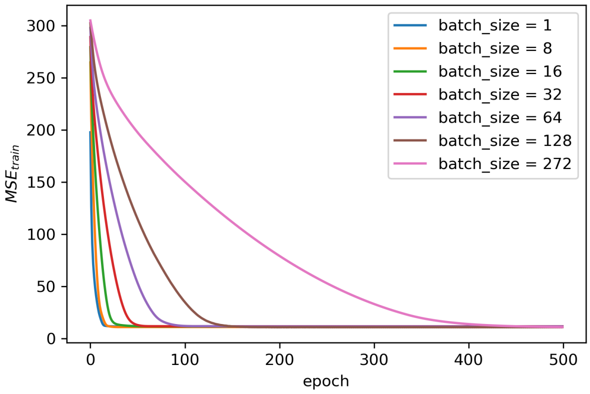

Fig. 2. The MSE of training set under different batch size.

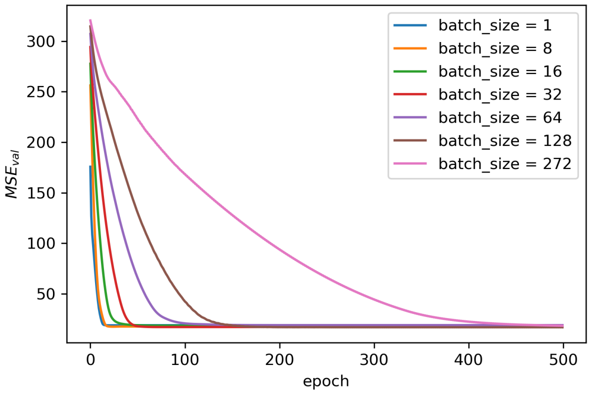

Fig. 3. The MSE of validation set under different batch size.

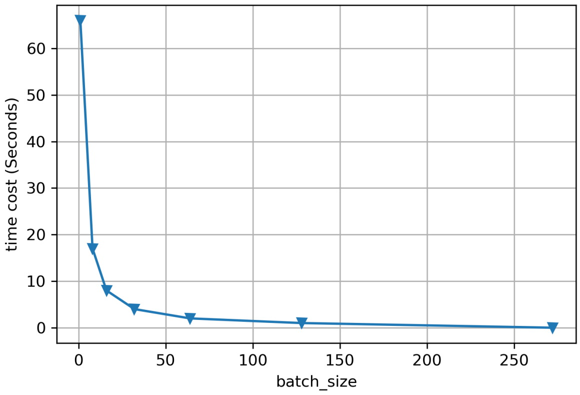

Fig. 4. The time cost of 100 epoches under different batch size.

Then, we experiments on comparing the weight magnitude of CFS and MBGD optimized LR. In MBGD, we set learning rate to 0.01 and batch size to 64. We can see that MBGD optimized LR's weight magnitude is stable after some epoches.

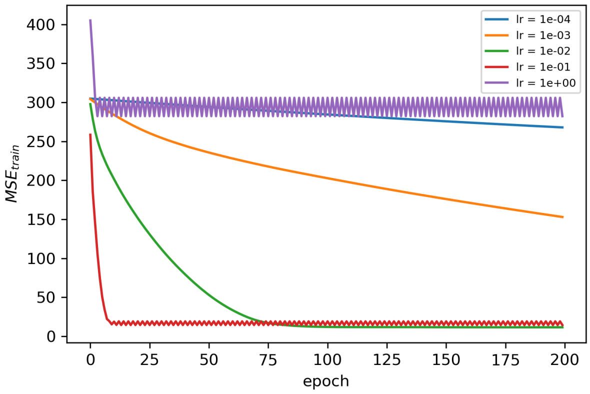

Fig. 5. The MSE in training set under different learning rate.

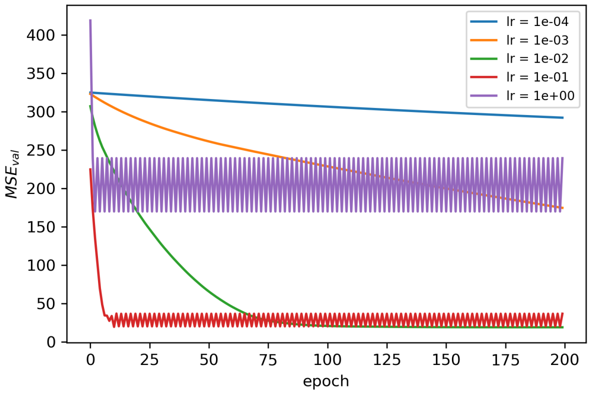

Fig. 6. The MSE in validation set under different learning rate.

But MBGD optimized weight's magnitude has a large margin(e.g. about 0.6) to CFS optimized weight. The reason is still unclear.

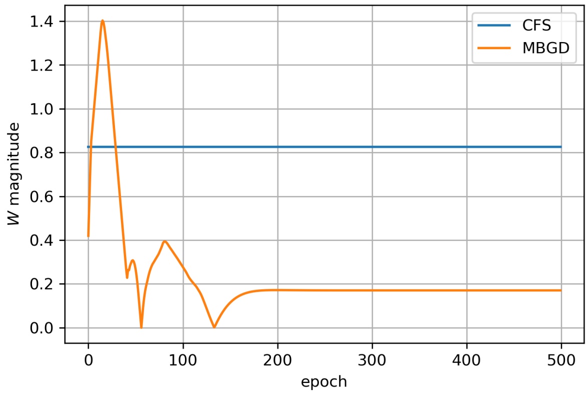

Fig. 7. The magnitude of W_{mbgd} under different epoches..

G. Performance

In this section, we report the performance on CFS optimized LR, RR and MBGD optimized LR in training set, validation set and test set. The λ is set to 1e-7 after validation. And batch size is set to 64 in MBGD, learing rate set to 0.01 and epoch set to 200 by validation.

From Table III, we can see that alougth MBGD optimized LR's MSE_{train} and MSE_{val} are not the best, but its result in MSE_{test} is outstanding. The reason remains unclear, maybe the same reason that the magnitude between CFS optimized LR and MBGD optimized LR has a large margin.

TABLE III
FINAL PERFORMANCE OF CFS OPTIMIZED LR, RR AND MBGD OPTIMIZED LR, BEST RESULT ARE IN BOLD.

	LR	RR	MBGD
MSE_{train}	**10.23**	**10.23**	11.37
MSE_{val}	**17.08**	**17.08**	19.04
MSE_{test}	10.20	10.20	**10.14**

IV. CONCLUSION

In this report, we explore the CFS optimized LR, CFS optimized RR and MBGD optimized LR. We conduct the experiments on the CFS optimized LR and RR in terms of performance and weight magnitude. We also explore GD, SGD, MBGD in terms of convergence and time efficiency. Moreover, we tuning the hyper-parameter learning rate in MBGD. We also compare the weight magnitude between CFS optimized LR and MBGD optimized LR. Finally, we report the performance in CFS optimized LR, RR and MBGD LR. After a series of experiments, we now get more insight of LR, RR and GD and its variants SGD and MBGD.