华南理工大学

South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Da Ren

Supervisor:
Qingyao Wu

Student ID:
201720144900

Grade:
Graduate

December 15, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

***Abstract*—Gradient descent plays an important role in machine learning. It can help us get the global optimal value of a function. However, one of the most important task in machine learning is to get the optimal value for loss function. Gradient descent is widely used in this task. However, people gradually find that traditional gradient descent performs not well when dealing with some tasks. Therefore, there are many modified gradient descent methods which are proposed. I implement NAG, RMSProp, AdaDelta and Adam in logistic regression and linear regression in this experiment.**

I. Introduction

GRADIENT descent is a important method in machine learning. The most important tasks in many machine learning models is to update or calculate their parameters which can lead the corresponding loss functions value to be minimal value. Although gradient descent is widely used, researchers find that it has its own limitation. Therefore, there are many modified gradient descent methods which are proposed, *e.g*. NAG, RMSProp, AdaDelta. In this experiment, I implement logistic regression and linear classification. And I use different methods to optimize their loss functions. My experiments' results show that all of them can lead the loss functions to global minimal. However, their speed may be different.

II. Methods and Theory

Gradient descent is flexible since it can be easily used in different kinds of loss function. It is based on the observation that if the multi-variable function $F(x)$ is defined and differentiable in a neighborhood of a point a, then $F(x)$ decreases fastest if one goes from a in the direction of the negative gradient of F at a. Therefore, if we want to minimize the value of $F(x)$, we can move a against the gradient of F. Therefore, a can be updated by Eq. (1).

$$a_{t+1} = a_t + \lambda \Delta F(a_t) \tag{1}$$

where a_t is the value of a at time t, λ is the learning rate which is always set to be a small value like 0.1. $\Delta F(a_t)$ is the gradient value of $F(a_t)$.

There are two experiments in my work. One is logistic regression, the other is linear classification. Therefore I have to choose different loss function to different tasks.

Logistic regression is a function that its output is limited from 0 to 1. It can be calculated as following:

$$y' = 1/(1 + e^{-z}) \tag{2}$$

where

$$z = W^T X + b \tag{3}$$

where W is the weight matrix and b is the bias variable. However, we can add 1 at the end of X so that the weight of the last column can act at the bias variable. Note X' to be the features which is add 1 at the last column. We can calculate z as following:

$$z = W^T X' \tag{4}$$

To train the logistic regression, I choose log-likehood function as my loss function. It can be calculated by Eq. (5)

$$L_1 = -1/n[\sum_{i=1}^{n} y_i log h_w(x_i) + (1 - y_i) log(1 - h_w(x_i))] \tag{5}$$

where

$$h_w(X) = 1/(1 + e^{-W^T X}) \tag{6}$$

In Eq. (5), n is the total number of data and y_i is the target value of data. The next step is to calculate the derivation of Eq. (5). The calculation process is in the following:

$$\begin{aligned} \Delta L_1/\Delta w = &-y * 1/h_w(X) * \Delta h_w(X)/\Delta W + \\ &(1 - y) * 1/(1 - h_w(X)) * \Delta h_w(X)/\Delta W \\ = &(h_w(X) - y)X \end{aligned} \tag{7}$$

After we get the the derivation of Eq. (6), we can calculate the derivative according to Eq. (7) and update weight matrix according Eq. (1)

In the linear classification task, I use the Eq. (8) to calculate output.

$$y' = W^T X' \tag{8}$$

Since the function is different, I choose another loss function. The loss function I choose is called hinge loss, which can be described in Eq. (9)

$$L_2 = \left\|W\right\|^2 /2 + C/m \sum_{i=1}^{m} max(0, 1 - y_i(W^T x_i + b)) \tag{9}$$

where C is a hyperparameter which can be adjusted in different dataset.

To use the gradient descent to optimize the weight matrix, we have to calculate the the derivation of Eq. (9).

$$\Delta L_2/\Delta w = W + C/m \sum_{i=1}^{m} g_w(x_i) \tag{10}$$

where

$$g_w(x_i) = \begin{cases} -y_i x_i & 1 - y_i(W^T x_i') \geq 0 \\ 0 & 1 - y_i(W^T x_i') < 0 \end{cases} \tag{11}$$

We can update the linear classification according to Eq. (1) and use Eq. (11) to calculate the derivative.

In this experiment, I not only used traditional gradient descent to train the models, but also use modified gradient descent. Their introduction is in the following.

Nesterov accelerated gradient(NAG) is method which considered momentum. It can be calculated as following:

$$g_t = \Delta L(W_{t-1} - \gamma v_{t-1}) \tag{12}$$

$$v_t = \gamma v_{t-1} + \eta g_t \tag{13}$$

$$W_t = W_{t-1} - v_t \tag{14}$$

where γ and η are hyperparameters which can be set to be 0.9 and 0.1 respectively.

RMSProp is a method which can solve the problem of AdaGrad that the learning rate tends to be 0. It can be calculated as following:

$$g_t = \Delta L(W_{t-1}) \tag{15}$$

$$G_t = \gamma G_{t-1} + (1-\gamma) g_t^2 \tag{16}$$

$$W_t = W_{t-1} - (\eta / \sqrt{G_t + \epsilon}) * g_t \tag{17}$$

AdaDelta is a special method that we don't need to set the learning rate. It can be calculated as following:

$$g_t = \Delta L(W_{t-1}) \tag{18}$$

$$G_t = \gamma G_{t-1} + (1-\gamma) g_t^2 \tag{19}$$

$$\Delta weight = -(\sqrt{\Delta_{t-1} + \epsilon}) / (\sqrt{G_t + \epsilon}) * g_t \tag{20}$$

$$weight = weight + \Delta weight \tag{21}$$

$$\Delta_t = \gamma \Delta_{t-1} + (1-\gamma) \Delta weight^2 \tag{22}$$

Adaptive estimates of lower-order moments (Adam) can be calculated as:

$$g_t = \Delta L(W_{t-1}) \tag{23}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \tag{24}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \tag{25}$$

$$m_t' = m_t / (1-\beta_1^t) \tag{26}$$

$$v_t' = v_t / (1-\beta_2^t) \tag{27}$$

$$weight = weight - \eta m_t' / (\sqrt{v_t' + \epsilon}) \tag{28}$$

where t is initialized to be 0 and add 1 at each step.

After choosing the loss function and calculate their corresponding derivation, I can use different kinds of gradient descent to train these two models. My experiment will be described in following.

III. Experiments

There will be two experiments which will be introduced. One is training the logistic regression and the other is training the linear classification. Both of them is trained in different kinds of gradient descent.

A. Dataset

This is two different tasks, so we have to use two different datasets. To train the linear regression model, I use the housing data in LIBSVM Data. There are 506 data and each of them has 13 attributes. For the linear classification tasks, I use the australian data in LIBSVM Data. There are 690 data and each of them has 14 attributes. All the datasets I use is the scaled version.

B. Implementation

In this section, I will introduce the linear regression experiment and linear classification experiment in detail.

In the linear regression experiment, the learning rate λ is set to be 0.1. The weight matrix is initialized randomly. After I read the data from file, I split the dataset into training set and validation set. There are 354 data in training set and 152 data in validation set. The iteration's number is set to 300. The loss value of training set and validation set is described in Fig. 1.

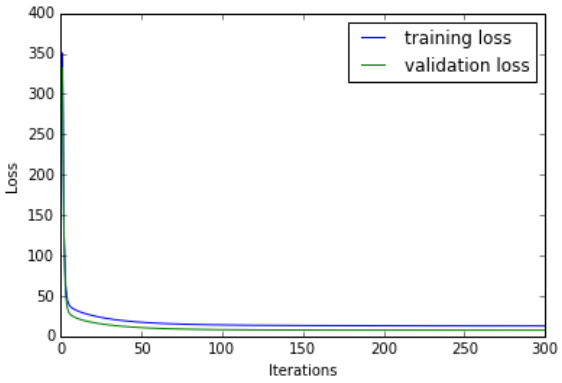

Fig. 1. The loss value of training set and test set in linear regression.

We can find that both of the loss of training set and test set decrease quickly at the beginning and close to unchange at the end of iterations. It means that the value of loss function is closed to global minimal value.

In the linear classfication experiment, the learning rate λ is set to be 0.1. The weight matrix is initialized randomly. I split the dataset into training set and validation set. There are 483 data in training set and 207 data in validation set. The iteration's number is set to 300. The loss value of training set and validation set is described in Fig. 2.

We can find that both of the loss of training set and test set decrease quickly at the beginning and close to unchange at the end of iterations. It means that the value of loss function is closed to global minimal value.

After finish training, I set a threshold to classify the data in validation set. I set the threshold to be 0, if the value calculated by model is smaller than 0, the corresponding data will be classified into one class (the label of this class is -1). If the value is larger or equal than 0, the corresponding data will be classified into another class (the label of this class is 1). And then I calculate the precision, recall and f1-score which is shown in Table I.

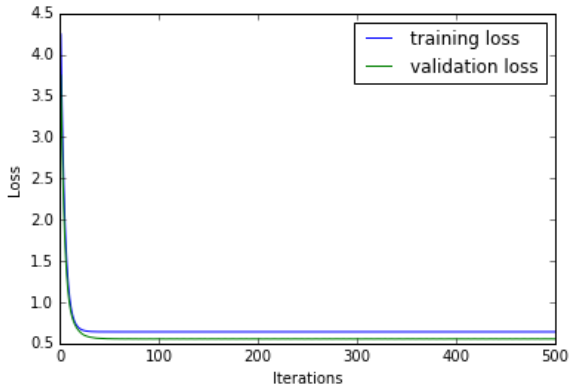

Fig. 2. The loss value of training set and test set in linear classification.

TABLE I
CLASSIFICATION RESULTS

	precision	recall	f1-score	support
class1	0.93	0.81	0.87	117
class2	0.79	0.92	0.85	90
avg / total	0.87	0.86	0.86	207

From Table I, we can find that the precision, recall and f1-score of classification is larger than 0.85. It is a good result of linear classification.

IV. CONCLUSION

In this experiment, I implement gradient descent in linear regression and linear classification. Both of their weight matrix is updated so that corresponding loss functions can be minimized. However, the loss function I use is convex, so that I don't need to care about the problem of local minimal. If the loss function is non-convex, gradient descent maybe fail to find the global minimal. Learning how to use gradient descent to get global minimal in non-convex function is my future work.