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:
Haishan Ao

Supervisor:
Qingyao Wu

Student ID：
201720145051

Grade:
Graduate

December 15, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

Abstract—This experiment implements logistic regression and linear classification by using stochastic gradient descent(SGD). We use different optimized methods(NAG, RMSProp, AdaDelta and Adam) to update model parameters.

I. Introduction

THIS experiment implements logistic regression and linear classification by using stochastic gradient descent. The experiment helps us compare and understand the difference between gradient descent and stochastic gradient descent. Compare and understand the differences and relationships between Logistic regression and linear classification. What's more, we can further understand the principles of SVM and practice on larger data.

II. Methods and Theory

A. Logistic Regression

Logistic regression is a kind of linear regression, and it also uses as a classifier. The logistic regression learns fitting parameters from the samples and fits the target values between [0, 1].And then it discretizes the target values to achieve classification. A logistic regression model is described as:

$$h_w(\mathrm{x}) = g(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

where g is a sigmoid function.

Then we have the loss function:

$$J(w) = -\frac{1}{n}\left[\sum_{i=0}^{n} y_i log h_w(x_i) + (1 - y_i) log\big(1 - h_w(x_i)\big)\right]$$

The derivation of the loss function is:

$$\nabla_W J(w) = \frac{1}{n}\sum_{i=0}^{n}\left(h_w(x_i) - y_i\right) x_i$$

B. Linear Classification

We use support vector machine (SVM) to implement linear classification. We choose normalization such that $w^T x_+ + b = +1$ and $w^T x_- + b = -1$ for the positive and negative support vectors respectively. Then the margin is given by

$$\frac{w}{\| \mathrm{w} \|} \cdot (x_+ - x_-) = \frac{w^T (x_+ - x_-)}{\| \mathrm{w} \|} = \frac{2}{\| \mathrm{w} \|}$$

Learning the SVM can be formulated as an optimization:

$$\min_{w,b} J: \frac{\lambda \| \mathrm{w} \|^2}{2} + \frac{C}{n}\sum_{i=1}^{n} \max\big(0, 1 - y_i(w^T x_i + b)\big)$$

where λ is the regularization parameter.

Then the derivation of the loss function is:

$$\nabla_w J(w,b) = \begin{cases} \lambda w \;\; if\; 1 - y_i(w^T x_i + b) < 0 \\ \lambda w + \dfrac{C}{n}\displaystyle\sum_{i=1}^{n}(-y_i x_i) \;\; otherwise \end{cases}$$

$$\nabla_b J(w,b) = \begin{cases} 0 \quad if\; 1 - y_i(w^T x_i + b) < 0 \\ \dfrac{C}{n}\displaystyle\sum_{i=1}^{n}(-y_i) \quad otherwise \end{cases}$$

C. Gradient Descent Optimization Algorithms

Gradient descent is a way to minimize an objective function $J(w)$ parameterized by a model's parameters *w* by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_W J(w)$ w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley. When the amount of data is larger, we usually use Stochastic gradient descent (SGD) is described as:

$$g_t \leftarrow \nabla J_i(w_{t-1})$$
$$w_t \leftarrow w_{t-1} - \eta g_t$$

In this experiment, we use some different methods to update the parameters as follows.

1) Nesterov accelerated gradient (NAG) is a way to give our momentum term this kind of prescience. We know that we will use our momentum term γv_{t-1} to move the parameters w. Computing $\mathrm{w} - \gamma v_{t-1}$ thus gives us an approximation of the next position of the parameters (the gradient is missing for the full update), a rough idea where our parameters are going to be. We can now effectively look ahead by calculating the gradient not w.r.t. to our current parameters w but w.r.t. the approximate future position of our parameters:

$$g_t \leftarrow \nabla J(w_{t-1} - \gamma v_{t-1})$$
$$v_t \leftarrow \gamma v_{t-1} + \eta g_t$$
$$w_t \leftarrow w_{t-1} - v_t$$

2)RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class.

RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates. RMSprop in fact is identical to the first update vector of Adadelta that we derived above:

$$g_t \leftarrow \nabla J(w_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma) g_t \odot g_t$$
$$w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

3)Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate.

Instead of inefficiently storing w previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients. The running average $\mathrm{E}[g^2]_t$ at time step t then depends (as a fraction γ similarly to the Momentum term) only on the previous average and the current gradient:

$$g_t \leftarrow \nabla J(w_{t-1})$$
$$\mathrm{E}[g^2]_t = \gamma \mathrm{E}[g^2]_{t-1} + (1-\gamma) g_t^2$$

We set γ to a similar value as the momentum term, around 0.9. For clarity, we now rewrite our vanilla SGD update in terms of the parameter update vector $\Delta\theta_t$:

$$\Delta\theta_t = g_t$$
$$\theta_{t+1} = \theta_t - \eta \Delta\theta_t$$

The parameter update vector of Adagrad that we derived previously thus takes the form:

$$\Delta\theta_t = \frac{1}{\sqrt{G_t + \epsilon}} \odot g_t$$

We now simply replace the diagonal matrix G_t with the decaying average over past squared gradients $\mathrm{E}[g^2]_t$:

$$\Delta\theta_t = \frac{1}{\sqrt{\mathrm{E}[g^2]_t + \epsilon}} g_t$$

As the denominator is just the root mean squared (RMS) error criterion of the gradient, we can replace it with the criterion short-hand:

$$\Delta\theta_t = \frac{1}{RMS[g]_t} g_t$$

The authors note that the units in this update (as well as in SGD, Momentum, or Adagrad) do not match, i.e. the update should have the same hypothetical units as the parameter. To realize this, they first define another exponentially decaying average, this time not of squared gradients but of squared parameter updates:

$$\mathrm{E}[\Delta\theta^2]_t = \gamma \mathrm{E}[\Delta\theta^2]_{t-1} + (1-\gamma) \Delta\theta_t^2$$

The root mean squared error of parameter updates is thus:

$$RMS[\Delta\theta]_t = \sqrt{\mathrm{E}[\Delta\theta^2]_t + \epsilon}$$

Since $RMS[\Delta\theta]_t$ is unknown, we approximate it with the RMS of parameter updates until the previous time step. Replacing the learning rate η in the previous update rule with $RMS[\Delta\theta]_{t-1}$ finally yields the Adadelta update rule:

$$\Delta\theta_t = \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t - \eta \Delta\theta_t$$

In this experiment we describe the process of Adadelta is shown as:

$$g_t \leftarrow \nabla J(w_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma) g_t \odot g_t$$
$$\Delta w_t \leftarrow -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \odot g_t$$
$$w_t \leftarrow w_{t-1} + \Delta w_t$$
$$\Delta_t \leftarrow \gamma \Delta_{t-1} + (1-\gamma) \Delta w_t$$

4)Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t , similar to momentum:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small.

They counteract these biases by computing bias-corrected first and second moment estimates:

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\widehat{v}_t = \frac{v_t}{1-\beta_2^t}$$

They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v}_t + \epsilon}} \widehat{m}_t$$

The process of Adam in this experiment is described as follows:

$$g_t \leftarrow \nabla J(w_{t-1})$$
$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$G_t \leftarrow \gamma G_t + (1-\gamma) g_t \odot g_t$$
$$\alpha \leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$
$$w_t \leftarrow w_{t-1} - \alpha \frac{m_t}{\sqrt{G_t + \epsilon}}$$

III. Experiment

A. Dataset

This experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features.

B. Implementation

The experiment process as follows:

Step1: Load the experiment dataset as training set and validation set respectively.

Step2: Initialize linear model parameters. Set all parameter into

zero, initialize it randomly or with normal distribution.

Step3: Define the loss function of the linear regression to be negative log likelihood function and the loss function of the linear classification to be Hinge loss.

Step4: Compute the gradient of the loss function with respect to the weight W and bias b in different Gradient Descent Optimization Algorithms.

Step5: Update the parameters W and b in different Gradient Descent Optimization Algorithms.

Step6: Repeat above steps for several times until convergence.

In both logistic regression and linear classification, we train the dataset with 500 epochs. And we update the gradient of parameter w by mini-batch SGD with the batch size of 5000. We set the same hyper-parameters in logistic regression and linear classification. And we initialize w randomly in normal distribution and the threshold is set as 0.5. Other parameters is shown in TABLE1.

Table 1 HYPER-PARAMETERS

	learning_rate	γ	ϵ	β
SGD	0.5	-	-	-
NAG	0.1	0.9	-	-
RMSProp	0.05	0.9	1e-8	-
AdaDelta	-	0.95	0.01	-
Adam	0.1	0.999	1e-8	0.9

What's more, parameter λ in SVM is set as 0.01. In addition, the negative class label in dataset is -1, as a result, we need to replace all -1 with 0 in advance when implementing logistic regression.

The loss of different optimization algorithms used in logistic regression on the validation set is shown in Fig. 1.

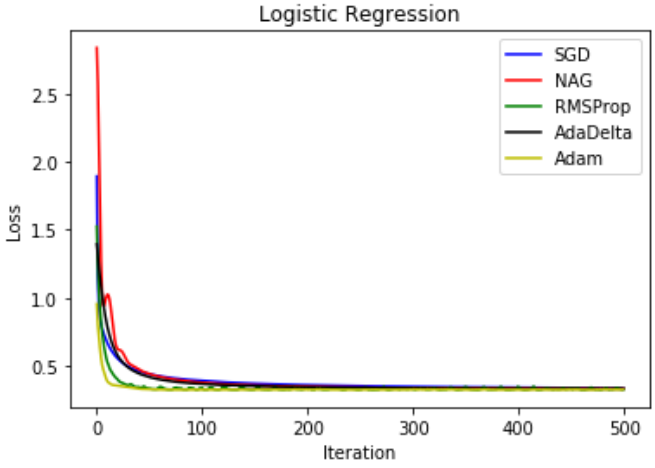

Fig.1. $L_{\text{validation}}$ of logistic regression

The loss of different optimization algorithms used in linear classification (SVM) on the validation set is shown in Fig. 2.

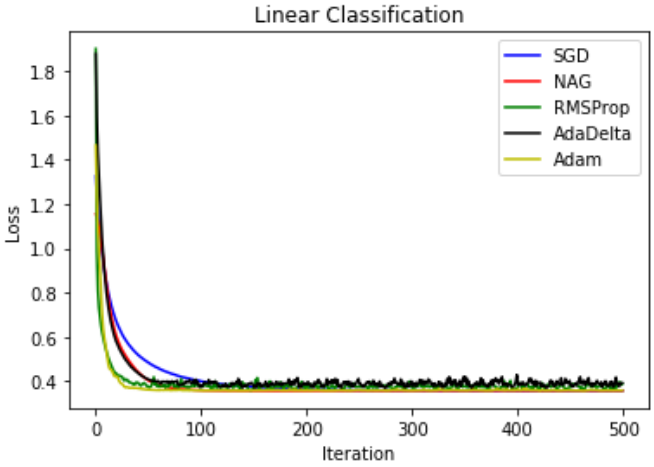

Fig.2. $L_{\text{validation}}$ of linear classification

The accuracy of different optimization algorithms used in linear classification (SVM) on the validation set is shown in Fig. 3.

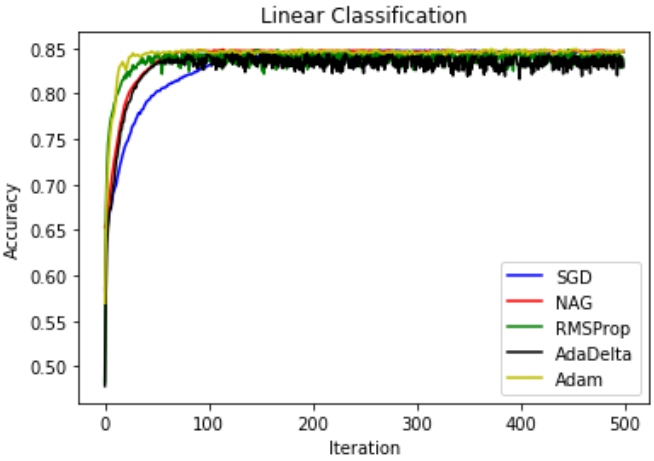

Fig.3. Accuracy of linear classification

According to the result of the experiment, the performance of Adam seems to be the best. RMSprop, Adam are relatively similar algorithms that perform similarly under similar conditions. SGD usually can reach the convergence but it needs more time than others.

IV. Conclusion

This experiment aims to compare and understand the differences and relationships between GD and SGD, the differences and relationships between logistic regression and linear classification. And further to learn different gradient descent optimization algorithms. SGD is always used in a large data set. When we meet a classification problem, we prefer logistic regression rather than linear classification. Although we should try different optimization algorithms aimed at different tasks. Adam can be the first choice if you really have no thoughts.SGD generally takes much time to train, but the results are more reliable under good initialization and learning rate scheduling schemes.