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:

常沛炜

Supervisor:

Mingkui Tan

Student ID：

201530611128

Grade:

Undergraduate

December 9, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent.

Abstract—

I. INTRODUCTION

In the Machine learning lab2, I did this experiment, Logistic Regression, Linear Classification and Stochastic Gradient Descent. That really make sense, which make me understand the algorithm of Logistic Regression and Linear Classification. And several methods for optimization. Now let me introduce my work in this experiment.

II. METHODS AND THEORY

Logistic regression, is a regression model where the dependent variable is categorical, which means, the output can only be discrete. We use sigmoid function to map the output into a series in [0,1] and define a threshold to judge the final answer. The linear classification model is SVM which I have done in the experiment before, so to be brief, that is a model which to find a hyperplane to classify the sample to be this side or the other side.

The main theory in this experiment are the four optimized methods. NGA, RMSProp, AdaDelta and Adam for the parameters updating during the process of gradient descent. I will speak briefly to show what they do.

The NGA, using a parameter called Momentum, to make the model converge in the local minimum point.

$$g_t = \nabla J(\theta_{t-1} - \gamma v_{t-1})$$

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

$$\theta_t = \theta_{t-1} - v_t$$

The RMSProp, using a G to remember the information of gradient before to judge that if the feature is usually updated, and use a parameter to prevent the learning rate converge to 0.

$$g_t = \nabla J(\theta_{t-1})$$

$$G_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

The AdaDelta, not need to set the initial learning rate, use the step length before to estimate the next step length, that is, the learning rate.

$$g_t = \nabla J(\theta_{t-1})$$

$$G_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$

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

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

$$\Delta_t = \gamma \Delta_{t-1} + (1-\gamma) \Delta\theta_t \odot \Delta\theta_t$$

And at last, the Adam, which is adaptive estimates of lower-order moments, it can make Initialization Bias Correction. It can converge in a very high speed.

$$g_t = \nabla J(\theta_{t-1})$$

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

$$G_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$

$$\alpha = \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{m_t}{\sqrt{G_t + \epsilon}}$$

III. EXPERIMENT

1. **In every missions, first I read the data in file,:**

```
# load the data file
X_train, Y_train = load_svmlight_file('a9a.txt')
X_test, Y_test = load_svmlight_file('a9a.t')
```

2. **Then initialize the parameters in each optimized methods:**

```
# NAG parameters
W_NAG = np.zeros((column + 1, 1))  # Merge the W and b
V_NAG = np.zeros((column + 1, 1))
momentum_NAG = 0.5
```

```
Vt = np.zeros((column + 1, 1))
# RMSProp parameters
W_RMSProp = np.zeros((column + 1, 1))
steadyNum_RMSProp = 0.0001
decreaseSpeed = 0.9
R_RMSProp = 0
# AdaDelta parameters
W_AdaDelta = np.zeros((column + 1, 1))
steadyNum_AdaDelta = 0.0001
R_AdaDelta = 0
# Adam parameters
W_Adam = np.zeros((column + 1, 1))
steadyNum_Adam = 0.0001
momentum1_Adam = 0.9
momentum2_Adam = 0.999
S_Adam = np.zeros((column + 1, 1))
R_Adam = 0


gradient_rounds = 200  # rounds for training
threshold = 0
```

The 2 steps are the same in logistic regression and linear classification.

3. Compute the gradient

In logistic regression:

```
predict = sigmond(np.dot(X, W_NAG))
wrong = Y_train.T - predict
gradient = np.dot(X.T, wrong)
```

In linear classification:

```
random_num = np.random.random_integers(row - 1)
it = np.reshape(X[random_num], (1, column + 1))
gradient=np.reshape(np.dot(Y_train.T[random_num], it), (column + 1, 1))
```

Yes that look like SGD

4. Updated the parameters

NAG:

```
V_NAG = momentum_NAG * V_NAG + learning_rate * gradient
W_NAG = W_NAG + V_NAG
```

RMSProp:

```
R_RMSProp = decreaseSpeed * R_RMSProp + (1 - decreaseSpeed) * np.dot(gradient.T, gradient)
change_RMSProp = learning_rate * (row / 2) / (steadyNum_RMSProp + np.sqrt(R_RMSProp)) * gradient
W_RMSProp = W_RMSProp + change_RMSProp
```

AdaDelta:

```
R_AdaDelta = R_AdaDelta + np.dot(gradient.T, gradient)
change_AdaDelta = learning_rate * (row / 2) / (steadyNum_AdaDelta + np.sqrt(R_AdaDelta)) * gradient
W_AdaDelta = W_AdaDelta + change_AdaDelta
```

Adam:

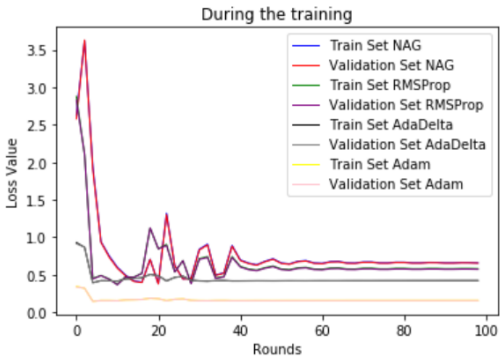
```
S_Adam = momentum1_Adam * S_Adam + (1 - momentum1_Adam) * gradient
R_Adam = momentum2_Adam * R_Adam + (1 - momentum2_Adam) * np.dot(gradient.T, gradient)
S_predict = S_Adam / (1 - momentum1_Adam)
R_predict = R_Adam / (1 - momentum2_Adam)
change_Adam = (learning_rate / (np.sqrt(R_predict) + steadyNum_Adam) * S_predict)
W_Adam = W_Adam + change_Adam
```

All of the updating processes are the same as the formulas before.

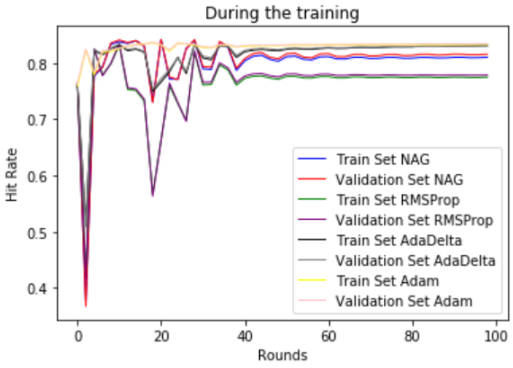
5. Results:

I use matplotlib to show the loss value and also the hit rate during the training. Let see the answers.

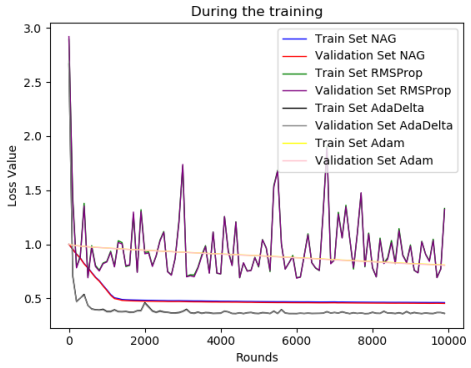
Logistic Regression: Loss:



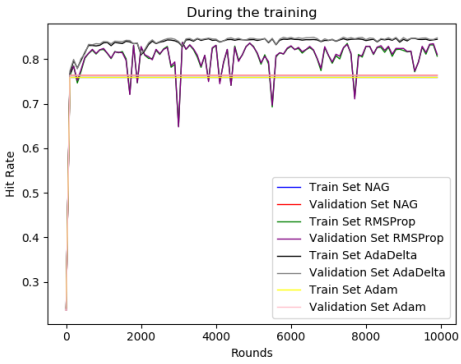
Hit Rate:



Linear Classification: Loss:



Hit Rate:



IV. CONCLUSION

After this experiment, I understand the algorithm of logistic regression and linear regression more and better. The experiment before was based on a dataset much smaller than this. And this time I need to use 4 optimized methods to update the parameters, which means nearly 4 times cost of time. So even it cost a bit long time.

By the implementing the 4 optimized methods, I can know what's the problem in gradient descent and how to solve it. Maybe the speed of convergence would be very low where we need to speed up, and maybe the speed would be very high and it cannot reach the local minimum! I can find a way to solve the problems in every scene, choosing the actual optimized method, and get the model what we want. Then after the experiment, I think I can build some models with low loss value, and it won't cost so much time even in a very huge data matrix.

I really hope that I can learn more and do best in the coming experiments.