The Experiment Report of Machine Learning



Grade:

Undergraduate

Student ID：

201530611128

Supervisor:

Mingkui Tan

Author:

常沛炜

December 9, 2017

**SUBJECT:**SOFTWARE ENGINEERING

**SCHOOL:** SCHOOL OF SOFTWARE ENGINEERING

[[1]](#footnote-1)Logistic Regression, Linear Classification and Stochastic Gradient Descent.

Abstract—

# 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.

# 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.

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.

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.

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.

# 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')

1. **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)

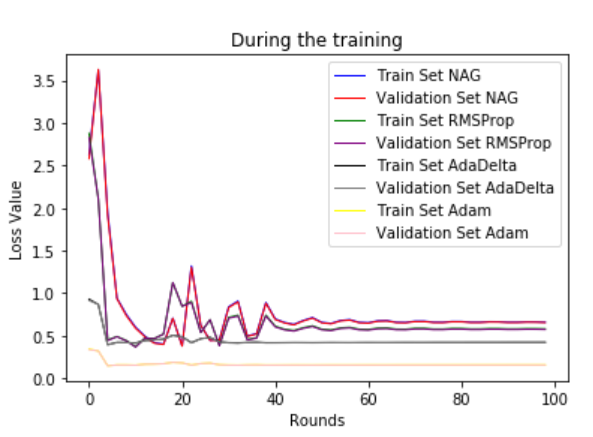
W\_Adam = W\_Adam + change\_Adam

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

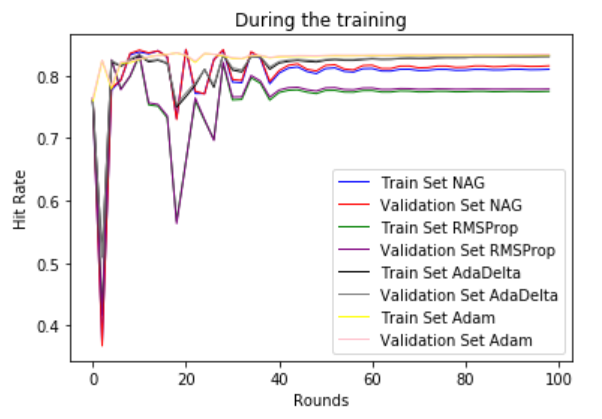
**5. Results:**

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

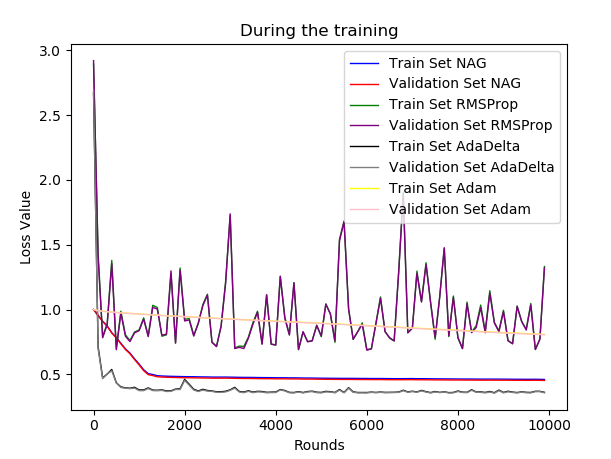
**Logistic Regression: Loss:**



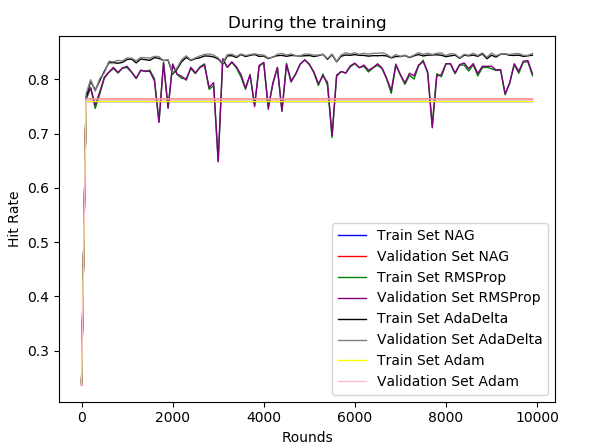
**Hit Rate:**



**Linear Classification: Loss:**



**Hit Rate:**



# 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.

1. [↑](#footnote-ref-1)