# The Experiment Report of Machine Learning

**SCHOOL:** SCHOOL OF SOFTWARE ENGINEERING

**SUBJECT:** SOFTWARE ENGINEERING

Author:
Wenhui Chen

Supervisor:
Mingkui Tan

Student ID：201530611272

Grade:
Undergraduate

December 14, 2017

# Experimental Study on Stochastic Gradient Descent for Solving Classification Problems

**Abstract—**

## I. Introduction

1. The experimental study is about study on stochastic gradient descent for classification problems. Two experiment about logistic regression and linear classification on stochastic gradient descent with updating model parameters using different optimized methods(NAG，RMSProp， AdaDelta and Adam).

## II. METHODS AND THEORY

Optimized methods NAG:

$$\begin{aligned} \mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1} - \gamma \mathbf{v}_{t-1}) \\ \mathbf{v}_t &\leftarrow \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t \end{aligned}$$

Optimized methods RMSProp:

$$\begin{aligned} \mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\ G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t \end{aligned}$$

Optimized methods AdaDelta :

$$\begin{aligned} \mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\ G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\ \Delta\boldsymbol{\theta}_t &\leftarrow -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t \\ \Delta_t &\leftarrow \gamma \Delta_{t-1} + (1-\gamma)\Delta\boldsymbol{\theta}_t \odot \Delta\boldsymbol{\theta}_t \end{aligned}$$

Optimized methods Adam:

$$\begin{aligned} \mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\ \mathbf{m}_t &\leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \\ G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\ \alpha &\leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t} \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t + \epsilon}} \end{aligned}$$

## III. Experiment

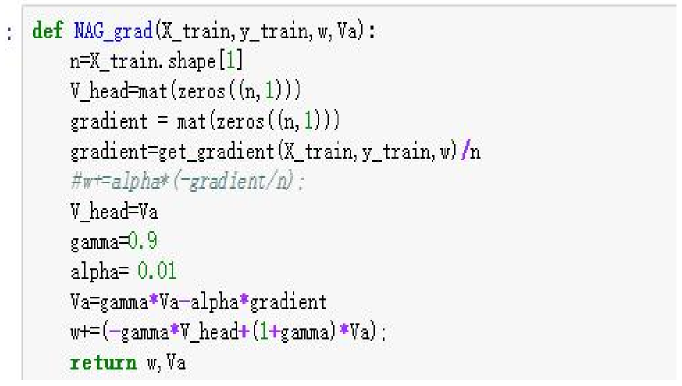
A. Data set :

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. Please download the training set and validation set. It should be transform yi to 0 when yi=-1 in logistic regression, if not the loss function may to be negative.

B. Implementation:

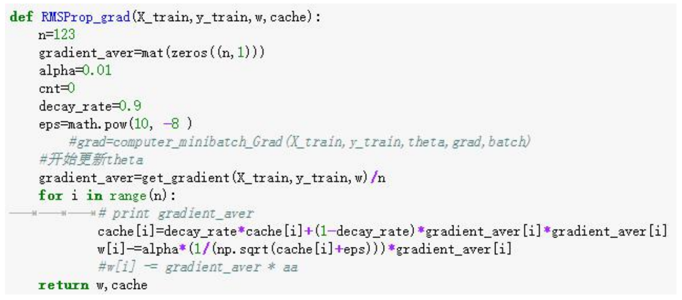
(1)logistic regression :

1. Initialize logistic regression model parameters, I choose initializing zeros,random numbers or normal distribution.
2. NAG:



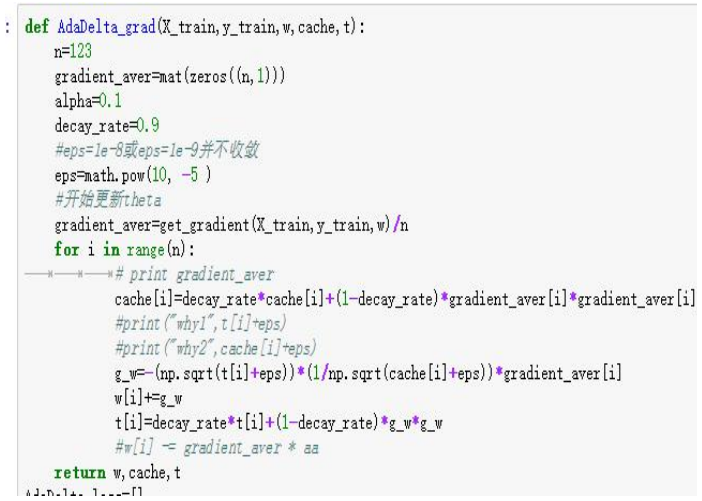
```
def NAG_grad(X_train,y_train,w,Va):
    n=X_train.shape[1]
    V_head=mat(zeros((n,1)))
    gradient = mat(zeros((n,1)))
    gradient=get_gradient(X_train,y_train,w)/n
    #w+=alpha*(-gradient/n);
    V_head=Va
    gamma=0.9
    alpha= 0.01
    Va=gamma*Va-alpha*gradient
    w+=(-gamma*V_head+(1+gamma)*Va);
    return w,Va
```

3.RMSProp



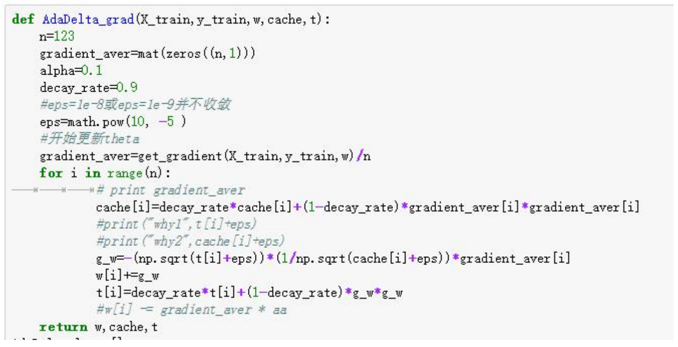
```
def RMSProp_grad(X_train,y_train,w,cache):
    n=123
    gradient_aver=mat(zeros((n,1)))
    alpha=0.01
    cnt=0
    decay_rate=0.9
    eps=math.pow(10, -8 )
        #grad=computer_minibatch_Grad(X_train,y_train,theta,grad,batch)
    #开始更新theta
    gradient_aver=get_gradient(X_train,y_train,w)/n
    for i in range(n):
            # print gradient_aver
            cache[i]=decay_rate*cache[i]+(1-decay_rate)*gradient_aver[i]*gradient_aver[i]
            w[i]-=alpha*(1/(np.sqrt(cache[i]+eps)))*gradient_aver[i]
            #w[i] -= gradient_aver * aa
    return w,cache
```

4.AdaDelta



```
def AdaDelta_grad(X_train,y_train,w,cache,t):
    n=123
    gradient_aver=mat(zeros((n,1)))
    alpha=0.1
    decay_rate=0.9
    #eps=1e-8或eps=1e-9并不收敛
    eps=math.pow(10, -5 )
    #开始更新theta
    gradient_aver=get_gradient(X_train,y_train,w)/n
    for i in range(n):
            # print gradient_aver
            cache[i]=decay_rate*cache[i]+(1-decay_rate)*gradient_aver[i]*gradient_aver[i]
            #print("why1",t[i]+eps)
            #print("why2",cache[i]+eps)
            g_w=-(np.sqrt(t[i]+eps))*(1/np.sqrt(cache[i]+eps))*gradient_aver[i]
            w[i]+=g_w
            t[i]=decay_rate*t[i]+(1-decay_rate)*g_w*g_w
            #w[i] -= gradient_aver * aa
    return w,cache,t
```

5.Adam:



```
def AdaDelta_grad(X_train,y_train,w,cache,t):
    n=123
    gradient_aver=mat(zeros((n,1)))
    alpha=0.1
    decay_rate=0.9
    #eps=1e-8或eps=1e-9并不收敛
    eps=math.pow(10, -5 )
    #开始更新theta
    gradient_aver=get_gradient(X_train,y_train,w)/n
    for i in range(n):
            # print gradient_aver
            cache[i]=decay_rate*cache[i]+(1-decay_rate)*gradient_aver[i]*gradient_aver[i]
            #print("why1",t[i]+eps)
            #print("why2",cache[i]+eps)
            g_w=-(np.sqrt(t[i]+eps))*(1/np.sqrt(cache[i]+eps))*gradient_aver[i]
            w[i]+=g_w
            t[i]=decay_rate*t[i]+(1-decay_rate)*g_w*g_w
            #w[i] -= gradient_aver * aa
    return w,cache,t
```

6.result

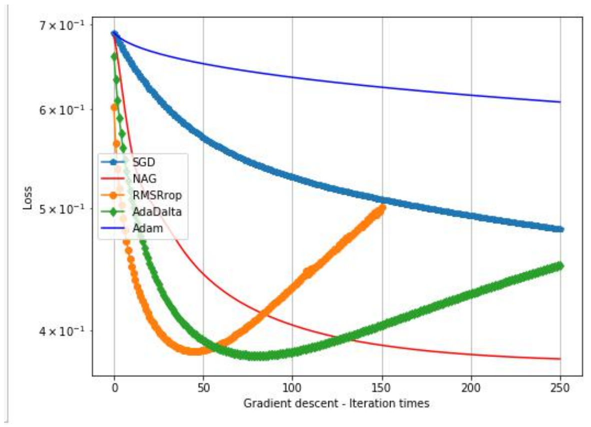
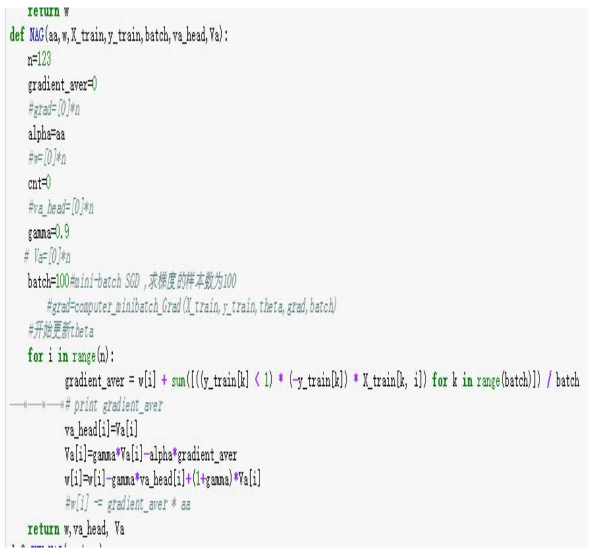


表 1 logistic regression

(2) classification problems:

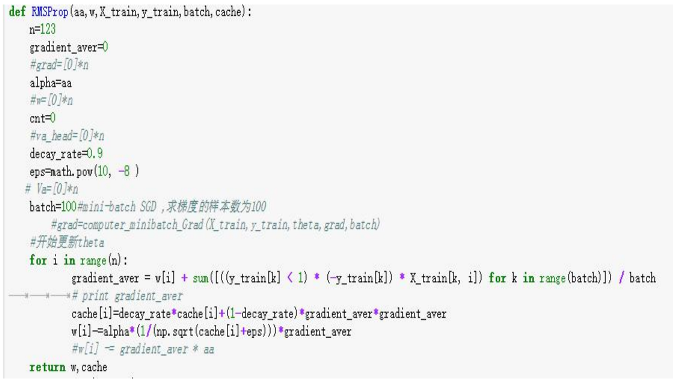
1.Initialize logistic regression model parameters, I choose random numbers or normal distribution.

7.NAG:



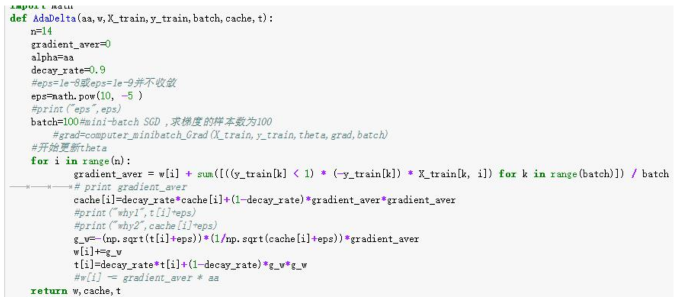
```
    return w
def NAG(aa,w,X_train,y_train,batch,va_head,Va):
    n=123
    gradient_aver=0
    #grad=[0]*n
    alpha=aa
    #w=[0]*n
    cnt=0
    #va_head=[0]*n
    gamma=0.9
   # Va=[0]*n
    batch=100#mini-batch SGD ,求梯度的样本数为100
        #grad=computer_minibatch_Grad(X_train,y_train,theta,grad,batch)
    #开始更新theta
    for i in range(n):
            gradient_aver = w[i] + sum([((y_train[k] < 1) * (-y_train[k]) * X_train[k, i]) for k in range(batch)]) / batch
            # print gradient_aver
            va_head[i]=Va[i]
            Va[i]=gamma*Va[i]-alpha*gradient_aver
            w[i]=w[i]-gamma*va_head[i]+(1+gamma)*Va[i]
            #w[i] -= gradient_aver * aa
    return w,va_head, Va
```

8.RMSProp:



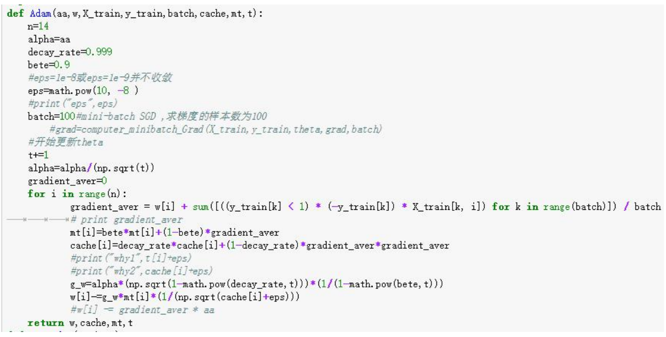
```
def RMSProp(aa,w,X_train,y_train,batch,cache):
    n=123
    gradient_aver=0
    #grad=[0]*n
    alpha=aa
    #w=[0]*n
    cnt=0
    #va_head=[0]*n
    decay_rate=0.9
    eps=math.pow(10, -8 )
   # Va=[0]*n
    batch=100#mini-batch SGD ,求梯度的样本数为100
        #grad=computer_minibatch_Grad(X_train,y_train,theta,grad,batch)
    #开始更新theta
    for i in range(n):
            gradient_aver = w[i] + sum([((y_train[k] < 1) * (-y_train[k]) * X_train[k, i]) for k in range(batch)]) / batch
            # print gradient_aver
            cache[i]=decay_rate*cache[i]+(1-decay_rate)*gradient_aver*gradient_aver
            w[i]-=alpha*(1/(np.sqrt(cache[i]+eps)))*gradient_aver
            #w[i] -= gradient_aver * aa
    return w,cache
```

9.AdaDelta :



```
import math
def AdaDelta(aa,w,X_train,y_train,batch,cache,t):
    n=14
    gradient_aver=0
    alpha=aa
    decay_rate=0.9
    #eps=1e-8或eps=1e-9并不收敛
    eps=math.pow(10, -5 )
    #print("eps",eps)
    batch=100#mini-batch SGD ,求梯度的样本数为100
        #grad=computer_minibatch_Grad(X_train,y_train,theta,grad,batch)
    #开始更新theta
    for i in range(n):
            gradient_aver = w[i] + sum([((y_train[k] < 1) * (-y_train[k]) * X_train[k, i]) for k in range(batch)]) / batch
            # print gradient_aver
            cache[i]=decay_rate*cache[i]+(1-decay_rate)*gradient_aver*gradient_aver
            #print("why1",t[i]+eps)
            #print("why2",cache[i]+eps)
            g_w=-(np.sqrt(t[i]+eps))*(1/np.sqrt(cache[i]+eps))*gradient_aver
            w[i]+=g_w
            t[i]=decay_rate*t[i]+(1-decay_rate)*g_w*g_w
            #w[i] -= gradient_aver * aa
    return w,cache,t
```

10. Adam:



```
def Adam(aa,w,X_train,y_train,batch,cache,mt,t):
    n=14
    alpha=aa
    decay_rate=0.999
    bete=0.9
    #eps=1e-8或eps=1e-9并不收敛
    eps=math.pow(10, -8 )
    #print("eps",eps)
    batch=100#mini-batch SGD ,求梯度的样本数为100
        #grad=computer_minibatch_Grad(X_train,y_train,theta,grad,batch)
    #开始更新theta
    t+=1
    alpha=alpha/(np.sqrt(t))
    gradient_aver=0
    for i in range(n):
            gradient_aver = w[i] + sum([((y_train[k] < 1) * (-y_train[k]) * X_train[k, i]) for k in range(batch)]) / batch
            # print gradient_aver
            mt[i]=bete*mt[i]+(1-bete)*gradient_aver
            cache[i]=decay_rate*cache[i]+(1-decay_rate)*gradient_aver*gradient_aver
            #print("why1",t[i]+eps)
            #print("why2",cache[i]+eps)
            g_w=alpha*(np.sqrt(1-math.pow(decay_rate,t)))*(1/(1-math.pow(bete,t)))
            w[i]-=g_w*mt[i]*(1/(np.sqrt(cache[i]+eps)))
            #w[i] -= gradient_aver * aa
    return w,cache,mt,t
```

## Result

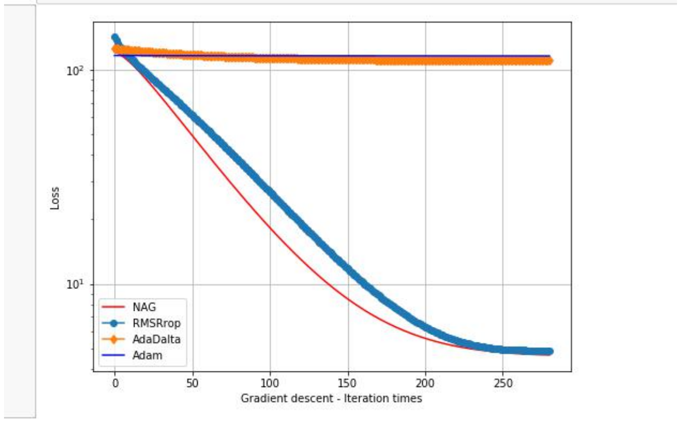


表 2SVM

## IV. CONCLUSION

11. The two experiment of logistic regression and SVM on Stochastic Gradient Decent Methods with four methods to update parameters which is NAG， RMSProp， AdaDelta and Adam. First, on my experiment，I found the Adam was most convenient among them because it do not to set the parameter of learning rate. The loss function of Adam is accelerate the convergence speed. However the value of convergence is high. I think it is because the learning rate is descent too fast. And RMSProp is the most fast convergence speed. Second, the SVM is overfitting with initializing zeros,

so I should choose Initialize logistic initializing gauss distribution, but the initial value of loss function is large which cause the change of AdaDelta and Adam is small comparing NAG and RMSProp.