华南理工大学

South China University of Technology

The Experiment Report of *Machine Learning*

College **Software College**

Subject **Software Engineering**

Members JunPeng Su

Student ID 201530612743

E-mail sujunpeng.chn.@gmail.com

Tutor MingKui Tan

Date submitted 2017. 12 . 14

1. Topic:

Comparison of Various Stochastic Gradient Descent Methods for Solving Classification Problems

2. Time: 2017-12-02 2:00-5:00 PM B7-138

3. Reporter: JunPeng Su

4. Purposes:

1) Compare and understand the difference between Gradient Descent and Stochastic Gradient Descent.
2) Compare and understand the difference and relationship between Logistic Regression and Linear Classification.
3) Further understand the principles of SVM and practice on larger data.
4) Compare the performance of different variants of Stochastic Gradient Descent

5. Data sets and data analysis:

Experimental dataset is a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features.

6. Experimental steps:

Logistic Regression

1) Load train data and validation data.
2) Set the hyper-parameters of Logistic Regression model and select an appropriate threshold.
3) Initialize linear regression model parameters by setting all parameters into 0.
4) Choose the loss function and calculate the derivative.
5) Calculate gradient towards the loss function from partial samples
6) Update linear regression model using different optimization methods (NAG, RMSProp, AdaDelta, Adam).
7) Mark samples whose predict score greater than threshold as positive , on the contrary as negative.
8) Predict under validation dataset and calculate the loss
9) Repeat step 5) - 8) several times
10) Draw graph of loss with the number of iterations

Linear Classification

1) Load train data and validation data.
2) Set the hyper-parameters of Linear Classification model and select an appropriate threshold.

3) Initialize SVM model parameters by setting all parameters into 0.
4) Choose the loss function and calculate the derivative.
5) Calculate gradient towards the loss function from partial samples
6) Update linear regression model using different optimization methods (NAG, RMSProp, AdaDelta, Adam).
7) Mark samples whose predict score greater than threshold as positive , on the contrary as negative.
8) Predict under validation dataset and calculate the loss
9) Repeat step 5) - 8) several times
10) Draw graph of loss with the number of iterations

7. Code:

Logistic Regression

#calculate gradient and loss

```
#define loss function
def loss(w,x,y):
    #get data volume
    data_volume=x.shape[0]
    #calculate positive probability
    positive_probability=sigmoid(w,x)
    #calculate negative probability using positive probability
    negative_probability=1-positive_probability
    #sum the loss
    loss_total=np.dot(y.T,np.log(positive_probability))+np.dot((1-y).T,np.log(negative_probability))
    #mean the loss
    result=-loss_total/data_volume
    return result

#define gradient calculating function
def gradient(w,x,y):
    #get data volume
    data_volume=x.shape[0]
    #calculate positive probability
    positive_probability=sigmoid(w,x)
    #calculate gradient
    grad=np.dot((positive_probability-y).T,x).T/data_volume
    return grad
```

#RMSProp optimization method

```
#define RMSProp optimization method
def RMSProp_Optimization(threshold,w,x_train,y_train,x_test,y_test,runs):
    #print a signal
    print('RMSProp Optimization')
    #set parameter
    learning_rate=0.02
    gama=0.9
    e=0.0000001
    #initialize medium variable
    G=0
    #initialize container for loss
    test_losses=np.zeros(runs)
    #optimize
    for i in range(runs):
        #generate random integer
        index=np.arange(5000)
        np.random.shuffle(index)
        #get a batch of stochastic training sample
        x_train_part=x_train[index]
        y_train_part=y_train[index]
        #calculate gradient
        grad=gradient(w,x_train_part,y_train_part)
        G=gama*G+(1-gama)*np.dot(grad.T,grad)
        w=w-learning_rate/np.sqrt(G+e)*grad
        #calculate the loss
        error=loss(w,x_test,y_test)
        #save the loss in container
        test_losses[i]=error
        #log every 100 epochs
        if(i%100==99):
            #calculate the accuracy
            accur=accuracy(threshold,w,x_test,y_test)
            print("epochs "+str(i+1)+" Accuracy: "+str(accur))
    #draw graph of loss
    x_axis=np.arange(runs)
    plt.plot(x_axis,test_losses,'r',label='RMSProp')
    return w
```

#preprocess data

```python
#load data
train_data=load_svmlight_file("a9a.txt")
test_data=load_svmlight_file("a9a.t",n_features=123)
#sperate features and labels
X_train,y_train=train_data[0],train_data[1]
X_test,y_test=test_data[0],test_data[1]
#change the range of y from [-1,1] to [0,1]
y_train=y_train/2+0.5
y_test=y_test/2+0.5
#change sparse martix to numpy array
X_train=X_train.toarray()
X_test=X_test.toarray()
#integrate features with bias
#create bias vector
X_train_bias=np.ones((X_train.shape[0],1))
X_test_bias=np.ones((X_test.shape[0],1))
#add bias column to features
X_train=np.hstack([X_train,X_train_bias])
X_test=np.hstack([X_test,X_test_bias])
```

Linear Classification

#calculate the gradient and loss

```python
#define loss function
def loss(C,threshold,w,x,y):
    loss_part_1=np.dot(w.T,w)/2
    loss_part_2=0
    #get the volume of data
    data_volume=x.shape[0]
    #calculate loss for each record
    for i in range(data_volume):
        hinge=1-y[i]*(np.dot(x[i],w))
        if hinge>=0:
            loss_part_2+=hinge
    #calculate the total loss
    loss_all=(loss_part_1+C*loss_part_2)/data_volume
    return loss_all

#define gradient calculating function
def gradient(C,threshold,w,b,x,y):
    grad_w=np.zeros_like(w)
    grad_b=0
    #get the volume of data
    data_volume=x.shape[0]
    #calculate gradient for each record
    for i in range(data_volume):
        hinge=1-y[i]*(np.dot(x[i],w))
        #deal with uncontinuous gradient
        if hinge>=threshold:
            grad_w=grad_w-C*y[i]*x[i].T
            grad_b=grad_b-C*y[i]

    grad_w=grad_w+w
    return grad_w/data_volume,grad_b/data_volume
```

#NAG optimization method

```python
#define NAG optimization method
def NAG_Optimization(C,threshold,w,b,x_train,y_train,x_test,y_test,runs):
    #print a signal
    print('NAG')
    #set parameter
    gama=0.9
    learning_rate=0.01
    #initialize medium variables
    v_w=np.zeros_like(w)
    v_b=0
    #initialize container for loss
    test_losses=np.zeros(runs)
    #oprimize
    for i in range(runs):
        #generate random integer
        index=np.arange(5000)
        np.random.shuffle(index)
        #get a batch of stochastic training sample
        x_train_part=x_train[index]
        y_train_part=y_train[index]
        #calculate gradient
        grad_w,grad_b=gradient(C,threshold,w,b,x_train_part,y_train_part)
        grad_w=grad_w+gama*v_w
        grad_b=grad_b+gama*v_b
        #update v
        v_w=gama*v_w+learning_rate*grad_w
        v_b=gama*v_b+learning_rate*grad_b
        #update weight vector
        w=w-v_w
        b=b-v_b
        #calculate the loss
        error=loss(C,threshold,w,x_test,y_test)
        #save the loss in container
        test_losses[i]=error
        #log every 100 epochs
        if(i%100==99):
            #calculate the accuracy
            accur=accuracy(threshold,w,b,x_test,y_test)
            print("epochs "+str(i+1)+" Accuracy: "+str(accur))
    #draw graph of loss
    x_axis=np.arange(runs)
    plt.plot(x_axis,test_losses,'b',label='NAG')
    return w
```

#preprocess the data

```
#Load data
train_data=load_svmlight_file("a9a.txt")
test_data=load_svmlight_file("a9a.t",n_features=123)
#sperate features and labels
X_train,y_train=train_data[0],train_data[1]
X_test,y_test=test_data[0],test_data[1]
#change sparse martix to numpy array
X_train=X_train.toarray()
X_test=X_test.toarray()
```

p.s. Only part of the codes are displayed here, the whole codes are in RegressionExperiment.ipynb and ClassificationExperiment.ipynb.

8. The initialization method of model parameters:

Set all model parameters into 0.

9. The selected loss function and its derivatives:

Logistic Regression

Loss function:

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

Derivatives:

$$\frac{\partial J(w)}{\partial w} = (h_w(X) - y)X$$

Linear Classification

loss function:

$$L_D(w) = \frac{||w||^2}{2} + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(w^T x_i + b)\right)$$

derivatives:

$$\frac{\partial L_D(w,b)}{\partial w} = w + C\sum_{i=1}^{n} g_w(x_i)$$

$$\frac{\partial L_D(w,b)}{\partial b} = C\sum_{i=1}^{n} g_b(x_i)$$

here,

$$g_w(x_i) = \begin{cases} -y_i x_i & 1 - y_i(w^T x_i + b) \geq 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases}$$

$$g_b(x_i) = \begin{cases} -y_i & 1 - y_i(w^T x_i + b) \geq 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases}$$

10. Experimental results and curve:

Hyper-parameter selection:

Logistic Regression

RMSProp			AdaDelta		Adam				NAG		Momentum	
η	γ	e	γ	e	β	γ	η	e	γ	η	γ	η
0.001	0.9	1e-7	0.95	1e-7	0.9	0.999	0.1	1e-7	0.9	0.001	0.9	0.0001
0.005	0.9	1e-7	0.90	1e-7	0.9	0.999	0.05	1e-7	0.9	0.005	0.9	0.001
0.01	0.9	1e-7	0.60	1e-7	0.8	0.999	0.1	1e-7	0.9	0.01	0.9	0.05
0.01	0.9	1e-7	0.60	1e-5	0.8	0.999	0.1	1e-7	0.9	0.01	0.9	0.05
0.1	0.9	1e-7	0.80	1e-5	0.8	0.999	0.1	1e-6	0.9	0. 1	0.9	0.05
0.05	0.9	1e-7	0.80	1e-4	0.8	0.8	0.1	1e-6	0.9	0.05	0.9	0.05
0.02	0.9	1e-7	0.80	1e-4	0.8	0.9	0.1	1e-6	0.9	0.01	0.9	0.05

p.s. To compare the performance of different optimization methods, epochs used in all methods are set into the same number 1000.

Linear Classification

RMSProp			AdaDelta		Adam				NAG		Momentum	
η	γ	e	γ	e	β	γ	η	e	γ	η	γ	η
0.005	0.9	1e-7	0.95	1e-7	0.9	0.999	0.1	1e-7	0.9	0.001	0.9	0.01
0.02	0.9	1e-7	0.80	1e-4	0.8	0.9	0.1	1e-6	0.9	0.01	0.9	0.05
0.1	0.9	1e-7	0.95	1e-5	0.9	0.999	0.2	1e-7	0.9	0.001	0.9	0.001
0.005	0.9	1e-7	0.95	1e-5	0.9	0.999	0.05	1e-7	0.9	0.1	0.9	0.1
0.005	0.9	1e-7	0.95	1e-5	0.9	0.999	0.05	1e-7	0.9	0.05	0.9	0.08
0.005	0.9	1e-7	0.95	1e-5	0.9	0.999	0.05	1e-7	0.9	0.01	0.9	0.02
0.005	0.9	1e-7	0.95	1e-5	0.3	0.999	0.2	1e-7	0.9	0.01	0.9	0.02

p.s. To compare the performance of different optimization methods, epochs used in all methods are set into the same number 400.

Predicted Results (Best Results):

Logistic Regression

RMSProp			AdaDelta		Adam				NAG		Momentum	
η	γ	e	γ	e	β	γ	η	e	γ	η	γ	η
0.02	0.9	1e-7	0.80	1e-4	0.8	0.9	0.1	1e-6	0.9	0.01	0.9	0.05

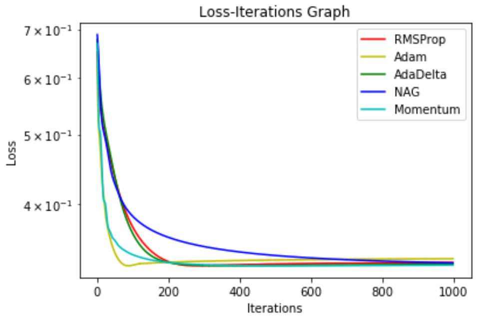
Linear Classification

RMSProp			AdaDelta		Adam				NAG		Momentum	
η	γ	e	γ	e	β	γ	η	e	γ	η	γ	η

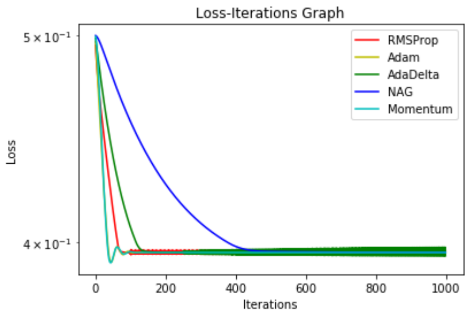
0.005	0.9	1e-7	0.95	1e-5	0.3	0.999	0.2	1e-7	0.9	0.01	0.9	0.02

Loss curve:

Logistic Regression



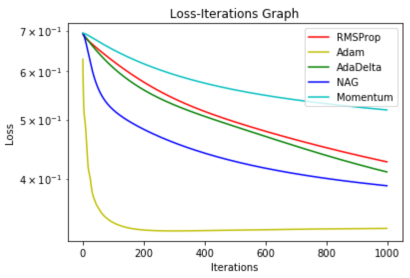
Linear Classification



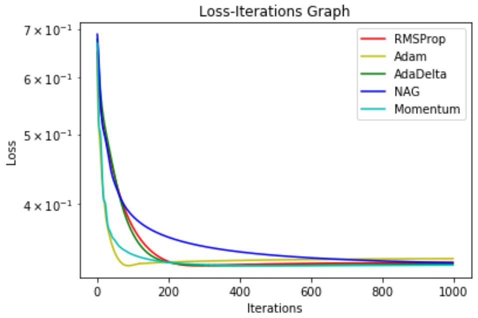
11. Results analysis:

Logistic Regression

The loss curves below are generated in the first try when all hyper-parameters are choose randomly. It indicates that Adam Optimization method performs best. As an improvement version of Momentum Optimization method, NAG Optimization method performs much better than Momentum. And RMSProp Optimization method and AdaDelta Optimization method are comparable.



The next loss curves are generated after the process of tuning hyper-parameters. It shows that selecting appropriate hyper-parameters improve the performance and all optimization methods are able to obtain similar test loss at last.

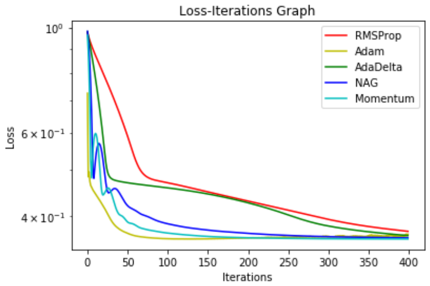


Accuracy of different optimization method is shown in the table below. NAG obtain the highest accuracy while Adam obtain the lowest.

Optimization Method	RMSProp	AdaDelta	Adam	NAG	Momentum
Accuracy	0.84381	0.84467	0.84264	0.84860	0.84546

Linear Classification

The loss curves below are generated after the process of tuning hyper-parameters. The graph shows that Adam optimization method and Momentum optimization method have the best performance. And all method obtain similar test loss at last.



Accuracy of different optimization method is shown in the table below. Momentum obtain the highest accuracy while RMSProp obtain the lowest.

Optimization Method	RMSProp	AdaDelta	Adam	NAG	Momentum
Accuracy	0.80075	0.81316	0.81795	0.84215	0.84706

Comparison between loss curves of logistic regression and linear classification indicates that Adam optimization method is the fastest to reduce the test loss. However, accuracy of the model optimized by Adam optimization method is unsatisfied which means that using Adam optimization method is prone to over-fitting.

12. Similarities and differences between logistic regression and linear classification：

Similarities: Logistic regression and linear classification are both attempting to represent the relationship between features and labels in a linear way. They both use the formula y=Wx.

Difference: Logistic regression need to transform y to a real value between 0 and 1 which represents the probability of positive or negative sample.

13. Summary:

The loss curves show that Adam optimization method is the fastest optimization method among all methods. Adam optimization method reduces the loss fast and is easy to select hyper-parameters while it leads to over-fitting sometimes. From the accuracy table, I learned that NAG optimization method and Momentum optimization method perform well.

By comparing the loss curves of the same optimization method using different hyper-parameters, I realize the importance of selecting appropriate hyper-parameters. After a hard struggle in tuning hyper-parameters, I learned some skills about selecting appropriate hyper-parameters. Too large learning rate will make the loss curve fluctuates heavily. Different hyper-parameters have different influence on the performance such as hyper-parameter e behaves better than γ in speeding up the process of loss reduction.