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:
Longkang Li

Supervisor:
Qingyao Wu

Student ID:
201530612002

Grade:
Undergraduate

December 12, 2017

Linear Regression, Linear Classification and Gradient Descent

Abstract— This report is mainly to illustrate the experiment of logistic regression, linear classification and stochastic gradient descent (SGD)

I. INTRODUCTION

Logistic regression and linear classification are the two of most fundamental machine learning models,The model of logistic regression is a nonlinear model, Stochastic gradient descent(SGD), an improved version of traditional GD, accelerates the process reaching the solution,We implement SVM on big data sets.

II. METHODS AND THEORY

Logistic Regression and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initialize logistic regression model parameters, you can consider initializing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation, find more detail in PPT.
4. Calculate gradient G toward loss function from partial samples.
5. Update model parameters using different optimized methods(NAG, RMSProp, AdaDelta and Adam).
6. Select the appropriate threshold, mark the sample whose predict scores **greater than the threshold as positive, on the contrary as negative**. Predict under validation set and get the different optimized method loss L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}.
7. Repeat step 4 to 6 for several times, and drawing graph of L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam} with the number of iterations.

Linear Classification and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initialize SVM model parameters, you can consider initializing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation, find more detail in PPT.
4. Calculate gradient G toward loss function from partial samples.
5. Update model parameters using different optimized methods(NAG, RMSProp, AdaDelta and Adam).
6. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method loss L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$and L_{Adam}..
7. Repeat step 4 to 6 for several times, and drawing graph of L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$and L_{Adam}. with the number of iterations.

III. EXPERIMENT

Logistic Regression:

Loss function: $-\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$

Derivatives: $-\frac{1}{m}\sum_{i=1}^{m} \alpha(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$

Linear Classification

Loss function: hingeloss $\frac{\|w\|^2}{2} + C\sum_{i=1}^{n} max\,(0,1 - y_i(x_i w + b))$

Derivatives: $g_t = \begin{cases} w + C\sum_{i=1}^{n} - x_i^T y_i & 1 - y_i(x_i w + b) \geq 0 \\ w & 1 - y_i(x_i w + b) < 0 \end{cases}$

NAG:

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

RMSProp

$$g_t = \nabla J(\theta_{t-1})$$
$$G_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$
$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

AdaDelta

$$g_t = \nabla J(\theta_{t-1})$$
$$G_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$
$$\Delta\theta_t = -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \odot g_t$$
$$\theta_t = \theta_{t-1} + \Delta\theta_t$$
$$\Delta_t = \gamma\Delta_{t-1} + (1-\gamma)\Delta\theta_t \odot \Delta\theta_t$$

Adam

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

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$G_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$
$$\alpha = \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$
$$\theta_t = \theta_{t-1} - \alpha \frac{m_t}{\sqrt{G_t - \epsilon}}$$

PARAMETER:

NAG：iter_num=2000
learning_rate=0.002

RMSProp：g=0
iter_num=2000
learning_rate=0.1

AdaDelta：g=0.0
delta=0.003
iter_num=2000

Adam:g=0.0
learning_rate=0.1
iter_num=2000

CODE:

RegressionExperiment:

```
import numpy as np
from sklearn.datasets import load_svmlight_file
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
data=load_svmlight_file('a9a.txt')
X_train=data[0].todense()
y_train=data[1]
```

```
data=load_svmlight_file('a9a.t')
X_validation=data[0].todense()
y_validation=data[1]
```

```
X_train=X_train.T
x_0=np.ones(X_train.shape[1])
X_train=np.row_stack((X_train,x_0))
y_train=np.mat(y_train)

X_validation=X_validation.T
a=np.zeros(X_validation.shape[1])
x_0=np.ones(X_validation.shape[1])
X_validation=np.row_stack((X_validation,a))
X_validation=np.row_stack((X_validation,x_0))
y_validation=np.mat(y_validation)
```

```
for i in range(y_train.shape[1]):
    if y_train[0,i]==-1.0:
        y_train[0,i]=0
for i in range(y_validation.shape[1]):
    if y_validation[0,i]==-1.0:
        y_validation[0,i]=0
```

```
def Loss(w,x,y):
    loss=0
    tmp=w.T*x
    loss+=-(tmp*y.T)[0,0]
    for i in range(tmp.shape[1]):
        loss+=np.log(1+np.exp(tmp[0,i]))
    return loss/float(y.shape[1])
```

```
def SGD(w,x,y,learning_rate):
    gradient=np.mat(np.zeros(w.shape))
    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient+=-x[:,i]*(y[0,i]-(np.exp(w.T*x[:,i])/(1+np.exp(w.T*x[:,i]))))
    w=w-learning_rate*gradient
    return w
```

```
def SGD_NAG(w,m,x,y,learning_rate):
    momentum=m
    gradient=np.mat(np.zeros(w.shape))

    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient+=-x[:,i]*(y[0,i]-(np.exp((w-momentum).T*x[:,i])/(1+np.exp((w-momentum).T*x[:,i]))))
    momentum=0.9*momentum+learning_rate*gradient
    w=w-momentum
    return w,momentum
```

```
def SGD_RMSProp(w,g,x,y,learning_rate):
    gradient=np.mat(np.zeros(w.shape))
    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient+=-x[:,i]*(y[0,i]-(np.exp(w.T*x[:,i])/(1+np.exp(w.T*x[:,i]))))
    g=0.9*g+0.1*(gradient.T*gradient)[0,0]
    w=w-(learning_rate/(np.sqrt(g+1e-8)))*gradient
    return w,g
```

```
def SGD_AdaDelta(w,g,delta,x,y):
    gradient=np.mat(np.zeros(w.shape))
    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient=-x[:,i]*(y[0,i]-(np.exp(w.T*x[:,i])/(1+np.exp(w.T*x[:,i]))))
    g=0.95*g+0.05*(gradient.T*gradient)[0,0]

    step_length=(np.sqrt(delta+1e-8)/np.sqrt(g+1e-8))*gradient
    w=w-step_length
    delta=0.95*delta+0.05*(step_length.T*step_length)[0,0]
    return w,g,delta
```

```
def SGD_Adam(w,m,g,x,y,learning_rate,iter_num):
    gradient=np.mat(np.zeros(w.shape))
    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient=-x[:,i]*(y[0,i]-(np.exp(w.T*x[:,i])/(1+np.exp(w.T*x[:,i]))))
    m=0.9*m+0.1*gradient
    g=0.999*g+0.001*gradient.T*gradient
    alpha=learning_rate*np.sqrt(1-0.999**iter_num)/(1-0.9**iter_num)
    w=w-alpha*m/np.sqrt(g+1e-8)
    return w,m,g
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
iter_num=2000
learning_rate=0.002

loss_validation=[]
loss_validation.append(Loss(W,X_validation,y_validation))
for i in range(iter_num):
    W=SGD(W,X_train,y_train,learning_rate)
    loss_validation.append(Loss(W,X_validation,y_validation))
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
momentum=np.mat(np.zeros(X_train.shape[0])).T
iter_num=2000
learning_rate=0.002

loss_NAG_validation=[]
loss_NAG_validation.append(Loss(W,X_validation,y_validation))
for i in range(iter_num):
    W,momentum=SGD_NAG(W,momentum,X_train,y_train,learning_rate)

    loss_NAG_validation.append(Loss(W,X_validation,y_validation))
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
g=0
iter_num=2000
learning_rate=0.1

loss_RMSProp_validation=[]
loss_RMSProp_validation.append(Loss(W,X_validation,y_validation))
for i in range(iter_num):
    W,g=SGD_RMSProp(W,g,X_train,y_train,learning_rate)
    loss_RMSProp_validation.append(Loss(W,X_validation,y_validation))
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
g=0.0
delta=0.003
iter_num=2000
loss_AdaDelta_validation=[]
loss_AdaDelta_validation.append(Loss(W,X_validation,y_validation))
for i in range(iter_num):
    W,g,delta=SGD_AdaDelta(W,g,delta,X_train,y_train)
    loss_AdaDelta_validation.append(Loss(W,X_validation,y_validation))
```

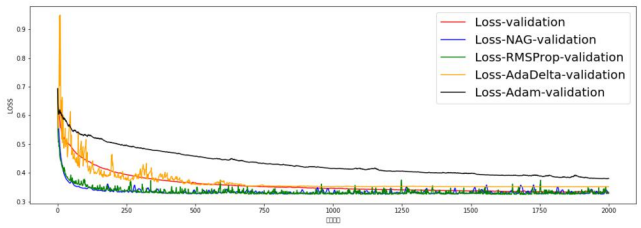
```
W=np.mat(np.zeros(X_train.shape[0])).T
g=0.0
learning_rate=0.1
momentum=np.mat(np.zeros(W.shape))
iter_num=2000

loss_Adam_validation=[]
loss_Adam_validation.append(Loss(W,X_validation,y_validation))

for i in range(iter_num):
    W,m,g=SGD_Adam(W,momentum,g,X_train,y_train,learning_rate,i+1)

    loss_Adam_validation.append(Loss(W,X_validation,y_validation))
```

```
plt.figure(figsize=(18,6))
plt.plot(loss_validation,color='red',label='Loss-validation')
plt.plot(loss_NAG_validation,color='blue',label='Loss-NAG-validation')
plt.plot(loss_RMSProp_validation,color='green',label='Loss-RMSProp-validation')
plt.plot(loss_AdaDelta_validation,color='orange',label='Loss-AdaDelta-validation')
plt.plot(loss_Adam_validation,color='black',label='Loss-Adam-validation')
plt.xlabel('迭代次数')
plt.ylabel('LOSS')
plt.legend(fontsize=20)
plt.show()
```



ClassificationExperiment：

```
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.datasets import load_svmlight_file
```

```
data=load_svmlight_file('a9a.txt')
X_train=data[0].todense()
y_train=data[1]
```

```
data=load_svmlight_file('a9a.t')
X_validation=data[0].todense()
y_validation=data[1]
```

```
X_train=X_train.T
x_0=np.ones(X_train.shape[1])
X_train=np.row_stack((X_train,x_0))
y_train=np.mat(y_train)

X_validation=X_validation.T
a=np.zeros(X_validation.shape[1])
x_0=np.ones(X_validation.shape[1])
X_validation=np.row_stack((X_validation,a))
X_validation=np.row_stack((X_validation,x_0))
y_validation=np.mat(y_validation)
```

```
def Loss(w,x,y):
    loss=0
    for i in range(x.shape[1]):
        if (1-y[0,i]*(w.T*x[:,i]))>0:
            loss+=1-y[0,i]*(w.T*x[:,i])
    losl=loss/float(y.shape[1])
    loss+=(w.T*w)/2.0
    return loss[0,0]
```

```
def SGD(w_current,x,y,learning_rate):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current
    new_w=w_current-learning_rate*gradient
    return new_w
```

```
def SGD_NAG(w_current,momentum,x,y,learning_rate):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current-momentum
    momentum=0.9*momentum+learning_rate*gradient

    new_w=w_current-momentum
    return new_w,momentum
```

```
def SGD_RMSProp(w_current,g,x,y,learning_rate):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current
    g=0.9*g+0.1*(gradient.T*gradient)[0,0]
    new_w=w_current-(learning_rate/(np.sqrt(g+1e-8)))*gradient
    return new_w,g
```

```
def SGD_AdaDelta(w_current,g,delta,x,y):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current
    g=0.95*g+0.05*(gradient.T*gradient)[0,0]

    step_length=(np.sqrt(delta+1e-8)/np.sqrt(g+1e-8))*gradient
    new_w=w_current-step_length
    delta=0.95*delta+0.05*(step_length.T*step_length)[0,0]
    return new_w,g,delta
```

```
def SGD_Adam(w_current,m,g,x,y,learning_rate,iter_num):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current
    m=0.9*m+0.1*gradient
    g=0.999*g+0.001*gradient.T*gradient
    alpha=learning_rate*np.sqrt(1-0.999**iter_num)/(1-0.9**iter_num)
    new_w=w_current-alpha*m/np.sqrt(g+1e-8)
    return new_w,m,g
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
iter_num=1000
learning_rate=0.001

loss_validation=[]
loss_validation.append(Loss(W,X_validation,y_validation))
for i in range(iter_num):
    W=SGD(W,X_train,y_train,learning_rate)
    if (i+1)%10==0:
        loss_validation.append(Loss(W,X_validation,y_validation))
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
momentum=np.mat(np.zeros(X_train.shape[0])).T
iter_num=1000
learning_rate=0.001

loss_NAG_validation=[]
loss_NAG_validation.append(Loss(W,X_validation,y_validation))
for i in range(iter_num):
    W,momentum=SGD_NAG(W,momentum,X_train,y_train,learning_rate)
    if (i+1)%10==0:
        loss_NAG_validation.append(Loss(W,X_validation,y_validation))
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
g=0
iter_num=1000
learning_rate=0.1

loss_RMSProp_validation=[]
loss_RMSProp_validation.append(Loss(W,X_validation,y_validation))
for i in range(iter_num):
    W,g=SGD_RMSProp(W,g,X_train,y_train,learning_rate)
    if (i+1)%10==0:
        loss_RMSProp_validation.append(Loss(W,X_validation,y_validation))
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
g=0.0
delta=0.003
iter_num=1000
loss_AdaDelta_validation=[]
loss_AdaDelta_validation.append(Loss(W,X_validation,y_validation))
```

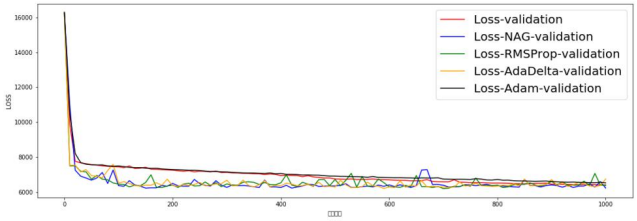
```
W=np.mat(np.zeros(X_train.shape[0])).T
g=0.0
delta=0.003
iter_num=1000
loss_AdaDelta_validation=[]
loss_AdaDelta_validation.append(Loss(W,X_validation,y_validation))
for i in range(iter_num):
    W,g,delta=SGD_AdaDelta(W,g,delta,X_train,y_train)
    if (i+1)%10==0:
        loss_AdaDelta_validation.append(Loss(W,X_validation,y_validation))
```

```
W=np.mat(np.zeros(X_train.shape[0])).T
g=0.0
learning_rate=0.1
momentum=np.mat(np.zeros(W.shape))
iter_num=1000

loss_Adam_validation=[]
loss_Adam_validation.append(Loss(W,X_validation,y_validation))

for i in range(iter_num):
    W,m,g=SGD_Adam(W,momentum,g,X_train,y_train,learning_rate,i+1)
    if (i+1)%10==0:
        loss_Adam_validation.append(Loss(W,X_validation,y_validation))
```

```
plt.figure(figsize=(18,6))
plt.plot([i*10 for i in range(101)],loss_validation,color='red',label='Loss-validation')
plt.plot([i*10 for i in range(101)],loss_NAG_validation,color='blue',label='Loss-NAG-validation')
plt.plot([i*10 for i in range(101)],loss_RMSProp_validation,color='green',label='Loss-RMSProp-validation')
plt.plot([i*10 for i in range(101)],loss_AdaDelta_validation,color='orange',label='Loss-AdaDelta-validation')
plt.plot([i*10 for i in range(101)],loss_Adam_validation,color='black',label='Loss-Adam-validation')
plt.xlabel('迭代次数')
plt.ylabel('LOSS')
plt.legend(fontsize=20)
plt.show()
```



IV. CONCLUSION

SGD run quicker than GD. SGD choose one example to compute the gradient to represent the whole gradient. So SGD can improve the algroithm speed. But learning curve using SGD will get some fluctuations, because we may choose the noise as training data. To avoid the noise, we randomly choose a part of data to compute the gradient.