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:
Guobin Wu

Supervisor:
Qingyao Wu

Student ID：
201530613030

Grade:
Undergraduate

December 11, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

Abstract—This report tends to illustrate the experiments we have done about logistic regression, linear classification and stochastic gradient descent(SGD), with respect to understanding and comprehending the core of this mentioned topics.

I. Introduction

Logistic regression and linear classification are the two of most fundamental machine learning models. Additionally, gradient decent(GD) is one of the most widely-used optimizing methods to reach local optimal solution. Stochastic gradient descent(SGD), an improved version of traditional GD, accelerates the process reaching the solution. This experiment aims to compare GD to SGD, to help understanding the differences and relations between them. What's more, we also compare logistic regression to linear classification, figuring out what is and is not similar to each other. Lastly, we practice SVM on larger data to have a better command of its principles.

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

A. Data Set

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

B. Implementation

1) Logistic regression:

Loss function of logistic regression is as follows.

$$L(\theta) = -\frac{1}{m}\sum_{i=0}^{n}\left(y_i \log\left(h_\theta(X_i)\right) + (1-y_i)\log\left(1-\log\left(h_\theta(X_i)\right)\right)\right)$$

where $h_\theta(X) = \frac{1}{1+e^{-\theta^T \cdot X}}$

Compute the gradient of L(θ), we obtain,

$$\frac{\partial L}{\partial \theta} = \frac{1}{n}\sum_{i=0}^{n} X_i\left(h_\theta(X) - y_i\right)$$

Having defined loss function and its gradient, we can use SGD to get the final solution. We use four method respectively to reach the local optimal solution including NAG, RMSProp, AdaDelta and Adam. The super parameters we select are as follows.

NAG	learning rate	0.002
	gamma	0.9
	iteration	2000
RMSProp	learning rate	0.005
	gamma	0.9
	epsilon	1e-8
	iteration	2000
AdaDelta	learning rate	0.005
	gamma	0.95

	epsilon	1e-8
	iteration	2000
Adam	learning rate	0.005
	beta1	0.9
	beta2	0.999
	epsilon	1e-8
	iteration	2000

Next, we program to implement the above methods. The following are the screenshots of source code.

In [3]:
```
import numpy as np
from sklearn.datasets import load_svmlight_file
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [4]:
```
data=load_svmlight_file('a9a.txt')
X_train=data[0].todense()
y_train=data[1]
```

In [5]:
```
data=load_svmlight_file('a9a.t')
X_validation=data[0].todense()
y_validation=data[1]
```

In [6]:
```
#x为列向量
X_train=X_train.T
#x0=1
x_0=np.ones(X_train.shape[1])
X_train=np.row_stack((X_train,x_0))
#y为行向量
y_train=np.mat(y_train)

#x为列向量
X_validation=X_validation.T
a=np.zeros(X_validation.shape[1])
#x0=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为行向量
y_validation=np.mat(y_validation)
```

In [7]:
```
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
```

In [8]:
```
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])
```

In [9]:
```
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
```

In [10]:
```
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*
    momentum=0.9*momentum+learning_rate*gradient
    w=w-momentum
    return w,momentum
```

In [11]:
```
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
```

In [12]:
```
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
```

In [ ]:
```
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
```

In [ ]:
```
#------------------没有优化------------
#初始化参数
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))
```

In [ ]:
```
#-----------------NAG优化
#初始化参数
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))
```

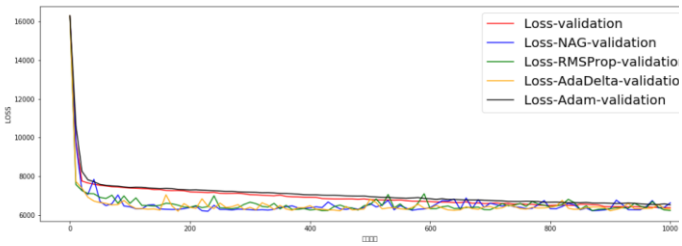
In [ ]:
```
#------------RMSProp优化
#初始化参数
W=np.mat(np.zeros(X_train.shape[0])).T
g=0
iter_num=2000
learning_rate=0.1

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))
```

In [ ]:
```
#-------------AdaDelta
#初始化参数
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))
```

In [ ]:
```
#--------------------Adam优化
#初始化参数
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))
```

In [ ]:
```
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()
```

We get the following loss graphs as results after running the program.



From the graph we find that AdaDelta reaches the local optimal solution fastest, but it also has obvious vibration. By contrast, NAG is slower than AdaDelta with a smoother curve. RMSProp and Adam are closely overlapped, which are slower than AdaDelta and faster than NAG. Four methods reaches optimal solution far faster than traditional GD.

2) Linear classification:

Loss function of logistic regression is as follows.

$$L(\theta) = \frac{1}{2n}\sum_{i=0}^{n}(y_i - h_\theta(X_i))^2 \quad \text{where} \quad h_\theta(X) = \sum_{i=0}^{n}\theta_i X_i$$

Compute the gradient of L(θ), we obtain,

$$\frac{\partial L}{\partial \theta} = \frac{1}{n}\sum_{i=0}^{n}(y_i - h_\theta(X_i)) \cdot X_i$$

Having defined loss function and its gradient, we can use SGD to get the final solution. We use four method respectively to reach the local optimal solution including NAG, RMSProp, AdaDelta and Adam. The super parameters we select are as follows.

NAG	learning rate	0.005
	gamma	0.9
	iteration	3000
RMSProp	learning rate	0.005
	gamma	0.9
	epsilon	1e-8
	iteration	3000
AdaDelta	learning rate	0.005
	gamma	0.9
	epsilon	1e-8
	iteration	3000
Adam	learning rate	0.005
	beta1	0.9
	beta2	0.999
	epsilon	1e-8
	iteration	3000

Next, we program to implement the above methods. The following are the screenshots of source code.

From the graph we find that NAG reaches the local optimal solution fastest, but it also has obvious vibration. By contrast, AdaDelta is slower than NAG with a smoother curve. RMSProp and Adam are closely overlapped, which are slower than NAG and faster than AdaDelta. Four methods reaches optimal solution far faster than traditional GD.

IV. CONCLUSION

SGD improves traditional GD with a faster converging speed and considerable result. Four specific methods are different improvement of plain SGD with respective advantages. Logistic regression and linear classification both solves classification problem to predict new samples. The major difference between them is the way they evaluate final super plane. Logistic regression tries to find a super plane which makes all sample points get away from it, whereas linear classification finds a plane defined by so-called ‘support vectors’. This experiment makes comparisons between GD and SGD, logistic regression and linear classification, evaluates four SGD methods in this two model and analyzes the experiment results.

In [1]:
```
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.datasets import load_svmlight_file
```

In [2]:
```
data=load_svmlight_file('a9a.txt')
X_train=data[0].todense()
y_train=data[1]
```

In [3]:
```
data=load_svmlight_file('a9a.t')
X_validation=data[0].todense()
y_validation=data[1]
```

In [4]:
```
#x为列向量
X_train=X_train.T
#x0=1
x_0=np.ones(X_train.shape[1])
X_train=np.row_stack((X_train,x_0))
#y为行向量
y_train=np.mat(y_train)

#x为列向量
X_validation=X_validation.T
a=np.zeros(X_validation.shape[1])
#x0=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为行向量
y_validation=np.mat(y_validation)
```

In [5]:
```
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]
```

In [6]:
```
#没有优化
def SGD(w_current,x,y,learning_rate):
    gradient=np.mat(np.zeros(w_current.shape))
    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
```

In [7]:
```
#NAG优化
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
```

In [8]:
```
#RMSProp优化
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
```

In [9]:
```
#AdaDelta优化
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
```

In [10]:
```
#Adam优化
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
```

In [11]:
```
#-------------------没有优化-------------
#初始化参数
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))
```

In [12]:
```
#-----------------NAG优化
#初始化参数
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))
```

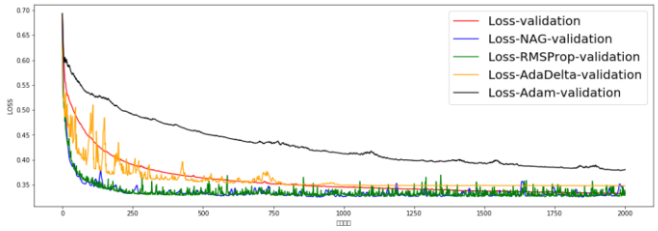
In [13]:
```
#------------RMSProp优化
#初始化参数
W=np.mat(np.zeros(X_train.shape[0])).T
g=0
iter_num=1000
learning_rate=0.1

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))
```

In [14]:
```
#--------------AdaDelta
#初始化参数
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))
```

In [15]:
```
#--------------------Adam优化
#初始化参数
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))
```

In [16]:
```
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()
```

We get the following loss graphs as results after running the program.


Loss-validation
Loss-NAG-validation
Loss-RMSProp-validation
Loss-AdaDelta-validation
Loss-Adam-validation
LOSS
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
250
500
750
1000
1250
1500
1750
2000