

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **李恺哲**

**Student ID 201530611982**

**E-mail 623332032@qq.com**

**Tutor**   **吴庆耀**

**Date submitted** **2017.12 .04**

**1. Topic:** Logistic Regression, Linear Classification and Stochastic Gradient Descent

**2. Time: 2017.12.04**

**3. Reporter:李恺哲**

**4. Purposes:** Compare and understand the difference between gradient descent and stochastic gradient descent.

Compare and understand the differences and relationships between Logistic regression and linear classification.

Further understand the principles of SVM and practice on larger data.

**5. Data sets and data analysis:** 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.

**6. Experimental steps:**

*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 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
7. Repeat step 4 to 6 for several times

*Linear Classification and Stochastic Gradient Descent*

1. Load the training set and validation set.
2. Initialize SVM model parameters, you can consider initializing zeros
3. Select the loss function and calculate its derivation
4. Calculate gradient 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.
7. Repeat step 4 to 6 for several times
8. **Code: regression:**

from numpy import \*

import matplotlib.pyplot as plt

from sklearn.datasets import load\_svmlight\_file

import numpy as np

from sklearn.cross\_validation import train\_test\_split

t\_X,t\_y=load\_svmlight\_file('a9a.t') #读取数据

X\_train,X\_test, y\_train, y\_test =train\_test\_split(t\_X,t\_y,test\_size=0.4, random\_state=1)#分割数据

def sigmoid(inX):

return 1.0/(1+exp(-inX))

def Loss(y,output):

loss = +(y \* np.log(output)).sum(axis=1)

return loss

lossList=[]

def trainLogRegres(train\_x,train\_y,opts):

numSamples,numFeatures=shape(train\_x)

iter=opts['iter']

alpha=opts['alpha']

weights=zeros((numFeatures,1))

Gt=zeros((numFeatures,1))

deltat = zeros((numFeatures, 1))

mt = zeros((numFeatures, 1))

for t in range(iter):

if opts['optimizeType']=='stocGradDecent':#随机梯度

i=random.randint(0,numSamples)

output=sigmoid(train\_x[i,:]\*weights)

error=train\_y[i]-output

loss=Loss(train\_y,output)

lossList.append(loss)

print (loss)

g = train\_x[i,:].transpose() \* error

weights = weights - alpha \* g

elif opts['optimizeType']=='NAG': #NGA

yita=opts['yita']

vt = zeros(train\_y.shape)

i = random.randint(0, numFeatures)

output = sigmoid(train\_x[i, :] \* weights-yita\*vt)

error = train\_y[i] - output

loss = Loss(train\_y, output)

lossList.append(loss)

print(loss)

vt=yita\*vt+alpha \* train\_x[i, :].transpose() \* error

weights = weights - vt

elif opts['optimizeType']=='RMSProp': #RMSProp

yita=opts['yita']

i = random.randint(0, numFeatures)

output = sigmoid(train\_x[i, :] \* weights)

error = train\_y[i] - output

g = train\_x[i, :].transpose() \* error

Gt=yita\*Gt+(1-yita)\*multiply(g,g)

loss = Loss(train\_y, output)

lossList.append(loss)

print (loss)

weights=weights-multiply(alpha/np.sqrt(Gt+1e-8),g)

elif opts['optimizeType']=='AdamDelta': #AdamDelta

yita = opts['yita']

i=random.randint(0,numFeatures)

output=sigmoid(train\_x[i,:]\*weights)

error=train\_y[i]-output

g = train\_x[i,:].transpose() \* error

Gt = Gt + (1 - yita) \* multiply(g, g)

deltaWeight=-multiply((np.sqrt(deltat+1e-6)/np.sqrt(Gt+1e-6)),g)

loss = Loss(train\_y, output)

lossList.append(loss)

print(loss)

weights=weights+deltaWeight

deltat=yita\*deltat+(1-yita)\*multiply(deltaWeight,deltaWeight)

elif opts['optimizeType']=='Adam': #Adam

yita = opts['yita']

beta=opts['beta']

alpha=opts['alpha']

beta1=0.9

i=random.randint(0,numFeatures)

output=sigmoid(train\_x[i,:]\*weights)

loss = Loss(train\_y, output)

lossList.append(loss)

print(loss)

error=train\_y[i]-output

g = train\_x[i,:].transpose() \* error

mt=beta\*mt+(1-beta)\*g

Gt = Gt + (1 - yita) \* multiply(g, g)

alpha=alpha\*math.sqrt(1-yita\*\*t)/(1-beta1\*\*(t+1))

weights=weights-alpha\*mt/np.sqrt(Gt+1e-8)

#opts={'alpha':0.01,'iter':1000,'optimizeType':'stocGradDecent','yita':0.95,'beta':0.9}

#trainLogRegres(X\_train,y\_train,opts)

#opts={'alpha':0.01,'iter':1000,'optimizeType':'NAG','yita':0.95,'beta':0.9}

#trainLogRegres(X\_train,y\_train,opts)

#opts={'alpha':0.001,'iter':1000,'optimizeType':'RMSProp','yita':0.9,'beta':0.9}

#trainLogRegres(X\_train,y\_train,opts)

opts={'alpha':0.01,'iter':10000,'optimizeType':'AdamDelta','yita':0.9,'beta':0.9}

trainLogRegres(X\_train,y\_train,opts)

#opts={'alpha':0.001,'iter':1000,'optimizeType':'Adam','yita':0.999,'beta':0.9}

#trainLogRegres(X\_train,y\_train,opts)

iter=list(range(0,10000))

plt.plot(iter,lossList)

plt.show()

**8. The initialization method of model parameters:** all initialized to zero

**9. The selected loss function and its derivatives:** **loss = +(y \* np.log(output)).sum(axis=1)**

**10. Experimental results and curve:**(Fill in this content for various methods of gradient descent respectively)

## Hyper-parameter selection:

'alpha':0.01,'iter':1000,'optimizeType':'stocGradDecent','yita':0.95,'beta':0.9

'alpha':0.01,'iter':1000,'optimizeType':'NAG','yita':0.95,'beta':0.9

'alpha':0.001,'iter':1000,'optimizeType':'RMSProp','yita':0.9,'beta':0.9

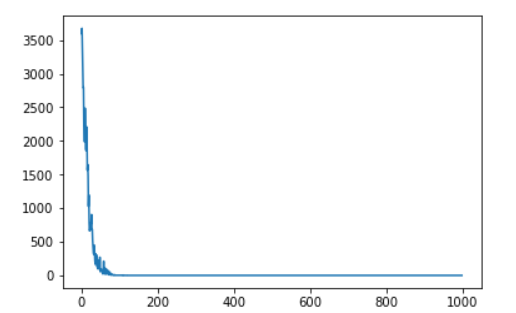
'alpha':0.01,'iter':10000,'optimizeType':'AdamDelta','yita':0.9,'beta':0.9

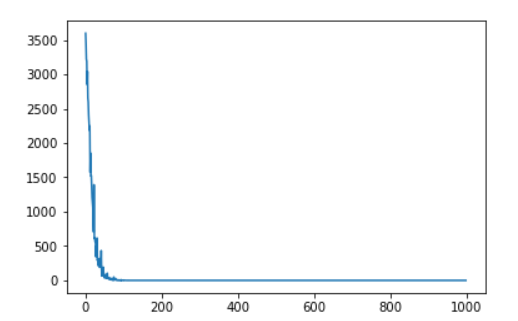
'alpha':0.001,'iter':1000,'optimizeType':'Adam','yita':0.999,'beta':0.9

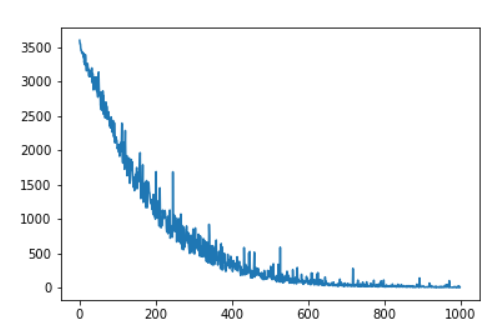
## Predicted Results (Best Results):

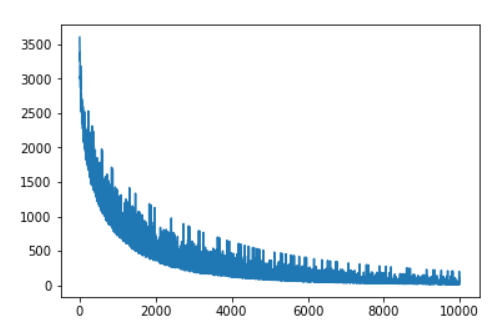
After 1000 times of iter, the result is that stoc and nag works fast, while adamdelta works really slow, and rmsprop and adam on a middle speed.

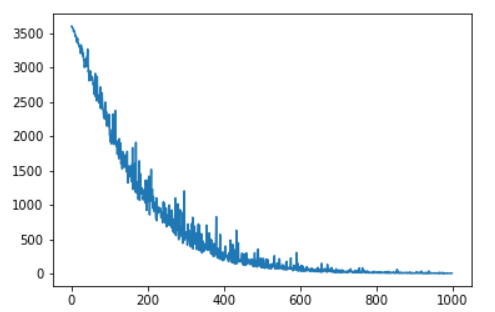
## Loss curve:

Sotc: 

NAG: 

RMSProp: 

AdamDelta：

Adam: 

**11. Results analysis:** The NAG has the best result, and the stoc works better than the others, while in those complex methods, AdamDelta works so slow that I have to use 10000 times of iter to low its loss, The Adam has a better work in those complex works, maybe it can work best to getting to the right result.

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

Similarity: logistic regression and linearsvm are both kind of linear classification methods. They are both trying to make those points with more weight which are making more efforts to the classification.

Difference: logistic regression uses log loss while linearsvm uses hinge loss as their loss function. The logistic regression is easy to understand so that easy to realize, oppositely, linear svm is easy to calculate, and its model is simple.

**13. Summary:**

After this lab, I know about some methods of gradient descent, and do some tests about them. The result shows that even though the basic method of gradient descent has a faster speed to get closed to the lower loss, the other methods show their power to get a better result. The method Adam shows a great ability to handle the problem. To realize those methods, I referred to several blogs, from which I learn deeper about their theory.

**Code: SVM**

**8. The initialization method of model parameters:** all initialized to zero

**9. The selected loss function and its derivatives:** *Li*=∑*j*≠*yi*max(0,*wTjxi*−*wTyixi*+Δ)

**10. Experimental results and curve:**(Fill in this content for various methods of gradient descent respectively)

## Hyper-parameter selection:

'alpha':0.01,'iter':300,'optimizeType':'stocGradDecent','yita':0.95,'beta':0.9

'alpha':0.01,'iter':300,'optimizeType':'NAG','yita':0.95,'beta':0.9

'alpha':0.001,'iter':10000,'optimizeType':'RMSProp','yita':0.9,'beta':0.9

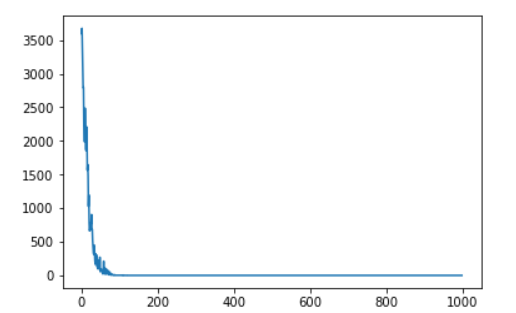
'alpha':0.01,'iter':10000,'optimizeType':'AdamDelta','yita':0.9,'beta':0.9

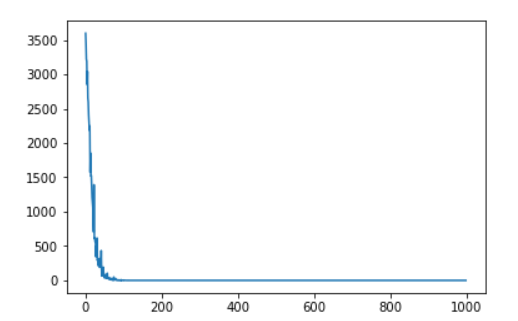
'alpha':0.001,'iter':300,'optimizeType':'Adam','yita':0.999,'beta':0.9

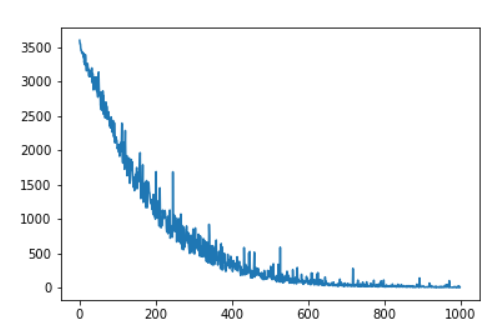
## Predicted Results (Best Results):

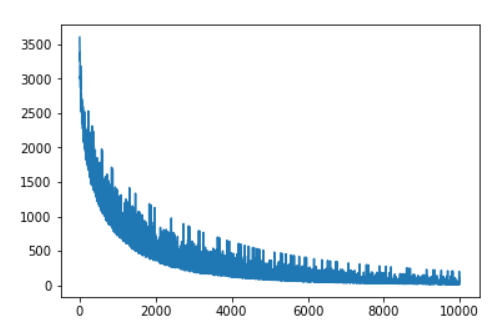
After 1000 times of iter, the result is that

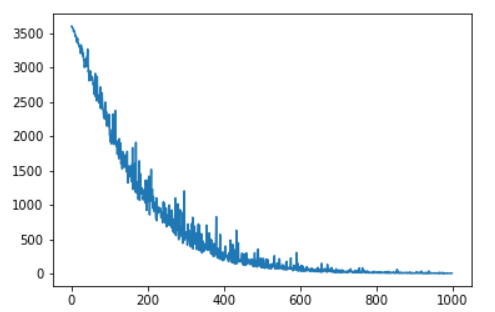
## Loss curve:

Sotc: 

NAG: 

RMSProp: 

AdamDelta：

Adam: 

**11. Results analysis:** To get the similar result as the basic method, I have to run the AdamDelta in 10000 times, amazingly, the AdamDelta method shows similar result as RMSProp in 10000 times, which is not like the result before, it shows that this method is not as I think before that bad, even its costs 2 or 3 more seconds than other methods.

**13. Summary:**

The complex methods shows great accuracy in linear classification after running for more than 500 times, better than the simple methods.

But Adam works fast and good in 200 times of iters.