

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **郑孟丹**

**Student ID 20153061379**

**E-mail 1228804246@qq.com**

**Tutor**   **谭明奎**

**Date submitted** **2017.12.02**

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

**2. Time:** 2017.12.02

**3. Reporter:** 郑孟丹

**4. Purposes:**①Compare and understand the difference between gradient descent and stochastic gradient descent.

②Compare and understand the differences and relationships between Logistic regression and linear classification.

③Further 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.

**6. Experimental steps:**Logistic Regression and Stochastic Gradient Descent

①Load the training set and validation set.

②Initalize logistic regression model parameters, you can consider initalizing zeros, random numbers or normal distribution.

③Select the loss function and calculate its derivation, find more detail in PPT.

④Calculate gradient G toward loss function from partial samples.

⑤Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).

⑥Select 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.

⑦Repeate 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

①Load the training set and validation set.

②Initalize SVM model parameters, you can consider initalizing zeros, random numbers or normal distribution.

③Select the loss function and calculate its derivation, find more detail in PPT.

④Calculate gradient G toward loss function from partial samples.

⑤Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).

⑥Select 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 .

⑦Repeate 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.

**7. Code:  
1 Logistic Regression and Stochastic Gradient Descent**  
  
*# coding: utf-8  
  
# In[1]:***from** sklearn.externals.joblib **import** Memory  
**from** sklearn.datasets **import** load\_svmlight\_file  
**from** sklearn **import** datasets  
**from** sklearn **import** model\_selection  
**import** numpy **as** np  
**import** matplotlib **as** mp  
**import** jupyter  
**import** matplotlib.pyplot **as** plt  
**import** math  
**import** random  
get\_ipython().magic(**'matplotlib inline'**)  
  
  
*# In[2]:*mem = Memory(**"./mycache"**)  
  
@mem.cache  
**def** get\_data(mysvmlightfile):  
 data = load\_svmlight\_file(mysvmlightfile)  
 **return** data[0], data[1]  
  
  
*# In[3]:  
  
#定义logistic function***def** Logistic(a):  
 **return** 1/(1+pow(np.e,-a))  
  
  
*# In[4]:  
  
#读取数据集*x\_train,y\_train=get\_data(**'a9a.txt'**)  
x\_test,y\_test=get\_data(**'a9a.t'**)  
  
y\_train=np.mat(y\_train).T  
y\_test=np.mat(y\_test).T  
  
  
*# In[5]:  
  
#增加输入x矩阵一列全为1是的线性模型满足y=w\*x*x\_train=x\_train.todense()  
one=np.ones(x\_train.shape[0])  
x\_train=np.column\_stack((x\_train,one))  
x\_test=x\_test.todense()  
  
*#测试集x值补全一列全为0*zero=np.zeros(x\_test.shape[0])  
x\_test=np.column\_stack((x\_test,zero))  
  
one=np.ones(x\_test.shape[0])  
x\_test=np.column\_stack((x\_test,one))  
  
  
*# In[6]:  
  
#参数随机初始化*w=np.random.random(size=(1,x\_train.shape[1]))  
**print**(w)  
w=np.mat(w)  
w=w.T  
  
  
*# In[7]:  
  
#sgd*sgd\_rate=0.01  
sgd\_w=w  
  
*#momentum*momentum\_gamma=0.9  
momentum\_rate=0.01  
momentum\_v = 0  
momentum\_w=w  
  
*#NAG*nag\_gamma=0.9  
nag\_rate=0.1  
nag\_v=0  
nag\_w=w  
  
*#adagrad*adagrad\_rate=0.01  
adagrad\_G=0  
adagrad\_epsilon=math.pow(np.e,-6)  
adagrad\_w=w  
  
*#rmsprop*rmsprop\_rate=0.05  
rmsprop\_G=0  
rmsprop\_epsilon=math.pow(np.e,-6)  
rmsprop\_gamma=0.9  
rmsprop\_w=w  
  
*#adadelta*adadelta\_rate=0.1  
adadelta\_G=0  
adadelta\_epsilon=math.pow(np.e,-6)  
adadelta\_gamma=0.5  
adadelta\_t=0  
adadelta\_w=w  
  
*#adam*adam\_rate=0.1  
adam\_G=0  
adam\_epsilon=math.pow(np.e,-6)  
adam\_gamma=0.99  
adam\_t=0  
adam\_m=0  
adam\_beta=0.9  
adam\_w=w  
  
  
*# In[8]:  
  
#循环次数*time=0  
  
*#画图存数据的列表*time\_list=[]  
sgd\_L\_list=[]  
momentum\_L\_list=[]  
nag\_L\_list=[]  
adagrad\_L\_list=[]  
rmsprop\_L\_list=[]  
adadelta\_L\_list=[]  
adam\_L\_list=[]  
  
  
*# loss function：jw=log(1+e\*\*(-yi\*wt\*xi))+0.5\*w.T\*w，grad\_w= w-（yi\*xi）/(1+e\*\*(-yi\*wt\*xi))  
  
# In[9]:***while** time<20:  
 time=time+1  
  
 *#随机取十个样本计算梯度* Jgrad = 0  
 **for** i **in** random.sample(range(x\_train.shape[0]),10):  
 Jgrad=Jgrad-(y\_train[i]\*x\_train[i]).T/(1+pow(np.e,(np.array(y\_train[i]\*x\_train[i]\*w)[0][0])))  
 Jgrad=Jgrad/10  
   
 *#sgd* sgd\_w=(1-sgd\_rate)\*sgd\_w-sgd\_rate\*Jgrad  
 sgd\_L = 0  
 **for** i **in** range(x\_test.shape[0]):  
 sgd\_L = sgd\_L+math.log(1/Logistic(np.array(y\_test[i]\*x\_test[i]\*sgd\_w)[0][0]))  
 sgd\_L=sgd\_L/x\_test.shape[0]+0.5\*sgd\_w.T\*sgd\_w  
 **print**(**"epoch:"**, time)  
 **print**(**"sgd\_test\_loss:"**, sgd\_L)  
 time\_list.append(time)  
 sgd\_L\_list.append(np.array(sgd\_L)[0][0])  
  
 *#momentum* momentum\_v = momentum\_gamma \* momentum\_v + momentum\_rate \* Jgrad  
 momentum\_w = momentum\_w - momentum\_v  
 momentum\_L = 0  
 **for** i **in** range(x\_test.shape[0]):  
 momentum\_L = momentum\_L + math.log(1 / Logistic(np.array(y\_test[i] \* x\_test[i] \* momentum\_w)[0][0]))  
 momentum\_L = momentum\_L / x\_test.shape[0]+0.5\*momentum\_w.T\*momentum\_w  
 **print**(**"momentum\_test\_loss:"**, momentum\_L)  
 momentum\_L\_list.append(np.array(momentum\_L)[0][0])  
  
 *#nag* nag\_v = 0  
 nag\_\_Jgrad=0  
 **for** i **in** random.sample(range(x\_train.shape[0]), 10):  
 nag\_\_Jgrad = nag\_\_Jgrad-(y\_train[i] \* x\_train[i]).T / (1 + pow(np.e, (np.array(y\_train[i] \* x\_train[i] \* (w - nag\_gamma \* nag\_v))[0][0])))  
 nag\_\_Jgrad=nag\_\_Jgrad/10  
 nag\_v = nag\_gamma \* nag\_v + nag\_rate \* nag\_\_Jgrad  
 nag\_w = nag\_w - nag\_v  
 nag\_L = 0  
 **for** i **in** range(x\_test.shape[0]):  
 nag\_L = nag\_L + math.log(1 / Logistic(np.array(y\_test[i] \* x\_test[i] \* nag\_w)[0][0]))  
 nag\_L = nag\_L / x\_test.shape[0]+0.5\*nag\_w.T\*nag\_w  
 **print**(**"nag\_test\_loss:"**, nag\_L)  
 nag\_L\_list.append(np.array(nag\_L)[0][0])  
  
 *#adagrad* adagrad\_Jgrad=Jgrad/10  
 adagrad\_G = adagrad\_G + np.array(adagrad\_Jgrad) \* np.array(adagrad\_Jgrad)  
 adagrad\_w = adagrad\_w - (adagrad\_rate / np.sqrt(adagrad\_epsilon + adagrad\_G)) \* np.array(Jgrad)  
 adagrad\_L = 0  
 **for** i **in** range(x\_test.shape[0]):  
 adagrad\_L = adagrad\_L + math.log(1 / Logistic(np.array(y\_test[i] \* x\_test[i] \* adagrad\_w)[0][0]))  
 adagrad\_L = adagrad\_L / x\_test.shape[0]+0.5\*adagrad\_w.T\*adagrad\_w  
 **print**(**"adagrad\_test\_loss:"**, adagrad\_L)  
 adagrad\_L\_list.append(np.array(adagrad\_L)[0][0])  
  
 *#rmsprop* rmsprop\_G = rmsprop\_gamma\*rmsprop\_G +(1-rmsprop\_gamma)\*np.array(Jgrad) \* np.array(Jgrad)  
 rmsprop\_w = rmsprop\_w - (rmsprop\_rate / np.sqrt(rmsprop\_epsilon + rmsprop\_G)) \* np.array(Jgrad)  
 rmsprop\_L = 0  
 **for** i **in** range(x\_test.shape[0]):  
 rmsprop\_L = rmsprop\_L + math.log(1 / Logistic(np.array(y\_test[i] \* x\_test[i] \* rmsprop\_w)[0][0]))  
 rmsprop\_L =rmsprop\_L / x\_test.shape[0]+0.5\*rmsprop\_w.T\*rmsprop\_w  
 **print**(**"rmsprop\_test\_loss:"**,rmsprop\_L)  
 rmsprop\_L\_list.append(np.array(rmsprop\_L)[0][0])  
  
 *#adadelta* adadelta\_G = adadelta\_gamma \* adadelta\_G + (1 - adadelta\_gamma) \* np.array(Jgrad) \* np.array(Jgrad)  
 adadelta\_delta\_w=-np.sqrt(adadelta\_t+adadelta\_epsilon)/np.sqrt(adadelta\_G+adadelta\_epsilon) \* np.array(Jgrad)  
 adadelta\_w = adadelta\_w + adadelta\_delta\_w  
 adadelta\_t=adadelta\_gamma\*adadelta\_t+(1-adadelta\_gamma)\*np.array(adadelta\_delta\_w)\*np.array(adadelta\_delta\_w)  
 adadelta\_L = 0  
 **for** i **in** range(x\_test.shape[0]):  
 adadelta\_L = adadelta\_L + math.log(1 / Logistic(np.array(y\_test[i] \* x\_test[i] \* adadelta\_w)[0][0]))  
 adadelta\_L = adadelta\_L / x\_test.shape[0]+0.5\*adadelta\_w.T\*adadelta\_w  
 **print**(**"adadelta\_test\_loss:"**, adadelta\_L)  
 adadelta\_L\_list.append(np.array(adadelta\_L)[0][0])  
  
 *#adam* adam\_m=adam\_beta\*adam\_m+(1-adam\_beta)\*Jgrad  
 adam\_G = adam\_gamma \* adam\_G + (1 - adam\_gamma) \* np.array(Jgrad) \* np.array(Jgrad)  
 adam\_alpha=adam\_rate\*math.sqrt(1-math.pow(adam\_gamma,time))/(1-math.pow(adam\_beta,time))  
 adam\_w = adam\_w - adam\_alpha\*adam\_m/np.sqrt(adam\_G+adam\_epsilon)  
 adam\_L = 0  
 **for** i **in** range(x\_test.shape[0]):  
 adam\_L = adam\_L + math.log(1 / Logistic(np.array(y\_test[i] \* x\_test[i] \* adam\_w)[0][0]))  
 adam\_L = adam\_L / x\_test.shape[0]+0.5\*adam\_w.T\*adam\_w  
 **print**(**"adam\_test\_loss:"**, adam\_L)  
 adam\_L\_list.append(np.array(adam\_L)[0][0])  
  
  
*# In[10]:  
  
#画图*plt.xlabel(**'epoch'**)  
plt.ylabel(**'loss'**)  
line1=plt.plot(time\_list,sgd\_L\_list)  
line2=plt.plot(time\_list,momentum\_L\_list)  
line3=plt.plot(time\_list,nag\_L\_list)  
line4=plt.plot(time\_list,adagrad\_L\_list)  
line5=plt.plot(time\_list,rmsprop\_L\_list)  
line6=plt.plot(time\_list,adadelta\_L\_list)  
line7=plt.plot(time\_list,adam\_L\_list)  
label = [**"sgd"**,**"momentum"**,**"nag"**,**"adagrad"**,**"rmsprop"**,**"adadelta"**,**"adam"**]  
plt.legend(label, loc = 0, ncol = 7)  
plt.show()  
  
  
*# In[ ]:*

**8\_1. The initialization method of model parameters:**set parameter to random numbers

**9\_1. The selected loss function and its derivatives:**loss function：jw=log(1+e\*\*(-yi\*w.T\*xi))+0.5\*w.T\*w，  
its derivatives：grad\_w= w-（yi\*xi）/(1+e\*\*(-yi\*w.T\*xi))

**10\_1. Experimental results and curve:**  
**NAG：**  
Hyper-parameter selection:

nag\_gamma=0.9  
nag\_rate=0.1  
nag\_v=0

epoch=20

Predicted Results (Best Results):loss=17.20281231

**RMSProp：**  
Hyper-parameter selection:

rmsprop\_rate=0.05  
rmsprop\_G=0  
rmsprop\_epsilon=math.pow(np.e,-6)  
rmsprop\_gamma=0.9

epoch=20

Predicted Results (Best Results):loss=13.31214314

**AdaDelta：**  
Hyper-parameter selection:  
adadelta\_G=0  
adadelta\_epsilon=math.pow(np.e,-6)  
adadelta\_gamma=0.5  
adadelta\_t=0

epoch=20

Predicted Results (Best Results):loss= 14.22180884

**Adam：**  
Hyper-parameter selection:  
adam\_G=0  
adam\_epsilon=math.pow(np.e,-6)  
adam\_gamma=0.99  
adam\_t=0  
adam\_m=0  
adam\_beta=0.9

epoch=20  
Predicted Results (Best Results):loss=14.78705493

## Loss curve:

**11\_1. Results analysis:**nag’s effect is not very good, rmsprop, adadelta, adam’s decline rate is fast

2 Linear Classification and Stochastic Gradient Descent

*# coding: utf-8  
  
# In[21]:***from** sklearn.externals.joblib **import** Memory  
**from** sklearn.datasets **import** load\_svmlight\_file  
**from** sklearn **import** datasets  
**from** sklearn **import** model\_selection  
**import** numpy **as** np  
**import** matplotlib **as** mp  
**import** jupyter  
**import** matplotlib.pyplot **as** plt  
**import** math  
**import** random  
get\_ipython().magic(**'matplotlib inline'**)  
  
  
*# In[22]:*mem = Memory(**"./mycache"**)  
  
@mem.cache  
**def** get\_data(mysvmlightfile):  
 data = load\_svmlight\_file(mysvmlightfile)  
 **return** data[0], data[1]  
  
  
*# In[23]:  
  
#读取数据集*x\_train,y\_train=get\_data(**'a9a.txt'**)  
x\_test,y\_test=get\_data(**'a9a.t'**)  
  
y\_train=np.mat(y\_train).T  
y\_test=np.mat(y\_test).T  
  
x\_train=x\_train.todense()  
one=np.ones(x\_train.shape[0])  
x\_train=np.column\_stack((x\_train,one))  
x\_test=x\_test.todense()  
zero=np.zeros(x\_test.shape[0])  
x\_test=np.column\_stack((x\_test,zero))  
one=np.ones(x\_test.shape[0])  
x\_test=np.column\_stack((x\_test,one))  
  
  
*# In[24]:  
  
#参数全零初始化*w=np.zeros(x\_train.shape[1])  
w=np.mat(w)  
w=w.T  
  
  
*# In[25]:  
  
#sgd*sgd\_rate=0.0008  
sgd\_w=w  
  
  
*#momentum*momentum\_gamma\_w=0.9  
momentum\_rate=0.0005  
momentum\_v\_w = 0  
momentum\_gamma\_b=0.9  
momentum\_w=w  
  
  
*#NAG*nag\_gamma\_w=0.9  
nag\_rate=0.0005  
nag\_v\_w = 0  
nag\_w=w  
  
*#adagred*adagrad\_rate=0.02  
adagrad\_G\_w=0  
adagrad\_epsilon=math.pow(np.e,-6)  
adagrad\_w=w  
  
*#rmsprop*rmsprop\_rate=0.007  
rmsprop\_G\_w=0  
rmsprop\_epsilon=math.pow(np.e,-6)  
rmsprop\_gamma\_w=0.9  
rmsprop\_w=w  
  
*#adadelta*adadelta\_G\_w=0  
adadelta\_epsilon=math.pow(np.e,-6)  
adadelta\_gamma\_w=0.09  
adadelta\_t\_w=0  
adadelta\_w=w  
  
*#adam*adam\_rate=0.01  
adam\_G\_w=0  
adam\_epsilon=math.pow(np.e,-6)  
adam\_gamma\_w=0.99  
adam\_w=w  
adam\_t\_w=0  
adam\_m\_w=0  
adam\_beta\_w=0.9  
  
  
*# In[26]:  
  
#循环次数*time=0  
c=0.9  
  
  
*# In[27]:  
  
#画图存数据的列表*time\_list=[]  
sgd\_L\_list=[]  
momentum\_L\_list=[]  
nag\_L\_list=[]  
adagrad\_L\_list=[]  
rmsprop\_L\_list=[]  
adadelta\_L\_list=[]  
adam\_L\_list=[]  
  
  
*# In[28]:***while** time<20:  
 time = time + 1  
   
 *#sgd* sgd\_grad\_w =0  
 sgd\_L\_test =0  
 **for** i **in** random.sample(range(x\_train.shape[0]),10):  
 **if**(1-(y\_train[i]\*(x\_train[i]\*sgd\_w))>0):  
 sgd\_grad\_w=sgd\_grad\_w+sgd\_w-c\*(y\_train[i]\*x\_train[i]).T  
 **else**:  
 sgd\_grad\_w=sgd\_grad\_w+sgd\_w  
 sgd\_w=sgd\_w-sgd\_rate\*sgd\_grad\_w  
 **for** i **in** range(x\_test.shape[0]):  
 **if**(1-y\_test[i]\*(x\_test[i]\*sgd\_w)>0):  
 sgd\_L\_test=sgd\_L\_test+(1-y\_test[i]\*(x\_test[i]\*sgd\_w))  
 sgd\_L\_test=0.5\*sgd\_w.T\*sgd\_w+c\*sgd\_L\_test  
 sgd\_L\_test=sgd\_L\_test/x\_test.shape[0]  
 print(**"epoch:"**,time)  
 print(**"sgd\_test\_loss:"**,sgd\_L\_test)  
 time\_list.append(time)  
 sgd\_L\_list.append(np.array(sgd\_L\_test)[0][0])  
  
 *#momentum* momentum\_grad\_w = 0  
 momentum\_L\_test = 0  
 **for** i **in** random.sample(range(x\_train.shape[0]), 10):  
 **if** (1 - (y\_train[i] \* (x\_train[i] \* momentum\_w)) > 0):  
 momentum\_grad\_w = momentum\_grad\_w +momentum\_w - c \* (y\_train[i] \* x\_train[i]).T  
 **else**:  
 momentum\_grad\_w = momentum\_grad\_w + momentum\_w  
 momentum\_v\_w = momentum\_gamma\_w \* momentum\_v\_w + momentum\_rate \* momentum\_grad\_w  
 momentum\_w = momentum\_w - momentum\_v\_w  
 **for** i **in** range(x\_test.shape[0]):  
 **if** (1 - y\_test[i] \* (x\_test[i] \* momentum\_w ) > 0):  
 momentum\_L\_test = momentum\_L\_test + (1 - y\_test[i] \* (x\_test[i] \* momentum\_w))  
 momentum\_L\_test = 0.5 \* momentum\_w.T \* momentum\_w + c \* momentum\_L\_test  
 momentum\_L\_test = momentum\_L\_test / x\_test.shape[0]  
 print(**"momentum\_test\_loss:"**, momentum\_L\_test)  
 momentum\_L\_list.append(np.array(momentum\_L\_test)[0][0])  
  
 *#NAG* nag\_grad\_w = 0  
 nag\_L\_test = 0  
 **for** i **in** random.sample(range(x\_train.shape[0]), 10):  
 **if** (1 - (y\_train[time] \* (x\_train[time] \* nag\_w )) > 0):  
 nag\_grad\_w = nag\_grad\_w + (nag\_w - nag\_gamma\_w \* nag\_v\_w) - c \* (y\_train[time] \* x\_train[time]).T  
 **else**:  
 nag\_grad\_w = nag\_grad\_w + (nag\_w - nag\_gamma\_w \* nag\_v\_w)  
 nag\_v\_w = nag\_gamma\_w \* nag\_v\_w + nag\_rate \* nag\_grad\_w  
 nag\_w = nag\_w - nag\_v\_w  
 **for** i **in** range(x\_test.shape[0]):  
 **if** (1 - y\_test[i] \* (x\_test[i] \* nag\_w ) > 0):  
 nag\_L\_test = nag\_L\_test + (1 - y\_test[i] \* (x\_test[i] \* nag\_w))  
 nag\_L\_test = 0.5 \* nag\_w.T \* nag\_w + c \* nag\_L\_test  
 nag\_L\_test = nag\_L\_test / x\_test.shape[0]  
 print(**"nag\_test\_loss:"**, nag\_L\_test)  
 nag\_L\_list.append(np.array(nag\_L\_test)[0][0])  
  
 *#adagrad* adagrad\_grad\_w = 0  
 adagrad\_L\_test = 0  
 **for** i **in** random.sample(range(x\_train.shape[0]), 10):  
 **if** (1 - (y\_train[i] \* (x\_train[i] \* adagrad\_w )) > 0):  
 adagrad\_grad\_w = adagrad\_grad\_w + adagrad\_w - c \* (y\_train[i] \* x\_train[i]).T  
  
 **else**:  
 adagrad\_grad\_w = adagrad\_grad\_w + adagrad\_w  
 adagrad\_G\_w = adagrad\_G\_w + np.array(adagrad\_grad\_w) \* np.array(adagrad\_grad\_w)  
 adagrad\_w = adagrad\_w - adagrad\_rate \* np.array(adagrad\_grad\_w) / np.sqrt(adagrad\_G\_w + adagrad\_epsilon)  
 **for** i **in** range(x\_test.shape[0]):  
 **if** (1 - y\_test[i] \* (x\_test[i] \* w ) > 0):  
 adagrad\_L\_test = adagrad\_L\_test + (1 - y\_test[i] \* (x\_test[i] \* adagrad\_w))  
 adagrad\_L\_test = 0.5 \* adagrad\_w.T \* adagrad\_w + c \* adagrad\_L\_test  
 adagrad\_L\_test = adagrad\_L\_test / x\_test.shape[0]  
 print(**"adagrad\_test\_loss:"**, adagrad\_L\_test)  
 adagrad\_L\_list.append(np.array(adagrad\_L\_test)[0][0])  
  
 *#rmsprop* rmsprop\_grad\_w = 0  
 rmsprop\_L\_test = 0  
 **for** i **in** random.sample(range(x\_train.shape[0]), 10):  
 **if** (1 - (y\_train[i] \* (x\_train[i] \* rmsprop\_w )) > 0):  
 rmsprop\_grad\_w = rmsprop\_grad\_w + rmsprop\_w - c \* (y\_train[i] \* x\_train[i]).T  
  
 **else**:  
 rmsprop\_grad\_w = rmsprop\_grad\_w + rmsprop\_w  
 rmsprop\_G\_w = rmsprop\_G\_w\*rmsprop\_gamma\_w +np.array(rmsprop\_grad\_w) \*np.array(rmsprop\_grad\_w)\*(1-rmsprop\_gamma\_w)  
 rmsprop\_w = rmsprop\_w - rmsprop\_rate \*rmsprop\_grad\_w / np.sqrt(rmsprop\_G\_w + rmsprop\_epsilon)  
 **for** i **in** range(x\_test.shape[0]):  
 **if** (1 - y\_test[i] \* (x\_test[i] \* rmsprop\_w ) > 0):  
 rmsprop\_L\_test = rmsprop\_L\_test + (1 - y\_test[i] \* (x\_test[i] \* rmsprop\_w))  
 rmsprop\_L\_test = 0.5 \* rmsprop\_w.T \* rmsprop\_w + c \* rmsprop\_L\_test  
 rmsprop\_L\_test = rmsprop\_L\_test / x\_test.shape[0]  
 print(**"rmsprop\_test\_loss:"**, rmsprop\_L\_test)  
 rmsprop\_L\_list.append(np.array(rmsprop\_L\_test)[0][0])  
  
 *#adadelta* adadelta\_grad\_w = 0  
 adadelta\_L\_test = 0  
 **for** i **in** random.sample(range(x\_train.shape[0]), 10):  
 **if** (1 - (y\_train[i] \* (x\_train[i] \* adadelta\_w)) > 0):  
 adadelta\_grad\_w = adadelta\_grad\_w + adadelta\_w - c \* (y\_train[i] \* x\_train[i]).T  
  
 **else**:  
 adadelta\_grad\_w = adadelta\_grad\_w + adadelta\_w  
 adadelta\_G\_w = adadelta\_G\_w \* adadelta\_gamma\_w + np.array(adadelta\_grad\_w) \* np.array(adadelta\_grad\_w) \* (1 - adadelta\_gamma\_w)  
 adadelta\_delta\_w=-np.sqrt(adadelta\_t\_w+adadelta\_epsilon)/np.sqrt(adadelta\_G\_w+adadelta\_epsilon) \* 0.1\*np.array(adadelta\_grad\_w)  
 adadelta\_w = adadelta\_w +adadelta\_delta\_w  
 adadelta\_t\_w = adadelta\_gamma\_w \* adadelta\_t\_w + (1 - adadelta\_gamma\_w) \* np.array(adadelta\_delta\_w) \* np.array(adadelta\_delta\_w)  
 **for** i **in** range(x\_test.shape[0]):  
 **if** (1 - y\_test[i] \* (x\_test[i] \* adadelta\_w ) > 0):  
 adadelta\_L\_test = adadelta\_L\_test + (1 - y\_test[i] \* (x\_test[i] \* adadelta\_w))  
 adadelta\_L\_test = 0.5 \* adadelta\_w.T \* adadelta\_w + c \* adadelta\_L\_test  
 adadelta\_L\_test = adadelta\_L\_test / x\_test.shape[0]  
 print(**"adadelta\_test\_loss:"**, adadelta\_L\_test)  
 adadelta\_L\_list.append(np.array(adadelta\_L\_test)[0][0])  
  
 *#adam* adam\_grad\_w = 0  
 adam\_L\_test = 0  
 **for** i **in** random.sample(range(x\_train.shape[0]), 10):  
 **if** (1 - (y\_train[time] \* (x\_train[time] \* adam\_w)) > 0):  
 adam\_grad\_w =adam\_grad\_w + adam\_w - c \* (y\_train[time] \* x\_train[time]).T  
 **else**:  
 adam\_grad\_w = adam\_grad\_w + adam\_w  
 adam\_m\_w = adam\_beta\_w \* adam\_m\_w + (1 - adam\_beta\_w) \* adam\_grad\_w  
 adam\_G\_w = adam\_G\_w \* adam\_gamma\_w + np.array(adam\_grad\_w) \* np.array(adam\_grad\_w) \* (1 -adam\_gamma\_w)  
 adam\_alpha\_w = adam\_rate \* math.sqrt(1 - math.pow(adam\_gamma\_w, time)) / (1 - math.pow(adam\_beta\_w, time))  
 adam\_w = adam\_w - adam\_alpha\_w\*adam\_m\_w/np.sqrt(adam\_G\_w+adam\_epsilon)  
 **for** i **in** range(x\_test.shape[0]):  
 **if** (1 - y\_test[i] \* (x\_test[i] \* adam\_w ) > 0):  
 adam\_L\_test = adam\_L\_test + (1 - y\_test[i] \* (x\_test[i] \* adam\_w))  
 adam\_L\_test = 0.5 \* adam\_w.T \* adam\_w + c \* adam\_L\_test  
 adam\_L\_test = adam\_L\_test / x\_test.shape[0]  
 print(**"adam\_test\_loss:"**, adam\_L\_test)  
 adam\_L\_list.append(np.array(adam\_L\_test)[0][0])  
  
  
*# In[30]:  
  
#画图*plt.xlabel(**'epoch'**)  
plt.ylabel(**'loss'**)  
line1=plt.plot(time\_list,sgd\_L\_list)  
line2=plt.plot(time\_list,momentum\_L\_list)  
line3=plt.plot(time\_list,nag\_L\_list)  
line4=plt.plot(time\_list,adagrad\_L\_list)  
line5=plt.plot(time\_list,rmsprop\_L\_list)  
line6=plt.plot(time\_list,adadelta\_L\_list)  
line7=plt.plot(time\_list,adam\_L\_list)  
label = [**"sgd"**,**"momentum"**,**"nag"**,**"adagrad"**,**"rmsprop"**,**"adadelta"**,**"adam"**]  
plt.legend(label, loc = 0, ncol = 7)  
plt.show()  
  
  
*# In[ ]:* **8\_2. The initialization method of model parameters:** set parameter to all zero

**9\_2. The selected loss function and its derivatives:**loss function=1/2 w.T w + cmax(0,1-yi(w.Txi+b))。   
for **its derivatives of w**:  
if 1-yi(w.Txi+b)>=0，grad\_w = w-c(yixi).T,   
if 1-yi(w.Txi+b)< 0， grad\_w=w.   
**its derivatives of b**，  
if 1-yi(w.Txi+b)>=0，grad\_b = -cyi,   
if 1-yi(w.T\*xi+b)< 0，grad\_b=0.

**10\_2. Experimental results and curve:**

**NAG：**  
Hyper-parameter selection:

nag\_gamma\_w=0.9  
nag\_rate=0.0005  
nag\_v\_w = 0

epoch=20

Predicted Results (Best Results):loss=0.4823759

**RMSProp：**  
Hyper-parameter selection:

rmsprop\_rate=0.007  
rmsprop\_G\_w=0  
rmsprop\_epsilon=math.pow(np.e,-6)  
rmsprop\_gamma\_w=0.9

epoch=20

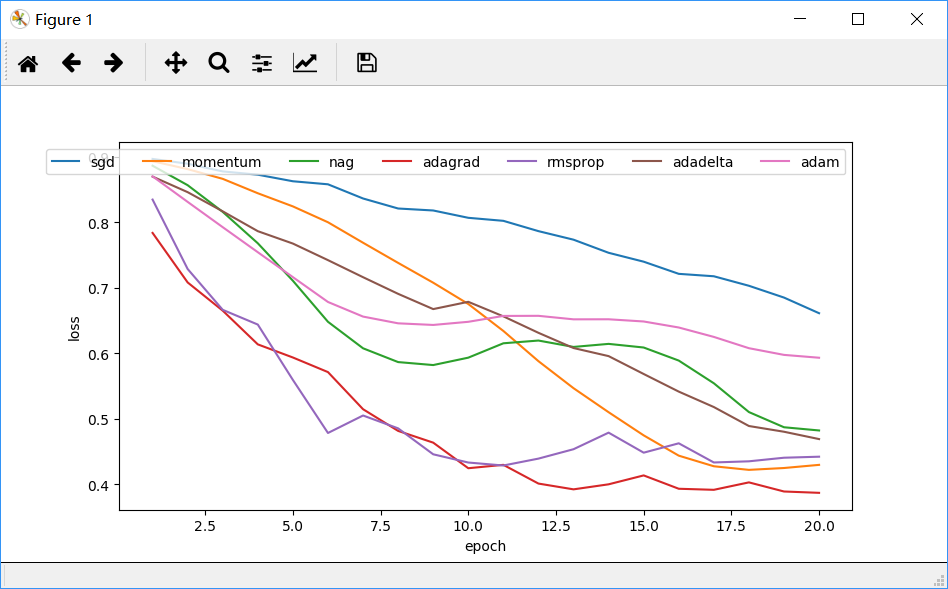
Predicted Results (Best Results):loss= 0.42886678

**AdaDelta：**  
Hyper-parameter selection:  
adadelta\_G\_w=0  
adadelta\_epsilon=math.pow(np.e,-6)  
adadelta\_gamma\_w=0.09  
adadelta\_t\_w=0

epoch=20

Predicted Results (Best Results):loss= 0.46921928

**Adam：**  
Hyper-parameter selection:  
adam\_G\_w=0  
adam\_epsilon=math.pow(np.e,-6)  
adam\_gamma\_w=0.99  
adam\_t\_w=0  
adam\_m\_w=0  
adam\_beta\_w=0.9  
epoch=20  
Predicted Results (Best Results):loss=0.59343847

Loss curve:  


**11\_2. Results analysis:**rmsprop’s effect is best. nag and adadelta have the similar effect. adam is worst .

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

The two methods are common classification algorithms, from the objective function point of view, the difference is that the logistic regression using logistical loss, svm using hinge loss. The purpose of these two loss functions are to increase the classification of data Point weight and reduce the weight of the data points less relevant to the classification.

**13. Summary:**

I learned a lot from this experiment.such as those optimized methods,which are hard to understand and code to achieve. In a word , I have a deeper understanding of Logistic Regression, Linear Classification and Stochastic Gradient Descent.