South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Xiaotao Liu

Supervisor:
Mingkui Tan

Student ID：
201530612378

Grade:
Undergraduate

December 12, 2017

Comparison of Various Stochastic Gradient Descent for solving Classification Problem

Abstract—The purpose of this experiment is to request us to independently accomplish logistic regression and linear classification (SVM) by using SGD. And then compare various optimization algorithm of stochastic Gradient on solving classification problem such as SVM and logistic regression , with respect to further understand the essence of the 4 algorithms and their differences.

I. Introduction

Logistic regression and linear classification are the two of most fundamental machine learning models. Stochastic gradient descent(SGD) is an improved version of traditional GD.It accelerates the process that the model reaches its convergence point. From the overall effect, most of the time it can only approach the local optimal solution, so it is suitable for larger training set case. This experiment aims to compare Logistic regression and SVM using SGD to help understanding the differences and relations. What's more, we compare 4 kinds of optimization algorithms of stochastic gradient descent on classification problem and understand their differences. Lastly, we practice SVM on larger data to have a better command of its principles.

II. Methods and Theory

A.The selected loss function and its derivatives

1) *Logistic regression:*

The loss function I select is cross-entropy cost function.

$$L_{\mathrm{D}}(\mathrm{w}) = -\frac{1}{m}\sum_{i=1}^{m}\ln(g(y_i \cdot \mathrm{w}^T x_i))$$

where $g_{\mathrm{w}}(X) = \frac{1}{1+e^{-\mathrm{w}^{\mathrm{T}}\cdot X}}$

Its derivatives:

$$\frac{\partial L_D(\mathrm{w})}{\partial \mathrm{w}} = -\frac{1}{m}\sum_{i=1}^{m}\frac{1}{g(y_i \mathrm{w}^T x_i)}\cdot\frac{\partial g(y_i \cdot \mathrm{w}^{\mathrm{T}} x_i)}{\partial \mathrm{w}}$$

While, $g^{'}(x) = g(x)(1-g(x))$

$$\text{So,} \frac{\partial L_{\mathrm{D}}(\mathrm{w})}{\partial \mathrm{w}} = -\frac{1}{m}\sum_{i=1}^{m}(1-g(y_i\cdot \mathrm{w}^T x_i))(y_i \cdot x_i)$$

2) *Linear classification(SVM):*

The loss function I select is cross-entropy cost function:

$$L_{\mathrm{D}}(\mathrm{w}) = \frac{1}{2}(\|\mathrm{w}\|^2) + \mathrm{C}\sum_{i=1}^{N}\max(0, 1-y_i(\mathrm{w}^T x_i + b))$$

$$\mathrm{hinge}eloss = \varepsilon_i = \max(0, 1-y_i(\mathrm{w}^T x_i + b))$$

Its derivatives:

$$1.\ \frac{1}{2}\frac{\partial(\|\mathrm{w}\|^2)}{\partial \mathrm{w}} = \mathrm{w}$$

$$2.\ \mathrm{g}_{\mathrm{w}}(x_{\mathrm{i}}) = \frac{\partial(\sum_{i=1}^{N}\max(0, 1-y_i(w^T x_i + b)))}{\partial w}$$

if $y_i(w^T x_i + b) <= 1$,

$$\mathrm{g}_{\mathrm{w}}(x_{\mathrm{i}}) = \frac{\partial(-y_i(\mathrm{w}^T x_i + b))}{\partial \mathrm{w}} = -\frac{\partial(y_i \mathrm{w}^T x_i)}{\partial \mathrm{w}} = -\mathrm{y}_i x_i$$

else

$$\mathrm{g}_{\mathrm{w}}(x_{\mathrm{i}}) = 0$$

It can turn into another form:

$$\frac{\partial \mathrm{L}(\mathrm{w})}{\partial w_{\mathrm{j}}} = w_j - C\sum_{i=1}^{N}(x_i y_i) y_{\_i}$$

if $y_i(w^T x_i + b) <= 1$ then $\mathrm{y}_{\_i} = 1$,

else $\mathrm{y}_{\_i} = 0$

B. Experiment 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 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 using in the experiment and data analysis

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

Data analysis:There are 123 features and 1 label in training set. However, it has 122 features and 1 label in testing set.After checking, I find that the last features in training set don't be in testing set after compression.

B. Initialization of model parameters

For both Logistic regression and SVM,I initialize their parameters into zeros

C. Implementation

1) Logistic regression:

Having defined loss function(cross-entropy cost function) and its gradient, we can use SGD to realize logistic regression. We use five method respectively to reach the local optimal solution including SGD(without optimization), NAG, RMSProp, AdaDelta and Adam. The super parameters we select are as follows.

SGD	learning rate(eta)	0.008
	epcohs	4000
NAG	learning rate(eta)	0.001
	gamma	0.9
	epcohs	4000
RMSProp	learning rate(eta)	0.001
	gamma	0.9
	epsilon	0.0001
	epcohs	4000
AdaDelta	learning rate(eta)	0.00001
	gamma	0.9
	epsilon	1e-6
	epcohs	4000
Adam	learning rate(eta)	0.0008
	belta	0.9
	gamma	0.9
	epsilon	0.0001
	epcohs	4000

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

```
In [71]: import numpy as np
         import math
         import random
         import matplotlib.pyplot as plt
         from sklearn.externals.joblib import Memory
         from sklearn.datasets import load_svmlight_file
         from sklearn.model_selection import train_test_split
```

```
In [72]: 的函数
         data(file):
         =load_svmlight_file(file)              #调用load_svmlight () 用于读取函数
         rn data[0].todense(),data[1]              #返回值，第一个为训练特征数据，第二个为训练标签
```

```
In [73]: #训练集读取数据
         X_train,y_train=get_data('./a9a')
         #测试机读取数据
         X_test,y_test=get_data('./a9a.t')

         (n,m)=np.shape(X_train)                          #获取X_train的横纵维度
         X_train=np.hstack((X_train,np.ones((n,1))))      #为训练数据的每一条数据增加一列，作为
         (n,m)=np.shape(X_test)                           #获取X_test的横纵维度
         X_test=np.hstack((X_test,np.zeros((n,1))))       #经发现，为第123列没有数据，故补充一列
         X_test=np.hstack((X_test,np.ones((n,1))))        #为测试数据的每一条数据增加一列，bias
```

```
In [74]: #将y_train和y_test变为n*1的列向量
         y_train=np.reshape(y_train,(len(y_train),1))
         y_test=np.reshape(y_test,(len(y_test),1))
```

```
In [75]: #将结果分类的函数
         def sigmoid(X,w):
             h=1.0/(1+np.exp(-X*w))                       #用于预测结果分类的sigmod函数
             return h
```

sigmoid函数公式为：

$$h_w(x) = \frac{1}{1+e^{-w^T x}}$$

```
In [76]: #求梯度函数，其中正则系数lambda默认为0，还没有使用
         def gradient(X,y,w,lamda=0):
             y_=X*w                                       #X与w相乘，得到预测结果
             y_y=np.multiply(y,y_)                        #y_y为y的i真实值和预测值的相应位置相
             y_x=np.multiply(y,X)                         #y_x为矩阵X与y的相应位置相乘组成的
             grad=-np.mean(y_x/(1+np.exp(y_y)),0).T       #梯度计算公式，见以下公式

             return grad
```

梯度公式为:

$$\frac{\partial J}{\partial w} = -\frac{1}{m}\sum_{i=1}^{m}(1-g(y_i \cdot W^T x_i))\ (y_i \cdot x_i)$$

```
In [77]: #预测函数，预测y值大小
         def predict(X,w):
             threshold=0.5                        #设置阈值
             y=np.ones((np.shape(X)[0],1))
             h=sigmoid(X,w)                       #调用sigmod函数，计算h值的，用于预测y为1或者
             y[h>=threshold]=1                    #对于计算值h大于0.5的预测为1
             y[h<threshold]=-1                    #计算值值小于0.5的预测为-1
             return y
```

In [78]:
```python
#准确率计算函数
def accuracy(X,w,y):
    y_predict=predict(X,w)                  #得到X，w的预测值{-1，1}
    N=np.zeros((len(y),1))                  #N用于接收结果
    N[y_predict==y]=1                       #如果预测值与真实值相同，则设置1，

    return np.mean(N)                       #返回0，1矩阵的均值，代表准确率
```

准确率计算公式为

$$accuracy = \frac{N_{true}}{N_{total}}$$

In [79]:
```python
#交叉熵损失函数
def crossEntropyCost(X,y,w):
    y_=X*w                                  #X与w相乘，得到预测结果
    y_y=np.multiply(y_,y)                   #y_y为y的i真实值和预测值的相应位置
    loss=np.mean(np.log(1+np.exp(-y_y)))    #对y_y取对数求和并作均值计算，得到
    return loss
```

交叉熵损失函数的公式为：

$$L_D(W) = -\frac{1}{m}\sum_{i=1}^{m} ln(g(y_i \cdot W^T x_i))$$

In [80]:
```python
#未优化的sgd的实现函数
def SGD_gradient(X,y,w,eta):
    grad=gradient(X,y,w)        #计算梯度
    w=w-eta*grad                #进行一步梯度下降
    return w
```

SGD实现原理：

$$g_t = \frac{\partial J(W_{t-1})}{\partial w}$$
$$W_t = W_{t-1} - \eta g_t$$

In [81]:
```python
#NAG sgd的实现函数
def NAG_gradient(X,y,w,v,gamma,eta):
    grad=gradient(X,y,w-gamma*v)    #计算梯度
    v=gamma*v+eta*grad              #更新v值
    w=w-v                           #梯度下降
    return w,v
```

NAG实现原理：

$$g_t = \frac{\partial J}{\partial w}(W_{t-1} - \gamma v_{t-1})$$
$$v_t = \gamma v_{t-1} + \eta g_{t-1}$$
$$W_t = W_{t-1} - v_t$$

In [82]:
```python
#RMS sgd的实现函数
def RMS_gradient(X,y,w,G,gamma,etha,epison):
    grad=gradient(X,y,w)                            #计算梯度
    dotMultiply=np.multiply(grad,grad)              #梯度进行点乘
    G=gamma*G+(1-gamma)*dotMultiply                 #G值由过去的G值和新的梯度点乘求得
    dot=np.multiply(etha/(np.sqrt(G+epison)),grad)
    w=w-dot                                         #梯度下降
    return w,G
```

RMS实现原理：

$$g_t = \frac{\partial J(W_{t-1})}{\partial w}$$
$$G_t = \gamma G_t + (1-\gamma) g_t \cdot g_t)$$
$$W_t = W_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t$$

In [83]:
```python
#Adadelta sgd实现
def Adadelta_gradient(X,y,w,G,gamma,delta,epision):
    grad=gradient(X,y,w)                                                    #计算梯度
    dotMultiply=np.multiply(grad,grad)                                      #梯度进行点乘
    G=gamma*G+(1-gamma)*dotMultiply                                         #G值由过去的G值和新的
    w_delta=-np.multiply(np.sqrt(delta+epision)/np.sqrt(G+epision),grad)   #w_delta计算
    w=w+w_delta                                                             #梯度下降
    delta=gamma*delta+(1-gamma)*(np.multiply(w_delta,w_delta))              #新delta由过去的delt
    return w,G,delta
```

#Adadelta实现原理

$$g_t = \Delta J(W_{t-1}$$
$$G_t = \gamma W_t + (1-\gamma) g_t \cdot g_t$$
$$\Delta W_t = -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \cdot g_t$$
$$W_t = W_{t-1} + \Delta W_t$$
$$\Delta_t = \gamma \Delta_{t-1} + (1-\gamma)\Delta W_t \cdot \Delta W_t$$

In [84]:
```python
#Adam sgd实现
def Adam_gradient(X,y,w,G,m,gamma,t,etha,belta,epision):
    grad=gradient(X,y,w)                                                    #计算梯度
    m=belta*m+(1-belta)*grad                                                #根据原有的m值和新
    G=gamma*G+(1-gamma)*(np.multiply(grad,grad))                            #G值由过去的G值和新
    alpha=etha*(np.sqrt(1-math.pow(gamma,t))/(1-math.pow(belta,t)))        #计算alpha
    w=w-alpha*(m/(np.sqrt(G+epision)))                                      #梯度下降
    return w,G,m
```

Adam的实现原理：

$$g_t = \frac{\partial J(W_{t-1})}{\partial} w$$
$$m_t = \beta_1 m_{t-1} + (1-\beta) g_t$$
$$G_t = \gamma G_t + (1-\gamma) g_t \cdot g_t$$
$$\alpha = \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$
$$\Theta = \Theta_{t-1} - \alpha \frac{m_t}{\sqrt{G_t + \epsilon}}$$

In [85]:
```python
#分别为NAG,RMSProb,Adadelta,Adam初始化权重向量等
(n,m)=np.shape(X_train)
#SGD初始化权重矩阵
w_sgd=np.zeros((m,1))


#NAG初始化权重矩阵
w_nag=np.zeros((m,1))
v_nag=np.zeros((m,1))
#RMS初始化权重矩阵
w_rms=np.zeros((m,1))
G_rms=np.zeros((m,1))
#Adadelta初始化权重矩阵
w_ada=np.zeros((m,1))
G_ada=np.zeros((m,1))
delta=np.zeros((m,1))
#Adam初始化权重矩阵
w_adam=np.zeros((m,1))
G_adam=np.zeros((m,1))
m_adam=np.zeros((m,1))

#迭代次数
epco=4000
times=range(epco)
```

```python
#定义未优化的SGD函数
def SGD(X,y,w_sgd,eta,train_size,epco,gradient=SGD_gradient,loss=crossEntropyCost,accuracy=accuracy):
    sgd_train=[]                    #训练误差列表
    sgd_test=[]                     #测试误差列表
    sgd_accuracy_train=[]           #训练准确率列表
    sgd_accuracy_test=[]            #测试准确率列表
    #开始进行梯度下降过程
    for i in range(epco):
        #得到用于梯度下降的小部分样本，即X_trainset,y_trainset
        X_trainset,X_other,y_trainset,y_other=train_test_split(X,y,test_size=1-train_size,random_state=random.randint(0,1000))
        #调用未优化的sgd的梯度下降函数，每一次迭代进行一步梯度下降
        w_sgd=gradient(X_trainset,y_trainset,w_sgd,eta)
        sgd_train.append(loss(X_trainset,y_trainset,w_sgd))         #得到训练和测试的loss
        sgd_test.append(loss(X_test,y_test,w_sgd))
        sgd_accuracy_train.append(accuracy(X_train,w_sgd,y_train))      #计算训练和测试的准确率
        sgd_accuracy_test.append(accuracy(X_test,w_sgd,y_test))
    return sgd_train,sgd_test,sgd_accuracy_train,sgd_accuracy_test
#参数初始化
eta=0.008                   #学习率设为0.01
train_size=0.0005           #指用于训练SGD的样本数占X_train的比例,约有15个样本用于训练
#开始训练，得到训练误差，测试误差，训练准确率和测试准确率
sgd_train,sgd_test,sgd_accuracy_train,sgd_accuracy_test=SGD(X=X_train,y=y_train,w_sgd=w_sgd,eta=eta,train_size=train_size,epco=epco)
```

```python
#定义NAG sgd的训练函数
def NAG(X,y,w_nag,v_nag,gamma,eta,train_size,gradient=NAG_gradient,loss=crossEntropyCost,accuracy=accuracy):
    nag_train=[]                #训练误差列表
    nag_test=[]                 #测试误差列表
    nag_accuracy_train=[]       #训练准确率列表
    nag_accuracy_test=[]        #测试准确率列表
    for i in range(epco):
        #得到用于随机梯度下降训练的小部分样本 (即X_trainset,y_trainset)
        X_trainset,X_other,y_trainset,y_other=train_test_split(X,y,test_size=1-train_size,random_state=random.randint(0,1000))
        #调用NG_gradient,每一次迭代进行一次梯度下降
        w_nag,v_nag=gradient(X_trainset,y_trainset,w_nag,v_nag,gamma,eta)
        nag_train.append(loss(X_trainset,y_trainset,w_nag))             #得到训练和测试的loss
        nag_test.append(loss(X_test,y_test,w_nag))
        nag_accuracy_train.append(accuracy(X_trainset,w_nag,y_trainset))  #计算训练和测试的准确率
        nag_accuracy_test.append(accuracy(X_test,w_nag,y_test))
    return nag_train,nag_test,nag_accuracy_train,nag_accuracy_test
#参数初始化
gamma=0.9
eta=0.001                   #学习率设为0.001
train_size=0.0005           #指用于训练SGD的样本数占X_train的比例，约有15个样本用于训练
#开始训练，得到训练误差，测试误差，训练准确率和测试准确率
nag_train,nag_test,nag_accuracy_train,nag_accuracy_test=NAG(X=X_train,y=y_train,w_nag=w_nag,v_nag=v_nag,gamma=gamma,eta=eta,train_size=train_si
```

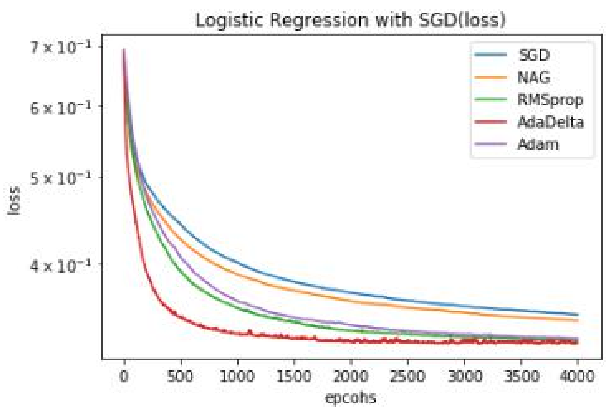
```
#定义RMS sgd的实现方法
def RMS(X,y,w_rms,G_rms,gamma,eta,epco,train_size,gradient=RMS_gradient,loss=crossEntropyCost,accuracy=accuracy):
    epision=0.0001                          #设置epision为0.0001，既避免Gt为零时的错误，也减小该参数对结果的影响
    rms_train=[]                            #训练误差列表
    rms_test=[]                             #测试误差列表
    rms_accuracy_train=[]                   #训练准确率列表
    rms_accuracy_test=[]                    #测试准确率列表
    for i in range(epco):
        #得到用于随机梯度下降训练的小部分样本
        X_trainset,X_other,y_trainset,y_other=train_test_split(X,y,test_size=1-train_size,random_state=random.randint(0,1000))
        w_rms,G_rms=gradient(X_trainset,y_trainset,w_rms,G_rms,gamma,eta,epision)      #梯度下降更新
        rms_train.append(loss(X_trainset,y_trainset,w_rms))             #得到训练和测试的loss
        rms_test.append(loss(X_test,y_test,w_rms))
        rms_accuracy_train.append(accuracy(X_trainset,w_rms,y_trainset))   #计算准确率
        rms_accuracy_test.append(accuracy(X_test,w_rms,y_test))
    return rms_train,rms_test,rms_accuracy_train,rms_accuracy_test
#参数初始化
gamma=0.9
eta=0.001                                               #学习率设为0.001
train_size=0.0005                                       #指用于训练SGD的样本数占X_train的比例
#开始训练，得到训练误差，测试误差，训练准确率和测试准确率
rms_train,rms_test,rms_accuracy_train,rms_accuracy_test=RMS(X=X_train,y=y_train,w_rms=w_rms,G_rms=G_rms,gamma=gamma,eta=eta,epco=epco,train_siz
```

```
#定义Adadelta sgd的实现方法
def Adadelta(X,y,w_ada,G_ada,delta,gamma,eta,epco,train_size,gradient=Adadelta_gradient,loss=crossEntropyCost,accuracy=accuracy):
    epision=0.000001                        #设置epision为0.0001，既避免Gt为零时的错误，也减小该参数对结果的影响
    ada_train=[]                            #训练误差列表
    ada_test=[]                               #测试误差列表
    ada_accuracy_train=[]                       #训练准确率列表
    ada_accuracy_test=[]                          #测试准确率列表
    for i in range(epco):
        #得到用于随机梯度下降训练的小部分样本
        X_trainset,X_other,y_trainset,y_other=train_test_split(X,y,test_size=1-train_size,random_state=random.randint(0,1000))
        #调用Adadelta_gradient,每一次迭代进行一次梯度下降
        w_ada,G_ada,delta=gradient(X_trainset,y_trainset,w_ada,G_ada,gamma,delta,epision)
        ada_train.append(loss(X_trainset,y_trainset,w_ada))             #得到训练和测试的loss
        ada_test.append(loss(X_test,y_test,w_ada))
        ada_accuracy_train.append(accuracy(X_trainset,w_ada,y_trainset)) #计算训练和测试准确率
        ada_accuracy_test.append(accuracy(X_test,w_ada,y_test))

    return ada_train,ada_test,ada_accuracy_train,ada_accuracy_test
#参数初始化
gamma=0.9
eta=0.00001                                         #学习率设为0.001
train_size=0.0005                                   #指用于训练SGD的样本数占X_train的比例
#开始训练，得到训练误差，测试误差，训练准确率和测试准确率
ada_train,ada_test,ada_accuracy_train,ada_accuracy_test=Adadelta(X=X_train,y=y_train,w_ada=w_ada,G_ada=G_ada,delta=delta,gamma=gamma,eta=eta,ep
```

```
#定义Adam sgd的实现方法
def Adam(X,y,w_adam,G_adam,m_adam,gamma,eta,epco,train_size,gradient=Adam_gradient,loss=crossEntropyCost,accuracy=accuracy):
    epision=0.0001                          #设置epision为0.0001，既避免Gt为零时的错误，也减小该参数对结果的影响
    adam_train=[]                       #训练误差列表
    adam_test=[]                        #测试误差列表
    adam_accuracy_train=[]              #训练准确率列表
    adam_accuracy_test=[]               #测试准确率列表
    for i in range(epco):
        #得到用于随机梯度下降训练的小部分样本
        X_trainset,X_other,y_trainset,y_other=train_test_split(X,y,test_size=1-train_size,random_state=random.randint(0,1000))
        # 调用Adam_gradient,每一次迭代进行一次梯度下降
        w_adam,G_adam,m_adam=gradient(X_trainset,y_trainset,w_adam,G_adam,m_adam,gamma,i+1,etha,belta,epision)
        adam_train.append(loss(X_trainset,y_trainset,w_adam))               #得到训练和测试的loss
        adam_test.append(loss(X_test,y_test,w_adam))
        adam_accuracy_train.append(accuracy(X_trainset,w_adam,y_trainset))   #计算训练和测试准确率
        adam_accuracy_test.append(accuracy(X_test,w_adam,y_test))
    return adam_train,adam_test,adam_accuracy_train,adam_accuracy_test
#参数初始化
gamma=0.9
etha=0.0008                                 #学习率设为0.01
belta=0.9
train_size=0.0005                           #指用于训练SGD的样本数占X_train的比例
#开始训练，得到训练误差，测试误差，训练准确率和测试准确率
adam_train,adam_test,adam_accuracy_train,adam_accuracy_test=Adam(X=X_train,y=y_train,w_adam=w_adam,G_adam=G_adam,m_adam=m_adam,gamma=gamma,eta=
```

```
#打印出各种实现方法的loss曲线
plt.yscale('log')
plt.title('Logistic Regression with SGD(loss)')
plt.xlabel('epcohs')
plt.ylabel('loss')
print(np.shape(sgd_test))
plt.plot(times,sgd_test,label="SGD")
plt.plot(times,nag_test,label="NAG")
plt.plot(times,rms_test,label="RMSprop")
plt.plot(times,ada_test,label="AdaDelta")
plt.plot(times,adam_test,label="Adam")
plt.legend()
plt.show()
```

Then, we get the following loss graphs as results after running the program.



Result Analysis:

1. From the graph we find that AdaDelta reaches the local optimal solution fastest in solving logistic regression, but it also has obvious shaking at end than others. Its biggest characteristic is self-adaptive and high-speeding. That is because it only use first-order information and has small calculation costs.
2. RMSprop is a variant of AdaDelta, slightly less effective than AdaDelta. From the graph, we find that it have a more stable but slower curve. It is suitable for handling non-stationary targets
3. Adam is essentially RMSprop with momentum terms. It has a closely same result as RMSprop at end but slower than it at first. From its formula,it needs less memory requirement and calculate different adaptive learning rates for different parameters
4. By contrast, SGD is slowest one for which it use typically learning rate and gradient to update w. Which has a slow convergence speed. NAG plays relatively well than SGD.
5. Four methods reaches optimal solution far faster than traditional GD because they only randomly use a set of training set to train model.

2) Linear classification(SVM):

Having defined loss function(hinge loss)and its gradient, we can use SGD to get realize the SVM. We use five method respectively to reach the local optimal solution including SGD(without optimization), NAG, RMSProp, AdaDelta and Adam. The super parameters we select are as follows.

SGD	learning rate(eta)	0.0005
	C	1
	threshold	0.5
	epcohs	4000
NAG	learning rate(eta)	0.00001
	gamma	0.9
	C	1
	threshold	0.5
	epcohs	4000
RMSProp	learning rate(eta)	0.001
	gamma	0.9
	epsilon	0.00001
	C	1
	threshold	0.5
	epcohs	4000
AdaDelta	learning rate(eta)	0.000001
	gamma	0.9
	epsilon	1e-7
	C	1
	threshold	0.5
	epcohs	4000
Adam	learning rate(eta)	0.001
	belta	0.9
	epsolon	0.0001
	C	1
	threshold	0.5
	gamma	0.9

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

```
import numpy as np
import math
import random
import matplotlib.pyplot as plt
from sklearn.externals.joblib import Memory
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
```

```
#读取数据的函数
def get_data(file):
    data=load_svmlight_file(file)                #调用load_svmlight () 用于读取函数
    return data[0].todense(),data[1]             #返回值, 第一个为训练特征数据, 第二个
#训练集和测试集读取数据
X_train,y_train=get_data('./a9a')
X_test,y_test=get_data('./a9a.t')
```

```
#获取X_train的横纵维度
(n,m)=np.shape(X_train)
X_train=np.hstack((X_train,np.ones((n,1))))     #为训练数据的每一条数据增加一列,作为b
(n,m)=np.shape(X_test)                          #获取X_test的横纵维度
X_test=np.hstack((X_test,np.zeros((n,1))))      #经发现, 为第123列没有数据, 故补充一列
X_test=np.hstack((X_test,np.ones((n,1))))       #为测试数据的每一条数据增加一列,作为b

#将y_train和y_test变为n*1的列向量
y_train=np.reshape(y_train,(len(y_train),1))
y_test=np.reshape(y_test,(len(y_test),1))
```

```
#定义hingeloss损失函数
def hingeloss(X,y,w):
    l=1-np.multiply(y,X*w)              #y与X*w先做点乘,相应位置相乘
    l2=(l>=0)                           #然后求出哪个元素大于等于0, 化成布尔矩
    result=np.multiply(l,l2)            #通过l和l2点乘进行筛选, 大于等于0的
    return np.mean(result)
```

hinge loss的公式:

$$Hingeloss = \epsilon_i = max(0 - 1 - y_i(w^T x_i + b))$$

```
#准确率计算函数
def accuracy(X,w,y):
    threshold=0.5                      #设置阈值未0.5
    y_predict=X*w                      #得到X, w的预测结果
    val=np.multiply(y,y_predict)       #y预测值与y真实值相乘
    N=np.zeros((len(y),1))
    N[val>=threshold]=1        #y预测值与y真实值相乘,积大于阈值赋值1, 表示分类正确,否则
    return np.mean(N)          #得到准确率
```

```
#求梯度函数, 其中正则系数C默认值为1
def gradient(X,y,w,C=1):
    l=1-np.multiply(y,X*w)             #先做点乘
    l2=(l>=0)                          #然后求出哪个元素大于等于0, 化成布尔
    tmp=np.multiply(y,l2)              #通过点乘进行筛选, 大于等于0的项保留
    return w-C*np.sum(np.multiply(X,tmp),0).T   #计算最终梯度
```

梯度公式:

$$gradient = w - \frac{C}{n}\sum_{i=1}^{n}(x_i y_i)y\_i$$

其中y_i=1 if $1 - y_i(w^T x_i)$ else $y\_i = 0$
在上述gradient函数实现中，代码中的C相当于此处C/n,视线中，我将除以n包进C中

```
#未优化的sgd的实现函数
def SGD_gradient(X,y,w,eta,C):
    grad=gradient(X,y,w,C)        #计算梯度
    w=w-eta*grad                  #进行一步梯度下降
    return w
```

SGD实现原理:

$$g_t = \frac{\partial J(W_{t-1})}{\partial w}$$
$$W_t = W_{t-1} - \eta g_t$$

```
#NAG sgd的实现函数
def NAG_gradient(X,y,w,v,gamma,eta,C):
    grad=gradient(X,y,w-gamma*v,C)       #计算梯度
    v=gamma*v+eta*grad                   #更新v值
    w=w-v                                #梯度下降
    return w,v
```

NAG实现原理:

$$g_t = \frac{\partial J}{\partial w}(W_{t-1} - \gamma v_{t-1})$$
$$v_t = \gamma v_{t-1} + \eta g_{t-1}$$
$$W_t = W_{t-1} - v_t$$

```
#RMS sgd的实现函数
def RMS_gradient(X,y,w,G,gamma,eta,epision,C):
    grad=gradient(X,y,w,C)                          #计算梯度
    G=gamma*G+(1-gamma)*np.square(grad)             #G值由过去的G值和新的梯度点
    dot=np.multiply(eta/(np.sqrt(G+epision)),grad)
    w=w-dot                                         #梯度下降
    return w,G
```

RMS实现原理:

$$g_t = \frac{\partial J(W_{t-1})}{\partial w}$$
$$G_t = \gamma G_t + (1 - \gamma)g_t \cdot g_t)$$
$$W_t = W_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t$$

```
#Adadelta sgd实现
def Adadelta_gradient(X,y,w,G,gamma,delta,epision,C):
    grad=gradient(X,y,w,C)                    #计算梯度
    dot=np.square(grad)                       #梯度进行点乘
    G=gamma*G+(1-gamma)*dot                   #G值由过去的G值和新的梯度点乘求得
    delta_w=np.multiply((np.sqrt(delta+epision)/np.sqrt(G+epision)),grad)  #w_delta计算
    w=w-delta_w                               #梯度下降
    delta=gamma*delta+(1-gamma)*(np.square(delta_w))   #新delta由过去的delta值和新的w_
    return w,G,delta
```

#Adadelta实现原理

$$g_t = \Delta J(W_{t-1}$$
$$G_t = \gamma W_t + (1 - \gamma)g_t \cdot g_t$$
$$\Delta W_t = -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \cdot g_t$$
$$W_t = W_{t-1} + \Delta W_t$$
$$\Delta_t = \gamma \Delta_{t-1} + (1 - \gamma)\Delta W_t \cdot \Delta W_t$$

```
#Adam sgd实现
def Adam_gradient(X,y,w,G,m,gamma,t,eta,belta,epision,C):
    grad=gradient(X,y,w,C)                            #计算梯度
    m=belta*m+(1-belta)*grad                          #根据原有的m值和
    G=gamma*G+(1-gamma)*(np.multiply(grad,grad))      #G值由过去
    alpha=eta*(np.sqrt(1-math.pow(gamma,t))/(1-math.pow(belta,t)))   #计算alpha
    w=w-alpha*(m/(np.sqrt(G+epision)))                #梯度下降
    return w,G,m
```

Adam的实现原理:

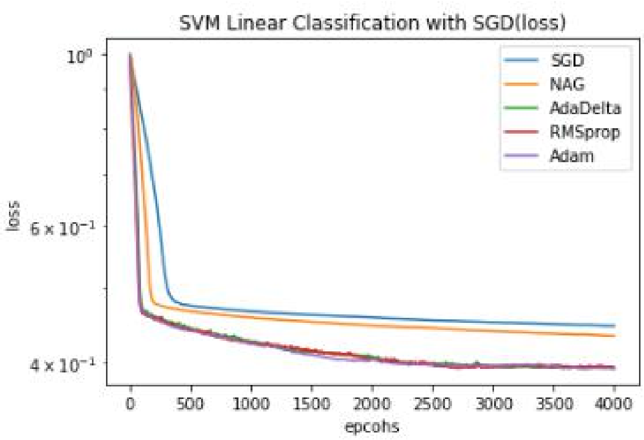
$$g_t = \frac{\partial J(W_{t-1})}{\partial} w$$
$$m_t = \beta_1 m_{t-1} + (1 - \beta)g_t$$
$$G_t = \gamma G_t + (1 - \gamma)g_t \cdot g_t$$
$$\alpha = \eta \frac{\sqrt{1 - \gamma^t}}{1 - \beta^t}$$
$$\Theta = \Theta_{t-1} - \alpha \frac{m_t}{\sqrt{G_t + \epsilon}}$$

```
#分别为NAG,RMSProb,Adadelta,Adam初始化权重向量等
(n,m)=np.shape(X_train)
#SGD的初始化权重矩阵
w_sgd=np.zeros((m,1))

#NAG的初始化权重矩阵
w_nag=np.zeros((m,1))
v_nag=np.zeros((m,1))
#RMS的初始化权重矩阵
w_rms=np.zeros((m,1))
#w_rms=np.random.standard_normal((X_train.shape[1],1))
G_rms=np.zeros((m,1))
#Adadelta的初始化权重矩阵
w_ada=np.zeros((m,1))
G_ada=np.zeros((m,1))
delta=np.zeros((m,1))
#Adam的初始化权重矩阵
w_adam=np.zeros((m,1))
G_adam=np.zeros((m,1))
m_adam=np.zeros((m,1))

#迭代次数
epco=4000
times=range(epco)
```

```
#定义未优化的SGD函数
def SGD(X,y,w_sgd,eta,train_size,epco,C,gradient=SGD_gradient,loss=hingeloss,accuracy=accuracy):
    sgd_train=[]                      #训练误差列表
    sgd_test=[]                          #测试误差列表
    sgd_accuracy_train=[]                 #训练准确率列表
    sgd_accuracy_test=[]                  #测试准确率列表
    #开始进行梯度下降过程
    for i in range(epco):
        #得到用于梯度下降的小部分样本，即X_trainset,y_trainset
        X_trainset,X_other,y_trainset,y_other=train_test_split(X,y,test_size=1-train_size,random_state=random.randint(0,1000))
        #调用未优化的sgd的梯度下降函数，每一次迭代进行一步梯度下降
        w_sgd=gradient(X_trainset,y_trainset,w_sgd,eta)
        sgd_train.append(loss(X_trainset,y_trainset,w_sgd))          #得到训练和测试的loss
        sgd_test.append(loss(X_test,y_test,w_sgd))
        sgd_accuracy_train.append(accuracy(X_train,w_sgd,y_train))    #计算训练和测试的准确率
        sgd_accuracy_test.append(accuracy(X_test,w_sgd,y_test))
    return sgd_train,sgd_test,sgd_accuracy_train,sgd_accuracy_test
#参数初始化
eta=0.00005                      #学习率设为0.008
train_size=0.0005               #指用于训练SGD的样本数占X_train的比例,约有15个样本用于训练
C=1
#开始训练，得到训练误差，测试误差，训练准确率和测试准确率
sgd_train,sgd_test,sgd_accuracy_train,sgd_accuracy_test=SGD(X=X_train,y=y_train,w_sgd=w_sgd,eta=eta,train_size=train_size,epco=epco,C=C)
```

```
#定义NAG sgd的训练函数
def NAG(X,y,w_nag,v_nag,gamma,eta,train_size,C,gradient=NAG_gradient,loss=hingeloss,accuracy=accuracy):
    nag_train=[]                  #训练误差列表
    nag_test=[]                   #测试误差列表
    nag_accuracy_train=[]         #训练准确率列表
    nag_accuracy_test=[]          #测试准确率列表
    for i in range(epco):
        #得到用于随机梯度下降训练的小部分样本  (即X_trainset,y_trainset)
        X_trainset,X_other,y_trainset,y_other=train_test_split(X,y,test_size=1-train_size,random_state=random.randint(0,1000))
        #调用NG_gradient,每一次迭代进行一次梯度下降
        w_nag,v_nag=gradient(X_trainset,y_trainset,w_nag,v_nag,gamma,eta,C)
        nag_train.append(loss(X_trainset,y_trainset,w_nag))              #得到训练和测试的loss
        nag_test.append(loss(X_test,y_test,w_nag))
        nag_accuracy_train.append(accuracy(X_trainset,w_nag,y_trainset))  #计算训练和测试的准确率
        nag_accuracy_test.append(accuracy(X_test,w_nag,y_test))
    return nag_train,nag_test,nag_accuracy_train,nag_accuracy_test
#参数初始化
gamma=0.9
eta=0.00001                                                  #学习率设为0.001
train_size=0.0005                                            #指用于训练SGD的样本数占X_train的比例
C=1
#开始训练，得到训练误差，测试误差，训练准确率和测试准确率
nag_train,nag_test,nag_accuracy_train,nag_accuracy_test=NAG(X=X_train,y=y_train,w_nag=w_nag,v_nag=v_nag,gamma=gamma,eta=eta,train_size=train_si
```

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



Result analysis:

1. From the graph we find that AdaDelta, RMSprop, Adam perform almost when solving SVM, but AdaDelta also has obvious shaking at end than others that is because its biggest characteristic is self-adaptive and high-speeding.
2. RMSprop is a variant of AdaDelta, slightly less effective than AdaDelta. From the graph, we find that it have a more stable curve than AdaDelta. That means it is more suitable for handling non-stationary targets
3. Adam is essentially RMSprop with momentum terms. It has a closely same result as RMSprop. From its formula,it needs less memory requirement and calculate different adaptive learning rates for different parameters
4. By contrast, SGD is slowest one for which it use typically learning rate and gradient to update w.It plays a more stable curve then three algorithms above while it has a slow convergence speed. NAG plays relatively well than SGD. And It's curve is also more stable more first three algorithm.
5. Four methods reaches optimal solution far faster than traditional GD because they only randomly use a set of training set to train model.

D.Similarities and differences between linear regression and linear classification:

1) Similarities:The fundamental purpose of both is the same. Logistic regression is a classifier, not really regression. The purpose of these two loss functions is to increase the weight of the data points that have a greater impact on the classification and reduce the weight of the data points that have a smaller relationship with the classification.SVM processing method only considers support vectors.And the logistic regression through nonlinear Mapping significantly reduces the weight of points farther from the classification plane and relatively increases the weight of the data points most relevant to the classification.
2) Differences:
   a) From the objective function point of view, the difference is that logistic loss is used in logistic regression, svm uses hinge loss.
   b) SVM more belongs to the non-parametric model, and logistic regression is a parameter model so that they are essentially different.
   c) The major difference between them is the way they evaluate final super plane.

IV. CONCLUSION

1. SGD improves traditional GD with a faster converging speed and considerable result. SGD has more performance while deal with a large data set.
2. Four specific methods are different improvement of plain SGD with respective advantages. From this experiment, I can summaries some features of these four algorithms whild solving the classification.
   a) SGD generally takes longer to train, but with good initialization and learning rate scheduling schemes, the results are more reliable
   b) Adadelta, RMSprop, Adam are relatively similar algorithms that perform similarly under similar conditions. As an adaptive algorithm, AdaDelta tends to be more convergent and prone to frequent jitter
   c) RMSprop and Adam are driven by momentum, often with relatively fast and stable performance
3. The four kinds of optimization algorithms of SGD have different performance in solving the classification problem, but all of them can speed up the convergence speed and reach a lower error
4. 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.SVM more belongs to the non-parametric model, and logistic regression is a parameter model so that they are essentially different.