

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**  **吴伟民**

**学 号 201530613061**

**邮 箱 2727709668@qq.com**

**指导教师**  **吴庆耀**

**提交日期** **2017年 12月 8日**

## 1. 实验题目: 线性回归、线性分类与梯度下降

## 2. 实验时间：2017年 12月 2日

## 3. 报告人:吴伟民

## 4. 实验目的:

进一步理解线性回归和梯度下降的原理。

在小规模数据集上实践。

体会优化和调参的过程。

## 数据集以及数据分析：

线性回归使用的是LIBSVM Data中的Housing数据，包含506个样本，每个样本有13个属性。请自行下载scaled版本，并将其切分为训练集，验证集。

线性分类使用的是LIBSVM Data中的australian数据，包含690个样本，每个样本有14 个属性。请自行下载scaled版本，并将其切分为训练集，验证集。

## 实验步骤:



## 代码内容:

线性回归与梯度下降

class LinearRegressionByMyself(object):

def \_\_init\_\_(self,Learning\_rate=0.001,epoch=8):

self.Learning\_rate=Learning\_rate

self.epoch=epoch

def fit(self,x,y):

self.w=np.zeros(1+x.shape[1])

self.cost\_list=[]

self.row=x.shape[0]

for i in range(self.epoch):

output=self.Regression\_input(x)

error=(y-output)

self.w[1:]+=self.Learning\_rate\*x.T.dot(error)

self.w[0]+=self.Learning\_rate\*error.sum()

cost=(error\*\*2).sum()/2.0

self.cost\_list.append(cost)

return self

def Regression\_input(self,x):

#f=self.w[0]+self.w[1:]

#print('self.w[1:]:',self.w[1:])

#print('f:',f)

#print('self.w[0]:',self.w[0])

#print('x\*w:',x[0].dot(self.w[1:]))

return x.dot(self.w[1:])+self.w[0]

线性分类与梯度下降

class LinearClassification(object):

def \_\_init\_\_(self,Learning\_rate=0.001,threshold=0.0,epoch=8):

self.Learning\_rate=Learning\_rate

self.epoch=epoch

def fit(self,x,y,threshold,C):

self.w=np.zeros((1,1+x.shape[1]))

self.cost\_list=[]

self.threshold=threshold

self.C=C

#对每个x进行计算

for i in range(self.epoch):

cost=0

for j in range(x.shape[0]):

output=self.Classification\_input(x[j,:],self.threshold)

if(y[j]!=output):

#计算y\*x

t=self.C\*y[j]\*x[j,:]

#计算梯度

dw=self.w[0,1:]-t

#计算cost

cost+=((1-y[j]\*output)\*self.C)

#更新w

self.w[0,1:]=self.w[0,1:]+self.Learning\_rate\*dw

self.w[0,0]=self.w[0,0]+self.Learning\_rate\*(self.w[0,0]-self.C\*y[j])

self.w1=self.w.T

cost+=((self.w1\*self.w.T).sum())/2

self.cost\_list.append(cost)

return self

#计算每个f(X)

def Classification\_input(self,x,threshold):

f=x\*self.w[0,1:]+self.w[0,0]

if(f>=threshold):

return 1

else:

return -1

（针对线性回归和线性分类分别填写8-12内容）

## 选择的评估方法（留出法，交叉验证，k折交叉验证等）:

## 留一验证法

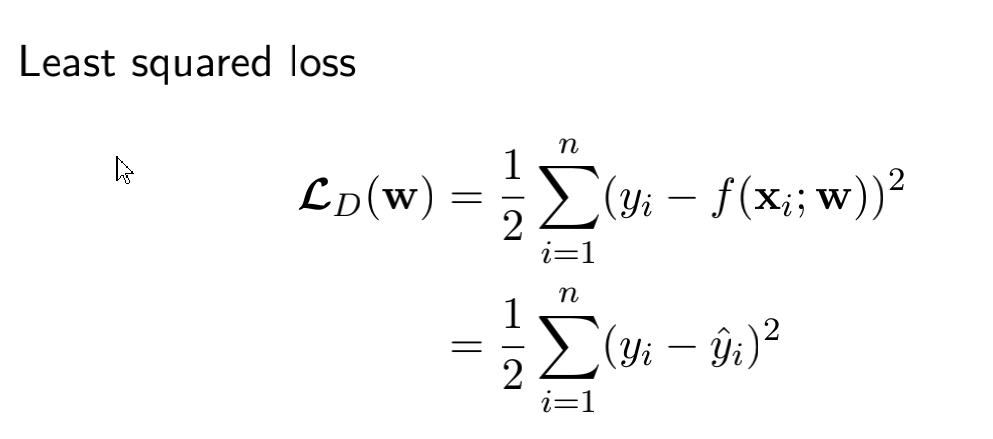
## 模型参数的初始化方法:

全零初始化

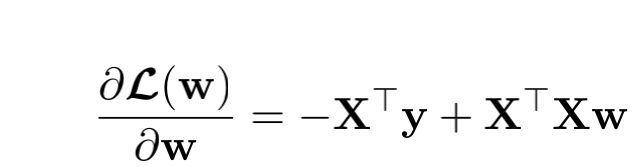
## 选择的loss函数及其导数:

线性回归与梯度下降

Loss函数

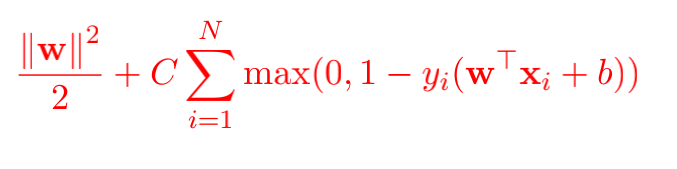


导数



线性分类与梯度下降

Loss函数



导数

2

## 11.实验结果和曲线图:

## 超参数选择（η,epoch等）：

线性回归

η=0.001

epoch=8

线性分类

η=0.001

epoch=8

C=0.9

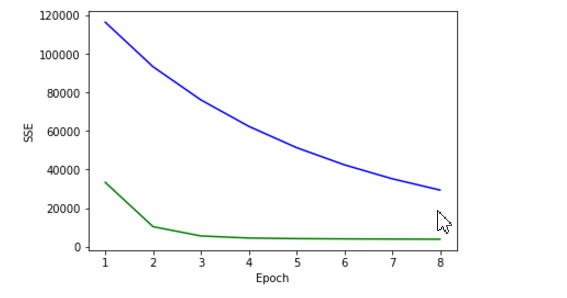
## 评估结果（根据选择的评估方法）：

达到80%拟合

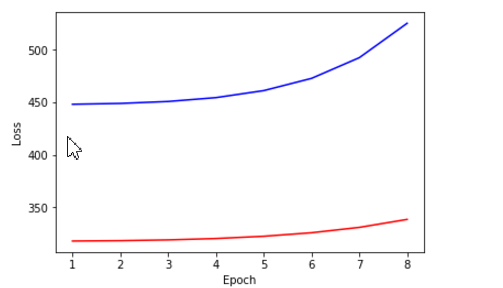
## 预测结果（最佳结果）：

## loss曲线图：

线性回归



线性分类



## 实验结果分析:

线性回归随着迭代次数增加，损失值变小，线性分类则是相反

## 对比线性回归和线性分类的异同点：

线性回归的幅度相对比线性分类的幅度大，而且前者会随迭代次数增加而减少损失值，后者则相反

相同点是他们模型函数都是线性的

## 14.实验总结：

通过动手操作并且自己独立分析，对这些算法有更深入的了解，但是需要花更多时间去调节好参数

## 1. 实验题目: 逻辑回归、线性分类与随机梯度下降

## 2. 实验时间：2017年 12月 2日

## 3. 报告人:吴伟民

## 4. 实验目的:

对比理解梯度下降和随机梯度下降的区别与联系。

对比理解逻辑回归和线性分类的区别与联系。

进一步理解SVM的原理并在较大数据上实践。

## 数据集以及数据分析：

实验使用的是LIBSVM Data的中的a9a数据，包含32561 / 16281(testing)个 样本，每个样本有123/123 (testing)个属性。请自行下载训练集和验证集。

## 实验步骤:



## 代码内容:

线性分类

class LinearClassification(object):

def \_\_init\_\_(self,Learning\_rate=0.001,threshold=0.0,epoch=8):

self.Learning\_rate=Learning\_rate

self.epoch=epoch

def fit\_nag(self,x,Y,threshold,C):

self.w=np.zeros((1,1+x.shape[1]))

self.v=np.zeros((1,x.shape[1]))

self.cost\_list=[]

self.threshold=threshold

self.C=C

self.y=0.9

for i in range(self.epoch):

cost=0

for j in range(x.shape[0]):

output=self.Classification\_input(x[j,:],self.threshold)

if(output!=Y[j]):

#计算动量项

t=Y[j]\*(x[j,:])

#print('t.shape:',t.shape)

#print('self.w.shape:',self.w.shape)

#print('self.v.shape:',self.v.shape)

#print('self.y.shape:',self.y.shape)

dwv=-self.C\*t+self.w[0,1:]-self.y\*self.v

self.v=self.y\*self.v+self.Learning\_rate\*dwv

#计算cost值

cost+=(1-Y[j]\*output)

#print('cost:',cost)

#更新w

self.w[0,1:]=self.w[0,1:]-self.v

self.w[0,0]=self.w[0,0]-self.v.sum()

self.w1=self.w.T

cost=cost+(self.w\*self.w1).sum()/2

self.cost\_list.append(cost)

return self

def fit\_adadelta(self,x,Y,threshold,C):

self.e=10\*\*(-8)

self.cost\_list=[]

self.w=np.zeros((1,1+x.shape[1]))

self.v=np.zeros((1,1+x.shape[1]))

self.threshold=threshold

self.C=C

self.y=0.9

for i in range(self.epoch):

cost=0

for j in range(x.shape[0]):

output=self.Classification\_input(x[j,:],self.threshold)

if(output!=Y[j]):

#计算梯度gt

t=Y[j]\*(x[j,:])

g=self.w[0,1:]-self.C\*t

#计算E[g^2]t

g1=g.T

E=(g\*g1).sum()

#计算RMS[g]t

RMSgt=np.sqrt(self.e+E)

#计算dθt,保留学习率

dw=(-self.Learning\_rate/RMSgt)\*g

#计算E[dw^2]t-1

dw1=dw.T

Edwt=(dw\*dw1).sum()

#计算RMS[dw]t-1

RMSwt1=np.sqrt(Edwt+self.e)

#计算dθt,去除学习率

dwt=-(RMSwt1/RMSgt)\*g

#计算E[dw^2]t,dw1是dw转置

dw1=dw.T

Edwt=self.y\*Edwt+(1-self.y)\*((dw\*dw1).sum())

#计算cost

cost+=(1-Y[j]\*output)

#更新w

self.w[0,1:]=self.w[0,1:]+dwt

self.w[0,0]=self.w[0,0]+dwt.sum()

self.w1=self.w.T

cost=cost+(self.w\*self.w1).sum()/2

self.cost\_list.append(cost)

return self

def fit\_rms(self,x,Y,threshold,C):

self.e=10\*\*(-8)

self.cost\_list=[]

self.w=np.zeros((1,1+x.shape[1]))

self.v=np.zeros((1,1+x.shape[1]))

self.threshold=threshold

self.C=C

self.y=0.9

for i in range(self.epoch):

cost=0

for j in range(x.shape[0]):

output=self.Classification\_input(x[j,:],self.threshold)

if(output!=Y[j]):

#计算梯度gt

t=Y[j]\*(x[j,:])

g=self.w[0,1:]-self.C\*t

#计算E[g^2]t

g1=g.T

E=(g\*g1).sum()

#计算RMS[g]t

RMSgt=np.sqrt(self.e+E)

#计算dw

dw=(-self.Learning\_rate/RMSgt)\*g

#计算cost

cost+=(1-Y[j]\*output)

#更新w

self.w[0,1:]=self.w[0,1:]+dw

self.w[0,0]=self.w[0,0]+dw.sum()

self.w1=self.w.T

cost=cost+(self.w\*self.w1).sum()/2

self.cost\_list.append(cost)

return self

def fit\_Adam(self,x,Y,threshold,C):

self.e=10\*\*(-8)

self.cost\_list=[]

self.w=np.zeros((1,1+x.shape[1]))

self.m=np.zeros((1,x.shape[1]))

self.v=0

self.C=C

self.threshold=threshold

self.p1=0.9

self.p2=0.999

for i in range(self.epoch):

cost=0

for j in range(x.shape[1]):

output=self.Classification\_input(x[j,:],self.threshold)

if(output!=Y[i]):

#计算梯度gt

t=Y[j]\*(x[j,:])

g=self.w[0,1:]-self.C\*t

#计算m

self.m=self.m\*self.p1+(1-self.p1)\*g

#计算m的修正误差M

M=self.m/(1-(self.p1\*\*(j+1)))

#计算v

g1=g.T

G=(g1\*g).sum()

self.v=self.v\*self.p2+(1-self.p2)\*G

#计算V

V=self.v/(1-(self.p2\*\*(j+1)))

#计算cost

cost+=(1-Y[j]\*output)

#计算W

self.w[0,1:]=self.w[0,1:]-(self.Learning\_rate/(np.sqrt(V)+self.e))\*M

self.w[0,0]=self.w[0,0]-((self.Learning\_rate/(np.sqrt(V)+self.e))\*M).sum()

self.w1=self.w.T

cost=cost+(self.w\*self.w1).sum()/2

self.cost\_list.append(cost)

return self

def Classification\_input(self,x,threshold):

#print('x.shape:',x.shape)

#print('w[0,1:].shape:',self.w[0,1:].shape)

f=x\*self.w[0,1:]+self.w[0,0]

if(f>=threshold):

return 1

else:

return -1

逻辑回归

class LogisticRegression(object):

def \_\_init\_\_(self,Learning\_rate=0.001,epoch=6):

self.Learning\_rate=Learning\_rate

self.epoch=epoch

self.threshold=0.5

self.y=0.9

def fit\_NAG(self,x,Y):

self.w=np.zeros((1,x.shape[1]))

self.cost\_list=[]

self.v=np.zeros((1,x.shape[1]))

for i in range(self.epoch):

cost=0

for j in range(x.shape[0]):

output=self.Logistic\_input(x[j,:],self.threshold)

if(output!=Y[j]):

t=-Y[j]\*(x[j,:])

dwv=self.w-self.y\*self.v

t=t.T.dot(dwv)

t=t.sum()

dv=self.w-(Y[j]\*(x[j,:]))/(1+np.exp(t))

self.v\*=self.y

self.v+=self.Learning\_rate\*dv

self.w=self.w-self.v

cost+=np.log(1+np.exp((-Y[j]\*x[j,:].T.dot(self.w)).sum()))

cost=cost/x.shape[0]+(self.w\*\*2).sum()/2

self.cost\_list.append(cost)

#print('self.cost\_list:',self.cost\_list)

return self

def fit\_Rms(self,x,Y):

self.e=10\*\*(-8)

self.cost\_list=[]

self.w=np.zeros((1,x.shape[1]))

for i in range(self.epoch):

cost=0

for j in range(x.shape[0]):

output=self.Logistic\_input(x[j,:],self.threshold)

if(output!=Y[j]):

#计算梯度gt

t=Y[j]\*(x[j,:])

h=self.Logistic\_input(t,self.threshold)

g=self.w-t\*h

#计算E[g^2]t

g1=g.T

E=g\*g1

E=E.sum()

#计算RMS[g]t

RMSgt=np.sqrt(self.e+E)

#计算dw

dw=(-self.Learning\_rate/RMSgt)\*g

#更新w

self.w=self.w+dw

#计算cost

cost+=np.log(1+np.exp((-Y[j]\*x[j,:].T.dot(self.w)).sum()))

self.w1=self.w.T

cost=cost/x.shape[0]+(self.w\*self.w1).sum()/2

self.cost\_list.append(cost)

return self

def fit\_Adadelta(self,x,Y):

self.e=10\*\*(-8)

self.cost\_list=[]

self.w=np.zeros((1,x.shape[1]))

self.v=np.zeros((1,x.shape[1]))

for i in range(self.epoch):

cost=0

for j in range(x.shape[0]):

output=self.Logistic\_input(x[j,:],self.threshold)

if(output!=Y[i]):

#计算梯度gt

t=Y[j]\*(x[j,:])

h=self.Logistic\_input(t,self.threshold)

g=self.w-t\*h

#计算E[g^2]t

g1=g.T

E=g\*g1

E=E.sum()

#print('E:',E)

#E=np.mean(E)

#计算RMS[g]t

RMSgt=np.sqrt(self.e+E)

#计算dθt,保留学习率

dw=(-self.Learning\_rate/RMSgt)\*g

#计算E[dw^2]t-1

dw1=dw.T

Edwt=dw\*dw1

Edwt=Edwt.sum()

#Edwt=np.mean(Edwt)

#计算RMS[dw]t-1

RMSwt1=np.sqrt(Edwt+self.e)

#计算dθt，去掉学习率

dwt=-(RMSwt1/RMSgt)\*g

#计算E[dw^2]t,dw1是dw转致

dw1=dw.T

Edwt=self.y\*Edwt+(1-self.y)\*((dw\*dw1).sum())

#Wt+1=Wt+dWt，更新w

#print(self.w.shape)

#print('dwt.shape:',dwt.shape)

self.w=self.w+dwt

#print('self.w.shape:',self.w.shape)

#print('self.x[j,:].shape:',x[j,:].shape)

#计算cost

cost+=np.log(1+np.exp((-Y[j]\*x[j,:].T.dot(self.w)).sum()))

#print(cost)

self.w1=self.w.T

cost=cost/x.shape[0]+(self.w\*self.w1).sum()/2

self.cost\_list.append(cost)

return self

def fit\_Adam(self,x,Y):

self.e=10\*\*(-8)

self.cost\_list=[]

self.w=np.zeros((1,x.shape[1]))

self.m=np.zeros((1,x.shape[1]))

self.v=0

self.p1=0.900

self.p2=0.999

for i in range(self.epoch):

cost=0

for j in range(x.shape[1]):

output=self.Logistic\_input(x[j,:],self.threshold)

if(output!=Y[i]):

#计算梯度gt

t=Y[j]\*(x[j,:])

h=self.Logistic\_input(t,self.threshold)

g=self.w-t\*h

#计算m

self.m=self.m\*self.p1+(1-self.p1)\*g

#计算m的修正误差M

M=self.m/(1-(self.p1\*\*(j+1)))

#计算v

g1=g.T

G=(g1\*g).sum()

self.v=self.v\*self.p2+(1-self.p2)\*G

#计算V

V=self.v/(1-(self.p2\*\*(j+1)))

#计算cost

cost+=np.log(1+np.exp((-Y[j]\*x[j,:].T.dot(self.w)).sum()))

#计算W

self.w=self.w-(self.Learning\_rate/(np.sqrt(V)+self.e))\*M

self.w1=self.w.T

cost=cost/x.shape[0]+(self.w\*self.w1).sum()/2

self.cost\_list.append(cost)

return self

def Logistic\_input(self,x,threshold):

t=x.T.dot(self.w)

t=t.sum()

h=1/(1+np.exp(t))

if(h<self.threshold):

return -1

else:

return 1

（针对逻辑回归和线性分类分别填写8-11内容）

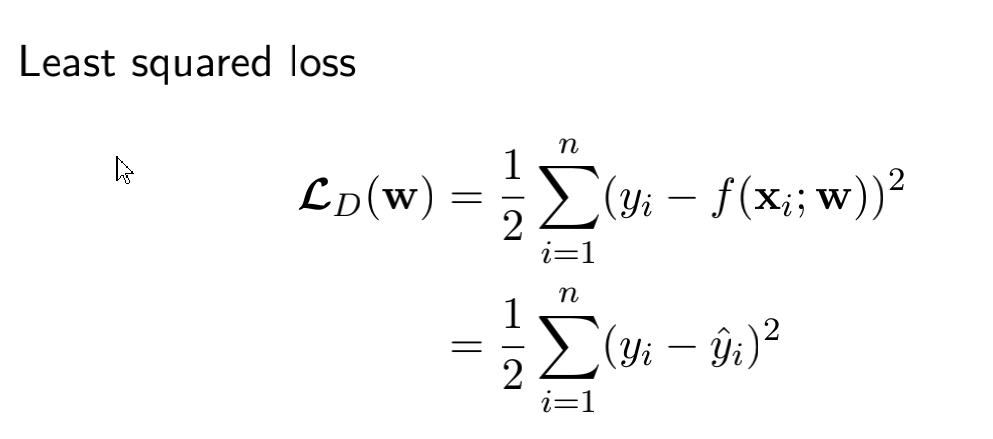
## 模型参数的初始化方法:

全零初始化

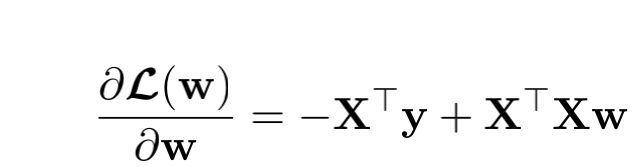
## 选择的loss函数及其导数:

线性分类

Loss函数

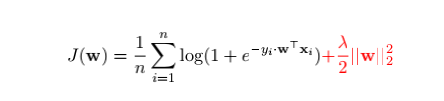


导数

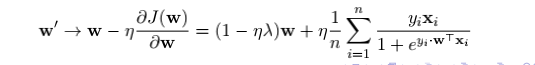


逻辑回归

Loss函数



导数



## 10.实验结果和曲线图:（各种梯度下降方式分别填写此项）

## 超参数选择：

学习率设为0.001

Epoch为6

C为0.9

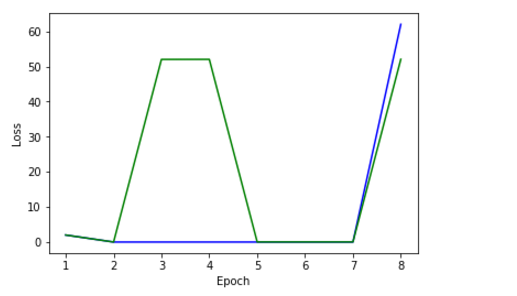
## 评估结果（根据选择的评估方法）：

## 预测结果（最佳结果）：

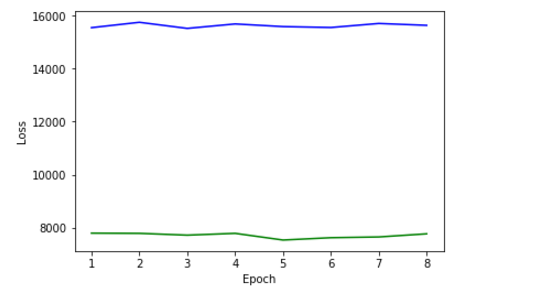
## loss曲线图：

线性分类

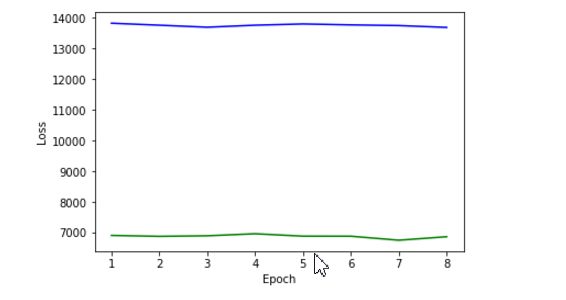
Adam



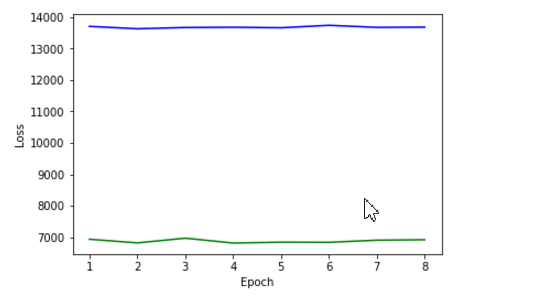
NAG



Adadelta

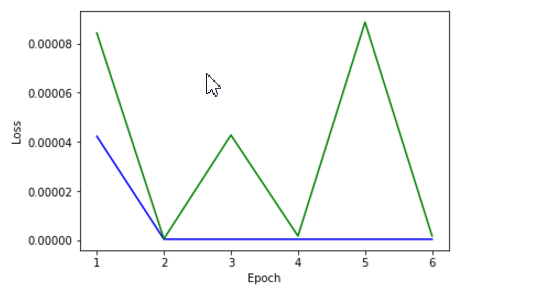


Rmsprop

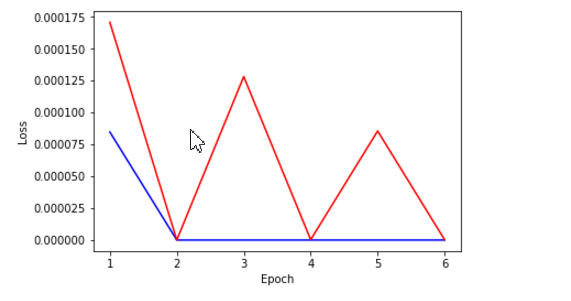


逻辑回归

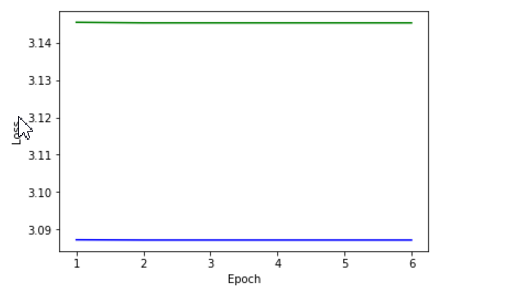
Adadelta



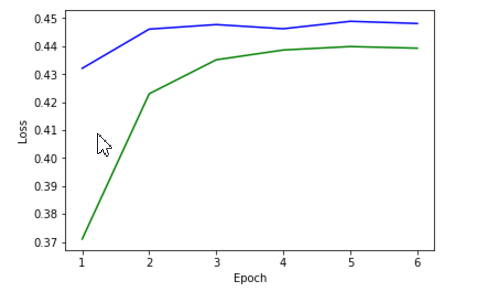
Adam



NAG



RMSprop



## 实验结果分析:

在线性分类中，通过adam算法可以有效把验证集合测试集的loss函数相互靠拢，其他三个优化算法验证集和测试集loss靠拢程度不是很高，需要减少参数

在逻辑回归中，通过adadelta和adam，RMSprop可以做到相互靠拢，NAG算法还需要减少参数来使两者更加靠龙

## 12.对比逻辑回归和线性分类的异同点：

逻辑回归相对来说模型更简单,好理解,实现起来,特别是大规模线性分类时比较方便.而SVM的理解和优化相对来说复杂一些,SVM转化为对偶问题后,分类只需要计算与少数几个支持向量的距离,这个在进行复杂核函数计算时优势很明显,能够大大简化模型和计算

相同点他们都根据阈值给出函数值

## 13.实验总结：

通过动手操作并且自己独立分析，对这些算法有更深入的了解，但是需要花更多时间去调节好参数