

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**  **肖亮昱**

**学 号 201530613146**

**邮 箱 624198747@qq.com**

**指导教师**  **吴庆耀**

**提交日期** **2017年 12月7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 9 日

## 3. 报告人:肖亮昱

## 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。

## 5. 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。

观察数据集我们可以看出，训练集共有123个属性，整个数据集是一个稀疏矩阵，而测试集则有122个属性，按照稀疏矩阵的格式，说明测试集上第123个属性的值为零，故在读取数据时要对测试数据进行额外处理，即增加第123行属性值，其值为0。

同时在数据集上引入一个为1的属性来引入bias，这样，数据集一共具有124个属性

## 6. 实验步骤:

### 逻辑回归与随机梯度下降

1. 读取实验训练集和验证集。
2. 逻辑回归模型参数初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。
3. 选择Loss函数及对其求导，过程详见课件ppt。
4. 求得部分样本对Loss函数的梯度。
5. 使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta和Adam）。
6. 选择合适的阈值，将验证集中计算结果大于阈值的标记为正类，反之为负类。在验证集上测试并得到不同优化方法的Loss函数值。
7. 重复步骤4-6若干次，画出Loss和随迭代次数的变化图。

### 线性分类与随机梯度下降

1. 读取实验训练集和验证集。
2. 支持向量机模型参数初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。
3. 选择Loss函数及对其求导，过程详见课件ppt。
4. 求得部分样本对Loss函数的梯度。
5. 使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta和Adam）。
6. 选择合适的阈值，将验证集中计算结果大于阈值的标记为正类，反之为负类。在验证集上测试并得到不同优化方法的Loss函数值。
7. 重复步骤4-6若干次，画出Loss和随迭代次数的变化图。

## 7. 代码内容:

**逻辑回归与随机梯度下降**

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

"""

-------------------------------------------------

File Name： LogisticRegressionWithSGD

Description :

Author : Nathan

date： 2017/12/4

-------------------------------------------------

Change Activity:

2017/12/4:

-------------------------------------------------

"""

\_\_author\_\_ = 'Nathan'

from sklearn.datasets import load\_svmlight\_file

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

import numpy as np

import matplotlib.pyplot as plt

import math

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

feature\_num=123

batch\_size=128

SGD\_methods=["SGD","NAG","RMSProp","AdaDelta","Adam"]

parm={"SGD":{"learning rate":0.01},\

"NAG":{"learning rate":0.01,"Gamma":0.9},\

"RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},\

"AdaDelta":{"Gamma":0.95,"Epsilon":10e-6},\

"Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}}

temp\_list={"NAG":np.zeros([feature\_num + 1, 1]),\

"RMSProp":np.zeros([feature\_num+1,1]),\

"AdaDelta":{"EG":np.zeros([feature\_num+1,1]),"EX":np.zeros([feature\_num+1,1])},\

"Adam":{"M":np.zeros([feature\_num+1,1]),"G":np.zeros([feature\_num+1,1]),"t":0}}

def sigmoid(z):

return 1/(1+np.exp(-1.0\*z))

def compute\_loss(W, X\_test,y\_test):

logits = np.matmul(X\_test, W)

loss = -np.mean(y\_test \* np.log(sigmoid(logits)) + (1 - y\_test) \* np.log(1 - sigmoid(logits)))

return loss

def compute\_gradient(W,X\_train,y\_train):

logits = np.matmul(X\_train, W)

output = sigmoid(logits)

error = output - y\_train

gradient = np.matmul(X\_train.transpose(), error) / y\_train.shape[0]

return gradient

def SGD(W,X\_train,y\_train):

W-=parm.get("SGD").get("learning rate")\*compute\_gradient(W,X\_train,y\_train)

return W

def NAG(W,X\_train,y\_train):

global temp\_list

global parm

momentum=temp\_list.get('NAG')

#learning\_rate=parm.get("NAG").get("learning\_rate")

Gamma=parm.get("NAG").get("Gamma")

gradient=compute\_gradient(W-(Gamma\*momentum),X\_train,y\_train)

update\_momentum = momentum \* Gamma+ gradient \* parm.get("NAG").get("learning rate")

temp\_list["NAG"]=update\_momentum

W-=update\_momentum

return W

def RMSProp(W,X\_train,y\_train):

G=temp\_list.get("RMSProp")

Gamma =parm.get("RMSProp").get("Gamma")

Epsilon=parm.get("RMSProp").get("Epsilon")

learning\_rate=parm.get("RMSProp").get("learning rate")

gradient=compute\_gradient(W,X\_train,y\_train)

G=G+(1-Gamma)\*gradient\*\*2

temp\_list["RMSProp"]=G

W-=learning\_rate\*gradient/np.sqrt(G+Epsilon)

return W

def AdaDelta(W,X\_train,y\_train):

EG=temp\_list.get("AdaDelta").get("EG")

EX=temp\_list.get("AdaDelta").get("EX")

Gamma=parm.get("AdaDelta").get("Gamma")

Epsilon=parm.get("AdaDelta").get("Epsilon")

gradient=compute\_gradient(W,X\_train,y\_train)

EG=Gamma\*EG+(1-Gamma)\*gradient\*\*2

temp\_list.get("AdaDelta")["EG"]=EG

delta=-1\*gradient\*np.sqrt(EX+Epsilon)/np.sqrt(EG+Epsilon)

EX=Gamma\*EX+(1-Gamma)\*delta\*\*2

temp\_list.get("AdaDelta")["EX"]=EX

W+=delta

return W

def Adam(W,X\_train,y\_train):

Beta=parm.get("Adam").get("Beta")

Gamma=parm.get("Adam").get("Gamma")

Epsilon=parm.get("Adam").get("Epsilon")

learning\_rate=parm.get("Adam").get("learning rate")

M=temp\_list.get("Adam").get("M")

G=temp\_list.get("Adam").get("G")

t=temp\_list.get("Adam").get("t")

t=t+1

temp\_list.get("Adam")["t"]=t

gradient=compute\_gradient(W,X\_train,y\_train)

M=Beta\*M+(1-Beta)\*gradient

temp\_list.get("Adam")["M"]=M

G=Gamma\*G+(1-Gamma)\*gradient\*\*2

temp\_list.get("Adam")["G"]=G

M\_bias=M/(1-Beta\*\*t)

G\_bias=G/(1-Gamma\*\*t)

W-=learning\_rate\*M\_bias/(np.sqrt(G\_bias)+Epsilon)

return W

def opitimizer(W,X\_train,y\_train,method):

if method=="SGD":

return SGD(W,X\_train,y\_train)

if method=="NAG":

return NAG(W,X\_train,y\_train)

if method=="RMSProp":

return RMSProp(W,X\_train,y\_train)

if method=="AdaDelta":

return AdaDelta(W,X\_train,y\_train)

if method=="Adam":

return Adam(W,X\_train,y\_train)

def getdata():

X\_train, y\_train = load\_svmlight\_file(r'C:\Users\jy\Desktop\Libsvmdata\a9a.txt')

datasize,features=X\_train.shape

X\_train=np.c\_[np.ones(len(X\_train.toarray())), X\_train.toarray()]

for i in range(0, len(y\_train)):

if y\_train[i] == -1:

y\_train[i] = 0

X\_test,y\_test=load\_svmlight\_file(r'C:\Users\jy\Desktop\Libsvmdata\a9a\_test.txt')

X\_test=np.c\_[X\_test.toarray(),np.zeros(len(X\_test.toarray()))]

X\_test=np.c\_[np.ones(len(X\_test)),X\_test]

for i in range(0, len(y\_test)):

if y\_test[i] == -1:

y\_test[i] = 0

y\_train = y\_train.reshape([len(y\_train), 1])

y\_test = y\_test.reshape([len(y\_test), 1])

X\_train,y\_train=shuffle(X\_train,y\_train)

X\_test,y\_test=shuffle(X\_test,y\_test)

return X\_train,y\_train,X\_test,y\_test,datasize,features

def get\_sub\_batch(batch\_count,X,y,data\_size):

if (1+batch\_count)\*batch\_size<=data\_size:

return X[batch\_count\*batch\_size:(batch\_count + 1) \* batch\_size],y[batch\_count\*batch\_size:(batch\_count + 1) \* batch\_size]

else:

return X[batch\_count\*batch\_size:data\_size],y[batch\_count\*batch\_size:data\_size]

def shuffle(X,y):

rng\_state = np.random.get\_state()

np.random.shuffle(X)

np.random.set\_state(rng\_state)

np.random.shuffle(y)

return X,y

def LogisicRegressionModel():

X\_train, y\_train, X\_test, y\_test, data\_size, features\_num = getdata()

plt.xlabel('Iteration')

plt.ylabel('Loss')

for method in SGD\_methods:

W = np.random.rand(features\_num + 1, 1)

iter\_ = []

error = []

num = 0

for j in range(2):

for i in range(0, int(data\_size / batch\_size ) + 1):

iter\_.append(num)

X,y=get\_sub\_batch(i,X\_train,y\_train,data\_size)

W=opitimizer(W,X,y,method)

error.append(compute\_loss(W,X\_test,y\_test))

num+=1

plt.plot(iter\_, error, label=method)

plt.legend()

plt.show()

LogisicRegressionModel()

**线性分类**：

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

"""

-------------------------------------------------

File Name： LinearClassificationWIthSGD

Description :

Author : Nathan

date： 2017/12/9

-------------------------------------------------

Change Activity:

2017/12/9:

-------------------------------------------------

"""

\_\_author\_\_ = 'Nathan'

\_\_author\_\_ = 'Nathan'

from sklearn.datasets import load\_svmlight\_file

import numpy as np

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

feature\_num=123

batch\_size=512

SGD\_methods=["SGD","NAG","RMSProp","AdaDelta","Adam"]

parm={"C":0.9,\

"SGD":{"learning rate":0.01},\

"NAG":{"learning rate":0.005,"Gamma":0.9},\

"RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},\

"AdaDelta":{"Gamma":0.95,"Epsilon":10e-6},\

"Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}}

temp\_list={"NAG":np.zeros([feature\_num + 1, 1]),\

"RMSProp":np.zeros([feature\_num+1,1]),\

"AdaDelta":{"EG":np.zeros([feature\_num+1,1]),"EX":np.zeros([feature\_num+1,1])},\

"Adam":{"M":np.zeros([feature\_num+1,1]),"G":np.zeros([feature\_num+1,1]),"t":0}}

def sigmoid(z):

return 1/(1+np.exp(-1.0\*z))

def compute\_loss(W, X\_test,y\_test):

L = 0.

N = y\_test.shape[0]

temp=1-y\_test\*np.matmul(X\_test,W)

L = sum(np.maximum(0, temp))#hinge loss

loss =0.5 \* np.matmul(W.T, W)[0][0] + (L \* parm.get("C"))/N

return loss

def compute\_gradient(W,X\_train,y\_train):

L\_dW = np.zeros((124,1))

temp=1-y\_train\*np.matmul(X\_train,W)

temp=np.maximum(temp/np.abs(temp),0)

y=y\_train\*temp

L\_dW=-np.matmul(X\_train.T,y)

return (parm.get("C") \* L\_dW) + W

def SGD(W,X\_train,y\_train):

W-=parm.get("SGD").get("learning rate")\*compute\_gradient(W,X\_train,y\_train)

return W

def NAG(W,X\_train,y\_train):

global temp\_list

global parm

momentum=temp\_list.get('NAG')

#learning\_rate=parm.get("NAG").get("learning\_rate")

Gamma=parm.get("NAG").get("Gamma")

gradient=compute\_gradient(W-(Gamma\*momentum),X\_train,y\_train)

update\_momentum = momentum \* Gamma+ gradient \* parm.get("NAG").get("learning rate")

temp\_list["NAG"]=update\_momentum

W-=update\_momentum

return W

def RMSProp(W,X\_train,y\_train):

G=temp\_list.get("RMSProp")

Gamma =parm.get("RMSProp").get("Gamma")

Epsilon=parm.get("RMSProp").get("Epsilon")

learning\_rate=parm.get("RMSProp").get("learning rate")

gradient=compute\_gradient(W,X\_train,y\_train)

G=G+(1-Gamma)\*gradient\*\*2

temp\_list["RMSProp"]=G

W-=learning\_rate\*gradient/np.sqrt(G+Epsilon)

return W

def AdaDelta(W,X\_train,y\_train):

EG=temp\_list.get("AdaDelta").get("EG")

EX=temp\_list.get("AdaDelta").get("EX")

Gamma=parm.get("AdaDelta").get("Gamma")

Epsilon=parm.get("AdaDelta").get("Epsilon")

gradient=compute\_gradient(W,X\_train,y\_train)

EG=Gamma\*EG+(1-Gamma)\*gradient\*\*2

temp\_list.get("AdaDelta")["EG"]=EG

delta=-1\*gradient\*np.sqrt(EX+Epsilon)/np.sqrt(EG+Epsilon)

EX=Gamma\*EX+(1-Gamma)\*delta\*\*2

temp\_list.get("AdaDelta")["EX"]=EX

W+=delta

return W

def Adam(W,X\_train,y\_train):

Beta=parm.get("Adam").get("Beta")

Gamma=parm.get("Adam").get("Gamma")

Epsilon=parm.get("Adam").get("Epsilon")

learning\_rate=parm.get("Adam").get("learning rate")

M=temp\_list.get("Adam").get("M")

G=temp\_list.get("Adam").get("G")

t=temp\_list.get("Adam").get("t")

t=t+1

temp\_list.get("Adam")["t"]=t

gradient=compute\_gradient(W,X\_train,y\_train)

M=Beta\*M+(1-Beta)\*gradient

temp\_list.get("Adam")["M"]=M

G=Gamma\*G+(1-Gamma)\*gradient\*\*2

temp\_list.get("Adam")["G"]=G

M\_bias=M/(1-Beta\*\*t)

G\_bias=G/(1-Gamma\*\*t)

W-=learning\_rate\*M\_bias/(np.sqrt(G\_bias)+Epsilon)

return W

def opitimizer(W,X\_train,y\_train,method):

if method=="SGD":

return SGD(W,X\_train,y\_train)

if method=="NAG":

return NAG(W,X\_train,y\_train)

if method=="RMSProp":

return RMSProp(W,X\_train,y\_train)

if method=="AdaDelta":

return AdaDelta(W,X\_train,y\_train)

if method=="Adam":

return Adam(W,X\_train,y\_train)

def getdata():

X\_train, y\_train = load\_svmlight\_file(r'C:\Users\jy\Desktop\Libsvmdata\a9a.txt')

datasize,features=X\_train.shape

X\_train=np.c\_[np.ones(len(X\_train.toarray())), X\_train.toarray()]

for i in range(0, len(y\_train)):

if y\_train[i] == -1:

y\_train[i] = 0

X\_test,y\_test=load\_svmlight\_file(r'C:\Users\jy\Desktop\Libsvmdata\a9a\_test.txt')

X\_test=np.c\_[X\_test.toarray(),np.zeros(len(X\_test.toarray()))]

X\_test=np.c\_[np.ones(len(X\_test)),X\_test]

for i in range(0, len(y\_test)):

if y\_test[i] == -1:

y\_test[i] = 0

y\_train = y\_train.reshape([len(y\_train), 1])

y\_test = y\_test.reshape([len(y\_test), 1])

X\_train,y\_train=shuffle(X\_train,y\_train)

X\_test,y\_test=shuffle(X\_test,y\_test)

return X\_train,y\_train,X\_test,y\_test,datasize,features

def get\_sub\_batch(batch\_count,X,y,data\_size):

if (1+batch\_count)\*batch\_size<=data\_size:

return X[batch\_count\*batch\_size:(batch\_count + 1) \* batch\_size],y[batch\_count\*batch\_size:(batch\_count + 1) \* batch\_size]

else:

return X[batch\_count\*batch\_size:data\_size],y[batch\_count\*batch\_size:data\_size]

def shuffle(X,y):

rng\_state = np.random.get\_state()

np.random.shuffle(X)

np.random.set\_state(rng\_state)

np.random.shuffle(y)

return X,y

def LinearClassificationModel():

X\_train, y\_train, X\_test, y\_test, data\_size, features\_num = getdata()

plt.xlabel('Iteration')

plt.ylabel('Loss')

for method in SGD\_methods:

W = np.random.rand(features\_num + 1, 1)

iter\_ = []

error = []

num = 0

for j in range(2):

for i in range(0, int(data\_size / batch\_size ) + 1):

iter\_.append(num)

X,y=get\_sub\_batch(i,X\_train,y\_train,data\_size)

W=opitimizer(W,X,y,method)

error.append(compute\_loss(W,X\_test,y\_test))

num+=1

print(method+"completed!")

plt.plot(iter\_, error, label=method)

plt.legend()

plt.show()

LinearClassificationModel()

（针对线性回归和线性分类分别填写8-12内容）

## 逻辑回归与随机梯度下降：

### 8. 模型参数的初始化方法:

采用随机方式初始化权重W

SGD，NAG，RMSProp，AdaDelta，Adam的各项初始化参数均参考论文里的推荐数值进行设置。

parm={"C":0.9,\

"SGD":{"learning rate":0.01},\

"NAG":{"learning rate":0.005,"Gamma":0.9},\

"RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},\

"AdaDelta":{"Gamma":0.95,"Epsilon":10e-6},\

"Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}}

### 9.选择的loss函数及其导数:

其导数为：

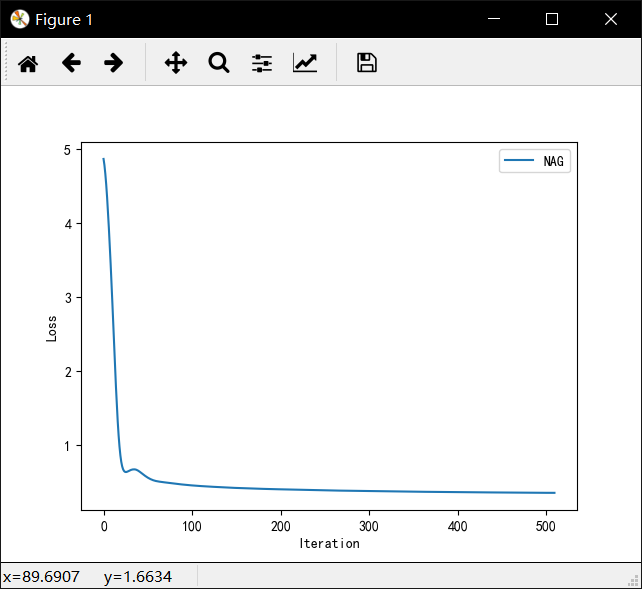
### 10.实验结果和曲线图:

#### NAG

##### 超参数选择："NAG":{"learning rate":0.005,"Gamma":0.9},

##### 预测结果（最佳结果）：

##### loss曲线图：

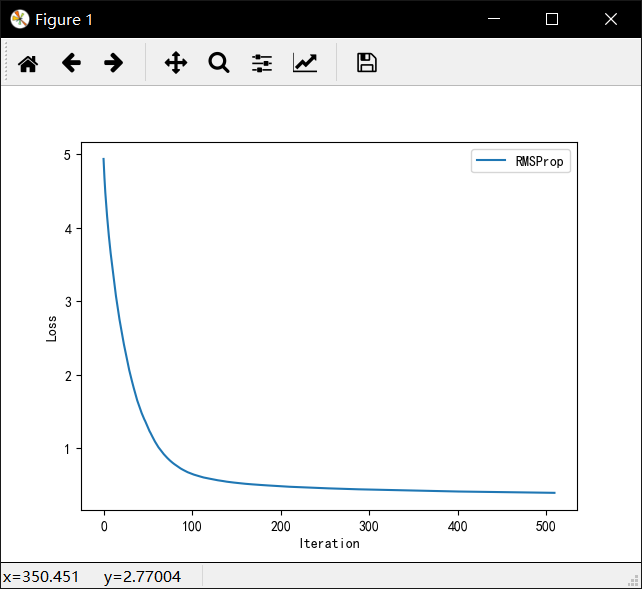


RMSProp：

##### 超参数选择：RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},

##### 预测结果（最佳结果）：

##### loss曲线图：

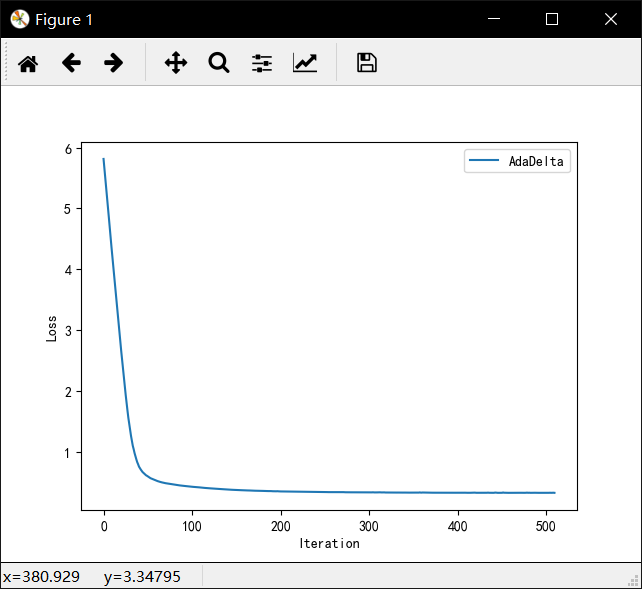


##### AdaDelta

##### 超参数选择："AdaDelta":{"Gamma":0.95,"Epsilon":10e-6}

##### 预测结果（最佳结果）：

##### loss曲线图：

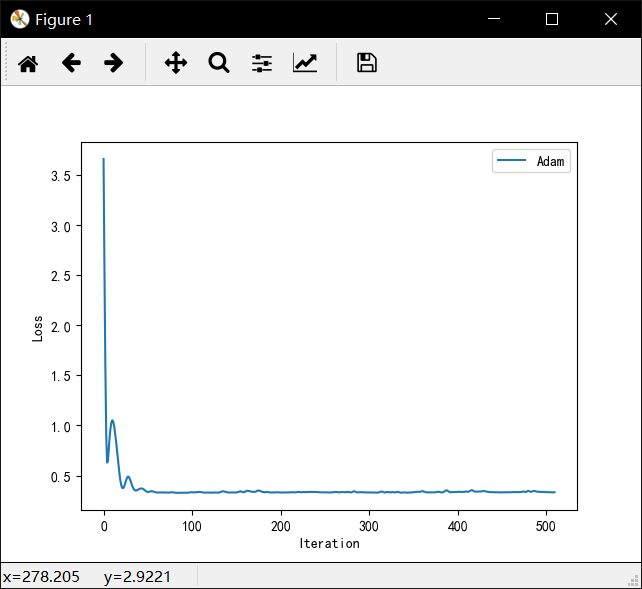


##### Adam

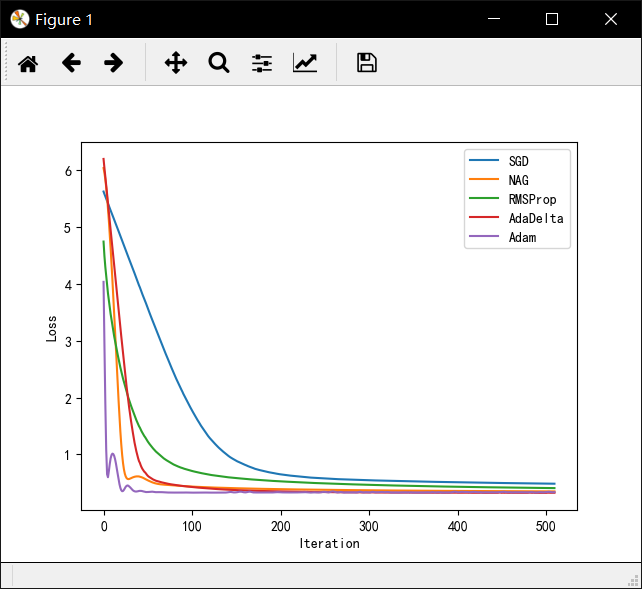
##### 超参数选择："Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}

##### 预测结果（最佳结果）：

##### loss曲线图：



### 12.实验结果分析:



从这五条曲线我们可以看出，单纯的SGD的loss在下降速度上比较缓慢，这说明了SGD在面对大量数据时梯度下降的准确率略低，同时learning rate 的选取比较困难，容易使得loss曲线振荡且易收敛到局部最优。NAG引入了动量影响的使得梯度的更新更加灵活，AdaDelta不再依赖于人为设定的全局学习率，在初期加速效果不错，可在后期在最小值附近振荡，RMSprop可看做是AdaDelta 的特例，但仍依赖于全局学习率，Adam本质上是带有动量项的RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率，可以看出它的loss曲线下降的速度是最快的，Adam的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。

## 线性分类与随机梯度下降：

### 8. 选择的评估方法（留出法，交叉验证，k折交叉验证等）:

采用留出法来切分数据集：调用train\_test\_split函数切分数据集, test\_size=0.33,选取0.33的数据作为验证集，随机选择。

### 9. 模型参数的初始化方法:

采用随机方式初始化权重W

SGD，NAG，RMSProp，AdaDelta，Adam的各项初始化参数均参考论文里的推荐数值进行设置。

parm={"C":0.9,\

"SGD":{"learning rate":0.01},\

"NAG":{"learning rate":0.005,"Gamma":0.9},\

"RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},\

"AdaDelta":{"Gamma":0.95,"Epsilon":10e-6},\

"Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}}

### 10.选择的loss函数及其导数:

Loss函数为margin loss和hinge loss的组合

其导数为：

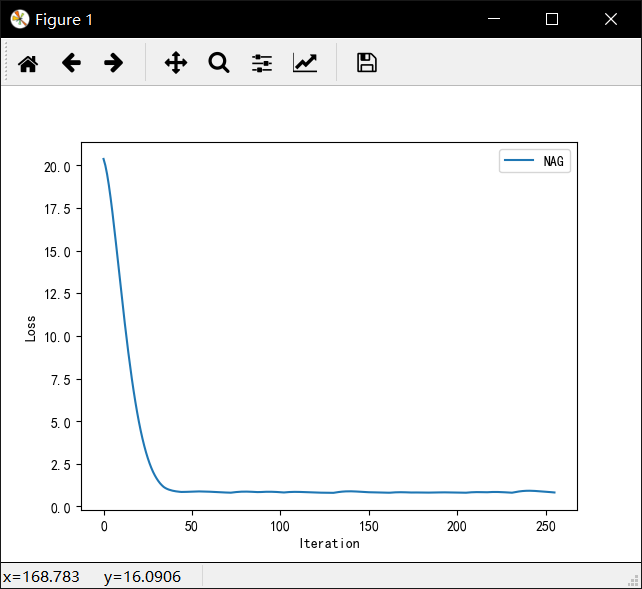
### 11.实验结果和曲线图:

#### NAG

##### 超参数选择："NAG":{"learning rate":0.005,"Gamma":0.9},

##### 预测结果（最佳结果）：

##### loss曲线图：

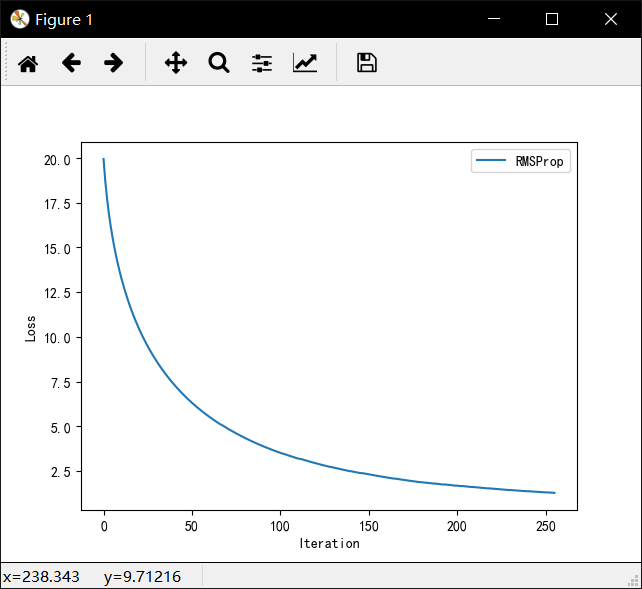


RMSProp：

##### 超参数选择：RMSProp":{"learning rate":0.01,"Gamma":0.9,"Epsilon":10e-8},

##### 预测结果（最佳结果）：

##### loss曲线图：

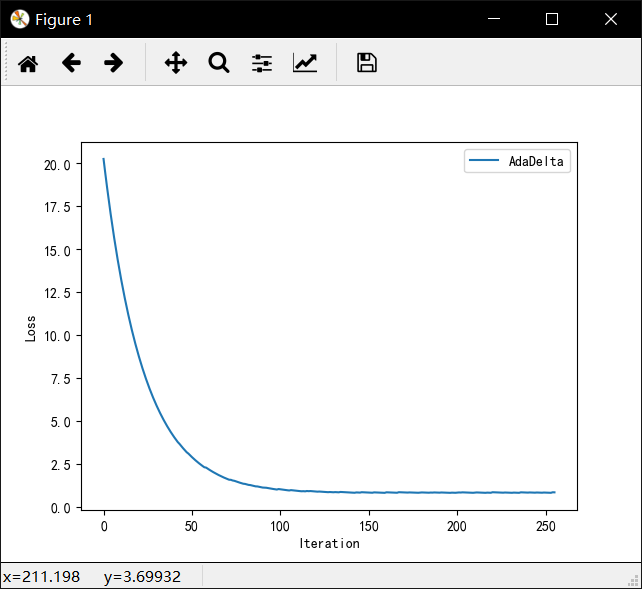


##### AdaDelta

##### 超参数选择："AdaDelta":{"Gamma":0.95,"Epsilon":10e-6}

##### 预测结果（最佳结果）：

##### loss曲线图：

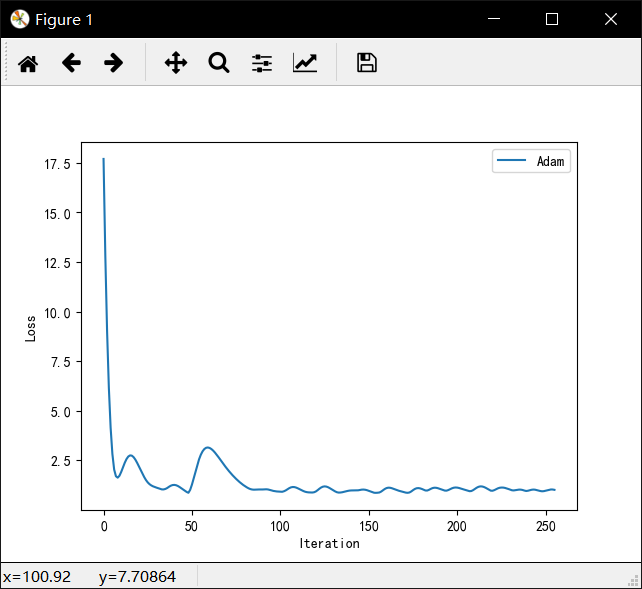


##### Adam

##### 超参数选择："Adam":{"Beta":0.9,"Gamma":0.999,"learning rate":0.1,"Epsilon":10e-8}

##### 预测结果（最佳结果）：

##### loss曲线图：



### 12.实验结果分析:

从这五条曲线我们可以看出，单纯的SGD的loss在下降速度上比较缓慢，这说明了SGD在面对大量数据时梯度下降的准确率略低，同时learning rate 的选取比较困难，容易使得loss曲线振荡且易收敛到局部最优。NAG引入了动量影响的使得梯度的更新更加灵活，AdaDelta不再依赖于人为设定的全局学习率，在初期加速效果不错，可在后期在最小值附近振荡，RMSprop可看做是AdaDelta 的特例，但仍依赖于全局学习率，Adam本质上是带有动量项的RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。

## 13.对比线性回归和线性分类的异同点：

线性回归和线性分类本质上都是模型的拟合，都是在最小化loss的过程中调整参数来拟合数据。他们的不同在于输出变量的类型，线性回归输出的是定量分析，也就是预测一个实际值，线性分类的输出是定性分析，也就是做分类判断。在最小化loss函数的过程中，他们的思路是相通相近的，都可以用递归下降的方式来优化权重

## 14.实验总结：

通过此次实验，进一步理解了线性回归，线性分类，随机梯度下降以及其他四种随机梯度下降的原理与实现方法。在大规模数据集上实践体会优化和调参的过程。同时也训练了我们对矩阵运算的使用，尤其实在大规模的数据上，要尽快能的利用numpy库里的运算函数而不是自己写for循环，这样才能充分使用计算机的性能，加快运算速度，提高训练的迭代次数。