

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**   **王圣杰**

**学 号 201530612873**

**邮 箱 2256989051@qq.com**

**指导教师**  **吴庆耀**

**提交日期** **2017年 12 月 14 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: 王圣杰

## 4. 实验目的:

## (1)对比理解梯度下降和随机梯度下降的区别与联系。

## (2)对比理解逻辑回归和线性分类的区别与联系。

## (3)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:

**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G。
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LNAG，LRMSProp，LAdaDelta和LAdam。
7. 重复步骤4-6若干次，**画出LNAG，LRMSProp，LAdaDelta和LAdam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G。
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LNAG，LRMSProp，LAdaDelta和LAdam。
7. 重复步骤4-6若干次，**画出LNAG，LRMSProp，LAdaDelta和LAdam随迭代次数的变化图**。

## 7. 代码内容:

## 逻辑回归和随机梯度下降

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.datasets import load\_svmlight\_file

#下面是对数据进行处理的代码

data1 = load\_svmlight\_file("a9a.txt")

X\_train, y\_train = data1[0], data1[1]

data2=load\_svmlight\_file("a9a\_t.txt")

X\_test,y\_test=data2[0],data2[1]

y\_train[y\_train==-1]=0

y\_test[y\_test==-1]=0

X\_train=X\_train.dot(np.eye(123))

X\_train=np.column\_stack((X\_train,np.ones((32561,1))))

X\_test=X\_test.dot(np.eye(122))

X\_test=np.column\_stack((X\_test,np.zeros((16281,1))))

X\_test=np.column\_stack((X\_test,np.ones((16281,1))))

y\_train=y\_train.reshape(32561,1)

y\_test=y\_test.reshape(16281,1)

m\_train=X\_train.shape[0]

m\_test=X\_test.shape[0]

#定义sigmoid函数

def sigmoid(z):

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

#定义预测准确率函数

def predict(w,X,y):

right=0

p=X.dot(w)

for i in range(X.shape[0]):

if p[i]>=0:

p[i]=1

else:

p[i]=0

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

if p[j]==y[j]:

right=right+1

P=right/X.shape[0]

return P

#定义学习率

alpha=0.1

#初始化所有参数，这里全部采用全零初始化

w\_SGD=np.zeros((124,1))

G\_SGD=np.zeros((124,1))

D\_SGD=np.zeros((124,1))

L\_SGD=np.zeros((1000,1))

P=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((300,124))

z\_y=np.zeros((300,1))

for j in range(300):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

for k in range(124):

G\_SGD[k]=((sigmoid(z\_x.dot(w\_SGD))-z\_y).T).dot(z\_x[:,k])

G\_SGD=G\_SGD/300

D\_SGD=-G\_SGD

w\_SGD=w\_SGD+alpha\*D\_SGD

for l in range(m\_test):

L\_SGD[i]=L\_SGD[i]-(y\_test[l]\*np.log(sigmoid((X\_test[l,:].reshape(1,124)).dot(w\_SGD)))+(1-y\_test[l])\*np.log(1-sigmoid((X\_test[l,:].reshape(1,124)).dot(w\_SGD))))

L\_SGD[i]=L\_SGD[i]/m\_test

P[i]=predict(w\_SGD,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_SGD.argmin()+1, L\_SGD.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_SGD,label='L\_SGD')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

#定义学习率和一些超参数

alpha=0.1

gamma=0.9

#初始化所有参数，这里全部采用全零初始化

w\_NAG=np.zeros((124,1))

G\_NAG=np.zeros((124,1))

v\_NAG=np.zeros((124,1))

L\_NAG=np.zeros((1000,1))

P=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((300,124))

z\_y=np.zeros((300,1))

for j in range(300):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

for k in range(124):

G\_NAG[k]=((sigmoid(z\_x.dot(w\_NAG-gamma\*v\_NAG))-z\_y).T).dot(z\_x[:,k])

G\_NAG=G\_NAG/300

v\_NAG=gamma\*v\_NAG+alpha\*G\_NAG

w\_NAG=w\_NAG-v\_NAG

for l in range(m\_test):

L\_NAG[i]=L\_NAG[i]-(y\_test[l]\*np.log(sigmoid((X\_test[l,:].reshape(1,124)).dot(w\_NAG)))+(1-y\_test[l])\*np.log(1-sigmoid((X\_test[l,:].reshape(1,124)).dot(w\_NAG))))

L\_NAG[i]=L\_NAG[i]/m\_test

P[i]=predict(w\_NAG,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_NAG.argmin()+1, L\_NAG.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_NAG,label='L\_NAG')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

#定义学习率和一些超参数

alpha=0.001

gamma=0.9

epsilon=0.00000001

#初始化所有参数，这里全部采用全零初始化

w\_RMS=np.zeros((124,1))

G\_RMS=np.zeros((124,1))

g\_RMS=np.zeros((124,1))

L\_RMS=np.zeros((1000,1))

P=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((300,124))

z\_y=np.zeros((300,1))

for j in range(300):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

for k in range(124):

g\_RMS[k]=((sigmoid(z\_x.dot(w\_RMS))-z\_y).T).dot(z\_x[:,k])

g\_RMS=g\_RMS/300

G\_RMS=gamma\*G\_RMS+(1-gamma)\*g\_RMS\*g\_RMS

w\_RMS=w\_RMS-alpha/np.sqrt(G\_RMS+epsilon)\*g\_RMS

for l in range(m\_test):

L\_RMS[i]=L\_RMS[i]-(y\_test[l]\*np.log(sigmoid((X\_test[l,:].reshape(1,124)).dot(w\_RMS)))+(1-y\_test[l])\*np.log(1-sigmoid((X\_test[l,:].reshape(1,124)).dot(w\_RMS))))

L\_RMS[i]=L\_RMS[i]/m\_test

P[i]=predict(w\_RMS,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_RMS.argmin()+1, L\_RMS.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_RMS,label='L\_RMS')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

#定义一些超参数

gamma=0.95

epsilon=0.000001

#初始化所有参数，这里全部采用全零初始化

w\_AdaD=np.zeros((124,1))

G\_AdaD=np.zeros((124,1))

g\_AdaD=np.zeros((124,1))

deltaT=np.zeros((124,1))

deltaW=np.zeros((124,1))

L\_AdaD=np.zeros((1000,1))

P=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((300,124))

z\_y=np.zeros((300,1))

for j in range(300):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

for k in range(124):

g\_AdaD[k]=((sigmoid(z\_x.dot(w\_AdaD))-z\_y).T).dot(z\_x[:,k])

g\_AdaD=g\_AdaD/300

G\_AdaD=gamma\*G\_AdaD+(1-gamma)\*g\_AdaD\*g\_AdaD

deltaW=-np.sqrt((deltaT+epsilon)/(G\_AdaD+epsilon))\*g\_AdaD

w\_AdaD=w\_AdaD+deltaW

deltaT=gamma\*deltaT+(1-gamma)\*deltaW\*deltaW

for l in range(m\_test):

L\_AdaD[i]=L\_AdaD[i]-(y\_test[l]\*np.log(sigmoid((X\_test[l,:].reshape(1,124)).dot(w\_AdaD)))+(1-y\_test[l])\*np.log(1-sigmoid((X\_test[l,:].reshape(1,124)).dot(w\_AdaD))))

L\_AdaD[i]=L\_AdaD[i]/m\_test

P[i]=predict(w\_AdaD,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_AdaD.argmin()+1, L\_AdaD.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_AdaD,label='L\_AdaD')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_SGD,label='L\_SGD')

plt.plot(x,L\_NAG,label='L\_NAG')

plt.plot(x,L\_RMS,label='L\_RMS')

plt.plot(x,L\_AdaD,label='L\_AdaD')

plt.plot(x,L\_Adam,label='L\_Adam')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

## 线性分类和随机梯度下降

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.datasets import load\_svmlight\_file

#下面是对数据进行处理的代码

data1 = load\_svmlight\_file("a9a.txt")

X\_train, y\_train = data1[0], data1[1]

data2=load\_svmlight\_file("a9a\_t.txt")

X\_test,y\_test=data2[0],data2[1]

X\_train=X\_train.dot(np.eye(123))

X\_train=np.column\_stack((X\_train,np.ones((32561,1))))

X\_test=X\_test.dot(np.eye(122))

X\_test=np.column\_stack((X\_test,np.zeros((16281,1))))

X\_test=np.column\_stack((X\_test,np.ones((16281,1))))

y\_train=y\_train.reshape(32561,1)

y\_test=y\_test.reshape(16281,1)

m\_train=X\_train.shape[0]

m\_test=X\_test.shape[0]

#定义损失函数

def loss(w,X,y,C):

M\_test=1-y\*(X.dot(w))

M\_test[M\_test<0]=0

loss=sum(w\*w)/2+C\*sum(M\_test)/X.shape[0]

return loss

#定义计算梯度的函数

def grad(w,X,y,C):

M\_train=1-y\*(X.dot(w))

minus\_y=-y

minus\_y[M\_train<0]=0

g=w+C\*((X.T).dot(minus\_y))/16

g[-1]=C\*sum(minus\_y)/16

return g

#定义计算预测准确度的函数

def predict(w,X,y):

right=0

p=X.dot(w)

for i in range(X.shape[0]):

if p[i]>=0:

p[i]=1

else:

p[i]=-1

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

if p[j]==y[j]:

right=right+1

P=right/X.shape[0]

return P

#定义学习率和一些超参数

C=100

alpha=0.0005

#初始化所有参数，这里全部采用全零初始化

w\_SGD=np.zeros((124,1))

g\_SGD=np.zeros((124,1))

L\_SGD=np.zeros((1000,1))

P=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((16,124))

z\_y=np.zeros((16,1))

for j in range(16):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

g\_SGD=grad(w\_SGD,z\_x,z\_y,C)

w\_SGD=w\_SGD-alpha\*g\_SGD

L\_SGD[i]=loss(w\_SGD,X\_test,y\_test,C)

P[i]=predict(w\_SGD,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_SGD.argmin()+1, L\_SGD.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_SGD,label='L\_SGD')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

#定义学习率和一些超参数

C=100

alpha=0.0001

gamma=0.9

#初始化所有参数，这里全部采用全零初始化

w\_NAG=np.zeros((124,1))

g\_NAG=np.zeros((124,1))

v\_NAG=np.zeros((124,1))

L\_NAG=np.zeros((1000,1))

P=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((16,124))

z\_y=np.zeros((16,1))

for j in range(16):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

g\_NAG=grad(w\_NAG-gamma\*v\_NAG,z\_x,z\_y,C)

v\_NAG=gamma\*v\_NAG+alpha\*g\_NAG

w\_NAG=w\_NAG-v\_NAG

L\_NAG[i]=loss(w\_NAG,X\_test,y\_test,C)

P[i]=predict(w\_NAG,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_NAG.argmin()+1, L\_NAG.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_NAG,label='L\_NAG')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

#定义学习率和一些超参数

C=100

alpha=0.001

gamma=0.9

epsilon=0.00000001

#初始化所有参数，这里全部采用全零初始化

w\_RMS=np.zeros((124,1))

G\_RMS=np.zeros((124,1))

g\_RMS=np.zeros((124,1))

L\_RMS=np.zeros((1000,1))

P=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((300,124))

z\_y=np.zeros((300,1))

for j in range(300):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

g\_RMS=grad(w\_RMS,z\_x,z\_y,C)

G\_RMS=gamma\*G\_RMS+(1-gamma)\*g\_RMS\*g\_RMS

w\_RMS=w\_RMS-alpha/np.sqrt(G\_RMS+epsilon)\*g\_RMS

L\_RMS[i]=loss(w\_RMS,X\_test,y\_test,C)

P[i]=predict(w\_RMS,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_RMS.argmin()+1, L\_RMS.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_RMS,label='L\_RMS')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

#定义学习率和一些超参数

C=100

gamma=0.95

epsilon=0.000001

#初始化所有参数，这里全部采用全零初始化

w\_AdaD=np.zeros((124,1))

G\_AdaD=np.zeros((124,1))

g\_AdaD=np.zeros((124,1))

deltaT=np.zeros((124,1))

deltaW=np.zeros((124,1))

P=np.zeros((1000,1))

L\_AdaD=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((300,124))

z\_y=np.zeros((300,1))

for j in range(300):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

g\_AdaD=grad(w\_AdaD,z\_x,z\_y,C)

G\_AdaD=gamma\*G\_AdaD+(1-gamma)\*g\_AdaD\*g\_AdaD

deltaW=-np.sqrt((deltaT+epsilon)/(G\_AdaD+epsilon))\*g\_AdaD

w\_AdaD=w\_AdaD+deltaW

deltaT=gamma\*deltaT+(1-gamma)\*deltaW\*deltaW

L\_AdaD[i]=loss(w\_AdaD,X\_test,y\_test,C)

P[i]=predict(w\_AdaD,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_AdaD.argmin()+1, L\_AdaD.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_AdaD,label='L\_AdaD')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

#定义学习率和一些超参数

C=100

eta=0.002

gamma=0.999

belta=0.9

epsilon=0.00000001

#初始化所有参数，这里全部采用全零初始化

P=np.zeros((1000,1))

w\_Adam=np.zeros((124,1))

g\_Adam=np.zeros((124,1))

mt=np.zeros((124,1))

G\_Adam=np.zeros((124,1))

L\_Adam=np.zeros((1000,1))

#迭代1000次

for i in range(1000):

z\_x=np.zeros((300,124))

z\_y=np.zeros((300,1))

for j in range(300):

r = np.random.randint(m\_train)

z\_x[j,:] = X\_train[r,:]

z\_y[j]=y\_train[r]

g\_Adam=grad(w\_Adam,z\_x,z\_y,C)

mt=belta\*mt+(1-belta)\*g\_Adam

G\_Adam=gamma\*G\_Adam+(1-gamma)\*g\_Adam\*g\_Adam

alpha=eta\*np.sqrt(1-gamma)/(1-belta)

w\_Adam=w\_Adam-alpha\*mt/np.sqrt(G\_Adam+epsilon)

L\_Adam[i]=loss(w\_Adam,X\_test,y\_test,C)

P[i]=predict(w\_Adam,X\_test,y\_test)

print("Best result: iteration %d, accuracy %f" %(P.argmax()+1, P.max()))

print("Lowest loss: iteration %d, loss %f" %(L\_Adam.argmin()+1, L\_Adam.min()))

x=np.arange(0,1000,1)

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_Adam,label='L\_Adam')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

plt.rcParams['figure.figsize'] = (10.0, 8.0)

plt.plot(x,L\_SGD,label='L\_SGD')

plt.plot(x,L\_NAG,label='L\_NAG')

plt.plot(x,L\_RMS,label='L\_RMS')

plt.plot(x,L\_AdaD,label='L\_AdaD')

plt.plot(x,L\_Adam,label='L\_Adam')

plt.legend(loc='upper right')

plt.xlabel('Num of iterations')

plt.ylabel('Loss')

plt.show()

## 逻辑回归和随机梯度下降8-11：

## 8.模型参数的初始化方法:

全零初始化。

## 9.选择的loss函数及其导数:

Loss函数：

其中，

梯度： 的梯度

（对于w14(b)而言x=1）

则整个向量的梯度向量

## 10.实验结果和曲线图:

## 超参数选择：

SGD的学习率α取0.1

NAG的学习率α取0.1,γ取0.9

RMSProp的学习率α取0.001,γ取0.9,ε取1e-8

AdaDelta的γ取0.95,ε取1e-6

Adam的学习率α取0.002,γ取0.999,ε取1e-8,β取0.9

## 预测结果（最佳结果）：

**SGD：**

Best result: iteration 997, accuracy 0.848351

Lowest loss: iteration 998, loss 0.330177

**NAG：**

Best result: iteration 699, accuracy 0.852098

Lowest loss: iteration 413, loss 0.323873

**RMSProp：**

Best result: iteration 988, accuracy 0.849948

Lowest loss: iteration 999, loss 0.328353

**AdaDelta：**

Best result: iteration 819, accuracy 0.852650

Lowest loss: iteration 911, loss 0.323587

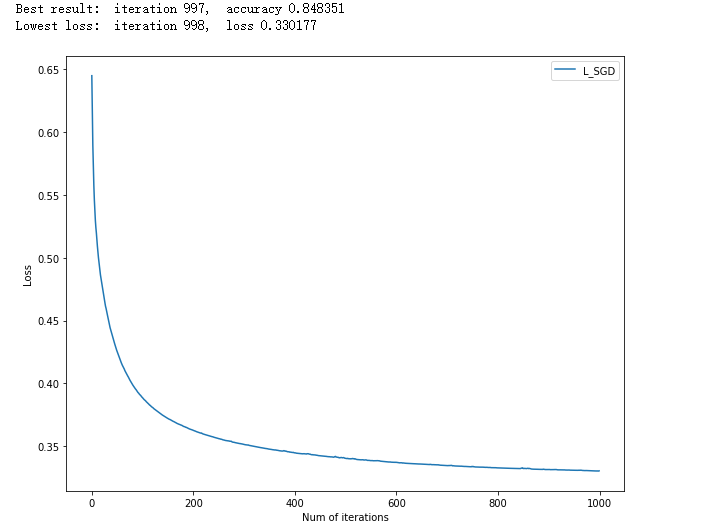
**Adam：**

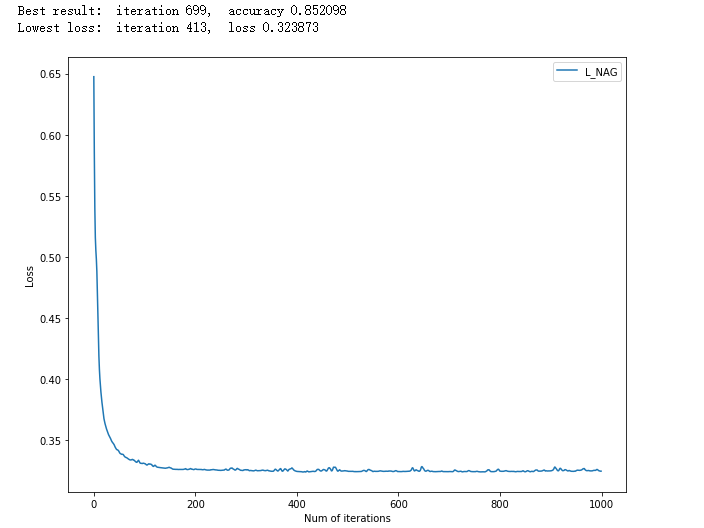
Best result: iteration 990, accuracy 0.848904

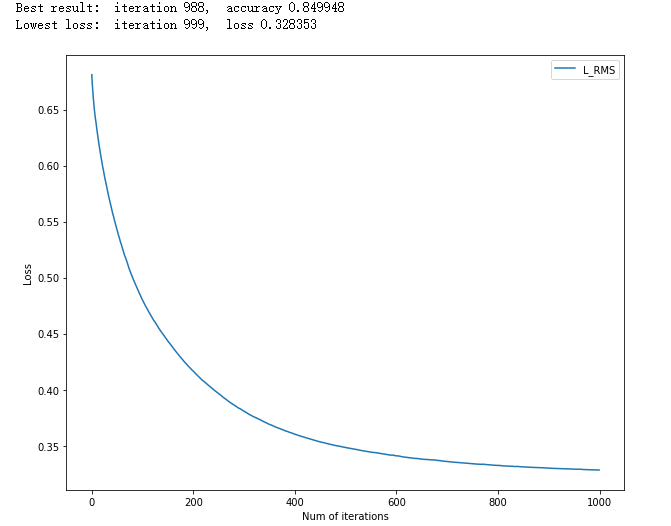
Lowest loss: iteration 1000, loss 0.331318

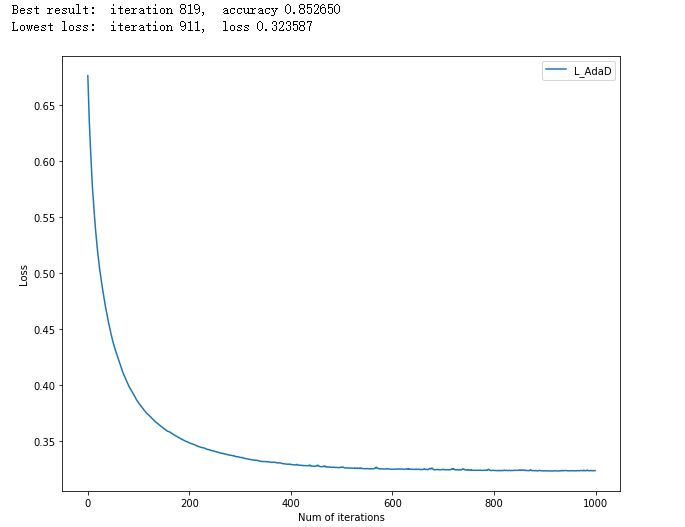
## loss曲线图：

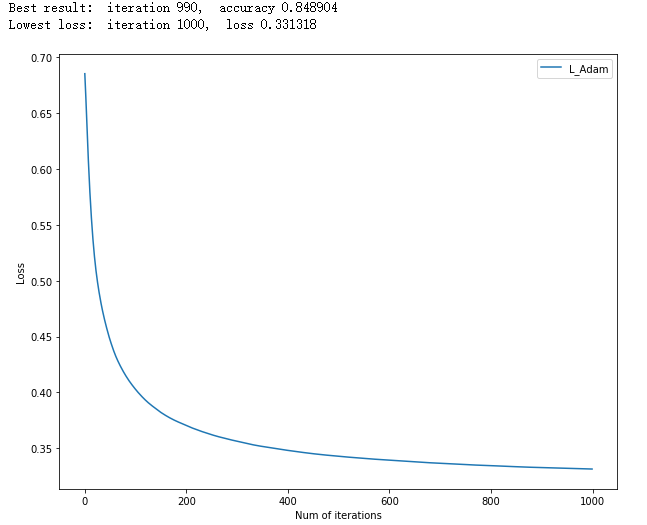
**SGD：**

**NAG：**

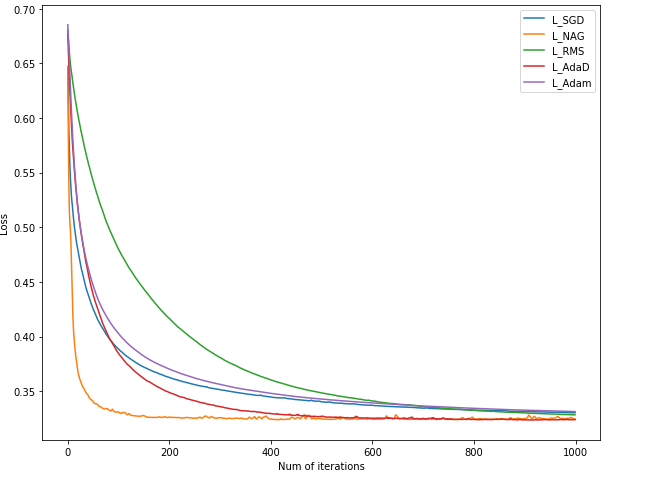
**RMSProp：**

**AdaDelta：**

**Adam：**



**五种汇总图：**



## 11.实验结果分析:

5种优化模型均在前200次迭代内就能使测试集上的Loss值大幅下降，然后在200-10000次迭代内Loss值缓慢地下降，最终达到0.33左右，此时在测试集上对应的正确率均可达84%以上。

NAG和AdaDelta模型最后的loss很相近，而且比另外三种模型低一些，并且两者的最高准确率达到了85.2%，高于另外三种模型

RMSProp，SGD和Adam模型的曲线比较相近，其中RMSProp的前600次迭代比另外两种稍微慢一点，600之后基本相同，并且最后的准确率和loss都很相近。

## 线性分类和随机梯度下降8-11：

## 8.模型参数的初始化方法:

全零初始化。

## 9.选择的loss函数及其导数:

Loss函数：

梯度：对wi的梯度

## 10.实验结果和曲线图:

## 超参数选择：

SGD的学习率α取0.0005，C=100

NAG的学习率α取0.0001,γ取0.9,C=100

RMSProp的学习率α取0.001,γ取0.9,ε取1e-8,C=100

AdaDelta的γ取0.95,ε取1e-6,C=100

Adam的学习率α取0.002,γ取0.999,ε取1e-8,β取0.9,C=100

预测结果（最佳结果）：

**SGD**

Best result: iteration 872, accuracy 0.845587

Lowest loss: iteration 744, loss 37.945012

**NAG**

Best result: iteration 746, accuracy 0.846140

Lowest loss: iteration 457, loss 37.853306

**RMSProp**

Best result: iteration 998, accuracy 0.848658

Lowest loss: iteration 690, loss 38.435086

**AdaDelta**

Best result: iteration 740, accuracy 0.850378

Lowest loss: iteration 280, loss 38.685186

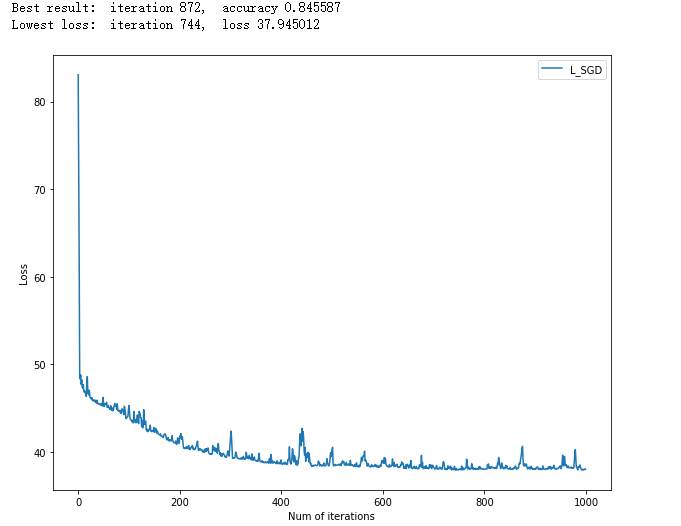
**Adam**

Best result: iteration 982, accuracy 0.847491

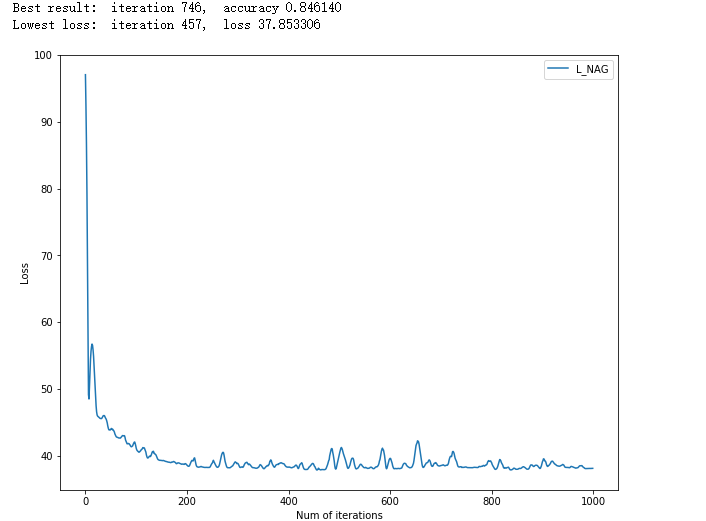
Lowest loss: iteration 732, loss 38.837161

## loss曲线图：

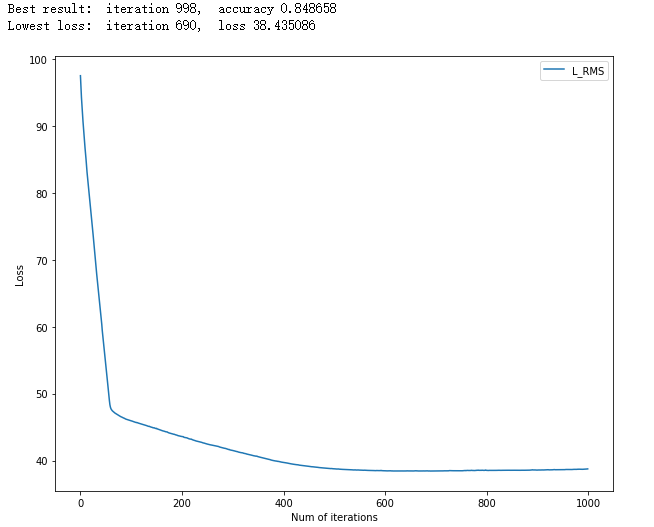
**SGD:**



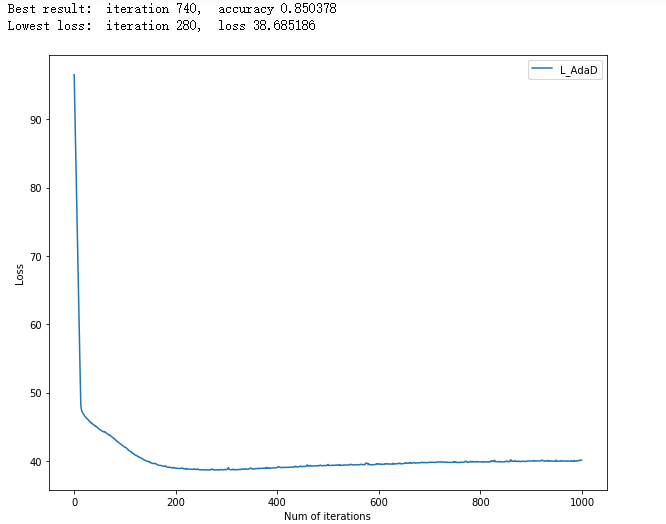
**NAG:**



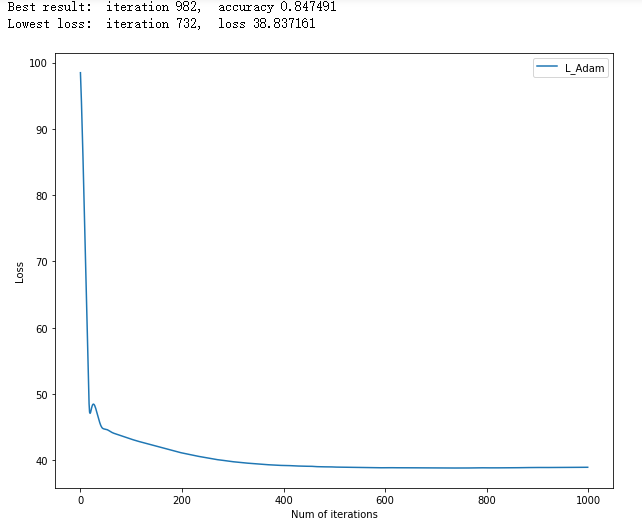
**RMSProp:**



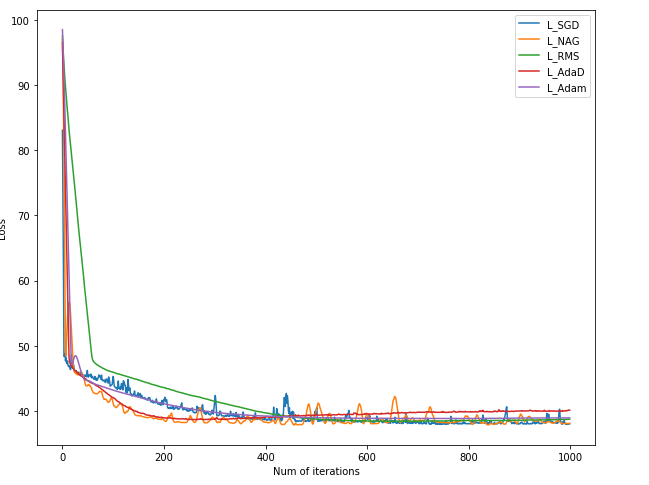
**AdaDelta:**



**Adam;**



**五种汇总图：**



## 11.实验结果分析:

5种优化模型均在前200次迭代内就能使测试集上的Loss值大幅下降，然后在200-10000次迭代内Loss值缓慢地下降，最终达到38左右，此时在测试集上对应的正确率均可达84%以上。

AdaDelta在测试集上最高可以达到85%以上的准确率，而且算法也比较稳定。

以上5种优化模型的曲线除了RMSProp在整个迭代中均有不同程度的上下波动，其中SGD和NAG的波动程度比较大。

## 对比逻辑回归和线性分类的异同点：

两种方法都是常见的分类算法,从目标函数来看,区别在于逻辑回归采用的是logistical loss,svm采用的是hinge loss.这两个损失函数的目的都是增加对分类影响较大的数据点的权重,减少与分类关系较小的数据点的权重.SVM的处理方法是只考虑support vectors,也就是和分类最相关的少数点,去学习分类器.而逻辑回归通过非线性映射,大大减小了离分类平面较远的点的权重,相对提升了与分类最相关的数据点的权重.两者的根本目的都是一样的.此外,根据需要,两个方法都可以增加不同的正则化项,如l1,l2等等.所以在很多实验中,两种算法的结果是很接近的

但是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.但是SVM的理论基础更加牢固,有一套结构化风险最小化的理论基础,虽然一般使用的人不太会去关注.还有很重要的一点,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

svm 更多的属于非参数模型,而logistic regression是参数模型,本质不同.其区别就可以参考参数模型和非参模型的区别就好了

logic 能做的 svm能做,但可能在准确率上有问题,svm能做的logic有的做不了

## 实验总结：

通过本次实验，我深入的学习了逻辑回归以及采用SVM实现的线性分类模型的实现和参数调试，学习了如何利用Jupyter Notebook来编写模型代码，了解了sklearn，numpy，jupyter，matplotlib包的一些基本操作。通过自己手写代码，让我进一步理解了逻辑回归、线性分类和梯度下降的原理，并且在小规模的数据集上实践，通过绘制图像来可视化优化和调参的过程，让这种过程给自己更深的体会和收获，在试验过程总，我有几次打错了，算法的代码，都是通过绘制出的Loss-Iterations图观察出来的，还有在调参的过程中通过更改参数来观察图形的变化让我直观地体会到了调参的过程。通过这次试验，让我对于机器学习有了真实的接触，为我之后的实验打下了必要的的基础，并在过程中收获了对于不同问题的处理方法，积累了宝贵的经验。