

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**  **黄景超**

**学 号 201530611722**

**邮 箱 [1620265690@qq.com](mailto:1620265690@qq.com)**

**指导教师**  **吴庆耀**

**提交日期** **2017年 12 月 9 日**

## 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:黄景超

## 4. 实验目的:

A.对比理解梯度下降和随机梯度下降的区别与联系。

B.对比理解逻辑回归和线性分类的区别与联系。

C.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。

## 实验步骤:



## （逻辑回归）代码内容:

import numpy as np

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

def get\_data(path,feature):

data = load\_svmlight\_file(path,feature)

return data

def predict(W , X, y ,threshold):

z= np.dot(W.T ,X)

temp= 1./(1+np.exp( -z))

y\_pred=np.zeros(temp.shape)

y\_pred[temp> threshold]=1;

y\_pred[temp<=threshold]=0;

cmp=y\_pred==y

accuracy=len(cmp[cmp==True])/y.shape[1]

return y\_pred,accuracy

# 读取数据

data = get\_data(path="a9a",feature=123)

# 数据预处理

x\_train=data[0].toarray()

x\_train=np.column\_stack((x\_train,np.ones([x\_train.shape[0],1])))

x\_train=x\_train.T

y\_train=data[1]

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

y\_train=y\_train.astype(np.int)

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

D\_in, N =x\_train.shape

D\_out = y\_train.shape[0]

# 读取数据

data = get\_data(path="a9a.t",feature=123)

# 数据预处理

x\_test=data[0].toarray()

x\_test=np.column\_stack((x\_test,np.ones([x\_test.shape[0],1])))

x\_test=x\_test.T

y\_test=data[1]

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

y\_test=y\_test.astype(np.int)

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

# 参数初始化

maxIterations=10000 # 迭代

eta = 0.05 # 学习率

threshold=0.5 # 大于阈值的标记为正类，反之为负类

def logisticRegression(W ,xtrain, ytrain, xtest ,ytest):

N = xtrain.shape[1]

gradNum=10

ind=random.sample(range(0,N),gradNum)

xtrain\_batch=xtrain[:,ind]

ytrain\_batch=ytrain[:,ind]

train\_loss = 0

test\_loss =0

dW = np.zeros(W.shape)

z\_train= np.dot(W.T ,xtrain)

a\_train= 1./(1+np.exp( -z\_train))

a\_train\_batch=a\_train[:,ind]

z\_test= np.dot(W.T ,xtest)

a\_test= 1./(1+np.exp( -z\_test))

#逻辑回归Loss

train\_loss= -1/N \*(np.dot(np.log(a\_train),ytrain.T)+np.dot(np.log(1.0-a\_train),(1-ytrain).T))

test\_loss=-1/N \*(np.dot(np.log(a\_test),ytest.T)+np.dot(np.log(1.0-a\_test),(1-ytest).T))

#loss

dz = a\_train\_batch-ytrain\_batch

dW = 1/N \* np.dot(xtrain\_batch, dz.T)

return train\_loss, test\_loss, dW

# NAG

# 参数初始化

W = np.zeros((D\_in, D\_out)) # weights

pre\_d = np.zeros\_like(W)

pre\_grad = np.zeros\_like(W)

gamma =0.9 #动量因子

L\_NAG =[]; # 验证loss

for t in range(maxIterations):

# 计算loss

train\_loss,test\_loss ,grad =logisticRegression(W, x\_train, y\_train, x\_test, y\_test)

# 保存

L\_NAG.append ( test\_loss)

# 更新weight

d = gamma \* pre\_d + grad + gamma \* (grad - pre\_grad)

dW = -eta \* d

W += dW

pre\_d = d

pre\_grad = grad

L\_NAG=np.array(L\_NAG)

L\_NAG=L\_NAG[:,:,0]

y\_pred\_NAG\_train,training\_accuracy\_NAG =predict(W , x\_train, y\_train ,threshold)

y\_pred\_NAG\_test,test\_accuracy\_NAG=predict(W , x\_test, y\_test ,threshold)

# NAG

# 参数初始化

W = np.zeros((D\_in, D\_out)) # weights

pre\_d = np.zeros\_like(W)

pre\_grad = np.zeros\_like(W)

gamma =0.9 #动量因子

L\_NAG =[]; # 验证loss

for t in range(maxIterations):

# 计算loss

train\_loss,test\_loss ,grad =logisticRegression(W, x\_train, y\_train, x\_test, y\_test)

# 保存

L\_NAG.append ( test\_loss)

# 更新weight

d = gamma \* pre\_d + grad + gamma \* (grad - pre\_grad)

dW = -eta \* d

W += dW

pre\_d = d

pre\_grad = grad

L\_NAG=np.array(L\_NAG)

L\_NAG=L\_NAG[:,:,0]

y\_pred\_NAG\_train,training\_accuracy\_NAG =predict(W , x\_train, y\_train ,threshold)

y\_pred\_NAG\_test,test\_accuracy\_NAG=predict(W , x\_test, y\_test ,threshold)

# AdaDelta

# 参数初始化

W = np.zeros((D\_in, D\_out)) # weights

E\_g2 = np.zeros\_like(W)

E\_dW2 = np.zeros\_like(W)

gamma =0.9 # 衰退因子

epsilon = 1e-3

L\_AdaDelta =[] # 验证loss

for t in range(maxIterations):

train\_loss,test\_loss ,grad =logisticRegression(W, x\_train, y\_train, x\_test, y\_test)

L\_AdaDelta.append ( test\_loss)

E\_g2 = gamma \* E\_g2 + (1-gamma) \* np.power(grad,2)

dW = - np.sqrt(E\_dW2+epsilon) / np.sqrt(E\_g2+epsilon) \* grad

W += dW

E\_dW2 = gamma \* E\_dW2 + (1-gamma) \* np.power(dW , 2)

L\_AdaDelta=np.array(L\_AdaDelta)

L\_AdaDelta=L\_AdaDelta[:,:,0]

y\_pred\_AdaDelta\_train,training\_accuracy\_AdaDelta =predict(W , x\_train, y\_train ,threshold)

y\_pred\_AdaDelta\_test,test\_accuracy\_AdaDelta =predict(W , x\_test, y\_test ,threshold)

# Adam

# 参数初始化

W = np.zeros((D\_in, D\_out)) # weights

n = np.zeros\_like(W)

m = np.zeros\_like(W)

mu = 0.9 # m的衰退因子

v = 0.9 # n的衰退因子

epsilon = 1e-3

L\_adam=[] # 验证loss

for t in range(maxIterations):

train\_loss,test\_loss ,grad =logisticRegression(W, x\_train, y\_train, x\_test, y\_test)

L\_adam.append ( test\_loss)

# 梯度估计

m = mu \* m + (1-mu) \* grad

n = v \* n + (1-v) \* np.power(grad,2)

m\_hat = m / (1-np.power(mu,t)+epsilon)

n\_hat = n / (1-np.power(v,t)+epsilon)

W -= m\_hat \* eta /(np.sqrt(n\_hat) + epsilon)

L\_adam=np.array(L\_adam)

L\_adam=L\_adam[:,:,0]

y\_pred\_Adam\_train,training\_accuracy\_Adam =predict(W , x\_train, y\_train ,threshold)

y\_pred\_Adam\_test,test\_accuracy\_Adam =predict(W , x\_test, y\_test ,threshold)

# 制图

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

plt.plot(L\_RMSProp,'g',label='L\_RMSProp')

plt.plot(L\_AdaDelta,'r',label='L\_AdaDelta')

plt.plot(L\_adam,'y',label='L\_adam')

plt.title('Loss Curve')

plt.xlabel('Iterations')

plt.ylabel('Loss')

plt.legend()

plt.show()

# 评估和预测结果

print('training accuracy\_NAG=',training\_accuracy\_NAG,

'\ntraining accuracy\_RMSProp=',training\_accuracy\_RMSProp,

'\ntraining accuracy\_AdaDelta=',training\_accuracy\_AdaDelta,

'\ntraining accuracy\_Adam=',training\_accuracy\_Adam)

print('\ntest accuracy\_NAG=',test\_accuracy\_NAG,

'\ntest accuracy\_RMSProp=',test\_accuracy\_RMSProp,

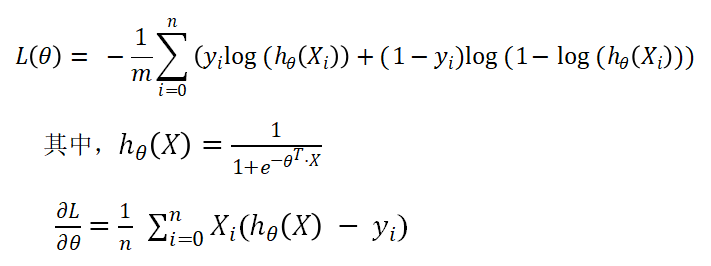
'\ntest accuracy\_AdaDelta=',test\_accuracy\_AdaDelta,

'\ntest accuracy\_Adam=',test\_accuracy\_Adam)

## （逻辑回归）模型参数的初始化方法:

模型参数的初始化方法采用的是全零初始化。

## 9.（逻辑回归）选择的loss函数及其导数:



## 10.（逻辑回归）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：

NAG：

threshold= 0.5

eta=0.05

maxIterations=10000

gamma=0.9

RMSProp：

threshold=0.5

eta=0.05

maxIterations=10000

gamma=0.9

epsilon=0.001

AdaDelta：

threshold=0.5

maxIterations=10000

gamma=0.9

epsilon=0.001

Adam：

threshold=0.5

maxIterations=10000

mu=0.9

v=0.9

epsilon=0.001

## 预测结果（最佳结果）：

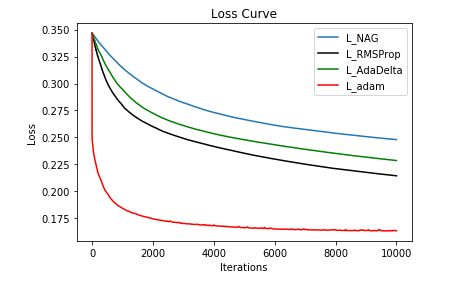
NAG：0.764

RMSProp：0.784

AdaDelta：0.765

Adam：0.850

## loss曲线图：



## （逻辑回归）实验结果分析:

从上述预测结果可以看出，所有的预测精度都很高，这说明模型的预测效果是比较好的。

从损失曲线来看，随着迭代次数增加，损失收敛到一个很小的数且接近于零。也就是说，我们所训练的模型是比较好的。

## （线性分类）代码内容:

import numpy as np

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

def get\_data(path,feature):

data = load\_svmlight\_file(path,feature)

return data

# 读取数据

data = get\_data(path="a9a",feature=123)

data = get\_data(path="a9a.t",feature=123)

# 数据预处理

x\_train=data[0].toarray()

x\_train=np.column\_stack((x\_train,np.ones([x\_train.shape[0],1])))

y\_train=data[1]

C=len(list(set(y\_train)))

y\_train=y\_train.astype(np.int)

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

N,D =x\_train.shape

x\_test=data[0].toarray()

x\_test=np.column\_stack((x\_test,np.ones([x\_test.shape[0],1])))

y\_test=data[1]

y\_test=y\_test.astype(np.int)

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

# 参数初始化

maxIterations=200

eta = 1e-3 # 学习率

def predict(W , X, y ):

# 所有样例的score

score = np.dot(X,W)

# 找到最大score

y\_pred= np.argmax(score, axis = 1)

# 计算准确度

cmp=(y\_pred == y)

accuracy=len(cmp[cmp==True])/len(cmp)

return y\_pred,accuracy

def svm(W, xtrain, ytrain, xtest, ytest, reg):

dW = np.zeros(W.shape)

num\_classes = W.shape[1]

#train

train\_loss = 0

scores\_train = xtrain.dot(W) # num\_train by C

num\_train = xtrain.shape[0]

scores\_train\_correct = scores\_train[np.arange(num\_train), ytrain] # 1 by num\_train

scores\_train\_correct = np.reshape(scores\_train\_correct, (num\_train, 1)) # num\_train by 1

margins\_train = scores\_train - scores\_train\_correct + 1.0 # num\_train by C

margins\_train[np.arange(num\_train), ytrain] = 0.0

margins\_train[margins\_train <= 0] = 0.0

train\_loss += np.sum(margins\_train) / num\_train

train\_loss += 0.5 \* reg \* np.sum(W \* W)

margins\_train[margins\_train > 0] = 1.0

row\_sum = np.sum(margins\_train, axis=1) # 1 by num\_train

margins\_train[np.arange(num\_train), ytrain] = -row\_sum

gradNum =20 # 样本数量

ind=random.sample(range(0,num\_train),gradNum)

xtrain\_batch=xtrain[ind,:]

margins\_train\_batch=margins\_train[ind,:]

dW += np.dot(xtrain\_batch.T, margins\_train\_batch)/gradNum + reg \* W # D by C

#test

test\_loss = 0

scores\_test = xtest.dot(W) # num\_test by C

num\_test = xtest.shape[0]

scores\_test\_correct = scores\_test[np.arange(num\_test), ytest] # 1 by N

scores\_test\_correct = np.reshape(scores\_test\_correct, (num\_test, 1)) # N by 1

margins\_test = scores\_test - scores\_test\_correct + 1.0 # N by C

margins\_test[np.arange(num\_test), ytest] = 0.0

margins\_test[margins\_test <= 0] = 0.0

test\_loss += np.sum(margins\_test) / num\_test

test\_loss += 0.5 \* reg \* np.sum(W \* W)

return train\_loss, test\_loss, dW

# NAG

# 参数初始化

W = np.zeros((D, C)) # weights

pre\_d = np.zeros\_like(W)

pre\_grad = np.zeros\_like(W)

gamma =0.9 #动量因子

L\_NAG =[]; # 验证loss

for t in range(maxIterations):

# 计算loss

train\_loss,test\_loss ,grad =svm(W, x\_train, y\_train, x\_test, y\_test, reg=0.1)

# 保存

L\_NAG.append ( test\_loss)

# 更新weights

d = gamma \* pre\_d + grad + gamma \* (grad - pre\_grad)

dW = -eta \* d

W += dW

pre\_d = d

pre\_grad = grad

L\_NAG=np.array(L\_NAG)

# 预测结果

y\_pred\_NAG\_train,training\_accuracy\_NAG = predict(W , x\_train, y\_train )

y\_pred\_NAG\_test,test\_accuracy\_NAG = predict(W , x\_test, y\_test )

# RMSProp

# 参数初始化

W = np.zeros((D, C))

n = np.zeros\_like(W)

gamma =0.9 # 衰退因子

epsilon = 0.001

L\_RMSProp =[] # 验证loss

for t in range(maxIterations):

train\_loss,test\_loss ,grad =svm(W, x\_train, y\_train, x\_test, y\_test, reg=0.1)

L\_RMSProp.append ( test\_loss)

n = gamma \* n + (1-gamma) \* np.power(grad,2)

dW = -eta /np.sqrt(n + epsilon ) \* grad

W += dW

L\_RMSProp=np.array(L\_RMSProp)

y\_pred\_RMSProp\_train,training\_accuracy\_RMSProp = predict(W , x\_train, y\_train )

y\_pred\_RMSProp\_test,test\_accuracy\_RMSProp = predict(W , x\_test, y\_test )

# AdaDelta

# 参数初始化

W = np.zeros((D, C))

E\_g2 = np.zeros\_like(W)

E\_dW2 = np.zeros\_like(W)

gamma =0.8

epsilon = 1e-6

L\_AdaDelta =[]

for t in range(maxIterations):

train\_loss,test\_loss ,grad =svm(W, x\_train, y\_train, x\_test, y\_test, reg=0.1)

L\_AdaDelta.append ( test\_loss)

E\_g2 = gamma \* E\_g2 + (1-gamma) \* np.power(grad,2)

dW = - np.sqrt(E\_dW2+epsilon) / np.sqrt(E\_g2+epsilon) \* grad

W += dW

E\_dW2 = gamma \* E\_dW2 + (1-gamma) \* np.power(dW , 2)

L\_AdaDelta=np.array(L\_AdaDelta)

y\_pred\_AdaDelta\_train,training\_accuracy\_AdaDelta = predict(W , x\_train, y\_train )

y\_pred\_AdaDelta\_test,test\_accuracy\_AdaDelta= predict(W , x\_test, y\_test )

# Adam

# 参数初始化

W = np.zeros((D, C))

n = np.zeros\_like(W)

m = np.zeros\_like(W)

mu = 0.9

v = 0.9

epsilon = 1e-3

L\_adam=[]

for t in range(maxIterations):

train\_loss,test\_loss ,grad =svm(W, x\_train, y\_train, x\_test, y\_test, reg=0.1)

L\_adam.append ( test\_loss)

m = mu \* m + (1-mu) \* grad

n = v \* n + (1-v) \* np.power(grad,2)

m\_hat = m / (1-np.power(mu,t)+epsilon)

n\_hat = n / (1-np.power(v,t)+epsilon)

W -= m\_hat \* eta /(np.sqrt(n\_hat) + epsilon)

L\_adam=np.array(L\_adam)

y\_pred\_Adam\_train,training\_accuracy\_Adam = predict(W , x\_train, y\_train )

y\_pred\_Adam\_test,test\_accuracy\_Adam = predict(W , x\_test, y\_test )

# 制图

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

plt.plot(L\_RMSProp,'black',label='L\_RMSProp')

plt.plot(L\_AdaDelta,'green',label='L\_AdaDelta')

plt.plot(L\_adam,'red',label='L\_adam')

plt.title('Error Curve')

plt.xlabel('iterations')

plt.ylabel('error')

plt.legend()

plt.show()

# 评估和预测结果

print('training accuracy\_NAG=',training\_accuracy\_NAG,

'\ntraining accuracy\_RMSProp=',training\_accuracy\_RMSProp,

'\ntraining accuracy\_AdaDelta=',training\_accuracy\_AdaDelta,

'\ntraining accuracy\_Adam=',training\_accuracy\_Adam)

print('\ntest accuracy\_NAG=',test\_accuracy\_NAG,

'\ntest accuracy\_RMSProp=',test\_accuracy\_RMSProp,

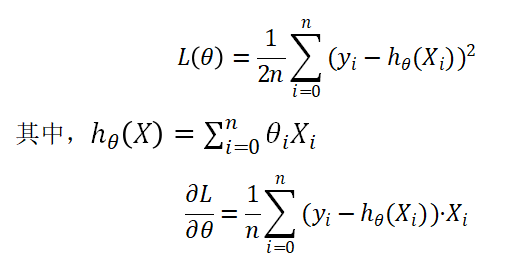
'\ntest accuracy\_AdaDelta=',test\_accuracy\_AdaDelta,

'\ntest accuracy\_Adam=',test\_accuracy\_Adam)

## （线性分类）模型参数的初始化方法:

模型参数的初始化方法采用的是全零初始化。

## 14.（线性分类）选择的loss函数及其导数:



## 15.（线性分类）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：

NAG：

eta=0.001

maxIterations=200

gamma=0.9

RMSProp：

eta=0.001

maxIterations=200

gamma=0.9

epsilon=0.001

AdaDelta：

maxIterations=200

gamma=0.8

epsilon=1e-6

Adam：

maxIterations=200

mu=0.9

v=0.9

epsilon=0.001

## 预测结果（最佳结果）：

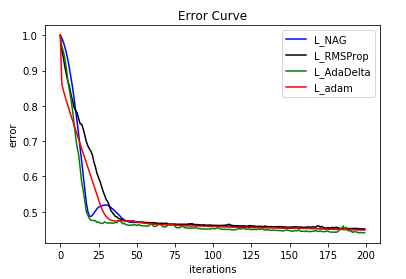
NAG：0.765

RMSProp：0.764

AdaDelta：0.798

Adam：0.768

## loss曲线图：



## （线性分类）实验结果分析:

从上述预测结果可以看出，所有的预测精度都很高，这说明模型的预测效果是比较好的。

从损失曲线来看，随着迭代次数增加，损失收敛到一个很小的数且接近于零。也就是说，我们所训练的模型是比较好的。

## 17.对比逻辑回归和线性分类的异同点：

同：

都属于分类问题，都用于预测。

异：

找最优超平面的方法不同，，形象点说，logistic模型找的那个超平面，是尽量让所有点都远离它，而SVM线性分类寻找的那个超平面，是只让最靠近中间分割线的那些点尽量远离，即只用到那些“支持向量”的样本。

逻辑回归只可以处理线性可分情况；SVM则二者皆可。

## 18.实验总结：

本次实验当中，我学习到了很多SGD在实践运用的经验，将课程中学习到的知识运用在实际问题上。但是由于自身对于知识把握懂得程度不高，在实现的过程中遇到了诸如无法正确实现SGD优化算法，调参不够灵活的问题，在总结反思之后解决了问题并顺利完成了实验。

在对模型的训练过程中，我体会到了灵活调参的重要性。在一开始，因为超参数C, learning rate等设置得不合理，导致loss图像与预期相差甚远，模型参数无法收敛或者收敛过慢，跑数据集时间过长等问题，导致无法拟合数据；或者是因为迭代次数过少，参数还未收敛便停止了训练。在进行数次的不同的调参后，模型往预期方向改变，我也从中学得一些经验。