

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**   **黄班班**

**学 号 201530611647**

**邮 箱 863237407@qq.com**

**指导教师**  **吴庆耀**

**提交日期** **2017年 12 月 14 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: 黄班班

## 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。

## 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。

## 实验步骤:



## 代码内容:

1.逻辑回归与随机梯度下降

import numpy

import random

import jupyter

import math

from sklearn.datasets import load\_svmlight\_file

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

from matplotlib import pyplot

def f\_loss(x, y, w, C, random\_i):

loss = 0

n = len(random\_i)

for m in range(n):

loss += math.log(1+math.exp(-y[m]\*(x[m,:].dot(w.T))))

loss = loss/n + C/2 \* w.dot(w.T)

return loss[0,0]

def f\_gradient(x, y, w, C):

gradient = (-y / (1+math.exp(y\*x.dot(w.T))))\*x + C\*w

return gradient

def NAG\_train(x, y, x\_test, y\_test, w, C, lr, gamma, threshold, iteration):

vt = numpy.zeros(w.shape)

lossvalue = []

testloss = []

random\_index = []

random\_test\_index = []

for i in range(iteration):

random\_num = random.randint(0, x.shape[0]-1)

random\_test\_num = random.randint(0, x\_test.shape[0]-1)

random\_index.append(random\_num)

random\_test\_index.append(random\_test\_num)

for i in range(iteration):

gradient = f\_gradient(x[random\_index[i],:], y[random\_index[i]], w-gamma\*vt, C)

vt = gamma\*vt - lr\*gradient

w += vt

loss = f\_loss(x, y, w, C, random\_index)

lossvalue.append(loss)

testloss.append(f\_loss(x\_test, y\_test, w, C, random\_test\_index))

if loss < threshold :

break

return w, lossvalue, testloss

def RMSProp\_train(x, y, x\_test, y\_test, w, C, lr, gamma, threshold, iteration):

Gt = 0

lossvalue = []

testloss = []

random\_index = []

random\_test\_index = []

for i in range(iteration):

random\_num = random.randint(0, x.shape[0]-1)

random\_test\_num = random.randint(0, x\_test.shape[0]-1)

random\_index.append(random\_num)

random\_test\_index.append(random\_test\_num)

for i in range(iteration):

gradient = f\_gradient(x[random\_index[i],:], y[random\_index[i]], w, C)

Gt = gamma\*Gt + (1-gamma)\*gradient.dot(gradient.T)

w -= lr \* gradient / math.sqrt(Gt+1e-8)

loss = f\_loss(x, y, w, C, random\_index)

lossvalue.append(loss)

testloss.append(f\_loss(x\_test, y\_test, w, C, random\_test\_index))

if loss < threshold :

break

return w, lossvalue, testloss

def AdaDelta\_train(x, y, x\_test, y\_test, w, C, lr, gamma, threshold, iteration):

Gt = 0

variable\_t = 0

lossvalue = []

testloss = []

random\_index = []

random\_test\_index = []

for i in range(iteration):

random\_num = random.randint(0, x.shape[0]-1)

random\_test\_num = random.randint(0, x\_test.shape[0]-1)

random\_index.append(random\_num)

random\_test\_index.append(random\_test\_num)

for i in range(iteration):

gradient = f\_gradient(x[random\_index[i],:], y[random\_index[i]], w, C)

Gt = gamma\*Gt + (1-gamma)\*gradient.dot(gradient.T)

variable\_w = - math.sqrt(variable\_t + 1e-8) \* gradient / math.sqrt(Gt + 1e-8)

w += variable\_w

variable\_t = gamma\*variable\_t + (1-gamma)\*variable\_w.dot(variable\_w.T)

loss = f\_loss(x, y, w, C, random\_index)

lossvalue.append(loss)

testloss.append(f\_loss(x\_test, y\_test, w, C, random\_test\_index))

if loss < threshold :

break

return w, lossvalue, testloss

def Adam\_train(x, y, x\_test, y\_test, w, C, lr, gamma, threshold, iteration):

Gt = 0

moment = numpy.zeros((1, x.shape[1]))

B = 0.9

lossvalue = []

testloss = []

random\_index = []

random\_test\_index = []

for i in range(iteration):

random\_num = random.randint(0, x.shape[0]-1)

random\_test\_num = random.randint(0, x\_test.shape[0]-1)

random\_index.append(random\_num)

random\_test\_index.append(random\_test\_num)

for i in range(iteration):

gradient = f\_gradient(x[random\_index[i],:], y[random\_index[i]], w, C)

moment = B\*moment + (1-B)\*gradient

Gt = gamma\*Gt + (1-gamma)\*gradient.dot(gradient.T)

a = lr \* math.sqrt(1 - pow(gamma, iteration)) / (1-pow(B, iteration))

w -= a \* moment / math.sqrt(Gt + 1e-8)

loss = f\_loss(x, y, w, C, random\_index)

lossvalue.append(loss)

testloss.append(f\_loss(x\_test, y\_test, w, C, random\_test\_index))

if loss < threshold :

break

return w, lossvalue, testloss

x, y\_train = load\_svmlight\_file("F:\\machinelearning\\a9a.txt")

x\_train = x.toarray()

x, y\_test = load\_svmlight\_file("F:\\machinelearning\\a9at.txt")

x\_test = x.toarray()

X\_train = numpy.hstack([x\_train, numpy.ones((x\_train.shape[0], 1))])

X\_test = numpy.hstack([x\_test, numpy.zeros((x\_test.shape[0], 1))])

X\_test = numpy.hstack([X\_test, numpy.ones((x\_test.shape[0], 1))])

iteration = 1000

# NAG

NAG\_w = numpy.zeros((1, X\_train.shape[1]))

NAG\_w, NAG\_loss, NAGtest\_loss = NAG\_train(X\_train, y\_train, X\_test, y\_test, NAG\_w, 0.3, 0.001, 0.9, 0.001, iteration)

# RMSProp

RMS\_w = numpy.zeros((1, X\_train.shape[1]))

RMS\_w, RMS\_loss, RMStest\_loss = RMSProp\_train(X\_train, y\_train, X\_test, y\_test, RMS\_w, 0.3, 0.001, 0.9, 0.001, iteration)

# AdaDelta

AdaDelta\_w = numpy.zeros((1, X\_train.shape[1]))

AdaDelta\_w, AdaDelta\_loss, AdaDeltatest\_loss = AdaDelta\_train(X\_train, y\_train, X\_test, y\_test, AdaDelta\_w, 0.3, 0.001, 0.9, 0.001, iteration)

# Adam

Adam\_w = numpy.zeros((1, X\_train.shape[1]))

Adam\_w, Adam\_loss, Adamtest\_loss = Adam\_train(X\_train, y\_train, X\_test, y\_test, Adam\_w, 0.3, 0.001, 0.9, 0.001, iteration)

pyplot.plot(NAGtest\_loss, label = 'NAG\_loss')

pyplot.plot(RMStest\_loss, label = 'RMSProp\_loss')

pyplot.plot(AdaDeltatest\_loss, label = 'AdaDelta\_loss')

pyplot.plot(Adamtest\_loss, label = 'Adam\_loss')

pyplot.legend(loc='upper right')

pyplot.ylabel('loss')

pyplot.xlabel('iteration')

pyplot.title('model')

pyplot.show()

2.线性分类与随机梯度下降

import numpy

import random

import jupyter

import math

from sklearn.datasets import load\_svmlight\_file

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

from matplotlib import pyplot

def f\_gradient(x, y, w):

gradient = x \* (y - x.dot(w.T))

return gradient

def f\_loss(x, y, w, random\_i):

loss = 0

a = len(random\_i)

for m in range(a):

loss += 0.5 \* ((y[random\_i[m]] - x[random\_i[m],:].dot(w.T)) \*\* 2)

return loss/a

def NAG\_train(x, y, x\_test, y\_test, w, C, lr, gamma, threshold, iteration):

vt = numpy.zeros(w.shape)

lossvalue = []

testloss = []

random\_index = []

random\_test\_index = []

for i in range(iteration):

random\_num = random.randint(0, x.shape[0]-1)

random\_test\_num = random.randint(0, x\_test.shape[0]-1)

random\_index.append(random\_num)

random\_test\_index.append(random\_test\_num)

for i in range(iteration):

gradient = f\_gradient(x[random\_index[i],:], y[random\_index[i]], w-gamma\*vt)

vt = gamma\*vt - lr\*gradient

w -= vt

loss = f\_loss(x, y, w, random\_index)

lossvalue.append(loss)

testloss.append(f\_loss(x\_test, y\_test, w, random\_test\_index))

if loss < threshold :

break

return w, lossvalue, testloss

def RMSProp\_train(x, y, x\_test, y\_test, w, C, lr, gamma, threshold, iteration):

Gt = 0

lossvalue = []

testloss = []

random\_index = []

random\_test\_index = []

for i in range(iteration):

random\_num = random.randint(0, x.shape[0]-1)

random\_test\_num = random.randint(0, x\_test.shape[0]-1)

random\_index.append(random\_num)

random\_test\_index.append(random\_test\_num)

for i in range(iteration):

gradient = f\_gradient(x[random\_index[i],:], y[random\_index[i]], w)

Gt = gamma\*Gt + (1-gamma)\*gradient.dot(gradient.T)

w += lr \* gradient / math.sqrt(Gt+1e-8)

loss = f\_loss(x, y, w, random\_index)

lossvalue.append(loss)

testloss.append(f\_loss(x\_test, y\_test, w, random\_test\_index))

if loss < threshold :

break

return w, lossvalue, testloss

def AdaDelta\_train(x, y, x\_test, y\_test, w, C, lr, gamma, threshold, iteration):

Gt = 0

variable\_t = 0

lossvalue = []

testloss = []

random\_index = []

random\_test\_index = []

for i in range(iteration):

random\_num = random.randint(0, x.shape[0]-1)

random\_test\_num = random.randint(0, x\_test.shape[0]-1)

random\_index.append(random\_num)

random\_test\_index.append(random\_test\_num)

for i in range(iteration):

gradient = f\_gradient(x[random\_index[i],:], y[random\_index[i]], w)

Gt = gamma\*Gt + (1-gamma)\*gradient.dot(gradient.T)

variable\_w = - math.sqrt(variable\_t + 1e-8) \* gradient / math.sqrt(Gt + 1e-8)

w -= variable\_w

variable\_t = gamma\*variable\_t + (1-gamma)\*variable\_w.dot(variable\_w.T)

loss = f\_loss(x, y, w, random\_index)

lossvalue.append(loss)

testloss.append(f\_loss(x\_test, y\_test, w, random\_test\_index))

if loss < threshold :

break

return w, lossvalue, testloss

def Adam\_train(x, y, x\_test, y\_test, w, C, lr, gamma, threshold, iteration):

Gt = 0

moment = numpy.zeros((1, x.shape[1]))

B = 0.9

lossvalue = []

testloss = []

random\_index = []

random\_test\_index = []

for i in range(iteration):

random\_num = random.randint(0, x.shape[0]-1)

random\_test\_num = random.randint(0, x\_test.shape[0]-1)

random\_index.append(random\_num)

random\_test\_index.append(random\_test\_num)

for i in range(iteration):

gradient = f\_gradient(x[random\_index[i],:], y[random\_index[i]], w)

moment = B\*moment + (1-B)\*gradient

Gt = gamma\*Gt + (1-gamma)\*gradient.dot(gradient.T)

a = lr \* math.sqrt(1 - pow(gamma, iteration)) / (1-pow(B, iteration))

w += a \* moment / math.sqrt(Gt + 1e-8)

loss = f\_loss(x, y, w, random\_index)

lossvalue.append(loss)

testloss.append(f\_loss(x\_test, y\_test, w, random\_test\_index))

if loss < threshold :

break

return w, lossvalue, testloss

x, y\_train = load\_svmlight\_file("F:\\machinelearning\\a9a.txt")

x\_train = x.toarray()

x, y\_test = load\_svmlight\_file("F:\\machinelearning\\a9at.txt")

x\_test = x.toarray()

X\_train = numpy.hstack([x\_train, numpy.ones((x\_train.shape[0], 1))])

X\_test = numpy.hstack([x\_test, numpy.zeros((x\_test.shape[0], 1))])

X\_test = numpy.hstack([X\_test, numpy.ones((x\_test.shape[0], 1))])

iteration = 1000

# NAG

NAG\_w = numpy.zeros((1, X\_train.shape[1]))

NAG\_w, NAG\_loss, NAGtest\_loss = NAG\_train(X\_train, y\_train, X\_test, y\_test, NAG\_w, 0.3, 0.001, 0.9, 0.001, iteration)

# RMSProp

RMS\_w = numpy.zeros((1, X\_train.shape[1]))

RMS\_w, RMS\_loss, RMStest\_loss = RMSProp\_train(X\_train, y\_train, X\_test, y\_test, RMS\_w, 0.3, 0.001, 0.9, 0.001, iteration)

# AdaDelta

AdaDelta\_w = numpy.zeros((1, X\_train.shape[1]))

AdaDelta\_w, AdaDelta\_loss, AdaDeltatest\_loss = AdaDelta\_train(X\_train, y\_train, X\_test, y\_test, AdaDelta\_w, 0.3, 0.001, 0.9, 0.001, iteration)

#Adam

Adam\_w = numpy.zeros((1, X\_train.shape[1]))

Adam\_w, Adam\_loss, Adamtest\_loss = Adam\_train(X\_train, y\_train, X\_test, y\_test, Adam\_w, 0.3, 0.001, 0.9, 0.001, iteration)

pyplot.plot(NAGtest\_loss, label = 'NAG\_loss')

pyplot.plot(RMStest\_loss, label = 'RMSProp\_loss')

pyplot.plot(AdaDeltatest\_loss, label = 'AdaDelta\_loss')

pyplot.plot(Adamtest\_loss, label = 'Adam\_loss')

pyplot.legend(loc='upper right')

pyplot.ylabel('loss')

pyplot.xlabel('iteration')

pyplot.title('model')

pyplot.show()

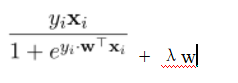
## 模型参数的初始化方法:

都是全零初始化

## 选择的loss函数及其导数:

1. 逻辑回归与随机梯度下降

loss函数：

导数：

1. 线性分类与随机梯度下降

loss函数：

导数：令

导数为：

## 实验结果和曲线图:

## 超参数选择：

1.逻辑回归与随机梯度下降

NAG：λ=0.3 learning rate=0.001

gamma = 0.9 threshold=0.001

RMSProp：λ=0.3 learning rate=0.001

gamma = 0.9 threshold=0.001

AdaDelta：λ=0.3 learning rate=0.001

gamma = 0.9 threshold=0.001

Adam：λ=0.3 learning rate=0.001

gamma = 0.9 threshold=0.001

2.线性分类与随机梯度下降

NAG：λ=0.3 learning rate=0.001

gamma = 0.9 threshold=0.001

RMSProp：λ=0.3 learning rate=0.001

gamma = 0.9 threshold=0.001

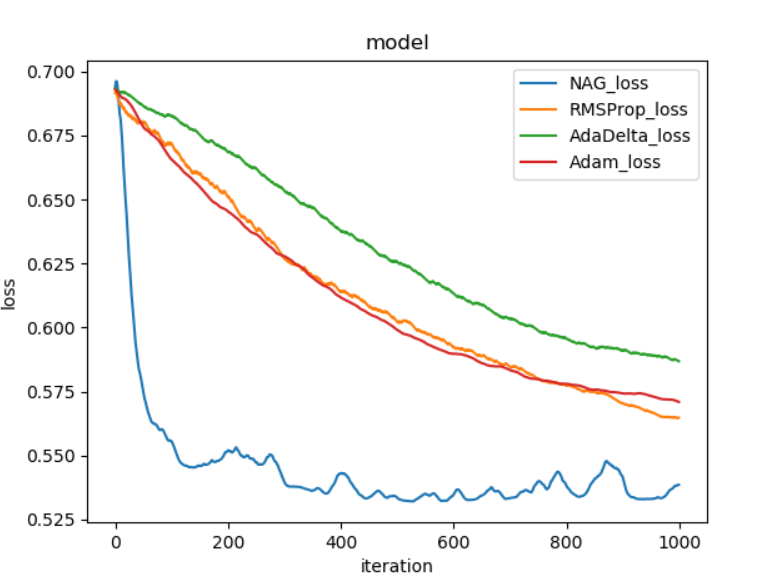
AdaDelta：λ=0.3 learning rate=0.001

gamma = 0.9 threshold=0.001

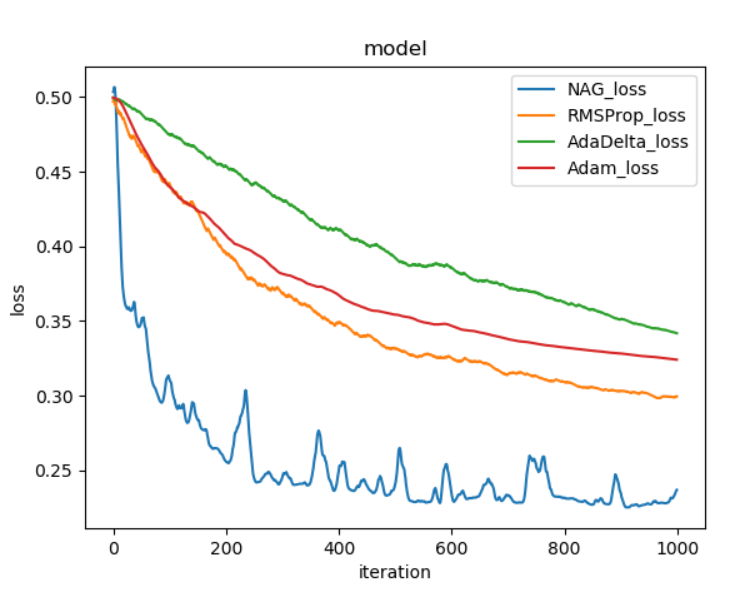
Adam：λ=0.3 learning rate=0.001

## 预测结果和loss曲线图：

1.逻辑回归与随机梯度下降



2.线性分类与随机梯度下降



## 实验结果分析:

从图中可以看到NAG优化算法梯度下降最快，也最不稳定，AdaDelta最慢但较稳定。

且由于采用的是随机梯度下降，曲线都有明显的起伏。

## 对比逻辑回归和线性分类的异同点：

同：都能用于分类用途

异：逻辑回归寻是找使模型出现的概率最大的参数集w的方法。显然，参数集w所确定的模型，其出现概率越大，模型的准确度越高,侧重于预测其概率。

线性分类是用一个线性函数来将样本点分开，着实的将数据划分为不同的一类。

## 实验总结：

这次试验中，我学到了随机梯度下降的方法来预测具有大规模的数据。并运用了其中的一些优化方法来改善随机下降的不稳定性。其次，实验当中免不了困难重重，调参问题、代码运行时间过长等等，都要我放平心态。本次实验也让我对分类问题有了更深入的了解，对如何解决这类问题有了一些经验。