

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**  郑铭莉

**学 号 201530613801**

**邮 箱 727457905@qq.com**

**指导教师** 吴庆耀

**提交日期** **2017年 12 月 15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: 郑铭莉

## 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。

## 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，，和。

7.重复步骤4-6若干次，画出，，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，，和。

7.重复步骤4-6若干次，画出，，和随迭代次数的变化图。

## 7. 代码内容:

逻辑回归：

def compute\_loss(w, x, y):

n = x.shape[0]

total = 0

for z in range(n):

total += np.log(1 + np.exp(np.sum(-y[z]\*x[z]\*w)))

loss = lamb \* np.sum(np.square(w)) / 2 + total/n

print(loss)

return loss

compute\_gradient:

for i in range(iteration):

num = 0

w\_gradient = np.zeros((x\_train.shape[1],1))

# parts of samples

pad = np.random.randint(1, 5)

for j in range(0, x\_train.shape[0], pad):

num += 1

temp = (1 + np.exp(np.sum(y\_train[j] \* x\_train[j] \* w)))

w\_gradient += np.sum(y\_train[j] / temp) \* x\_train[j].T

dw = learning\_rate \* lamb \* w - learning\_rate \* w\_gradient / num

线性分类：

def compute\_loss(w, b, x, y):

n = x.shape[1]

total = 0

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

if np.sum((1 - y[z] \* (x[z] \* w + b[z]))) > 0:

total += np.sum((1 - y[z] \* (x[z] \* w + b[z])))

loss = np.sum(np.square(w)) / (2\*n) + C \* total

print(loss)

return loss

compute\_gradient:

for i in range(iteration):

w\_gradient = np.zeros((x\_train.shape[1],1))

b\_gradient = 0

# parts of samples

pad = np.random.randint(1, 5)

for j in range(0, x\_train.shape[0], pad):

if np.sum((1 - y\_train[j] \* (x\_train[j] \* w + b[j]))) > 0:

w\_gradient += x\_train[j].T \* (-1 \* y\_train[j])

b\_gradient += -y\_train[j]

else:

w\_gradient += 0

b\_gradient += 0

dw = w + C \* w\_gradient

db = C \* b\_gradient

## 8. 模型参数的初始化方法:

逻辑回归：

init\_w = np.ones((x\_train.shape[1],1))

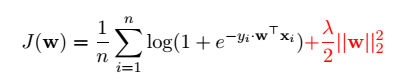
线性分类：

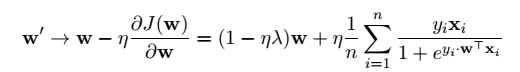
init\_b = np.zeros((x\_train.shape[0],1))

init\_w = np.ones((x\_train.shape[1],1))

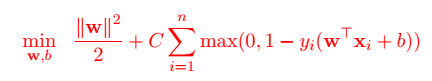
## 9.选择的loss函数及其导数:

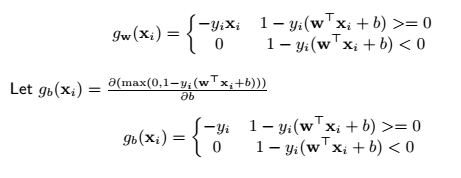
逻辑回归：

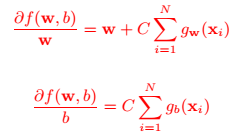




线性回归：







## 10.实验结果和曲线图:

## 超参数选择：

逻辑回归：

epoch = 100，learning\_rate = 0.1，λ = 0.01

线性分类：

epoch= 80，learning\_rate = 0.001，C = 0.01

*NAG：*

v=0, μ=0.9

*RMSprop：*

decay\_rate =0.9， eps = 1e-6，cache\_w = 0

*Adam：*

eps=1e-8，beta1=0.9, beta2=0.999

*AdaDelta：*

eps = 1e-8，decay\_rate = 0.9)

## 预测结果（最佳结果）：

逻辑回归：

= 0.585180861812，= 0.585174815604，= 0.590506979505

= 1.46940294028（多加训练100轮后为0.585193154449）

线性分类：

= 57.8448780028，= 57.7124158522，

= 57.6283568982，= 58.2194175894

## loss曲线图：

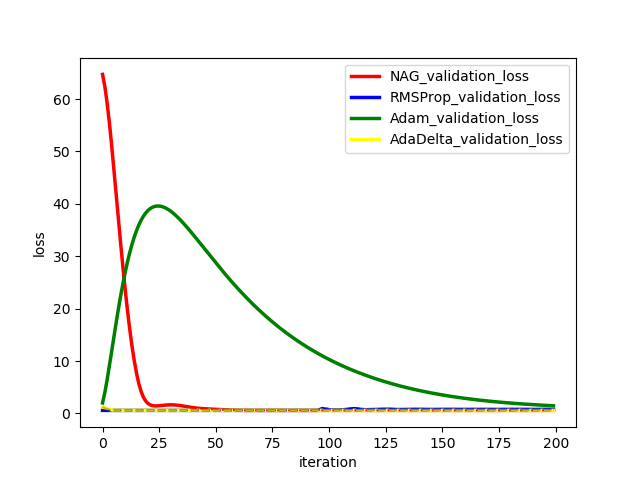


图1.逻辑回归

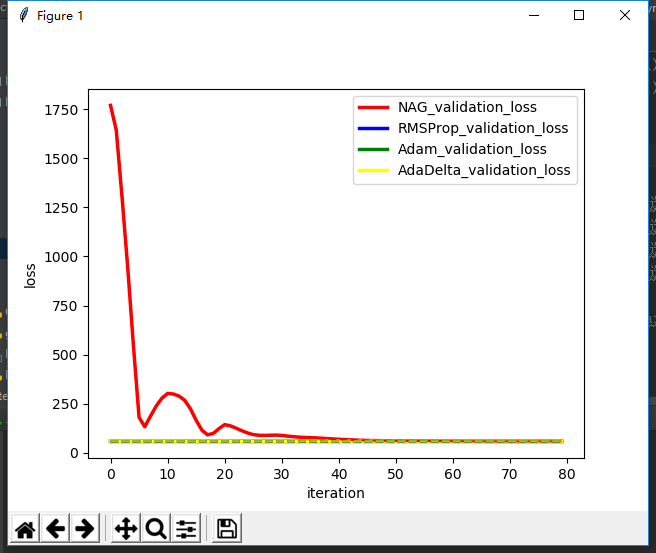


图2.线性分类（注：蓝、绿、黄线重叠）

## 11.实验结果分析:

逻辑回归中收敛速度很快，且最终的误差也较小，效果不错。总体比较，NAG在前二十几轮的下降速度很快，后面平缓收敛。Adam前期不稳定，有较大波动，随后逐渐收敛。RMSProp最开始就已经很接近最小值，然后缓慢趋近最小值。AdaDelta与RMSProp的效果差不多。

线性分类中NAG前几次迭代下降速度很快，后面逐渐稳定。而另外三种优化算法一开始就接近最优解，效果表现得差不多。

## 12.对比逻辑回归和线性分类的异同点：

相同的：

SVM：支持向量机SVM(Support Vector Machine）作为一种可训练的机器学习方法，依靠小样本学习后的模型参数进行导航星提取，可以得到分布均匀且恒星数量大为减少的导航星表

logistic：logistic回归(Logistic regression) 与多重线性回归实际上有很多相同之处，最大的区别就在于他们的因变量不同，其他的基本都差不多，正是因为如此，这两种回归可以归于同一个家族，即广义线性模型（generalized linear model）

不同点：

① 寻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——所以叫“支持向量机”。

② SVM可以处理非线性的情况

比logistic更强大的是，SVM还可以处理非线性的情况。

## 13.实验总结：

本次实验最大的收获是对各种优化算法有了更深的理解和体会，四个优化算法由于课件上没有提及需要去查找资料，网上有很多公式，但都有些误导，最后从官方网址和具有权威性的论文里得到优化的算法。

调参仍然是一个比较大的工作量，需要设定比较大的训练轮次使结果收敛稳定，进行参数对比，光这一个工作就花费不少时。

另外，在写代码的过程中也出过不少差错，如矩阵乘法、数据类型、维度等等运行时错误和逻辑错误，以及批量梯度下降求解过程折腾比较久最后才写明白。