

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员 刘海旭**

**学 号 201530612347**

**邮 箱 1059350952@qq.com**

**指导教师**  **谭明奎**

**提交日期 2017年12月8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: 刘海旭

## 4. 实验目的: 进一步理解线性回归和梯度下降的原理。

在小规模数据集的上实践。

体会优化和调参的过程。

## 5. 数据集以及数据分析：线性回归使用的是LIBSVM Data中的Housing数据，线性分类使用的是LIBSVM Data中的australian数据。

## 6. 实验步骤:

读取实验数据，使用sklearn库的load\_svmlight\_file函数读取数据。

将数据集切分为训练集和验证集，本次实验不切分测试集。使用train\_test\_split函数切分数据集。

实验1线性模型初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。

实验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。

选择Loss函数及对其求导，求得所有样本对Loss函数的梯度G。

取梯度G的负方向，记为D。

更新参数模型，Wt = Wt-1 + a \* D, a为学习率，是人为调整的超参数。

在训练集上测试并得到Loss的函数值Ltrain，在验证集上测试并得到Loss函数值Lvalidation。

实验2选择合适的阈值，将计算结果大于阈值的标记为正类，反之为负类。

画出Ltrain和Lvalidation随迭代次数的变化图。

## 7. 代码内容:

线性回归：

# 数据读取

data = load\_svmlight\_file("housing\_scale.txt")

# 数据切分

data\_train, data\_test, value\_train, value\_test = train\_test\_split(data[0], data[1], test\_size=0.33)

# 初始化

learning\_rate = 0.0005

initial\_w = random.random(size=(14, 1))

num\_iter = 1000

# 偏移值

bias\_train = np.mat(np.ones(shape=[data\_train.shape[0], 1]))

bias\_train = csr\_matrix(bias\_train).todense()

bias\_test = np.mat(np.ones(shape=[data\_test.shape[0], 1]))

bias\_test = csr\_matrix(bias\_test).todense()

# 训练集矩阵

x\_mat\_train = csr\_matrix(data\_train).todense()

x\_mat\_train = np.hstack((x\_mat\_train, bias\_train))

y\_train = np.mat(value\_train).T

y\_mat\_train = csr\_matrix(y\_train).todense()

# 验证集矩阵

x\_mat\_test = csr\_matrix(data\_test).todense()

x\_mat\_test = np.hstack((x\_mat\_test, bias\_test))

y\_test = np.mat(value\_test).T

y\_mat\_test = csr\_matrix(y\_test).todense()

# 求梯度

initial\_w = csr\_matrix(initial\_w).todense()

gradient = - np.dot(x\_mat\_train.T, y\_mat\_train) + np.dot(np.dot(x\_mat\_train.T, x\_mat\_train), initial\_w)

d = - gradient

w = initial\_w

# 用列表表示每一次迭代的损失率

loss\_list\_train = []

loss\_list\_test = []

# 梯度下降

for i in range(num\_iter):

# w，d的迭代更新

w = w + learning\_rate \* d

d = np.dot(x\_mat\_train.T, y\_mat\_train) - np.dot(np.dot(x\_mat\_train.T, x\_mat\_train), w)

# d\_test = np.dot(x\_mat\_test.T, y\_mat\_test) - np.dot(np.dot(x\_mat\_test.T, x\_mat\_test), w)

loss\_train = np.dot((y\_mat\_train - np.dot(x\_mat\_train, w)).T, (y\_mat\_train - np.dot(x\_mat\_train, w))) / 2 / 339

loss\_test = np.dot((y\_mat\_test - np.dot(x\_mat\_test, w)).T, (y\_mat\_test - np.dot(x\_mat\_test, w))) / 2 / 167

loss\_list\_train.append(loss\_train.tolist()[0][0])

loss\_list\_test.append(loss\_test.tolist()[0][0])

线性分类：

# 数据读取

data = load\_svmlight\_file("australian\_scale.txt")

# 数据切分

data\_train, data\_test, value\_train, value\_test = train\_test\_split(data[0], data[1], test\_size=0.33)

# 初始化

learning\_rate = 0.0005

initial\_w = random.random(size=(15, 1))

num\_iter = 1000

# 偏移值

bias\_train = np.mat(np.ones(shape=[data\_train.shape[0], 1]))

bias\_train = csr\_matrix(bias\_train).todense()

bias\_test = np.mat(np.ones(shape=[data\_test.shape[0], 1]))

bias\_test = csr\_matrix(bias\_test).todense()

# 训练集矩阵

x\_mat\_train = csr\_matrix(data\_train).todense()

x\_mat\_train = np.hstack((x\_mat\_train, bias\_train))

y\_train = np.mat(value\_train).T

y\_mat\_train = csr\_matrix(y\_train).todense()

# 验证集矩阵

x\_mat\_test = csr\_matrix(data\_test).todense()

x\_mat\_test = np.hstack((x\_mat\_test, bias\_test))

y\_test = np.mat(value\_test).T

y\_mat\_test = csr\_matrix(y\_test).todense()

# 求梯度

initial\_w = csr\_matrix(initial\_w).todense()

gradient = - np.dot(x\_mat\_train.T, y\_mat\_train) + np.dot(np.dot(x\_mat\_train.T, x\_mat\_train), initial\_w)

d = - gradient

w = initial\_w

# 用列表表示每一次迭代的损失率

loss\_list\_train = []

loss\_list\_test = []

# 梯度下降

for i in range(num\_iter):

# w，d的迭代更新

w = w + learning\_rate \* d

d = np.dot(x\_mat\_train.T, y\_mat\_train) - np.dot(np.dot(x\_mat\_train.T, x\_mat\_train), w)

# d\_test = np.dot(x\_mat\_test.T, y\_mat\_test) - np.dot(np.dot(x\_mat\_test.T, x\_mat\_test), w)

loss\_train = np.dot((y\_mat\_train - np.dot(x\_mat\_train, w)).T, (y\_mat\_train - np.dot(x\_mat\_train, w))) / 2 / 339

loss\_test = np.dot((y\_mat\_test - np.dot(x\_mat\_test, w)).T, (y\_mat\_test - np.dot(x\_mat\_test, w))) / 2 / 167

loss\_list\_train.append(loss\_train.tolist()[0][0])

loss\_list\_test.append(loss\_test.tolist()[0][0])

# 线性分类

negative\_class = []

positive\_class = []

for i in range(x\_mat\_train.shape[0]):

hyperplane = np.dot(w.T, x\_mat\_train[i].T).tolist()

if hyperplane[0][0] > 0:

positive\_class.append(hyperplane[0][0])

else:

negative\_class.append(hyperplane[0][0])

print("positive class is: ", positive\_class)

print("/n/n/n/n")

print("negative class is: ", negative\_class)

## 8. 选择的评估方法（留出法，交叉验证，k折交叉验证等）:

交叉验证法

## 9. 模型参数的初始化方法:

学习率初始化：learning\_rate = 0.0005

W初始化：initial\_w = random.random(size=(15, 1))

迭代轮数初始化：num\_iter = 1000

## 10.选择的loss函数及其导数:

Loss函数：1/2 \* 1/n \* (y - Xw)^T \* (y - Xw)

Loss函数导数：- X^Ty + X^T \* X \* w

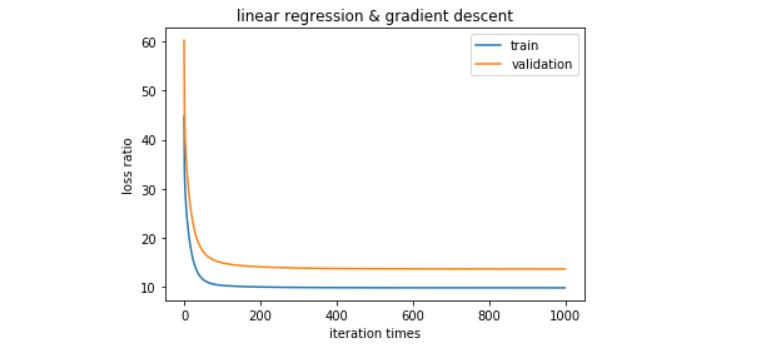
## 11.实验结果和曲线图:

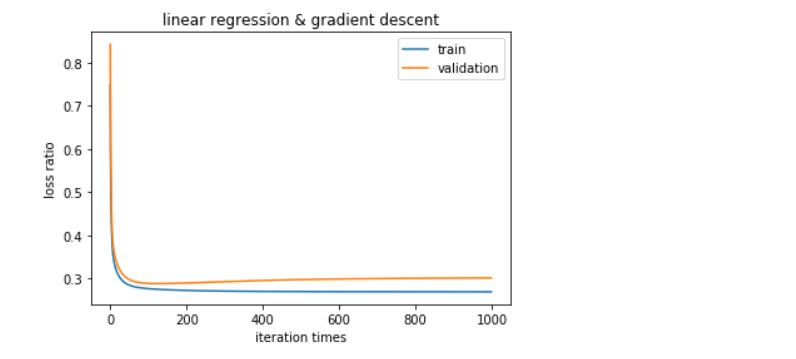
## 超参数选择（η,epoch等）：

## 评估结果（根据选择的评估方法）：

## 预测结果（最佳结果）：

## loss曲线图：





## 12.实验结果分析:

由图像可知，当训练次数足够大时，损失率降低至一个较为稳定的数值，训练集训练的效果比验证集要好，因而当测试集数据与训练集偏差较大时，训练出来的学习率，w等数据都有可能存在一定的误差，倒是损失率偏高。

## 13.对比线性回归和线性分类的异同点：

线性分类与线性回归的相似性在于我们都采用相同的模型，即y = w^T\*x+b,同时我们都需要对相同的损失函数 loss = 1/2 \* 1/n \* (y - Xw)^T \* (y - Xw)进行求导，得到梯度- X^Ty + X^T \* X \* w。不同点在于，线性回归要求我们采用梯度下降的方法不断优化更新w，从而训练出较为合理的拟合模型，而线性分类要求我们回到原来的y = w^T\*x+b，根据计算结果与阈值相比较将其进行二分类，同时线性分类涉及到了支持向量机的初始化和应用。

## 14.实验总结：

通过这次实验，我知道了如何用计算机语言模拟线性回归和线性分类方法，通过一次次的debug和修改代码，对线性分类，线性回归，梯度下降，支持向量机等各种机器学习的知识有了更深的了解。