

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**  **陈佳欣**

**学 号 201530611180**

**邮 箱 244821405@qq.com**

**指导教师**  **吴庆耀**

**提交日期** **2017年 12 月 2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:陈佳欣

## 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，包含506个样本，每个样本有13个属性。数据集为Housing\_scale

线性分类使用的是LIBSVM Data中的australian数据，包含690个样本，每个样本有14 个属性。数据集为austalian\_scale

## 6. 实验步骤:

**线性回归和梯度下降**

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

2.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。

3.线性模型参数初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。

4.选择Loss函数及对其求导，过程详见课件ppt。

5.求得所有样本对Loss函数的梯度。

6.取梯度的负方向，记为。

7.更新模型参数，。为学习率，是人为调整的超参数。

8.在训练集上测试并得到Loss函数值，在验证集上测试并得到Loss函数值。

9.重复步骤5-8若干次，画出和随迭代次数的变化图。

**线性分类和梯度下降**

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

2.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。

3.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。

4.选择Loss函数及对其求导，过程详见课件ppt。

5.求得所有样本对Loss函数的梯度。

6.取梯度的负方向，记为。

7.更新模型参数，。为学习率，是人为调整的超参数。

8.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。在训练集上测试并得到Loss函数值，在验证集上测试并得到Loss函数值。

9.重复步骤5-8若干次，画出和随迭代次数的变化图。

## 7. 代码内容:

**线性回归：**

import numpy as np

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

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

import matplotlib.pyplot as plt

mem = Memory('./mycache')

#读取数据

def get\_data():

data = load\_svmlight\_file('housing\_scale')

return data[0],data[1]

X, y = get\_data()

X\_train,X\_validation,y\_train,y\_validation = train\_test\_split(X,y,test\_size = 0.33,random\_state = 42)#数据分割

X\_train = X\_train.todense()

X\_validation = X\_validation.todense()

y\_train = y\_train.reshape(y\_train.shape[0],1)

y\_validation = y\_validation.reshape(y\_validation.shape[0],1)

m,n = np.shape(X\_train)

w = np.ones((n,1))

learning\_rate = 0.0003

def loss\_function(x,y,w):

loss = (1/2 \*(np.linalg.norm(x\*w-y))\*\*2)/ (x.shape[0])

return loss

train\_loss = []

validation\_loss = []

iteration =1000

for i in range(0,iteration):

gradient = X\_train.T \* ( X\_train \* w - y\_train) #x(wx-y)

#print(gradient)

w = w - learning\_rate \* gradient

#print(w)

train\_loss.append(loss\_function(X\_train,y\_train,w))

validation\_loss.append(loss\_function(X\_validation,y\_validation,w))

a = np.arange(0,iteration,1)

plt.plot(a,train\_loss,label = "train\_loss",color = "yellow",linewidth = 2)

plt.plot(a,validation\_loss,label = 'validation\_loss',color = 'blue',linewidth = 2)

plt.legend(loc='upper right')

plt.xlabel("Iteration")

plt.ylabel("Loss")

plt.show()

**线性分类:**

import numpy as np

from sklearn.datasets import load\_svmlight\_file

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

import matplotlib.pyplot as plt

def get\_data():

data = load\_svmlight\_file('australian\_scale')

return data[0],data[1]

X , Y = get\_data()

X\_train,X\_validation,Y\_train,Y\_validation = train\_test\_split(X , Y , test\_size = 0.33)

X\_train = X\_train.todense()

X\_validation = X\_validation.todense()

Y\_train = Y\_train.reshape(Y\_train.shape[0],1)

Y\_validation = Y\_validation.reshape(Y\_validation.shape[0],1)

m , n = np.shape(X\_train)

b = np.zeros((X\_train.shape[0],1))

w = np.empty((X\_train.shape[1],1))

def compute\_loss(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

iteration = 1000

learning\_rate = 0.5

C = 0.01

train\_loss=[]

validation\_loss=[]

for i in range(iteration):

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

if np.sum((1 - Y[j] \* (X[j] \* w + b[j] ))) > 0:

w\_gradient = w + C \* (-1 \* Y\_train[j] \* X\_train[j]).T

b\_gradient = -1 \* C \* Y\_train[j]

else:

w\_gradient = w

b\_gradient = 0

w = w - learning\_rate \* w\_gradient

b[j] = b[j] - learning\_rate \* b\_gradient

train\_loss.append( compute\_loss(X\_train,Y\_train) )

validation\_loss.append( compute\_loss(X\_validation,Y\_validation) )

a = np.arange(0,iteration,1)

plt.plot(a,train\_loss,label = "train\_loss",color = "yellow",linewidth = 2)

plt.plot(a,validation\_loss,label = 'validation\_loss',color = 'blue',linewidth = 2)

plt.legend(loc='upper right')

plt.xlabel("Iteration")

plt.ylabel("Loss")

plt.show()

## 8. 选择的评估方法（留出法，交叉验证，k折交叉验证等）:

线性回归：留出法

线性分类：留出法

## 9. 模型参数的初始化方法:

线性回归：

m,n = np.shape(X\_train) m,n为矩阵的行数和列数

w = np.ones((n,1)) w全部赋值为1

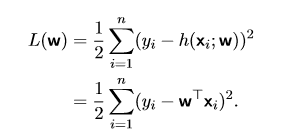
线性分类：

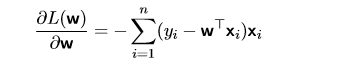
b = np.zeros((X\_train.shape[0],1)) b赋值为0

w = np.empty((X\_train.shape[1],1)) w随机赋值

## 10.选择的loss函数及其导数:

线性回归：





线性分类：







## 11.实验结果和曲线图:

## 超参数选择（η,epoch等）：

线性回归：η=0.01

线性分类：η=0.05，C=0.01

## 评估结果（根据选择的评估方法）：

线性回归：迭代次数不变，更改η

线性分类：迭代次数不变，控制C=0.01，更改η

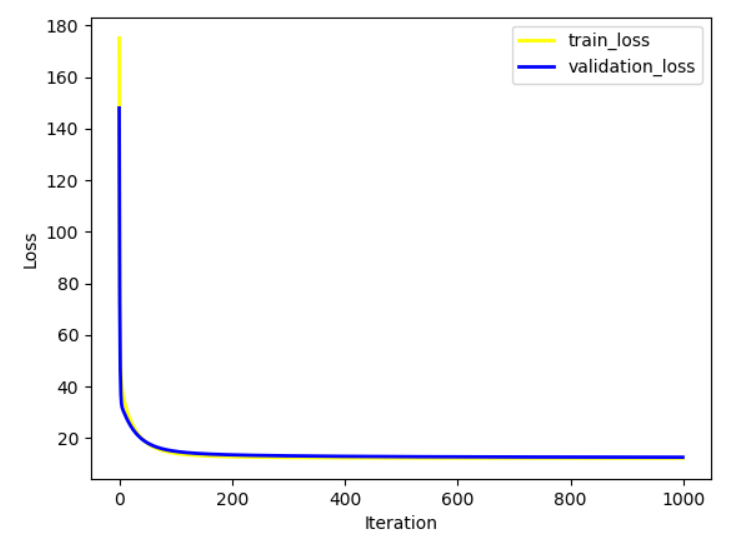
## 预测结果（最佳结果）：

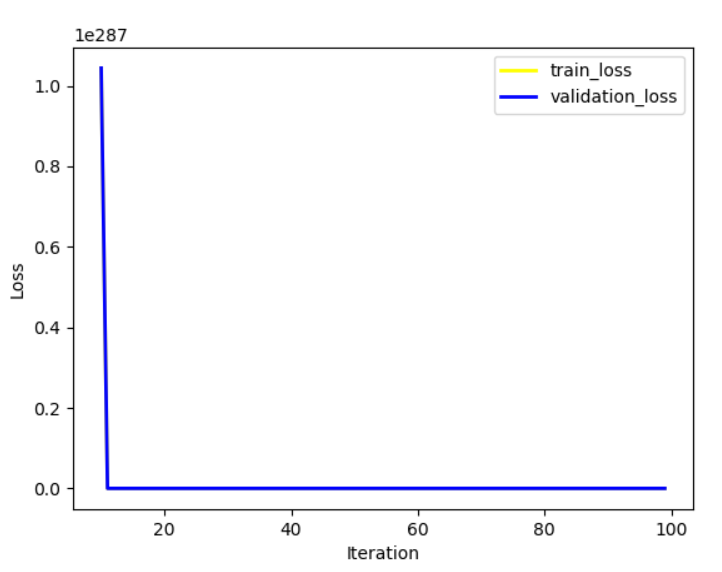
线性回归：η = 0.0003

线性分类：η = 0.05，C = 0.01

## loss曲线图：

线性回归：

 线性分类：



## 12.实验结果分析:

线性回归：经过不断地训练之后，和越来越小，并且很接近。

线性分类：经过训练之后，和越来越下，并趋于稳定

## 13.对比线性回归和线性分类的异同点：

总的来说两个问题本质上都是一致的，就是模型与数据的拟合。 但是分类问题的y值(也称为label), 更离散化一些. 而且， 同一个y值可能对应着一大批的x, 这些x是具有一定范围的。

所以分类问题更多的是 (一定区域的一些x) 对应 着 (一个y). 而回归问题的模型更倾向于 (很小区域内的x，或者一般是一个x) 对应着 (一个y).

## 14.实验总结：

通过这次实验，熟悉了线性回归与线性分类的实现过程，以及理论知识。对于回归和分类的异同体会更加深刻。

感受到调参这个过程的重要性，在公式以及算法实现后，更重要的就是调参来调整训练模型。

在这个过程中，对machine learning的小套路有了更好的把握。