

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**  **黄飞勤**

**学 号 201530611678**

**邮 箱 frankieafei@qq.com**

**指导教师**  **吴庆耀老师**

**提交日期** **2017年 12 月 7 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: 黄飞勤

## 4. 实验目的: 使用梯度下降方法实现线性回归和线性分类，以及SVM，并在实验过程中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。

线性分类使用的是LIBSVM Data中的australian数据，包好690个样本，每个样本有14个属性。

本次实验我将数据集以七比三的比例分成了训练集和验证集。

## 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G。

6. 取梯度G的负方向，记为D。

7. 更新模型参数。

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，画出两个Loss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G。

6. 取梯度的负方向G，记为D。

7. 更新模型参数。

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，画出两个Loss随迭代次数的变化图。

## 7. 代码内容:

线性回归：

from numpy import \*

import torch

import random

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

# 数据预处理，训练集和验证集七三分

dtype = torch.FloatTensor

mem = Memory("./mycache")

@mem.cache

def get\_data():

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

return data[0], data[1]

x, y = get\_data()

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.30, random\_state=42)

# 112x13

x\_test = x\_test.toarray()

x\_test = torch.from\_numpy(x\_test).type(dtype)

# 394x13

x\_train = x\_train.toarray()

x\_train = torch.from\_numpy(x\_train).type(dtype)

# 112x1

y\_test = array(y\_test).reshape(size(y\_test,0),1)

y\_test = torch.from\_numpy(y\_test).type(dtype)

# 394x1

y\_train = array(y\_train).reshape(size(y\_train,0),1)

y\_train = torch.from\_numpy(y\_train).type(dtype)

# 最大特征数 13

num\_feature = max(x\_train.size(1), x\_test.size(1))

bias = torch.ones(1, x\_train.size(0))

bias = bias.t()

x\_train = torch.cat((x\_train, bias), 1)

bias = torch.ones(1, x\_test.size(0))

bias = bias.t()

x\_test = torch.cat((x\_test, bias), 1)

w = []

# 全零初始化

w.append(torch.zeros(num\_feature + 1, 1))

#学习率

l\_r = 0.1

#训练集规模

len\_train = x\_train.size(0)

# 验证集规模

len\_test = x\_test.size(0)

#迭代次数

time\_iteration = 20

Y\_train = []

Y\_vali = []

# 开始迭代

for k in range(time\_iteration):

Loss\_sum = 0

for l in range(len\_train):

h = torch.mm(w[k][:,:].t(), x\_train[l:l+1][:].t())

Loss\_sum += (h - y\_train[l]) \* x\_train[l]

direction = -(Loss\_sum / len\_train)

w.append(w[k] + l\_r\*direction.t())

Y\_train.append(torch.mean(Loss\_sum))

Loss\_vali = 0

for l in range(len\_test):

h = torch.mm(w[k][:,:].t(), x\_test[l:l+1][:].t())

Loss\_vali += (h - y\_test[l]) \* x\_test[l]

Y\_vali.append(torch.mean(Loss\_vali))

# %matplotlib inline

iteras = range(1, time\_iteration + 1)

plt.rcParams['figure.figsize'] = (10, 10)

plt.xlabel('iteration')

plt.ylabel('loss')

plt.title('Loss history')

plt.plot(iteras, Y\_train, label = "train loss")

plt.plot(iteras, Y\_vali, label = "validation loss")

plt.legend(loc = 'best')

plt.show()

线性分类：

from numpy import \*

import torch

import random

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

# 数据预处理，训练集和验证集七三分

dtype = torch.FloatTensor

mem = Memory("./mycache")

@mem.cache

def get\_data():

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

return data[0], data[1]

x, y = get\_data()

x\_train, x\_vali, y\_train, y\_vali = train\_test\_split(x, y, test\_size=0.30, random\_state=42)

# 483 x 14

x\_train = x\_train.toarray()

x\_train = torch.from\_numpy(x\_train).type(dtype)

# 483 x 1

y\_train = array(y\_train).reshape(size(y\_train,0),1)

y\_train = torch.from\_numpy(y\_train).type(dtype)

# 207 x 14

x\_vali = x\_vali.toarray()

x\_vali = torch.from\_numpy(x\_vali).type(dtype)

# 207 x 1

y\_vali = array(y\_vali).reshape(size(y\_vali,0),1)

y\_vali = torch.from\_numpy(y\_vali).type(dtype)

# 最大特征数 14

num\_feature = max(x\_train.size(1), x\_vali.size(1))

bias = torch.ones(1, x\_train.size(0))

bias = bias.t()

x\_train = torch.cat((x\_train, bias), 1)

bias = torch.ones(1, x\_vali.size(0))

bias = bias.t()

x\_vali = torch.cat((x\_vali, bias), 1)

w = []

# 全零初始化

w.append(torch.zeros(num\_feature + 1, 1))

# 阈值

threshold = 0

# 学习率

l\_r = 0.005

#超参数

tradeoff = 0.01

# 训练集规模

len\_train = x\_train.size(0)

# 验证集规模

len\_vali = x\_vali.size(0)

#迭代次数

time\_iteration = 1000

iteras = arange(1, time\_iteration+1)

Loss\_train = []

Loss\_vali = []

# 开始迭代

print("开始计算，迭代次数：" + str(time\_iteration))

for k in range(time\_iteration):

Loss\_sum = 0

grad\_E = 0

# 迭代训练集

for l in range(len\_train):

E = y\_train[l][0] \* torch.mm(w[k].t(), x\_train[l:l+1, :].t())

if E[0][0] < 1:

Loss\_sum += 1-E[0][0]

grad\_E += y\_train[l][0] \* x\_train[l:l+1, :].t()

grad = w[k] - tradeoff \* grad\_E

grad[-1][0] -= w[k][-1][0]

direction = -grad

w.append(w[k] + l\_r \* direction)

Loss\_sum \*= tradeoff

Loss\_sum += square(torch.norm(w[k])) / 2

Loss = Loss\_sum / len\_train

Loss\_train.append(Loss)

# 迭代验证集

Loss\_sum = 0

for l in range(len\_vali):

E = y\_vali[l][0] \* torch.mm(w[k].t(), x\_vali[l:l+1, :].t())

if E[0][0] < 1:

Loss\_sum += 1-E[0][0]

Loss\_sum \*= tradeoff

Loss\_sum += square(torch.norm(w[k])) / 2

Loss = Loss\_sum / len\_vali

Loss\_vali.append(Loss)

# 测试准确率

time = 0

for l in range(len\_vali):

predict = torch.mm(w[time\_iteration].t(), x\_vali[l:l+1, :].t())

if predict[0][0] > 0 :

predict = 1

else:

predict = -1

if predict == y\_vali[l][0]:

time += 1

print('准确率：' + str(time/len\_vali))

plt.rcParams['figure.figsize'] = (10, 10)

plt.xlabel('iteration')

plt.ylabel('loss')

plt.plot(iteras, Loss\_train, label = "train loss")

plt.plot(iteras, Loss\_vali, label = "validation loss")

plt.legend(loc = 'best')

plt.show()

（针对线性回归和线性分类分别填写8-12内容）

## 选择的评估方法（留出法，交叉验证，k折交叉验证等）:

对于**线性回归**和**线性分类**都是采用的 **留出法** 进行模型的评估。将数据集以七三的比例切分为训练集和验证集，训练集用于训练模型，而验证集用于评估模型。

## 模型参数的初始化方法:

**线性回归**：采用了全零初始化模型参数的方法。

**线性分类**：采用了全零初始化模型参数的方法。

## 选择的loss函数及其导数:

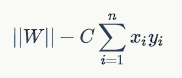
**线性回归：**

Loss函数是 dot( (Y - dot( W.T(), X ) ).T(), (Y - dot( W.T(), X ) ) ) / 2

梯度为 - dot( X.T, Y ) + dot( X.T, dot( W.T(), X) )

**线性分类：**

Loss函数：

梯度：

## 11.实验结果和曲线图:

## 超参数选择（η,epoch等）：

**线性回归**：学习率为0.1，迭代次数为20.

**线性分类**：学习率为0.005，迭代次数为1000，超参数C为0.01

## 评估结果（根据选择的评估方法）：

**线性回归**： 在第10次迭代后，训练集和验证集的Loss值已十分接近，并在之后接近0。

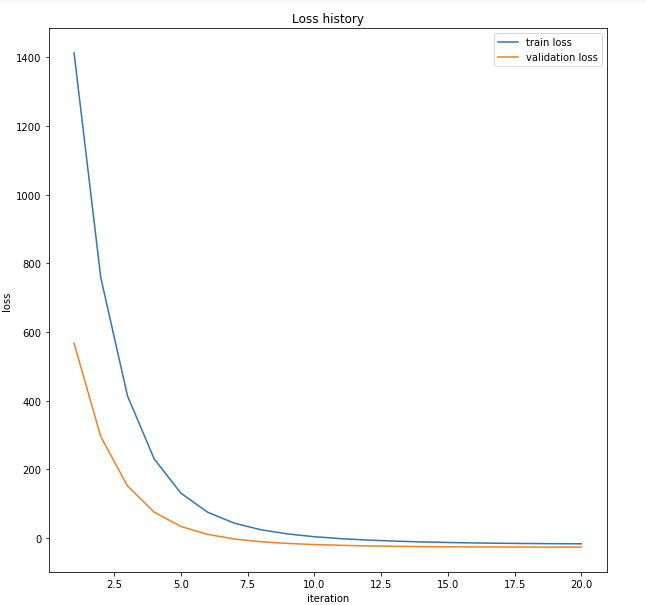
**线性分类**：训练集的Loss在大约第50次迭代降到2.0以下，并在之后稳定在大概1.9，验证集也在大约第50次迭代降到1.1，并在之后稳定在1.1和1.0之间。

## 预测结果（最佳结果）：

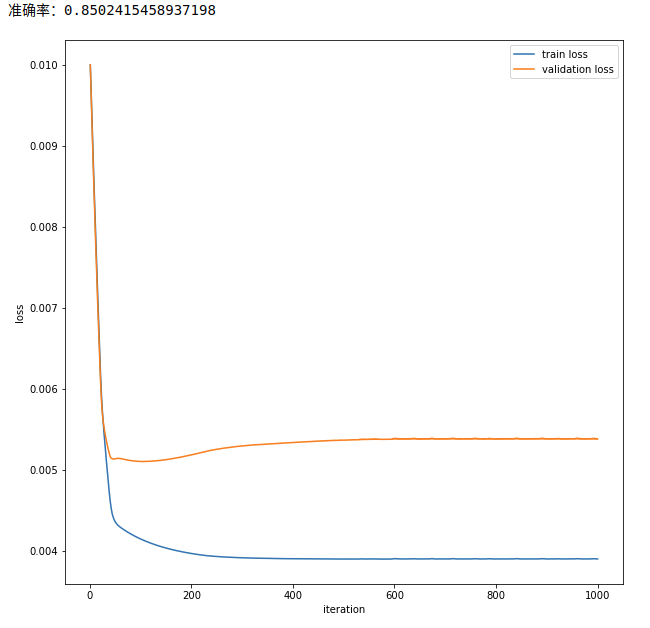
SVM的预测准确率约为0.85

## loss曲线图：

**线性回归：**



线性分类：



## 实验结果分析:

实验结果整体上还好，虽然数据规模比较小，但是还是有很多可以优化的地方，由于没有花太多时间在调参上，超参数也比较粗略。

**线性回归**的Loss函数，训练集和验证集上的都收敛得很快，并在后面稳步下降直到趋近于平行前进，整体看来，模型训练得很稳定，不会过度训练数据。

**线性分类**的Loss函数，训练集和验证集在第1-50次迭代之间下降得非常快，后面则趋于稳定，然而验证集后面的部分的Loss函数值一直大于训练集的，而且在第200次迭代后反而出现略微上升，这表明在200次迭代后数据训练过度了，需要控制下迭代次数。

## 对比线性回归和线性分类的异同点：

线性回归常用于分析两个变量X和Y之间的关系，一般应用于根据一个事物的特征值组去预测另一个值；而线性分类则是根据变量X去预测Y所属的区间（类别）。整体来说，线性回归和线性分类前面的步骤都差不多，主要的区别在于线性分类需要将最后的预测值归并到某个区间成为一个类别。

## 14.实验总结：

通过本次实验，更加深入了解线性回归和线性分类的原理和本质，并实践使用了梯度下降，通过不断降低Loss值去训练模型，同时还将课堂上将的SVM原理和公式应用于实践中。而对于梯度下降，实验过程中遇到了陷入了局部最优值，通过增大学习率和迭代次数解决了，尽管如此，实践证明，梯度下降也只是能不断逼近最优解，拿到的还是局部最优解中的优化解。