机器学习实验报告

学院：软件学院

专业：软件工程

作者:

侯斯扬

指导教师:

谭明奎

学号:

201930380488

班级:

软件 2 班

2021-10-7

线性回归与随机梯度下降

摘要—本实验主要时与线性回归有关的两部分：一个是求解线性回归中的闭式解，一个是随机梯度下降，并且在小规模的数据集（波士顿房价）上进行实践，体会优化和调参的过程

I. 介绍

本次实验，是为了进一步理解线性回归，闭式解和梯度下降的原理。

对于线性回归的方程的求解，采用闭式解可以得到最佳的参数，但是在实际的运用中，如果矩阵不可逆，或者当矩阵维度太大（对于矩阵求逆是一个复杂度很高的计算）时，不会选择进行闭式解的求解，而是利用梯度下降的方法迭代求解线性回归。（本次实验主要是随机梯度下降）

II. 方法和理论

A. 闭式解

本次实验我们选择的损失函数是

$$\mathcal{L}_D(\mathbf{w}, b) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i; \mathbf{w}, b))^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

其矩阵的表示形式为

$$\mathcal{L}_D(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i\mathbf{w})^2$$

$$= \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$= \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

线性回归中闭式解的求解如下

$$\mathcal{L}_D(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w}), \text{Let } \mathbf{a} = \mathbf{y} - \mathbf{X}\mathbf{w},$$

$$\frac{\partial \mathcal{L}_D(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \frac{\partial(\frac{1}{2}\mathbf{a}^T\mathbf{a})}{\partial \mathbf{a}}$$

$$= \frac{1}{2}\frac{\partial \mathbf{a}}{\partial \mathbf{w}}(2\mathbf{a})$$

$$= \frac{\partial(\mathbf{y} - \mathbf{X}\mathbf{w})}{\partial \mathbf{w}}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$= -\mathbf{X}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\frac{\partial \mathcal{L}_D(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{X}^{\mathrm{T}}\mathbf{y} + \mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w} = 0$$

$$\Longrightarrow \mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w} = \mathbf{X}^{\mathrm{T}}\mathbf{y}$$

$$\Longrightarrow \mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

最终可得闭式解为

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

B. 随机梯度下降

让由于闭式解往往不能直接求得，且让损失函数最小的最优解就是当导数为 0 时的解，所以我们只能通过在梯度的方向上不断尝试降低损失函数以此来对最小损失求精

III. 实验

A. 数据集

本次实验使用的是 LABSVM Data 中的 Housing 数据，包含 506 个样本，每个样本有 13 个属性；

并且按照 0.5 的比例切分训练集和验证集

B. 实现

1 导入相关的依赖包

```
import numpy as np
import pandas as pd
import sklearn.datasets as sd
import sklearn.model_selection as sms
import matplotlib.pyplot as plt
import math
import random
```

2 使用 sklearn 库的 load_svmlight_file 读取数据，切分为数据训练集和验证集（包括数据的预处理）

```
x, y=load_svmlight_file("data/housing_scale.txt")
```

```
# 划分训练集和验证集
train_X,test_X,train_y,test_y = train_test_split(X,Y,test_size = 0.5,random_state = 1)
```

```
# 对稀疏矩阵进行类型转换
X_train = X_train.toarray()
X_valid = X_valid.toarray()
y_train = y_train.reshape(len(y_train),1)
y_valid = y_valid.reshape(len(y_valid),1)
X_train.shape, X_valid.shape, y_train.shape, y_valid.shape
```

3 定义 loss 函数

```
def compute_cost(y_,y):
    m = y.shape[0]
    cost = np.sum(np.square(y_-y))/(2*m)
    return cost
```

4　求解闭式解

```
#计算w
def computeW(X,y):
    w = (X.T*X).I*X.T*y
    return w
```

```
# 求闭式解
w_train=computeW(train_X,train_y)
print("训练集: \n",w_train)

w_test=computeW(test_X,test_y)
print("测试集: \n",w_test)
```

求得的闭式解为

```
训练集:
[[22.04664032]
 [-0.99109301]
 [ 1.56537181]
 [-0.06652463]
 [ 0.39120819]
 [-2.08114709]
 [ 2.2799947 ]
 [ 0.3123258 ]
 [-2.99768235]
 [ 2.84535517]
 [-2.04428931]
 [-1.78206064]
 [ 0.77613042]
 [-3.95086247]]

测试集:
[[21.65720332]
 [-1.42570164]
 [ 0.62322042]
 [ 0.20386144]
 [ 1.116959  ]
 [-2.09583245]
 [ 2.76084236]
 [-0.27966032]
 [-3.50558602]
 [ 2.79006273]
 [-2.28357141]
 [-2.29942022]
 [ 0.81653894]
 [-4.08705092]]
```

5　最终，训练集和验证集上的 Loss 函数值为

```
Loss 12.135776624189537
Loss_train 11.038128867243747
Loss_test 10.258507283359462
```

6　定义梯度函数和下降函数

```
#定义梯度函数
def gradient(X, y, theta):
    return X.T.dot(X.dot(theta) - y)
```

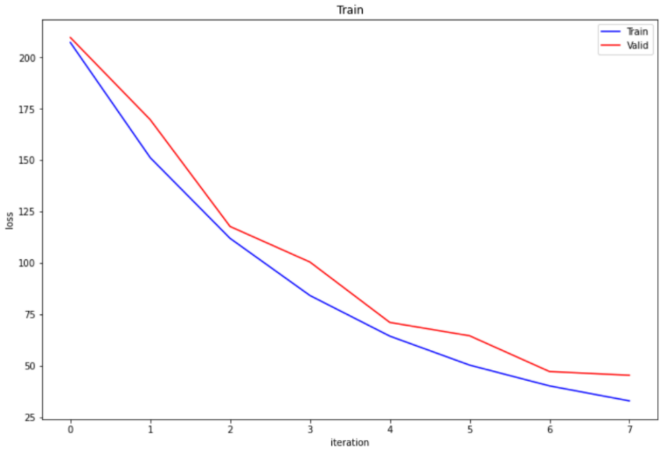
```
#定义下降函数
def descent(X, y, theta, alpha, iters, X_valid, y_valid):
    loss_train = np.zeros((iters,1))
    loss_valid = np.zeros((iters,1))
    for i in range(iters):
        grad = gradient(X, y, theta)
        theta = theta - alpha * grad
        loss_train[i] = compute_loss(X, y, theta)
        loss_valid[i] = compute_loss(X_valid, y_valid, theta)
    return theta, loss_train, loss_valid
```

7　梯度下降迭代求解

```
# 梯度下降
theta = np.zeros((14,1))
alpha = 0.001
iters = 8
opt_theta, loss_train, loss_valid = descent(X_train, y_train, theta, alpha, iters, X_valid, y_valid)
loss_train.min(), loss_valid.min()
```

```
(32.87629125429709, 45.348259487870116)
```

8　在训练集和测试集上测试并得到 Loss 函数的函数值，重复 8 次，输出的 loss_train 值和 loss_val 值如下图中所示



IV. 结论

本次实验，让我对于线性回归的闭式解以及梯度下降有了更深一步的了解；
在实验中，确实感受到闭式解时最为准确的方法，但是基于实际应用中的求解，梯度下降更加方便，通过迭代的过程去逐步求精。当然，这个过程中的参数设置非常重要：在梯度下降中，超参数的选择十分重要，过大，会使折线剧烈波动，无法收敛；过小，折线下滑，但下滑的幅度太小，很难收敛至最佳；
除了实验要求之外，还另外进行了以下准确率的比较，事实表明，两种结果的基本一致（代码在压缩包中另外的一个文件内）