机器学习 002

杨起

1610299

1 问题描述

基于housing数据集构造线性回归分类器， 对训练，测试有如下要求

训练：

- 尝试不同的学习率和迭代次数
- 采用梯度下降或随机梯度下降
- 加入L2正则化系数
- 比较梯度下降和随机梯度下降收敛速度和最终结果

测试：

- 采用留一法划分训练集和测试集，并输出测试集RMSE

2 解决方法

2.1 解决思路

基于`numpy`构建自己的线性回归模型， 尝试不同的学习率和迭代次数

2.2 基本理论

2.2.1 线性回归

我们拥有一组变量X = [x0, x1, ...,]，和一个结果Y，（在本题中是房价）， 我们希望找出影响房价的变量，从而使用他们预测房价Y，我们假设了一个模型，描述Y和所有的变量X的一次方是线性相关的。（本质上是最简单的多项式拟合），即，有

$$Y = \alpha_0 x_0 + \alpha_1 x_1 + \ldots + \beta \tag{1}$$

写为矩阵乘法形式， 有

$$Y = \alpha X + \beta \tag{2}$$

我们的目标就是找到一组合适的系数， 使得（2）式右边逼近左边

2.2.2 均方根误差（RMSE）

怎么判断右边逼近左边的程度呢？ 我们通常使用均方根误差作为度量， 有

$$RMSE = \sqrt{\sum_{1}^{n}(y - (\alpha X + \beta))^2} \tag{3}$$

2.2.3 梯度下降和随机梯度下降

梯度下降的原理： 导数代表了一个函数在某个点的变化率， 向着导数的反方向移动，能够到达函数值更小的点。当导数为零时候来到极值点。

由于RMSE根号不适合求导，我们往往使用MSE来求导

$$MSE = \sum_{1}^{n}(y - (\alpha X + \beta))^2 \tag{4}$$

有

$$\frac{\partial MSE}{\partial \alpha_j} = -\frac{2}{n}\sum_{i=1}^{n}(y - (\alpha X + \beta))x_j \tag{5}$$

以及

$$\frac{\partial MSE}{\partial \beta} = -\frac{2}{n}\sum_{1}^{n}(y - (\alpha X + \beta)) \tag{6}$$

梯度下降的具体实现，就是对于训练集里的每一组数据，对每一个参数，计算梯度，向着MSE减小的方向变化，

学习率体现变化的幅度

2.2.3.1学习率

$$\alpha_i = \alpha_i - learnrate \times \frac{\partial MSE}{\partial \alpha_i} \tag{7}$$

具体的代码如下

```
## Gradient descent
def gradient_descent(x, y, alpha, beta, learn_rate):
    # gradient_arr是整个 alpha 偏导数数组
    gradient_arr = np.zeros((1, x.shape[1]))
    gradient_beta = 0
    mean_s_err = 0
    for line in range(x.shape[0]):
        xline = x[line, :]
        yline = y[line]
        # err = y - (alpha x + beta)
        err = yline - predict(alpha, beta, xline)
        mean_s_err += err ** 2
        # 公式5 求和部分
        gradient_arr += err * xline
        gradient_beta += err

    # arr 是 alpha vector的梯度vec， 意思是 alpha0 是 arr[0]
    gradient_arr = gradient_arr * 2 / x.shape[0]
    gradient_beta = gradient_beta * 2 / x.shape[0]
    mean_s_err = mean_s_err / x.shape[0]

    # 学习率 公式 7
    alpha += np.reshape(gradient_arr, alpha.shape) * learn_rate
    beta += gradient_beta * learn_rate

    return alpha, beta, mean_s_err
```

随机梯度下降的原理和梯度下降的原理完全一致， 区别是它是一种数据比较大时候的工程化的处理方法， 每次梯度下降的时候只使用一组数据求梯度， 这样可以减少计算量。但是同时也会影响收敛速度。

2.2.4 正则化系数

过拟合现象，指在训练集上设定了太大的参数空间， 导致学习到了一些不必要的噪声，对训练数据拟合的非常好，但是预测能力很差，泛化能力很差。

为了防止过拟合， 方式有两种，

- 减少参数， 这个会在后文中探究
- 添加正则化系数

正则化系数本质上是对过大的参数空间的一种惩罚， 具体说来，以L2范数为例

$$MSE = \sum_{i=1}^{n}(y_i - (\alpha X_i + \beta))^2 + \lambda \sum_{j=1}^{m} \alpha_j^2 \tag{8}$$

m是α的数量， λ称为正则化系数， λ越大， α就被限制得越接近0， 也就是模型的拟合能力越弱， 泛化能力越强。选取合适的λ能够帮助我们在过拟合和欠拟合之间找到平衡

加入正则化系数之后，α的偏导数有所变化，有

$$\frac{\partial MSE}{\partial \alpha_j} = -\frac{2}{n}\sum_{i=1}^{n}(y-(\alpha X+\beta))x_j + 2\lambda\alpha_j \tag{9}$$

为此，微调代码

```
## Gradient descent
def gradient_descent_regularized(x, y, alpha, beta, learn_rate, _lambda):
    # gradient_arr是整个 alpha 偏导数数组
    gradient_arr = np.zeros((1, x.shape[1]))
    gradient_beta = 0
    mean_s_err = 0
    for line in range(x.shape[0]):
        xline = x[line, :]
        yline = y[line]
        # err = y - (alpha X + beta)
        err = yline - predict(alpha, beta, xline)
        mean_s_err += err ** 2
        # 公式5 求和部分
        gradient_arr += err * xline
        gradient_beta += err

    # arr 是 alpha vector的梯度vec, 意思是 alpha0 是 arr[0]
    gradient_arr = gradient_arr * 2 / x.shape[0]

    # 加入正则化系数
    gradient_arr = gradient_arr - 2 * _lambda * alpha

    gradient_beta = gradient_beta * 2 / x.shape[0]
    mean_s_err = mean_s_err / x.shape[0]

    # 学习率 公式 7
    alpha += np.reshape(gradient_arr, alpha.shape) * learn_rate
    beta += gradient_beta * learn_rate

    return alpha, beta, mean_s_err
```

2.3 算法流程

1. 读入数据
2. 归一化
3. 使用留一法区分训练集和测试集
4. 训练模型

3 实验分析

3.1 实验数据

本次实验采用实验数据 `housing.data`， 有 513 组数据， 每组数据分为 13个变量X和一个房价Y。

数据量不算很大。对于线性回归模型来说， 参数空间太大了， 可能引起过拟合

3.2 实验设计

1. 尝试在梯度下降和随机梯度下降下调整学习率和迭代次数， 具体参数如下

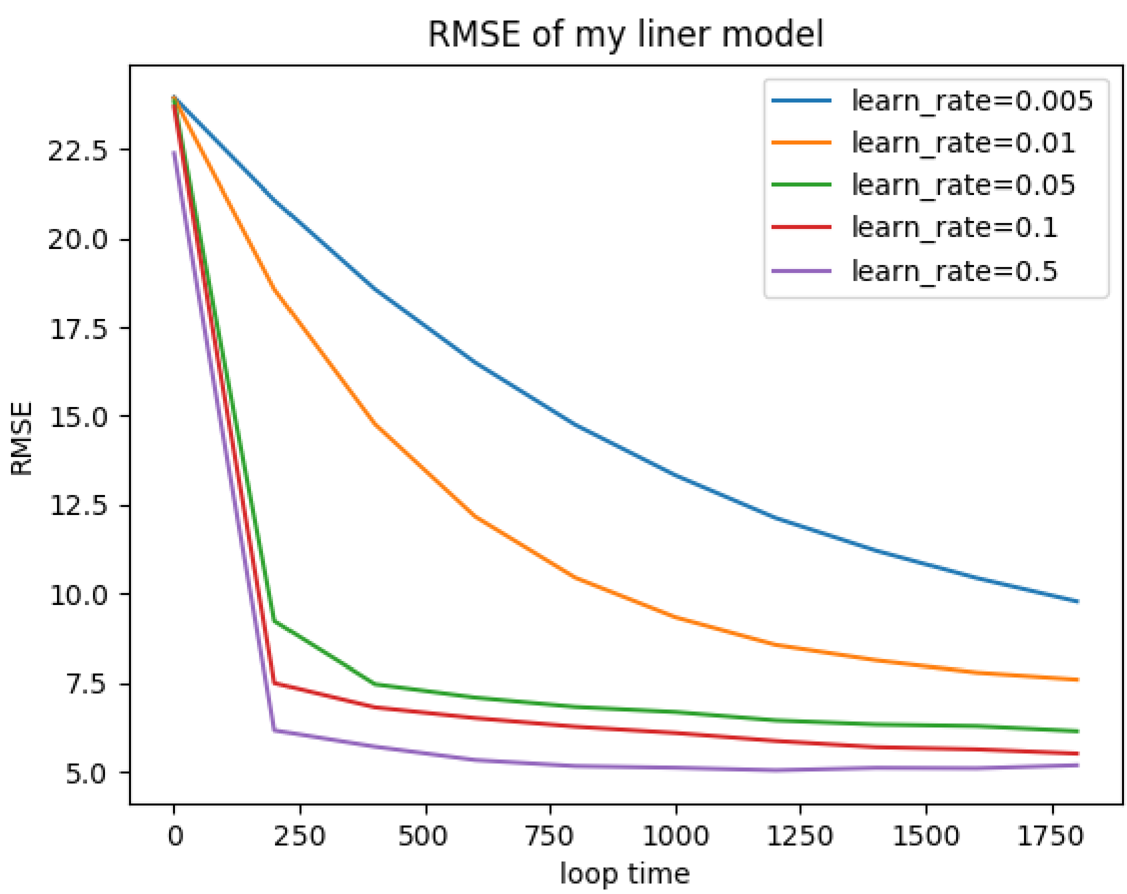
	学习率	迭代次数
梯度下降	[0.005, 0.01, 0,05, 0.1, 0.5]	[1-50]
随机梯度下降	[0.005, 0.01, 0,05, 0.1, 0.5]	[1-500]

2. 尝试加入L2正则化系数， 在同样条件下重复1的安排，比较加入正则化系数后结果
3. 综合比较梯度下降和随机梯度下降的优劣

3.3 实验结果和分析

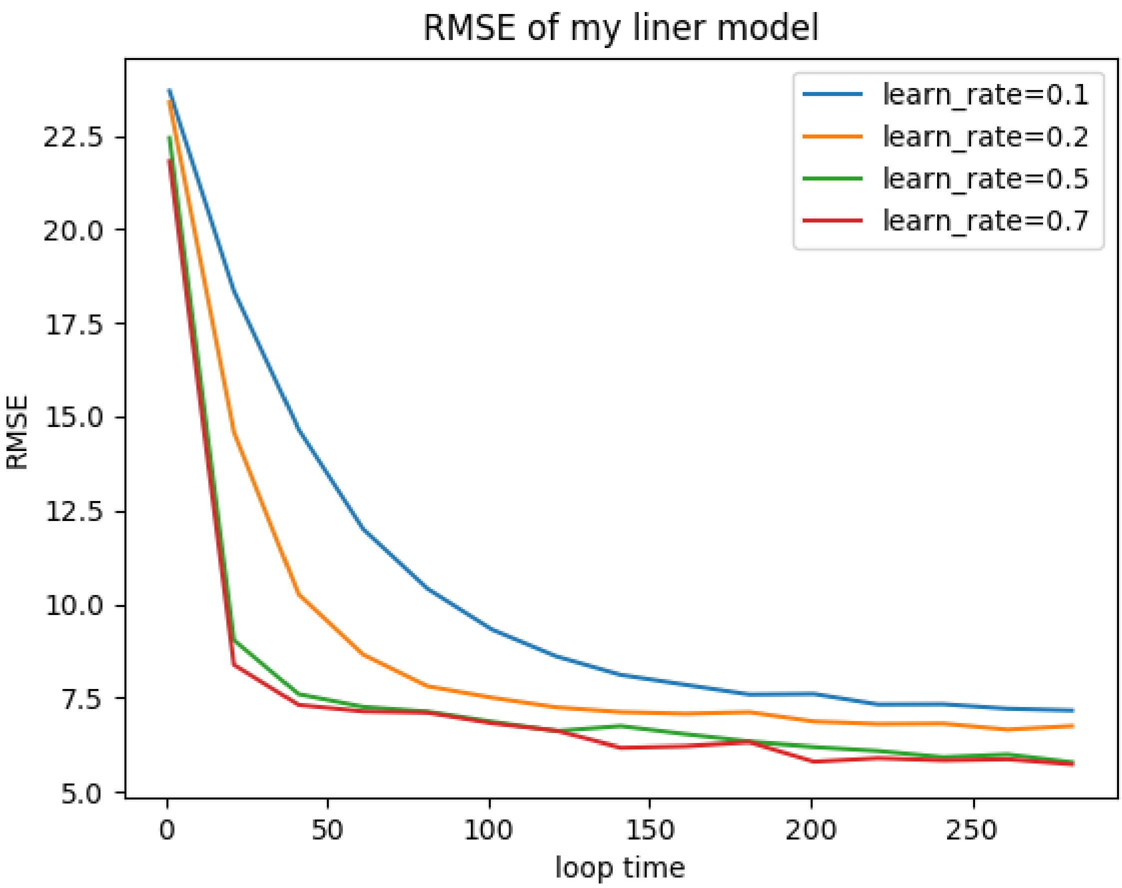
3.3.1 选择学习率

1. 使用随机梯度下降结果图片和分析（确定学习率选择）



可以看见， 大学习率情况下，RMSE 开始有较快下降。收敛速度明显快， 进一步测试大学习率， 减少loop数

	学习率	迭代次数
随机梯度下降	[0.1, 0.2, 0.5, 0.7]	[1-250]

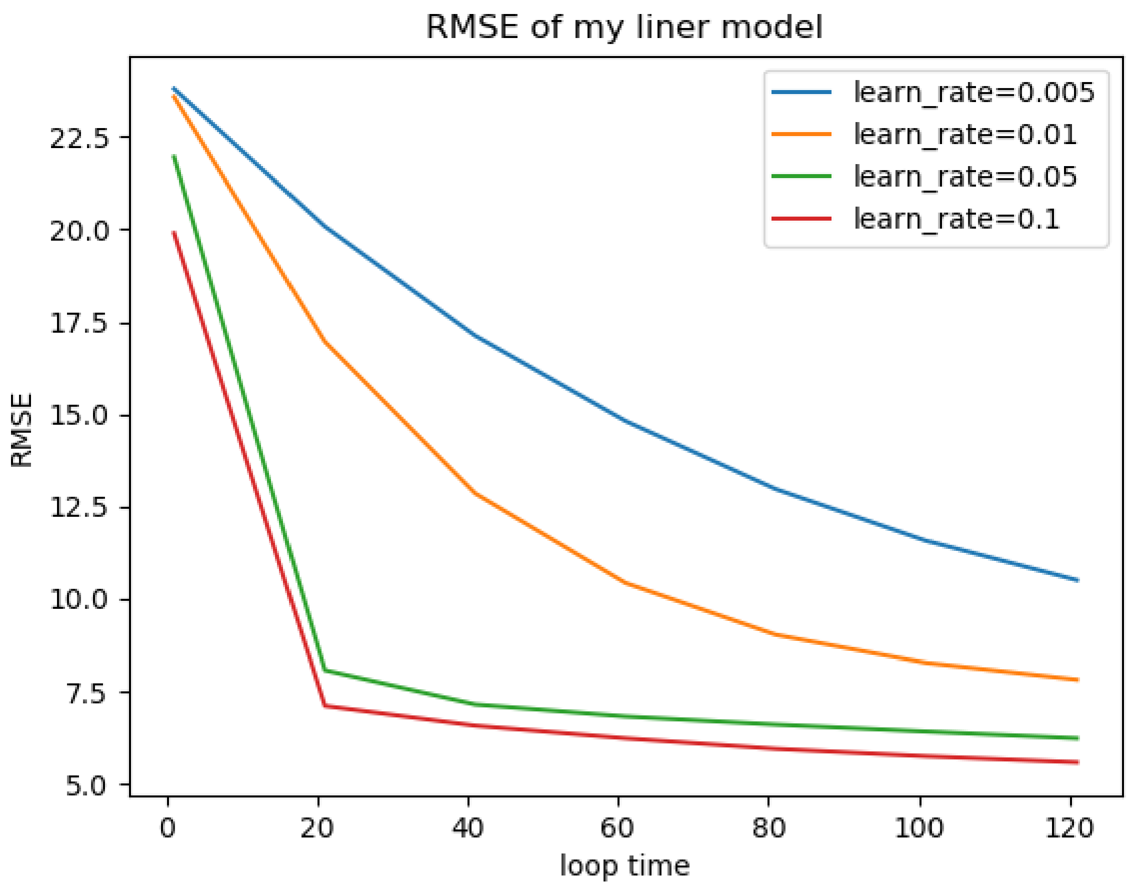


可以看见， **学习率取0.5的情况下， 完全可以在loop=100的时候得到比较满意的结果。**

大学习率优点是可以**加快收敛速度**，缺点是可能出现**精度低**的问题。 具体说来， 是长时间在极值点附近震荡，不能像小学习率机制底下以较高的精度到达极值点。但是从实验结果看来， 低学习率大loop的情况下也没有出现精度明显高的情况， 所以直接采取大学习率是可以接受的。（当然这种情况需要综合考虑问题能接受的精度大小）。

2. 使用梯度下降结果图片和分析（确定学习率选择）

   梯度下降



基本能得到相同的结果，区别只是loop小了一点

3.3.2 比较梯度下降和随机梯度下降：

1. 达到相近的RMSE， **随机梯度下降需要的计算量相对小。** 查看两张图片的`learn_rate`=0.05图片， RMSE=7时， 随机梯度下降需要loop=100， 梯度下降需要loop=40， 但是，考虑到梯度下降每次下降需要综合计算整个测试集，而随机梯度下降之需要计算一条记录， 200 < 40 * 500.
2. 随机梯度下降收敛速度具有不确定性。当然，每次选出来用于计算导数的记录是随机的。 但是这个问题在大的loop下一定程度上被抵消。只要loop的次数足够多，综合而言所有记录都可能被选到。

综合而言， 随机梯度下降在大数据集上的确是节省计算量的好方法。

3.3.3 加入正则化系数

加入正则化系数， 采取梯度下降， 得到情况如下

- 收敛速度变慢
- 收敛到RMSE相近

和我们使用正则化增加泛化能力的目的比较相符。

4 自主尝试： 减少参数数量

我们在正则化参数一节谈到过， 主要的问题是，在一个线性回归问题中， 十三个变量可能太多了。 参数空间太大，容易过拟合。 为此我们可以引入正则化， 也可以通过分析， 尝试人工筛选减少参数。

4.1 保留两个参数

最简单的想法是，把线性回归式子简化为

$$Y = \alpha_i x_i + \beta \tag{10}$$

即只使用一个最相关的变量， 这样我们就能把Y随x变化的图片画在二维平面上。

我们通过计算每一个变量x和Y的**皮尔逊相关系数**来选择x， 只要选择|r| 最大的x即可。

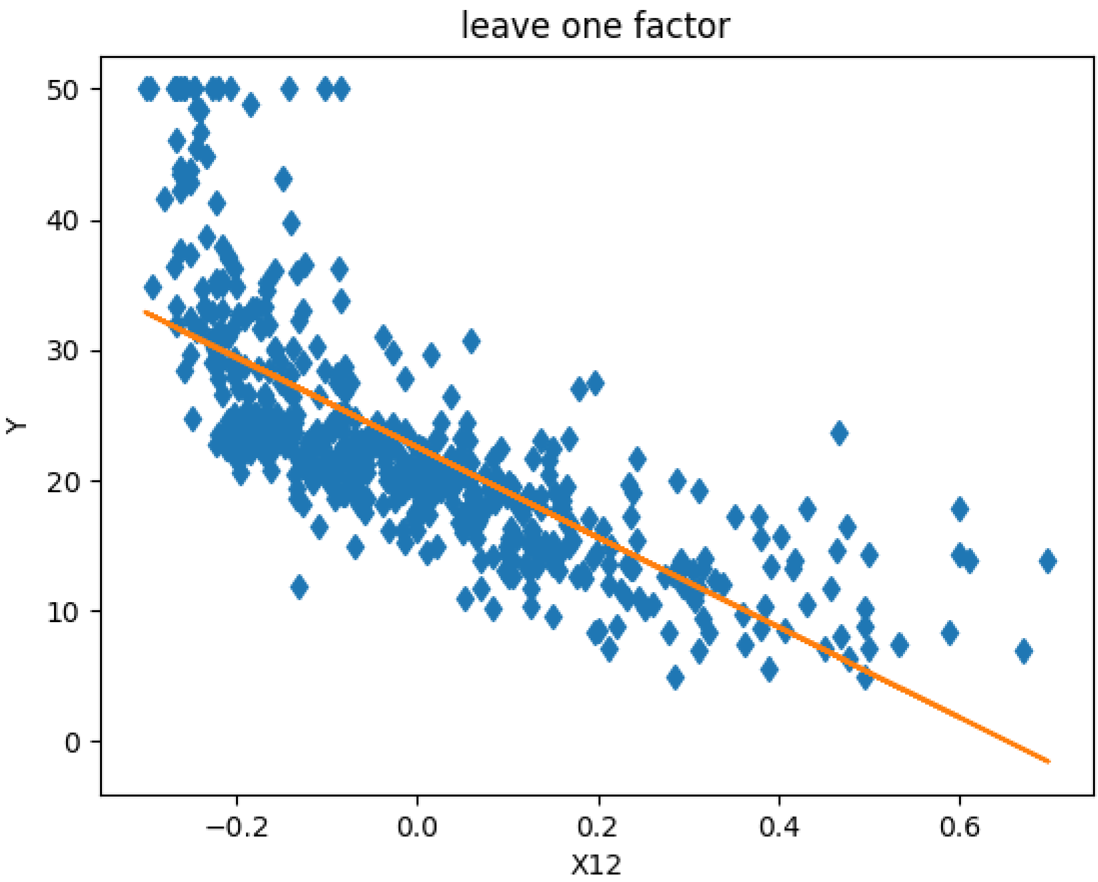
$$r = \frac{E[(X - \mu x)(Y - \mu y)]}{\sigma(x)\sigma(y)} \tag{11}$$

有代码如下:

```
def cal_r(x, y):
    x1 = x - np.mean(x)
    y1 = y - np.mean(y)
    nume = np.mean(x1 * y1)
    return nume / (np.std(x, ddof=1) * np.std(y, ddof=1))
```

通过这个方法， 选择最相关的X12作为唯一维度， （相关性系数为0.7左右）

最后变化如下



4.2 保留多个参数

公式2中， α本身就是度量变量x对Y的变化起多大决定作用的度量。 我们可以尝试直接丢弃一部分α小的x， 起到减小参数空间的作用。