# 机器学习课程报告

# Regression for housing price prediction

## 1.任务描述

**任务 1**：采用回归算法，预测波士顿地区房价
**任务 2**：回归算法应包含岭回归和 Lasso 回归。

## 2.数据集描述

**描述 1**：本次数据集为：波士顿房价数据集。
**描述 2**：波士顿房价数据集特征：实例数：506。属性数：13 个。

- CRIM 按城镇划分的人均犯罪率
- ZN 超过 25,000 平方英尺的住宅用地比例。
- INDUS 每个城镇的非零售商业用地比例
- CHAS Charles River 虚拟变量
- NOX 一氧化氮浓度
- RM 每户平均房间数
- AGE 1940 年之前建造的自有单位的比例
- DIS 到五个波士顿就业中心的加权距离
- RAD 径向高速公路的可达性指数
- TAX 每 10,000 美元的全额财产税税率
- PTRATIO 按城镇划分的师生比例
- B 1000(Bk - 0.63)^2 其中 Bk 是按城镇划分的黑人比例
- LSTAT 人口地位较低的百分比
- MEDV 自住房屋的中位数价值 1000 美元

## 3.方法介绍

### 3.1 数据预处理

**预处理 1**：检查空缺

```
df.isnull().sum()
```

数据在大多数情况下都有很多缺失数据，每个值缺失的原因可能不同，但缺失的数据会降低模型的预测能力。检查是否有数据空缺，若有少量空缺采用平均值。
**预处理 2**：数据标准化

```
ss = StandardScaler()
x = ss.fit_transform(x)
```

波士顿房价数据集的特征的量纲和数值得量级都是不一样的，在预测房价时，如果直接使用

原始的数据值，那么它们对房价的影响程度将是不一样的，而通过标准化处理，可以使得不同的特征具有相同的尺度。

**预处理 3**：数据切分

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

对数据进行分组处理，一部分用于训练，一部分用于测试。分组比例默认为训练:测试=0.7:0.3

## 3.2 算法描述

**描述 1**：梯度下降算法

梯度下降法的计算过程就是沿梯度下降的方向求解极小值。

当 Y 值的影响因素不是唯一时，采用多元线性回归模型：$h_\theta(x)=\theta^T x=\theta_0 x_0+\theta_1 x_1+\theta_2 x_2+\cdots+\theta_n x_n$

重复：

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$$

从而得到最终参数。

这里完成了简单的两特征梯度下降算法（即$h_\theta(x)=\theta_0+\theta_1 x_1+\theta_2 x_2$

一次循环中过程如下：

$$\theta_0 := \theta_0 - \alpha\tfrac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha\tfrac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha\tfrac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_2^{(i)}$$

)

：

```
# 学习率 learning rate
lr = 0.0001
# 参数
theta0 = 0
theta1 = 0
theta2 = 0
# 最大迭代次数
epochs = 1000

def gradient_descent_runner(x_data, y_data, theta0, theta1, theta2, lr, epochs):
    # 计算总数据量
    m = float(len(x_data))
    # 循环 epochs 次
```

```python
    for i in range(epochs):
        theta0_grad = 0
        theta1_grad = 0
        theta2_grad = 0
        # 计算梯度的总和再求平均
        for j in range(0, len(x_data)):
            theta0_grad += (1/m) * ((theta1 * x_data[j,0] + theta2*x_data[j,1] + theta0) - y_data[j])
            theta1_grad += (1/m) * x_data[j,0] * ((theta1 * x_data[j,0] + theta2*x_data[j,1] + theta0) - y_data[j])
            theta2_grad += (1/m) * x_data[j,1] * ((theta1 * x_data[j,0] + theta2*x_data[j,1] + theta0) - y_data[j])
        # 更新 b 和 k
        theta0 = theta0 - (lr*theta0_grad)
        theta1 = theta1 - (lr*theta1_grad)
        theta2 = theta2 - (lr*theta2_grad)
    return theta0, theta1, theta2
```

梯度下降算法的代价函数为：$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

```python
# 最小二乘法
def compute_error(theta0, theta1, theta2, x_data, y_data):
    totalError = 0
    for i in range(0, len(x_data)):
        totalError += (y_data[i] - (theta1 * x_data[i,0] + theta2*x_data[i,1] + theta0)) ** 2#真实值-预测值 的平方
    return totalError / float(len(x_data))/2.0
```

注意：上面只是两个特征的简单实现，对于本数据集中的多特征完成过于繁琐，下面处理中我们调用 sklearn.linear_model.LinearRegression。

sklearn.linear_model.LinearRegression 求解线性回归方程参数时，首先判断训练集 X 是否是稀疏矩阵，如果是，就用 Golub&Kanlan 双对角线化过程方法来求解；否则调用 C 库中 LAPACK 中的用基于分治法的奇异值分解来求解。

**描述 2**：岭回归

岭回归通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价获得回归系数更为符合实际、更可靠的回归方法。

岭回归代价函数：$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

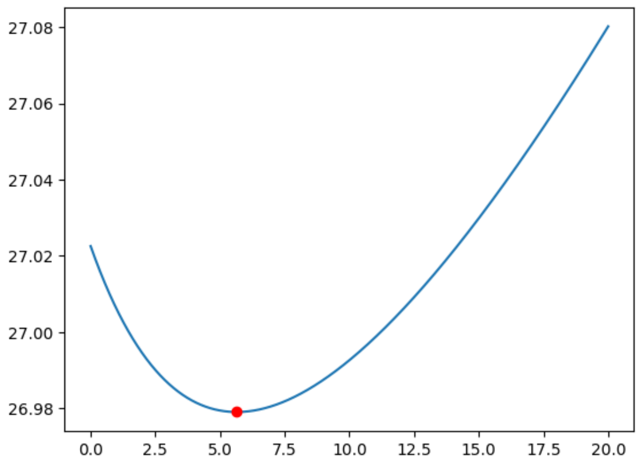
岭回归求解：ws = $(X^TX + \lambda I)^{-1}X^T$y

其中，ws 为系数矩阵，$\lambda$为岭系数

```python
# 岭回归标准方程法求解回归参数
def weights(xArr, yArr, lam=0.2):#lam 岭系数。自己暂时选定默认值为 0.2
    xMat = np.mat(xArr)
    yMat = np.mat(yArr)
    xTx = xMat.T*xMat # 矩阵乘法
    rxTx = xTx + np.eye(xMat.shape[1])*lam#xMat.shape[1]即 xMat 的列
    #np.eye(xMat.shape[1])生成 xMat.shape[1]行 xMat.shape[1]列的单位矩阵
    # np.eye(xMat.shape[1])*lam 就是 Iλ
    # 计算矩阵的值,如果值为 0，说明该矩阵没有逆矩阵
    if np.linalg.det(rxTx) == 0.0:
        print("This matrix cannot do inverse")
        return
    # xTx.I 为 xTx 的逆矩阵
    ws = rxTx.I*xMat.T*yMat#w=(xT*x+λ*I)**-1 *xT*y
    return ws
```

确定岭系数$\lambda$:

```python
#生成 1000 个值
alphas_to_test=np.linspace(0.001,20,10000)
#创建模型，保存误差值
model1=linear_model.RidgeCV(alphas=alphas_to_test,store_cv_values=True)#RidgeCV，Ridge 岭回归，CV 交叉验证。
#alphas 岭回归系数。store_cv_values 保存交叉验证的结果
model1.fit(x_train,y_train)#训练模型
```




选取让 loss 值最小的岭系数

**描述 3**：Lasso 回归

Lasso 回归是一种压缩估计。它通过构造一个惩罚函数得到一个较为精炼的模型，使得它压缩一些回归系数，即强制系数绝对值之和小于某个固定值；同时设定一些回归系数为零。因此保留了子集收缩的优点，是一种处理具有复共线性数据的有偏估计。

LASSO 代价函数: $$J(\theta)=\frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2+\lambda\sum_{j=1}^{n}\left|\theta_j\right|\right]$$

# 4.实验结果分析

## 4.1 评价指标

**指标 1**：系数复杂度

```
print('coefficients:',model.coef_)
```

**指标 2**：训练集准确性

```
model.score(x_train, y_train)
```

**指标 3**：测试集准确性

```
model.score(x_test, y_test)
```

## 4.2 可视化结果

**结果 1**：系数比较

| 系数 | 线性回归 | 岭回归 | Lasso 回归 |
|---|---|---|---|
| | -1.07836886 | -1.027062 | -1.01352027 |
| | 0.92960802 | 0.85542835 | 0.83972567 |
| | -0.13504102 | -0.22970732 | -0.15145667 |
| | 0.71845002 | 0.74032124 | 0.71669154 |
| | -2.21347077 | -2.01357156 | -2.08147133 |
| | 2.5631083 | 2.63412897 | 2.61698315 |
| | 0.02386238 | -0.02288094 | 0 |
| | -3.32090969 | -3.12983567 | -3.14681191 |
| | 2.84197152 | 2.41431383 | 2.46038977 |
| | -1.84090632 | -1.49960546 | -1.53781139 |
| | -2.25088858 | -2.18392073 | -2.20576643 |
| | 0.9423119 | 0.93369164 | 0.9212822 |
| | -3.98900942 | -3.87543458 | -3.96041114 |

可以看到在 Lasso 回归中，出现了系数为 0 的情况。Lasso 回归通过构造一个一阶惩罚函数获得一个精炼的模型；通过最终确定一些指标（变量）的系数为零，解释力很强。
普通线性回归和岭回归估计系数等于 0 的机会微乎其微，造成筛选变量困难

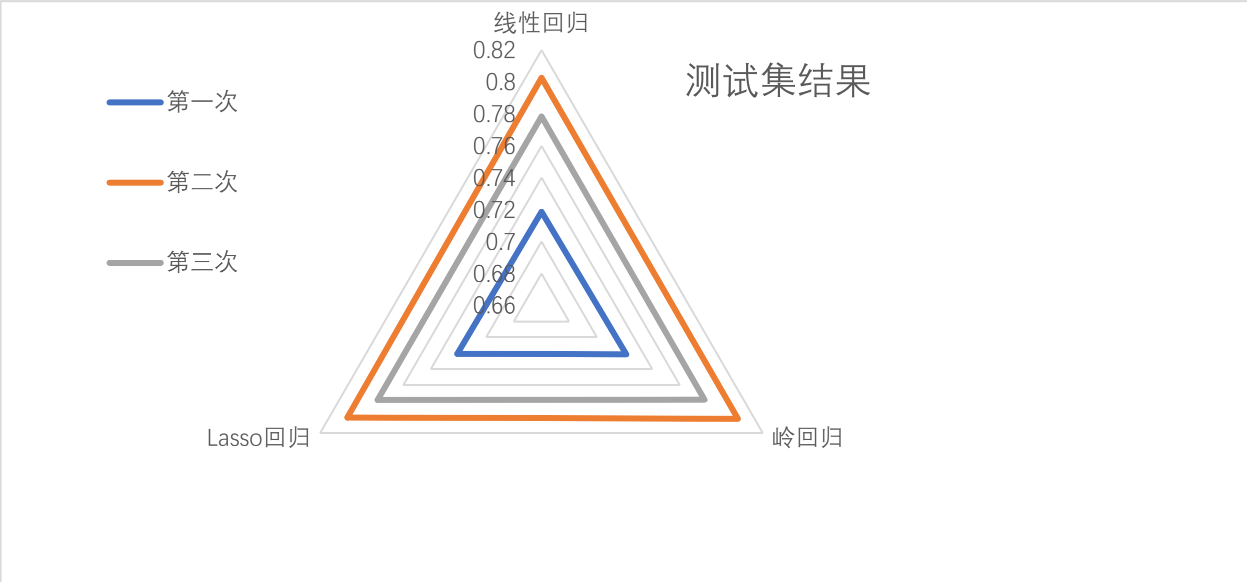
**结果 2**：截距结果

| 截距 | 线性回归 | 岭回归 | Lasso |
|---|---|---|---|
| 1 | 22.45721948 | 22.47162846 | 22.46748074 |

| | | | |
|---|---|---|---|
| 2 | 22.57106319 | 22.57186303 | 22.56913562 |
| 3 | 22.57844348 | 22.57239893 | 22.57300517 |
| 平均值 | 22.53557538 | 22.53863014 | 22.53654051 |

**结果 3**：测试集结果

| 测试集准确率 | 线性回归 | 岭回归 | LASSO 回归 |
|---|---|---|---|
| 1 | 0.718718327 | 0.721323065 | 0.72082316 |
| 2 | 0.802660438 | 0.80200947 | 0.8002924 |
| 3 | 0.778421718 | 0.777801959 | 0.778434596 |
| 平均值 | 0.766600161 | 0.767044831 | 0.766516719 |




可发现三种算法对波士顿房价预测准确率结果相近，未出现较大差异。

**结果 4**：训练集结果

| 训练集准确率 | 线性回归 | 岭回归 | Lasso 回归 |
|---|---|---|---|
| 1 | 0.747355074 | 0.746862056 | 0.746995897 |
| 2 | 0.703702235 | 0.702931747 | 0.703166678 |
| 3 | 0.721520465 | 0.720895337 | 0.721304396 |
| 平均值 | 0.724192591 | 0.723563047 | 0.723822323 |

未出现过拟合现象

## 5.总结

**总结 1**：梯度下降算法

优点：当特征值非常多的时候也，可以很好的工作

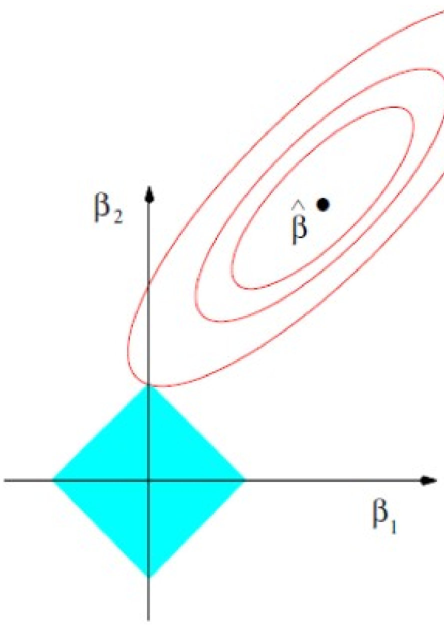
缺点：需要选择合适的学习率，需要迭代很多个周期，只能得到最优解的近似值。靠近极小值时收敛速度减慢。直线搜索时可能会产生一些问题。可能会“之字形”地下降。

**总结 2**：岭回归

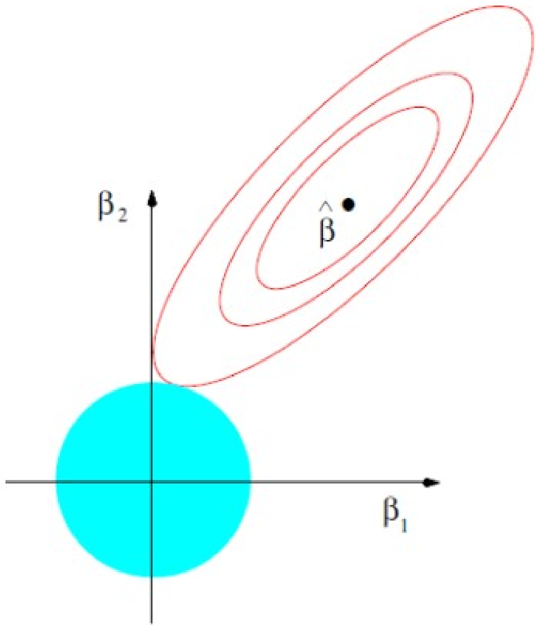
优点：岭回归不需要学习率，不需要迭代，可以得到全局最优解。岭回归方程回归系数的显著性往往明显高于普通回归，在存在共线性问题和病态数据偏多的研究中有较大的实用价值。有较高的数值稳定性，从而得到较高的计算精度。

缺点：岭回归方程的 R 平方值会稍低于普通回归分析，对于影响很小的因子的值不能趋近到

0。



Lasso



岭回归

**总结 3**：Lasso

优点：可以将影响很小的因子的值减到 0，更加便于筛选

缺点：没有真实的解，只能逼近和估计解

**总结 4**：总结

（1）不是所有多元线性回归都需要复杂的回归方式，如果变量较少，且相互独立没有变量间影响，就不要做这些操作

（2）岭回归和 Lasso 操作核心就是为了消除多变量带来的多重共线性问题

（3）且不要忘记数据的预处理和变量的标准化