三种线性回归算法在模拟数据和 Boston 房价数据上的实验探讨

实验目的：

1. 了解三种线性回归算法：ols, ridge regression, lars 的原理
2. 掌握三种算法的特点。利用模拟数据对比分析三种算法在线性回归中的优缺点以及不同的作用。
3. 以 Boston 房价数据为例，学习掌握对实际数据建立线性回归模型，并在实际数据中研究三种算法的特点。

实验要求：

一．三种算法的简单原理介绍

二．模拟数据实验

1. 数据的生成过程：有无共线性性；有无噪声；
2. 对比三种算法的性能；
3. Lars 变量选择的稳定性以及选择前后效果对比。

三．Boston 数据实验

1. Boston 房价数据的探索性分析
2. 数据的预处理（归一化之类，缺失值处理之类）
3. 数据的特征工程（可以加特征，或者特征选择）
4. 三种算法在该数据上的性能分析。

实验内容：

1 三种线性回归算法原理

线性回归是根据预测变量 X 预测定量响应变量 Y 的方法。在数学上，可以将这种数据学关系记为：

$$Y = X\beta + \varepsilon$$

其中， X 为 $n \times p$ 的矩阵， β, Y 为列向量。

在实践中，β 都是未知的。在利用上式做出预测之前，必须根据样本数据集估计未知参数。为了使线性模型较好地拟合现有数据，即找到合适的 $\hat{\beta}$ 使由此产生的回归平面尽可能接近样本数据。

衡量所估计的未知参数 $\hat{\beta}$ 产生的回归平面与样本数据接近程度的方法有很多，对应不同的损失函数有着不同的算法，常见有 OLS，岭回归，lasso，SCAD，梯度下降，偏最小二乘，LARS 等。

1.1 OLS(Ordinary least squares)

最小二乘法采用残差平方和(RSS)最小的原则估计未知参数 β 。即

$$\min_{\beta} \|Y - X\beta\|^2$$

为求解线性回归问题的最优参数，可以利用梯度下降法对上式进行估计，而最小二乘法通过则正规方程法对上式进行求解，即通过对残差平方和 $S(\beta)$ 求导计算其极值。

$$S(\beta) = \|Y - X\beta\|^2$$

令

$$\frac{dS(\beta)}{d\beta}=0$$

解得 $\hat{\beta}^{ols}=\left(X^TX\right)^{-1}X^TY$，$\hat{\beta}^{ols}$ 即为最小二乘法得到的估计。

最小二乘法具有原理清晰，解为全局最优解等优点。相较于梯度下降法，最小二乘法不需要选择学习率 α，也不需要进行多次迭代。此外，最小二乘所得估计具有无偏性、有效性等优良性质。

但最小二乘法也有一定的不足之处，X^TX 在特征之间具有多重共线性关系以及样本数 n <特征数 p 时，该矩阵不可逆。此外，在最小二乘法求解计算量约为 $o(p^3)$，这使得在特征数 p 较大时，模型的计算量偏大。

1.2 Ridge regression

为了解决最小二乘法中 X^TX 不可逆的问题以及改善过拟合问题，常利用正则化方法对损失函数添加惩罚项。

岭回归是一种改良的最小二乘法，岭回归牺牲了无偏性这一性质，通过惩罚系数 L2 范数，使系数尽可能地小以免过拟合，同时仍然给出很好地预测结果，使得回归系数更加符合实际。

岭回归采用残差平方和加正则项最小的原则估计未知参数 β，具体公式如下：

$$\min_{\beta}\|Y-X\beta\|^2+\lambda\|\beta\|^2$$

其中，λ 为正则化参数。

和最小二乘法的思路一样，同样对其关于 β 求导，最后得到 $\hat{\beta}^{ridge}$ 的矩阵表达式为：

$$\hat{\beta}^{ridge}=\left(X^TX+\lambda I\right)^{-1}X^TY$$

由岭回归得到的估计，在原先的最小二乘估计中加一个小扰动 λI，使原先无法求逆的情况变成可以求解，使得问题稳定并得以求解。

1.3 LARS(Least angle regression)

最小角回归算法是适用于高维数据的一种算法，特别适合那些样本维数远大于样本数量的情况。除此之外，它还具有很多的优点，如其计算速度几乎和前向选择算法一样；可以产生分段线性结果的完整路径，这在模型的交叉验证中极为有用。

其具体算法如下：从所有系数都为零开始，即 X 标准化、Y 中心化，首先找到和相应 y 最相关（余弦距离最大）的预测变量 x_k，类似于前向梯度算法中的残差计算方法，得到新目标 Y_{yes}，然后直接向前走直到出现一个 x_i，使得 x_i 和 Y_{yes} 的相关度与 x_k 和 Y_{yes} 的相关度是一样的，此时残差 Y_{yes} 在 x_k 和 x_i 的角平分线上，然后开始沿着这个残差角平分线走，直到出现第三个特征 x_p 和 Y_{yes} 的相关度足够大的时候，即 x_p 和 Y_{yes} 的相关度与 x_k，x_i 和 Y_{yes} 的一样，将其也归入 Y 的逼近特征集合中，并用 Y 的逼近特征集合的共同分线作为新的逼近方向，以此循环，直到

Y_{yes} 足够小或所有变量都已经取完了，算法停止。此时对应的系数 β 即为最终结果。

2 模拟数据分析

2.1 模拟数据构造

考虑线性模型

$$y = X^T\beta + \sigma\varepsilon$$

在模拟实验中，取 n=200，ε~$N(0,1)$，进行随机模拟，从而产生数据 x 和数据 y。取

$$\begin{cases} x_1 \sim N(10,2) \\ x_2 \sim N(5,3) \\ x_3 \sim U(1,10) \\ x_4 \sim U(10,20) \\ x_5 = x_1 + 2x_2 + 0.6x_4 \end{cases}$$

部分模拟数据数值如下表所示：

表 1 具体模拟数据

X_1	X_2	X_3	X_4	X_5
14.2489	8.2248	7.176042	17.74994	41.34846
10.50529	5.602509	6.982503	11.44458	28.57706
12.90836	4.365116	6.649343	13.56534	29.7778
11.13848	2.090914	8.682412	10.36055	21.53664
10.91645	4.555849	2.488577	14.42383	28.68244

根据 5 个特征，分别构造如下有无共线性以及有无噪声的四个因变量：

$$y_1 = 10+3x_1 + 1.5x_2 + 0.5x_3 - 2x_4 + x_5$$

$$y_2 = 10+3x_1 + 1.5x_2 + 0.5x_3 - 2x_4 + x_5 + 2\varepsilon$$

$$y_3 = 10+3x_1 + 1.5x_2 + 0.5x_3 - 2x_4$$

$$y_4 = 10+3x_1 + 1.5x_2 + 0.5x_3 - 2x_4 + 2\varepsilon$$

表 2 所有特征的描述性统计

	X_1	X_2	X_3	X_4	X_5	Y_1	Y_2	Y_3	Y_4
count	200	200	200	200	200	200	200	200	200
mean	10.11	4.9	5.55	15.43	29.18	48.77	48.61	19.59	19.84
std	2.01	3.04	2.67	2.73	6.56	13.5	13.59	9.1	9.1
min	3.53	-5.18	1.1	10.02	9.03	4.76	8.62	-7.39	-5.83
25%	8.84	3.16	3.23	13.17	25.89	40.34	40.43	12.72	12.53
50%	10.27	4.98	5.43	15.4	29.12	48.22	48.15	19.99	20.04
75%	11.41	6.92	7.94	17.83	33.07	57.65	57.66	25.75	26.09
max	16.27	14.59	9.98	19.99	48.57	84.54	85.03	44.51	46.07

由随机数的原始构造可知，变量 x_5 和 x_1, x_2, x_4 有一定的线性关系。观察下图

变量之间的散点图和相关系数热力图可以看出，x_5 与 x_2 之间的相关性较强。

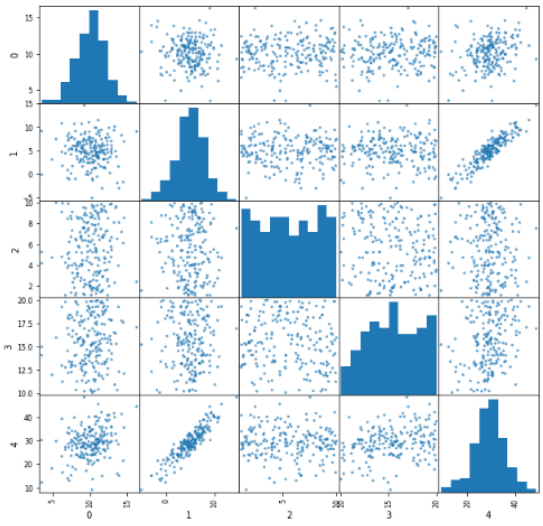

图 1 变量之间的散点图

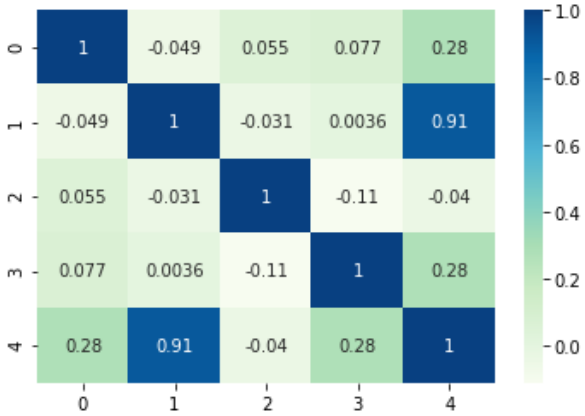

图 2 变量相关系数热力图

2.2 三种算法的性能对比

随机抽取数据集的 75%数据作为训练集，对模型进行训练。将其余的数据作为测试集，利用 R^2、MSE 均方误差、解释方差得分、MAE 平均绝对误差和绝对中位差等指标评估模型的效果。以下是不同数据集在不同的算法下的具体表现：

表 3 三种算法在不同数据集上的表现

R^2		OLS	Ridge	Lars
无共线性，无噪声	训练集	**1**	**1**	**1**
	测试集	**1**	**1**	**1**
无共线性，有噪声	训练集	**0.954180093**	**0.954180093**	0.95418009
	测试集	**0.946079566**	**0.946079566**	0.94607957
有共线性，无噪声	训练集	**1**	**1**	0.96861864
	测试集	**1**	**1**	0.96490446
有共线性，有噪声	训练集	**0.981731362**	0.981729507	0.96610231
	测试集	**0.971110714**	0.97099233	0.94169071

表 4 三种算法在其他评价参数中的表现

		OLS	Ridge	Lars
无共线性，无噪声	MSE 均方误差	2.85E-28	3.64513E-13	**1.78724E-28**
	解释方差得分	**1**	**1**	**1**
	MAE 平均绝对误差	1.52056E-14	4.76292E-07	**1.25056E-14**
	绝对中位差	1.42109E-14	3.69603E-07	**1.06581E-14**
无共线性，有噪声	MSE 均方误差	3.715437126	**3.715437077**	3.715437126
	解释方差得分	0.946719478	**0.946719478**	0.946719478
	MAE 平均绝对误差	1.454631222	**1.45463117**	1.454631222
	绝对中位差	1.34653285	**1.346533217**	1.34653285
有共线性，无噪声	MSE 均方误差	**4.23082E-28**	4.26734E-11	6.131068528
	解释方差得分	**1**	**1**	0.966166322
	MAE 平均绝对误差	**1.63425E-14**	5.02293E-06	1.974385629
	绝对中位差	**1.42109E-14**	4.12341E-06	1.617279339
有共线性，有噪声	MSE 均方误差	**5.530137438**	5.552799148	11.16186773
	解释方差得分	**0.97254124**	0.972429666	0.945243583
	MAE 平均绝对误差	**1.907700436**	1.911977373	2.613200276
	绝对中位差	1.821587004	**1.820300099**	2.428830926

由上面两个表格可知，对于无共线性、无噪声的数据三种算法在精确度上的表现都很好，相对而言 Lars 算法在 MSE 等指标上的表现更优。

对于有共线性、无噪声的数据，Lars 算法则远差于 OLS 和 Ridge Regression。在 OLS 和 Ridge Regression 中，两者在精确度上并无明显差异，在 MSE 等指标上 OLS 算法略胜一筹。

在有噪声的数据集中，各算法表现与在无噪声的数据集中的表现类似，但准确性有所降低。

总体而言，OLS 算法和 Ridge Regression 在有无共线性和有无噪声的数据集中的表现都和优异，且并无明显差异。而 LARS 算法在共线性数据中的表现与 OLS 算法和 Ridge Regression 的表现相比则不尽人意。

2.3 Lars 变量选择的稳定性以及选择前后效果对比

通过分层十折交叉验证，利用打乱样本选取不同的训练集对模型进行评价，综合考虑与其他两种算法相比，Lars 算法的稳定性。

考虑到三种算法在无共线性，无噪声的数据集中能够准确计算出模拟数据之间的关系。因此，排除该数据集对 Lars 算法稳定性的探索。

以下是三种算法在无共线性、有噪声，有共线性、无噪声以及有共线性、有噪声的数据集中，分别利用分层十折交叉验证得到的模型得分分布密度和模型得分分布柱状图。排除由于数据随机抽样对模型带来的本质影响，将 Lars 算法和 OLS 算法以及 Ridge Regression 算法对比可以发现，在无共线性的数据中，Lars 算法表现和 OLS 算法以及 Ridge Regression 一致。但在存在共线性的数据中，Lars 算法明显不比 OLS 以及 OLS 算法以及 Ridge Regression 稳定。

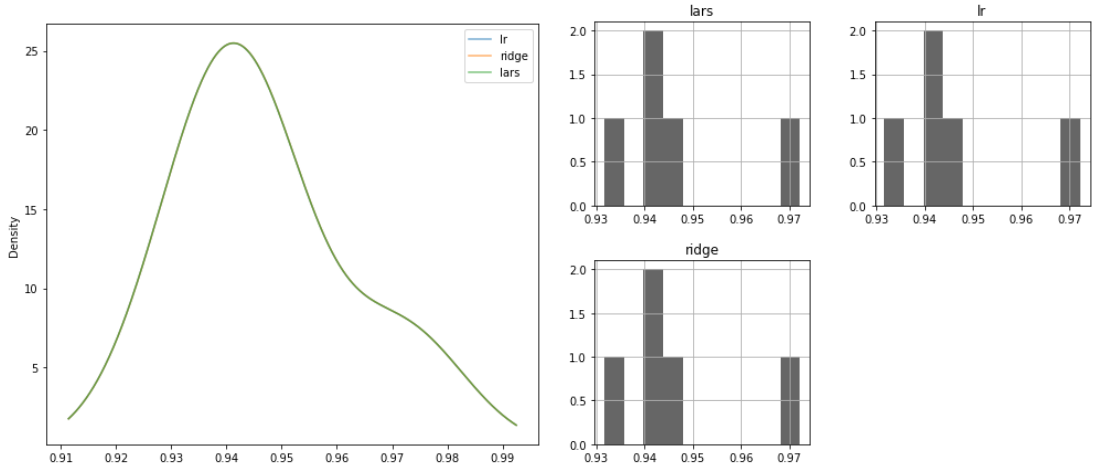

图 3 无共线性，有噪声

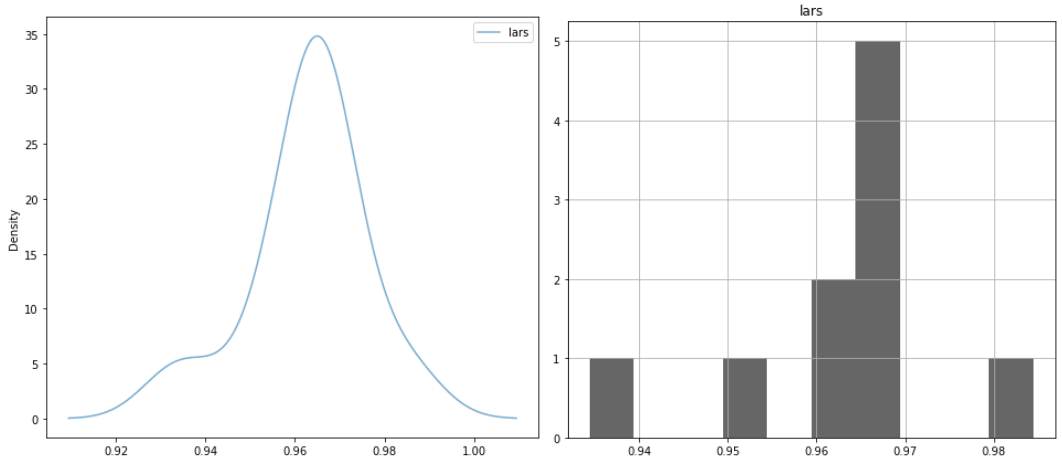

图 4 有共线性，无噪声

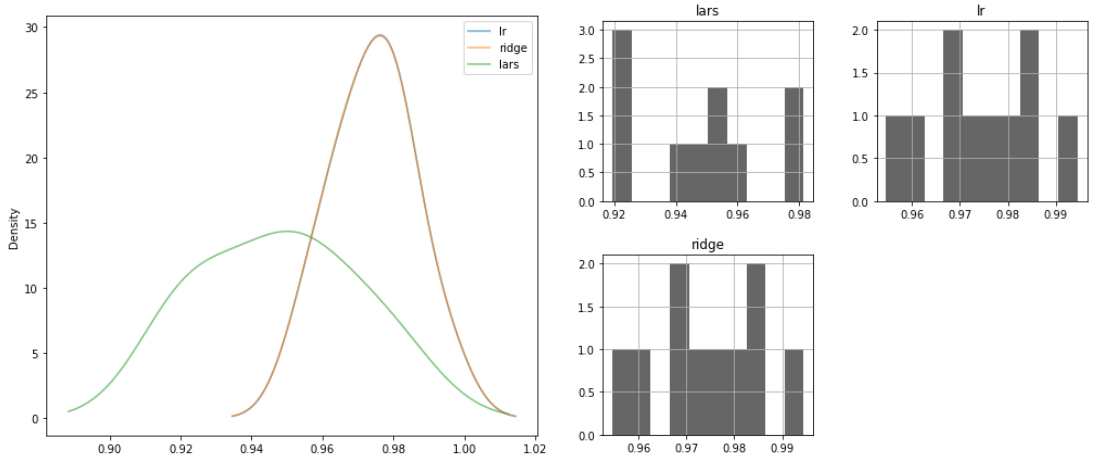

图 5 有共线性，有噪声

3 Boston 数据分析

3.1 Boston 数据介绍

Boston 数据集是一份源于美国某经济学杂志，分析研究波士顿房价的数据集。

表 5 Boston 数据集所有特征的含义

特征	含义	特征	含义
CRIM	城镇人均犯罪率	ZN	超过 25000 平方英尺住宅所占比例
INDUS	非零售业务用地的比例	CHAS	是否临近查尔斯河
NOX	一氧化氮浓度	RM	每个住宅的平均房间数
AGE	1940 年前建造住房的比例	DIS	到五个就业中心的加权距离
RAD	径向公路的可达性指数	TAX	每 10,000 欧元的全值财产税率
PRTATIO	按镇划分的师生比例	B	1000（Bk-0.63）^ 2, Bk 是黑人比例
LSTAT	社会经济地位低的家庭占比	MEDV	街区的房价中位数

数据集共包含 506 个数据点和 2 个特征。根据数据集提供的信息，我们需要利用犯罪率、是否临近查尔斯河、公路可达性等信息，来预测 20 世纪 70 年代波士顿地区房价的中位数。

3.2 探索性分析

表 6 数据集结构

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33

表 7 各变量统计性描述

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
mean	3.61	11.36	11.13	0.06	0.55	6.28	68.57	3.79	9.54	408.23	18.45	356.67	12.65
std	8.60	23.32	6.86	0.25	0.11	0.70	28.14	2.10	8.70	168.53	2.16	91.29	7.14
min	0.01	0	0.46	0	0.385	3.561	2.9	1.13	1	187	12.6	0.32	1.73
25%	0.08	0	5.19	0	0.44	5.88	45.02	2.10	4	279	17.4	375.37	6.95
50%	0.25	0	9.69	0	0.53	6.20	77.5	3.20	5	330	19.05	391.44	11.36
75%	3.67	12.5	18.1	0	0.62	6.62	94.07	5.18	24	666	20.2	396.22	16.95
max	88.97	100	27.74	1	0.87	8.78	100	12.1	24	711	22	396.9	37.97

观察表二和表三，可以发现数据集所有特征均不存在缺失值。再结合数据变量的含义，可以知道变量 CHAS 是分类变量，CHAS 变量的分布情况如下表所示。506 各街区中有 471 个街区依傍查尔斯河流，35 个街区没有依傍查尔斯河流。由此说明该变量有严重的数据不平衡问题，因此在后续抽取训练集和测试集时要特别注意变量 CHAS 的分布情况。

表 8 CHAS 变量分布情况

CHAS	1	0
Counts	471	35

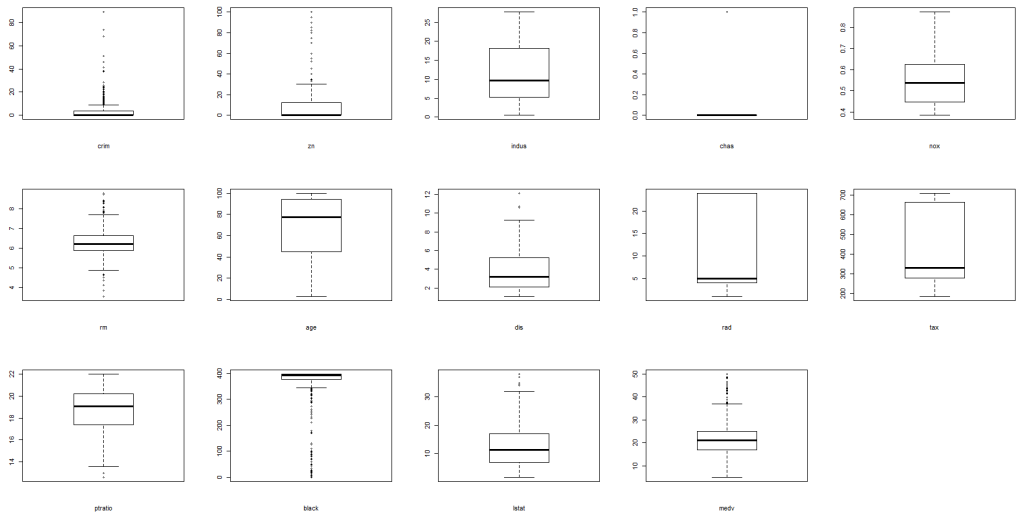

图 6 各特征箱线图

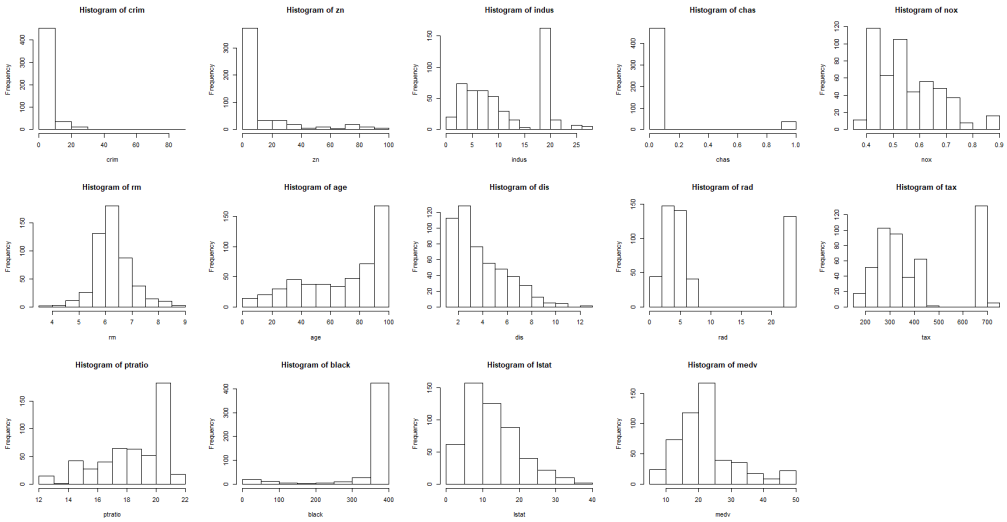

图 7 各特征频数直方图

由上述箱线图和频数直方图观察数据的分布情况，可以看到 CRIM，ZN，RM，DIS，BLACK，LSTAT 等变量均存在较多离群值。

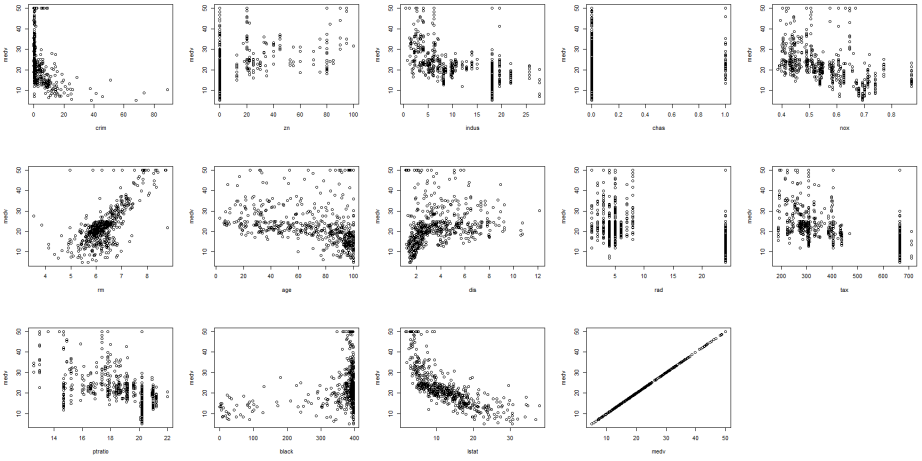

图 8 各特征关于 medv 的散点图

由各个特征与因变量 MEDV 的散点图，可以看出特征 RM 和 LSTAT 与因变量 MEDV 之间有较为明显的线性关系。

表 9 相关性矩阵

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
CRIM	1.00	-0.20	0.41	-0.06	0.42	-0.22	0.35	-0.38	0.63	0.58	0.29	-0.39	0.46
ZN	-0.20	1.00	-0.53	-0.04	-0.52	0.31	-0.57	0.66	-0.31	-0.31	-0.39	0.18	-0.41
INDUS	0.41	-0.53	1.00	0.06	**0.76**	-0.39	0.64	**-0.71**	0.60	**0.72**	0.38	-0.36	0.60
CHAS	-0.06	-0.04	0.06	1.00	0.09	0.09	0.09	-0.10	-0.01	-0.04	-0.12	0.05	-0.05
NOX	0.42	-0.52	0.76	0.09	1.00	-0.30	**0.73**	**-0.77**	0.61	0.67	0.19	-0.38	0.59
RM	-0.22	0.31	-0.39	0.09	-0.30	1.00	-0.24	0.21	-0.21	-0.29	-0.36	0.13	-0.61
AGE	0.35	-0.57	0.64	0.09	0.73	-0.24	1.00	**-0.75**	0.46	0.51	0.26	-0.27	0.60
DIS	-0.38	0.66	-0.71	-0.10	-0.77	0.21	-0.75	1.00	-0.49	-0.53	-0.23	0.29	-0.50
RAD	0.63	-0.31	0.60	-0.01	0.61	-0.21	0.46	-0.49	1.00	**0.91**	0.46	-0.44	0.49
TAX	0.58	-0.31	0.72	-0.04	0.67	-0.29	0.51	-0.53	0.91	1.00	0.46	-0.44	0.54
PTRATIO	0.29	-0.39	0.38	-0.12	0.19	-0.36	0.26	-0.23	0.46	0.46	1.00	-0.18	0.37
B	-0.39	0.18	-0.36	0.05	-0.38	0.13	-0.27	0.29	-0.44	-0.44	-0.18	1.00	-0.37
LSTAT	0.46	-0.41	0.60	-0.05	0.59	-0.61	0.60	-0.50	0.49	0.54	0.37	-0.37	1.00

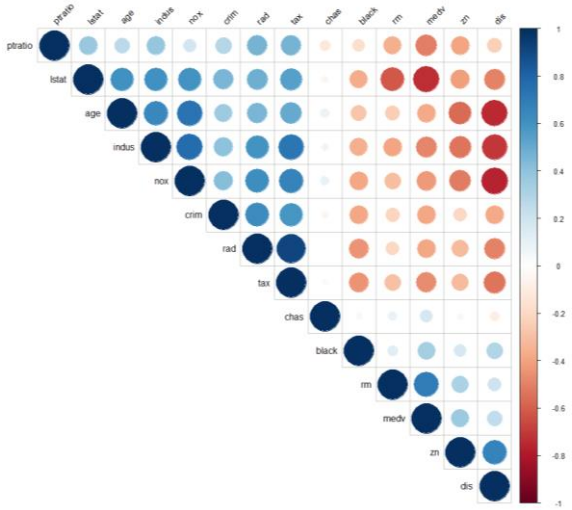

图 9 各变量相关性

结合各个特征之间的散点图以及对变量进行相关性分析，可以看出部分特征之间具有较强的线性关系，如 DIS 与 AGE，CRIM 与 INDUS 等。显然，数据之间具有较强的多重共线性问题。

3.3 数据预处理

- **离群值**

通过上述探索性分析，结合各特征的分布箱线图以及频数分布直方图可知，Boston 数据集存在较多离群值。结合各个特征的实际意义以及分析该数据来源的可靠性，可以认为该数据集真实可靠，统计意义上的离群值可能恰恰是建模中的关键信息，因此选择保留离群值。

- **最小-最大规范化**

为了便于算法的计算高效和准确，在建模前对数据进行规范化处理，利用最

小-最大规范化将数据集所有特征压缩到 0-1 之间。具体公式如下：

$$X = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

- **不平衡数据**

考虑到 Boston 数据集中变量 CHAS 存在不平衡现象。因此，在训练集和测试集划分时特别注意 CHAS 变量比例的问题，并且在模型评价时选用分层 k 折交叉验证，利用打乱样本标签的排序缓解由于数据不平衡对模型带来的影响。

3.4 特征工程

要想利用线性模型能够在数据集上有良好的表现，良好的特征显得尤为重要。为了丰富特征，常用的方法有添加原始数据的交互特征以及多项式特征。

为了提高模型的性能，选择从原始数据中提取多项式特征和交互项特征，对次数最高为 2 和次数最高为 3 的新特征分别建模。

原始数据特征数为 13，丰富特征后，特征数分别为 105 和 560。

以下是对原始数据、最高次为 2 的多项式特征和交互特征数据以及最高次为 3 的多项式特征和交互特征数据进行建模，利用分层 5 折交叉验证得到的模型得分分布密度和模型得分分布柱状图。

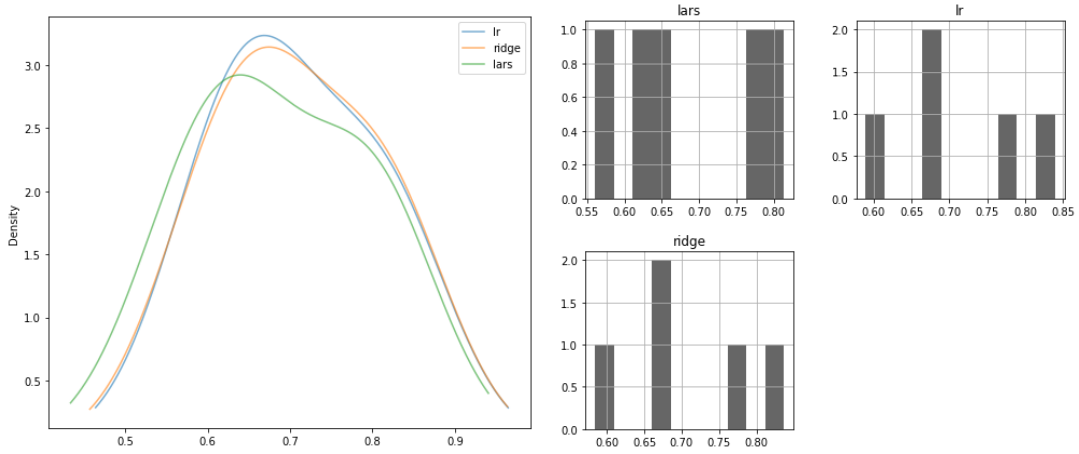

图 10 原始数据

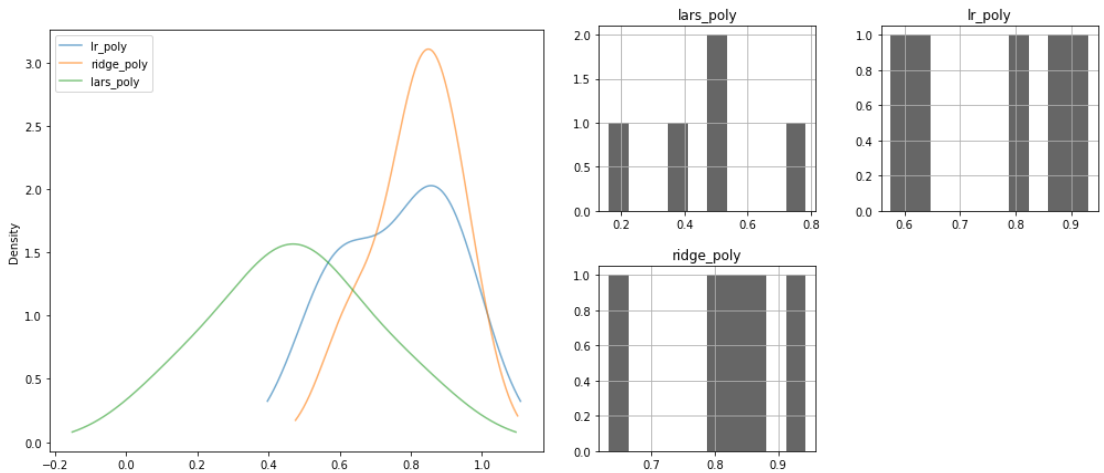

图 11 特征工程 degree=2

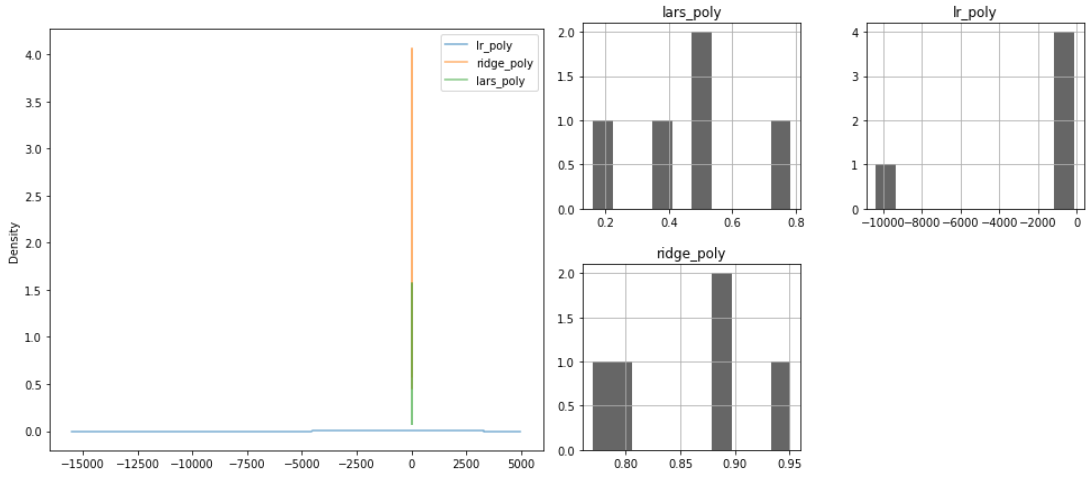

图 12 特征工程 degree=3

3.5 特征选择

考虑到特征工程处理后，13 个原始特征增加到 105 个特征甚至 560 个特征，而数据样本量只有 506 个。因此，特征工程处理过后的数据中有大量无效数据。于是考虑对特征工程后的数据进行单变量统计的自动化特征选择，并设置选取 50%的特征。特征选择后，数据集特征数分别为 52 和 280。

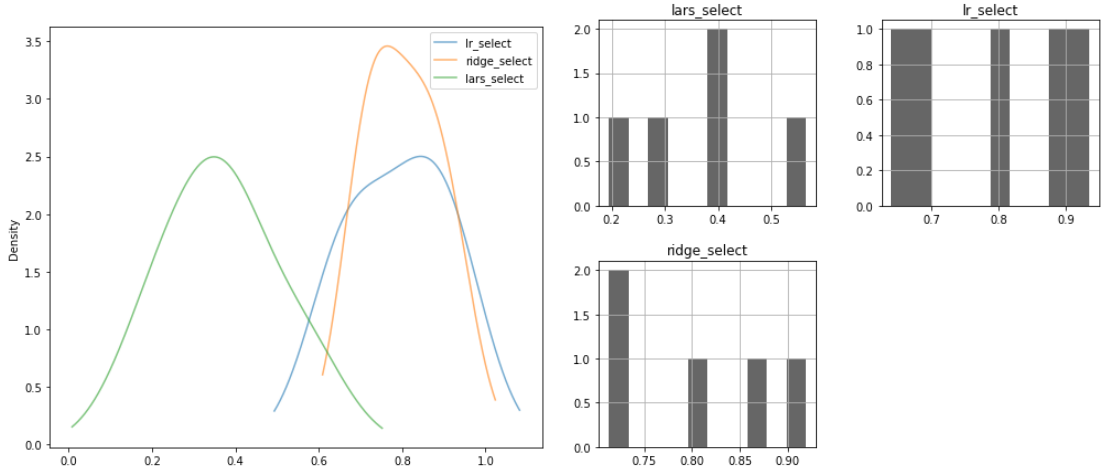

图 13 degree=2 的特征选择

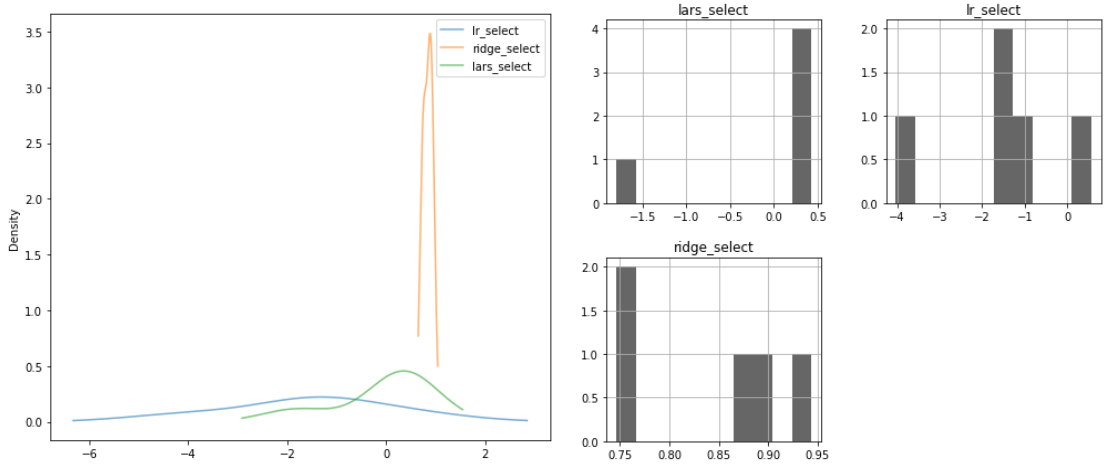

图 14 degree=3 的特征选择

以上是利用特征选择后数据建模，利用分层 5 折交叉验证得到的模型得分分布密度和模型得分分布柱状图。

3.6 三种算法在 Boston 数据上的性能分析

随机抽取数据集的 75%数据作为训练集，对模型进行训练。将其余的数据作为测试集，评估模型的效果。以下是不同数据集在不同的算法下的具体表现：

表 10 三种算法在不同数据集上的表现

R^2		OLS	Ridge	Lars
原有数据集	训练集	0.7468034	0.7460747	0.7415707
	测试集	0.7059096	0.7113378	**0.7119917**
特征工程 degree=2	训练集	0.9338219	0.9324552	0
	测试集	0.859791	**0.8720804**	-0.003799
degree=2 的特征选择	训练集	0.8660538	0.8328518	0
	测试集	0.800821	**0.8454519**	-0.003799
特征工程 degree=3	训练集	1	0.9195351	0.6102866
	测试集	-65.71507	**0.89537**	0.6221072
degree=3 的特征选择	训练集	0.9861102	0.8835069	0.6077417
	测试集	-2.845147	**0.8936534**	0.6196786

表 11 三种算法在其他评价参数中的表现

		OLS	Ridge	Lars
原始数据	MSE 均方误差	24.27460831	23.82656168	**23.77258294**
	解释方差得分	0.708621701	**0.713648688**	0.713450833
	MAE 平均绝对误差	3.290018353	3.231303192	**3.16218496**
	绝对中位差	2.461253433	2.293128218	**2.153202811**
degree=2	MSE 均方误差	11.57303693	**10.55865539**	82.85487779
	解释方差得分	0.861670414	**0.873786089**	3.33067E-16
	MAE 平均绝对误差	2.503492025	**2.418121688**	6.607720275
	绝对中位差	**2.021818589**	2.033845442	4.873350923
特征选择 degree=2	MSE 均方误差	16.4405009	**12.75660457**	82.85487779
	解释方差得分	0.801172608	**0.845458128**	3.33067E-16
	MAE 平均绝对误差	2.806053162	**2.542159427**	6.607720275
	绝对中位差	2.130534412	**1.991536124**	4.873350923
degree=3	MSE 均方误差	5506.750011	**8.636296786**	31.19177028
	解释方差得分	-65.21627891	**0.895832558**	0.62355116
	MAE 平均绝对误差	35.22797943	**2.140530653**	3.871317313
	绝对中位差	14.68410975	**1.519259164**	2.816520135
特征选择 degree=3	MSE 均方误差	317.3834963	**8.777991232**	31.39222996
	解释方差得分	-2.819101669	**0.893811611**	0.621132121
	MAE 平均绝对误差	8.338032846	**2.2143788**	3.885151321
	绝对中位差	3.012324031	**2.2143788**	2.846918331

综合三个算法在不同指标上的表现，可以看出 Lars 算法在原始数据集中的表现最佳。特征工程添加最高次为 2 次的多项式和交互项后，OLS 算法和 Ridge Regression 都能在一定程度上提高了训练集和测试集的准确性，但都出现了过拟合线性。此外该数据集中 Lars 算法的表现很差。对该数据集进行特征选择后，模型精度有一定的下降，但是模型过拟合程度有所缓解，泛化能力更强。综合不同的指标，在最高此为 2 次的数据集中 Ridge Regression 的表现更优。

为了探索在高维数据中三中算法的表现，在原始数据集中添加最高次为 3 次的多项式和交互项，数据集维度从 13 变为 560，此时样本量小于特征数。结合三个算法在不同指标中的表现，可以看出在高维数据中 OLS 算法的表现不尽人意。与最高次为 2 次不同，Lars 算法在最高次为 3 次的数据集中表现远优于最高次为 2 次，且 Ridge Regression 的精度也有所提高，可以认为在 3 次多项式以及相互项中，有对因变量 MEDV 而言较为重要的指标。

4 主要结论

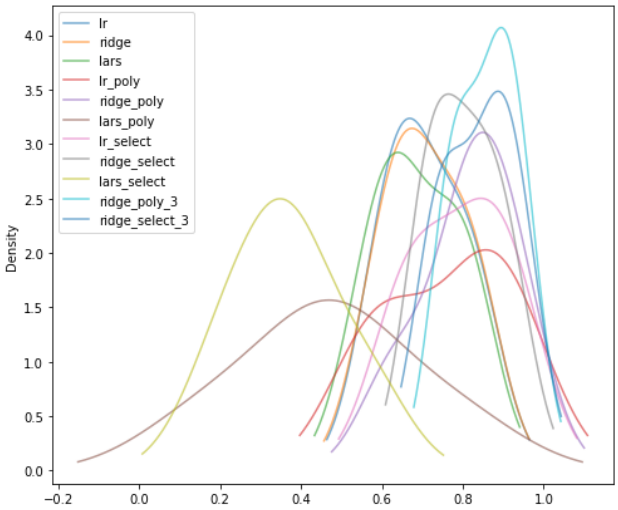

图 15 不同模型得分分布密度合集

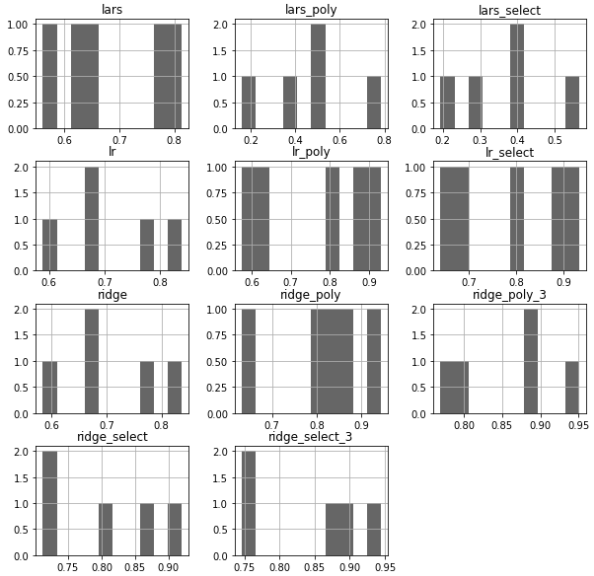

图 16 不同模型得分柱状图合集

综上，不论是在小数据集还是高维数据集，Ridge Regression 的表现都很稳定，模型准确性以及泛化能力也很强。OLS 算法更适合于小数据集，特证数不能

多于样本量。Lars 算法则在高维数据中，尤其在特证数大于样本量的数据中表现优异，但算法运行时间较长。

5 实验程序

random_data.py:

```
#加载所需要的包
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import
explained_variance_score,mean_absolute_error,mean_squared_error,median_absolute
_error
import warnings
warnings.filterwarnings("ignore")


#数据模拟
np.random.seed(1024)
x1=np.random.normal(10, 2, 200)
x2=np.random.normal(5, 3, 200)
x3=np.random.uniform(1,10,200)
x4=np.random.uniform(10,20,200)
x5=x1+2*x2+0.6*x4
y1=10+3*x1+1.5*x2+0.5*x3-2*x4+x5 #有共线性，无噪声
y2=10+3*x1+1.5*x2+0.5*x3-2*x4+x5+2*np.random.randn(200) #有共线性，
有噪声
y3=10+3*x1+1.5*x2+0.5*x3-2*x4+0*x5 #无共线性，无噪声
y4=10+3*x1+1.5*x2+0.5*x3-2*x4+0*x5+2*np.random.randn(200) #无共线
性，有噪声

#无共线性，无噪声
df_random=pd.DataFrame(np.vstack((x1,x2,x3,x4))).T
#df_random=pd.DataFrame(np.vstack((x1,x2,x3,x4,x5))).T
df_random.head()


#探索性分析
corr = df_random.corr()
sns.heatmap(corr,annot=True, cmap='GnBu')
plt.show()
pd.plotting.scatter_matrix(df_random, alpha=0.7, figsize=(8,8));


#训练集，测试集
```

```
from sklearn.model_selection import train_test_split
y=pd.DataFrame(y1)
#y=pd.DataFrame(y2)
#y=pd.DataFrame(y3)
#y=pd.DataFrame(y4)
X_random_train,X_random_test,y_random_train,y_random_test =
train_test_split(df_random,y,random_state=0)
print(df_random.shape)
print(X_random_train.shape)
print(X_random_test.shape)
print(y_random_train.shape)
print(y_random_test.shape)


#linear regression
from sklearn.linear_model import LinearRegression
lr_random = LinearRegression().fit(X_random_train,y_random_train)
print(lr_random.coef_)
print(lr_random.intercept_)
print(lr_random.score(X_random_train,y_random_train))
print(lr_random.score(X_random_test,y_random_test))
y_random_pre=lr_random.predict(X_random_test)
print(mean_squared_error(y_random_test, y_random_pre))
print(explained_variance_score(y_random_test, y_random_pre))
print(mean_absolute_error(y_random_test, y_random_pre))
print(median_absolute_error(y_random_test, y_random_pre))
#分层 k 折交叉验证
from sklearn.model_selection import KFold
kfold=KFold(n_splits=10,shuffle=True,random_state=0)
from sklearn.model_selection import cross_val_score
lr_random_score = cross_val_score(lr_random, df_random, y, cv=kfold)
lr_random_score


#ridge
from sklearn.linear_model import RidgeCV
ridge_random =
RidgeCV(alphas=np.linspace(0.00001,15),store_cv_values=True).fit(X_random_train
,y_random_train)
print(ridge_random.alpha_)
plt.plot(np.linspace(0.00001,15),(ridge_random.cv_values_.mean(axis=0)).T);
plt.plot(ridge_random.alpha_,
min(ridge_random.cv_values_.mean(axis=0).T),'ro');
print(ridge_random.score(X_random_train,y_random_train))
```

```
print(ridge_random.score(X_random_test,y_random_test))
y_random_pre=ridge_random.predict(X_random_test)
print(mean_squared_error(y_random_test, y_random_pre))
print(explained_variance_score(y_random_test, y_random_pre))
print(mean_absolute_error(y_random_test, y_random_pre))
print(median_absolute_error(y_random_test, y_random_pre))
ridge_random_score = cross_val_score(ridge_random, df_random, y, cv=kfold)
ridge_random_score


#lars
from sklearn.linear_model import LarsCV
lars_random = LarsCV().fit(X_random_train,y_random_train)
print(lars_random.score(X_random_train,y_random_train))
print(lars_random.score(X_random_test,y_random_test))
lars_random_score = cross_val_score(lars_random, df_random, y, cv=kfold)
lars_random_score
y_random_pre=lars_random.predict(X_random_test)
print(mean_squared_error(y_random_test, y_random_pre))
print(explained_variance_score(y_random_test, y_random_pre))
print(mean_absolute_error(y_random_test, y_random_pre))
print(median_absolute_error(y_random_test, y_random_pre))
evaluating = {
            'lr':lr_random_score,
            'ridge':ridge_random_score,
            'lars':lars_random_score
            }
evaluating = pd.DataFrame(evaluating)
evaluating.plot.kde(alpha=0.6,figsize=(8,7));
evaluating.hist(color='k',alpha=0.6,figsize=(8,7));


#无共线性，有噪声


#有共线性，无噪声


#有共线性，有噪声
```

Boston.py:

```
#加载所需要的包
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import
mean_squared_error,explained_variance_score,mean_absolute_error,mean_squared_e
rror,median_absolute_error
import warnings
warnings.filterwarnings("ignore")


#boston 数据导入
from sklearn import datasets
boston = datasets.load_boston()
boston.keys()


#探索数据背后含义
boston.DESCR
### boston 数据的特征 df
df_features=pd.DataFrame(boston.data,columns=boston.feature_names)
df_features.head()
#boston 数据集的 y
df_target=pd.DataFrame(boston.target)
df_target.head()
#同数据介绍一致，没有缺失值
df_features.info()
#CHAS 为分类数据，1 代表道临近查尔斯河
df_features.CHAS.value_counts()
#boston 特征数据的分布形况
df_features.describe()
#散点图矩阵
pd.plotting.scatter_matrix(df_features, alpha=0.7, figsize=(15,15),
diagonal='kde');
#相关性分析
corr = df_features.corr()
##相关性矩阵
corr
##相关性热力图
sns.heatmap(corr, cmap='GnBu')#annot=True
plt.show()
```

```
#特征归一化
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
for feature in df_features.columns:
    df_features[feature] = scaler.fit_transform(df_features[[feature]])
df_features.head()
df_features.describe()


#随机分成训练集和测试集
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,
y_test=train_test_split(df_features,df_target,random_state=5)
print(df_features.shape)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
print(df_features.CHAS.value_counts())
print(X_train.CHAS.value_counts())
print(X_test.CHAS.value_counts())

#分层 k 折交叉验证
from sklearn.model_selection import KFold
kfold=KFold(n_splits=5,shuffle=True,random_state=0)

#OLS （查看源代码发现 sklearn 不是用梯度下降求解线性回归的，利用
ols 求解。）
from sklearn.linear_model import LinearRegression
lr = LinearRegression().fit(X_train,y_train)
print(lr.coef_)
print(lr.intercept_)
#OLS R^2
print(lr.score(X_train,y_train))
print(lr.score(X_test,y_test))
#对数据集进行分层 5 折交叉验证，评估模型的泛化能力
from sklearn.model_selection import cross_val_score
lr_score = cross_val_score(lr, df_features, df_target, cv=kfold)
lr_score
y_pre=lr.predict(X_test)
print(mean_squared_error(y_test, y_pre))
print(explained_variance_score(y_test, y_pre))
```

```
print(mean_absolute_error(y_test, y_pre))
print(median_absolute_error(y_test, y_pre))

#ridge
from sklearn import linear_model
model =
linear_model.RidgeCV(alphas=np.linspace(0.001,15),store_cv_values=True).fit(X_tr
ain,y_train)
print(model.alpha_)
from sklearn.linear_model import Ridge
ridge = Ridge(model.alpha_).fit(X_train,y_train)
print(ridge.coef_)
print(ridge.intercept_)
#ridge regression R^2
print(ridge.score(X_train,y_train))
print(ridge.score(X_test,y_test))
ridge_score = cross_val_score(ridge, df_features, df_target, cv=kfold)
ridge_score
y_pre=ridge.predict(X_test)
print(mean_squared_error(y_test, y_pre))
print(explained_variance_score(y_test, y_pre))
print(mean_absolute_error(y_test, y_pre))
print(median_absolute_error(y_test, y_pre))


#lars
from sklearn.linear_model import LarsCV
lars = LarsCV().fit(X_train,y_train)
#lars R^2
print(lars.score(X_train,y_train))
print(lars.score(X_test,y_test))
lars_score = cross_val_score(lars, df_features, df_target, cv=kfold)
lars_score
y_pre=lars.predict(X_test)
print(mean_squared_error(y_test, y_pre))
print(explained_variance_score(y_test, y_pre))
print(mean_absolute_error(y_test, y_pre))
print(median_absolute_error(y_test, y_pre))

evaluating = {
        'lr':lr_score,
        'ridge':ridge_score,
        'lars':lars_score
        }
```

```
evaluating = pd.DataFrame(evaluating)
evaluating.plot.kde(alpha=0.6,figsize=(8,7));
evaluating.hist(color='k',alpha=0.6,figsize=(8,7));


#特征工程（添加交互式特征以及多项式特征）
from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures(degree=2).fit(X_train) #三次以上特征多于样本数
#poly=PolynomialFeatures(degree=3).fit(X_train) #三次以上特征多于样本数
df_features_poly=poly.transform(df_features)
X_train_poly=poly.transform(X_train)
X_test_poly=poly.transform(X_test)
print(df_features.shape)
print(df_features_poly.shape)
print(X_train.shape)
print(X_train_poly.shape)
print(X_test.shape)
print(X_test_poly.shape)
print(poly.get_feature_names())

X_train_poly = pd.DataFrame(X_train_poly,columns=poly.get_feature_names())
X_test_poly = pd.DataFrame(X_test_poly,columns=poly.get_feature_names())
X_train_poly.head()

lr_poly = LinearRegression().fit(X_train_poly,y_train)
print(lr_poly.score(X_train_poly,y_train))
print(lr_poly.score(X_test_poly,y_test))
print(lr_poly.coef_)
print(lr_poly.intercept_)
lr_poly_score = cross_val_score(lr_poly, df_features_poly, df_target, cv=kfold)
#lr_poly_score_3 = cross_val_score(lr_poly, df_features_poly, df_target,
cv=kfold)
lr_poly_score
y_pre=lr_poly.predict(X_test_poly)
print(mean_squared_error(y_test, y_pre))
print(explained_variance_score(y_test, y_pre))
print(mean_absolute_error(y_test, y_pre))
print(median_absolute_error(y_test, y_pre))

model =
linear_model.RidgeCV(alphas=np.linspace(0.001,15),store_cv_values=True).fit(X_tr
ain_poly,y_train)
print(model.alpha_)
ridge_poly = Ridge(alpha=model.alpha_).fit(X_train_poly,y_train)
```

```
print(ridge_poly.score(X_train_poly,y_train))
print(ridge_poly.score(X_test_poly,y_test))
ridge_poly_score = cross_val_score(ridge_poly, df_features_poly, df_target,
cv=kfold)
#ridge_poly_score_3 = cross_val_score(ridge_poly, df_features_poly, df_target,
cv=kfold)
ridge_poly_score
y_pre=ridge_poly.predict(X_test_poly)
print(mean_squared_error(y_test, y_pre))
print(explained_variance_score(y_test, y_pre))
print(mean_absolute_error(y_test, y_pre))
print(median_absolute_error(y_test, y_pre))

lars_poly = LarsCV().fit(X_train_poly,y_train)
print(lars_poly.score(X_train_poly,y_train))
print(lars_poly.score(X_test_poly,y_test))
lars_poly_score = cross_val_score(lars_poly, df_features_poly, df_target,
cv=kfold)
#lars_poly_score_3 = cross_val_score(lars_poly, df_features_poly, df_target,
cv=kfold)
lars_poly_score
y_pre=lars_poly.predict(X_test_poly)
print(mean_squared_error(y_test, y_pre))
print(explained_variance_score(y_test, y_pre))
print(mean_absolute_error(y_test, y_pre))
print(median_absolute_error(y_test, y_pre))

evaluating = {
        'lr_poly':lr_poly_score,
        'ridge_poly':ridge_poly_score,
        'lars_poly':lars_poly_score
        }
evaluating = pd.DataFrame(evaluating)
evaluating.plot.kde(alpha=0.6,figsize=(8,7));
evaluating.hist(color='k',alpha=0.6,figsize=(8,7));


#特征选择
from sklearn.feature_selection import SelectPercentile
select = SelectPercentile(percentile=50).fit(df_features_poly,df_target)
df_features_poly_selected = select.transform(df_features_poly)
X_train_selected = select.transform(X_train_poly)
X_test_selected = select.transform(X_test_poly)
print(df_features_poly_selected.shape)
```

```
        print(X_train_poly.shape)
        print(X_train_selected.shape)

        lr_selected = LinearRegression().fit(X_train_selected,y_train)
        print(lr_selected.score(X_train_selected,y_train))
        print(lr_selected.score(X_test_selected,y_test))

        lr_selected_score = cross_val_score(lr_selected, df_features_poly_selected,
df_target, cv=kfold)
        #lr_selected_score_3 = cross_val_score(lr_selected, df_features_poly_selected,
df_target, cv=kfold)
        print(lr_selected_score)
        y_pre=lr_selected.predict(X_test_selected)
        print(mean_squared_error(y_test, y_pre))
        print(explained_variance_score(y_test, y_pre))
        print(mean_absolute_error(y_test, y_pre))
        print(median_absolute_error(y_test, y_pre))

        #ridge
        model_selected =
linear_model.RidgeCV(alphas=np.linspace(0.00000001,15),store_cv_values=True).fit
(X_train_selected,y_train)
        print(model_selected.alpha_)
        print(model_selected.score(X_train_selected,y_train))
        print(model_selected.score(X_test_selected,y_test))

        ridge_selected_score = cross_val_score(model_selected,
df_features_poly_selected, df_target, cv=kfold)
        #ridge_selected_score_3 = cross_val_score(model_selected,
df_features_poly_selected, df_target, cv=kfold)
        print(ridge_selected_score)
        y_pre=model_selected.predict(X_test_selected)
        print(mean_squared_error(y_test, y_pre))
        print(explained_variance_score(y_test, y_pre))
        print(mean_absolute_error(y_test, y_pre))
        print(median_absolute_error(y_test, y_pre))

        lars_selected = LarsCV().fit(X_train_selected,y_train)
        print(lars_selected.score(X_train_selected,y_train))
        print(lars_selected.score(X_test_selected,y_test))
        lars_selected_score = cross_val_score(lars_selected, df_features_poly_selected,
df_target, cv=kfold)
        #lars_selected_score_3 = cross_val_score(lars_selected,
df_features_poly_selected, df_target, cv=kfold)
```

```
print(lars_selected_score)

y_pre=lars_selected.predict(X_test_selected)
print(mean_squared_error(y_test, y_pre))
print(explained_variance_score(y_test, y_pre))
print(mean_absolute_error(y_test, y_pre))
print(median_absolute_error(y_test, y_pre))

evaluating = {
        'lr_select':lr_selected_score,
        'ridge_select':ridge_selected_score,
        'lars_select':lars_selected_score
        }
evaluating = pd.DataFrame(evaluating)
evaluating.plot.kde(alpha=0.6,figsize=(8,7));
evaluating.hist(color='k',alpha=0.6,figsize=(8,7));

evaluating_all = {
        'lr':lr_score,
        'ridge':ridge_score,
        'lars':lars_score,
        'lr_poly':lr_poly_score,
        'ridge_poly':ridge_poly_score,
        'lars_poly':lars_poly_score,
        'lr_select':lr_selected_score,
        'ridge_select':ridge_selected_score,
        'lars_select':lars_selected_score,
        #'ridge_poly_3':ridge_poly_score_3,
        #'ridge_select_3':ridge_selected_score_3
        }
evaluating_all = pd.DataFrame(evaluating_all)
evaluating_all.plot.kde(alpha=0.6,figsize=(8,7));
evaluating_all.hist(color='k',alpha=0.6,figsize=(10,10));
```

部分 R 代码：

```
library(MASS)
library(Hmisc)
library(corrplot)
df<-Boston
head(df)
describe(df)
table(df$chas)

#箱线图
```

```
par(mfrow = c(3, 5))
for(i in 1:dim(df)[2]) {
   boxplot(df[,i],xlab =colnames(df)[i])
}

#直方图
par(mfrow = c(3, 5))
for(i in 1:dim(df)[2]) {
   hist(df[,i],xlab =colnames(df)[i],main = paste("Histogram of" ,
colnames(df)[i]))
}

#散点图
par(mfrow = c(3, 5))
for(i in 1:dim(df)[2]) {
   plot(df[,i],y=df$medv,xlab=colnames(df)[i],ylab='medv')
}

##相关性分析
cor_matr<-cor(df)
cor_matr#变量间的相关系数矩阵
symnum(cor_matr)
corrplot(cor_matr, type="upper", order="hclust", tl.col="black", tl.srt=45)
```