高维数据推断大报告　变量与特征选择

统计 81 于越 *

November 20, 2021

Contents

1 摘要与引言 **2**

2 系数压缩与特征选择原理 **2**

2.1 Lasso 方法 2

2.1.1 LARS 算法 3

2.1.2 广义线性模型：glmnet 包 4

2.2 弹性网 elastic net 方法 4

2.3 SCAD 算法 5

2.4 自适应 lasso 6

3 相关实验一：乳腺癌数据集 **7**

3.1 诊断数据集 (wdbc) 相关实验 7

3.1.1 Lasso 方法 7

3.1.2 elastic net 弹性网算法 13

3.1.3 自适应 Lasso 15

3.1.4 SCAD 算法 17

3.2 预测数据集 (wpbc) 相关实验 19

3.2.1 Lasso 算法 19

3.2.2 elastic net 弹性网算法 25

3.2.3 自适应 Lasso 27

3.2.4 SCAD 算法 28

*学号：2183210516

4 相关实验二：步法分类数据集 (n<p 情形) 30
4.0.1 Lasso 算法 . . . 31
4.0.2 elastic net 弹性网算法 . . . 34
4.0.3 自适应 Lasso . . . 35
4.0.4 SCAD 算法 . . . 36

5 相关实验三：与充分降维方法比较 38
5.1 特征选择方法 . . . 39
5.1.1 Lasso 算法 . . . 39
5.1.2 elastic net 弹性网算法 . . . 41
5.1.3 自适应 Lasso . . . 43
5.1.4 SCAD 算法 . . . 44
5.2 对比充分降维的效果 . . . 46

6 总结 48

A 相关代码 49

1 摘要与引言

本篇报告主要探讨了变量与特征选择问题。主要涉及几种对于目标优化函数的惩罚方式，进而构造出正则化形式下的目标优化函数，通过新的目标优化函数达到求解稀疏解的目的，是对模型中的自变量系数进行压缩的方法。最后对于压缩后的系数，若系数被压缩为 0，则该变量为不重要变量，在建立模型中可以舍弃掉。若系数被压缩不为 0，则该变量可视为重要变量，在建立模型时需要被选择上。

该报告主要介绍并实验了以下几种特征选择的方法：Lasso 正则方法 (L1 正则化方法)，弹性网正则方法，自适应 Lasso 方法以及 SCAD 方法。主要用到了以下几个数据集进行实验：威斯康星大学乳腺癌诊断与预测数据集 (下简写为 wdbc 与 wpbc)，一般人群步法运动分类数据集 (下简写为 pgds)，阿尔及利亚森林大火分类数据集 (下简写为 forest). 主要做法是对于这些高维数据集，通过提到的这几种方法进行变量与特征选择，对于特征选择后的新自变量建立模型，进而对数据样本进行分类，与真实结果进行比较，计算正确率。主要的考察指标为分类正确率与特征压缩数。因为我们一方面需要保证模型的精度与正确性，另一方面需要保证模型的约简性。

最后对于 forest 数据集，我将之前用 SIR 方法进行充分降维得到的分类结果与通过特征选择方法得到的模型进而分类的结果进行对比，比较了两种方法的性能。这启发我们对于不同的实际数据集，需要选择适当的方法处理问题，具体问题具体分析。

关键词：特征选择　　正则　　惩罚项　　约简模型

2 系数压缩与特征选择原理

我已经学习过了许多分类与回归的方法。这些方法都需要去依赖于因变量数据本身。在我们的一般理解下，往往自变量越多意味着我们拥有更多的信息。显然信息越多，对于我们解释一些事物的原理，以及对未来做预测都是有所帮助的。但是，我们也应该有理由去相信，自变量的影响也应该是“稀疏”的。很多时候，决定事物发展的很有可能是少数的自变量因素，因此我们就希望可以忽略掉一些自变量，希望从大量的自变量中建立更为简约的但是仍旧是十分有效的模型，这样就可以大大节省自变量测量的成本。因此，变量与特征选择的方法就应运而生。这也是回归系数进行“压缩”的方法。

下面，我主要介绍几种可以用来做变量与特征选择的收缩方法，例如 lasso 回归，弹性网 elastic net，自适应 lasso(adaptive lasso) 以及 scad 方法。

2.1 Lasso 方法

我们已经知道了最为常用的最小二乘方法 (OLS)，以及在最小二乘方法基础上对其进行一定改进的岭回归方法。岭回归的基本原理在于，对原数据集的目标优化加上 L2 范数的正则惩罚项。这样就可以有效地控制回归系数，使得原无约束优化问题变为一约

束优化问题，以防止过拟合的情形发生。另一种对于岭回归的解释在于，通过控制回归系数以达到控制并减小方差的目的。这也是有效减少 MSE 的一个重要方法。

尽管岭回归可以有效提升模型拟合的精确度，但是其得到的拟合系数都是非零的。换言之，岭回归是无法进行变量选择的。故 Tibshirani 在 1996 年提出了 lasso 方法。

lasso 方法的原理很简单，主要是通过 L1 正则项替换岭回归中的 L2 正则项，使得在 L2 范数意义下的非凸约束优化问题变为一个 L1 正则意义下的凸约束优化问题。故在 lasso 中优化问题的最优解通常是在边界处达到。这样就可以有效达到“稀疏解”的效果。因此一些系数项就可以恰好被压缩为 0，我们也就实现了变量选择。[1]

lasso 方法的主要优化目标如下所示：

$$\hat{\beta}^{LASSO} = \underset{\beta}{argmin} \frac{1}{2}||y - X\beta||^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{1}$$

或者等价为

$$\begin{aligned} &\hat{\beta}^{LASSO} = \underset{\beta}{argmin} \frac{1}{2}||y - X\beta||^2 \\ &s.t. \sum_{j=1}^{p} |\beta_j| \leq t \end{aligned} \tag{2}$$

2.1.1 LARS 算法

正如前文中 lasso 方法的主要优化目标，鉴于 L1 正则范数难以求导，我们无法用一般的优化方法求解最优解。此时对于这一问题，我们就需要引入一些特殊的算法进行最优解的计算。

LARS 是最小角回归算法的缩写，出自著名统计学家 Efron. 该算法假定 p 个预测变量 $X_1, X_2, \cdots, X_p$ 是线性的。我们已经知道了在选择变量时的逐步回归算法，在其中最为常用的是向前回归。向前回归，就是先选择和响应最相关的变量，进行最小二乘回归。然后在这个模型的基础上，再选择和此时残差相关度最高的 (也就是相关度次高) 的变量，加入模型重新最小二乘回归。之后再如法继续，直到在某些度量模型的最优性准则之下达到最优，从而选取一个最优的变量子集进行回归分析，得到的模型是相比原模型更加简便，更易于解释的[2]。逐步回归法这种方法，优势在于提高了模型的精确度 (方差变小)，且计算的效率较高，但是却牺牲了模型准确性 (预测有偏)。

与此同时也比较著名的选择变量的方法还有 Forward Stepwise 算法。Forward Stepwise 算法的基本思想在于先找出和响应最相关的一个变量，找到第一个变量后不急于做最小二乘回归，而是在变量的 solution path 上一点一点的前进 (所谓 solution path 是

[1]我们通常认为在 lasso 以及其他用于变量选择的算法中，压缩后稀疏变为 0 的是不重要变量，在变量选择过程中可以进行剔除。

[2]该段文字摘自 https://blog.csdn.net/mimihou/article/details/8224714

指一个方向，逐步回归是在这个方向上进行)，每前进一点，都要计算一下当前的残差和原有的所有变量的相关系数，找出绝对值最大的相关系数对应的变量。总之，Forward Stepwise 每次都是根据选择的变量子集，完全拟合出线性模型，计算出 RSS，再设计统计量（如 AIC）对较高的模型复杂度作出惩罚，这样的策略过于保守，每次只加入改变量的很小一部分，会导致计算速度较慢。

而 Efron 提出的 LARS 算法可以认为是以上两种算法的折中。它从所有系数均为 0 的情况开始，每次加入最相关的变量。虽然此时没有把变量完全加入，但是却计算了可能的最大步长，这使得其在不损失计算精度的情况下，也保证了计算的效率。

通常情况下，LARS 的算法实际执行步骤如下[3]：

1. 对自变量进行标准化（去除不同尺度的影响），对目标变量进行中心化（去除截距项的影响），初始的所有系数都设为 0。

2. 找出和残差相关度最高的自变量 X_j.

3. 将 X_j 的系数 β_j 从 0 开始沿着最小二乘方向 (只有一个变量 X_j 的最小二乘估计) 的方向变化，直到某个新的变量 X_k 与残差的相关性大于 X_j.

4. X_j 和 X_k 的系数 β_j 和 β_k，一起沿着新的最小二乘方向 (加入了新变量 X_k 的最小二乘估计) 的方向移动，直到有新的变量被选入

5. 重复 2，3，4，直到所有变量被选入，最后得到的估计就是普通线性回归的 OLS.

在 R 语言中，可以使用 Lars 扩展包进行 LARS 算法的相关应用。

值得一提的是，LARS 算法固然有着一些优势，但是对于很多数据集，其精度还是存在一些牺牲的。这点会在后续实验中有所体现。

2.1.2 广义线性模型：glmnet 包

在 R 语言中，要解决 lasso 的计算问题，较为推荐使用的是 glmnet 扩展工具包。该工具包可以很好地用于解决用 lasso 方法进行变量选择后的广义线性模型的拟合。

该包的优势在于，对于不同类型的广义线性模型变量都可以进行求解，包括二分类 logistic 回归模型、多分类 logistic 回归模型、Possion 模型、Cox 比例风险模型与 SVM.

该扩展包的另一个优势在于其对于超参数 λ 的选择。改包提供了极为全面的交叉验证选择超参数的方法，可以通过可视化的方式有效帮助我们选择最佳的 λ 进行变量与特征的选择。

2.2 弹性网 elastic net 方法

我已经对于 lasso 回归做了初步介绍。lasso 具有“稀疏化”的功能，可以协助我们选择出较好的回归因变量，而岭回归可以有效地压缩系数。因此，我们考虑一种将二者相结合的方法，可以有效地综合二者的优点，是一种较为折中的方法，就是弹性网 elastic net 方法。

[3]详细的 LARS 算法推导过程可以参考《大数据分析：方法与应用》，王星等编著，清华大学出版社

此时的约束优化目标函数为

$$\hat{\beta}^{enet} = \underset{\beta}{argmin}\frac{1}{2}||y - X\beta||^2 + \lambda_1\sum_{j=1}^{p}|\beta_j| + \lambda_2\sum_{j=1}^{p}\beta_j^2 \tag{3}$$

或者为

$$\begin{aligned}\hat{\beta}^{enet} &= \underset{\beta}{argmin}\frac{1}{2}||y - X\beta||^2 \\ s.t.&\lambda_1\sum_{j=1}^{p}|\beta_j| + \lambda_2\sum_{j=1}^{p}\beta_j^2 \le t\end{aligned} \tag{4}$$

此时可以用 shooting 算法进行梯度下降算法的迭代计算[4]。在此处，对完整推导不做过多赘述，主要给出最后的迭代公式结论。此时算法中的每一步迭代公式为

$$\beta_j = \frac{sign(X_j^{\mathrm{T}}\epsilon_{-j})(|X_j^{\mathrm{T}}\epsilon_{-j} - \lambda_1|)_+}{1 + \lambda_2} \tag{5}$$

其中 $\epsilon_{-j} = y - X_{-j}\beta_{-j}$. 其中 X_{-j} 是矩阵 X 去掉第 j 列后的矩阵。

2.3 SCAD 算法

前文中的算法主要是设计对于变量的“惩罚”，以达到较好的系数估计与变量选择效果。但是尽管惩罚项的存在时的最小二乘的估计更为精确，但是惩罚项只要存在，我们得到的估计值 $\hat{\beta}$ 一定是有偏的，这是因为估计期望的绝对值要比真实值的绝对值更小。故为了克服这一精度上的缺陷，Fan 提出了 SCAD 惩罚项，形成了 SCAD 算法。该算法的优势在于，可以在 β_j 具有较高的显著性时，取消对于系数的惩罚。这样就可以使得在本身数据集的一些自变量较为显著的情况下，减小系数估计时的偏，以达到增大算法精度的目的。

SCAD 惩罚项的导数形式为

$$p'_\lambda(\beta_j) = \lambda\{I(|\beta_j| \le \lambda) + \frac{(a\lambda - |\beta_j|)_+}{(a-1)\lambda}I(|\beta_j| > \lambda)\}sign(\beta_j) \tag{6}$$

其中的 a>2 为一常数。此时的求解算法仍旧是 Shooting 算法。此时算法中的每一步迭代公式为

$$\beta_j = \begin{cases}\epsilon_{-j}^{\mathrm{T}}X_j, & |\epsilon_{-j}^{\mathrm{T}}X_j| > a\lambda \\ sign(X_j^{\mathrm{T}}\epsilon_{-j})(|X_j^{\mathrm{T}}\epsilon_{-j}| - \lambda)_+, & |\epsilon_{-j}^{\mathrm{T}}X_j| < 2\lambda \\ \{(a-1)\epsilon_{-j}^{\mathrm{T}}X_j - sign(\epsilon_{-j}^{\mathrm{T}}X_j)a\lambda\}/(a-2), & \text{其他}\end{cases} \tag{7}$$

[4]该方法对于 lasso 正则的求解可以参考网站 http://jayveehe.github.io/2016/09/09/Shooting-Algorithm/

从上式中我们可以看出，当 X_j 与残差 ϵ_j 的相关性足够高时，即 $|X_j^{\mathrm{T}}\epsilon_{-j}| > a\lambda$ 时，就无需对 β_j 再进行惩罚了。此时的 SCAD 惩罚项也是一个非凸的惩罚项。

SCAD 惩罚项在一定程度上消除了由惩罚项的存在而带来的偏差。值得一提的是，SCAD 惩罚项具有极其优秀与神奇的 Oracle 性质。这是由 Fan 所证明的性质。我们假定超参数 λ 与样本量有关，即 λ_n 表示样本量为 n 时所选择的调节系数 λ. 探讨其极限性质，即当 $\lambda_n \to 0, \sqrt{n}\lambda_n \to \infty$ 的条件下，SCAD 可以保证：

$$\begin{aligned} &(1)P(\beta_{A^c} = 0) \to 1 \\ &(2)\hat{\beta}_A \text{以} \sqrt{n} \text{的速率收敛到} \beta_A^0, \text{且可以达到最优估计效率} \end{aligned} \tag{8}$$

这一性质表明，在 $\lambda_n \to 0$ 的过程中，也希望继续保留对于系数项为 0 的项的惩罚 (即 $\beta_j^0 = 0$) 的惩罚。这样可以有助于我们以较高的概率 (趋近于 1) 得到真实的非零系数的集合 $A = \{j : \beta_j^0 \neq 0\}$. 而与此同时，想要保持对原有的系数为零项 $\beta_j^0 = 0$ 的惩罚，$\lambda_n \to 0$ 的速度不能太快，惩罚区域的缩小速度不能过快，所以需要限制其速度为 $\sqrt{n}$.

Oracle 性质又意味神谕性质，表面 SCAD 方法下，可以事先知道 β_j 系数中存在的非零分量，且可以用最优估计效率对它们进行估计。而对于其他本应该为零的分量，系数估计值可以自动依概率 1 收敛到 0. 由于证明过于复杂，此处不过多赘述[5]。

SCAD 算法在 R 语言中可以由 ncvreg 扩展包进行实现。后文中会对其实验结果进行分析。

2.4 自适应 lasso

自适应 lasso 算法是 Zou Hui 于 2006 年提出的惩罚项算法。该算法结合了岭回归的 L2 惩罚项与 Lasso 回归的 L1 惩罚项，在自适应 Lasso 中，将惩罚项设置为系数绝对值的加权平均值 $\sum_{j=1}^{p} \omega_j|\beta_j|$. 可以说，岭回归方法与 Lasso 回归方法是自适应 Lasso(Adaptive Lasso) 方法的特例。

自适应 Lasso 也可以看成是一种 Lasso 方法的两部回归修正。它的惩罚形式也可以写为

$$p_\lambda(\beta_j) = \frac{|\beta_j|}{|\hat{\beta}_j|^q} \tag{9}$$

其中，$\hat{\beta}_j$ 是系数的 $n^{\frac{1}{2}}$ 相合估计。估计系数时，我们可以第一步通过一般的 Lasso 算法计算 $\hat{\beta}_j$, 并且采用交叉验证的方法调节参数 λ。第二步将 $\hat{\beta}_j$ 代入 Adaptive Lasso, 再用交叉验证的方法选择参数 λ，如此循环往复即可。

值得一提的是，Adaptive Lasso 算法与 SCAD 算法一样，同样也具有 Oracle 性质，可以事先知道自变量中系数的非零分量[6]。

[5]详细证明可参见 Fan J.Li R. Variable Selection via nonconcave penalized likelihood and its Oracle Property.

[6]详细证明见 Zou,Hastie.Regression and Variable Selection via the Elastic Net.

Adaptive Lasso 算法在 R 语言中可以通过 msgps 扩展包进行实现。该包提供了自适应 Lasso 回归的求解方法，且对于不同的优化标准都可以进行结果的预测，例如 C_p 统计量，AIC，GCV 与 BIC 等等。详情可参见后文中的实验分析。

3 相关实验一：乳腺癌数据集

此处用的数据集为两个乳腺癌数据集[7]。该数据集描述了威斯康星大学的乳腺癌调研数据集，分为诊断数据集 (wdbc) 与预测数据集 (wpbc) 两种。对于诊断数据集 (wdbc)，主要描述了 569 位乳腺癌患者的所处地区、身体状况、所处环境、营养参数等共 30 个特征对于其乳腺癌诊断结果的影响，主要依据这些因素将 569 位患者分为 B(benign 良性乳腺癌) 与 M(malignant 恶性乳腺癌) 两种结果。而对于预测数据集 (wpbc)，主要描述了对于 198 位乳腺癌患者的各种身体信息、基因影响、身处环境、营养参数等 32 个特征对于其乳腺癌是否能得到治愈的影响，主要依据这些因素将 198 位患者分为 R(recur 乳腺癌能治愈) 与 N(nonrecur 乳腺癌不能治愈) 两种结果。在这里，对于患者的诊断与预测所用的特征数过多，此时通过这些因变量生成的模型约简性质不够，故我考虑利用前文中介绍过的特征与变量选择方法进行变量系数的压缩，进而建立 Logistic 回归模型进行预测，将预测结果与实际结果进行比较，计算分类的正确率。

3.1 诊断数据集 (wdbc) 相关实验

首先我对诊断数据集 (wdbc) 进行实验。主要采用的变量与特征选择方法为前文中介绍过的 lasso 方法、elastic net 弹性网方法、Adaptive lasso 自适应 Lasso 以及 SCAD 算法。在这里，我默认分类为 B(良性乳腺癌) 的患者记为 0，而分类为 M(恶性乳腺癌) 的患者记为 1.

3.1.1 Lasso 方法

首先我用 Lasso 方法对于 wdbc 数据集进行实验。在这里，我主要把两种数据包 Lars 包与 glmnet 包都对该数据集进行了实验，对比一下实验结果。在这里，我将 wdbc 中共 569 个样本数据集以数量的 7：3 分为训练集与测试集进行实验，用训练集训练得到的特征选择结果作用于测试集，进一步建立模型观察模型预测的结果如何。

首先用 Lars 包进行实验。先动用 lars 包中的 CV.fit 函数来求解出最适用于 L1 正则项系数的超参数 λ 的值。值得一提的是，CV.fit 函数的默认方法是十折交叉验证方法，将数据集分为十组进行实验，每次取其中的 9 组为训练集，最后一组为测试集，进行测试得到模型的 MSE 值，再以横轴为超参数 λ 的值，纵轴为模型 MSE 值可视化，进而找到 MSE 取最小值位置的 λ 值选择为真实实验时所用的超参数值。

[7]数据集来源：http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

在这里采用 Lars 包中的 CV.fit 函数得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

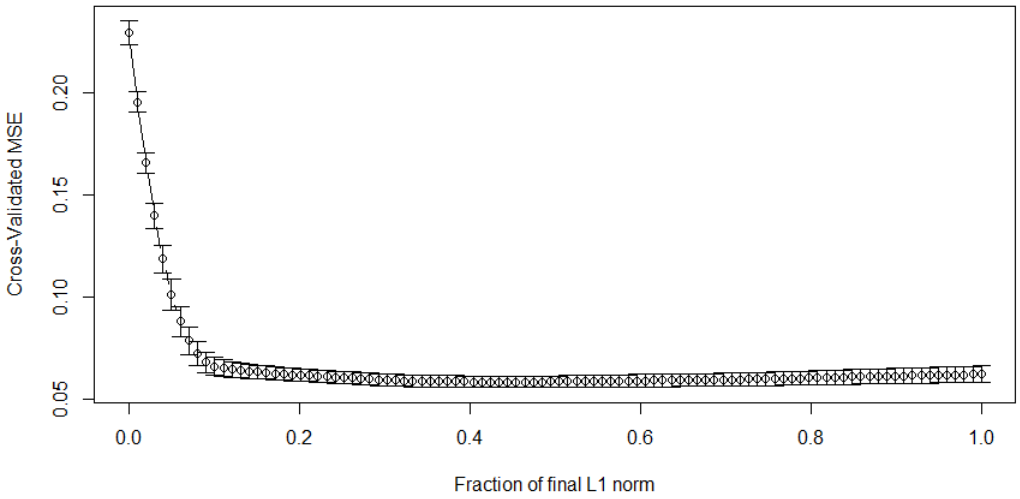

Figure 1: Lars 包作用于 wdbc 数据集 CV 结果

经过观察，可以取出交叉验证 MSE 值最小位置对应的 λ 值为:

```
> tcv$index[which.min(tcv$cv)]
[1] 0.4242424
```

Figure 2: Lars 包作用于 wdbc 数据集 CV 结果 (最小值)

可以看到当 λ=0.424 时，交叉验证 MSE 值取为最小值。
接着，采用 lars 函数对训练集数据进行变量选择，可以得到的变量选择图如下所示：

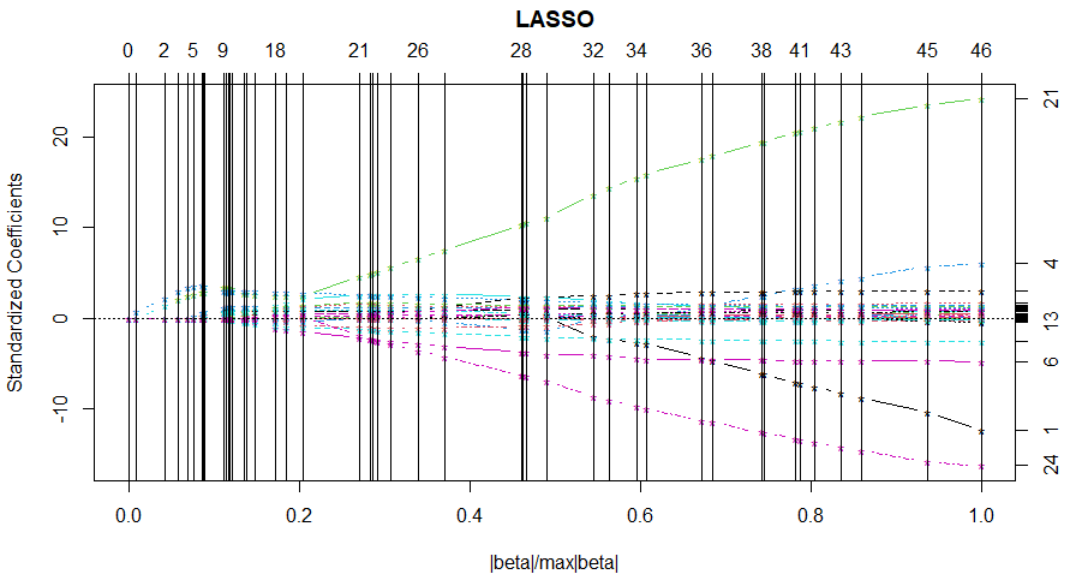

Figure 3: Lars 包作用于 wdbc 数据集变量选择结果

该图的横轴可以理解为系数的压缩程度。横轴数值越小表示压缩的程度越大。可以看到随着压缩程度增大，越来越多的变量被剔除出模型选择变量的范围。从图上还可以看出，变量 V21 与变量 V24 对于模型选择的影响较大，故对于特征选择完成的模型，这两个自变量应该被包含其中。

接着，我动用 predict 函数得到选择交叉验证 MSE 值最小时的超参数 λ 情况下的用 Lasso 算法计算得到的各自变量对应的系数值，如下所示：

```
> trainfit
    V3     V4     V5     V6     V7     V8     V9    V10    V11    V12    V13    V14    V15    V16
 0.000  0.000  0.000  0.000 -0.359 -3.194  1.107  1.614  0.500  0.000  0.462  0.028  0.000 -0.001
   V17    V18    V19    V20    V21    V22    V23    V24    V25    V26    V27    V28    V29    V30
 0.000  0.000 -2.978  2.455  1.258  0.000  0.096  0.009  0.000  0.000  1.942  0.000  0.324  1.579
   V31    V32
 0.491  2.641
```

Figure 4: Lars 包作用于 wdbc 数据集系数结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> active.index
 V7  V8  V9 V10 V11 V13 V14 V16 V19 V20 V21 V23 V24 V27 V29 V30 V31 V32
  5   6   7   8   9  11  12  14  17  18  19  21  22  25  27  28  29  30
```

Figure 5: Lars 包作用于 wdbc 数据集变量选择结果

可以看到，此时的变量由原来的 30 个压缩成了 18 个。故我在这 18 个变量的基础上建立模型进行预测。对于 logistic 模型，常用的分类模型为，对于得到的因变量结果大于 0.5 的值分类为 1，而得到的因变量结果小于 0.5 的分类为 0. 若采用 lars 包自带的预测函数进行预测，根据得到的预测结果计算与真实结果之间的正确率如下所示：

```
> wrong_num
[1] 176
> rightrate_train
[1] 0.5510204
```

Figure 6: Lars 包作用于 wdbc 训练数据集变量选择正确率

可以看到 392 个训练数据集中分类出错了 176 个，正确率仅为 55.1%，可以看到正确率较低。

将 30 个变量压缩为 18 个变量的数据集结果作用在测试集上进行实验，得到的预测结果计算与真实结果之间的正确率如下所示：

```
> wrong_num
[1] 68
> rightrate_test
[1] 0.6158192
```

Figure 7: Lars 包作用于 wdbc 测试数据集变量选择正确率

对于 177 个测试数据集分类错误 68 个，实验正确率为 61.6%. 可以看到采用 R 语言 Lars 包中自带的预测函数实则效果一般。

下用 glmnet 包对 wdbc 数据集进行 Lasso 算法的实验。glmnet 包中同样有着 CV.fit 函数。利用 CV.fit 函数可以求解出最适用于 L1 正则项系数的超参数 λ 的值。调用 glmnet 包中的 CV.fit 函数，我可以得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

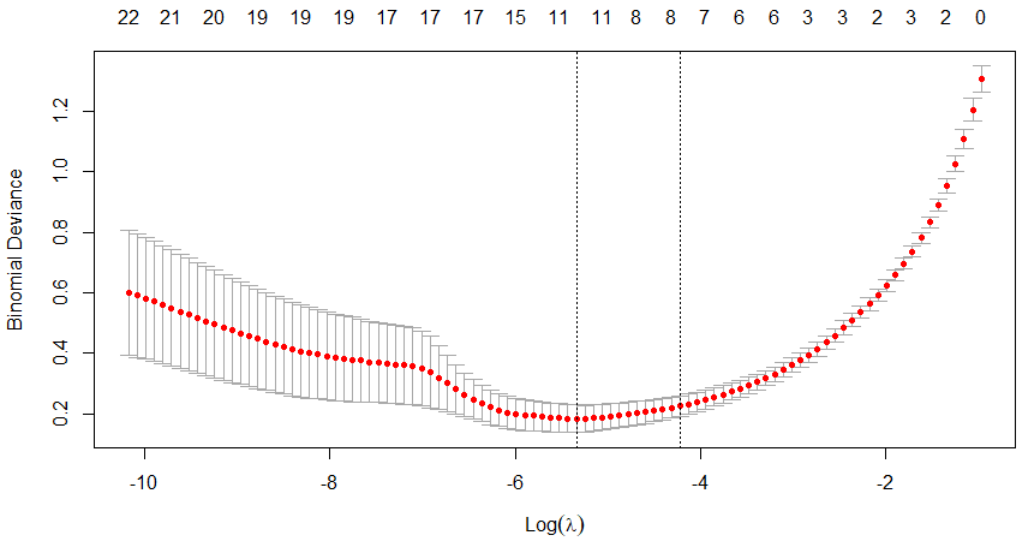

Figure 8: glmnet 包作用于 wdbc 数据集 CV 结果

经过观察，可以取出交叉验证 MSE 值最小位置对应的 λ 值[8]为:

[8]注意下图中的 λ_{min} 表示交叉验证 MSE 最小值处的 λ 值，而 λ_{lse} 表示满足误差在最小值区域内一个单位的 λ 值的最大值。在这里主要采用 λ_{min} 进行实验

```
> cv.fit$lambda.min
[1] 0.004811109
> cv.fit$lambda.1se
[1] 0.01469243
```

Figure 9: glmnet 包作用于 wdbc 数据集 CV 结果

可以看到当 λ=0.0048 时，交叉验证 MSE 值取为最小值。

接着，采用 glmnet 函数对训练集数据进行变量选择，取定 λ 值为 0.0048，可以得到的变量选择图如下所示：

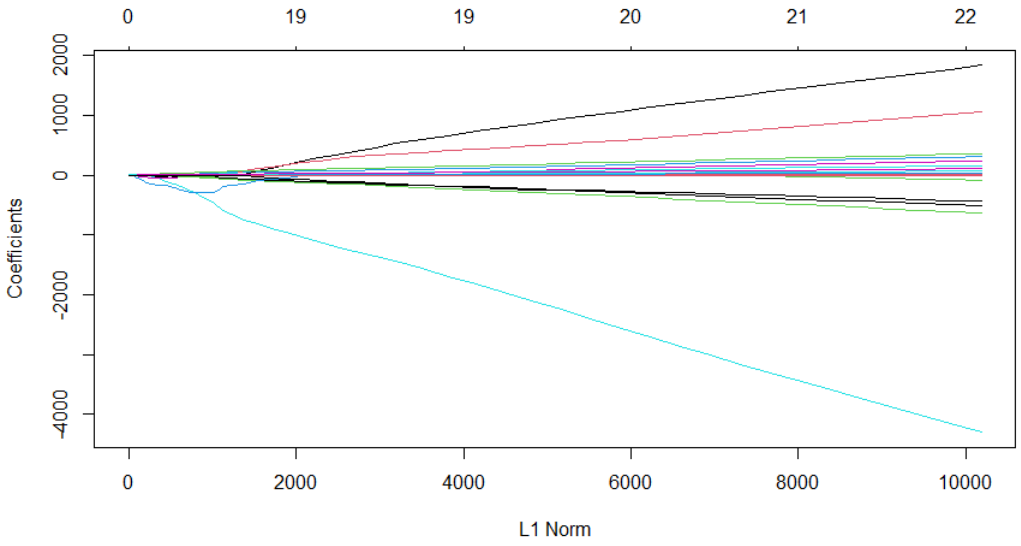

Figure 10: glmnet 包作用于 wdbc 数据集变量选择结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> Active.Index
 [1]  7 11 16 20 21 22 25 27 28 29
> Active.coefficients
 [1]   4.4734739   4.9892731 -23.8061616 -76.4051121   0.7063797   0.2061126  20.9846519
 [8]   2.0790990  20.2757346   6.7007695
```

Figure 11: glmnet 包作用于 wdbc 数据集变量选择结果

我们可以很明显地发现，用 glmnet 包进行特征选择，选择出来的特征数相较 Lars 包进行特征选择选择出来的特征数要少很多，由 18 个减少为 10 个。

比较两张 Lasso 的算法变量选择图，不难看出，在用 glmnet 包进行特征选择时，当范数较大时的变量范围会划分得更开一些。这在一定程度上说明了此时的特征选择可以更易于将重要变量与不重要变量划分开来。

对于这 10 个特征，我采用 glm 函数建立 Logistic 回归模型，得到的模型结果如下图 12 所示：

可以看到得到模型进行了 10 次迭代，得到的模型中进行系数检验，截距项，与变量 13、23、24 的系数检验结果是显著的。这说明这些变量在模型中所起到的影响因素要大一些。

对于得到的 logistic 回归模型结果，作用于训练集本身得到的预测结果与真实结果进行比较，得到的正确率如下图 13 所示：

```
> summary(glmfit_train)

Call:
glm(formula = trainy ~ newtrainx, family = binomial(link = "logit"),
    data = as.data.frame(cbind(newtrainx, trainy)))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.55409  -0.02267  -0.00074   0.00001   2.99483

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)   -52.9556    13.2928  -3.984 6.78e-05 ***
newtrainxV9    12.2132    26.6821   0.458 0.647146
newtrainxV13   21.8131     7.3789   2.956 0.003115 **
newtrainxV18  -95.7244    70.0133  -1.367 0.171553
newtrainxV22 -660.1240   709.3245  -0.931 0.352041
newtrainxV23    1.1135     0.4239   2.627 0.008621 **
newtrainxV24    0.4489     0.1200   3.742 0.000183 ***
newtrainxV27   33.0580    35.0018   0.944 0.344932
newtrainxV29    9.5113     9.0085   1.056 0.291055
newtrainxV30   35.3854    27.7964   1.273 0.203011
newtrainxV31   23.3437    14.4068   1.620 0.105163
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 508.581  on 391  degrees of freedom
Residual deviance:  35.488  on 381  degrees of freedom
AIC: 57.488

Number of Fisher Scoring iterations: 10
```

Figure 12: glmnet 包作用于 wdbc 训练数据集选择变量拟合模型结果

```
> count_right
[1] 386
> rate
[1] 0.9846939
```

Figure 13: glmnet 包作用于 wdbc 训练数据集选择变量拟合模型正确率

可以看到此时的 392 个变量中分类正确 386 个，正确率达到了 98.5%. 这说明用 glmnet 包对训练集训练的效果是较为可观的。

用之前训练所选择出的 10 个变量作用在测试集上，对于 177 个测试集进行分类实验，得到的分类正确率如下图所示：

```
> count_right
[1] 177
> rate
[1] 1
```

Figure 14: glmnet 包作用于 wdbc 测试数据集选择变量拟合模型正确率

可以看到在测试集上用这 10 个变量建立模型，并代入测试集数据，对于全部的 177 个样本的分类完全正确，没有任何差错。这说明用 glmnet 包的 Lasso 方法进行特征选择的效果十分优秀。

3.1.2 elastic net 弹性网算法

接下来用 elastic net 弹性网的算法进行实验。在这里我取定的弹性网中的系数 α=0.5，即对于 L2 惩罚项与 L1 惩罚项各取一半的占比进行实验。此时的约束优化目标函数为

$$\hat{\beta}^{enet} = \underset{\beta}{argmin}\frac{1}{2}||y - X\beta||^2 + 0.5\sum_{j=1}^{p}|\beta_j| + 0.5\sum_{j=1}^{p}\beta_j^2 \tag{10}$$

此处我还是使用 glmnet 包进行实验。此处的实验代码与之前的相类似。且在已经确定了 L2 惩罚项与 L1 惩罚项各取一半的占比进行实验的实验方案，故此时不用进行交叉验证确定 λ 的操作。此时进行特征选择得到的特征选择图像结果如下所示：

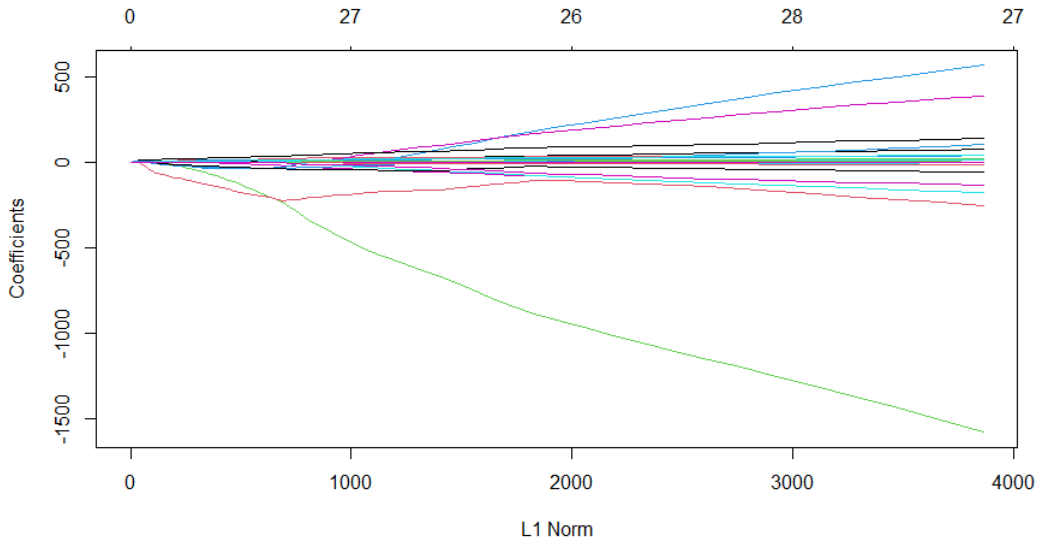

Figure 15: glmnet 包作用于 wdbc 数据集弹性网变量选择结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> Active.Index
 [1]  1  3  4  7  8  9 10 11 13 14 15 16 19 20 21 22 23 24 25 27 28 29
> Active.coefficients
 [1]  4.415228e-02  2.379451e-03  6.529421e-04  5.951388e+00  1.435584e+01  1.177231e-01
 [7] -3.186542e+01  3.636222e+00  2.440087e-01  1.314520e-02 -6.053015e+01 -2.779117e+01
[13] -1.970403e+01 -1.291563e+02  1.873737e-01  2.047867e-01  2.187354e-02  1.447597e-03
[19]  2.664479e+01  2.777962e+00  1.265059e+01  1.007784e+01
```

Figure 16: glmnet 包作用于 wdbc 数据集弹性网变量选择结果

从这里，可以很明显地看出，与之前 glmnet 包对 Lasso 算法进行实验的结果产生了相当大的变化。用弹性网算法得到的重要变量个数要明显多于 Lasso 算法，此时的特征数从 10 个增加到了 22 个。

给出对于这 22 个变量建立 logistic 回归模型的模型结果如下所示：

```
> summary(glmfit_train)

Call:
glm(formula = trainy ~ newtrainx, family = binomial(link = "logit"),
    data = as.data.frame(cbind(newtrainx, trainy)))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.84071  -0.00590  -0.00014   0.00000   2.28655

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)   4.074e+01  6.366e+01   0.640   0.5222
newtrainxV3  -6.516e+00  2.268e+01  -0.287   0.7739
newtrainxV5  -1.134e+00  1.746e+00  -0.649   0.5162
newtrainxV6   1.455e-01  2.386e-01   0.610   0.5422
newtrainxV9   2.060e+01  9.079e+01   0.227   0.8205
newtrainxV10  1.686e+02  1.832e+02   0.921   0.3572
newtrainxV11 -9.192e+00  7.225e+01  -0.127   0.8988
newtrainxV12 -8.895e+01  4.399e+02  -0.202   0.8398
newtrainxV13 -2.066e+01  5.756e+01  -0.359   0.7197
newtrainxV15 -6.774e-01  6.272e+00  -0.108   0.9140
newtrainxV16  7.284e-01  8.859e-01   0.822   0.4110
newtrainxV17 -7.341e+02  1.148e+03  -0.639   0.5226
newtrainxV18 -4.752e+01  1.657e+02  -0.287   0.7743
newtrainxV21 -4.343e+02  4.976e+02  -0.873   0.3827
newtrainxV22 -6.037e+02  1.714e+03  -0.352   0.7247
newtrainxV23  8.323e-01  1.382e+01   0.060   0.9520
newtrainxV24  6.380e-01  2.916e-01   2.188   0.0287 *
newtrainxV25  5.914e-02  9.682e-01   0.061   0.9513
newtrainxV26 -8.566e-03  1.610e-01  -0.053   0.9576
newtrainxV27  6.756e+01  9.778e+01   0.691   0.4896
newtrainxV29  1.818e+00  2.497e+01   0.073   0.9420
newtrainxV30  4.108e+01  6.039e+01   0.680   0.4964
newtrainxV31  6.883e+01  6.723e+01   1.024   0.3059
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 508.58  on 391  degrees of freedom
Residual deviance:  26.43  on 369  degrees of freedom
AIC: 72.43

Number of Fisher Scoring iterations: 13
```

Figure 17: glmnet 包作用于 wdbc 数据集弹性网变量选择模型建立

可以看到，此时的模型结果中仅有变量 V23 的系数检验是显著的，而且此时的模型 AIC 值要明显大于刚才图 12 的结果。这说明此时的模型在约简程度与精度两方面都不如刚才的模型。

对于训练数据集进行预测并且与真实结果进行比对，得到的正确率结果如下所示：

```
> count_right
[1] 387
> rate
[1] 0.9872449
```

Figure 18: glmnet 包作用于 wdbc 训练数据集弹性网正确率

在 392 个样本量中正确了 387 个，正确率达到了 98.7%，要略高于刚才 Lasso 算法

特征选择的结果。而将选择出的这 22 个特征作用到测试集上进行实验，得到的正确率结果如下所示：

```
> count_right
[1] 177
> rate
[1] 1
```

Figure 19: glmnet 包作用于 wdbc 测试数据集弹性网正确率

结果与刚才的 Lasso 算法一样，对于测试集的预测都是完全正确的。

综合看来，对于 wdbc 数据集，尽管采用弹性网的训练集正确率要稍好一些，但是采用弹性网算法需要选出 22 个特征进行模型建立，而用一般的 Lasso 仅需要 10 个特征即可。因此，综合考虑模型的约简程度，对于乳腺癌诊断数据集，采用一般 Lasso 算法进行变量选择是更好的方法。

3.1.3 自适应 Lasso

自适应 Lasso 在 R 语言中可以通过 msgps 包完成。msgps 的特点在于对于不同的优化标准都可以进行结果的预测，例如 C_p 统计量，AIC，GCV 与 BIC 等等。

我们已经知道自适应 Lasso 估计的优化目标函数为

$$\hat{\beta}^{ALASSO} = \underset{\beta}{argmin}\frac{1}{2}||y - X\beta||^2 + \lambda\sum_{j=1}^{p}\omega_j|\beta_j| \tag{11}$$

而在该式中，有 $\lambda \geq 0$ 且 $\omega_j = 1/(\hat{\beta}_j)^{\gamma}$. 其中的 γ 是一个调整参数。而 msgps 包可以协助完成调整参数 γ 的选择。

加载程序包，可以在模型建立时有效计算出 γ 的值如下所示：

```
tuning.max: 46.72
```

Figure 20: msgps 包作用于 wdbc 测试数据集得到 γ 值

可以看到模型中调整参数 γ=46.72. 在此基础上进行变量选择得到的结果如下图 21 所示：

从图上可以看出此时的变量选择结果主要是在压缩程度达到 0.75 左右时进行特征选择。此时的大多数变量都已经得到了压缩，会有少数变量的系数仍旧较大，在某种意义上可以达到“稀疏”的效果。

此时基于 4 个完全不同的准则 C_p 准则，AIC,GCV 与 BIC 准则得到的在特征选择后的模型对应调整参数如下图 22 所示：

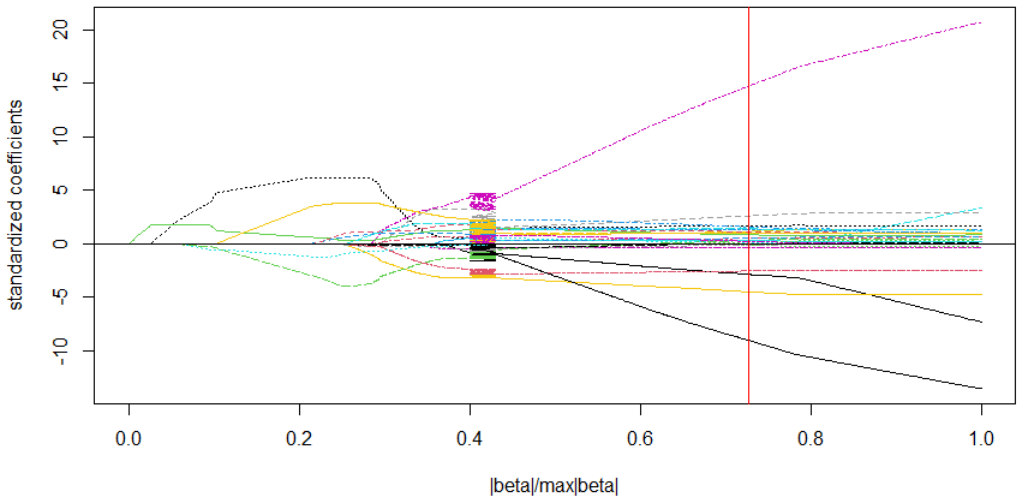

Figure 21: msgps 包作用于 wdbc 测试数据集变量选择

```
ms.tuning:
     Cp AICC GCV   BIC
[1,] 37   37  37 37.43

ms.df:
        Cp  AICC   GCV   BIC
[1,] 23.63 23.01 23.63 22.57
```

Figure 22: msgps 包作用于 wdbc 测试数据集变量选择调整参数

事实上基于不同准则的参数值大相径庭。我们可以基于 msgps 包的特点，基于这 4 种完全不同的准则，对于建立好的模型中的因变量进行预测与分类。此时基于这 4 种准则，将因变量预测结果与真实结果比较得到的正确率分别如下所示[9]：

```
> rate1
[1] 0.9668367
> rate2
[1] 0.9668367
> rate3
[1] 0.9668367
> rate4
[1] 0.9642857
```

Figure 23: msgps 包作用于 wdbc 训练数据集基于不同准则预测正确率

发现 C_p 准则，AIC 与 GCV 准则得到的分类结果均为 96.7%，而 BIC 准则对应得到的分类准确率略差一些为 96.4%.

我们再将训练集已经得到得到变量选择结果作用于测试数据集，进行分类结果的正确率统计，得到的结果如下图所示：

[9]此时的 rate1-rate4 分别对应 C_p 准则，AIC,GCV 与 BIC 准则

```
> rate1
[1] 0.9322034
> rate2
[1] 0.9322034
> rate3
[1] 0.9322034
> rate4
[1] 0.9265537
```

Figure 24: msgps 包作用于 wdbc 测试数据集基于不同准则预测正确率

可以看到 C_p 准则，AIC 与 GCV 准则得到的分类结果均为 93.2%，而 BIC 准则对应得到的分类准确率略差一些为 92.7%.

3.1.4 SCAD 算法

对于 wdbc 数据集，我最后采用 SCAD 惩罚项进行实验。SCAD 算法在 R 语言中的实现需要用到 ncvreg 扩展包。ncvreg 扩展包的功能也十分强大，除了 SCAD 惩罚项，对于 Lasso 惩罚项与弹性网方法同样也可以加以实现，在此处不做过多赘述。

利用 ncvreg 扩展包中的 CV.fit 函数可以通过交叉验证的方法选择出交叉验证 MSE 值最小的 λ 值。此时得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

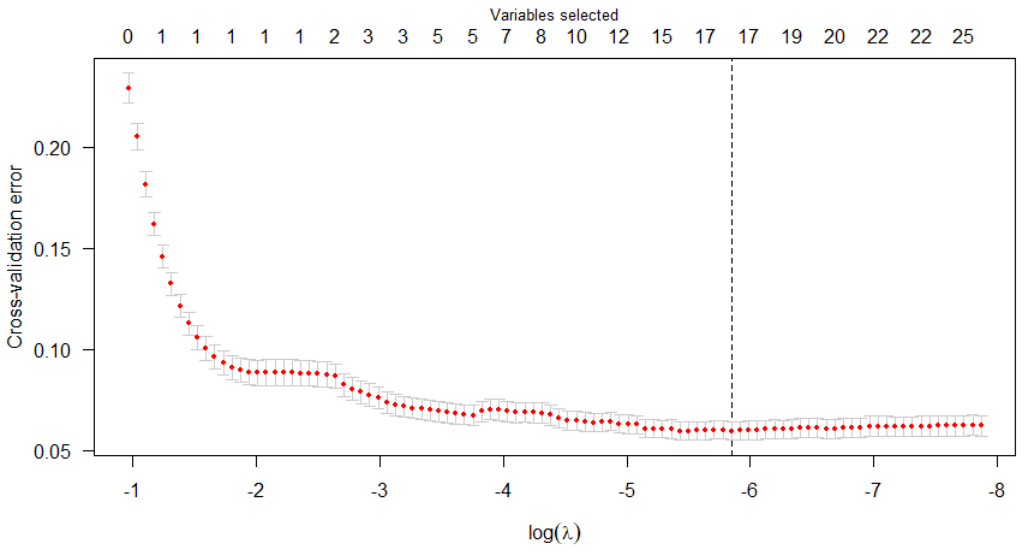

Figure 25: ncvreg 包作用于 wdbc 数据集 CV 方法选择 λ

具体选择的 λ 数值结果为:

```
> cv.fit$lambda.min
[1] 0.002884184
```

Figure 26: ncvreg 包作用于 wdbc 数据集 CV 方法选择 λ

后续我选择超参数为 0.0029 进行实验。此时得到的变量选择图像如下所示：

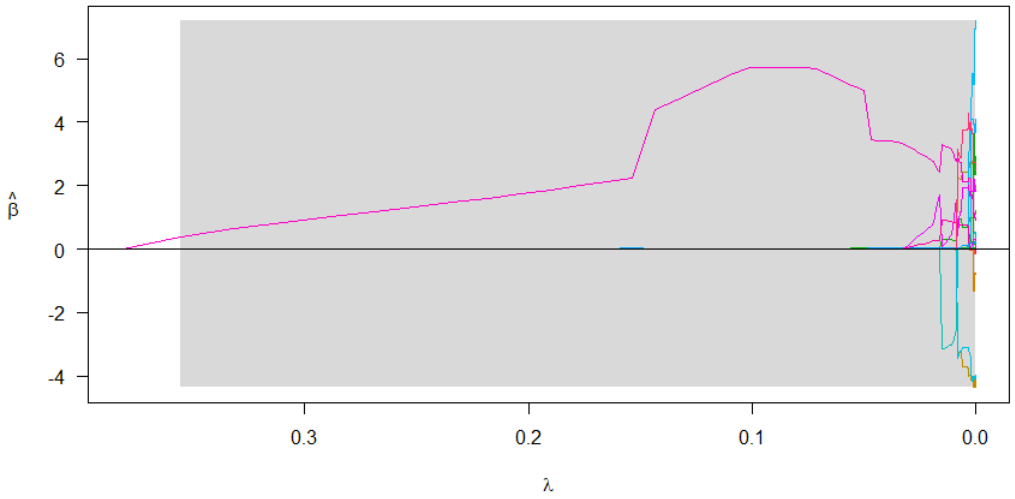

Figure 27: ncvreg 包作用于 wdbc 数据集变量选择

从图上可以发现，此时的变量选择当 λ 趋于 0 时十分复杂，随着 λ 的增大，可以大幅度减小特征数。此时进行系数压缩后的数值结果如下所示：

```
> coef(cv.fit)
  (Intercept)           V3           V4           V5           V6           V7           V8
-2.4185647167  0.0000000000  0.0000000000  0.0000000000  0.0000000000 -2.3201341241 -3.9404966477
          V9          V10          V11          V12          V13          V14          V15
 0.0000000000  3.2300694084  0.8749658649  8.5763964922  0.6011020661  0.0285479370  0.0000000000
         V16          V17          V18          V19          V20          V21          V22
-0.0018089817  0.0000000000  0.0000000000 -3.3054271888  0.0000000000  1.3803766863  6.8269363965
         V23          V24          V25          V26          V27          V28          V29
 0.1220562418  0.0094776486  0.0000000000 -0.0007655924  2.8282198295  0.0000000000  0.6989326222
         V30          V31          V32
 1.3797594024  0.4163321801  0.0000000000
```

Figure 28: ncvreg 包作用于 wdbc 数据集变量选择系数

此时的重要变量为 17 个。对训练集数据进行预测得到的正确率结果如下所示：

```
> count_right
[1] 387
> rate
[1] 0.9872449
```

Figure 29: ncvreg 包作用于 wdbc 训练数据集模型分类正确率

发现对于 392 个训练数据集，分类正确 387 个，分类正确率为 98.7%. 这个结果是比较可观的。

将特征选择的结果作用于测试数据集，得到的正确率结果如下所示：

```
> count_right
[1] 177
> rate
[1] 1
```

Figure 30: ncvreg 包作用于 wdbc 测试数据集模型分类正确率

作用于 177 个测试数据集的结果为正确率 100%. 可以看到 SCAD 惩罚项对于 wdbc 数据集特征选择完拟合得到的 Logistic 回归模型的分类效果十分好。

3.2 预测数据集 (wpbc) 相关实验

接着对预测数据集 (wpbc) 进行实验。在这里，我默认分类为 R(可治愈乳腺癌) 的患者记为 0，而分类为 N(不可治愈乳腺癌) 的患者记为 1.

3.2.1 Lasso 算法

首先用 Lasso 方法对于 wpbc 数据集进行实验。在这里，我主要把两种数据包 Lars 包与 glmnet 包都对该数据集进行了实验，对比一下实验结果。在这里，我将 wpbc 中共 198 个样本数据集以数量的 7：3 分为训练集与测试集进行实验，用训练集训练得到的特征选择结果作用于测试集，进一步建立模型观察模型预测的结果如何。

首先用 Lars 包中自带的预测函数进行实验。使用 CV.fit 函数得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

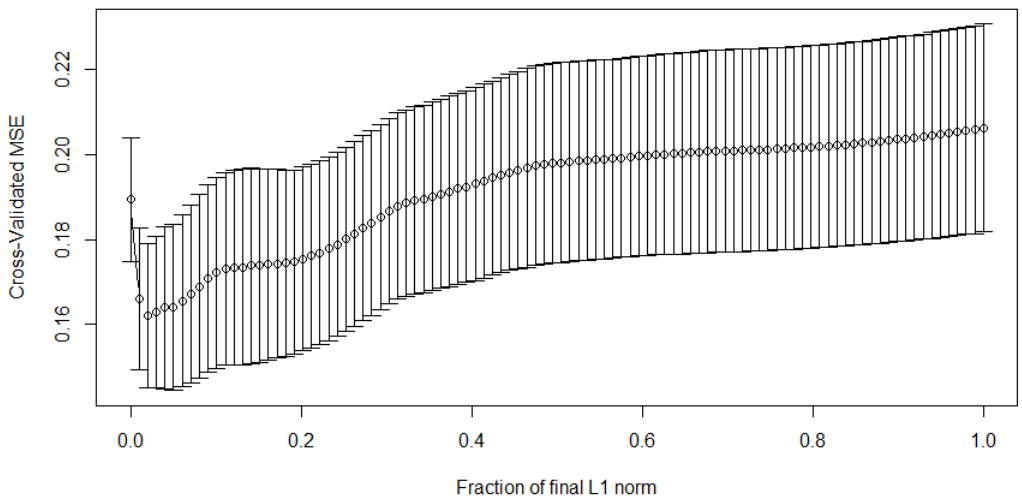

Figure 31: Lars 包作用于 wpbc 数据集 CV 结果

经过观察，可以取出交叉验证 MSE 值最小位置对应的 λ 值为:

```
> tcv$index[which.min(tcv$cv)]
[1] 0.02020202
```

Figure 32: Lars 包作用于 wpbc 数据集 CV 结果 (最小值)

可以看到当 λ=0.020 时，交叉验证 MSE 值取为最小值。

接着，采用 lars 函数对训练集数据进行变量选择，可以得到的变量选择图如下所示：

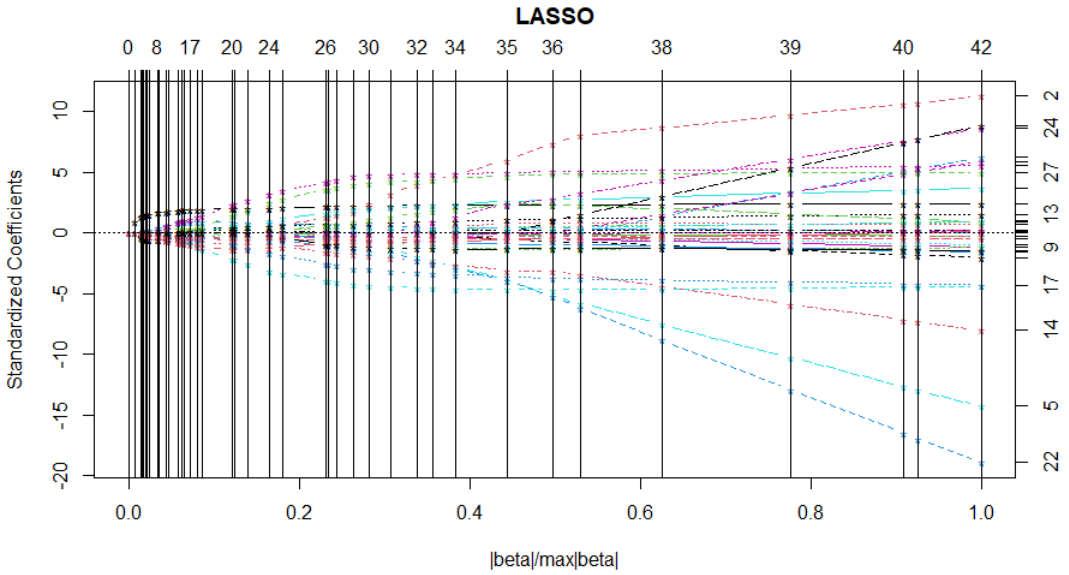

Figure 33: Lars 包作用于 wpbc 数据集变量选择结果

从这幅变量选择图上可以看出，当数据压缩程度较大时，只有几个变量蕴含在模型中。此时我们合理选择压缩的程度，可以大大约简模型的特征数。

接着，我动用 predict 函数得到选择交叉验证 MSE 值最小时的超参数 λ 情况下的用 Lasso 算法计算得到的各自变量对应的系数值，如下所示：

```
> trainfit
    V3     V4     V5     V6     V7     V8     V9    V10    V11    V12    V13    V14    V15    V16
 0.004  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.424  0.000  0.000  0.000  0.000
   V17    V18    V19    V20    V21    V22    V23    V24    V25    V26    V27    V28    V29    V30
 0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000 -0.448  0.000  0.000
   V31    V32    V33    V34
 0.000  0.000  0.000 -0.007
```

Figure 34: Lars 包作用于 wpbc 数据集系数结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> active.index
 V3 V12 V28 V34
  1  10  26  32
```

Figure 35: Lars 包作用于 wpbc 数据集变量选择结果

可以看到对于 wpbc 数据集，数据被极大程度地压缩，特征数由 32 个直接压缩为 4 个。此时的模型建立应当是相当简约的。

故我在这 4 个变量的基础上建立模型进行预测。还是用之前的预测概率结果大于 0.5 为 1，小于 0.5 为 0 的分类准则。若采用 lars 包自带的预测函数进行预测，根据得到的预测结果计算与真实结果之间的正确率如下所示：

```
> wrong_num
[1] 36
> rightrate_train
[1] 0.75
```

Figure 36: Lars 包作用于 wpbc 训练数据集变量选择正确率

可以看到 139 个训练数据集中分类出错了 36 个，正确率为 75%。

将 32 个变量压缩为 4 个变量的数据集结果作用在测试集上进行实验，得到的预测结果计算与真实结果之间的正确率如下所示：

```
> wrong_num
[1] 11
> rightrate_test
[1] 0.7962963
```

Figure 37: Lars 包作用于 wpbc 测试数据集变量选择正确率

对于 59 个测试数据集分类错误 1 个，实验正确率为 79.6%.

除了 Lars 包中自带的预测函数外，我还尝试直接使用 predict.glm 函数建立广义线性模型，进而对已经特征筛选过后的模型进行预测。此时得到的模型作用在训练集上的正确率结果如下所示：

```
> count_right
[1] 120
> rate
[1] 0.8333333
```

Figure 38: Lars 包作用于 wpbc 训练数据集变量选择正确率 (predict.glm)

可以看到此时的 139 个训练样本集中正确了 120 个，正确率为 83.3%.

而将模型作用于测试集，得到的正确率结果为：

```
> count_right
[1] 46
> rate
[1] 0.8518519
```

Figure 39: Lars 包作用于 wpbc 测试数据集变量选择正确率 (predict.glm)

测试集 59 个样本分类正确 46 个，正确率为 85.2%. 由此可见，对于 Lars 包而言，在特征选择完成后将选择好的特征建立模型进行分类结果的预测正确率要高于采用 Lars 包自带的预测函数进行预测的正确率。

下用 glmnet 包对 wpbc 数据集进行 Lasso 算法的实验。调用 glmnet 包中的 CV.fit 函数，我可以得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

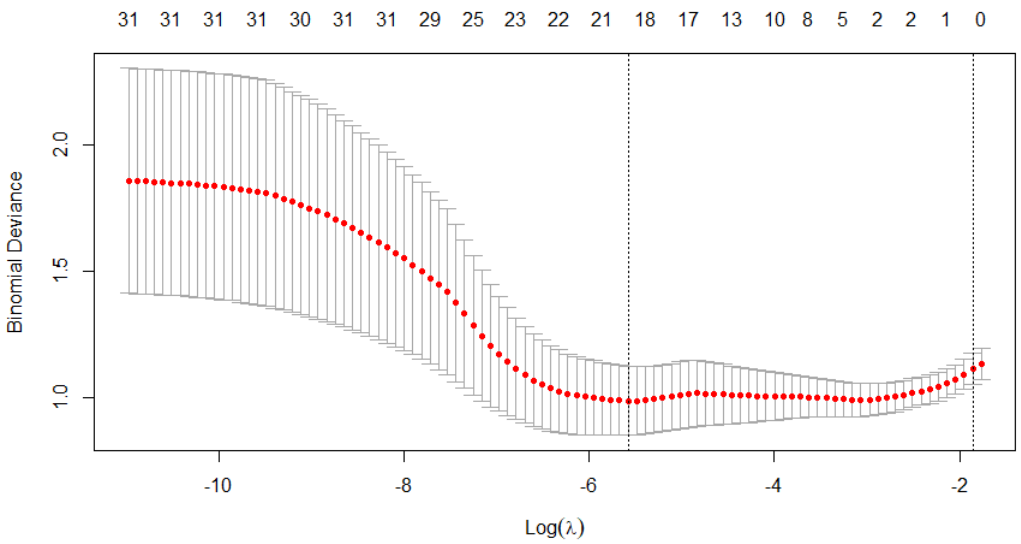

Figure 40: glmnet 包作用于 wpbc 数据集 CV 结果

经过观察，可以取出交叉验证 MSE 值最小位置对应的 λ 值为:

```
> cv.fit$lambda.min
[1] 0.003778737
> cv.fit$lambda.1se
[1] 0.1561379
```

Figure 41: glmnet 包作用于 wpbc 数据集 CV 结果

可以看到当 λ=0.0038 时，交叉验证 MSE 值取为最小值。

接着，采用 glmnet 函数对训练集数据进行变量选择，取定 λ 值为 0.0038，可以得到的变量选择图如下所示：

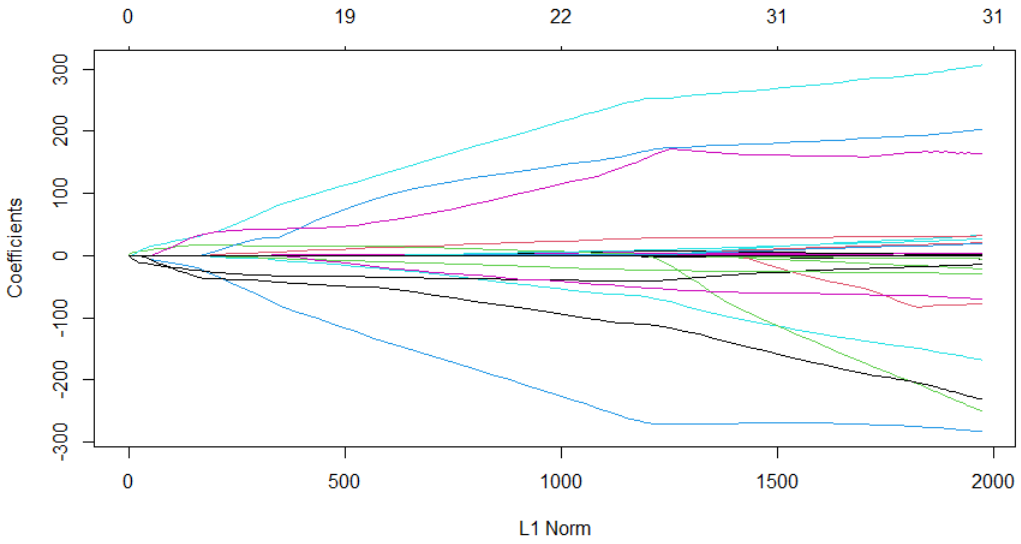

Figure 42: glmnet 包作用于 wpbc 数据集变量选择结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> Active.Index
 [1]  1  2  3  6  7 10 11 13 14 15 17 18 19 20 26 27 28 31 32
> Active.coefficients
 [1]  4.561081e-02  6.790790e-02  1.842227e-02 -1.680376e+01  2.479520e-01  1.574590e+01
 [7]  8.134589e+01  8.168911e-01 -4.353934e-01  9.186038e-04 -1.229419e+02  1.187243e+02
[13]  4.904753e+01 -5.020689e+01 -3.389441e+01  1.156605e+01 -8.554888e+00 -1.419747e+01
[19] -1.439366e-01
```

Figure 43: glmnet 包作用于 wpbc 数据集变量选择结果

我们可以很明显地发现，对于 wpbc 数据集用 glmnet 包进行特征选择，选择出来的特征数相较 Lars 包进行特征选择选择出来的特征数要多一些，由 4 个增加为 19 个。这与 wdbc 数据集的结果正好相反。

对于这 19 个特征，我采用 glm 函数建立 Logistic 回归模型，得到的模型结果如下图 44 所示：

可以看到得到模型进行了 7 次迭代，得到的模型中进行系数检验，截距项，与变量 3、16、19、20、29、30 的系数检验结果是显著的。这说明这些变量在模型中所起到的影响因素要大一些。

对于得到的 logistic 回归模型结果，作用于训练集本身得到的预测结果与真实结果进行比较，得到的正确率如下图 15 所示：

```
> summary(glmfit_train)

Call:
glm(formula = trainy ~ newtrainx, family = binomial(link = "logit"),
    data = as.data.frame(cbind(newtrainx, trainy)))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.9277  -0.0008   0.1881   0.4303   1.6074

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    -2.46741    9.84934  -0.251 0.802189
newtrainxV3     0.06532    0.01516   4.308 1.65e-05 ***
newtrainxV4     0.08795    0.25952   0.339 0.734699
newtrainxV5     0.02158    0.08376   0.258 0.796712
newtrainxV8   -81.42542   67.83647  -1.200 0.230015
newtrainxV9     9.81882   27.83665   0.353 0.724291
newtrainxV12   16.81407   19.81603   0.849 0.396155
newtrainxV13  225.47126  159.93458   1.410 0.158607
newtrainxV15    1.25669    0.98218   1.279 0.200723
newtrainxV16   -1.38910    0.64083  -2.168 0.030186 *
newtrainxV17    0.03375    0.02690   1.254 0.209711
newtrainxV19 -347.82134  105.59549  -3.294 0.000988 ***
newtrainxV20  301.65716   89.37702   3.375 0.000738 ***
newtrainxV21  148.24548  125.60014   1.180 0.237882
newtrainxV22  -77.35926   51.86796  -1.491 0.135839
newtrainxV28  -40.51634   28.54073  -1.420 0.155725
newtrainxV29   36.13087   11.30424   3.196 0.001392 **
newtrainxV30  -25.62941    8.55013  -2.998 0.002722 **
newtrainxV33  -73.29799   56.71443  -1.292 0.196217
newtrainxV34   -0.13894    0.14578  -0.953 0.340555
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 161.953  on 143  degrees of freedom
Residual deviance:  84.565  on 124  degrees of freedom
AIC: 124.57

Number of Fisher Scoring iterations: 7
```

Figure 44: glmnet 包作用于 wpbc 训练数据集选择变量拟合模型结果

```
> count_right
[1] 128
> rate
[1] 0.8888889
```

Figure 45: glmnet 包作用于 wpbc 训练数据集选择变量拟合模型正确率

可以看到此时的 139 个变量中分类正确 128 个，正确率达到了 88.9%. 这说明用 glmnet 包对训练集训练的效果是较为可观的。

用之前训练所选择出的 19 个变量作用在测试集上，对于 59 个测试集进行分类实验，得到的分类正确率如下图所示：

```
> count_right
[1] 54
> rate
[1] 1
```

Figure 46: glmnet 包作用于 wpbc 测试数据集选择变量拟合模型正确率

可以看到在测试集上用这 19 个变量建立模型，并代入测试集数据，对于全部的 54

个样本的分类完全正确，没有任何差错。这说明用 glmnet 包的 Lasso 方法进行特征选择的效果十分优秀。

3.2.2 elastic net 弹性网算法

接下来用 elastic net 弹性网的算法进行实验。在这里我仍旧取定的弹性网中的系数 α=0.5，即确定了 L2 惩罚项与 L1 惩罚项各取一半的占比进行实验的实验方案。

此处我还是使用 glmnet 包进行实验。此时进行特征选择得到的特征选择图像结果如下所示：

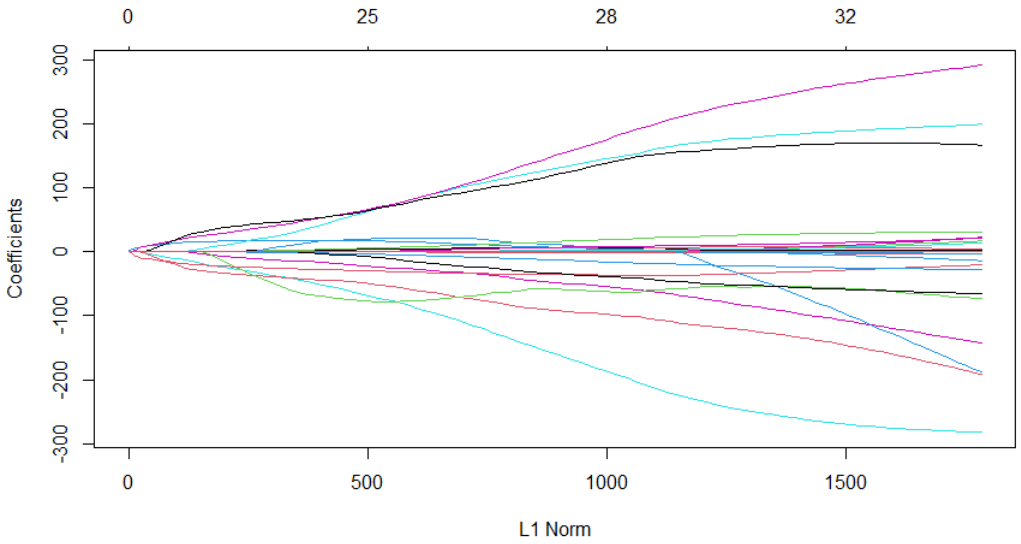

Figure 47: glmnet 包作用于 wpbc 数据集弹性网变量选择结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> Active.Index
 [1]  1  2  3  4  6  7 10 11 12 13 14 15 16 17 18 19 20 21 22 23 26 27 28 30 31 32
> Active.coefficients
 [1]   0.045299255   0.094737026   0.015831940   0.001234114 -28.657651667   4.609663851
 [7]  14.268188181  87.461927376  -0.049849156   0.826266977  -0.510010650   0.008333402
[13]  21.815936593 -91.781413818  87.172581764  82.512532580 -62.552657140 -75.232823704
[19]  -0.077726141   0.006817479 -32.269945784   8.131797494  -6.014174924   2.395046578
[25] -16.517215366  -0.140692546
```

Figure 48: glmnet 包作用于 wpbc 数据集弹性网变量选择结果

从这里，可以很明显地看出，与之前 glmnet 包对 Lasso 算法进行实验的结果产生了相当大的变化。用弹性网算法得到的重要变量个数要明显多于 Lasso 算法，此时的特征数从 Lars 包的 4 个与 glmnet 包的 19 个增加到了 26 个。

给出对于这 26 个变量建立 logistic 回归模型的模型结果如下所示：

```
> summary(glmfit_train)

Call:
glm(formula = trainy ~ newtrainx, family = binomial(link = "logit"),
    data = as.data.frame(cbind(newtrainx, trainy)))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.97289  -0.00072   0.17637   0.40553   1.47794

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    -0.47233   11.66419  -0.040  0.96770
newtrainxV3     0.06755    0.01597   4.228 2.35e-05 ***
newtrainxV4    -0.82293    3.48835  -0.236  0.81350
newtrainxV5     0.04964    0.23925   0.207  0.83563
newtrainxV6     0.16137    0.52066   0.310  0.75661
newtrainxV8   -75.49003   81.79627  -0.923  0.35606
newtrainxV9     9.34997   39.38046   0.237  0.81233
newtrainxV12   -2.34507   30.63891  -0.077  0.93899
newtrainxV13  201.36925  182.54093   1.103  0.26996
newtrainxV14    2.81084    9.59657   0.293  0.76960
newtrainxV15    1.45136    1.90720   0.761  0.44666
newtrainxV16   -1.66056    1.27320  -1.304  0.19215
newtrainxV17    0.03745    0.03210   1.167  0.24326
newtrainxV18  -64.13162  349.75981  -0.183  0.85452
newtrainxV19 -343.47314  124.05368  -2.769  0.00563 **
newtrainxV20  309.38675   99.11402   3.122  0.00180 **
newtrainxV21  186.04984  138.33068   1.345  0.17864
newtrainxV22 -143.83163  111.89809  -1.285  0.19866
newtrainxV23   17.73918  865.93124   0.020  0.98366
newtrainxV24   -0.19767    0.46235  -0.428  0.66900
newtrainxV25   -0.01515    0.21910  -0.069  0.94487
newtrainxV28  -40.47785   51.40187  -0.787  0.43100
newtrainxV29   35.18097   14.09385   2.496  0.01255 *
newtrainxV30  -27.23137    9.99649  -2.724  0.00645 **
newtrainxV32   13.86269   17.80425   0.779  0.43621
newtrainxV33  -71.44769   93.97415  -0.760  0.44708
newtrainxV34   -0.12984    0.15272  -0.850  0.39519
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 161.953  on 143  degrees of freedom
Residual deviance:  83.704  on 117  degrees of freedom
AIC: 137.7

Number of Fisher Scoring iterations: 7
```

Figure 49: glmnet 包作用于 wpbc 数据集弹性网变量选择模型建立

可以看到，此时的模型结果中变量 V3,V19,V20,V29 与 V30 的系数检验是显著的，而且此时的模型 AIC 值要明显大于刚才图 44 的结果。这说明此时的模型在约简程度与精度两方面都不如刚才的模型。

对于训练数据集进行预测并且与真实结果进行比对，得到的正确率结果如下所示：

```
> count_right
[1] 127
> rate
[1] 0.8819444
```

Figure 50: glmnet 包作用于 wpbc 训练数据集弹性网正确率

在 139 个样本量中正确了 127 个，正确率达到了 88.2%，要略高于刚才 Lasso 算法特征选择的结果。而将选择出的这 22 个特征作用到测试集上进行实验，得到的正确率结果如下所示：

```
> count_right
[1] 54
> rate
[1] 1
```

Figure 51: glmnet 包作用于 wpbc 测试数据集弹性网正确率

结果与刚才的 Lasso 算法一样，对于测试集的预测都是完全正确的。

综合看来，对于 wpbc 数据集，尽管采用弹性网的训练集正确率要稍好一些，但是采用弹性网算法需要选出 26 个特征进行模型建立，而用一般的 Lasso 仅需要 4 个特征即可。因此，综合考虑模型的约简程度，对于乳腺癌诊断数据集，仍旧采用一般 Lasso 算法进行变量选择是更好的方法。

3.2.3 自适应 Lasso

通过加载 msgps 程序包，此时基于 4 个完全不同的准则 C_p 准则，AIC,GCV 与 BIC 准则得到的在特征选择后的模型对应调整参数如下图 22 所示：

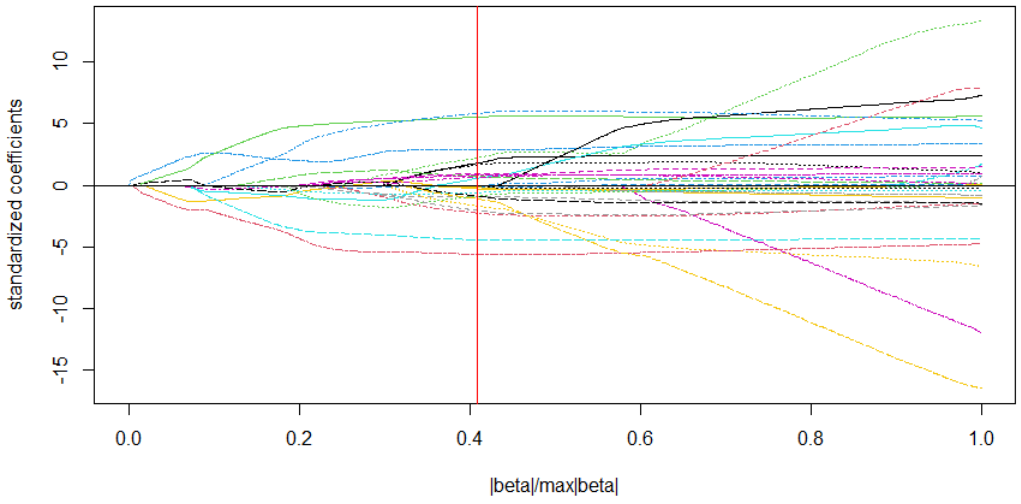

Figure 52: msgps 包作用于 wpbc 测试数据集变量选择

```
ms.tuning:
        Cp  AICC   GCV    BIC
[1,] 28.71 28.71 28.73 0.1068

ms.df:
        Cp  AICC   GCV    BIC
[1,] 22.62 22.62 22.73 0.3419
```

Figure 53: msgps 包作用于 wpbc 测试数据集变量选择调整参数

事实上基于不同准则的参数值大相径庭。我们可以基于 msgps 包的特点，基于这 4

种完全不同的准则，对于建立好的模型中的因变量进行预测与分类。此时基于这 4 种准则，将因变量预测结果与真实结果比较得到的正确率分别如下所示：

```
> rate1
[1] 0.8680556
> rate2
[1] 0.8680556
> rate3
[1] 0.8680556
> rate4
[1] 0.75
```

Figure 54: msgps 包作用于 wpbc 训练数据集基于不同准则预测正确率

发现 C_p 准则，AIC 与 GCV 准则得到的分类结果均为 86.8%，而 BIC 准则对应得到的分类准确率略差一些为 75%.

我们再将训练集已经得到得到变量选择结果作用于测试数据集，进行分类结果的正确率统计，得到的结果如下图所示：

```
> rate1
[1] 0.7407407
> rate2
[1] 0.7407407
> rate3
[1] 0.7407407
> rate4
[1] 0.7962963
```

Figure 55: msgps 包作用于 wpbc 测试数据集基于不同准则预测正确率

可以看到 C_p 准则，AIC 与 GCV 准则得到的分类结果均为 74.1%，而 BIC 准则对应得到的分类准确率略差一些为 79.6%.

3.2.4 SCAD 算法

对于 wpbc 数据集，我最后采用 SCAD 惩罚项进行实验。

利用 ncvreg 扩展包中的 CV.fit 函数可以通过交叉验证的方法选择出交叉验证 MSE 值最小的 λ 值。此时得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

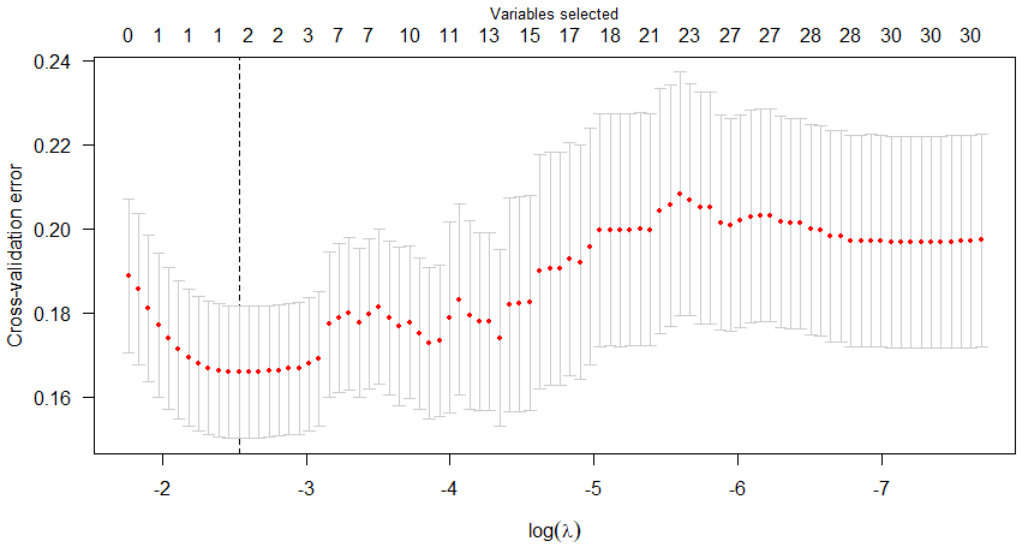

Figure 56: ncvreg 包作用于 wpbc 数据集 CV 方法选择 λ

具体选择的 λ 数值结果为:

```
> cv.fit$lambda.min
[1] 0.0795388
```

Figure 57: ncvreg 包作用于 wpbc 数据集 CV 方法选择 λ

后续我选择超参数为 0.0795 进行实验。此时得到的变量选择图像如下所示：

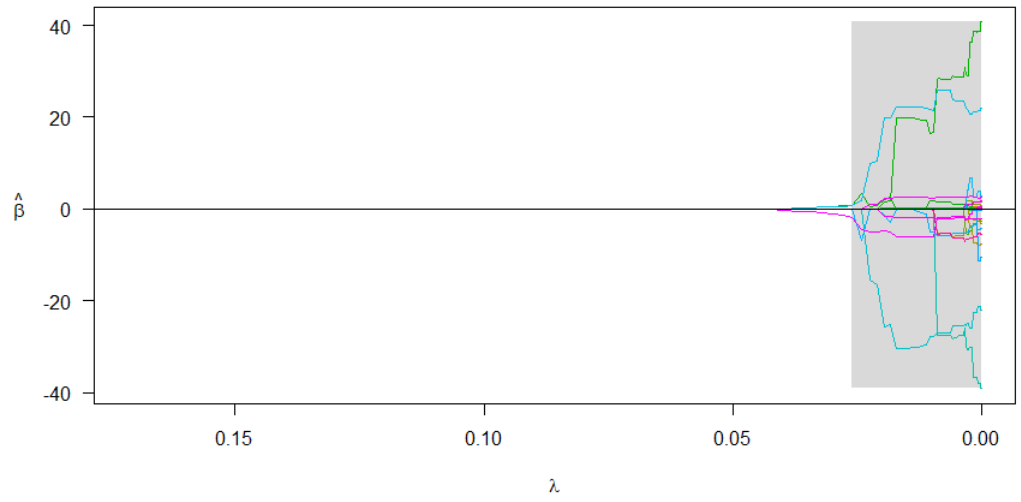

Figure 58: ncvreg 包作用于 wpbc 数据集变量选择

此时进行系数压缩后的数值结果如下所示：

```
> coef(cv.fit)
(Intercept)          V3          V4          V5          V6          V7          V8          V9
0.561762814 0.003978593 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
        V10         V11         V12         V13         V14         V15         V16         V17
0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
        V18         V19         V20         V21         V22         V23         V24         V25
0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
        V26         V27         V28         V29         V30         V31         V32         V33
0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
        V34
0.000000000
```

Figure 59: ncvreg 包作用于 wpbc 数据集变量选择系数

此时的重要变量变为仅仅一个。这是十分出人意料的。事实上从图 58 的结果也可以看出，选择 λ 值为 0.0795 时，在变量选择图上仅有唯一的一个变量出现。

```
> count_right
[1] 110
> rate
[1] 0.7638889
```

Figure 60: ncvreg 包作用于 wpbc 训练数据集模型分类正确率

发现对于 139 个训练数据集，分类正确 110 个，分类正确率为 76.4%. 这个结果是比较可观的。

将特征选择的结果作用于测试数据集，得到的正确率结果如下所示：

```
> count_right
[1] 43
> rate
[1] 0.7962963
```

Figure 61: ncvreg 包作用于 wpbc 测试数据集模型分类正确率

作用于 54 个测试数据集的结果为正确率 79.6%. 可以看到 SCAD 惩罚项对于 wdbc 数据集特征选择完拟合得到的 Logistic 回归模型的分类效果并不是特别好。当然这与模型特征被压缩到只有一个有着很大的关系，此时的模型太过于简约，反而分类效果就没有特别好。

4 相关实验二：步法分类数据集 (n<p 情形)

在这里我进行实验的第二个数据集为 PersonGaitDataSet. 该数据集主要涉及对于 48 名实验志愿者的 321 个运动参数的统计，进而对于这 48 名志愿者进行分类。原数据集将 48 名志愿者分类为 16 类，在这里为了实验的简便性，将 16 类中的第 1 到第 8 类归为“0”类，而第 9 到第 16 类归为“1”类。

值得注意的是，该数据集是一个典型的特征数要远大于样本数的数据集。这很好地符合了高维数据的特征。此时我们动用特征选择方法对该数据集进行特征选择进而进行分类，研究一下对于高维数据集，采用这些特征选择方法是否合适。

4.0.1 Lasso 算法

首先用 Lasso 方法对于 pgds 数据集进行实验。在这里，和之前一样，我将 pgds 中共 198 个样本数据集以数量的 7：3 分为训练集与测试集进行实验，用训练集训练得到的特征选择结果作用于测试集，进一步建立模型观察模型预测的结果如何。

此处主要用 glmnet 包中自带的预测函数进行实验。在这里不用 Lars 包的原因是因为 Lars 算法对于高维数据的计算复杂度过高。此时的 Lars 算法在优化过程中的步长选择过于严苛，综合考虑，采用 glmnet 包进行 Lasso 算法的实验。

使用 CV.fit 函数得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

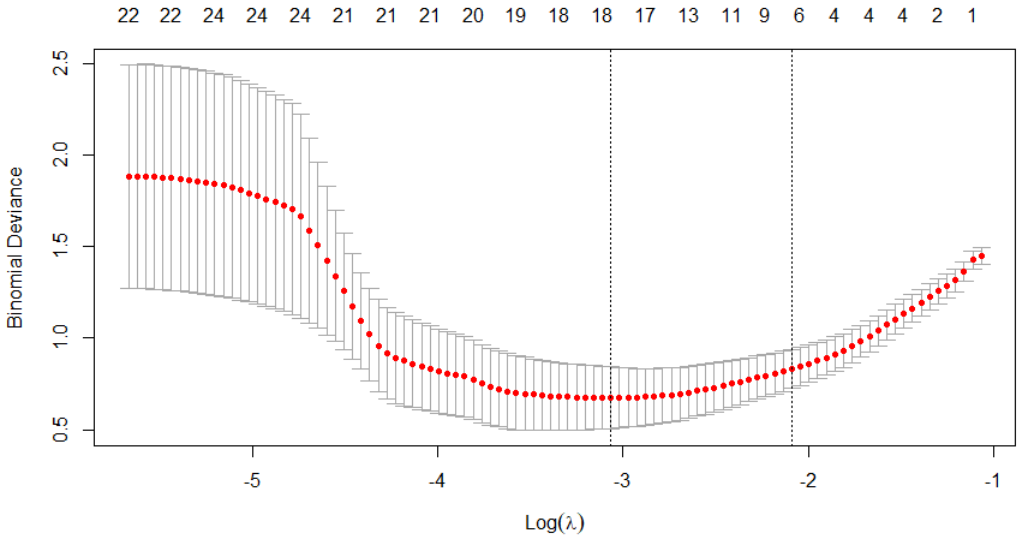

Figure 62: glmnet 包作用于 pgds 数据集 CV 结果

经过观察，可以取出交叉验证 MSE 值最小位置对应的 λ 值为:

```
> cv.fit$lambda.min
[1] 0.04665288
> cv.fit$lambda.1se
[1] 0.1239141
```

Figure 63: glmnet 包作用于 pgds 数据集 CV 结果

可以看到当 λ=0.0467 时，交叉验证 MSE 值取为最小值。

接着，采用 glmnet 函数对训练集数据进行变量选择，取定 λ 值为 0.0467，可以得到的变量选择图如下所示：

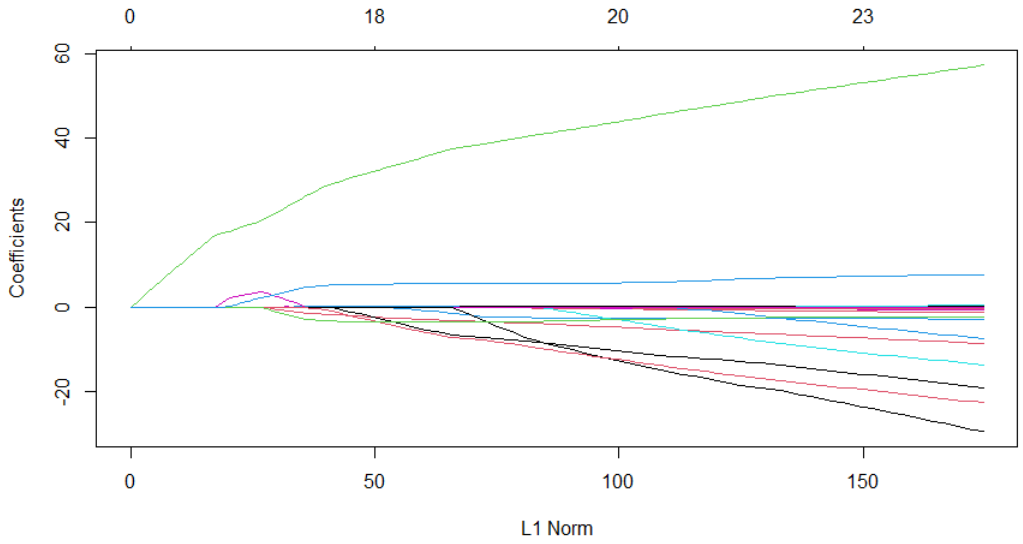

Figure 64: glmnet 包作用于 pgds 数据集变量选择结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> Active.Index
 [1]   2  23  64  82  98 129 163 181 183 189 195 214 262 288 289 313 314 317
> Active.coefficients
 [1]  0.018739749 -0.006206509 -0.009261492 -0.055946516  0.003137799 -0.144781371 -0.103343686
 [8] -0.042829150 -0.014413628 -0.018802324 -2.448826776 -0.010795162 -2.719500681 32.443537889
[15]  0.177351591 -3.612774449 -3.495776911  5.417376970
```

Figure 65: glmnet 包作用于 pgds 数据集变量选择结果

我们可以很明显地发现，对于 pgds 数据集用 glmnet 包进行特征选择的特征数为 18 个。可以看到由原来的 321 个特征压缩为 18 个，压缩的数量相当可观。

对于这 18 个特征，我采用 glm 函数建立 Logistic 回归模型，得到的模型结果如下图 44 所示：

可以看到得到模型进行了 25 次迭代. 对于得到的 logistic 回归模型结果，作用于训练集本身得到的预测结果与真实结果进行比较，得到的正确率如下图 67 所示：

```
> summary(glmfit_train)

Call:
glm(formula = trainy ~ newtrainx, family = binomial(link = "logit"),
    data = as.data.frame(cbind(newtrainx, trainy)))

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-8.416e-06  -1.588e-06  -2.110e-08   1.314e-06   7.327e-06

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)   -6.678e+03  8.400e+08       0        1
newtrainxV2    2.835e-01  3.781e+04       0        1
newtrainxV23  -1.627e-01  4.427e+03       0        1
newtrainxV64  -2.497e+02  1.395e+07       0        1
newtrainxV82   7.249e+01  8.392e+06       0        1
newtrainxV98   7.817e+01  8.425e+06       0        1
newtrainxV129 -2.598e+00  2.403e+04       0        1
newtrainxV163 -4.121e-01  5.418e+04       0        1
newtrainxV181 -1.759e+00  4.072e+04       0        1
newtrainxV183 -2.736e-01  9.200e+03       0        1
newtrainxV189  6.238e+00  1.311e+05       0        1
newtrainxV195 -8.630e+01  1.208e+06       0        1
newtrainxV214  2.592e-01  6.151e+04       0        1
newtrainxV262 -4.846e+02  1.135e+07       0        1
newtrainxV288  3.600e+02  2.788e+06       0        1
newtrainxV289 -2.847e+01  1.710e+06       0        1
newtrainxV313 -2.820e+02  5.264e+06       0        1
newtrainxV314  1.606e+01  2.384e+06       0        1
newtrainxV317  1.222e+01  2.078e+06       0        1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5.3834e+01  on 38  degrees of freedom
Residual deviance: 4.3146e-10  on 20  degrees of freedom
AIC: 38

Number of Fisher Scoring iterations: 25
```

Figure 66: glmnet 包作用于 pgdds 训练数据集选择变量拟合模型结果

```
> count_right
[1] 39
> rate
[1] 1
```

Figure 67: glmnet 包作用于 pgds 训练数据集选择变量拟合模型正确率

可以看到此时的 39 个变量中分类完全正确。这说明用 glmnet 包对训练集训练的效果相当好。

用之前训练所选择出的 18 个变量作用在测试集上，对于 9 个测试集进行分类实验，得到的分类正确率如下图所示：

```
> count_right
[1] 9
> rate
[1] 1
```

Figure 68: glmnet 包作用于 pgds 测试数据集选择变量拟合模型正确率

可以看到在测试集上用这 18 个变量建立模型，并代入测试集数据，对于全部的 9 个样本的分类完全正确，没有任何差错。这说明用 glmnet 包的 Lasso 方法进行特征选

择的效果十分优秀。

4.0.2 elastic net 弹性网算法

接下来用 elastic net 弹性网的算法进行实验此处我还是使用 glmnet 包进行实验。此时进行特征选择得到的特征选择图像结果如下所示：

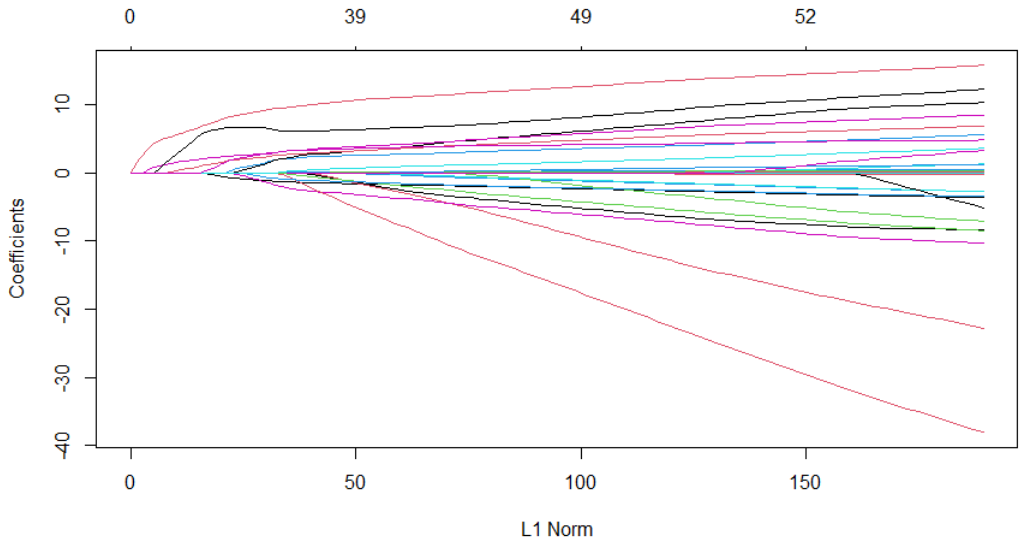

Figure 69: glmnet 包作用于 pgds 数据集弹性网变量选择结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> Active.Index
 [1]   2  23  64  69  74  82  83  84  98  99 100 112 119 123 129 132 137 162 163 164 165 170 171
[24] 181 183 189 195 199 209 214 239 240 242 243 247 262 263 266 269 280 288 289 292 303 312 313
[47] 314 316 317
> Active.coefficients
 [1]   0.032369714  -0.018228072  -1.117075823   0.009733408   0.005502434  -0.061593328
 [7]  -0.029940194  -0.021291487   0.062060130   0.033069080   0.021347052  -0.022365797
[13]  -0.105016879   0.004951659  -0.083788066   0.017915397  -0.038604068  -0.041486786
[19]  -0.053719154  -0.020974600  -0.021242382  -0.005273579  -0.005521225  -0.020097681
[25]  -0.021246021  -0.084609172  -2.235649901   0.035866045  -0.022895494  -0.060265681
[31]  -4.885316468  -8.439285911  -1.472824598   3.460551344  -0.960195749  -5.797616355
[37]   5.801752550   4.623847158   0.218116654   5.528156056  12.443059706   0.148061430
[43]   0.438913107   7.900174116 -16.279286144  -3.945191468  -2.162506587   1.504171391
[49]   4.076618598
```

Figure 70: glmnet 包作用于 pgds 数据集弹性网变量选择结果

从这里，可以很明显地看出，与之前 glmnet 包对 Lasso 算法进行实验的结果产生了相当大的变化。用弹性网算法得到的重要变量个数要明显多于 Lasso 算法，此时的特征数从 glmnet 包的 18 个增加到了 49 个。

此时鉴于在特征选择后仍需要对 49 个特征进行建模，模型形式过于复杂，因此具体模型系数及其检验结果不再给出。

对于训练数据集进行预测并且与真实结果进行比对，得到的正确率结果如下所示：

```
> count_right
[1] 39
> rate
[1] 1
```

Figure 71: glmnet 包作用于 pgds 训练数据集弹性网正确率

在 39 个样本量中完全正确。而将选择出的这 49 个特征作用到测试集上进行实验，得到的正确率结果如下所示：

```
> count_right
[1] 9
> rate
[1] 1
```

Figure 72: glmnet 包作用于 pgds 测试数据集弹性网正确率

结果与刚才的 Lasso 算法一样，对于测试集的预测都是完全正确的。

综合看来，对于 pgds 数据集，Lasso 与弹性网都可以保证最终分类结果的正确性。综合模型的约简程度考虑，在这里选择 Lasso 方法更为合适。

4.0.3 自适应 Lasso

通过加载 msgps 程序包，此时基于 4 个完全不同的准则 C_p 准则，AIC,GCV 与 BIC 准则得到的在特征选择后的模型对应调整参数如下图 22 所示：

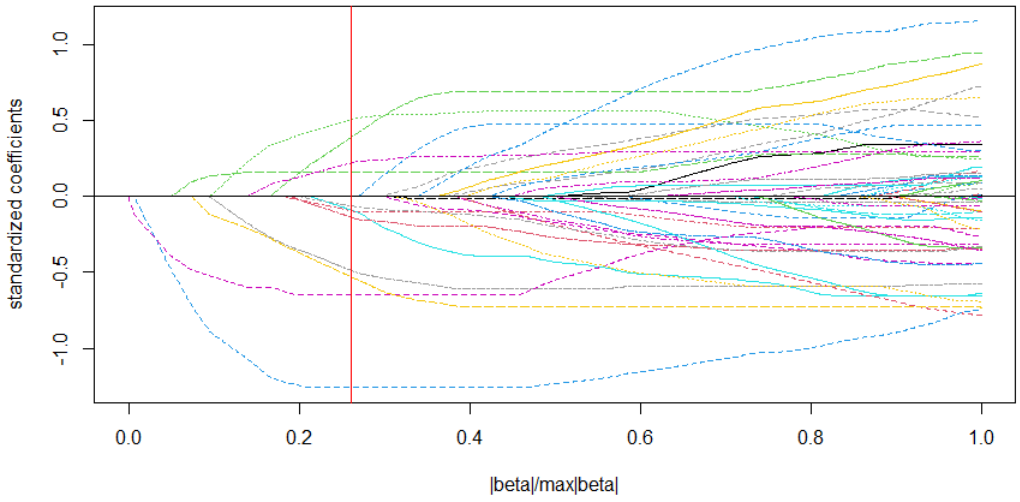

Figure 73: msgps 包作用于 pgds 测试数据集变量选择

```
ms.tuning:
        Cp AICC   GCV   BIC
[1,] 25.16  204 829.7 16.27

ms.df:
        Cp  AICC   GCV  BIC
[1,] 4.445 25.77 35.51 2.38
```

Figure 74: msgps 包作用于 pgds 测试数据集变量选择调整参数

事实上基于不同准则的参数值大相径庭。我们可以基于 msgps 包的特点，基于这 4 种完全不同的准则，对于建立好的模型中的因变量进行预测与分类。此时基于这 4 种准则，将因变量预测结果与真实结果比较得到的正确率分别如下所示：

```
> rate1
[1] 1
> rate2
[1] 1
> rate3
[1] 1
> rate4
[1] 0.974359
```

Figure 75: msgps 包作用于 pgds 训练数据集基于不同准则预测正确率

发现 C_p 准则，AIC 与 GCV 准则得到的分类结果均为 100%，而 BIC 准则对应得到的分类准确率略差一些为 97.4%.

我们再将训练集已经得到得到变量选择结果作用于测试数据集，进行分类结果的正确率统计，得到的结果如下图所示：

```
> rate1
[1] 0.6666667
> rate2
[1] 0.4444444
> rate3
[1] 0.4444444
> rate4
[1] 0.5555556
```

Figure 76: msgps 包作用于 pgds 测试数据集基于不同准则预测正确率

可以看到 C_p 准则得到的分类结果均为 66.7%，，AIC 与 GCV 的分类结果均为 44.4%，而 BIC 准则对应得到的分类准确率略差一些为 55.6%. 可以看出用自适应 Lasso 方法对于测试集的结果远比训练集的结果要差。当然这一部分原因可能是测试集中的样本个数过少所导致的。但是基于 4 种不同的准则，测试集的分类正确率都不高，这说明自适应 Lasso 方法不太适用于此时的高维数据集。

4.0.4 SCAD 算法

对于 pgds 数据集，我最后采用 SCAD 惩罚项进行实验。

利用 ncvreg 扩展包中的 CV.fit 函数可以通过交叉验证的方法选择出交叉验证 MSE 值最小的 λ 值。此时得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

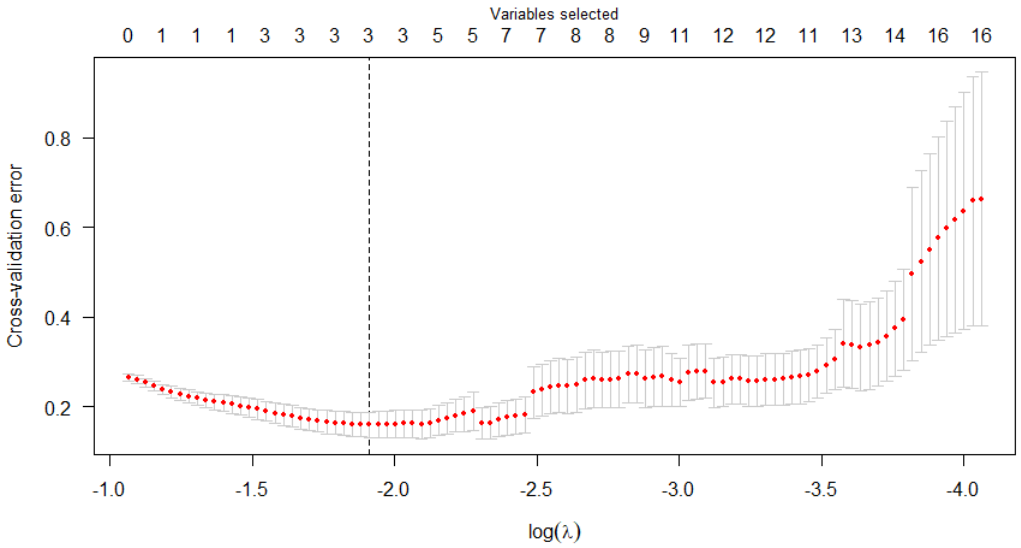

Figure 77: ncvreg 包作用于 pgds 数据集 CV 方法选择 λ

得到的具体选择的 λ 数值结果为:

```
> cv.fit$lambda.min
[1] 0.1477737
```

Figure 78: ncvreg 包作用于 pgds 数据集 CV 方法选择 λ

后续我选择超参数为 0.1478 进行实验。此时得到的变量选择图像如下所示：

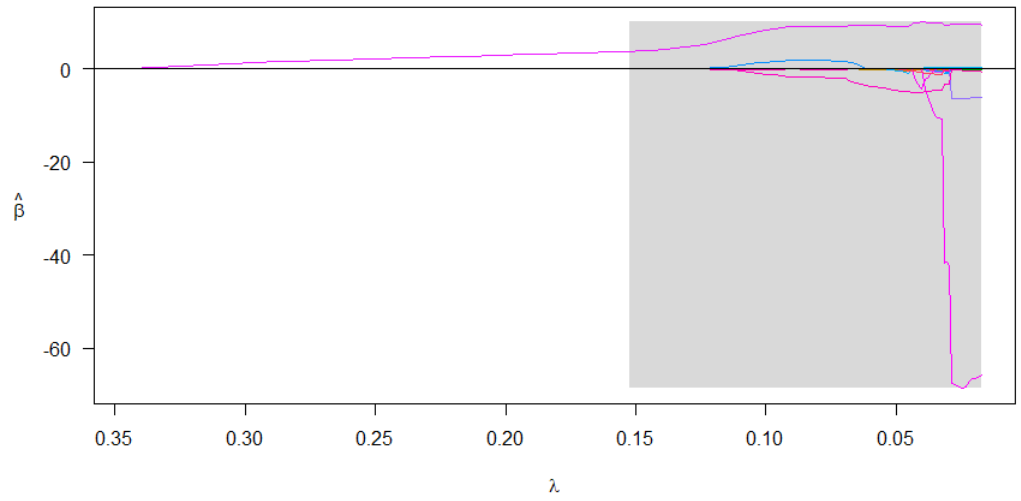

Figure 79: ncvreg 包作用于 pgds 数据集变量选择

此时进行系数压缩后的数值结果如下所示：

```
> Active.Index
V129 V189 V288
 129  189  288
> Active.coefficients
        V129        V189        V288
-0.01096689 -0.03201029  6.82182210
```

Figure 80: ncvreg 包作用于 pgds 数据集变量选择系数

此时的重要变量也仅只有 3 个。我们不难去猜测，对于具有 Oracle 性质的算法，其对于高维数据集，都可以显著压缩特征个数。在这里都是将原来的 321 个变量直接压缩为个位数甚至是一个变量，这是算法中较为神奇的特性。

```
> count_right
[1] 39
> rate
[1] 1
```

Figure 81: ncvreg 包作用于 pgds 训练数据集模型分类正确率

发现对于 39 个训练数据集完全分类正确. 这个结果是相当优秀的。
将特征选择的结果作用于测试数据集，得到的正确率结果如下所示：

```
> count_right
[1] 7
> rate
[1] 0.7777778
```

Figure 82: ncvreg 包作用于 pgds 测试数据集模型分类正确率

作用于 9 个测试数据集的结果为正确率 77.8%. 可以看到 SCAD 惩罚项对于 pgds 数据集特征选择完拟合得到的 Logistic 回归模型的分类效果中规中矩。当然这与模型特征被压缩到只有三个的情况下，模型的约简性相当高，还可以保证如此的正确率，综合考虑，SCAD 算法对于这一高维数据集的效果可圈可点。

5 相关实验三：与充分降维方法比较

最后,我对于之前在充分降维中进行实践的数据集 Algerian.forest.fires.Dataset.Update[10]进行实验。该数据集为一分类数据集，主要描述了阿尔及利亚的两片森林森林火灾的情况，记录了 2012 年 6 月 30 日到 9 月 1 日的这两片森林的温度、湿度、降水量、粉尘含量等等特征以及是否发生火灾的情况 (用 0 和 1 表示)。该数据集特征较多，适宜于充分降维后构建广义线性模型，用 logistic 回归方法判断是否发生森林火灾。

[10]该数据集来自于网站 http://archive.ics.uci.edu/ml/index.php

此前，我已经对于该数据集中的两片森林数据通过充分降维的方式进行降维，进而建立 Logistic 模型，观察了对于这两片森林是否起火灾的预测效果。在这里，我动用之前的这些特征选择算法对于该数据集进行特征选择，进而建立 Logistic 模型，观察一些对于这两片森林火灾预测的效果如何，并且将两种方法的实验结果进行一下对比，观察一下二者作用于这一数据集孰优孰劣。

鉴于两片森林的数据结构类似，且具体代码只需更改数据集及一些细节即可，故在此只展示其中森林 1 的分类结果。

5.1 特征选择方法

首先展示特征选择方法得到的结果。

5.1.1 Lasso 算法

首先用 Lasso 方法对于 forest 数据集进行实验。在这里，我主要把两种数据包 Lars 包与 glmnet 包都对该数据集进行了实验，对比一下实验结果。在这里，我将 forest 中共 122 个样本数据集以数量的 7：3 分为训练集与测试集进行实验，用训练集训练得到的特征选择结果作用于测试集，进一步建立模型观察模型预测的结果如何。

首先用 glmnet 包中自带的预测函数进行实验。使用 CV.fit 函数得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

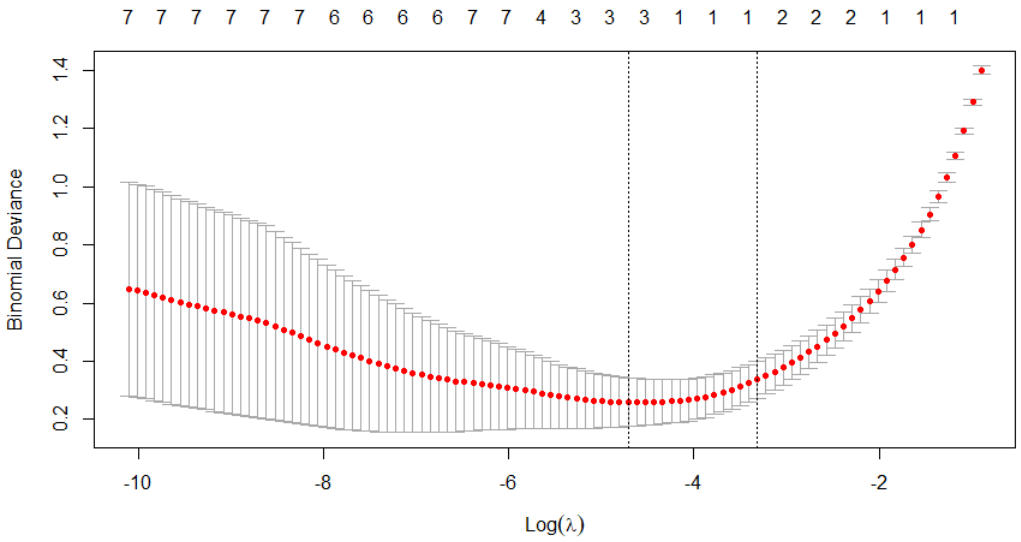

Figure 83: Lars 包作用于 forest 数据集 CV 结果

经过观察，可以取出交叉验证 MSE 值最小位置对应的 λ 值为:

```
> cv.fit$lambda.min
[1] 0.00901744
```

Figure 84: glmnet 包作用于 forest 数据集 CV 结果

可以看到当 λ=0.0090 时，交叉验证 MSE 值取为最小值。

接着，采用 glmnet 函数对训练集数据进行变量选择，取定 λ 值为 0.0090，可以得到的变量选择图如下所示：

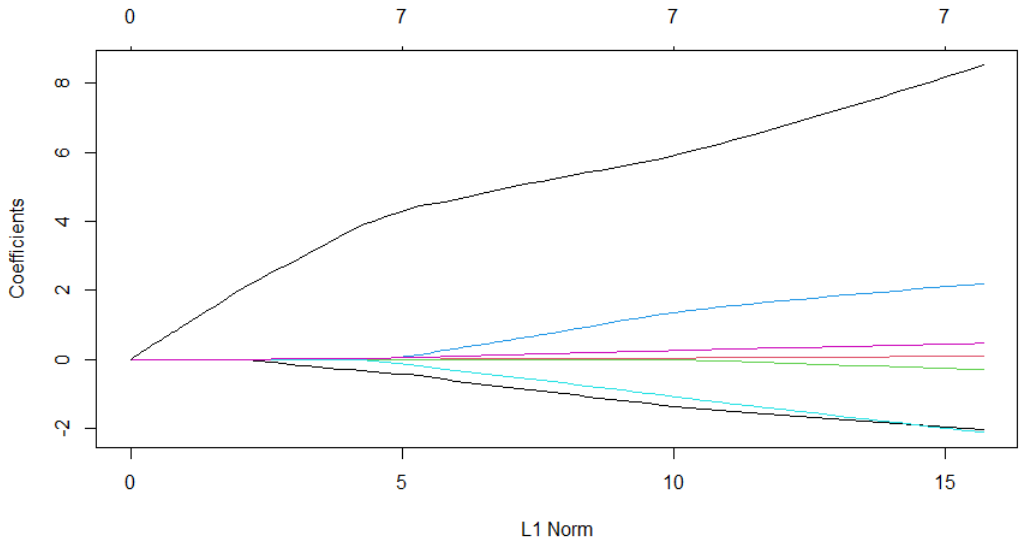

Figure 85: glmnet 包作用于 forest 数据集变量选择结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> Active.Index
[1] 1 7 8
> Active.coefficients
[1] -0.05537022  0.00224266  2.35690842
```

Figure 86: glmnet 包作用于 forest 数据集变量选择结果

我们可以很明显地发现，对于 pgds 数据集用 glmnet 包进行特征选择的特征数为 3 个。

对于这 3 个特征，我采用 glm 函数建立 Logistic 回归模型，得到的模型结果如下图 44 所示：

可以看到得到模型进行了 25 次迭代. 对于得到的 logistic 回归模型结果，作用于训练集本身得到的预测结果与真实结果进行比较，得到的正确率如下图 88 所示：

```
> summary(glmfit_train)

Call:
glm(formula = trainy ~ newtrainx, family = binomial(link = "logit"),
    data = as.data.frame(cbind(newtrainx, trainy)))

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-3.929e-04  -2.000e-08  -2.000e-08   2.000e-08   3.974e-04

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   520.719  42056.818   0.012    0.990
newtrainxV4   -66.808   5063.232  -0.013    0.989
newtrainxV10    5.079    390.076   0.013    0.990
newtrainxV11  515.143  38732.408   0.013    0.989

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1.2866e+02  on 92  degrees of freedom
Residual deviance: 4.8005e-07  on 89  degrees of freedom
AIC: 8

Number of Fisher Scoring iterations: 25
```

Figure 87: glmnet 包作用于 forest 训练数据集选择变量拟合模型结果

```
> count_right
[1] 93
> rate
[1] 1
```

Figure 88: glmnet 包作用于 forest 训练数据集选择变量拟合模型正确率

可以看到此时的 93 个变量中分类完全正确。这说明用 glmnet 包对训练集训练的效果相当好。

用之前训练所选择出的 3 个变量作用在测试集上，对于 28 个测试集进行分类实验，得到的分类正确率如下图所示：

```
> count_right
[1] 28
> rate
[1] 1
```

Figure 89: glmnet 包作用于 forest 测试数据集选择变量拟合模型正确率

可以看到在测试集上用这 3 个变量建立模型，并代入测试集数据，对于全部的 28 个样本的分类完全正确，没有任何差错。这说明用 glmnet 包的 Lasso 方法进行特征选择具有特别好的效果。

5.1.2 elastic net 弹性网算法

接下来用 elastic net 弹性网的算法进行实验此处我还是使用 glmnet 包进行实验。此时进行特征选择得到的特征选择图像结果如下所示：

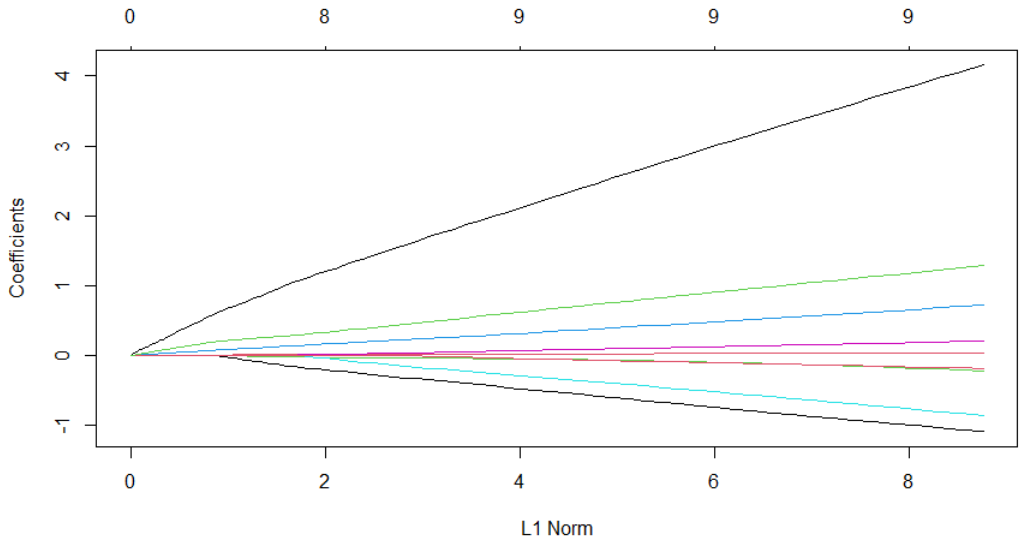

Figure 90: glmnet 包作用于 forest 数据集弹性网变量选择结果

将其中系数不为 0 的项所对应变量取出，如下所示：

```
> Active.Index
[1]  1  2  3  5  7  8 10
> Active.coefficients
[1] -0.110473373  0.010976089 -0.020441057  0.115324174  0.003351614  0.889910958  0.258885497
```

Figure 91: glmnet 包作用于 forest 数据集弹性网变量选择结果

从这里，可以很明显地看出，与之前 glmnet 包对 Lasso 算法进行实验的结果产生了相当大的变化。用弹性网算法得到的重要变量个数要明显多于 Lasso 算法，此时的特征数从 glmnet 包的 3 个增加到了 7 个。

此时鉴于在特征选择后仍需要对 7 个特征进行建模，模型形式过于复杂，因此具体模型系数及其检验结果不再给出。

对于训练数据集进行预测并且与真实结果进行比对，得到的正确率结果如下所示：

```
> count_right
[1] 93
> rate
[1] 1
```

Figure 92: glmnet 包作用于 forest 训练数据集弹性网正确率

在 93 个样本量中完全正确。而将选择出的这 7 个特征作用到测试集上进行实验，得到的正确率结果如下所示：

```
> count_right
[1] 28
> rate
[1] 1
```

Figure 93: glmnet 包作用于 forest 测试数据集弹性网正确率

结果与刚才的 Lasso 算法一样，对于测试集的预测都是完全正确的。

综合看来，对于 forest 数据集，Lasso 与弹性网都可以保证最终分类结果的正确性。综合模型的约简程度考虑，在这里选择 Lasso 方法更为合适。

5.1.3 自适应 Lasso

通过加载 msgps 程序包，此时基于 4 个完全不同的准则 C_p 准则，AIC,GCV 与 BIC 准则得到的在特征选择后的模型对应调整参数如下图 22 所示：

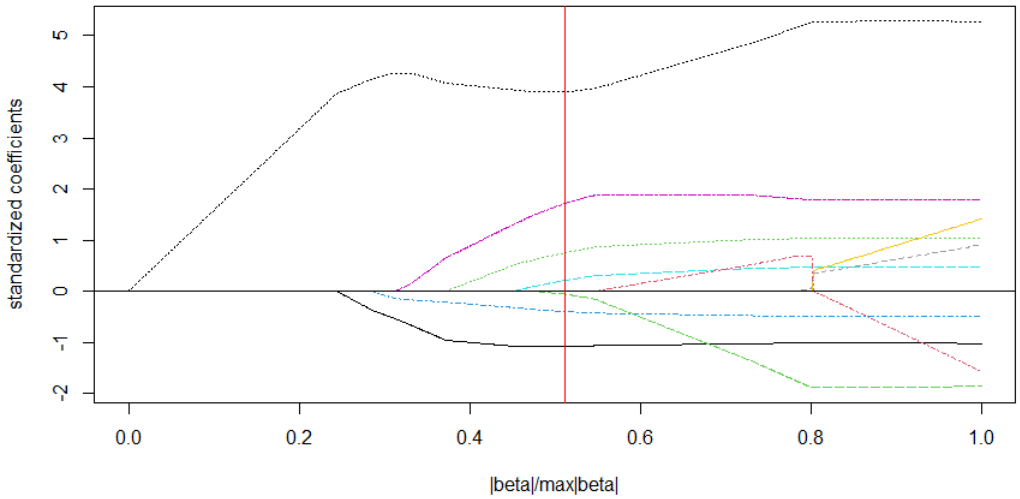

Figure 94: msgps 包作用于 forest 测试数据集变量选择

```
ms.tuning:
        Cp  AICC   GCV   BIC
[1,] 4.855 4.855 4.866 3.591

ms.df:
        Cp  AICC   GCV   BIC
[1,] 5.275 5.275 5.284 3.999
```

Figure 95: msgps 包作用于 forest 测试数据集变量选择调整参数

事实上基于不同准则的参数值大相径庭。我们可以基于 msgps 包的特点，基于这 4 种完全不同的准则，对于建立好的模型中的因变量进行预测与分类。此时基于这 4 种准则，将因变量预测结果与真实结果比较得到的正确率分别如下所示：

```
> rate1
[1] 0.9677419
> rate2
[1] 0.9677419
> rate3
[1] 0.9677419
> rate4
[1] 0.9677419
```

Figure 96: msgps 包作用于 forest 训练数据集基于不同准则预测正确率

发现所用的 4 个准则的对训练数据集的分类正确率均为 96.77%.

我们再将训练集已经得到得到变量选择结果作用于测试数据集，进行分类结果的正确率统计，得到的结果如下图所示：

```
> rate1
[1] 0.9642857
> rate2
[1] 0.9642857
> rate3
[1] 0.9642857
> rate4
[1] 0.9642857
```

Figure 97: msgps 包作用于 forest 测试数据集基于不同准则预测正确率

可以看到 4 种准则的对测试数据集的分类正确率均为 96.43%。此时的正确率略低于训练数据集的分类正确率，这是较为符合常理的。可以看到用自适应 Lasso 方法的分类正确率在一定程度上不如之前的方法效果好。

5.1.4 SCAD 算法

对于 forest 数据集，我最后采用 SCAD 惩罚项进行实验。

利用 ncvreg 扩展包中的 CV.fit 函数可以通过交叉验证的方法选择出交叉验证 MSE 值最小的 λ 值。此时得到的交叉验证 MSE 随 λ 值变动的图像如下所示：

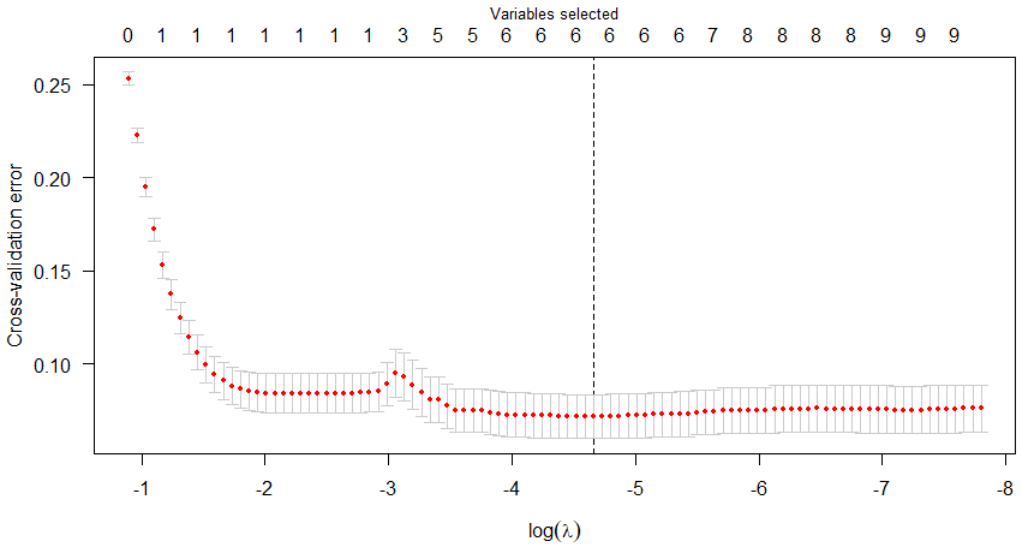

Figure 98: ncvreg 包作用于 forest 数据集 CV 方法选择 λ

得到的具体选择的 λ 数值结果为:

```
> cv.fit$lambda.min
[1] 0.009446812
```

Figure 99: ncvreg 包作用于 forest 数据集 CV 方法选择 λ

后续我选择超参数为 0.0094 进行实验。此时得到的变量选择图像如下所示：

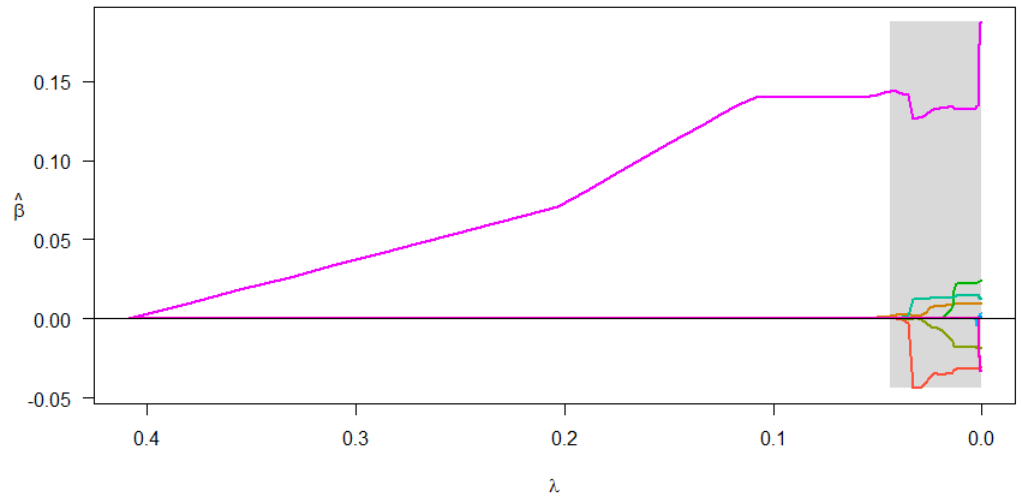

Figure 100: ncvreg 包作用于 forest 数据集变量选择

此时进行系数压缩后的数值结果如下所示：

```
> Active.Index
 V4  V5  V6  V7  V8 V11
  1   2   3   4   5   8
> Active.coefficients
          V4           V5           V6           V7           V8          V11
-0.031578703  0.009495636 -0.017777734  0.022356221  0.014640670  0.132496670
```

Figure 101: ncvreg 包作用于 forest 数据集变量选择系数

此时的重要变量为 6 个。值得注意的是，此时的数据集本身的特征数也并不是特别多，此时的特征选择并没有压缩特别多的特征。这与之前的步法分类数据集形成了鲜明的对比。之前的高维数据集是可以显著压缩特征数量的，而这里的特征压缩数量不够明显。

```
> count_right
[1] 90
> rate
[1] 0.9677419
```

Figure 102: ncvreg 包作用于 forest 训练数据集模型分类正确率

发现对于 93 个训练数据集分类正确 90 个，出现了 3 个错误。可以看到此时的分类效果不如之前。

将特征选择的结果作用于测试数据集，得到的正确率结果如下所示：

```
> count_right
[1] 28
> rate
[1] 1
```

Figure 103: ncvreg 包作用于 forest 测试数据集模型分类正确率

此时的 28 个测试数据集的分类是完全正确的。尽管对于训练集出现了一些差错，但是对于测试集而言完全正确。这从一定意义上说明了 SCAD 算法还是具有一定的优势的。

5.2 对比充分降维的效果

对于该数据集充分降维后建立 Logsitic 回归模型并计算分类正确率的实验在之前的小组作业中已经展示过了。在此处不再对实验步骤做出过多赘述，此处主要以结果展示为主。

我对该数据集采用 SIR 充分降维方法进行降维。我们构造数据估计矩 $\hat{M}$，特征分解 $\hat{M}$，进而对 $\hat{M}$ 的特征值进行序贯检验，得到的森林 1 的序贯检验 p 值结果如下图所示：

```
+   print(testresult)
+ }
[1] 0.2192153
[1] 0.5221847
[1] 0.6028308
[1] 0.9964875
[1] 0.9749894
[1] 0.999985
[1] 0.7710104
[1] 0.9806302
[1] 0.9815746
[1] NA
```

Figure 104: SIR 序贯检验结果 (森林 1)

事实上，在这里我们可以看到这两个数据集的序贯检验 p 值都没有小于检验显著水 0.05。对于这种情况，我们想要测试 SIR 降维的效果，还是将数据集降到一维进行实验。

在此处，我还是将全体样本分为训练集与测试集进行实验，以防止过拟合的情况发生，增大实验的可靠性。

将数据集降到一维后，我们对于降维后的训练集变量建立 Logistic 回归模型。对于森林 1 的 logistics 模型建立结果如下图所示：

```
> summary(fit_for)

Call:
glm(formula = y ~ xnew, family = binomial(link = "logit"), data = as.data.frame(cbind(xnew,
    y)))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.82217  -0.07254  -0.00930   0.08995   2.32588

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -17.006      4.108  -4.140 3.47e-05 ***
xnew           6.198      1.503   4.125 3.71e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 167.667  on 120  degrees of freedom
Residual deviance:  27.785  on 119  degrees of freedom
AIC: 31.785

Number of Fisher Scoring iterations: 8
```

Figure 105: SIR 充分降维后 Logistics 回归模型结果 (森林 1)

可以看到模型拟合出的结果中对于系数结果的检验都是显著的。此时的模型较为贴合将为之后的数据集。

接着我们计算降维后 logistic 回归分类的正确率，得到的验证结果如下所示：

```
> count_right
[1] 87
> rate
[1] 0.9354839
```

Figure 106: SIR 充分降维后 Logistics 回归模型预测正确率 (训练集)

可以看到对于森林 1 的 93 天数据中预测正确了 87 个，正确率达到 93.54%.

再将训练集得到的参数结果代入测试集进行数据分类，得到的正确率结果如下所示：

```
> count_right
[1] 27
> rate
[1] 0.9642857
```

Figure 107: SIR 充分降维后 Logistics 回归模型预测正确率 (测试集)

可以看到此时的 28 天测试集数据分类正确了 27 天，正确率为 96.42%. 说明此时的 SIR 充分降维方法对该数据集的分类结果不错。但是略逊色于之前能完全分类正确的几种特征选择后的分类结果。

6 总结

首先，我将之前的所有方法的特征选择数、对训练集以及测试集的分类正确率做成表格，如下所示：

Table 1: 所有方法实验结果汇总 (1)

	wdbc 正确率	wdbc 特征选择数	wpbc 正确率	wpbc 特征选择数
Lasso 训练集	98.47	10	88.89	19
Lasso 测试集	100	10	100	19
弹性网训练集	98.72	22	88.19	26
弹性网测试集	100	22	100	2
自适应 Lasso 训练集	96.68	–	86.81	–
自适应 Lasso 测试集	93.22	–	79.63	–
SCAD 训练集	98.72	17	76.39	1
SCAD 测试集	100	17	79.63	1
SIR 训练集	–	–	–	–
SIR 测试集	–	–	–	–

Table 2: 所有方法实验结果汇总 (2)

pgds 正确率	pgds 特征选择数	forest	forest 特征选择数
100	18	100	3
100	18	100	3
100	49	100	7
100	49	100	7
100	–	96.77	–
66.67	–	96.43	–
100	3	100	3
77.78	3	96.77	6
–	–	93.55	1
–	–	96.43	1

从这张表中可以看出，对于本篇报告所涉及的数据集，采用 Lasso 与弹性网进行特征选择得到的分类结果的正确率较高。而采用 SCAD 方法对于维数特别大的数据集进行特征选择可以极大程度上压缩数据集的特征数，压缩数据集的效率特别高，而且对于一些数据集的分类效果同样不错。而对于这里的 Forest 数据集，将特征选择方法与充分降维方法进行对比，发现采用特征选择方法得到的分类结果会稍好一些。充分降维的好处在于将数据集的维数压缩到特别低，方便我们通过可视化的方式选择具体模型。

事实上，不同的方法有着不同的优势。尽管在实验过程中，自适应 Lasso 方法的分类正确率并不是特别高，但是正如前文中提到的，自适应 Lasso 方法与 SCAD 方法具有 Oracle 性质。合理应用这一性质，可以在解决一些实际问题时起到事半功倍的效果，帮助我们预先选择出需要的特征。总之，合理应用特征选择方法，是处理高维数据的一个重要方法。

A 相关代码

由于不同实验主要体现在数据集不同，在此主要给出wdbc数据集的实验代码，其他的实验代码类似，故不再赘述。

Lasso作用于wdbc数据集(Lars包)

```
s<-read.table('C:\\Users\\user\\Documents\\统计实践与高维\\高维数据推断\\作业\\大报告\\dataset\\乳腺癌数据集\\wdbc.data.csv',sep = ",")
```

```
s <- s[-1,]
y <- as.numeric(s[,2])
nametri <- s[,1]
s <- as.matrix(s)
x <- as.data.frame(apply(s[,3:dim(s)[2]],2,as.numeric))
wdbc <- as.data.frame(cbind(y,x))
p <- dim(wdbc)[2]
#cor(x_std)#计算相关系数矩阵

#分为测试集与验证集探究实验结果:
set.seed(1234)
index <- sample(2,nrow(wdbc),replace = TRUE,prob = c(0.7,0.3))#
    设立种子并随机抽样，得到训练集与测试集
trainwdbc <- wdbc[index==1,]
testwdbc <- wdbc[index==2,]

##用lars包(最小角回归)计算Lasso解
require(lars)
trainy <- as.vector(trainwdbc[,1])
trainx <- as.matrix(trainwdbc[,2:p])#需要将数据集变为矩阵形式，
    否则报错无法运算
M_index_train <- which(trainy==1)
B_index_train <- which(trainy==0)
lassomod <- lars(trainx,trainy,type = "lasso")#用lasso建模
#Lasso图绘制:
plot(lassomod)

#交叉验证选择压缩程度t值:
set.seed(1234)
tcv <- cv.lars(trainx,trainy)
tcv$index[which.min(tcv$cv)]
#lasso的拟合系数结果:
trainfit <- round(predict(lassomod,s=tcv$index[which.min(tcv$cv
    )],type = "coef",mode = "fraction")$coef,3)
active.index <- which(trainfit!=0)
#用选择的变量进行预测
newtrainx <- as.matrix(trainx[,active.index])
```

```
#建立logistic回归模型
glmfit_train <- glm(trainy~newtrainx,data = as.data.frame(cbind
    (newtrainx,trainy)),family = binomial(link = 'logit'))
summary(glmfit_train)

#预测变量正确率
pred.train <- predict.glm(glmfit_train,newdata = as.data.frame(
    cbind(newtrainx,trainy)),type = 'response')
pred.train

predi <- function(pred.for){
        pred.result <- rep(0,length(pred.for))
        for(i in 1:length(pred.for)){
                if(pred.for[i]>=0.5){
                        pred.result[i] <- 1
                }
        }
        return(pred.result)
}

train.result <- predi(pred.train)

pred.diff <- trainy-train.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(train.result)
count_right
rate


predlars <- predict(lassomod,trainx,tcv$index[which.min(tcv$cv)
    ],mode = "fraction")
y_pred_train <- rep(0,length(predlars$fit))
for(i in 1:length(predlars$fit)){
        if(predlars$fit[i]>0){
                y_pred_train[i] <- 1
        }
        else{
```

```
                y_pred_train[i] <- 0
        }
}


rmse <- function(x,y) (((x-y)^2))
wrong_num <- sum(rmse(y_pred_train,trainy))
rightrate_train <- (length(trainy)-wrong_num)/length(trainy)

testy <- as.vector(testwdbc[,1])
testx <- as.matrix(testwdbc[,2:p])#需要将数据集变为矩阵形式，否
    则报错无法运算
predlars <- predict(lassomod,testx,tcv$index[which.min(tcv$cv)
    ],mode="fraction")
y_pred_test <- rep(0,length(predlars$fit))
for(i in 1:length(predlars$fit)){
        if(predlars$fit[i]>0){
                y_pred_test[i] <- 1
        }
        else{
                y_pred_test[i] <- 0
        }
}

rmse <- function(x,y) (((x-y)^2))
wrong_num <- sum(rmse(y_pred_test,testy))
rightrate_test <- (length(testy)-wrong_num)/length(testy)

newtestx <- as.matrix(testx[,active.index])
#建立logistic回归模型
glmfit_test <- glm(testy~newtestx,data = as.data.frame(cbind(
    newtestx,testy)),family = binomial(link = 'logit'))
summary(glmfit_test)


#预测变量正确率
```

```
pred.test <- predict.glm(glmfit_test,newdata = as.data.frame(
    cbind(newtestx,testy)),type = 'response')
pred.test

test.result <- predi(pred.test)


#作用于测试集正确率:
pred.diff <- testy-test.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(test.result)
count_right
rate
```

Lasso作用于wdbc数据集(glmnet包):

```
s<-read.table('C:\\Users\\user\\Documents\\统计实践与高维\\高维
    数据推断\\作业\\大报告\\dataset\\乳腺癌数据集\\wdbc.data.csv
    ',sep = ",")
s <- s[-1,]
y <- as.numeric(s[,2])
nametri <- s[,1]
s <- as.matrix(s)
x <- as.data.frame(apply(s[,3:dim(s)[2]],2,as.numeric))
wdbc <- as.data.frame(cbind(y,x))
p <- dim(wdbc)[2]
#cor(x_std)#计算相关系数矩阵

#分为测试集与验证集探究实验结果:
set.seed(1234)
index <- sample(2,nrow(wdbc),replace = TRUE,prob = c(0.7,0.3))#
    设立种子并随机抽样,得到训练集与测试集
trainwdbc <- wdbc[index==1,]
```

```
testwdbc <- wdbc[index==2,]

require(glmnet)
trainy <- as.vector(trainwdbc[,1])
trainx <- as.matrix(trainwdbc[,2:p])#需要将数据集变为矩阵形式，
   否则报错无法运算
M_index_train <- which(trainy==1)
B_index_train <- which(trainy==0)
#交叉验证选择
cv.fit<-cv.glmnet(trainx,trainy,family="binomial")
plot(cv.fit)
cv.fit$lambda.min
cv.fit$lambda.1se
trainfit<-glmnet(trainx,trainy,family="binomial")
plot(trainfit)
coefficients<-coef(trainfit,s=cv.fit$lambda.min)
coefficients <- coefficients[2:length(coefficients)]
Active.Index<-which(coefficients!=0)      #系数不为0的特征索引
Active.coefficients<-coefficients[Active.Index]   #系数不为0的
   特征系数值
#用选择的变量进行预测
newtrainx <- as.matrix(trainx[,Active.Index])
#建立logistic回归模型
glmfit_train <- glm(trainy~newtrainx,data = as.data.frame(cbind
   (newtrainx,trainy)),family = binomial(link = 'logit'))
summary(glmfit_train)

#预测变量正确率
pred.train <- predict.glm(glmfit_train,newdata = as.data.frame(
   cbind(newtrainx,trainy)),type = 'response')
pred.train

predi <- function(pred.for){
        pred.result <- rep(0,length(pred.for))
        for(i in 1:length(pred.for)){
                if(pred.for[i]>=0.5){
                        pred.result[i] <- 1
```

```
                }
        }
        return(pred.result)
}

train.result <- predi(pred.train)

pred.diff <- trainy-train.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(train.result)
count_right
rate

#作用于测试集的结果:
testy <- as.vector(testwdbc[,1])
testx <- as.matrix(testwdbc[,2:p])#需要将数据集变为矩阵形式，否
   则报错无法运算
#用选择的变量进行预测
newtestx <- as.matrix(testx[,Active.Index])
#建立logistic回归模型
glmfit_test <- glm(testy~newtestx,data = as.data.frame(cbind(
   newtestx,testy)),family = binomial(link = 'logit'))
summary(glmfit_test)

#预测变量正确率
pred.test <- predict.glm(glmfit_test,newdata = as.data.frame(
   cbind(newtestx,testy)),type = 'response')
pred.test

test.result <- predi(pred.test)

#作用于测试集正确率:
pred.diff <- testy-test.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(test.result)
count_right
rate
```

elastic net作用于wdbc数据集(glmnet):

```
s<-read.table('C:\\Users\\user\\Documents\\统计实践与高维\\高维
    数据推断\\作业\\大报告\\dataset\\乳腺癌数据集\\wdbc.data.csv
    ',sep = ",")
s <- s[-1,]
y <- as.numeric(s[,2])
nametri <- s[,1]
s <- as.matrix(s)
x <- as.data.frame(apply(s[,3:dim(s)[2]],2,as.numeric))
wdbc <- as.data.frame(cbind(y,x))
p <- dim(wdbc)[2]
#cor(x_std)#计算相关系数矩阵

#分为测试集与验证集探究实验结果:
set.seed(1234)
index <- sample(2,nrow(wdbc),replace = TRUE,prob = c(0.7,0.3))#
    设立种子并随机抽样,得到训练集与测试集
trainwdbc <- wdbc[index==1,]
testwdbc <- wdbc[index==2,]

require(glmnet)
trainy <- as.vector(trainwdbc[,1])
trainx <- as.matrix(trainwdbc[,2:p])#需要将数据集变为矩阵形式,
    否则报错无法运算
M_index_train <- which(trainy==1)
B_index_train <- which(trainy==0)
#交叉验证选择
cv.fit<-cv.glmnet(trainx,trainy,family="binomial")
plot(cv.fit)
cv.fit$lambda.min
```

```
cv.fit$lambda.1se
trainfit<-glmnet(trainx,trainy,family="binomial",alpha = 0.5)
plot(trainfit)
coefficients<-coef(trainfit,s=cv.fit$lambda.min)
coefficients <- coefficients[2:length(coefficients)]
Active.Index<-which(coefficients!=0)      #系数不为0的特征索引
Active.coefficients<-coefficients[Active.Index]    #系数不为0的
   特征系数值
#用选择的变量进行预测
newtrainx <- as.matrix(trainx[,Active.Index])
#建立logistic回归模型
glmfit_train <- glm(trainy~newtrainx,data = as.data.frame(cbind
   (newtrainx,trainy)),family = binomial(link = 'logit'))
summary(glmfit_train)

#预测变量正确率
pred.train <- predict.glm(glmfit_train,newdata = as.data.frame(
   cbind(newtrainx,trainy)),type = 'response')
pred.train

predi <- function(pred.for){
        pred.result <- rep(0,length(pred.for))
        for(i in 1:length(pred.for)){
                if(pred.for[i]>=0.5){
                        pred.result[i] <- 1
                }
        }
        return(pred.result)
}

train.result <- predi(pred.train)

pred.diff <- trainy-train.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(train.result)
count_right
rate
```

```
#作用于测试集的结果:
testy <- as.vector(testwdbc[,1])
testx <- as.matrix(testwdbc[,2:p])#需要将数据集变为矩阵形式，否
    则报错无法运算
#用选择的变量进行预测
newtestx <- as.matrix(testx[,Active.Index])
#建立logistic回归模型
glmfit_test <- glm(testy~newtestx,data = as.data.frame(cbind(
    newtestx,testy)),family = binomial(link = 'logit'))
summary(glmfit_test)

#预测变量正确率
pred.test <- predict.glm(glmfit_test,newdata = as.data.frame(
    cbind(newtestx,testy)),type = 'response')
pred.test

test.result <- predi(pred.test)

#作用于测试集正确率:
pred.diff <- testy-test.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(test.result)
count_right
rate
```

自适应Lasso作用于wdbc数据集(msgps包):

```
s<-read.table('C:\\Users\\user\\Documents\\统计实践与高维\\高维
    数据推断\\作业\\大报告\\dataset\\乳腺癌数据集\\wdbc.data.csv
    ',sep = ",")
s <- s[-1,]
```

```
y <- as.numeric(s[,2])
nametri <- s[,1]
s <- as.matrix(s)
x <- as.data.frame(apply(s[,3:dim(s)[2]],2,as.numeric))
wdbc <- as.data.frame(cbind(y,x))
p <- dim(wdbc)[2]
#cor(x_std)#计算相关系数矩阵

#分为测试集与验证集探究实验结果:
set.seed(1234)
index <- sample(2,nrow(wdbc),replace = TRUE,prob = c(0.7,0.3))#
    设立种子并随机抽样，得到训练集与测试集
trainwdbc <- wdbc[index==1,]
testwdbc <- wdbc[index==2,]

##用msgps包计算Lasso解
require(msgps)
trainy <- as.vector(trainwdbc[,1])
trainx <- as.matrix(trainwdbc[,2:p])#需要将数据集变为矩阵形式，
    否则报错无法运算
M_index_train <- which(trainy==1)
B_index_train <- which(trainy==0)

alasso <- msgps(trainx,trainy,penalty="alasso",gamma=1,lambda
    =0)
summary(alasso)
plot(alasso)
predmsgps <- predict(alasso,trainx)
head(predmsgps)#取头部几个预测值展示出来

predi <- function(pred.for){
        pred.result <- rep(0,length(pred.for))
        for(i in 1:length(pred.for)){
                if(pred.for[i]>=0.5){
                        pred.result[i] <- 1
                }
        }
```

```
        return(pred.result)
}

#用均方根误差衡量预测性能:
rmse <- function(x,y) sqrt(mean((x-y)^2))
#将预测值以0.5为划分依据转化为类别
predres1 <- predi(predmsgps[,1])
predres2 <- predi(predmsgps[,2])
predres3 <- predi(predmsgps[,3])
predres4 <- predi(predmsgps[,4])

#计算分类正确率的函数:
countright <- function(predres,trainy){
        pred.diff <- trainy-predres
        count_right <- tabulate(match(pred.diff,0))
        rate <- count_right/length(predres)
        count_right
        return(rate)
}

#计算不同准则下的分类正确率
rate1 <- countright(predres1,trainy)
rate2 <- countright(predres2,trainy)
rate3 <- countright(predres3,trainy)
rate4 <- countright(predres4,trainy)

testy <- as.vector(testwdbc[,1])
testx <- as.matrix(testwdbc[,2:p])#需要将数据集变为矩阵形式，否
   则报错无法运算
predmsgps <- predict(alasso,testx)
head(predmsgps)

#作用于测试集的运算结果:
predres1 <- predi(predmsgps[,1])
predres2 <- predi(predmsgps[,2])
predres3 <- predi(predmsgps[,3])
predres4 <- predi(predmsgps[,4])
```

```
#计算不同准则下的分类正确率
rate1 <- countright(predres1,testy)
rate2 <- countright(predres2,testy)
rate3 <- countright(predres3,testy)
rate4 <- countright(predres4,testy)
```

SCAD作用于wdbc数据集(ncvreg包):

```
s<-read.table('C:\\Users\\user\\Documents\\统计实践与高维\\高维
    数据推断\\作业\\大报告\\dataset\\乳腺癌数据集\\wdbc.data.csv
    ',sep = ",")
s <- s[-1,]
y <- as.numeric(s[,2])
nametri <- s[,1]
s <- as.matrix(s)
x <- as.data.frame(apply(s[,3:dim(s)[2]],2,as.numeric))
wdbc <- as.data.frame(cbind(y,x))
p <- dim(wdbc)[2]
#cor(x_std)#计算相关系数矩阵

#分为测试集与验证集探究实验结果:
set.seed(1234)
index <- sample(2,nrow(wdbc),replace = TRUE,prob = c(0.7,0.3))#
    设立种子并随机抽样,得到训练集与测试集
trainwdbc <- wdbc[index==1,]
testwdbc <- wdbc[index==2,]

##用ncvreg包计算Lasso解
require(ncvreg)
trainy <- as.vector(trainwdbc[,1])
```

```
trainx <- as.matrix(trainwdbc[,2:p])#需要将数据集变为矩阵形式，
    否则报错无法运算
M_index_train <- which(trainy==1)
B_index_train <- which(trainy==0)
#交叉验证选择超参lambda
cv.fit<-cv.ncvreg(trainx,trainy)
plot(cv.fit)
coef(cv.fit)
cv.fit$lambda.min

trainfit<-ncvreg(trainx,trainy,penalty = "SCAD")
plot(trainfit)
coefficients<-coef(cv.fit)
coefficients <- coefficients[2:length(coefficients)]
Active.Index<-which(coefficients!=0)      #系数不为0的特征索引
Active.coefficients<-coefficients[Active.Index]    #系数不为0的
    特征系数值
#用选择的变量进行预测
newtrainx <- as.matrix(trainx[,Active.Index])
#建立logistic回归模型
glmfit_train <- glm(trainy~newtrainx,data = as.data.frame(cbind
    (newtrainx,trainy)),family = binomial(link = 'logit'))
summary(glmfit_train)

#预测变量正确率
pred.train <- predict.glm(glmfit_train,newdata = as.data.frame(
    cbind(newtrainx,trainy)),type = 'response')
pred.train

predi <- function(pred.for){
        pred.result <- rep(0,length(pred.for))
        for(i in 1:length(pred.for)){
                if(pred.for[i]>=0.5){
                        pred.result[i] <- 1
                }
        }
        return(pred.result)
```

```
}

train.result <- predi(pred.train)

pred.diff <- trainy-train.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(train.result)
count_right
rate

#作用于测试集的结果:
testy <- as.vector(testwdbc[,1])
testx <- as.matrix(testwdbc[,2:p])#需要将数据集变为矩阵形式，否
    则报错无法运算
#用选择的变量进行预测
newtestx <- as.matrix(testx[,Active.Index])
#建立logistic回归模型
glmfit_test <- glm(testy~newtestx,data = as.data.frame(cbind(
    newtestx,testy)),family = binomial(link = 'logit'))
summary(glmfit_test)

#预测变量正确率
pred.test <- predict.glm(glmfit_test,newdata = as.data.frame(
    cbind(newtestx,testy)),type = 'response')
pred.test

test.result <- predi(pred.test)

#作用于测试集正确率:
pred.diff <- testy-test.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(test.result)
count_right
rate
```

SIR充分降维作用于forest数据集：

```
s<-read.table('C:\\Users\\user\\Documents\\统计实践与高维\\高维
    数据推断\\作业\\充分降维\\数据集\\
    Algerian_forest_fires_dataset_UPDATE2.csv',sep = ",")
path="C:\\Users\\user\\Documents\\统计实践与高维\\高维数据推断
    \\作业\\充分降维\\code"
setwd(path)
source('sequential_test_SAVE.R')
source('sequential_test_SIR.R')
source('SAVE_func.R')
source('SIR_func.R')
s <- as.matrix(s[2:122,])
s <- s[,4:dim(s)[2]]
n <- dim(s)[1]
p <- dim(s)[2]#取出数据表的维数
s <- apply(s, 2, as.numeric)
s <- as.data.frame(s)
x <- s[,1:(p-1)]
y <- s[,p]
y <- as.numeric(y)
h <- 20
#分为测试集与验证集探究实验结果：
set.seed(1234)
index <- sample(2,nrow(wpbc),replace = TRUE,prob = c(0.7,0.3))#
    设立种子并随机抽样，得到训练集与测试集
trains <- s[index==1,]
tests <- s[index==2,]
trainx <- x[index==1,]
testx <- x[index==2,]
trainy <- y[index==1]
testy <- y[index==2]

for(r in 1:(p-1)){
```

```
        testresult <- seqtestsir(trainx,trainy,h,r,"continuous
            ")
        print(testresult)
}
r <- 1
siry <- sir(trainx,trainy,h,r,"continuous")
xnew <- trainx%*%siry
plot(xnew,testy,xlab="x轴",ylab="y轴")

fit_for <- glm(trainy~xnew,data = as.data.frame(cbind(xnew,
    trainy)),family = binomial(link = 'logit'))
summary(fit_for)

pred.for <- predict.glm(fit_for,newdata = as.data.frame(cbind(
    xnew,trainy)),type = 'response')
pred.for

pred.result <- rep(0,length(pred.for))
for(i in 1:length(pred.for)){
        if(pred.for[i]>=0.5){
                pred.result[i] <- 1
        }
}
pred.diff <- trainy-pred.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(pred.for)
count_right
rate

siry <- sir(testx,testy,h,r,"continuous")
testx <- as.matrix(na.omit(testx))
xnew <- testx%*%siry

pred.for <- predict.glm(fit_for,newdata = as.data.frame(cbind(
    xnew,testy)),type = 'response')
pred.for
```

```
pred.result <- rep(0,length(pred.for))
for(i in 1:length(pred.for)){
        if(pred.for[i]>=0.5){
                pred.result[i] <- 1
        }
}
pred.diff <- testy-pred.result
count_right <- tabulate(match(pred.diff,0))
rate <- count_right/length(pred.for)
count_right
rate
```