监督学习方法

司继春

上海对外经贸大学

2025年3月

监督与无监督学习

最常见的机器学习任务可以分为两种:

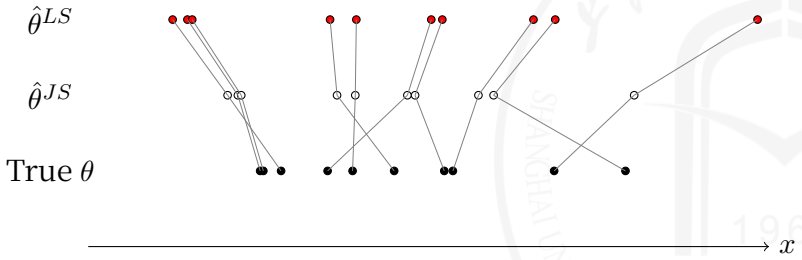
- **监督学习**（supervised learning）适用于带有**标签**（labels）的学习算法，通过算法和模型确定一些**特征**（features）的函数，对目标变量进行预测。比如我们常见的各种回归（regression）在机器学习中是典型的监督学习算法。根据标签的不同，监督学习又可以分为:
  - **分类**（classification），标签为离散变量
  - **回归**（regression），标签为连续变量
- **无监督学习**（unsupervised learning）：适用于没有标签的学习算法，通过发现数据中的结构对数据进行一些操作，比如**聚类**（clustering）、**降维**（dimension reduction）、**流形学习**（manifold learning）等。

在机器学习中，通常将已有的数据称为**训练集**（training set）或者**训练数据**（training data），机器学习的任务就是从训练集中找到特征的特定函数（模式），对新的数据进行预测。

收缩估计量

- 考虑如下问题。假设$\theta \in \mathbb{R}^K$为未知参数，观察到的样本$x \in \mathbb{R}^K$，且$x \sim \mathcal{N}(\theta, I)$，即我们有$K$个参数，同时观察到了$K$个样本，每个样本服从$x_k \sim \mathcal{N}(\theta_k, 1)$，且样本之间相互独立。
  - 如果θ_k代表每个人的真实英语水平
  - x_k为考试成绩
  - 问题：每个人的英语水平不相同，但是只进行了一次考试：每个参数只有一个样本
- 为了预测θ，我们可以使用最小二乘法，即最小化：$\hat{\theta}^{LS} = \min_{\vartheta} \sum_{k=1}^{K} (x_k - \theta_k)^2$从而得到：$\hat{\theta}_k^{LS} = x_k$，由于$\mathbb{E}\left(\hat{\theta}_k^{LS}\right) = \theta_k$，从而我们得到了无偏估计（当然，不是一致估计，why?）
  - 使用每个学生的考试成绩作为真实水平的度量
  - 是否最优?

James-Stein估计量



James-Stein估计量

- 然而，无偏的不一定是最好的。如果我们计算其均方误差，得到：

$$MSE^{LS} = \mathbb{E}\left[\sum_{k=1}^{K}(x_k - \theta_k)^2\right] = \mathbb{E}\left[(x-\theta)'(x-\theta)\right] = K$$

- 然而Stein(1956)指出，当$K \geq 3$时，最小二乘估计是inadmissible的。
- 进一步，James and Stein(1961)提出了如下估计量：

$$\hat{\theta}^{JS} = \left(1 - \frac{K-2}{\|x\|_2^2}\right)x$$

其中$\|x\|_2 = \sqrt{x'x}$为L^2范数。Stein(1961)证明，以上估计量的均方误差：$MSE^{JS} < K = MSE^{LS}$

- 实际上，如果我们观察James-Stein估计量，如果$K-2 < \|x\|_2^2$，那么$\hat{\theta}^{JS}$实际上将x向0进行了收缩（shrink）。

James-Stein估计量

- 实际上，不仅仅可以向0进行收缩，取任意的向量ϑ，估计量：

$$\hat{\theta}^{JS} = \left(1 - \frac{K-3}{\|x-\vartheta\|_2^2}\right)(x-\vartheta) + \vartheta = x + \frac{K-3}{\|x-\vartheta\|_2^2}(\vartheta - x)$$

  实现了向向量ϑ的收缩。

- 以上James-Stein估计量可以扩展到回归的情形。在线性回归中，如果我们令OLS估计量向0收缩，由于对Y的预测为$X\hat{\beta}$，于是我们就有了James-Stein估计量：

$$\hat{\beta}^{JS} = \hat{\beta}^{OLS}\left[1 - \frac{(K-2)\,\hat{\sigma}^2}{\hat{\beta}X'X\hat{\beta}}\right]$$

  与之前类似，使用该估计量可以降低对y进行估计的均方误差。

岭回归

- 线性回归通过设定线性函数形式，将预测问题转化为了K个参数的估计问题，极大的降低了未知参数的个数。然而，在现代的模型中，参数个数K越来越大，从而出现了「高维（high-dimensional）」的问题。
- **岭回归（ridge regression）**使用收缩的方法帮助提高β估计的表现。

岭回归

- 岭回归通过解：

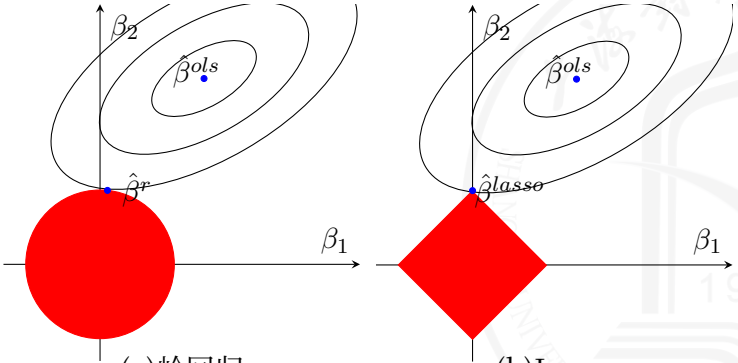
$$\hat{\beta}^r(t) = \arg\min_{\beta} \sum_{i=1}^{N} \left(y_i - x_i'\beta\right)^2$$
$$s.t. \ \|\beta\|_2^2 \leq t$$

解以上问题即最大化：

$$\mathscr{L}(\beta, \lambda) = \sum_{i=1}^{N} \left(y_i - x_i'\beta\right)^2 + \lambda\left(\|\beta\|_2^2 - t\right)$$

以上优化问题的解取决于t的取值，一旦t确定了，根据库恩塔克定理，$\lambda(t)$也就确定了。

岭回归和Lasso



(a)岭回归 (b)Lasso

岭回归

- 或者，反过来，我们可以给定λ，而不固定t的取值，从而以上最大化问题变为了：

$$\hat{\beta}^r(\lambda) = \arg\min_{\beta} \sum_{i=1}^{N} \left(y_i - x_i'\beta\right)^2 + \lambda \|\beta\|_2^2$$
$$= \arg\min_{\beta} (Y - X\beta)'(Y - X\beta) + \lambda\beta'\beta$$

以上优化问题对β的大小进行了限制，因而以上问题通常也被成为惩罚的最小二乘（penalized least squares）。

- 解以上问题，得到：

$$\hat{\beta}^r(\lambda) = \left(X'X + \lambda I\right)^{-1} X'Y$$

  - 注意到以上岭回归的计算式中，即使X不是列满秩的，矩阵$(X'X + \lambda I)$也是可逆的，也可以获得估计值。
  - 特别是在$K > N$的情况下，OLS无法获得估计，而岭回归仍然可以获得估计值。
  - 实际上，岭回归还可以看做是在特定先验下的贝叶斯线性回归。

岭回归

- 考虑X中没有常数项，且分量之间相互正交的情况。方便起见，我们记：

$$X = \begin{bmatrix} x^1 & x^2 & \cdots & x^K \end{bmatrix}$$

其中x^k为X矩阵的第k列，不失一般性我们假设$\sum_{i=1}^{N} x_i^k = 0$

- 如果X的分量之间正交，即$\left(x^k\right)' x^l = 0$，此时可以证明（习题）岭回归的系数满足：

$$\hat{\beta}_k^r = \frac{\left(x^k\right)' x^k}{\left(x^k\right)' x^k + \lambda} \hat{\beta}_k$$

其中$\left(x^k\right)' x^k = \sum_{i=1}^{N} \left(x_i^k\right)^2$为第$k$个变量的平方和。在这种情况下，岭回归即在OLS的基础上乘以一个收缩系数。

岭回归中的标准化

- 而注意到$\left(x^k\right)' x^k$的大小取决于x^k的方差以及样本量，从而对于方差不同的变量，其收缩幅度是不一样的
- 然而x^k的方差大并不代表对y的预测能力强，从而这样的收缩对于方差小的变量是不公平的。
- 为此，我们有必要在进行岭回归（包括接下来的Lasso回归等方法）时，对每个解释变量进行标准化，即使用：

$$\mathring{x}_i^k = \frac{x_i^k - \overline{x^k}}{s_{x^k}}$$

其中$\overline{x^k}$为x^k的均值，s_{x^k}为x^k的标准差。

岭回归中的标准化

- 在Stata官方命令中，lasso命令会自动帮助我们进行标准化，从而我们无需手动进行这一操作。
- 而在一些软件和包中（如Stata中非官方的Lasso命令：lasso2、cvlasso和rlasso等），标准化这一步骤默认通过“运行中（on the fly）”标准化的方式进行，即解最小化问题：

$$\arg\min_{\beta} (Y - X\beta)' (Y - X\beta) + \lambda \|\Psi\beta\|_2^2$$

其中$\Psi = \text{diag}[\psi_k]$为一个$K \times K$的对角矩阵，其对角线元素为“**惩罚载荷（penalty loadings）**”，默认使用

$$\psi_k = \sqrt{(x^k)' x^k / N}$$

，即x^k的标准差估计作为惩罚载荷

- 如此方差较大的变量会受到更多的惩罚，而方差较小的变量会受到较小的惩罚。

惩罚载荷

- 可以证明以上做法与事前进行标准化的做法是等价的。
- 事实上以上目标函数即

$$\arg\min_{\beta} (Y - X\beta)' (Y - X\beta) + \lambda \sum_{k=1}^{K} \psi_k \beta_k^2$$

在一些软件包中，ψ_k可以人为设定每个系数被惩罚的强度Stata官方的lasso命令中，由于预先进行标准化，从而设定$\psi_k = 1$，实际使用时也可以人为修改这一数值。

Lasso

- 岭回归使用了L^2范数，即$\|\beta\|_2^2$。而Lasso（Least absolute shrinkage and selection operator）估计量使用L^1范数，即：

$$\|\beta\|_1 = \sum_{k=1}^{K} |\beta_k|$$

对回归系数进行规范化，即：

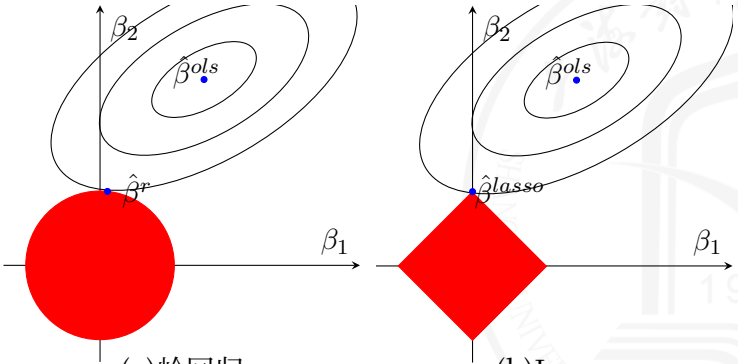
$$\hat{\beta}^{lasso}(t) = \arg\min_{\beta} \sum_{i=1}^{N} \left(y_i - x_i'\beta\right)^2$$

$$s.t. \|\beta\|_1 \leq t$$

或者等价的：

$$\hat{\beta}^{lasso}(\lambda) = \arg\min_{\beta} \sum_{i=1}^{N} \left(y_i - x_i'\beta\right)^2 + \lambda \|\beta\|_1$$

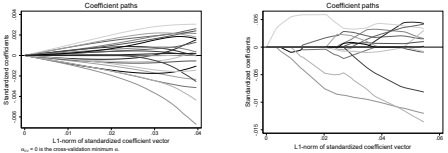
岭回归和Lasso



(a)岭回归 (b)Lasso

Lasso的变量选择

- Lasso的优化问题为**凸优化**（**convex optimization**）问题，有非常有效的全局算法。
- Lasso回归的一个优点是其同时实现了**收缩**以及**变量选择**。



(a)岭回归 (b)Lasso

稀疏一致性

- Lasso的理论性质依赖于**稀疏性（sparsity）**：
  - 记β_0为真实的β，令$S_0 = \{\beta_{0,k} \neq 0, k = 1, ..., K\}$为活跃集（active set），记$K_0 = |S_0|$为$\beta_0$中不为0的分量的个数，即稀疏指数（sparsity index）。
  - 允许变量个数$K \to \infty$，但是不能太快。
- 记$\hat{S} = \left\{\hat{\beta}_k^s \neq 0, k = 1, ..., K\right\}$，那么**选择一致性（selection consistency）**或**稀疏一致性（sparsistency）**定义即

$$P\left(\hat{S} = S_0\right) = 1$$

oracle性质

- 而如果在选择一致性的基础上，估计量额外满足

$$\sqrt{N}\left(\hat{\beta}^{s}_{S_0} - \beta_{0,S_0}\right) \overset{a}{\sim} \mathcal{N}\left(0, \Sigma\right)$$

其中Σ为估计量

$$\hat{\beta}^{oracle} = \arg\min_{\beta} \sum_{i=1}^{N} \left(y_i - x_i'\beta\right)^2$$
$$s.t.\ \beta_{S_0^c} = 0$$

的协方差矩阵，那么我们称估计量$\hat{\beta}^s$具有**oracle性质**（oracle property, Fan和Li，2001）。

- Oracle性质意味着估计量$\hat{\beta}$的统计性质要至少要与假设知道哪些系数为0的前提下所得到的估计量（$\hat{\beta}^{oracle}$）性质一样好。

Lasso的预测性质

- 从预测的角度，Lasso回归通常具有比较好的性质。
- 可以证明，随着K和N同时趋向于无穷，在λ取合适的值的情况下（$\lambda = O\left(\sigma\sqrt{\ln K/N}\right)$），以下关于平均预测误差的不等式：

$$\frac{1}{N}\left\|X\left(\hat{\beta}^{lasso}-\beta_0\right)\right\|_2^2 \leq C\cdot\sigma^2\frac{\log K}{N}K_0$$

  以很大的概率成立（见Bühlmann和van der Geer，2011），其中$\left\|X\left(\hat{\beta}^{lasso}-\beta_0\right)\right\|_2^2$即Lasso回归的残差平方和，$C$为常数。
- 以上不等式被称为oracle不等式，意味着为了达到预测的一致性，真实的模型必须是稀疏的（sparsity），至少要求$K_0 = o(N/\log K)$。

Lasso的稀疏一致性

- 然而，Lasso估计量的选择一致性需要更强的条件，其中比较重要的是不可代表性（irrepresentability），即存在$\gamma > 0$，有

$$\max_{j \in S_0^c} \left\| \left(X'_{S_0} X_{S_0} \right)^{-1} X'_{S_0} x_j \right\| \leq 1 - \gamma$$

  以上条件实际上要求S_0中的x对于非S_0中的x的相关性足够的弱，或者解释能力足够的小。
- 最理想的情况下，两类x应该正交，此时$\gamma = 0$。
- 可以证明在一定的条件和适当的λ的选取下，Lasso回归可以以一个比较高的概率达到选择一致性（Hastie，Tibshirani和Wainwright，2015）。

Lasso的oracle性质

- 然而具体到oracle性质，遗憾的是岭回归和Lasso回归通常不具有oracle性质：
  - 岭回归通常不具有选择一致性
  - 而Lasso回归虽然可能达到选择一致性，但是通常并不是一致估计量（渐进有偏）
- Oracle性质之所以吸引人主要在于其可以保证选择一致性的前提下，同时对非零参数达到了一致且渐进正态的估计，从而可以进一步对其进行统计推断，而简单的岭回归和Lasso回归却无法达到这样的效果。

选择后估计

- 基于此，一种方法是将Lasso回归作为变量选择的工具，即挑选出Lasso回归中系数不为0的变量，进而对$\hat{S}$集合中所对应的参数用OLS进行估计，这种做法也被称为**选择后估计（post-selection estimator）**，或者**后Lasso估计（post Lasso estimator）**。
- Leeb和Pötscher（2005，2008）指出，通常Lasso回归中的一致性或者收敛性都是点态收敛，而非一致收敛，这导致不同的β_0会需要不同的样本量来达到需要的统计性质，而实践中往往β_0是未知的而样本量是固定的，从而根据Lasso进行变量选择后的OLS估计的统计性质仍然是复杂的问题。
- Berk等人（2013）讨论了选择后估计的推断问题，提出可以通过拓宽置信区间的方法对参数进行合适的区间估计，而Lee等人（2016）则讨论了给定所选模型下的推断问题。

选择后估计

- 而另一方面，虽然有以上的缺点，选择后估计也的确有其优势。
- Belloni和Chernozhukov（2013）比较了模型选择后估计与Lasso估计，发现在高维稀疏模型中，即使Lasso方法可能无法达到选择一致性，不过选择后估计的在收敛速度和偏差方面表现得至少与Lasso估计一样好
- 在一些情况下其收敛速度和偏差可以严格由于单纯的Lasso估计
- 甚至在极端情况下，如果Lasso完美的选择出了正确的模型，选择后估计就成为oracle估计量。

超参数

- 在以上的岭回归和Lasso回归中，拉格朗日乘子λ，或者等价的t控制了收缩的程度
  - 一个更大的λ或者更小的t意味着更“小”的模型或者更多的向0收缩，从而偏差大但是方差小；
  - 在这里同样有偏差-方差的权衡问题
  - 应该存在一个“最优”的λ使得预测误差能够达到最小。
- 注意这里由于$\hat{y} = x'\hat{\beta}^{lasso/r}$，从而参数$\lambda$并不参与模型的预测，$\lambda$仅仅是为了“训练”在机器学习中通常将模型参数的过程称为“训练”（train）模型。
- 模型所需要的参数：$\hat{\beta}^{lasso/r}$是一个λ的函数，给定一个λ就会有一个对应的模型和参数，我们把这种训练模型之前需要设定的参数称为“**超参数**”（hyperparameter）或者“**调整参数**”（tuning parameter）。

超参数选择

- 最常用的方法是数据驱动的交叉验证法。
- 出于计算速度的要求，一般进行5-10折的交叉验证即可。
- 实际进行时会将样本随机分为S折，然后将λ从一个很大的数字到0进行格点（grid）化，针对每一个λ，使用交叉验证的方法留出1折数据作为验证集，其他的$S-1$折作为训练集，再在留出的1折数据中进行预测，最终挑出能够使得均方误差最小的λ。

交叉验证

预测香港GDP增速

我们使用Hsiao、Ching和Wan（2012）的数据，使用其他国家和地区的gdp增长率对香港回归前的gdp增长率进行预测。注意到数据集中香港回归前的数据只有18条，然而排除了中国大陆和台湾省之后的国家和地区数目有22个，从而出现了$N > K$的情况。我们可以考虑分别使用岭回归和Lasso回归来解决这一问题，并从这些可以用于相关GDP增长率的国家和地区中进行挑选。在Stata中，岭回归可以使用elasticnet命令进行，配合alpha(0)选项即得到了岭回归：

```
use "datasets/hcw.dta"
elasticnet linear HongKong Australia-Thailand if _n<=18, rseed(5)
    sel(cv) alpha(0)
```

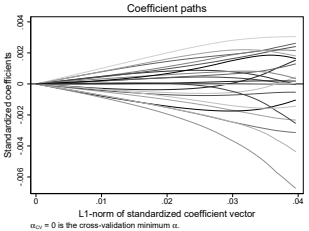
交叉验证

预测香港GDP增速

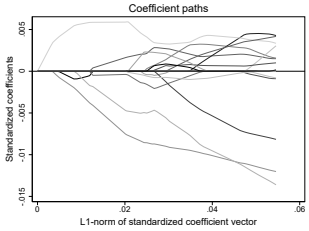
其中

- linear代表要在线性回归中加入惩罚项
- 选项sel(cv)代表使用交叉验证选取\lambda
- alpha(0)代表进行岭回归估计
- rseed()是为了保证每次运行结果都相同的一个随机数种子。
- 估计完成后可以使用lassoinfo命令查看选取的最优λ，也可以使用coefpath命令和cvplot命令可以分别画出系数路径图和交叉验证图。

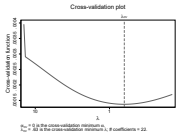
Stata中的Lasso



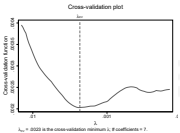
(a) 岭回归系数路径



(b) Lasso回归系数路径



(c) 岭回归交叉验证



(d) Lasso回归交叉验证

Stata中的Lasso

预测香港GDP增速

- 此外，还可以在sel(cv)中加入folds选项以设定交叉验证的折数
  - 比如sel(cv, folds(9))即进行9折交叉验证。
- 如果需要进行Lasso回归，可以设定alpha(1)，或者直接使用lasso命令：

```
lasso linear HongKong Australia-Thailand if _n<=18, rseed(5)
    sel(cv)
```

Stata中的Lasso

预测香港GDP增速

- 除了以上的估计后命令外，还可以使用lassocoef命令查看Lasso回归保留了哪些变量
- 也可以使用preidct命令进行预测，在预测时可以使用经过收缩的Lasso回归系数，也可以使用选择后估计进行预测：

```
predict HongKong_lasso
label variable HongKong_lasso "Lasso"
predict HongKong_lasso_post, post
label variable HongKong_lasso_post "PostLasso"
```

其中加入post选项的是进行后Lasso估计进行的预测。

Stata中的Lasso

预测香港GDP增速

- 如果需要手动进行后Lasso估计，可以使用Lasso估计之后保留的e(post_sel_vars)，该宏记录了所有被Lasso命令保留的变量，其中第一个变量为被解释变量：

```
lasso linear HongKong Australia-Thailand if _n<=18, rseed(5)
    sel(cv)
local post_vars=e(post_sel_vars)
reg `post_vars'
```

超参数选择

- 除了交叉验证外，一些信息准则，特别是BIC也会被用于λ的选择中。
- 此外，文献中对于λ的取值也有非常多的讨论（如Bickel、Ritov和Tsybakov，2009；Belloni和Chernozhukov，2011；Belloni等，2012；Belloni、Chernozhukov和Hansen，2014；Belloni、Chernozhukov和Wei，2016）
- 其形式大概如：

$$\lambda = \frac{c}{\sqrt{N}} \hat{\sigma} \Phi^{-1} \left(1 - \frac{\gamma}{2K}\right)$$

其中根据Belloni和Chernozhukov（2011）建议，$c = 1.1$，γ为未能成功排除真值为0的参数的概率，Stata中取$\gamma = \frac{0.1}{\ln(\max\{K,N\})}$。根据如上形式就可以使用插入（plug-in）法估计λ。

Stata中的Lasso

预测香港GDP增速

- 在Stata中，如果使用BIC选取超参数，可以使用sel(bic)选项：

```
lasso linear HongKong Australia-Thailand if _n<=18, rseed(5)
    sel(bic)
```

而如果需要使用插入法，可以使用sel(plugin)：

```
lasso linear HongKong Australia-Thailand if _n<=18, rseed(5)
    sel(plugin)
```

弹性网

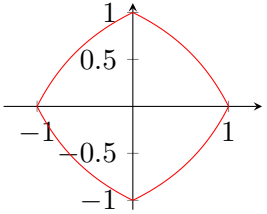
作为一种收缩估计量，岭回归和Lasso回归分别使用了L^2范数和L^1范数，而类似的可以定义很多不同的收缩方法。

- 首先，我们可以将Lasso回归和岭回归结合起来，即同时使用L^2范数和L^1范数的线性组合作为惩罚项，即最小化

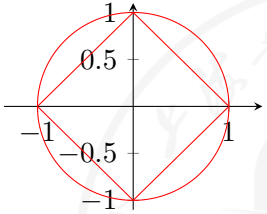
$$\hat{\beta}^{elastic}(\lambda,\alpha)=\arg\min_{\beta}\frac{1}{N}\sum_{i=1}^{N}\left(y_i-x_i'\beta\right)^2+\lambda\left(\alpha\|\beta\|_1+\frac{1-\alpha}{2}\|\beta\|_2^2\right)$$

  其中$\alpha\in[0,1]$为超参数。
  - 易知当$\alpha=0$时以上即岭回归
  - 而当$\alpha=1$时即Lasso回归
- 以上回归被称为**弹性网**（**elastic net**，Zou和Hastie，2005）。

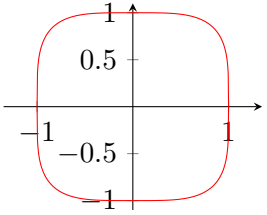
不同收缩方法的惩罚项



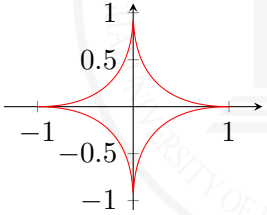
(a)$\alpha = 0.5$，弹性网



(b)$\gamma = 1, 2$，Lasso回归和岭回归



(c)$\gamma = 4$



(d)$\gamma = 0.5$

Bridge

- 如果我们将L^1和L^2范数改为更加一般的L^γ范数，那么我们就得到了**Bridge估计量**，即最小化

$$\hat{\beta}^{brige}(\lambda) = \arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i'\beta)^2 + \lambda \sum_{k=1}^{K} |\beta_k|^\gamma$$

- 显然当$\gamma = 1$时，即Lasso回归，而当$\gamma = 2$时，就得到了岭回归。
- 值得注意的是，当$\gamma < 1$时，可行集并不是一个凸集，当然以上优化问题也不是一个凸优化问题，在计算上存在困难，因而文献中经常假设$\gamma \geq 1$。
- 然而，$\gamma < 1$时的Bridge回归在偏差和选择一致性上都更有优势（Bühlmann和van der Geer，2011），相比较于Lasso具有更好的性质。

适应性Lasso

- 优化问题限制了$\gamma < 1$时的模型应用，此时我们可以使用Zou（2006）提出的**适应性Lasso（adaptive Lasso）**：

$$\hat{\beta}^{ada}(\lambda) = \arg\min_{\beta} \frac{1}{N}\sum_{i=1}^{N}\left(y_i - x_i'\beta\right)^2 + \lambda\sum_{k=1}^{K} w_k |\beta_k|$$

- 其中$w_k = \frac{1}{|\hat{\beta}_k^o|^\theta}$为权重，$\theta > 0$，
  - $\hat{\beta}_j^o$为β的一个一致估计量作为初始估计量，比如OLS估计量。
  - 如果$K > N$，也可以使用岭回归或者Lasso回归等估计量。
  - 注意到如果$\left|\hat{\beta}_k^o\right| = 0$，那么$w_k = \infty$，则$\hat{\beta}_k^{ada} = 0$，从而在估计适应性Lasso时，只需要对第一步Lasso回归中非0的参数进行估计即可。
- 以上步骤可以反复进行，即将$\hat{\beta}^{ada}$作为初始估计量计算新的权重，再次计算适应性Lasso，当然一般实践中只进行一次迭代效果已经足够好。

适应性Lasso

- 适应性Lasso具有很多的优良性质。
  - 首先从计算上来看，适应性Lasso仅仅在Lasso的基础上做了参数惩罚项上的权重调整，其计算代价相比Lasso是完全相同的。
  - 从统计性质上，对于$\theta > 0$，适应性Lasso总是给初始估计量中$\left|\hat{\beta}_k^o\right|$比较小的参数以更大的惩罚，从而使得估计量更加容易收缩至0，从而降低S_0^c中的参数被保留的可能性；
  - 而对于$\left|\hat{\beta}_k^o\right|$比较大的参数以更小的惩罚，从而无偏性受到的影响更小。
  - 综合起来，适应性Lasso可以达到更好的统计性质，实际上Zou（2006）证明适应性Lasso可以达到oracle性质。

Stata中的Lasso

预测香港GDP增速

- 在Stata中，如果需要做适应性Lasso，可以直接在Lasso命令中加入sel(adaptive)选项：

```
1 lasso linear HongKong Australia-Thailand if _n<=18, rseed(5)
     sel(adaptive)
2 lassocoef
```

平方根Lasso

- 以上Lasso的变种都集中在对惩罚项的设定上，而Belloni，Chernozhukov和Wang（2011）则建议将目标函数中的残差平方和改为其平方根，即

$$\hat{\beta}^{ada}(\lambda) = \arg\min_{\beta} \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - x_i'\beta)^2} + \frac{\lambda}{N}\sum_{k=1}^{K}|\beta_k|$$

  即**平方根Lasso（square-root Lasso）**。
- 该方法的好处是在使用插入法计算λ的最优选择时，无须估计σ。
- 此外，该方法可以在误差项非正态分布时也达近乎oracle的收敛速度。

Stata中的Lasso

预测香港GDP增速

- 在Stata中，如果需要做平方根Lasso，可以使用sqrtlasso命令：

```
1 sqrtlasso HongKong Australia-Thailand if _n<=18, rseed(5)
2 lassocoef
```

非线性模型与Lasso

- 通常极大似然估计通过最大化对数似然函数$\max_\beta L(\beta|y,x)$得到，我们可以将目标函数乘以-1从而将其转化为一个求最小化的问题，再加入惩罚项，比如使用L^1正则化项：

$$\min_\beta -L(\beta|y,x) + \lambda \|\beta\|_1$$

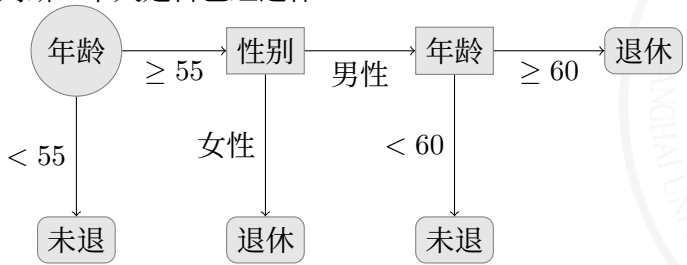
  比如，Probit和Logit回归等，如果需要使用交叉验证法选取超参数（通常是λ），可以通过最小化样本外的预测误差，或者最大化样本外的对数似然函数值进行选择。
- 需要注意的是，虽然在线性回归的目标函数中加入正则化项通常都是凸优化问题，并且可以使用LARS算法等优化方法，然而由于非线性模型中目标函数本身可能是非凸的，导致一般非线性模型的收缩估计量可能难以计算。
- 不过，对于广义线性模型而言，通常存在比较好的优化算法，从而一般的Probit、Logit、Poisson回归如果加入正则化项都可以被高效的计算。

决策树与随机森林

- Lasso回归在处理高维问题上有其优势，然而仍然需要对函数形式进行假设。
- 我们之前已经学过一些非参数回归的方法，然而这些方法很容易面临维数的诅咒。
- 在机器学习中，针对不同类型被解释变量，通常有不同的方法可以在不假设函数形式的前提下进行预测，比如基于树的方法（tree-based methods）、神经网络（neural networks）等，都可以用于回归和分类。这一节我们主要介绍基于树的方法。

决策树的基本概念

决策树（decision tree）使用一种称为树（tree）的数据结构解决分类和回归问题。数据结构中的“树”指的是一种层级结构，为了解释这个概念，我们考虑如何判断一个人是否已经退休：



决策树的基本概念

- 图中圆形的节点代表了所有的样本，我们称其为**根节点**（root node）。
- 圆形中的“年龄”代表我们根据年龄是否大于等于55岁将样本区分为两个子样本，其中小于55岁的样本被判定为没有退休。
- 由于该节点是从根节点划分出来的，我们将其称为根节点的**子节点**（child node）而根节点是该节点的**父节点**（parent node）。
- 一般而言，子节点也可以有子节点，然而对于小于55岁的节点已经没有子节点，我们将其称为**叶子节点**（leaf node）。
- 每个节点如果有子节点，子节点一定是两个，我们将这种在每个非叶子节点上二分的树称为**二叉树**（binary tree）。

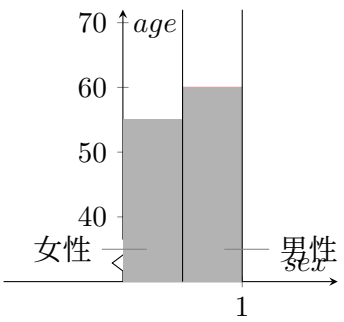
决策树的基本概念

- 使用树这一结构进行建模这一思想有很长的历史，比如在统计学和经济学中，在基于树的方法广泛应用之前，Morgan和Sonquist（1963）就使用了类似的思路和结构。
- 而决策树这一工具的广泛使用得益于一些算法的发明和推广，包括Brieman等人（1984）提出的**CART**（classification and regression trees）、Quinlan（1986）提出的**ID3**（iterative dichotomiser 3）以及Quinlan（1993）提出的ID3算法的改进——**C4.5**等。
- 正如CART的名字所显示的那样，决策树可以解决两类问题：分类问题和回归问题，分别被称为**分类树（classification tree）**和**回归树（regression tree）**。

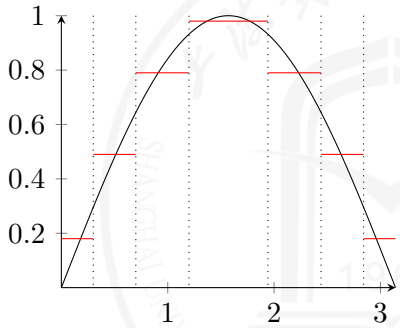
决策树的基本概念

- 实际上不管是分类数还是随机树，其本质是对**特征空间的划分（partition of feature space）**，即将所有x的取值范围划分为一个一个“矩形”，在每个矩形内部赋予同样的预测值。
  - 下图(a)展示了退休的例子中的特征空间及其划分，灰色的两个矩形共同构成了预测为不退休的区域，而白色的部分构成了预测为退休的区域。
  - 类似的，(b)展示了对一个一元函数（正弦函数）的预测，回归树所做的仅仅是将特征空间划分为很多不同的区域，在每个区域上用常数进行预测，从而呈现了阶梯型的函数。

特征空间的划分



(a)



(b)

特征空间的划分

我们可以首先将$x_i = [x_{i1}, ..., x_{iK}]'$的取值范围划分为$M$个区域：$R_1, ..., R_M$

- 对于回归树，可以定义一个阶梯函数：

$$f(x) = \sum_{m=1}^{M} c_m \cdot \mathbb{1}\{x \in R_m\}$$

c_m即区域R_m的预测值，

- 给定划分，我们可以使用R_m内的y的均值对c_m进行估计：

$$\hat{c}_m = \frac{\sum_{i=1}^{N} y_i \cdot \mathbb{1}\{x_i \in R_m\}}{\sum_{i=1}^{N} \mathbb{1}\{x_i \in R_m\}}$$

特征空间的划分

- 而对于分类树，如果可能的分类为$l \in \{1, 2, ..., L\}$，可以定义L个阶梯函数：

$$f_l(x) = \sum_{m=1}^{M} p_{ml} \cdot \mathbb{1}\{x \in R_m\} p_m$$

为区域R_m中y取值为l的概率，给定划分，我们可以使用R_m内的y取l的比例对p_{ml}进行估计：

$$\hat{p}_{ml} = \frac{\sum_{i=1}^{N} \mathbb{1}\{y_i = l\} \cdot \mathbb{1}\{x_i \in R_m\}}{\sum_{i=1}^{N} \mathbb{1}\{x_i \in R_m\}}$$

问题：**如何得到划分？？**

决策树的算法

- 构建决策树的算法通常是一种**贪婪算法**（**greedy algorithm**），即通过迭代的办法，针对每一个节点中的样本，寻找能够最大程度上降低预测误差的方法。
- 因而首先一个重要的问题是，该如何定义划分时的预测误差下降程度?

回归树的预测误差

- 对于回归树，我们可以使用预测误差的平方。现在考虑我们根据x_k对节点进行划分：

$$R_l = \{i|X_{ik} \leq s\}, R_r = \{i|X_{ik} > s\}$$

现在，我们可以根据这一划分分别计算两个子节点的y的均值：$\hat{c}_l, \hat{c}_r$，然而分别计算两个节点的均方误差：

$$Q_l = \frac{\sum_{i=1}^{N} (y_i - \hat{c}_l)^2 \cdot \mathbb{1}\{i \in R_l\}}{\sum_{i=1}^{N} \mathbb{1}\{x_i \in R_l\}}$$

对于RSS_r进行类似定义。

- 如此，我们可以在每个节点，找到能够最小化两个节点残差平方和的k和s：

$$\min_{k,s} N_l Q_l + N_r Q_r$$

从而获得在该节点处的划分规则，其中$N_l = \sum_{i=1}^{N} \mathbb{1}\{x_i \in R_l\}$从而$N_l Q_l + N_r Q_r$即两个节点的残差平方和。

分类树的预测误差

- 而对于分类树，每个节点都有y属于不同类别的样本，我们可以定义一个节点的**纯度（purity）**或者相反的，**杂度（impurity）**。
- 在计算每个节点m的概率估计$\hat{p}_{ml}$之后，我们可以定义几种不同的杂度：
  - 错分率（classification error rate）：如果我们将节点中y的取值个数最多的类别作为该节点的预测，那么被错误划分的比率即错分率，可以定义为$CER_m = 1 - \max_l \hat{p}_{ml}$
  - 基尼系数（Gini index）：$G_m = \sum_{l=1}^{L} \hat{p}_{ml}\,(1 - \hat{p}_{ml})$
  - 交叉熵（cross-entropy）：$CEnt_m = -\sum_{l=1}^{L} \hat{p}_{ml} \log \hat{p}_{ml}$
- 这些指标越大，意味着节点内的类别更混乱，从而纯度更低或者杂度更高。
- CART算法使用Gini系数做为杂度的指标，而C4.5则使用了交叉熵。

分类树的预测误差

- 现在再次考虑以上的二分类问题。如果我们将节点D划分为两个子节点：D_1, D_2
- 为了评估划分之后的杂度，我们还需要进行加权
  - 比如对于基尼系数，需要计算

$$G(D_1, D_2) = \frac{|D_1|}{|D|} G_{D_1} + \frac{|D_2|}{|D|} G_{D_2}$$

其中$|D|$代表D节点的样本量。

  - 而对于交叉熵，可以定义信息增益：

$$Gain(D, D_1, D_2) = CEnt_D - \left(\frac{|D_1|}{|D|} CEnt_{D_1} + \frac{|D_2|}{|D|} CEnt_{D_2} \right)$$

以及增益率：

$$Gain\_ratio(D, D_1, D_2) = \frac{Gain(D, D_1, D_2)}{-\left(\frac{|D_1|}{|D|} \log \frac{|D_1|}{|D|} + \frac{|D_2|}{|D|} \log \frac{|D_2|}{|D|} \right)}$$

C4.5算法中综合使用了信息增益和增益率（可参考周志华，2016）。

过拟合与超参数选择

- 根据以上介绍的算法可以构建一棵树，然而一个很重要的问题是，如果以上步骤无限制的重复下去，最终每个叶子节点中将会都只有一个样本，从而在每个叶子节点中都达到了完美拟合，这显然是一种过拟合。
- 在决策树中，我们通常将限制过拟合的过程称为“**剪枝**”（**pruning**）。
- 剪枝操作可以通过预留验证集的方法操作，此时可以使用预剪枝和后剪枝两种方法：
  - 通过评估验证集上划分是否可以提高预测精度来决定是否对节点进行划分，即预剪枝；
  - 通过构建一颗完整的数，在验证集上判断是否山道两个叶子节点可以提高预测精度，即后剪枝。

过拟合与超参数选择

- 另一种方法是通过一些规则限定树的大小。比如，我们可以规定一个杂度下降的一个最小幅度，如果划分一个节点杂度下降的幅度小于这个最小幅度，就停止划分该节点，并将该节点作为叶子节点。
- 再比如，我们可以通过限制树的层数（深度）限制树的大小。
- 或者，我们可以限制叶子结点的数量。如果记$|T|$为叶子结点的数量，我们可以要求$|T|$小于等于某一个上限。此外，还可以将$|T|$作为正则化项，通过最小化

$$\min \sum_{m=1}^{|T|} N_m Q_m + \alpha |T|$$

对于回归问题，Q_m即均方误差，而对于分类问题，Q_m则为基尼系数、交叉熵等的度量。

- 注意其中α是一个超参数，可以通过交叉验证等方法确定α。

集成学习

- **集成学习**（**ensemble learning**）是一类将多个不同的机器学习模型结合在一起完成预测任务的方法，这里多个不同的机器学习模型可以是不同的回归、不同的决策树等方法。
  - 比如，我们可以将一些深度比较浅的决策树结合在一起进行预测，往往可以达到比单一的树更好的预测效果。由于单颗树往往很不稳定，一些样本上的轻微变化都会导致最终模型的很大差异，使用集成学习方法往往可以解决这一问题。
- 集成学习大体可以分为两种：
  - 一种是不同的模型之间存在着相互依赖关系，必须按照顺序训练的方法，比如**boosting**方法（包括AdaBoot和gradient boosting等）；
  - 另一种是不同的模型之间相互不存在依赖关系，可以独立训练的方法，比如**bagging**和**随机森林**等方法。

Boosting方法简介

- Boosting方法通常是一类可加模型（additive model）：

$$f(x) = \sum_{m=1}^{M} \beta_m b(x, \gamma_m)$$

如果令$f_0(x) = 0$，boosting方法迭代地方法逐步解出β_m, γ_m从而得到最终的模型。

- 由于采用了迭代的方法，因而不同的m之间并不是独立的。
- 常见的boosting方法如AdaBoot和gradient boosting的算法和性质可以参考Hastie，Tibshirani和Friedman（2009），在此不再赘述。

Bagging

- 另一种集成学习的方法是训练一些相互不依赖的模型。
- 其中一种简单的方法是：
  - 将样本进行**有放回抽样**，即bootstrap样本，重复B次就可以得到B个不同的样本
  - 对每一个bootstrap样本都进行模型训练，得到B个模型
  - 然后将B个模型进行结合，比如简单的平均起来：

$$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)$$

  就可以得到最终的预测。如果是分类问题，也可以通过投票的方式，即B颗树中预测出现最多的分类即为预测。
- 这种方法也叫作**bagging（bootstrap aggregation）**。该方法对于偏差并没有太大影响，但是可以大大降低方差，从而提高模型的预测能力。

随机森林

- 而**随机森林**（**random forest**）则在bagging的基础上，额外对特征也进行抽样。
- 假设有K个预测变量，随机森林在每次bootstrap样本的训练时，仅仅使用随机抽取的$\tilde{K}$（对于分类问题$\approx \sqrt{K}$，回归问题$\approx K/3$）个预测变量构建决策树，如此构建B颗树，从而构成随机森林。
- 之所以对预测变量也进行抽样，其原理是降低了不同决策树之间的相关性，从而进一步降低了模型的方差。

Stata中的随机森林

随机森林

- Stata中可以通过pystacked命令调用Python中的scikit-learn包进行随机森林的估计。比如：

```
pystacked exit_labor `x', type(classify) m(rf) cmdopt1(
    n_estimators(100) max_leaf_nodes(20))
```

以上命令只能给定一些超参数（如树的数量、最大的叶子节点数、最小杂度下降以及树的深度等），对于超参数的选择，可以在Stata中通过调整超参数实现，当然也可以直接使用Python挑选好超参数后在Stata中直接使用。