MATH 680 Computation Intensive Statistics

December 3, 2018

Sparse Regularization

Contents

1 The Lasso Estimator **2**

2 Computation of the Lasso Solution **18**

2.1 Single Predictor: Soft Thresholding 19

2.2 Multiple Predictors: Cyclic Coordinate Descent 20

3 ℓ_q Penalties **22**

4 Advantages of ℓ_1-penalty **23**

5 Asymptotic Properties **25**

5.1 Smoothly Clipped Absolute Deviation (SCAD) 28

5.2 Adaptive Lasso 32

6 Generalizations of the Lasso Penalty **37**

6.1 The Elastic Net 38

6.2 The Group Lasso 43

6.3 Sparse Group Lasso 50

6.4 Simulation demo 50

1 The Lasso Estimator

There are two reasons why we might consider an alternative to the least-squares estimate.

- *Prediction accuracy:* The least-squares estimate often has low bias but large variance, and prediction accuracy can sometimes be improved by shrinking the values of the regression coefficients. By doing so, we introduce some bias but reduce the variance of the predicted values, and hence may improve the overall prediction accuracy.

- *Purposes of interpretation:* With a large number of predictors, we often would like to identify a smaller subset of these predictors that exhibit the strongest effects.

In this section, we discuss the various penalty functions $p_\lambda(\cdot)$ used in the penalized problem

$$\arg\min_{\boldsymbol{\beta}}\{L(\boldsymbol{\beta}) + p_\lambda(\boldsymbol{\beta})\}$$

for some loss function $L(\boldsymbol{\beta})$. We mainly use the least squares loss function throughout our discussion.

Definition 1 (The lasso estimator)**.** The lasso estimator, denoted by $\hat{\boldsymbol{\beta}}^{\text{lasso}}$, is defined as

$$\hat{\boldsymbol{\beta}}^{\text{lasso}} = \arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}, \qquad (\lambda > 0)$$

or equivalently,

$$\hat{\boldsymbol{\beta}}^{\text{lasso}} = \arg\min_{\boldsymbol{\beta}} \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta})^2$$
$$\text{subject to} \sum_{j=1}^{p} |\beta_j| \leq t \qquad (t > 0)$$

or equivalently,

$$\hat{\boldsymbol{\beta}}^{\text{lasso}} = \arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \|\mathbf{y} - \beta_0 \mathbf{1} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda ||\boldsymbol{\beta}||_1 \right\}, \qquad (\lambda > 0)$$

where $\mathbf{y} = (y_1, \ldots, y_n)$ denote the n-vector of responses, $\mathbf{X}$ be an $n \times p$ matrix with $\mathbf{x}_i \in \mathbb{R}^p$ in its ith row, $\mathbf{1}$ is the vector of n ones, and $\|\cdot\|_1$ is the ℓ_1-norm and $\|\cdot\|_2$ is the usual Euclidean norm.

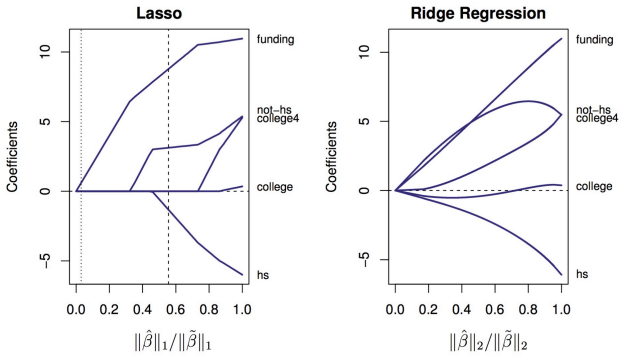
Why do we use the ℓ_1 norm? Why not use the ℓ_2 norm or any ℓ_q norm?

- The lasso yields sparse solution vectors.
- The value $q = 1$ is the smallest value that yields a convex problem.
- Theoretical guarantee.

Note: Typically, we first standardize the predictors $\mathbf{X}$ so that each column is centered $\frac{1}{n}\sum_{i=1}^{n} x_{ij} = 0$ and has unit variance $\frac{1}{n}\sum_{i=1}^{n} x_{ij}^2 = 1$. Without standardization, the lasso solutions would depend on the units. For convenience, we also assume that the outcome values y_i have been centered, meaning that $\frac{1}{n}\sum_{i=1}^{n} y_i = 0$. These centering conditions are convenient, since they mean that we can omit the intercept term β_0 in the lasso optimization. Given an optimal lasso solution $\hat{\boldsymbol{\beta}}$ on the centered data, we can recover the optimal solutions for the uncentered data: $\hat{\boldsymbol{\beta}}$ is the same, and the intercept $\hat{\beta}_0$ is given by

$$\hat{\beta}_0 = \bar{y} - \sum_{j=1}^{p} \bar{x}_j \hat{\beta}_j$$

where $\bar{y}$ and $\{\bar{x}_j\}_1^p$ are the original means. (This is typically only true for linear regression with squared-error loss; it's not true, for example, for lasso logistic regression). For this reason, we omit the intercept β_0 from the lasso for the remainder of this chapter.



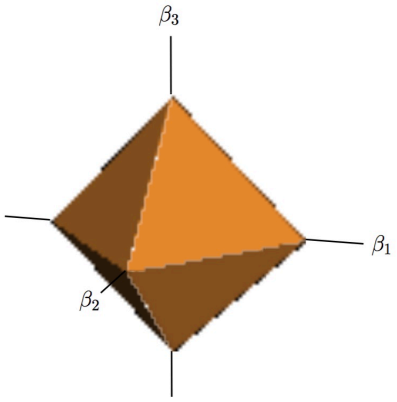


Figure 1: The ℓ_1 ball.

Table 2 shows the results of applying three fitting procedures to the crime data. The lasso bound t was chosen by cross-validation.

- The left panel corresponds to the full least-squares fit.
- The middle panel shows the lasso fit.
- On the right, we have applied least-squares estimation to the subset of three predictors with nonzero coefficients in the lasso. **(Relaxed Lasso)**

The standard errors for the least-squares estimates come from the usual formulas. No such simple formula exists for the lasso, so we have used the bootstrap to obtain the estimate of standard errors in the middle panel.

Overall it appears that funding has a large effect, probably indicating that police resources have been focused on higher crime areas. The other predictors have small to moderate effects. Note that the lasso sets two of the five coefficients to zero, and tends to shrink

the coefficients of the others toward zero relative to the full least-squares estimate. In turn, the least-squares fit on the subset of the three predictors tends to expand the lasso estimates away from zero. The nonzero estimates from the lasso tend to be biased toward zero, so the debiasing in the right panel can often improve the prediction error of the model. This two-stage process is also known as the **relaxed lasso** (Meinshausen 2007).

	LS coef	SE	Z	Lasso	SE	Z	LS	SE	Z
`funding`	10.98	3.08	3.6	8.84	3.55	2.5	11.29	2.90	3.9
`hs`	-6.09	6.54	-0.9	-1.41	3.73	-0.4	-4.76	4.53	-1.1
`not-hs`	5.48	10.05	0.5	3.12	5.05	0.6	3.44	7.83	0.4
`college`	0.38	4.42	0.1	0.0	-	-	0.0	-	-
`college4`	5.50	13.75	0.4	0.0	-	-	0.0	-	-

Figure 2: Results from analysis of the crime data. Left panel shows the least-squares estimates, standard errors, and their ratio (Z-score). Middle and right panels show the corresponding results for the lasso, and the least-squares estimates applied to the subset of predictors chosen by the lasso.

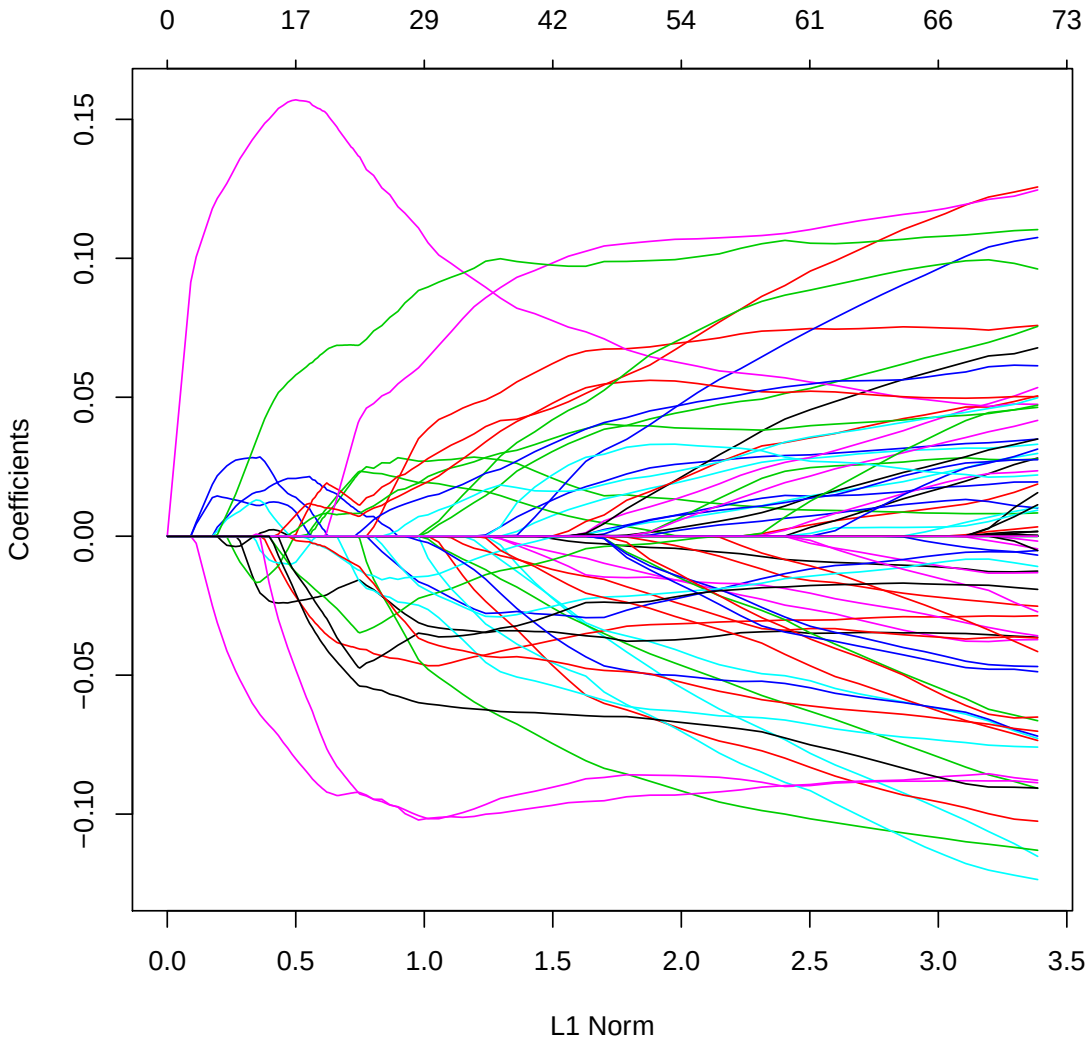
```
# to obtain glmnet and install it directly from CRAN.


# install.packages('glmnet', repos =
# 'http://cran.us.r-project.org')


# load the glmnet package:


library(glmnet)


# The default model used in the package is the Guassian
```

```r
# linear model or 'least squares', which we will demonstrate
# in this section. We load a set of data created beforehand
# for illustration. Users can either load their own data or
# use those saved in the workspace.
getwd()
```

```
## [1] "/Users/yiyang/Dropbox/Teaching/MATH680/notes/Topic4_lasso/note"
```

```r
load("Bardet.rda")

# The command loads an input matrix x and a response vector y
# from this saved R data archive.  We fit the model using the
# most basic call to glmnet.

fit = glmnet(x, y)



# 'fit' is an object of class glmnet that contains all the
# relevant information of the fitted model for further use.
# We do not encourage users to extract the components
# directly.  Instead, various methods are provided for the
# object such as plot, print, coef and predict that enable us
# to execute those tasks more elegantly.
```

```r
# We can visualize the coefficients by executing the plot
# function:

plot(fit)
```



```r
# Each curve corresponds to a variable. It shows the path of
# its coefficient against the l1-norm of the whole
# coefficient vector at as lambda varies. The axis above
# indicates the number of nonzero coefficients at the current
# lambda, which is the effective degrees of freedom (df) for
# the lasso. Users may also wish to annotate the curves; this
# can be done by setting label = TRUE in the plot command.

# A summary of the glmnet path at each step is displayed if
# we just enter the object name or use the print function:

print(fit)
```

```
##
## Call:  glmnet(x = x, y = y)
##
##        Df  %Dev Lambda
##   [1,]  0 0.000 0.1090
##   [2,]  1 0.051 0.1040
##   [3,]  1 0.098 0.0997
##   [4,]  1 0.141 0.0952
##   [5,]  1 0.179 0.0909
##   [6,]  4 0.223 0.0867
##   [7,]  4 0.265 0.0828
```

```
##   [8,]  4 0.304 0.0790
##   [9,]  4 0.339 0.0754
##  [10,]  8 0.373 0.0720
##  [11,]  8 0.405 0.0687
##  [12,]  9 0.435 0.0656
##  [13,]  9 0.462 0.0626
##  [14,]  9 0.486 0.0598
##  [15,] 10 0.509 0.0571
##  [16,] 10 0.530 0.0545
##  [17,] 10 0.549 0.0520
##  [18,] 11 0.567 0.0496
##  [19,] 12 0.583 0.0474
##  [20,] 13 0.598 0.0452
##  [21,] 13 0.613 0.0432
##  [22,] 13 0.627 0.0412
##  [23,] 15 0.641 0.0393
##  [24,] 17 0.653 0.0375
##  [25,] 17 0.665 0.0358
##  [26,] 18 0.676 0.0342
##  [27,] 17 0.686 0.0327
##  [28,] 17 0.695 0.0312
##  [29,] 19 0.703 0.0298
##  [30,] 19 0.711 0.0284
```

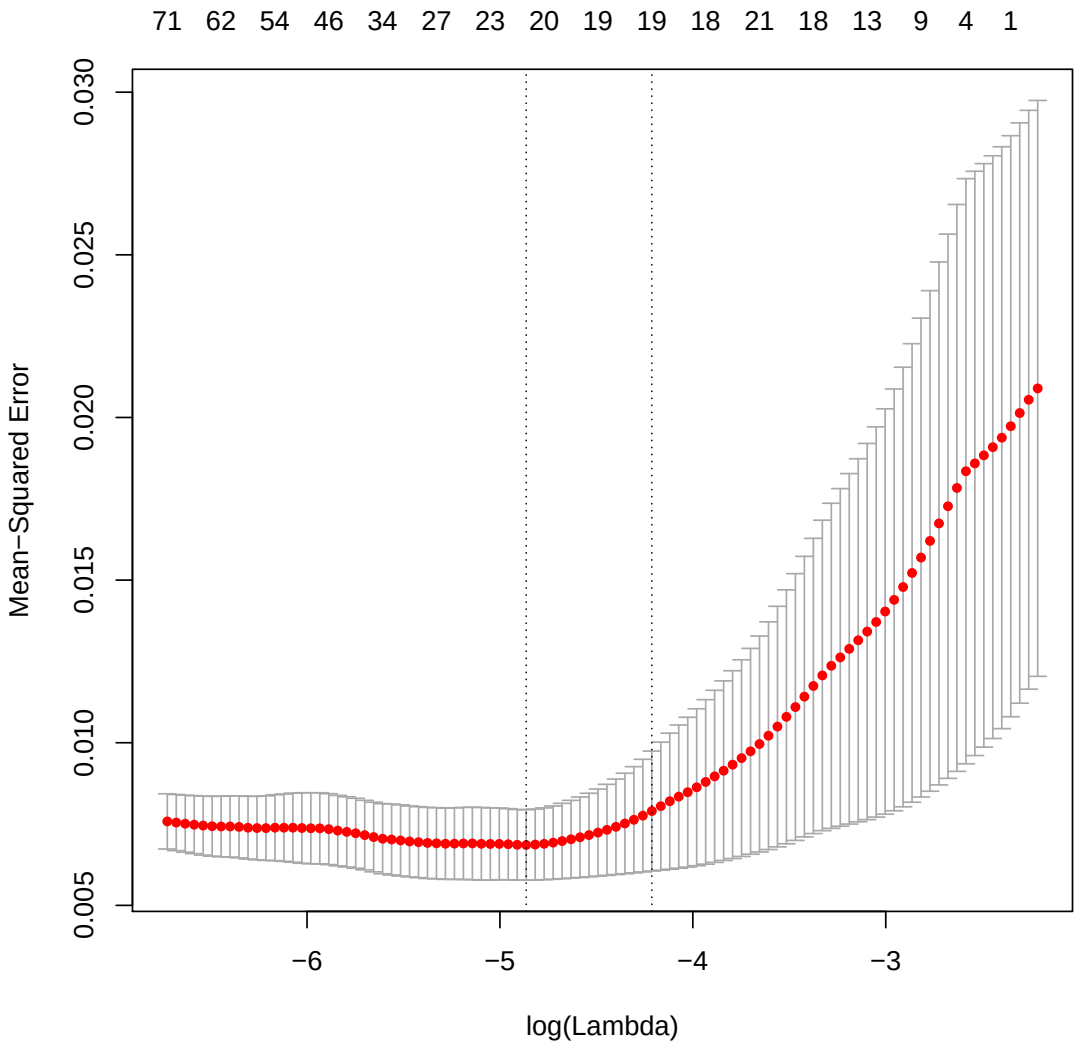
```
##  [31,] 20 0.717 0.0271
##  [32,] 21 0.724 0.0259
##  [33,] 20 0.730 0.0247
##  [34,] 19 0.735 0.0236
##  [35,] 18 0.739 0.0225
##  [36,] 18 0.744 0.0215
##  [37,] 18 0.748 0.0205
##  [38,] 18 0.751 0.0196
##  [39,] 19 0.754 0.0187
##  [40,] 18 0.757 0.0178
##  [41,] 18 0.760 0.0170
##  [42,] 18 0.763 0.0163
##  [43,] 18 0.765 0.0155
##  [44,] 19 0.767 0.0148
##  [45,] 19 0.769 0.0141
##  [46,] 19 0.771 0.0135
##  [47,] 19 0.772 0.0129
##  [48,] 19 0.774 0.0123
##  [49,] 19 0.775 0.0117
##  [50,] 19 0.777 0.0112
##  [51,] 19 0.778 0.0107
##  [52,] 19 0.779 0.0102
##  [53,] 19 0.780 0.0097
```

```
##  [54,] 19 0.780 0.0093
##  [55,] 19 0.781 0.0089
##  [56,] 20 0.782 0.0085
##  [57,] 20 0.783 0.0081
##  [58,] 20 0.783 0.0077
##  [59,] 20 0.784 0.0074
##  [60,] 21 0.785 0.0070
##  [61,] 21 0.789 0.0067
##  [62,] 23 0.794 0.0064
##  [63,] 24 0.799 0.0061
##  [64,] 25 0.804 0.0058
##  [65,] 24 0.809 0.0056
##  [66,] 25 0.812 0.0053
##  [67,] 25 0.816 0.0051
##  [68,] 27 0.820 0.0049
##  [69,] 29 0.824 0.0046
##  [70,] 30 0.828 0.0044
##  [71,] 31 0.833 0.0042
##  [72,] 31 0.838 0.0040
##  [73,] 32 0.842 0.0038
##  [74,] 34 0.846 0.0037
##  [75,] 38 0.851 0.0035
##  [76,] 40 0.857 0.0033
```

```
##  [77,] 40 0.863 0.0032
##  [78,] 41 0.868 0.0031
##  [79,] 42 0.873 0.0029
##  [80,] 46 0.878 0.0028
##  [81,] 50 0.883 0.0027
##  [82,] 52 0.888 0.0025
##  [83,] 55 0.893 0.0024
##  [84,] 55 0.899 0.0023
##  [85,] 54 0.904 0.0022
##  [86,] 54 0.909 0.0021
##  [87,] 55 0.913 0.0020
##  [88,] 57 0.917 0.0019
##  [89,] 59 0.921 0.0018
##  [90,] 61 0.925 0.0017
##  [91,] 62 0.929 0.0017
##  [92,] 62 0.933 0.0016
##  [93,] 62 0.936 0.0015
##  [94,] 64 0.939 0.0014
##  [95,] 64 0.942 0.0014
##  [96,] 66 0.945 0.0013
##  [97,] 69 0.947 0.0013
##  [98,] 71 0.950 0.0012
##  [99,] 73 0.953 0.0011
```

```
## [100,] 73 0.955 0.0011

# It shows from left to right the number of nonzero
# coefficients (Df), the values of -log(likelihood) (%dev)
# and the value of lambda (Lambda).  Although by default
# glmnet calls for 100 values of lambda the program stops
# early if %dev% does not change sufficently from one lambda
# to the next (typically near the end of the path.)

# We can obtain the actual coefficients at one or more
# lambda's within the range of the sequence:

coef0 = coef(fit, s = 0.1)



# The function glmnet returns a sequence of models for the
# users to choose from. In many cases, users may prefer the
# software to select one of them.  Cross-validation is
# perhaps the simplest and most widely used method for that
# task.  cv.glmnet is the main function to do
# cross-validation here, along with various supporting
# methods such as plotting and prediction. We still act on
# the sample data loaded before.
```

```
cvfit = cv.glmnet(x, y)


# cv.glmnet returns a cv.glmnet object, which is 'cvfit'
# here, a list with all the ingredients of the
# cross-validation fit. As for glmnet, we do not encourage
# users to extract the components directly except for viewing
# the selected values of lambda.  The package provides
# well-designed functions for potential tasks.


# We can plot the object.


plot(cvfit)
```




```
# It includes the cross-validation curve (red dotted line),
# and upper and lower standard deviation curves along the
# lambda sequence (error bars). Two selected lambda's are
# indicated by the vertical dotted lines (see below).
```

```r
# We can view the selected lambda's and the corresponding
# coefficients. For example,

cvfit$lambda.min
```

```
## [1] 0.0077
```

```r
# lambda.min is the value of lambda that gives minimum mean
# cross-validated error. The other lambda saved is
# lambda.1se, which gives the most regularized model such
# that error is within one standard error of the minimum. To
# use that, we only need to replace lambda.min with
# lambda.1se above.

coef1 = coef(cvfit, s = "lambda.min")


# Note that the coefficients are represented in the sparse
# matrix format. The reason is that the solutions along the
# regularization path are often sparse, and hence it is more
# efficient in time and space to use a sparse format.  If you
# prefer non-sparse format, pipe the output through
# as.matrix().
```

```r
# Predictions can be made based on the fitted cv.glmnet
# object. Let's see a toy example.

predict(cvfit, newx = x[1:5, ], s = "lambda.min")
```

```
##      1
## V2 8.4
## V3 8.3
## V4 8.4
## V5 8.3
## V6 8.4
```

```r
# newx is for the new input matrix and s, as before, is the
# value(s) of lambda at which predictions are made.
```

2 Computation of the Lasso Solution

Lasso prefers sparse solution. To see this, notice that, with ridge regression, the prior cost of a sparse solution, such as $\boldsymbol{\beta} = (1, 0)$, is the same as the cost of a dense solution, such as $\boldsymbol{\beta} = (1/\sqrt{2}, 1/\sqrt{2})$, as long as they have the same ℓ_2 norm:

$$\|(1, 0)\|_2 = \|(1/\sqrt{2}, 1/\sqrt{2})\|_2 = 1.$$

However, for lasso, setting $\boldsymbol{\beta} = (1, 0)$ is cheaper than setting $\boldsymbol{\beta} = (1/\sqrt{2}, 1/\sqrt{2})$, since

$$\|(1, 0)\|_1 = 1 < \|(1/\sqrt{2}, 1/\sqrt{2})\|_1 = \sqrt{2}.$$

The most rigorous way to see that ℓ_1 regularization results in sparse solutions is to examine the conditions that hold at the optimum.

2.1 Single Predictor: Soft Thresholding

In this section, z_i has been centered. Consider a single predictor setting, based on samples $\{(y_i, z_i)\}_{i=1}^n$ (for convenience we have renamed z_i to be x_{ij}). The problem then is to solve

$$\arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - z_i\beta)^2 + \lambda|\beta| \right\} \tag{1}$$

We cannot get the optimality condition directly, since $|\beta|$ does not have a derivative at $\beta = 0$. By direct inspection of the function (1), we find that

$$\hat{\beta} = \begin{cases} \frac{1}{n}\langle \mathbf{z}, \mathbf{y} \rangle - \lambda & \text{if } \frac{1}{n}\langle \mathbf{z}, \mathbf{y} \rangle > \lambda \\ 0 & \text{if } \frac{1}{n}|\langle \mathbf{z}, \mathbf{y} \rangle| \leq \lambda \\ \frac{1}{n}\langle \mathbf{z}, \mathbf{y} \rangle + \lambda & \text{if } \frac{1}{n}\langle \mathbf{z}, \mathbf{y} \rangle < -\lambda \end{cases},$$

which can be written as

$$\hat{\beta} = \mathcal{S}_\lambda(\frac{1}{n}\langle \mathbf{z}, \mathbf{y} \rangle),$$

where the soft-thresholding operator

$$\mathcal{S}_\lambda(x) = \text{sign}(x)(|x| - \lambda)_+.$$

when data is standardized $\frac{1}{n}\sum_i z_i^2 = 1$, it translates the usual least-squares estimate $\hat{\beta}^{\text{OLS}} = \langle \mathbf{z}, \mathbf{y} \rangle / \langle \mathbf{z}, \mathbf{z} \rangle = \frac{1}{n} \langle \mathbf{z}, \mathbf{y} \rangle$ toward zero by the amount λ. This is demonstrated in Figure 3.

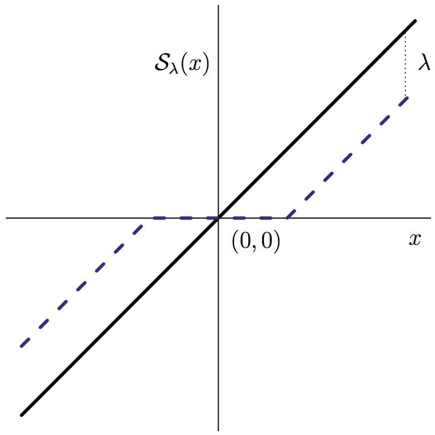


Figure 3: Soft thresholding function $\mathcal{S}_\lambda(x) = \text{sign}(x)(|x| - \lambda)_+$ is shown in blue (broken lines), along with the $45°$ line in black.

2.2 Multiple Predictors: Cyclic Coordinate Descent

We are simplifying notation by assuming that y is centered and X is column-centered. Using this intuition from the univariate case, we can now develop a simple coordinatewise scheme for solving the full lasso problem

$$\min_{\beta} \left\{ \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_1 \right\}. \tag{2}$$

Writing the objective in (2) as

$$\frac{1}{2n}\sum_{i=1}^{n}(y_i - \sum_{k\neq j} x_{ik}\beta_k - x_{ij}\beta_j)^2 + \lambda\sum_{k\neq j}|\beta_k| + \lambda|\beta_j|,$$

we see that solution for each β_j can be expressed succinctly in terms of the partial residual $r_i^{(j)} = y_i - \sum_{k\neq j} x_{ik}\hat{\beta}_k$, the j^{th} coefficient is updated as

$$\hat{\beta}_j = \mathcal{S}_\lambda\left(\frac{1}{n}\left\langle \mathbf{x}_j, \mathbf{r}^{(j)}\right\rangle\right).$$

Equivalently, the update can be written as

$$\hat{\beta}_j^{\text{new}} \leftarrow \mathcal{S}_\lambda\left(\hat{\beta}_j^{\text{old}} + \frac{1}{n}\left\langle \mathbf{x}_j, \mathbf{r}\right\rangle\right), \tag{3}$$

where $r_i = y_i - \sum_{j=1}^{p} x_{ij}\hat{\beta}_j$ are the full residuals **(Homework 2)**. The overall algorithm operates by applying this soft-thresholding update (3) repeatedly in a cyclical manner, updating the coordinates of $\hat{\boldsymbol{\beta}}$ (and hence the residual vectors) along the way.

In practice, one is often interested in finding the lasso solution not just for a single fixed value of λ, but rather the entire path of solutions over a range of possible λ values. A reasonable method for doing so is to begin with a value of λ just large enough so that the only optimal solution is the all-zeroes vector. As shown in **Homework 2**, this value is equal to $\lambda_{\max} = \max_j |\frac{1}{n}\left\langle \mathbf{x}_j, \mathbf{y}\right\rangle|$. Then we decrease λ by a small amount and run coordinate descent until convergence. Decreasing λ again and using the previous solution as a "warm start," we then run coordinate descent until convergence. In this way we can efficiently compute the solutions over a grid of λ values. We refer to this method as pathwise

coordinate descent. The full algorithm is described in Algorithm 1.

Algorithm 1: Cyclic Coordinate Descent.

Set a decreasing sequence $\lambda_1, \ldots, \lambda_K$, where $\lambda_1 = \lambda_{\max}$;
Initialize $\hat{\boldsymbol{\beta}}_{\lambda_0} = \mathbf{0}$;
for $k = 1, \ldots, K$ **do**
  Set $\lambda = \lambda_k$;
  Initialize $\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}_{\lambda_{k-1}}$;
  for $j = 1, \ldots, p, 1, \ldots, p, \ldots$ **do**
    1. Compute the current residual $r_i = y_i - \sum_{j=1}^{p} x_{ij}\hat{\beta}_j$ for $i = 1, \ldots, n$;
    2. Update

$$\hat{\beta}_j^{\text{new}} \leftarrow \mathcal{S}_\lambda\left(\hat{\beta}_j^{\text{old}} + \frac{1}{n}\langle \mathbf{x}_j, \mathbf{r}\rangle\right)$$

    3. Exit if $\hat{\boldsymbol{\beta}}$ converges.
  end
  Set $\hat{\boldsymbol{\beta}}_{\lambda_k} = \hat{\boldsymbol{\beta}}$;
end

3 ℓ_q Penalties

For a fixed real number $q \geq 0$, consider the criterion

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j|^q \right\}. \tag{4}$$

This is the lasso for $q = 1$ and ridge regression for $q = 2$. For $q = 0$, the term $\sum_{j=1}^{p} |\beta_j|^0$ counts the number of nonzero elements in β, and thus amounts to best-subset selection. Figure 4 displays the constraint regions corresponding to these penalties for the case of two predictors ($p = 2$).

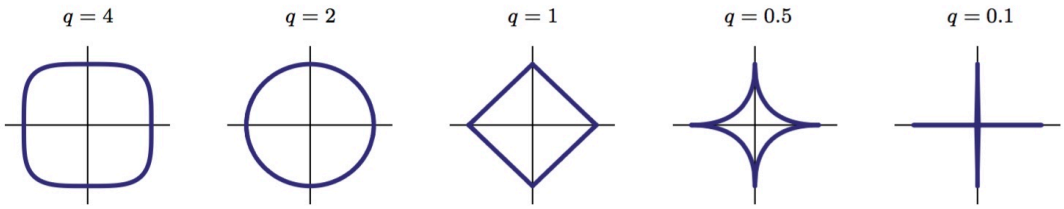


Figure 4: Constraint regions $\sum_{j=1}^{p} |\beta_j|^q \leq 1$ for different values of q. For $q < 1$, the constraint region is nonconvex.

In the special case of an orthonormal model matrix $\mathbf{X}$, all three procedures have explicit solutions. Each method applies a simple coordinate-wise transformation to the least-squares estimate $\tilde{\boldsymbol{\beta}}$ as detailed in Table 1.

The lasso is special in that the choice $q = 1$ is the smallest value of q (closest to best-subset) that leads to a convex constraint region and hence a convex optimization problem. In this sense, it is the closest convex relaxation of the best-subset selection problem.

q	Estimator	Formula
0	Best subset	$\tilde{\beta}_j \cdot \mathbb{I}[\lvert\tilde{\beta}_j\rvert > \sqrt{2\lambda}]$
1	Lasso	$\text{sign}(\tilde{\beta}_j)(\lvert\tilde{\beta}_j\rvert - \lambda)_+$
2	Ridge	$\tilde{\beta}_j/(1+\lambda)$

Table 1: Estimators of β_j from (4) in the case of an orthonormal model matrix $\mathbf{X}$.

4 Advantages of ℓ_1-penalty

- *Interpretation of the final model:* the ℓ_1-penalty provides a natural way to encourage or enforce sparsity and simplicity in the solution.

- *Statistical efficiency:* **bet-on-sparsity principle** – assume that the underlying true signal is sparse and we use an ℓ_1-penalty to try to recover it. If our assumption is correct, we can do a good job in recovering the true signal. But if we are wrong—the underlying truth is not sparse in the chosen bases—then the ℓ_1-penalty will not work well. However in that instance, no method can do well, relative to the Bayes error. There is now a large body of theoretical support for these loose statements.

  - We can think of this in terms of the amount of information n/p per parameter. If $p \gg n$ and the true model is not sparse, i.e. $k \approx n$, then the number of samples n is too small to allow for accurate estimation of the parameters.
  - But if the true model is sparse, so that only $k < n$ parameters are actually nonzero in the true underlying model, then it turns out that we can estimate the parameters effectively, using the lasso. This may come as somewhat of a surprise, because we are able to do this even though we are not told which k of the p parameters are actually nonzero. Of course we cannot do as well as we could if we had that information, but it turns out that we can still do reasonably well.

- *Computational efficiency:* *ℓ_1-based penalties are convex and this fact and the assumed sparsity can lead to significant computational advantages.*

5 Asymptotic Properties

Consider the linear regression model

$$y_i = \mathbf{x}_i^\top \boldsymbol{\beta}^* + \epsilon_i,$$

where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^\top$, $\boldsymbol{\beta}^* = (\beta_1^*, \ldots, \beta_p^*)$ and $\epsilon_1, \ldots, \epsilon_n$ are iid random errors with mean 0 and variance σ^2.

Let $\mathcal{A}^* = \{j : \beta_j^* \neq 0\}$ denote the support of the true parameter $\boldsymbol{\beta}^* = (\beta_1^*, \ldots, \beta_p^*)$ and $\mathcal{A}_n = \{j : \hat{\beta}_j \neq 0\}$ denote the support of the penalized estimator $\hat{\boldsymbol{\beta}}_n = (\hat{\beta}_{n,1}, \ldots, \hat{\beta}_{n,p})$. Two important properties that any penalized estimator $\hat{\boldsymbol{\beta}}$ should possess, which make up the so-called **oracle property**, are the following:

- *Variable selection consistency:*

$$\lim_{n\to\infty} P(\mathcal{A}_n \to \mathcal{A}^*) = 1$$

- $\sqrt{n}$*-estimation consistency:*

$$\sqrt{n}(\hat{\boldsymbol{\beta}}_{n,\mathcal{A}^*} - \boldsymbol{\beta}_{A^*}^*) \xrightarrow{d} \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_0)$$

  where $\boldsymbol{\Sigma}_0$ is the covariance matrix knowing the true subset model.

Variable selection consistency – whether the estimator can identify the true support, asymptotically, if the true parameter is sparse.

$\sqrt{n}$-*estimation consistency* – whether the estimator behaves as well as the ordinary least squares estimator and is asymptotically normal with the same covariance matrix knowing the true subset model.

- The asymptotic properties of the lasso estimator for fixed p were studied in Knight and Fu (2000), Fan and Li (2001), Zhao and Yu (2006), and Zou (2006). These authors showed that the lasso estimator is **estimation consistent**, but the optimal rate of estimation is available only when $\lambda_n = O(\sqrt{n})$. However, this leads to **inconsistent variable selection**.

- The question of interest then becomes whether consistency in variable selection can be achieved if we are willing to sacrifice the convergence rate in estimation. It turns out that a slower rate of convergence **does not** guarantee variable selection consistency.

- A necessary condition for the lasso to be variable selection consistent is the irrepresentable condition (Zhao and Yu, 2006), which concerns the design matrix $\mathbf{X}$.

- Zhao and Yu (2006) showed that the *irrepresentable condition* is **almost necessary and sufficient** for lasso to be variable selection consistent both for fixed p and diverging p as the sample size n increases.

We re-arrange β such that $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_s, \beta_{s+1}, \ldots, \beta_p)^\top$, $\beta_j \neq 0$ for $j = 1, \ldots, s$ and $\beta_j = 0$ for $j = s+1, \ldots, p$. Further, we let $C^{(n)} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{x}_i$,

$$C^{(n)} = \begin{pmatrix} C_{11}^{(n)} & C_{12}^{(n)} \\ C_{21}^{(n)} & C_{22}^{(n)} \end{pmatrix}$$

where $C_{11}^{(n)}$ is an $s \times s$ matrix and write $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^\top, \boldsymbol{\beta}_2^\top)$.

Definition 2 (Strong irrepresentable condition; Zhao and Yu, 2006)**.** There exists a positive constant vector $\boldsymbol{\eta}$ such that

$$|C_{21}^{(n)}(C_{11}^{(n)})^{-1}\text{sign}(\boldsymbol{\beta}_1)| \leq 1 - \boldsymbol{\eta}$$

where $\mathbf{1}$ is a $(p-s) \times 1$ vector of 1's and the inequality holds componentwise.

Definition 3 (Weak irrepresentable condition; Zhao and Yu, 2006)**.**

$$|C_{21}^{(n)}(C_{11}^{(n)})^{-1}\text{sign}(\boldsymbol{\beta}_1)| \leq 1$$

- The weak irrepresentable condition is necessary for variable selection consistency of the lasso. Zhao and Yu (2006), and Zou (2006)
- The strong irrepresentable condition is sufficient for selection consistency of the lasso.
- These conditions, however, can be restrictive in high dimensions. The weak irrepresentable condition states that the lasso is variable selection consistent if and (almost) only if the variables that are not in the true model are "irrepresentable" by variables that are in the true model.
- In other words, if an irrelevant variable is highly correlated with variables in the true model, the lasso may fail to distinguish it from the true variables even with large n.

- To improve the performance of the lasso, a variety of penalties have been proposed, such as the smoothly clipped absolute deviation (SCAD) penalty of Fan and Li (2001), and the adaptive lasso of Zou (2006).

5.1 Smoothly Clipped Absolute Deviation (SCAD)

SCAD estimator is defined as the minimizer of

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \sum_{j=1}^{p} p_\lambda(\beta_j) \right\}$$

where $p_\lambda(u) = \lambda|u|$ is a non-convex penalty (Fan and Li, 2001) whose derivative is

$$p'_\lambda(u) \quad = \quad \lambda \left\{ I(u \leq \lambda) + \frac{(a\lambda - u)_+}{\lambda(a-1)} I(u > \lambda) \right\} \quad (a > 2).$$

The SCAD penalty function is continuously differentiable everywhere except at 0 and its derivative vanishes outside $[-a\lambda, a\lambda]$. Thus, it can produce continuous and sparse solutions and unbiased estimates for large coefficients, satisfying the following properties:

Unbiasedness: The resulting estimator is nearly unbiased when the true unknown parameter is large.

Sparsity: The estimator is a thresholding rule, which automatically sets small estimated coefficients to zero to reduce model complexity.

Continuity: The estimator is continuous in the data to avoid instability in prediction.

We provide a plot of the SCAD penalty function in Figure 5 along with the lasso penalty function.

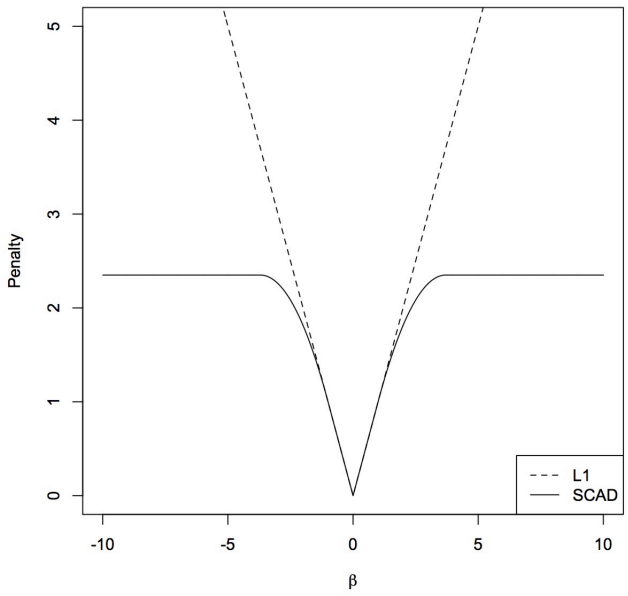


Figure 5: Plot of the lasso and SCAD penalty functions with $\lambda = 1$ and $a = 3.7$.

- **Sparsity**: the lasso and SCAD penalty functions behave similarly for small coefficients.
- **Continuity**: for larger coefficients, SCAD applies a constant penalty, whereas the lasso penalty increases linearly with the coefficient.
- **Unbiasedness**: hence, the SCAD penalty function results in asymptotically unbiased estimators while the lasso penalty function does not.

Fan and Li (2001) showed that there exists a penalized likelihood estimator that converges at the rate

$$O_p(\sqrt{n} + a_n)$$

where $a_n = \max\{p'_{\lambda_n}(\beta_j^*) : \beta_j^* \neq 0\}$. Therefore is $\sqrt{n}$-consistent if $\lambda_n \to 0$. They also

showed that under conditions (A)-(C) in Fan and Li (2001), if $\lambda_n \to 0$ and $\sqrt{n}\lambda_n \to \infty$ as $n \to \infty$, the SCAD estimator has the **oracle property**.

For the lasso penalty, $a_n = \lambda_n$ and $\lambda_n = O_p(\sqrt{n})$ is required for $\sqrt{n}$-consistency. However, the oracle property requires that $\lambda_n\sqrt{n} \to \infty$. These conditions cannot be simultaneously satisfied.

Fan and Peng (2004) extended the results of Fan and Li (2001) to the case where p diverges with the sample size n. They established an **oracle property** and the **asymptotic normality** in the setting with $p = o(n^{1/5})$.

Penalties such as MCP (**?**) and SICA (**?**) also satisfy these three aforementioned properties. They all depend on λ, so denote Their derivative are defined on $[0, \infty)$ by

$$\begin{aligned} \text{MCP}: \quad & p'_\lambda(|\beta_j|) = \frac{(a\lambda - |\beta_j|)_+}{a} \\ \text{SICA}: \quad & p'_\lambda(|\beta_j|) = \lambda \frac{a(a+1)}{(a+|\beta_j|)^2} \end{aligned}$$

where a is a tuning parameter that affects the range over which the penalty is applied. As shown in 6, these non-convex penalties share some common features: the similar or same rate of penalization as LASSO is applied when estimate sizes $|\beta_j|$ are small, but that penalization is continuously relaxed as size increases. This assures the elimination of the unimportant variables from the model while leaving the important ones unpenalized.

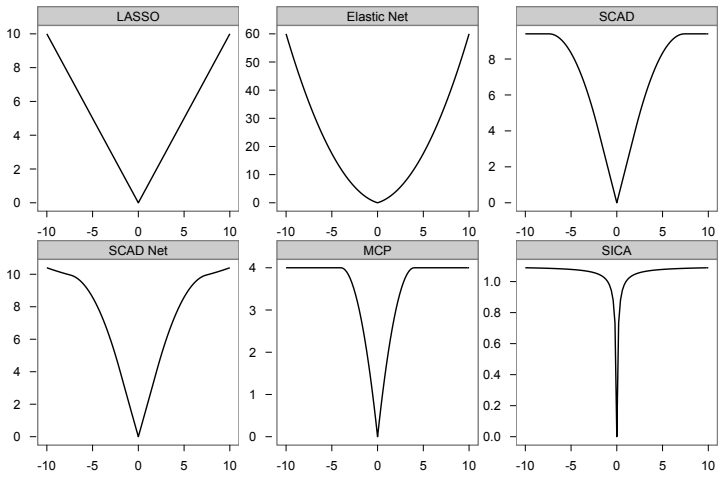
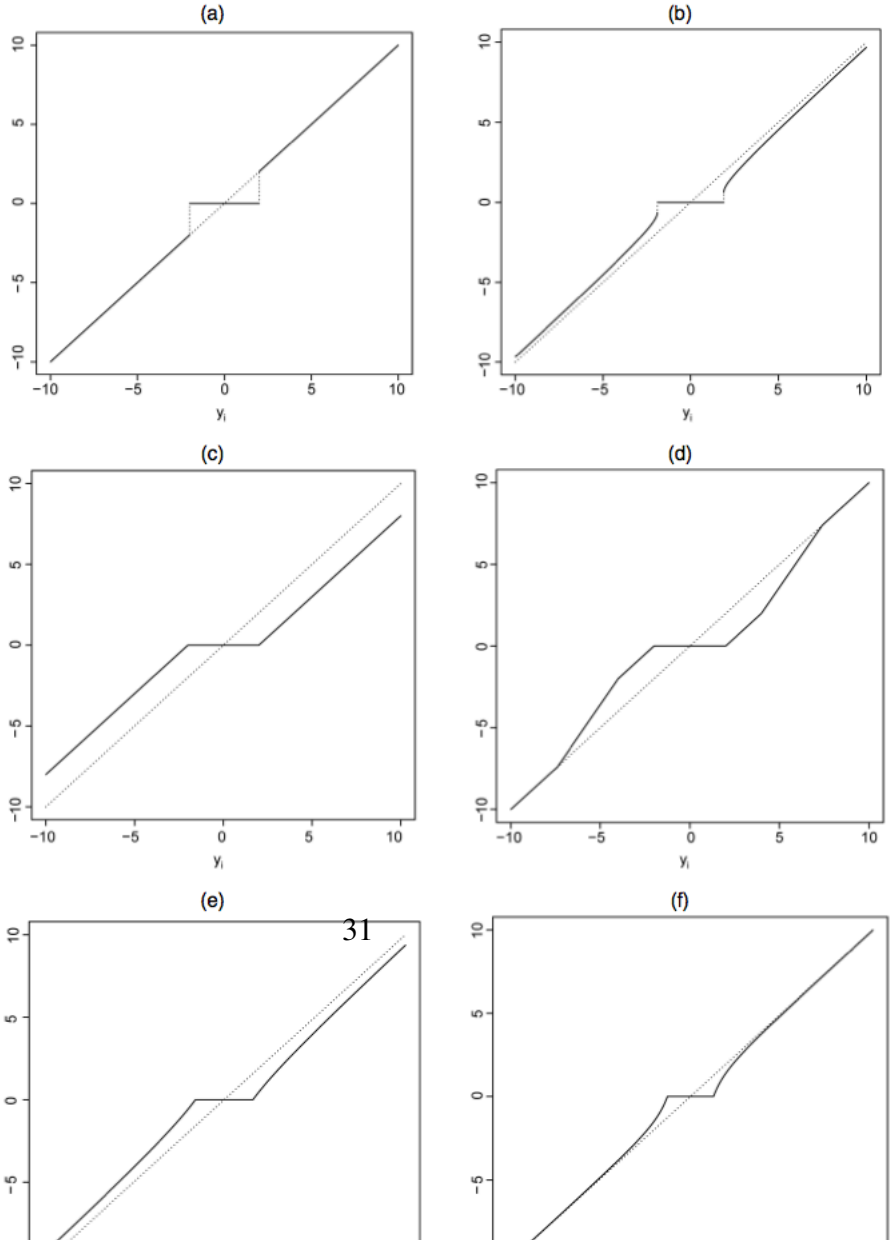


Figure 6: Different penalties.



5.2 Adaptive Lasso

The adaptive lasso estimator is defined as the minimizer of

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda_n \sum_{j=1}^{p} \hat{w}_j |\beta_j|^q \right\}, \tag{5}$$

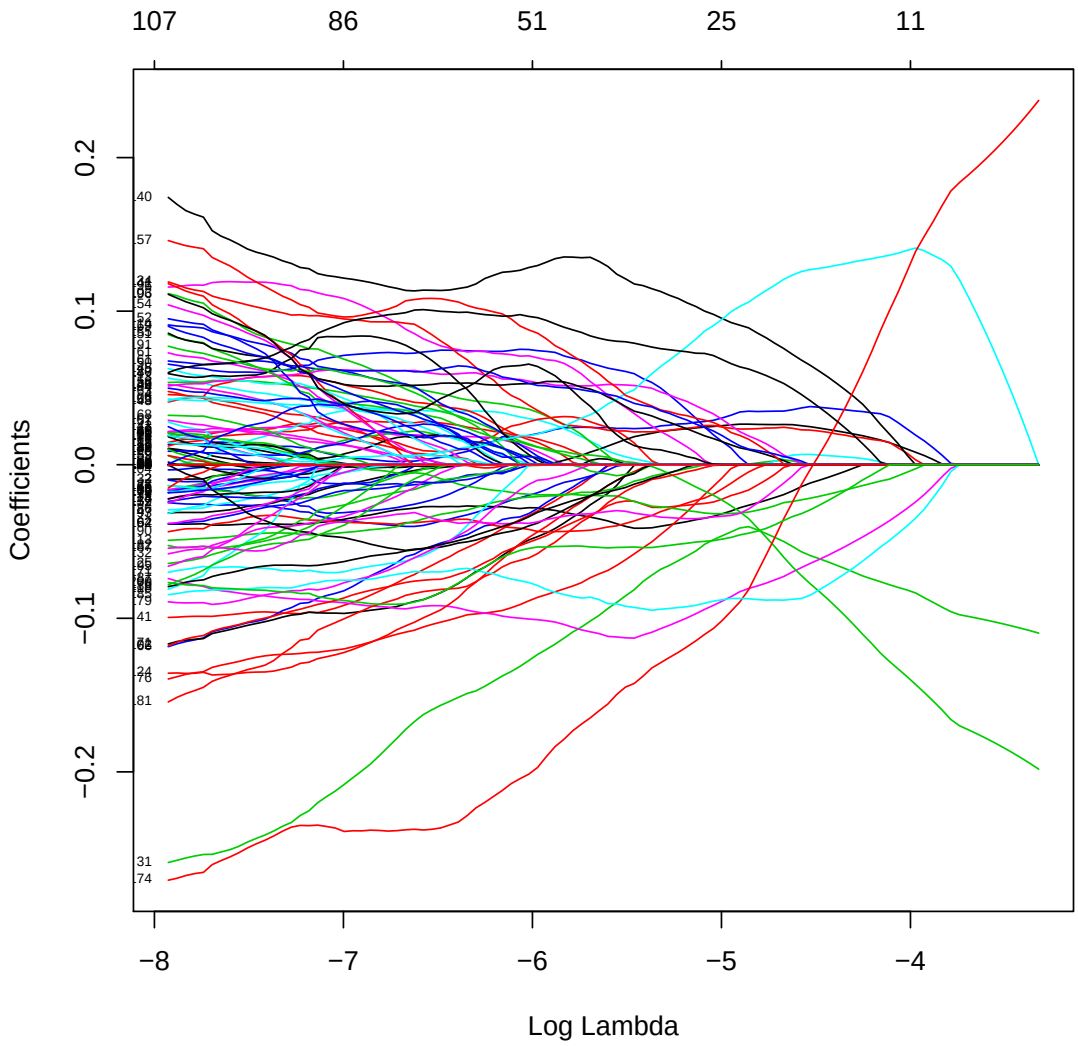
where $\hat{w}_j = \frac{1}{|\hat{\beta}_j|^\gamma}$ for some $\gamma > 0$ and a $\sqrt{n}$-consistent estimator $\hat{\beta}_j$ of β_j.

- As $n \to \infty$, the weights corresponding to insignificant variables tend to infinity, while the weights corresponding to significant variables converge to a finite constant. Thus, large coefficients can be estimated unbiasedly (asymptotically) and small coefficients can be thresholded, simultaneously.

- Zou (2006) studied the asymptotic properties of the adaptive lasso estimator for fixed p as $n \to \infty$. He showed that, under certain regularity conditions, if $\lambda_n/\sqrt{n} \to 0$ and and $\lambda_n n^{(\gamma-1)/2} \to \infty$, then the adaptive lasso estimator possesses the oracle property.

```
################################################ penalty.factor example



# [penalty.factor] argument allows users to apply separate
# penalty factors to each coefficient. Its default is 1 for
# each parameter, but other values can be specified. In
# particular, any variable with penalty.factor equal to zero
# is not penalized at all! Let v_j denote the penalty factor
```

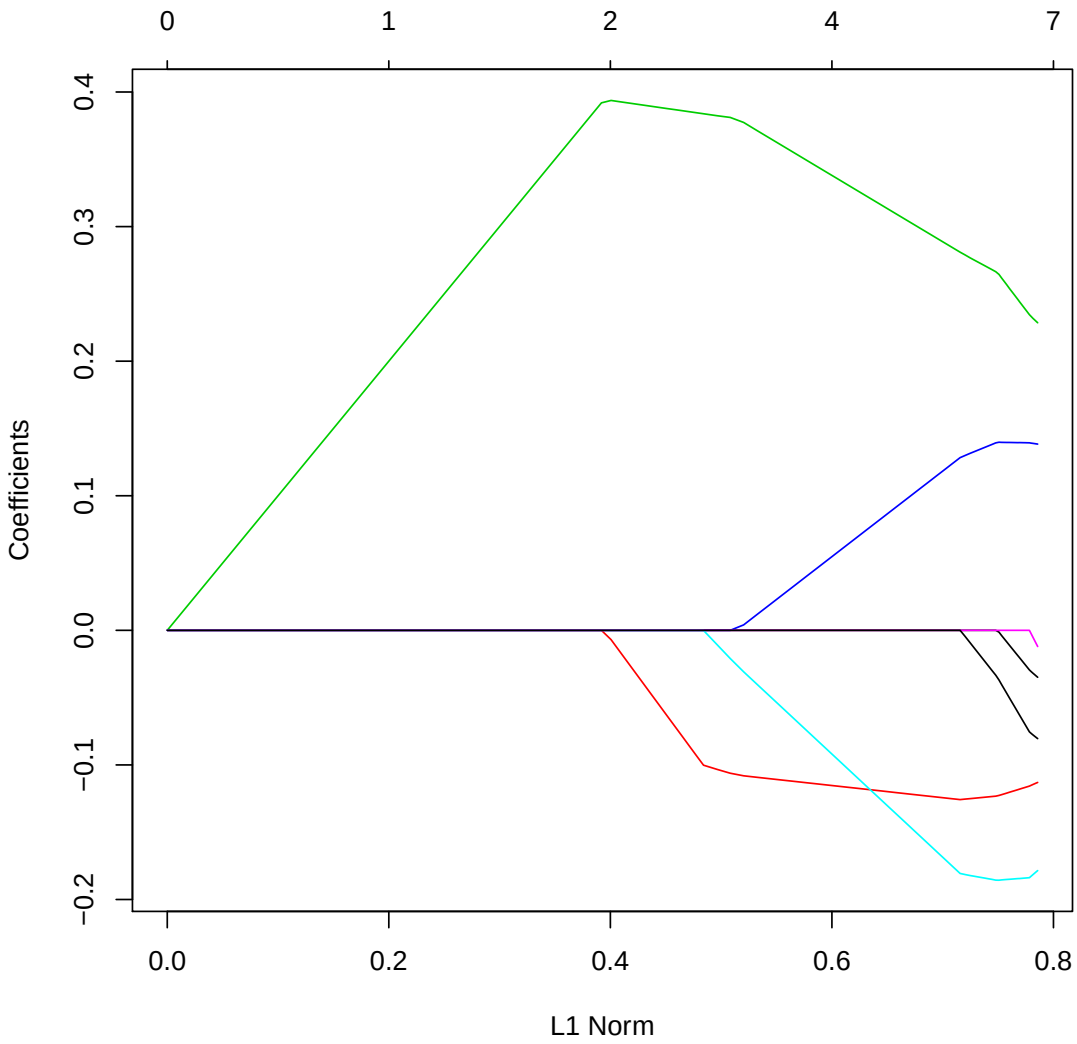
```
# for j-th variable.

# Note the penalty factors are internally rescaled to sum to
# nvars.

# This is very useful when people have prior knowledge or
# preference over the variables. In many cases, some
# variables may be so important that one wants to keep them
# all the time, which can be achieved by setting
# corresponding penalty factors to 0:

p.fac = rep(1, 200)
p.fac[c(31, 174, 151)] = 0
pfit = glmnet(x, y, penalty.factor = p.fac)
plot(pfit, xvar = "lambda", label = TRUE)
```




```
# We see from the labels that the three variables with 0
# penalty factors always stay in the model, while the others
# follow typical regularization paths and shrunken to 0
# eventually.
```

```r
################################################## Adaptive lasso example

# Some other useful arguments: [exclude] allows one to block
# certain variables from being the model at all. Of course,
# one could simply subset these out of x, but sometimes
# exclude is more useful, since it returns a full vector of
# coefficients, just with the excluded ones set to zero.
# There is also an intercept argument which defaults to TRUE;
# if FALSE the intercept is forced to be zero.


## The adaptive lasso needs a first stage that is consistent.
## Zou (2006) recommends OLS or ridge first stage lasso
thelasso.cv <- cv.glmnet(x, y, family = "gaussian", alpha = 1)
## Second stage weights from the coefficients of the first
## stage coef() is a sparseMatrix
bhat <- as.matrix(coef(thelasso.cv, s = "lambda.1se"))[-1, 1]
if (all(bhat == 0)) {
    ## if bhat is all zero then assign very close to zero weight
    ## to all.  Amounts to penalizing all of the second stage to
    ## zero.
```

```
    bhat <- rep(.Machine$double.eps * 2, length(bhat))
}
## the adaptive lasso weight
adpen <- (1/pmax(abs(bhat), .Machine$double.eps))
## Second stage lasso (the adaptive lasso)
m_adlasso <- glmnet(x, y, family = "gaussian", alpha = 1, exclude = which(bhat ==
    0), penalty.factor = adpen)
plot(m_adlasso)
```



6 Generalizations of the Lasso Penalty

Generalized penalties arise in a wide variety of settings:

- **Elastic net:** handle highly correlated features. e.g. genes.

- **Group lasso/overlap group lasso:** handle structurally grouped features. e.g. dummy variables.

6.1 The Elastic Net

The lasso does not handle highly correlated variables very well; the coefficient paths tend to be erratic and can sometimes show wild behavior. Consider a simple but extreme example, where the coefficient for a variable X_j with a particular value for λ is $\hat{\beta}_j = 1$. If we augment our data with an *identical* copy $X_{j'} = X_j$, then they can share this coefficient in infinitely many ways – any $\tilde{\beta}_j + \tilde{\beta}_{j'} = \hat{\beta}_j$ with both pieces positive – and the ℓ_1 penalty cost are indifferent, e.g.

$$\|(\tilde{\beta}_j, \tilde{\beta}_{j'})\|_1 = \|(0.5, 0.5)\|_1 = \|(0.3, 0.7)\|_1.$$

So the coefficients for $(\tilde{\beta}_j, \tilde{\beta}_{j'})$ are not definite. A quadratic penalty, on the other hand, will divide $\hat{\beta}_j$ exactly equally between $\tilde{\beta}_j, \tilde{\beta}_{j'}$ (**Homework 2**), e.g.

$$\|(\tilde{\beta}_j, \tilde{\beta}_{j'})\|_2 = \|(0.5, 0.5)\|_2 < \|(0.3, 0.7)\|_2.$$

In practice, we are unlikely to have an identical pair of variables, but often we do have groups of very correlated variables. e.g. in microarray studies, groups of genes in the same biological pathway tend to be expressed (or not) together, and hence measures of their expression tend to be strongly correlated.

Consider an example with a sample size of $n = 100$. There are two sets of three variables, with pairwise correlations around 0.97 in each group. The data were simulated as follows:

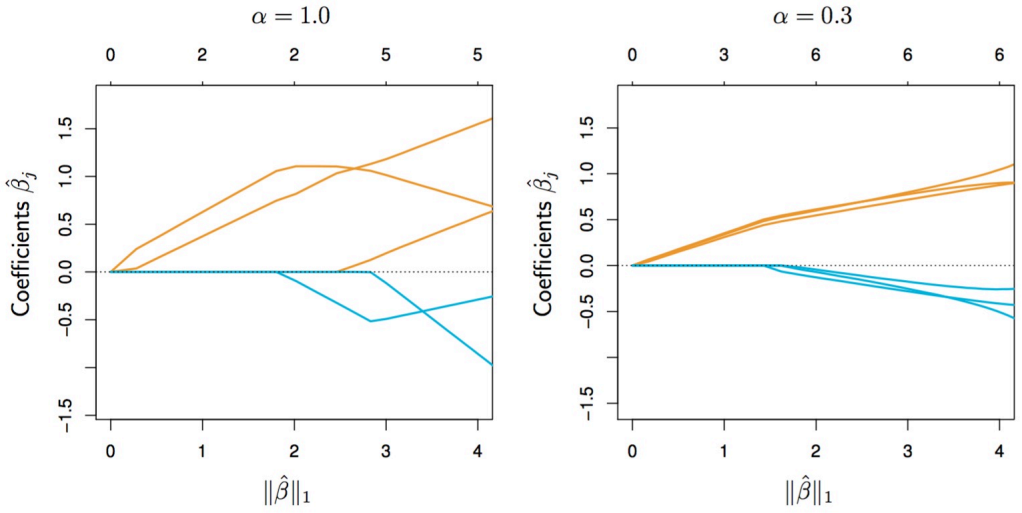


Figure 7: Six variables, highly correlated in groups of three. The lasso estimates ($\alpha = 1$), as shown in the left panel, exhibit somewhat erratic behavior as the regularization parameter λ is varied. In the right panel, the elastic net with ($\alpha = 0.3$) includes all the variables, and the correlated groups are pulled together.

$$
\begin{aligned}
&Z_1,\ Z_2 \sim N(0,1) \text{ independent,} \\
&\quad Y = 3 \cdot Z_1 - 1.5 Z_2 + 2\varepsilon, \text{ with } \varepsilon \sim N(0,1), \\
&\quad X_j = Z_1 + \xi_j/5, \text{ with } \xi_j \sim N(0,1) \text{ for } j = 1,2,3, \text{ and} \\
&\quad X_j = Z_2 + \xi_j/5, \text{ with } \xi_j \sim N(0,1) \text{ for } j = 4,5,6.
\end{aligned}
$$

As shown in the left panel of Figure 7, the lasso coefficients do not reflect the relative importance of the individual variables.

The **elastic net** makes a compromise between the ridge and the lasso penalties (Zou and Hastie 2005); it solves the convex program

$$
\min_{(\beta_0,\boldsymbol{\beta})} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \left[\frac{1}{2}(1-\alpha)\|\boldsymbol{\beta}\|_2^2 + \alpha \|\boldsymbol{\beta}\|_1 \right] \right\}, \qquad (\lambda > 0)
$$

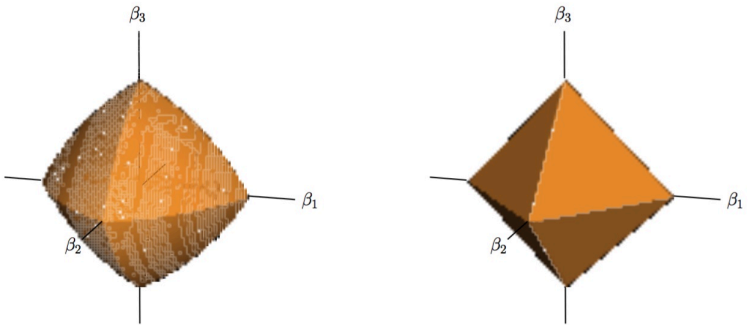


Figure 8: The elastic-net ball with $\alpha = 0.7$ (left panel) in $\mathbb{R}^3$, compared to the ℓ_1 ball (right panel). The curved contours encourage strongly correlated variables to share coefficients (see **Homework 2** for details).

where $\alpha \in [0, 1]$ is a parameter that can be varied. By construction, the penalty applied to an individual coefficient (disregarding the regularization weight $\lambda > 0$) is given by

$$\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha|\beta_j|.$$

When $\alpha = 1$, it is lasso. When $\alpha = 0$, it is ridge.

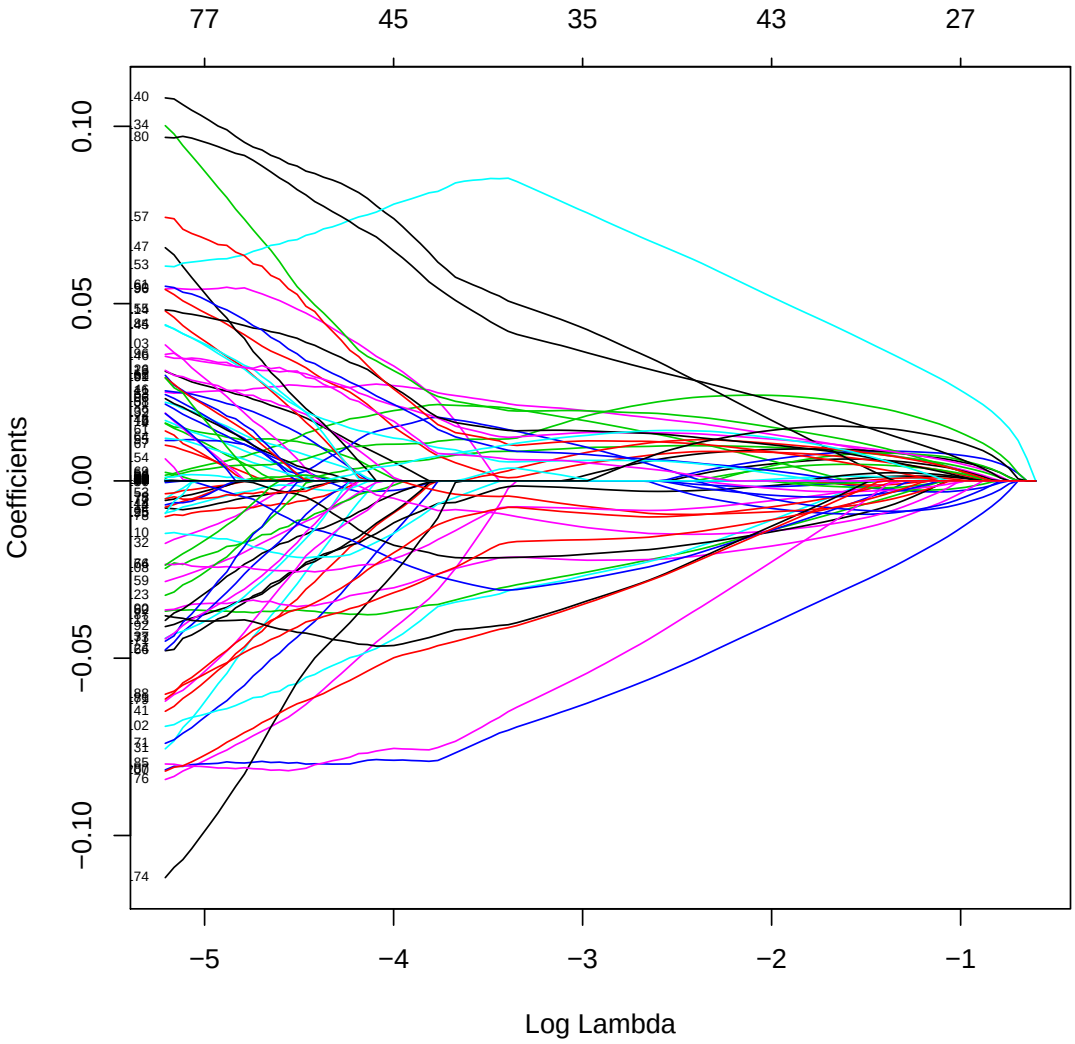
In the right-hand panel of Figure 7, it is the elastic-net coefficient path with $\alpha = 0.3$. We see that

- The coefficients are selected approximately together in their groups.
- The coefficients approximately share their values equally.

```
################################################# Elastic net example


# glmnet provides various options for users to customize the
```

```
# fit. We introduce some commonly used options here and they
# can be specified in the glmnet function.

# [family='gaussian'] is the default family option in the
# function glmnet. 'gaussian'

# [alpha] is for the elastic-net mixing parameter alpha, with
# range alpha in [0,1]. alpha=1 is the lasso (default) and
# alpha=0 is the ridge.

# [nlambda] is the number of lambda values in the sequence.
# Default is 100.


# As an example, we set alpha=0.2 (more like a ridge
# regression), and give double weights to the latter half of
# the observations.  To avoid too long a display here, we set
# nlambda to 20.  In practice, however, the number of values
# of lambda is recommended to be 100 (default) or more. In
# most cases, it does not come with extra cost because of the
# warm-starts used in the algorithm, and for nonlinear models
# leads to better convergence properties.
```

```r
fit = glmnet(x, y, alpha = 0.2, family = "gaussian")


plot(fit, xvar = "lambda", label = TRUE)
```



6.2 The Group Lasso

There are many regression problems in which the covariates have a natural group structure, and it is desirable to have all coefficients within a group become nonzero (or zero) simultaneously. e.g. dummy variables from the same categorical variable should be grouped together.

Consider a linear regression model involving J groups of covariates, where for $j = 1, \ldots, J$, the sub-vector $\mathbf{Z}_j \in \mathbb{R}^{p_j}$ represents the covariates in group j. Our goal is to predict a real-valued response $Y \in \mathbb{R}$ based on the collection of covariates $(\mathbf{Z}_1, \ldots, \mathbf{Z}_J)$. Let $\boldsymbol{\theta}_j \in \mathbb{R}^{p_j}$ be the sub-vector of the regression coefficient for group j.

Given n samples $\{(y_i, \mathbf{z}_{i1}, \ldots, \mathbf{z}_{iJ})\}_{i=1}^n$, the **group lasso** solves the convex problem

$$\min_{(\beta_0, \boldsymbol{\beta})} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \theta_0 - \sum_{j=1}^{J} \mathbf{z}_{ij}^\top \boldsymbol{\theta}_j)^2 + \lambda \sum_{j=1}^{J} \|\boldsymbol{\theta}_j\|_2 \right\},$$

where $\|\boldsymbol{\theta}_j\|_2$ is the Euclidean norm of the vector $\boldsymbol{\theta}_j$.

- depending on λ, either the entire vector $\hat{\boldsymbol{\theta}}_j$ will be zero, or all its elements will be nonzero;
- when $p_j = 1$, the problem reduces to the ordinary lasso.

Figure 9 compares the constraint region for the group lasso (left image) to that of the lasso (right image) when there are three variables.

```
################################################ Group lasso example


install.packages("gglasso", repos = "http://cran.us.r-project.org")
```

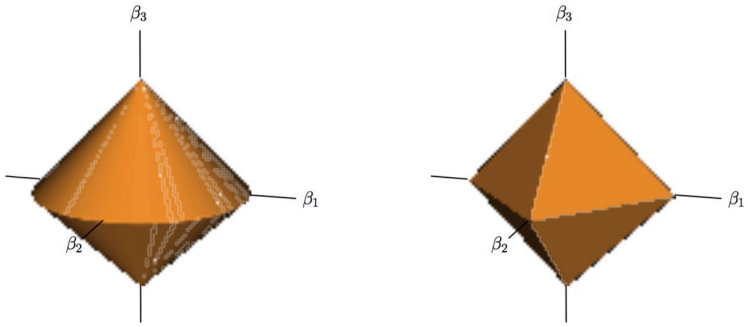
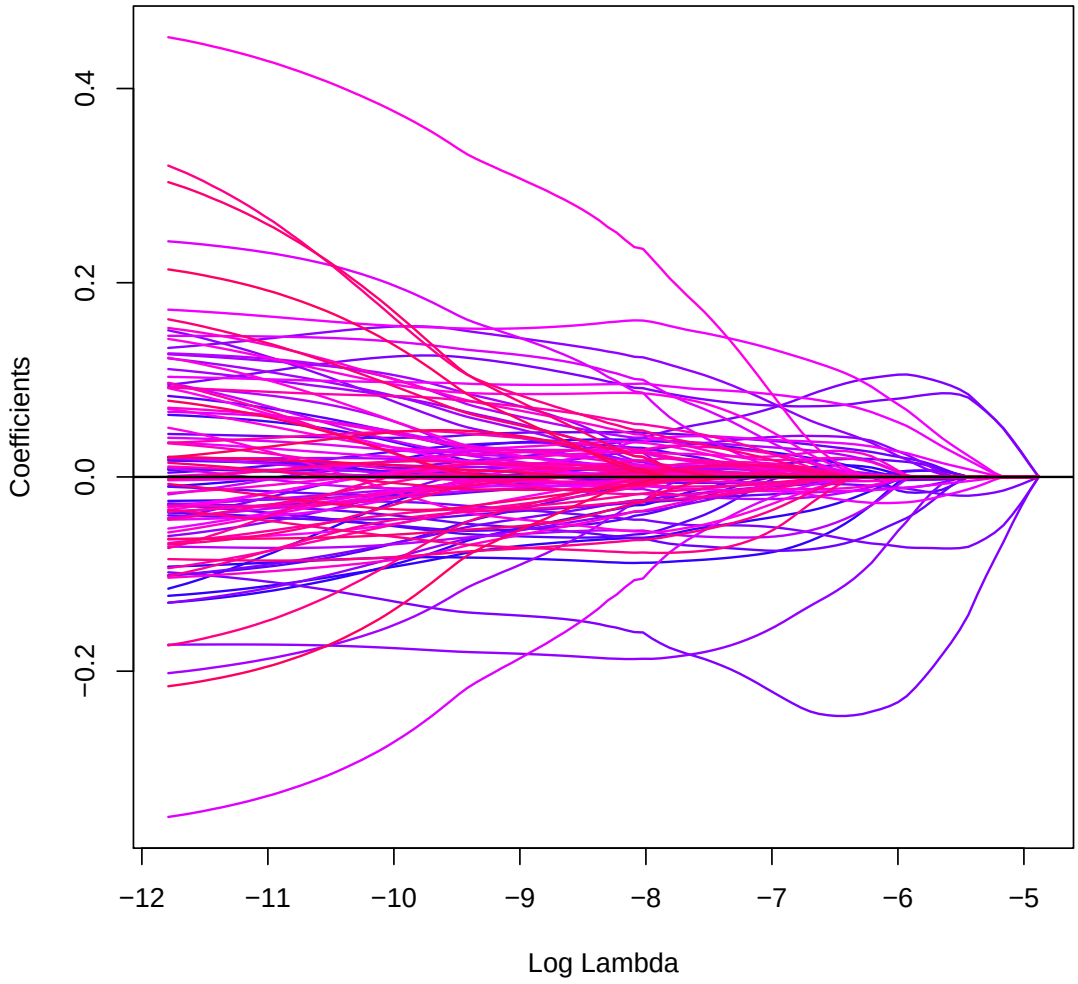


Figure 9: The group lasso ball (left panel) in $\mathbb{R}^3$, compared to the ℓ_1 (right panel). In this case, there are two groups with coefficients $\boldsymbol{\theta}_1 = (\beta_1, \beta_2) \in \mathbb{R}^2$ and $\boldsymbol{\theta}_2 = \beta_3 \in \mathbb{R}^1$.

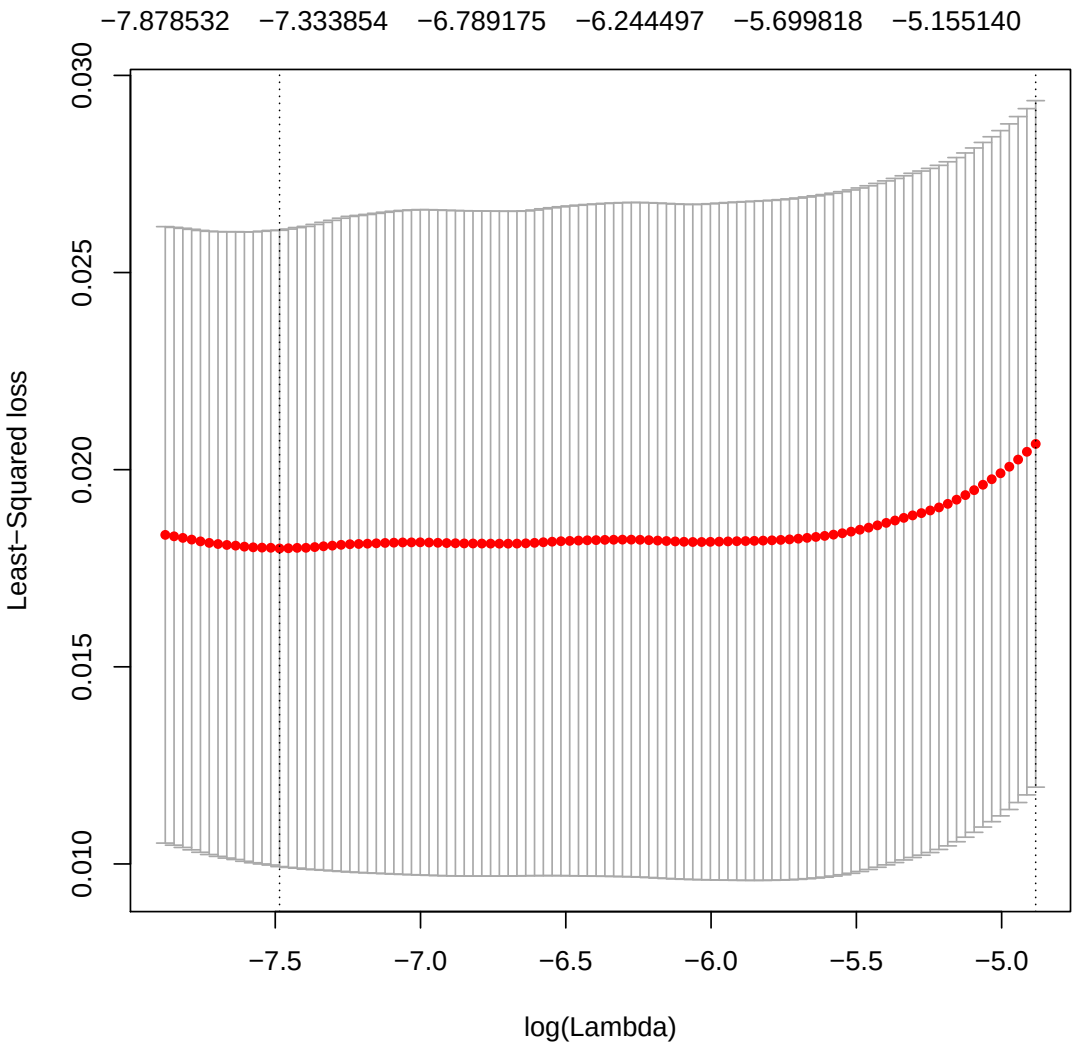
```
##

## The downloaded binary packages are in

##  /var/folders/_n/l1tgzpfx4wz54hhf5yw1x3r80000gn/T//Rtmp9KyIfX/downloaded_packages


# load gglasso library

library(gglasso)


# load bardet data set

data(bardet)


# define 20 groups

group1 <- rep(1:20, each = 5)


# fit group lasso penalized least squares

m1 <- gglasso(x = bardet$x, y = bardet$y, group = group1, loss = "ls")
```

```
plot(m1)
```

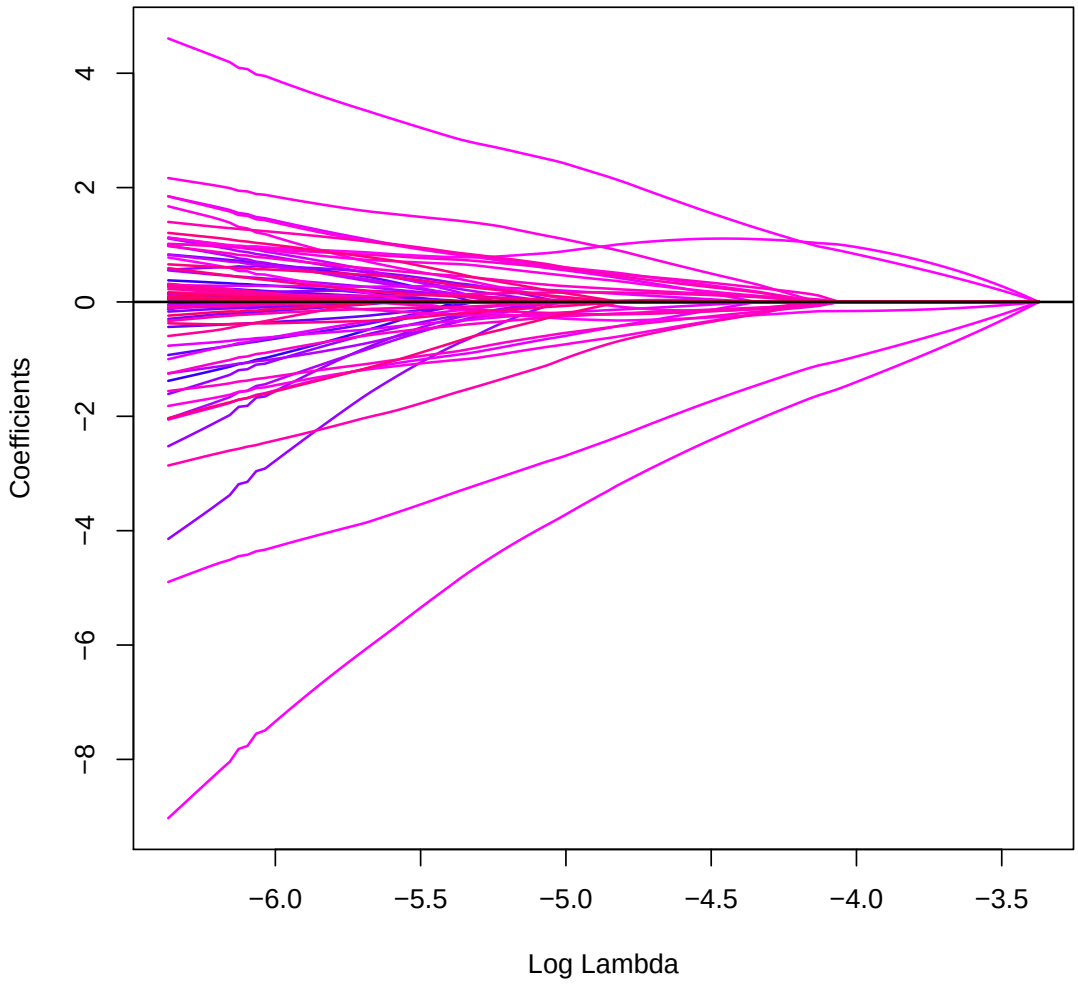



```
# 5-fold cross validation using group lasso penalized ls
# regression
```

```r
cv <- cv.gglasso(x = bardet$x, y = bardet$y, group = group1,
    loss = "ls", pred.loss = "L2", lambda.factor = 0.05, nfolds = 5)


plot(cv)
```

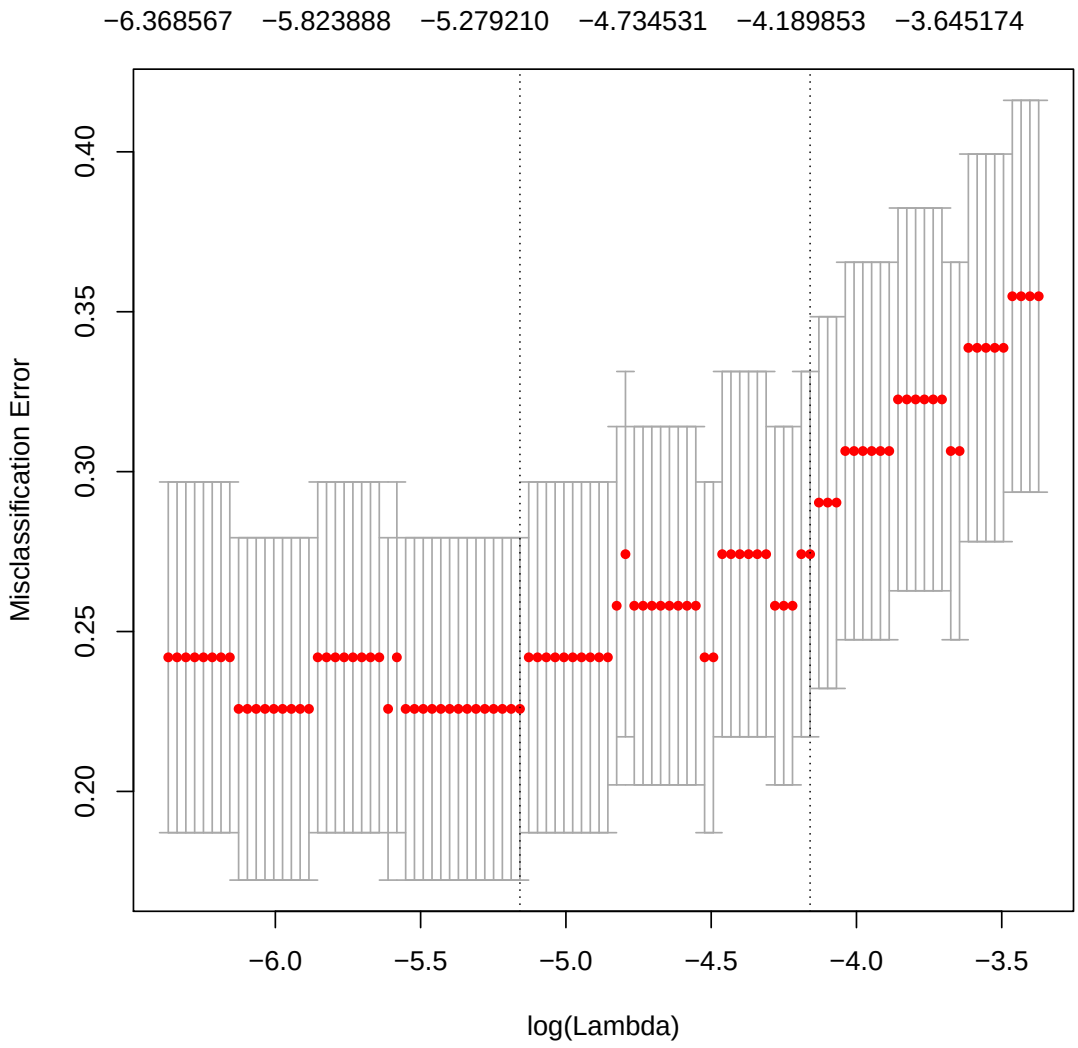


```r
# load colon data set
data(colon)


# define group index
group2 <- rep(1:20, each = 5)


# fit group lasso penalized logistic regression
m2 <- gglasso(x = colon$x, y = colon$y, group = group2, loss = "logit")


plot(m2)
```




```
# 5-fold cross validation using group lasso penalized
# logisitic regression
cv2 <- cv.gglasso(x = colon$x, y = colon$y, group = group2, loss = "logit",
    pred.loss = "misclass", lambda.factor = 0.05, nfolds = 5)
```

```
plot(cv2)
```




```
# the coefficients at lambda = lambda.1se

pre = coef(cv$gglasso.fit, s = cv$lambda.1se)
```

6.3 Sparse Group Lasso

When a group is included in a group-lasso fit, all the coefficients in that group are nonzero. This is a consequence of the ℓ_2 norm. Sometimes we would like sparsity both with respect to which groups are selected, and which coefficients are nonzero within a group. For example, although a biological pathway may be implicated in the progression of a particular type of cancer, not all genes in the pathway need be active. The **sparse group lasso** is designed to achieve such within-group sparsity.

In order to achieve within-group sparsity, we augment the basic group lasso with an additional ℓ_1-penalty, leading to the convex program

$$\min_{(\beta_0,\boldsymbol{\beta})} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \theta_0 - \sum_{j=1}^{J} \mathbf{z}_{ij}^{\top} \boldsymbol{\theta}_j)^2 + \lambda \sum_{j=1}^{J} \left[(1-\alpha) \|\boldsymbol{\theta}_j\|_2 + \alpha \|\boldsymbol{\theta}_j\|_1 \right] \right\},$$

with $\alpha \in [0, 1]$. The parameter – creates a bridge between the group lasso ($\alpha = 0$) and the lasso ($\alpha = 1$). Figure 10 contrasts the group lasso constraint region with that of the sparse group lasso for the case of three variables. Note that in the two horizontal axes, the constraint region resembles that of the elastic net.

6.4 Simulation demo

We generate the data $\{X, Y\}$ from the linear model $Y = X^T\beta + \epsilon$, where ϵ has Normal distribution with mean 0 and variance σ^2. X is generated from multivariate normal distribution with mean 0. In all the examples, the correlation of X_i and X_j is $\rho^{|i-j|}$, where ρ varies within $[0, 1]$.

Example 1. $n = 100$, $p = 200$,$\beta = (4, 4, 4, -6\sqrt{2}, \frac{3}{4}, 0, ..., 0)$.

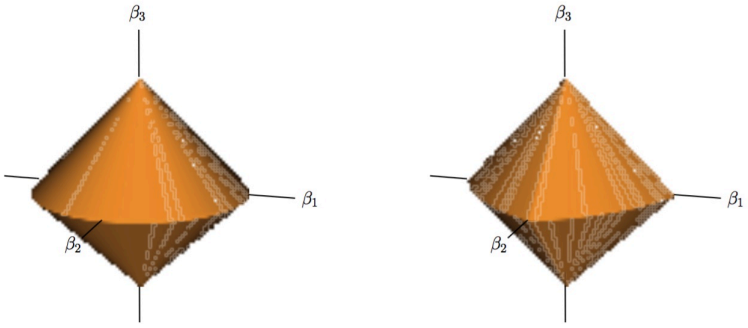
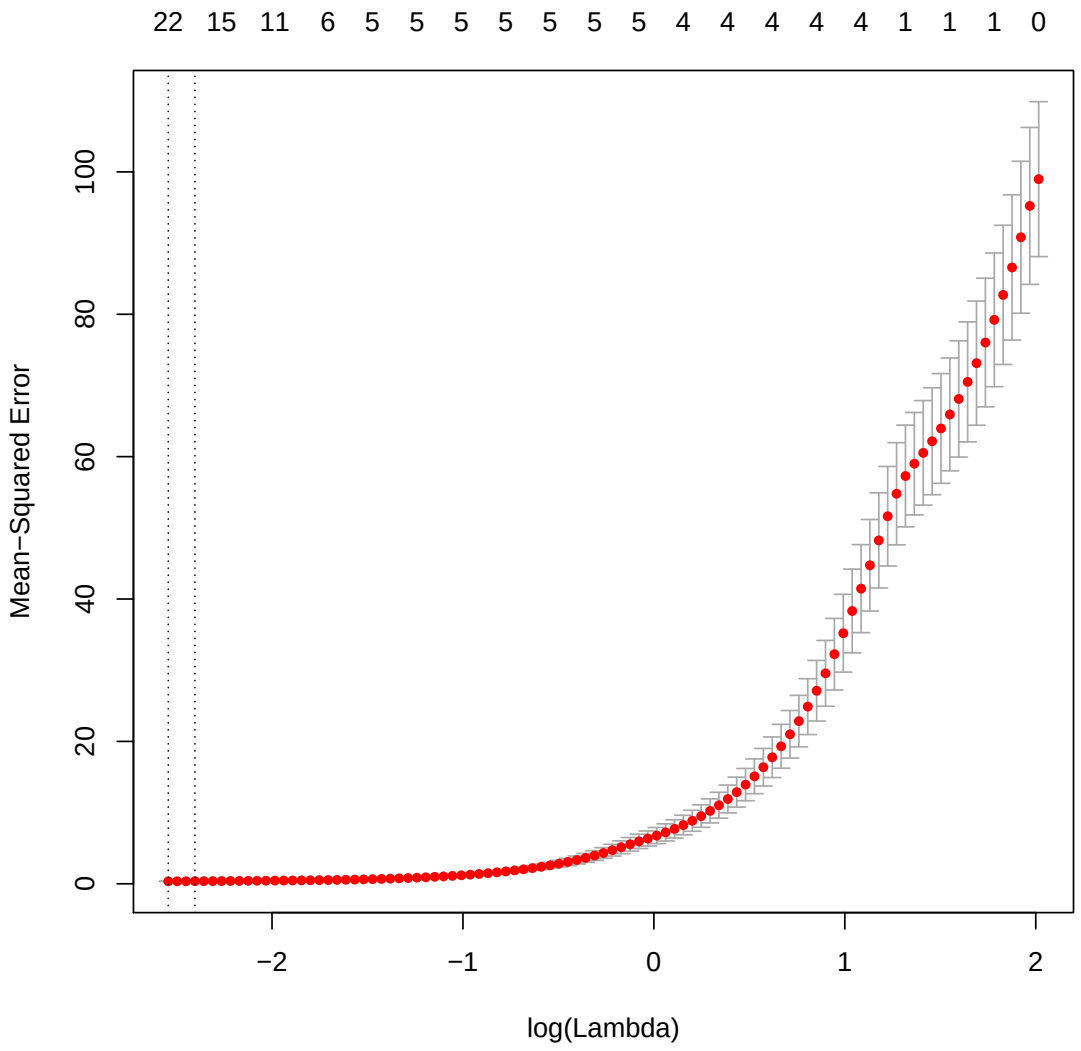


Figure 10: The group lasso ball (left panel) in $\mathbb{R}^3$, compared to the sparse group- lasso ball with $\alpha = 0.5$ (right panel). Depicted are two groups with coefficients $\boldsymbol{\theta}_1 = (\beta_1, \beta_2) \in \mathbb{R}^2$ and $\boldsymbol{\theta}_2 = \beta_3 \in \mathbb{R}^1$.

```
# Compare lasso, adaptive lasso, scad and mcp in the logistic
# regression and least squares load R libraries glmnet: for
# LASSO, adaptive LASSO, elastic net penalized least squares
# and logistic regression (all supports poisson, multinomial
# and cox model) ncvreg: for SCAD and MCP penalized least
# squares and logistic regression
library(glmnet)
library(ncvreg)
library(MASS)


############################## PART I Least squares


n = 100
p = 200
```

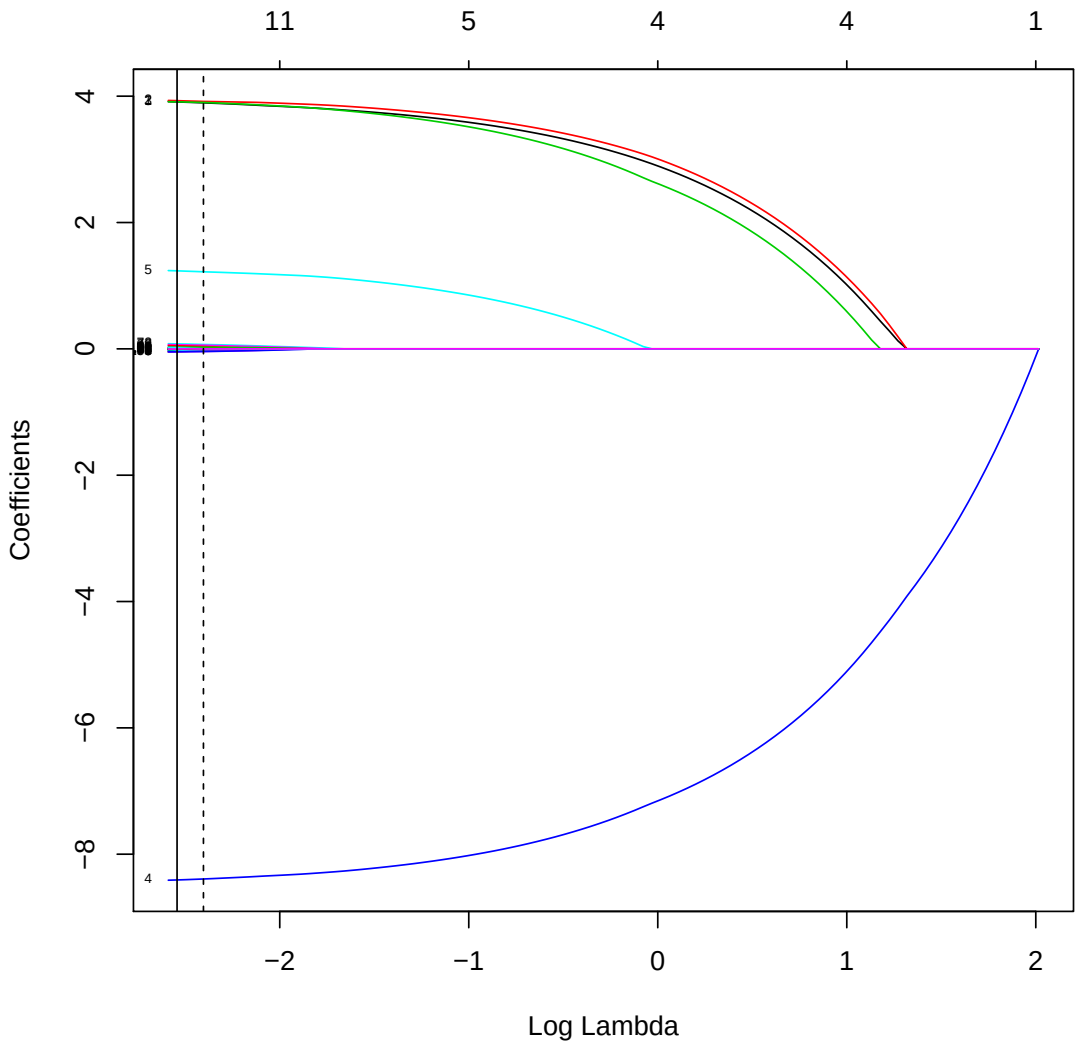
```r
# true beta
truebeta <- c(4, 4, 4, -6 * sqrt(2), 4/3, rep(0, p - 5))

# error variance
sigma2 <- 0.3

# The covariance between Xj and Xk is cov(X_j, X_k) =
# rho^|i-j| covariance matrix
covmat <- function(rho, p) {
    rho^(abs(outer(seq(p), seq(p), "-")))
}
# rho = 0.1, generate covariance matrix for X
sigma <- covmat(0.1, p)

# X ~ N(0, sigma) epsilon ~ N(0, sigma2) The true model: y =
# x*truebeta + epsilon
x <- mvrnorm(n, rep(0, p), sigma)
epsilon <- rnorm(n, 0, sd = sqrt(sigma2))
y <- x %*% truebeta + epsilon

# fit lasso and use five-fold CV to select lambda
cvfit <- cv.glmnet(x = x, y = y, alpha = 1, family = "gaussian")
```

```
plot(cvfit)
```




```
# lambda selected by CV

cvfit$lambda.min


## [1] 0.079
```

```r
# plot the solution paths
plot(cvfit$glmnet.fit, label = TRUE, xvar = "lambda")
# plot lambda.min
abline(v = log(cvfit$lambda.min), lty = 1)
# plot lambda.1se
abline(v = log(cvfit$lambda.1se), lty = 2)
```




```
model_compare <- matrix(NA, nrow = 5, ncol = p, dimnames = list(c("true model",
    "lasso", "adaptive lasso", "mcp", "scad"), paste("V", seq(p),
    sep = "")))
```

```r
# save the true model
model_compare[1, ] <- truebeta



## coefficients estimated by lasso
cvfit <- cv.glmnet(x = x, y = y, alpha = 1, family = "gaussian")
tmp <- cvfit$glmnet.fit$beta
lasso_beta <- as.matrix(tmp[, cvfit$lambda == cvfit$lambda.min])
model_compare[2, ] <- lasso_beta



# Compute weight and fit an adaptive lasso
weight = 1/(lasso_beta)^2
# Some est. coef. is zero, the corresponding weight is Inf to
# prevent numerical error, convert Inf to a large number
# (e.g. 1e6)
weight[weight == Inf] = 1e+06

cvfit <- cv.glmnet(x = x, y = y, alpha = 1, family = "gaussian",
    nfolds = 5, penalty.factor = weight)

## coefficients estimated by adaptive lasso
tmp <- cvfit$glmnet.fit$beta
```

```r
adaptive_lasso_beta <- as.matrix(tmp[, cvfit$lambda == cvfit$lambda.min])
model_compare[3, ] <- adaptive_lasso_beta

## coefficients estimated by mcp
cvfit <- cv.ncvreg(X = x, y = y, penalty = "MCP", family = "gaussian")
mcp_beta <- cvfit$fit$beta[, cvfit$min]
model_compare[4, ] <- mcp_beta[-1]

## coefficients estimated by scad
cvfit <- cv.ncvreg(X = x, y = y, penalty = "SCAD", family = "gaussian")
scad_beta <- cvfit$fit$beta[, cvfit$min]
model_compare[5, ] <- scad_beta[-1]

# make a comparison of the estimated coef. from four methods
# we see that lasso over-selected, adaptive lasso, scad and
# mcp fix the problem.

model_compare
##                 V1  V2  V3   V4  V5 V6    V7 V8 V9    V10
## true model     4.0 4.0 4.0 -8.5 1.3  0  0.00  0  0  0.000
## lasso          3.9 3.9 3.9 -8.4 1.2  0 -0.02  0  0 -0.015
## adaptive lasso 3.6 3.7 3.4 -8.2 0.0  0  0.00  0  0  0.000
## mcp            4.0 4.0 4.0 -8.5 1.4  0  0.00  0  0  0.000
```

```
## scad           4.0 4.0 4.0 -8.5 1.3  0  0.00  0  0  0.000
##                V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21
## true model       0   0   0   0   0   0   0   0   0   0   0
## lasso            0   0   0   0   0   0   0   0   0   0   0
## adaptive lasso   0   0   0   0   0   0   0   0   0   0   0
## mcp              0   0   0   0   0   0   0   0   0   0   0
## scad             0   0   0   0   0   0   0   0   0   0   0
##                V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32
## true model       0   0   0   0   0   0   0   0   0   0   0
## lasso            0   0   0   0   0   0   0   0   0   0   0
## adaptive lasso   0   0   0   0   0   0   0   0   0   0   0
## mcp              0   0   0   0   0   0   0   0   0   0   0
## scad             0   0   0   0   0   0   0   0   0   0   0
##                V33 V34 V35 V36 V37 V38 V39 V40 V41 V42 V43
## true model       0   0   0   0   0   0   0   0   0   0   0
## lasso            0   0   0   0   0   0   0   0   0   0   0
## adaptive lasso   0   0   0   0   0   0   0   0   0   0   0
## mcp              0   0   0   0   0   0   0   0   0   0   0
## scad             0   0   0   0   0   0   0   0   0   0   0
##                V44 V45 V46 V47 V48 V49 V50 V51 V52 V53 V54
## true model       0   0   0   0   0   0   0   0   0   0   0
## lasso            0   0   0   0   0   0   0   0   0   0   0
## adaptive lasso   0   0   0   0   0   0   0   0   0   0   0
```

```
## mcp              0   0   0   0   0   0   0   0   0   0   0
## scad             0   0   0   0   0   0   0   0   0   0   0
##                V55 V56 V57 V58 V59   V60 V61 V62 V63 V64
## true model       0   0   0   0   0 0.000   0   0   0   0
## lasso            0   0   0   0   0 0.013   0   0   0   0
## adaptive lasso   0   0   0   0   0 0.000   0   0   0   0
## mcp              0   0   0   0   0 0.000   0   0   0   0
## scad             0   0   0   0   0 0.000   0   0   0   0
##                V65 V66 V67 V68 V69 V70 V71   V72 V73 V74
## true model       0   0   0   0   0   0   0 0.000   0   0
## lasso            0   0   0   0   0   0   0 0.076   0   0
## adaptive lasso   0   0   0   0   0   0   0 0.000   0   0
## mcp              0   0   0   0   0   0   0 0.000   0   0
## scad             0   0   0   0   0   0   0 0.000   0   0
##                V75   V76 V77    V78 V79 V80 V81 V82 V83 V84
## true model       0 0.000   0  0.000   0   0   0   0   0   0
## lasso            0 0.062   0 -0.038   0   0   0   0   0   0
## adaptive lasso   0 0.000   0  0.000   0   0   0   0   0   0
## mcp              0 0.000   0  0.000   0   0   0   0   0   0
## scad             0 0.000   0  0.000   0   0   0   0   0   0
##                V85 V86 V87 V88 V89 V90 V91 V92   V93 V94
## true model       0   0   0   0   0   0   0   0 0.000   0
## lasso            0   0   0   0   0   0   0   0 0.045   0
```

```
## adaptive lasso   0   0   0   0   0   0   0   0 0.000   0
## mcp              0   0   0   0   0   0   0   0 0.000   0
## scad             0   0   0   0   0   0   0   0 0.000   0
##                V95 V96   V97 V98 V99 V100 V101 V102 V103
## true model       0   0 0.000   0   0    0    0    0    0
## lasso            0   0 0.026   0   0    0    0    0    0
## adaptive lasso   0   0 0.000   0   0    0    0    0    0
## mcp              0   0 0.000   0   0    0    0    0    0
## scad             0   0 0.000   0   0    0    0    0    0
##                V104 V105 V106 V107 V108 V109 V110 V111 V112
## true model        0    0    0    0    0    0    0    0    0
## lasso             0    0    0    0    0    0    0    0    0
## adaptive lasso    0    0    0    0    0    0    0    0    0
## mcp               0    0    0    0    0    0    0    0    0
## scad              0    0    0    0    0    0    0    0    0
##                V113 V114 V115 V116 V117 V118 V119 V120 V121
## true model        0    0    0    0    0    0    0    0    0
## lasso             0    0    0    0    0    0    0    0    0
## adaptive lasso    0    0    0    0    0    0    0    0    0
## mcp               0    0    0    0    0    0    0    0    0
## scad              0    0    0    0    0    0    0    0    0
##                V122 V123 V124 V125 V126 V127 V128 V129 V130
## true model        0    0    0    0    0    0    0    0    0
```

```
## lasso             0    0    0    0    0    0    0    0    0
## adaptive lasso    0    0    0    0    0    0    0    0    0
## mcp               0    0    0    0    0    0    0    0    0
## scad              0    0    0    0    0    0    0    0    0
##                 V131 V132 V133 V134 V135 V136 V137 V138
## true model      0.00    0    0    0    0    0    0    0
## lasso          -0.02    0    0    0    0    0    0    0
## adaptive lasso  0.00    0    0    0    0    0    0    0
## mcp             0.00    0    0    0    0    0    0    0
## scad            0.00    0    0    0    0    0    0    0
##                V139 V140 V141 V142 V143 V144 V145 V146 V147
## true model        0    0    0    0    0    0    0    0    0
## lasso             0    0    0    0    0    0    0    0    0
## adaptive lasso    0    0    0    0    0    0    0    0    0
## mcp               0    0    0    0    0    0    0    0    0
## scad              0    0    0    0    0    0    0    0    0
##                V148 V149 V150 V151 V152 V153 V154 V155
## true model        0    0    0    0    0    0    0    0
## lasso             0    0    0    0    0    0    0    0
## adaptive lasso    0    0    0    0    0    0    0    0
## mcp               0    0    0    0    0    0    0    0
## scad              0    0    0    0    0    0    0    0
##                  V156 V157    V158 V159 V160 V161 V162
```

```
## true model      0.000    0  0.0000    0    0    0    0
## lasso          -0.045    0 -0.0027    0    0    0    0
## adaptive lasso  0.000    0  0.0000    0    0    0    0
## mcp             0.000    0  0.0000    0    0    0    0
## scad            0.000    0  0.0000    0    0    0    0
##                   V163 V164 V165 V166 V167 V168 V169 V170
## true model      0.0000    0    0    0    0    0    0    0
## lasso          -0.0011    0    0    0    0    0    0    0
## adaptive lasso  0.0000    0    0    0    0    0    0    0
## mcp             0.0000    0    0    0    0    0    0    0
## scad            0.0000    0    0    0    0    0    0    0
##                V171 V172 V173 V174 V175 V176 V177   V178
## true model        0    0    0    0    0    0    0  0.000
## lasso             0    0    0    0    0    0    0 -0.046
## adaptive lasso    0    0    0    0    0    0    0  0.000
## mcp               0    0    0    0    0    0    0  0.000
## scad              0    0    0    0    0    0    0  0.000
##                V179 V180  V181 V182 V183 V184 V185 V186
## true model        0    0 0.000    0    0    0    0    0
## lasso             0    0 0.012    0    0    0    0    0
## adaptive lasso    0    0 0.000    0    0    0    0    0
## mcp               0    0 0.000    0    0    0    0    0
## scad              0    0 0.000    0    0    0    0    0
```

```
##                V187 V188 V189 V190 V191 V192 V193 V194 V195
## true model        0    0    0    0    0    0    0    0    0
## lasso             0    0    0    0    0    0    0    0    0
## adaptive lasso    0    0    0    0    0    0    0    0    0
## mcp               0    0    0    0    0    0    0    0    0
## scad              0    0    0    0    0    0    0    0    0
##                V196 V197 V198 V199 V200
## true model        0    0    0    0    0
## lasso             0    0    0    0    0
## adaptive lasso    0    0    0    0    0
## mcp               0    0    0    0    0
## scad              0    0    0    0    0
```

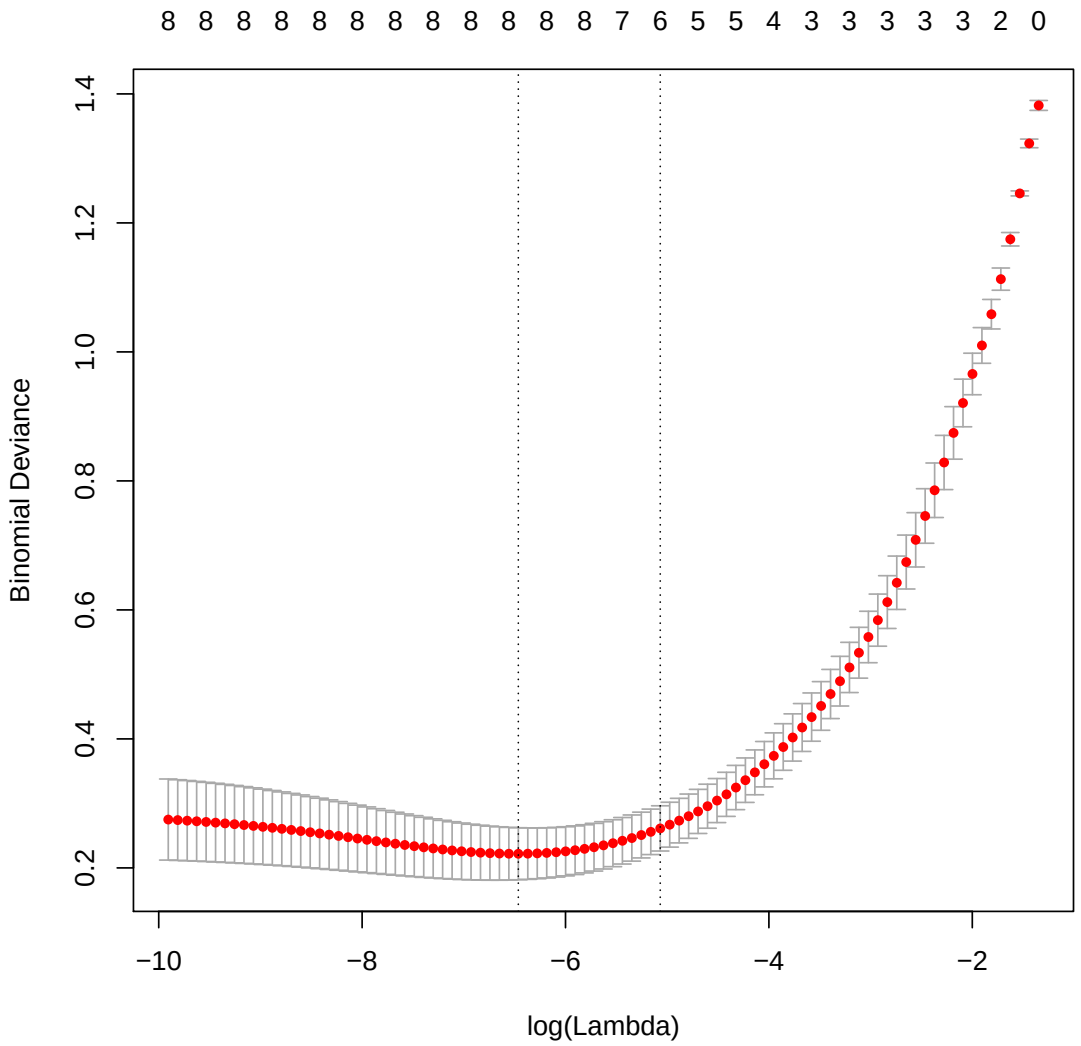
We simulate data from the logistic model. The response y was generated independently from the Bernoulli distribution with conditional success probability as $p(Y = 1|X) = \frac{\exp(X'\beta)}{1+\exp(X'\beta)}$.

Example 2. $n = 200, p = 8$, chose the true regression coefficients vector $\beta = (6, 3.5, 0, 0, 5, 0, 0, 0)^\intercal$. The rows of X were sampled as i.i.d. copies from $N_p(0, \boldsymbol{I})$.

```
############################### PART II logistic regression



# generate data
```

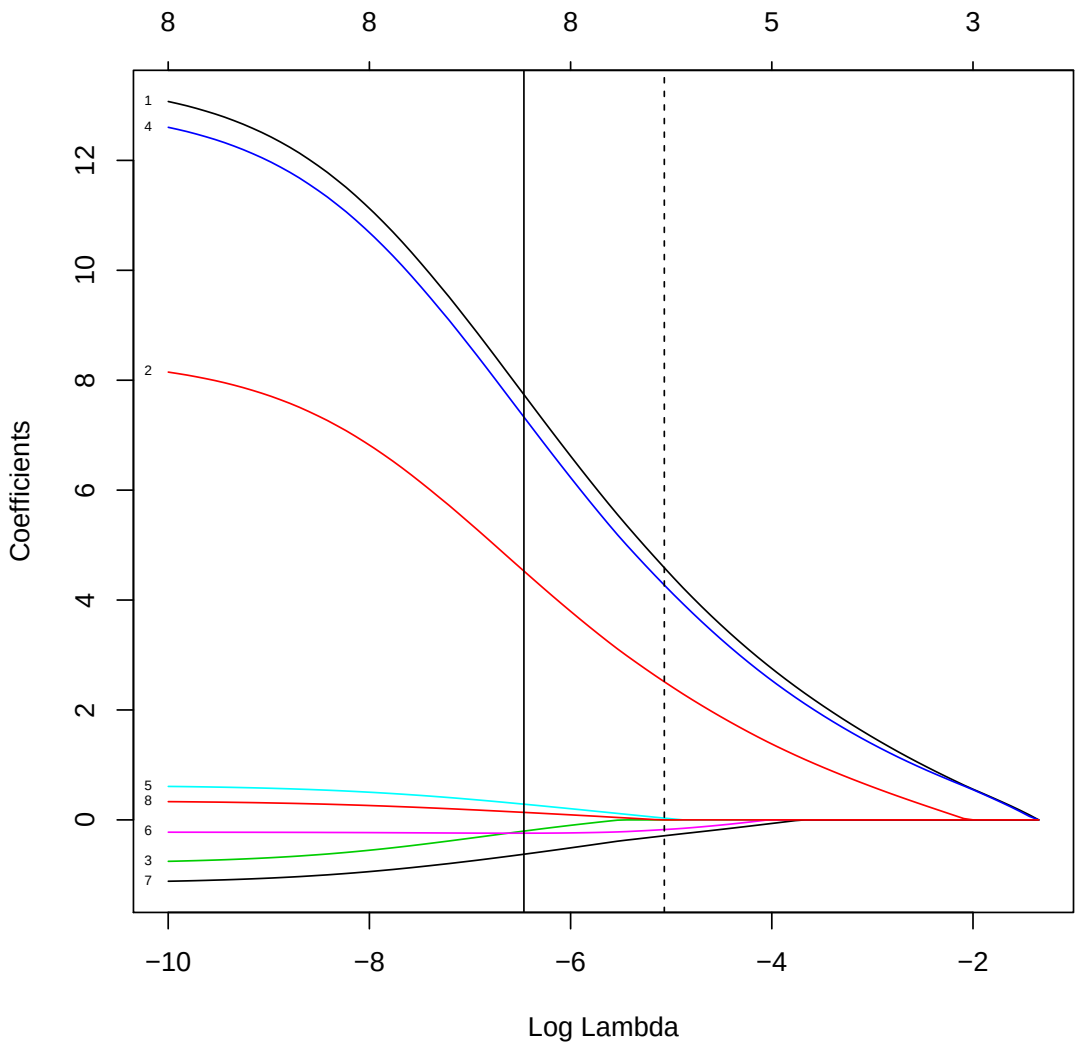
```r
n <- 200
p <- 8

# truebeta
truebeta <- c(6, 3.5, 0, 5, rep(0, p - 4))
truebeta

## [1] 6.0 3.5 0.0 5.0 0.0 0.0 0.0 0.0

# generate x and y from the true model The true model:
# P(y=1|x) = exp(x*truebeta)/(exp(x*truebeta)+1)
x <- matrix(rnorm(n * p), n, p)
feta <- x %*% truebeta
fprob <- ifelse(feta < 0, exp(feta)/(1 + exp(feta)), 1/(1 + exp(-feta)))
y <- rbinom(n, 1, fprob)


model_compare <- matrix(NA, nrow = 5, ncol = p, dimnames = list(c("true model",
    "lasso", "adaptive lasso", "mcp", "scad"), paste("V", seq(p),
    sep = "")))


# save the true model
model_compare[1, ] <- truebeta
```

```r
# lasso case cv.glmfit fit lasso model and use cross
# validation for lambda selection

# family = 'binomial', logistic regression family =
# 'gaussian', least squares

# alpha controls the degree of L1 penalty term, alpha = 1,
# lasso, alpha = 0, ridge, alpha = (0,1), elastic net

# nfolds = 5, five-fold cross validation (CV)
cvfit <- cv.glmnet(x = x, y = y, alpha = 1, family = "binomial",
    nfolds = 5)

# make a plot of the CV result the left vertical line
# (lambda.min) correspondes to the lambda that gives smallest
# deviance.  the right vertical line (lambda.1se)
# correspondes to the lambda from the one standard deviation
# rule
plot(cvfit)
```




```r
# plot the solution paths
plot(cvfit$glmnet.fit, label = TRUE, xvar = "lambda")
# plot lambda.min
abline(v = log(cvfit$lambda.min), lty = 1)
```

```r
# plot lambda.1se
abline(v = log(cvfit$lambda.1se), lty = 2)
```



```r
# save the Lasso coefficient from solution path, each column
# represents an estimate for a lambda value
tmp <- cvfit$glmnet.fit$beta
tmp
```

```
## 8 x 94 sparse Matrix of class "dgCMatrix"
##
## V1 . 0.091 0.18 0.26 0.33 0.41 0.48 0.56 0.633 0.714 0.80
## V2 . .     .    .    .    .    .    .    0.023 0.081 0.14
## V3 . .     .    .    .    .    .    .    .     .     .
## V4 . 0.065 0.15 0.24 0.32 0.40 0.47 0.55 0.619 0.688 0.76
## V5 . .     .    .    .    .    .    .    .     .     .
## V6 . .     .    .    .    .    .    .    .     .     .
## V7 . .     .    .    .    .    .    .    .     .     .
## V8 . .     .    .    .    .    .    .    .     .     .
##
## V1 0.88 0.97 1.05 1.15 1.24 1.33 1.43 1.53 1.63 1.74 1.85
## V2 0.19 0.25 0.31 0.37 0.43 0.49 0.55 0.62 0.68 0.75 0.81
## V3 .    .    .    .    .    .    .    .    .    .    .
## V4 0.83 0.90 0.98 1.06 1.14 1.22 1.31 1.40 1.49 1.59 1.69
## V5 .    .    .    .    .    .    .    .    .    .    .
## V6 .    .    .    .    .    .    .    .    .    .    .
## V7 .    .    .    .    .    .    .    .    .    .    .
## V8 .    .    .    .    .    .    .    .    .    .    .
```

```
##
## V1 1.96 2.07 2.2 2.3  2.428  2.555  2.686  2.820  2.958
## V2 0.88 0.95 1.0 1.1  1.177  1.256  1.338  1.421  1.509
## V3 .    .    .   .    .      .      .      .      .
## V4 1.79 1.89 2.0 2.1  2.229  2.348  2.471  2.598  2.730
## V5 .    .    .   .    .      .      .      .      .
## V6 .    .    .   .    .      .      .     -0.002 -0.022
## V7 .    .    .   .   -0.013 -0.034 -0.055 -0.075 -0.095
## V8 .    .    .   .    .      .      .      .      .
##
## V1  3.101  3.25  3.399  3.555  3.72  3.88  4.05  4.22
## V2  1.598  1.69  1.784  1.880  1.98  2.08  2.18  2.29
## V3  .      .     .      .      .     .     .     .
## V4  2.865  3.00  3.149  3.297  3.45  3.61  3.77  3.93
## V5  .      .     .      .      .     .     .     .
## V6 -0.042 -0.06 -0.078 -0.095 -0.11 -0.13 -0.14 -0.15
## V7 -0.116 -0.14 -0.155 -0.175 -0.19 -0.21 -0.23 -0.25
## V8  .      .     .      .      .     .     .     .
##
## V1  4.403  4.588  4.7755  4.965  5.159  5.355  5.5556
## V2  2.401  2.513  2.6281  2.745  2.865  2.986  3.1108
## V3  .      .      .       .      .      .     -0.0051
## V4  4.099  4.272  4.4478  4.627  4.810  4.996  5.1879
```

```
## V5  0.016  0.032  0.0488  0.066  0.083  0.100  0.1174
## V6 -0.166 -0.176 -0.1862 -0.196 -0.205 -0.213 -0.2193
## V7 -0.270 -0.289 -0.3088 -0.329 -0.348 -0.367 -0.3879
## V8  .      .      0.0051  0.015  0.025  0.034  0.0436
##
## V1  5.761  5.971  6.183  6.399  6.618  6.84  7.06  7.29
## V2  3.242  3.376  3.512  3.651  3.793  3.94  4.08  4.23
## V3 -0.023 -0.041 -0.060 -0.079 -0.099 -0.12 -0.14 -0.16
## V4  5.388  5.593  5.801  6.012  6.227  6.44  6.66  6.88
## V5  0.134  0.152  0.169  0.186  0.203  0.22  0.24  0.25
## V6 -0.224 -0.228 -0.232 -0.235 -0.237 -0.24 -0.24 -0.24
## V7 -0.412 -0.437 -0.461 -0.485 -0.509 -0.53 -0.56 -0.58
## V8  0.053  0.063  0.073  0.082  0.091  0.10  0.11  0.12
##
## V1  7.51  7.74  7.96  8.19  8.42  8.64  8.87  9.08  9.30
## V2  4.38  4.53  4.68  4.83  4.98  5.13  5.28  5.43  5.58
## V3 -0.18 -0.20 -0.23 -0.25 -0.27 -0.29 -0.32 -0.34 -0.36
## V4  7.11  7.33  7.55  7.78  8.00  8.23  8.45  8.66  8.88
## V5  0.27  0.29  0.30  0.32  0.33  0.35  0.36  0.38  0.39
## V6 -0.24 -0.24 -0.24 -0.24 -0.24 -0.24 -0.24 -0.24 -0.24
## V7 -0.60 -0.63 -0.65 -0.67 -0.69 -0.71 -0.73 -0.75 -0.77
## V8  0.13  0.14  0.15  0.15  0.16  0.17  0.18  0.19  0.20
##
```

```
## V1  9.52  9.72  9.93 10.13 10.32 10.51 10.70 10.87 11.04
## V2  5.72  5.86  6.00  6.14  6.27  6.40  6.53  6.65  6.76
## V3 -0.38 -0.41 -0.43 -0.45 -0.47 -0.49 -0.51 -0.53 -0.54
## V4  9.09  9.30  9.50  9.70  9.89 10.08 10.26 10.44 10.60
## V5  0.41  0.42  0.43  0.44  0.46  0.47  0.48  0.49  0.50
## V6 -0.24 -0.24 -0.24 -0.24 -0.24 -0.23 -0.23 -0.23 -0.23
## V7 -0.79 -0.81 -0.83 -0.85 -0.87 -0.89 -0.90 -0.92 -0.93
## V8  0.20  0.21  0.22  0.23  0.23  0.24  0.25  0.25  0.26
##
## V1 11.20 11.36 11.51 11.65 11.78 11.90 12.02 12.13 12.24
## V2  6.87  6.98  7.08  7.17  7.27  7.35  7.43  7.51  7.58
## V3 -0.56 -0.58 -0.59 -0.61 -0.62 -0.63 -0.65 -0.66 -0.67
## V4 10.76 10.92 11.07 11.20 11.33 11.45 11.57 11.68 11.78
## V5  0.51  0.52  0.52  0.53  0.54  0.55  0.55  0.56  0.56
## V6 -0.23 -0.23 -0.23 -0.23 -0.23 -0.23 -0.23 -0.23 -0.23
## V7 -0.95 -0.96 -0.98 -0.99 -1.00 -1.01 -1.02 -1.03 -1.04
## V8  0.26  0.27  0.27  0.28  0.28  0.29  0.29  0.30  0.30
##
## V1 12.33 12.42 12.51 12.58 12.66 12.72 12.79 12.85 12.90
## V2  7.64  7.71  7.76  7.82  7.87  7.91  7.95  7.99  8.03
## V3 -0.68 -0.69 -0.70 -0.70 -0.71 -0.72 -0.73 -0.73 -0.74
## V4 11.88 11.97 12.05 12.12 12.20 12.26 12.32 12.38 12.43
## V5  0.57  0.57  0.58  0.58  0.59  0.59  0.59  0.60  0.60
```

```
## V6 -0.23 -0.23 -0.23 -0.23 -0.23 -0.23 -0.23 -0.23 -0.22
## V7 -1.05 -1.06 -1.07 -1.07 -1.08 -1.09 -1.09 -1.10 -1.10
## V8  0.30  0.31  0.31  0.31  0.32  0.32  0.32  0.32  0.33
##
## V1 12.95 12.99 13.03 13.07
## V2  8.06  8.09  8.12  8.15
## V3 -0.74 -0.75 -0.75 -0.75
## V4 12.48 12.52 12.56 12.60
## V5  0.60  0.60  0.61  0.61
## V6 -0.22 -0.22 -0.22 -0.22
## V7 -1.11 -1.11 -1.11 -1.12
## V8  0.33  0.33  0.33  0.33
```

```
# the beta that correspondes to lambda.min selected by CV
lasso_beta <- as.matrix(tmp[, cvfit$lambda == cvfit$lambda.min])
model_compare[2, ] <- lasso_beta



# Compute weight and fit an adaptive lasso
weight = 1/(lasso_beta)^2
# Some est. coef. is zero, the corresponding weight is Inf to
# prevent numerical error, convert Inf to a large number
# (e.g. 1e6)
weight[weight == Inf] = 1e+06
```

```r
cvfit <- cv.glmnet(x = x, y = y, alpha = 1, family = "binomial",
    nfolds = 5, penalty.factor = weight)

## coefficients estimated by adaptive lasso
tmp <- cvfit$glmnet.fit$beta
adaptive_lasso_beta <- as.matrix(tmp[, cvfit$lambda == cvfit$lambda.min])
model_compare[3, ] <- adaptive_lasso_beta



## coefficients estimated by mcp
cvfit <- cv.ncvreg(X = x, y = y, penalty = "MCP", family = "binomial")
mcp_beta <- cvfit$fit$beta[, cvfit$min]
model_compare[4, ] <- mcp_beta[-1]

## coefficients estimated by scad
cvfit <- cv.ncvreg(X = x, y = y, penalty = "SCAD", family = "binomial")
scad_beta <- cvfit$fit$beta[, cvfit$min]
model_compare[5, ] <- scad_beta[-1]

# make a comparison of the estimated coef. from four methods
model_compare

##                  V1  V2   V3  V4   V5    V6    V7   V8
```

```
## true model      6.0 3.5  0.0 5.0 0.00  0.00  0.00 0.00
## lasso           7.7 4.5 -0.2 7.3 0.29 -0.24 -0.63 0.14
## adaptive lasso  7.7 4.4  0.0 7.3 0.00  0.00  0.00 0.00
## mcp            10.5 6.0  0.0 9.9 0.00  0.00  0.00 0.00
## scad           10.5 6.0  0.0 9.9 0.00  0.00  0.00 0.00
```