Lasso and Variable Selection

Ruoqing Zhu

Last Updated: September 26, 2024

Lasso

Lasso (Tibshirani, 1996) is among the most popular machine learning models. Different from the Ridge regression, its adds ℓ_1 penalty on the fitted parameters:

$$
\begin{aligned}
\widehat{\boldsymbol{\beta}}^{\text{lasso}} &= \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + n\lambda\|\boldsymbol{\beta}\|_1 &(1)\\
&= \arg\min_{\boldsymbol{\beta}} \frac{1}{n}\sum_{i=1}^n (y_i - x_i^{\mathrm{T}}\boldsymbol{\beta})^2 + \lambda\sum_{j=1}^p |\beta_j|, &(2)
\end{aligned}
$$

The main advantage of adding such a penalty is that small $\widehat{\beta}_j$ values can be **shrunk to zero**. This may prevents over-fitting and also improve the interpretability especially when the number of variables is large. This is called the **variable selection property** since only the nonzero parameters are selected be useful. Since we have already used the `glmnet` package before, let's use it as an example for Lasso and then we will analyze the Lasso with a single variable case to see why it can shrink a parameter estimate to be zero. We will also provide discussions on the path-wise coordinate descent algorithm, the relationship with the Ridge regression, and the bias-variance issue.

Using the `glmnet` package

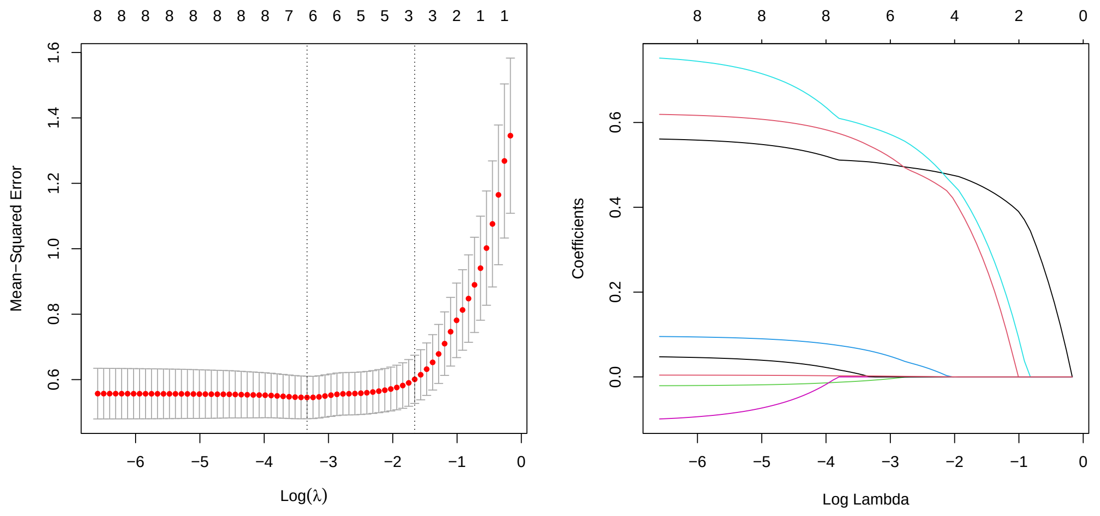
We still use the prostate cancer data `prostate` data. The dataset contains 8 explanatory variables and one outcome `lpsa`, the log prostate-specific antigen value. We fit the model using the `glmnet` package. The tuning parameter can be selected using cross-validation with the `cv.glmnet` function. You can specify `nfolds` for the number of folds in the cross-validation. The default is 10. For Lasso, we should use `alpha = 1`, while `alpha = 0` is for Ridge. However, it is the default value that you do not need to specify.

```r
library(glmnet)
library(ElemStatLearn)
data(prostate)
set.seed(3)
lasso.fit = cv.glmnet(data.matrix(prostate[, 1:8]), prostate$lpsa, nfolds = 10, alpha = 1)
```

The left plot demonstrates how λ changes the cross-validation error. There are two vertical lines, which represents `lambda.min` and `lambda.1se` respectively. The right plot shows how λ changes the parameter values, with each line representing a variable. The x-axis in the figure is in terms of $\log(\lambda)$, hence their is a larger penalty to the right. Please note again that the package will perform scaling before the model fitting. Then, the solution on the original scale will be retrieved once all the solutions are found. However, we usually do not need to worry about these computationally issues in practice. The main advantage of

Lasso is shown here that the model can be sparse, with some parameter estimates shrunk to exactly 0. This can be seen for larger values of λ, where most parameter estimates stays at 0.

```
    par(mfrow = c(1, 2))
    plot(lasso.fit)
    plot(lasso.fit$glmnet.fit, "lambda")
```



We can obtain the estimated coefficients from the best λ value. Similar to the ridge regression example, there are two popular options, `lambda.min` and `lambda.1se`. The first one is the value that minimizes the cross-validation error, the second one is slightly more conservative, which gives larger penalty value with more shrinkage. Again, the coefficients should be extracted using the `coef()` function:

```
    coef(lasso.fit, s = "lambda.min")
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept)  0.1537694867
## lcavol       0.5071477800
## lweight      0.5455934491
## age         -0.0084065349
## lbph         0.0618168146
## svi          0.5899942923
## lcp          .
## gleason      0.0009732887
## pgg45        0.0023140828
    coef(lasso.fit, s = "lambda.1se")
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                    s1
## (Intercept) 0.6435469
## lcavol      0.4553889
## lweight     0.3142829
## age         .
## lbph        .
## svi         0.3674270
## lcp         .
## gleason     .
## pgg45       .
```

An important thing to notice here is that `lambda.min` contains more nonzero parameters. This is because a larger penalty λ will force more parameters to be zero, hence the model is more **"sparse"**. We will explain this using a one-variable example.

One-Variable Lasso and Shrinkage

To illustrate how Lasso shrink a parameter estimate to zero, let's consider the following univariate regression model (suppose no intercept):

$$\underset{\beta}{\arg\min} \quad \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i\beta)^2 + \lambda|\beta|$$

With some derivation, and also utilize the OLS solution of the loss function, we have

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i\beta)^2 && (3)\\
=&\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i\widehat{\beta}^{\text{ols}} + x_i\widehat{\beta}^{\text{ols}} - x_i\beta)^2 && (4)\\
=&\frac{1}{n}\sum_{i=1}^{n}\Big[\underbrace{(y_i - x_i\widehat{\beta}^{\text{ols}})^2}_{\text{I}} + \underbrace{2(y_i - x_i\widehat{\beta}^{\text{ols}})(x_i\widehat{\beta}^{\text{ols}} - x_i\beta)}_{\text{II}} + \underbrace{(x_i\widehat{\beta}^{\text{ols}} - x_i\beta)^2}_{\text{III}}\Big] && (5)
\end{aligned}$$

The first term is the residual sum of squares from the OLS solution and does not depend on β. The second term is more interesting:

$$\begin{aligned}
&\sum_{i=1}^{n}2(y_i - x_i\widehat{\beta}^{\text{ols}})(x_i\widehat{\beta}^{\text{ols}} - x_i\beta) && (6)\\
=&(\widehat{\beta}^{\text{ols}} - \beta)\sum_{i=1}^{n}2(y_i - x_i\widehat{\beta}^{\text{ols}})x_i && (7)\\
=&(\widehat{\beta}^{\text{ols}} - \beta)0 && (8)\\
=&0 && (9)
\end{aligned}$$

Note that the $\sum_{i=1}^{n}2(y_i - x_i\widehat{\beta}^{\text{ols}})x_i$ is exactly the gradient at $\widehat{\beta}^{\text{ols}}$ when we perform optimization of the OLS problem, this has to be zero since $\widehat{\beta}^{\text{ols}}$ is already the optimizer. Hence, our original problem reduces to just the third term and the penalty:

$$\begin{aligned}
&\underset{\beta}{\arg\min} \quad \frac{1}{n}\sum_{i=1}^{n}(x_i\widehat{\beta}^{\text{ols}} - x_i\beta)^2 + \lambda|\beta| && (10)\\
=&\underset{\beta}{\arg\min} \quad \frac{1}{n}\left[\sum_{i=1}^{n}x_i^2\right](\widehat{\beta}^{\text{ols}} - \beta)^2 + \lambda|\beta| && (11)
\end{aligned}$$

Without loss of generality, we can assume that the covariate is standardized, meaning that the sample mean is 0 and variance $\frac{1}{n}\sum_{i=1}^n x_i^2 = 1$. This leads to a general problem of

$$\underset{\beta}{\arg\min} \quad (\beta - a)^2 + \lambda|\beta|,$$

and for our case, $a = \widehat{\beta}^{\text{ols}}$. We learned that to solve for an optimizer, we can set the gradient to be zero. However, the function is not everywhere differentiable. Still, we can separate this into two cases: $\beta > 0$ and $\beta < 0$. For the positive side, we have

$$\begin{aligned} 0 =& \frac{\partial}{\partial \beta} \, (\beta - a)^2 + \lambda|\beta| = 2(\beta - a) + \lambda & (12) \\ \Longrightarrow \quad \beta =& a - \lambda/2 & (13) \end{aligned}$$

However, this will maintain positive only when β is greater than $a - \lambda/2$. The negative size is similar. And whenever β falls in between, it will be shrunk to zero. Overall, for our previous univariate optimization problem, the solution is

$$\begin{aligned} \widehat{\beta}^{\text{lasso}} &= \begin{cases} \widehat{\beta}^{\text{ols}} - \lambda/2 & \text{if} \quad \widehat{\beta}^{\text{ols}} > \lambda/2 \\ 0 & \text{if} \quad |\widehat{\beta}^{\text{ols}}| < \lambda/2 \\ \widehat{\beta}^{\text{ols}} + \lambda/2 & \text{if} \quad \widehat{\beta}^{\text{ols}} < -\lambda/2 \end{cases} & (14) \\ &= \text{sign}(\widehat{\beta}^{\text{ols}}) \left(|\widehat{\beta}^{\text{ols}}| - \lambda/2\right)_+ & (15) \\ &\doteq \text{SoftTH}(\widehat{\beta}^{\text{ols}}, \lambda) & (16) \end{aligned}$$

This is called a **soft-thresholding function**. This implies that when λ is large enough, the estimated β parameter of Lasso will be shrunk towards zero. The following animated figure demonstrates how adding an ℓ_1 penalty can change the optimizer. The objective function is $(\beta - 1)^2$. Based on our analysis, once the penalty is larger than 2, the optimizer would stay at 0.

Variable Selection Property and Shrinkage

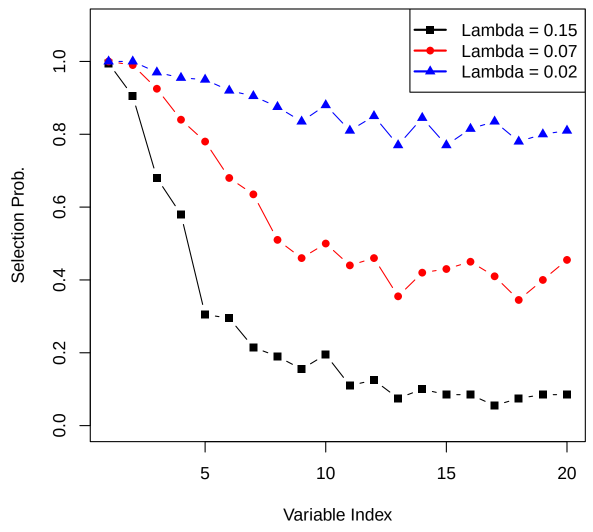
Since Lasso shrinks some parameter to exactly zero, it has the variable selection property — the ones that are nonzero are the ones being selected. This is a very nice properly in high-dimensional data analysis, where we cannot estimate the effects of all variables. Let's take a look at the simulation example we used previously. We define the true model as

$$\begin{aligned} Y =& X^{\text{T}} \boldsymbol{\beta} + \epsilon & (17) \\ =& \sum_{j=1}^p X_j \times 0.4^{\sqrt{j}} + \epsilon & (18) \end{aligned}$$

We can then look at the proportion of times a variable has a non-zero parameter estimation:

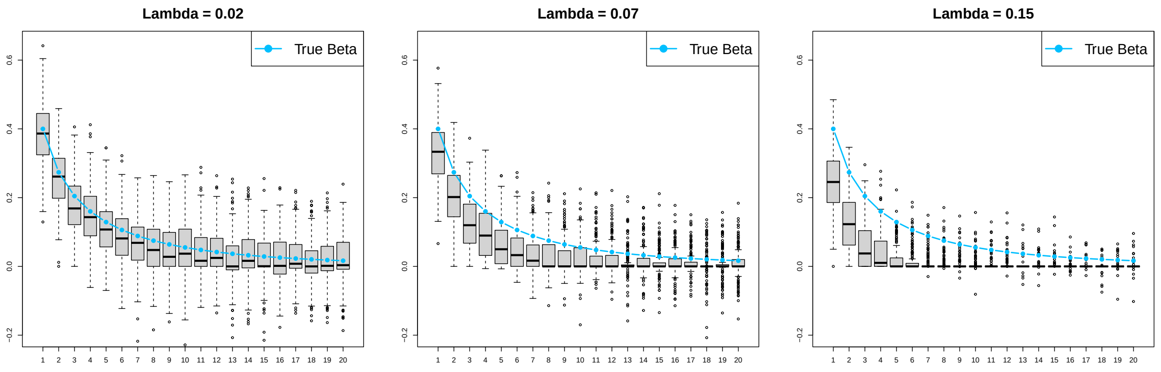
```
    nsim = 200
    n = 150
    p = 20
```

```r
betamat_1 = matrix(NA, nsim, p)
betamat_2 = matrix(NA, nsim, p)
betamat_3 = matrix(NA, nsim, p)

for (i in 1:nsim)
{
    # the design matrix
    X = matrix(rnorm(n*p), n, p)
    y = X %*% 0.4^sqrt(c(1:p)) + rnorm(n)

    # fit lasso on a grid of lambda values
    lasso.fit = glmnet(x = X, y = y, lambda = c(0.15, 0.07, 0.02))

    betamat_1[i, ] = lasso.fit$beta[, 1]
    betamat_2[i, ] = lasso.fit$beta[, 2]
    betamat_3[i, ] = lasso.fit$beta[, 3]
}

plot(colMeans(betamat_1 != 0), type = "b", pch = 15,
     xlab = "Variable Index", ylab = "Selection Prob.",
     ylim = c(0, 1.1))
lines(colMeans(betamat_2 != 0), type = "b", pch = 16, col = "red")
lines(colMeans(betamat_3 != 0), type = "b", pch = 17, col = "blue")
legend("topright", legend = paste("Lambda =", c(0.15, 0.07, 0.02)),
       col = c("black", "red", "blue"), lty = 1, lwd = 2, pch = c(15, 16, 17))
```



```r
par(mfrow = c(1, 3))

# lambda = 0.02
boxplot(betamat_3, ylim = c(-0.2, 0.65), main = "Lambda = 0.02", cex.main = 2)
lines(1:p, 0.4^sqrt(1:p), type = "b", col = "deepskyblue", pch = 19, lwd = 2)
legend("topright", "True Beta", col = "deepskyblue", pch = 19, lwd = 2, cex = 2)
```

```
# lambda = 0.07
boxplot(betamat_2, ylim = c(-0.2, 0.65), main = "Lambda = 0.07", cex.main = 2)
lines(1:p, 0.4^sqrt(1:p), type = "b", col = "deepskyblue", pch = 19, lwd = 2)
legend("topright", "True Beta", col = "deepskyblue", pch = 19, lwd = 2, cex = 2)

# lambda = 0.15
boxplot(betamat_1, ylim = c(-0.2, 0.65), main = "Lambda = 0.15", cex.main = 2)
lines(1:p, 0.4^sqrt(1:p), type = "b", col = "deepskyblue", pch = 19, lwd = 2)
legend("topright", "True Beta", col = "deepskyblue", pch = 19, lwd = 2, cex = 2)
```



Constrained Optimization View

Of course in a multivariate case, this is much more complicated since one variable may affect the optimizer of another. A commonly used alternative interpretation of the Lasso problem is the constrained optimization formulation:

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 \tag{19}$$

$$\text{subject to } \|\boldsymbol{\beta}\|_1 \leq s \tag{20}$$

We can see from the left penal of the following figure that, the Lasso penalty imposes a constraint with the rhombus, i.e., the solution has to stay within the shaded area. The objective function is shown with the contour, and once the contained area is sufficiently small, some β parameter will be shrunk to exactly zero. On the other hand, the Ridge regression also has a similar interpretation. However, since the constrained areas is a circle, it will never for the estimated parameters to be zero.

add image here

Bias-variance Trade-off

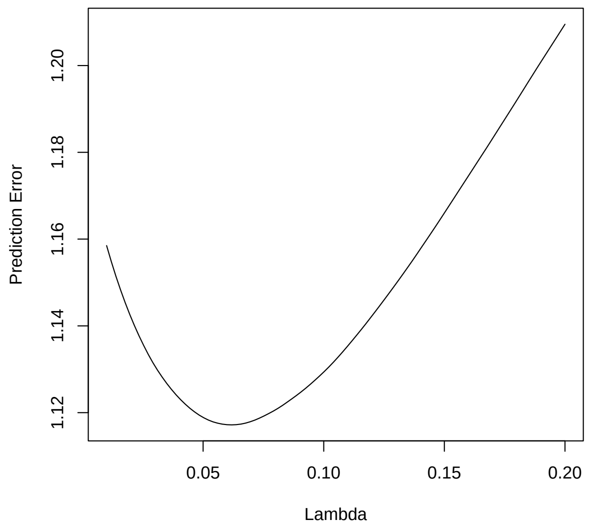
Lasso also enjoys similar bias-variance trade-off as the Ridge regression. The bias part comes from the fact that the Lasso estimator is different from the OLS estimator, which is unbiased. The variance can be reduced from OLS because some variables are removed from the model, making it more stable. Also, nonzero variables will be shrunk towards zero, making the parameter estimates more stable. We have the same conclusion as Ridge:

- As $\lambda \downarrow$ decrease, bias $\downarrow$ decrease and variance $\uparrow$ increases

- As $\lambda \uparrow$ increases, bias $\uparrow$ increases and variance $\downarrow$ decrease

We can use another simulation study to confirm this with the same experiment we used previously.

```r
nsim = 200
n = 150
p = 20

alllambda = seq(0.2, 0.01, length.out = 100)
allerror = matrix(NA, nsim, length(alllambda))

for (i in 1:nsim)
{
    # the training data
    Xtrain = matrix(rnorm(n*p), n, p)
    ytrain = Xtrain %*% 0.4^sqrt(c(1:p)) + rnorm(n)

    # the testing data
    Xtest = matrix(rnorm(n*p), n, p)
    ytest = Xtest %*% 0.4^sqrt(c(1:p)) + rnorm(n)

    # fit lasso on a grid of lambda values
    lasso.fit = glmnet(x = Xtrain, y = ytrain, lambda = alllambda)

    # record prediction
    allpred = predict(lasso.fit, Xtest)
    allerror[i, ] = colMeans((sweep(allpred, MARGIN = 1, ytest, FUN = "-"))^2)
}

plot(rev(alllambda), rev(colMeans(allerror)), type = "l",
     xlab = "Lambda", ylab = "Prediction Error")
```



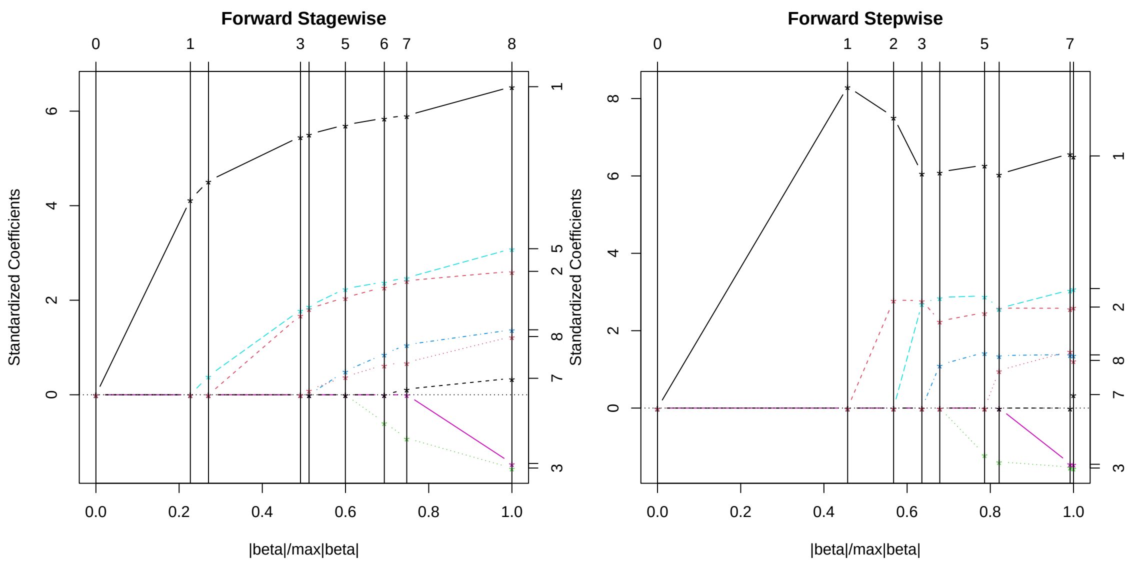
Forward-stagewise Regression and the Solution Path

We are interested in getting the fitted model with a given λ value, however, for selecting the tuning parameter, it would be much more stable to obtain the solution on a sequence of λ values. The corresponding $\boldsymbol{\beta}$ parameter estimates are called the solution path, i.e., the path how parameter changes as λ changes. For Lasso, the the solution path has an interpretation as the **forward-stagewise** regression. This is different than the forward stepwise model we introduced before. A forward stagewise regression works in the following way:

- Start with the Null model (intercept) and choose the best variable out of all p, such that when its parameter grows by a small magnitude ϵ (either positive or negative), the RSS reduces the most. Grow the parameter estimate of this variable by ϵ and repeat.

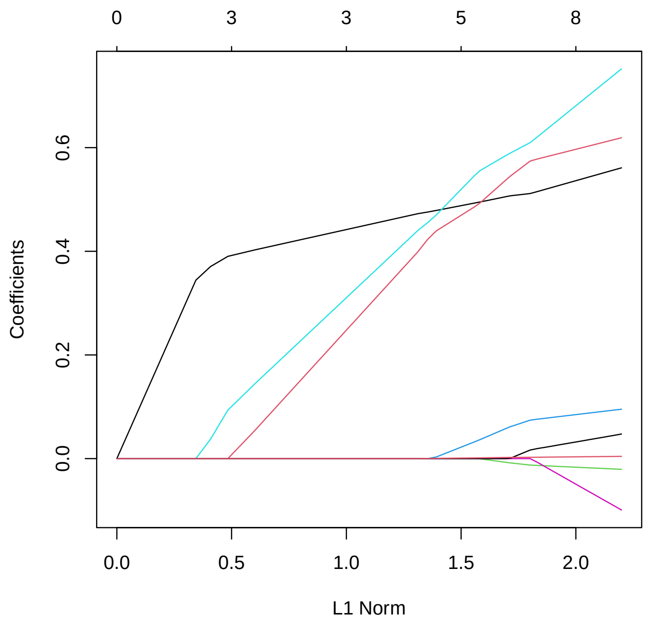
The stage-wise regression solution has been shown to give the same solution path as the Lasso, if we start with a sufficiently large λ, and gradually reduces it towards zero. This can be done with the least angle regression (`lars`) package. Note that the `lars` package introduces another computationally more efficient approach to obtain the same solution path, but we will not discuss it in details. We comparing the two approaches (stagewise and stepwise) using the `prostate` data. Note that stepwise regression solution is the same as the plot produced by `glmnet`, except that this is the regular scale, and the `glmnet` plot is on the log scale of λ.

```r
par(mar=c(4,4,4,2))
par(mfrow=c(1,2))

library(lars)

lars.fit = lars(x = data.matrix(prostate[, 1:8]), y = prostate$lpsa,
                type = "forward.stagewise")
plot(lars.fit)

lars.fit = lars(x = data.matrix(prostate[, 1:8]), y = prostate$lpsa,
                type = "stepwise")
plot(lars.fit)
```



Using `glmnet` we can also obtain the solution path. Keep in mind that for Lasso, `alpha = 1`, and for Ridge `alpha = 0`.

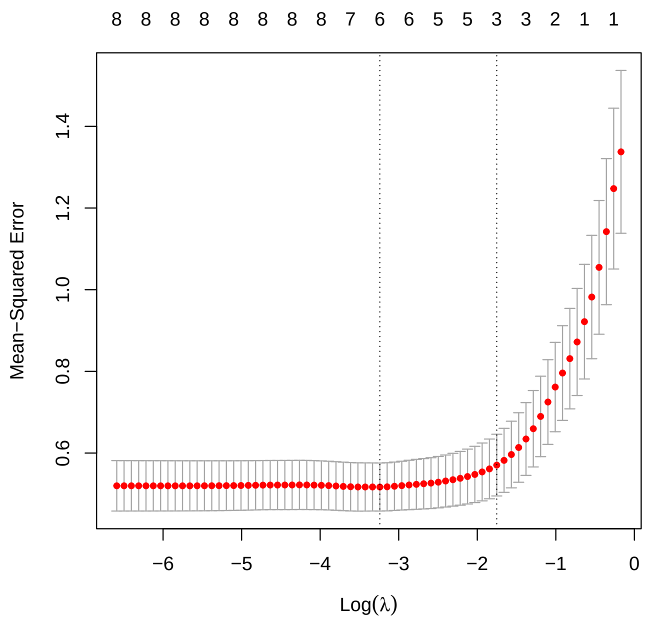
```r
# fit lasso
lasso.fit = cv.glmnet(data.matrix(prostate[, 1:8]),
                      prostate$lpsa, nfolds = 10, alpha = 1)

# to obtain the solution path
plot(lasso.fit$glmnet.fit)
```



Why are these two solution path different? This is because the `lars` package plot the standardized coefficients, meaning that, the parameters are presented in the scale as if their standard deviations are all 1. This is useful when variables are of dramatically different scales and it is meaningful to compare their effects.

After fitting the Lasso solution, we can utilize the cross-validation feature to select the best model. The smallest cross-validation error is 0.5167764.

```r
# plot cross-validation error
plot(lasso.fit)
```

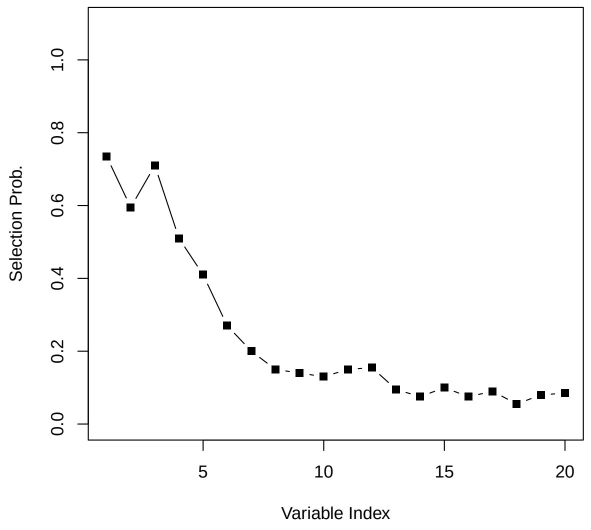


```
    # the best cross-validation error
    min(lasso.fit$cvm)
## [1] 0.5167764
```

Elastic-Net

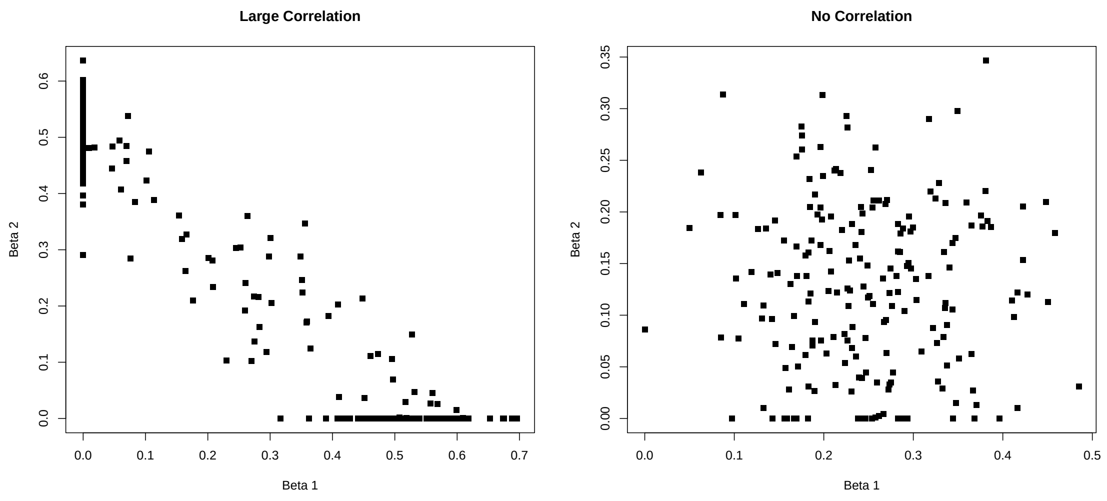
Lasso may suffer in the case where two variables are strongly correlated. The situation is similar to OLS, however, in Lasso, it would only select one out of the two, instead of letting both parameter estimates to be large. This is not preferred in some practical situations such as genetic studies because expressions of genes from the same pathway may have large correlation, but biologist want to identify all of them instead of just one. The Ridge penalty may help in this case because it naturally considers the correlation structure. The following simulation may show the effect.

```
    library(MASS)
##
## Attaching package: 'MASS'
## The following object is masked from 'package:plotly':
##
##     select

    nsim = 200
    n = 150
    p = 20

    lassobeta = matrix(NA, nsim, p)

    for (i in 1:nsim)
    {
        # the design matrix
        X = cbind(mvrnorm(n, mu = c(0, 0), Sigma = matrix(c(1, 0.99, 0.99, 1), 2, 2)),
                 matrix(rnorm(n*(p - 2)), n, p-2))

        y = X %*% 0.4^sqrt(c(1:p)) + rnorm(n)

        # fit lasso on a grid of lambda values
        lasso.sim = glmnet(x = X, y = y, lambda = 0.15)

        lassobeta[i, ] = lasso.sim$beta[, 1]
    }

    plot(colMeans(lassobeta != 0), type = "b", pch = 15,
         xlab = "Variable Index", ylab = "Selection Prob.",
         ylim = c(0, 1.1))
```



From the plot above, we can see the select probability for both β_1 and β_2 are reduced. This is because when one of them is selected, the other one is often shrunk to 0.

```
    par(mfrow = c(1, 2))

    # parameter estimations
    plot(lassobeta[, 1:2], pch = 15,
         xlab = "Beta 1", ylab = "Beta 2", main = "Large Correlation")

    # this is not the case for independent variables
    plot(betamat_1[, 1:2], pch = 15,
         xlab = "Beta 1", ylab = "Beta 2", main = "No Correlation")
```

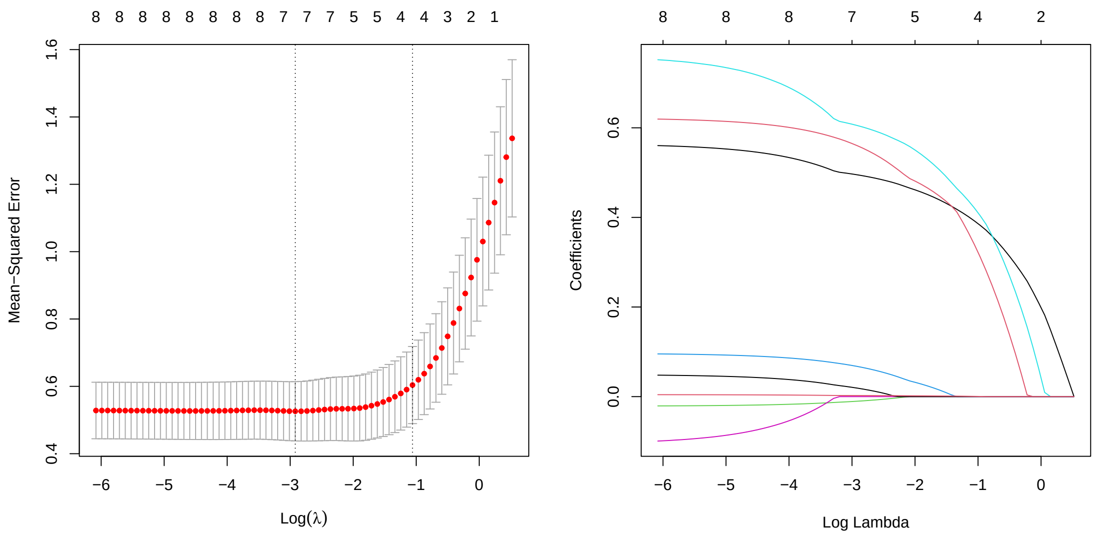


Hence the **Elastic-Net** (Zou and Hastie, 2005) penalty was proposed to address this issue: the data contains many correlated variables and we want to select them together if they are important for prediction. The `glmnet` package uses the following definition of an Elastic-Net penalty, which is a mixture of ℓ_1 and ℓ_2 penalties:

$$\lambda \left[(1-\alpha)/2 \|\boldsymbol{\beta}\|_2^2 + \alpha |\boldsymbol{\beta}|_1 \right],$$

which involves two tuning parameters. You can tune it based on cross-validation. The procedure would be to create a grid of combination of (λ, α) and calculate the cross-validation error for each combination and select the best one. Some times in practice we can simply use $\alpha = 0, 0.5$ and 1.

```
par(mar=c(4,4,4,2))
par(mfrow=c(1,2))
enet.fit = cv.glmnet(data.matrix(prostate[, 1:8]), prostate$lpsa, nfolds = 10, alpha = 0.5)
plot(enet.fit)
plot(enet.fit$glmnet.fit, "lambda")
```



```
  coef(enet.fit, s = "lambda.min")
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.10738525
## lcavol       0.49512547
## lweight      0.56022283
## age         -0.01026908
## lbph         0.06746212
## svi          0.60519272
## lcp          .
## gleason      0.01903197
## pgg45        0.00242396
```

We can obtain the cross-validation error corresponding to the best λ. It is slightly worse than the Lasso solution, however, it may be because of the random seed. Also, it is a sparse solution in this case.

```
  min(enet.fit$cvm)
## [1] 0.5259804
  min(lasso.fit$cvm)
## [1] 0.5167764
```

Fused Lasso and ADMM

The Fused Lasso is a particular type of regression that not only performs variable selection but also takes into account the structure among the variables. The objective function is:

$$\min_{\beta} \left\{ \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda_1\|\boldsymbol{\beta}\|_1 + \lambda_2 \sum_{i=1}^{n-1} |\beta_{i+1} - \beta_i| \right\}$$

Here, λ_1 controls the L1 penalty for sparsity, and λ_2 controls the fused penalty for smoothness between adjacent coefficients. Solving the Fused Lasso using standard coordinate descent is problematic. The challenge comes primarily from the fused penalty term $\lambda_2 \sum_{i=1}^{n-1} |\beta_{i+1} - \beta_i|$. This term couples the adjacent coefficients β_i and β_{i+1}, making it hard to minimize the objective function for one β_i while keeping all other β fixed, which is the core idea behind coordinate descent. Hence, one solution is to move two variables together. We will introduce the Augmented Direction Method of Multiplier (ADMM) that can solve this efficiently.

Augmented Lagrangian

Before discussing ADMM, let's understand the Augmented Lagrangian. For an optimization problem:

$$\min_{x} \quad f(x) \quad \text{s.t.} \quad Ax = b$$

The Augmented Lagrangian is:

$$\mathcal{L}_\rho(x, \lambda) = f(x) + \lambda^T(Ax - b) + \frac{\rho}{2}\|Ax - b\|^2$$

The ADMM Algorithm

ADMM combines the decomposability of dual methods (when the constrain is additive, say $Ax + Bz = c$ with the convergence properties of the Augmented Lagrangian, solving problems of the form:

$$\min_{x,z} \quad f(x) + g(z) \quad \text{s.t.} \quad Ax + Bz = c$$

The algorithm iteratively updates x, z, and u using:

1. x**-update**: $x^{k+1} = \arg\min_x \mathcal{L}(x, z^k, u^k)$
2. z**-update**: $z^{k+1} = \arg\min_z \mathcal{L}(x^{k+1}, z, u^k)$
3. u**-update**: $u^{k+1} = u^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$

Fused Lasso via ADMM

To solve the Fused Lasso problem using ADMM, one can introduce auxiliary variables and constraints to separate the ℓ_1 and fused penalties, thereby transforming it into a problem suitable for ADMM. We introduce new variables z_1 and z_2:

$$z_1 = \boldsymbol{\beta}$$
$$z_2 = D\boldsymbol{\beta}$$

where D is an $(n-1) \times n$ difference matrix

$$D = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

such that:

$$D\boldsymbol{\beta} = \begin{bmatrix} \beta_2 - \beta_1 \\ \beta_3 - \beta_2 \\ \vdots \\ \beta_n - \beta_{n-1} \end{bmatrix}$$

The problem becomes:

$$\min_{\boldsymbol{\beta}, z_1, z_2} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda_1\|z_1\|_1 + \lambda_2\|z_2\|_1 \text{s.t. } z_1 - \boldsymbol{\beta} = 0 z_2 - D\boldsymbol{\beta} = 0$$

The Augmented Lagrangian is:

$$\mathcal{L}(\boldsymbol{\beta}, z_1, z_2, u_1, u_2) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda_1\|z_1\|_1 + \lambda_2\|z_2\|_1 + u_1^T(z_1 - \boldsymbol{\beta}) + \frac{\rho_1}{2}\|z_1 - \boldsymbol{\beta}\|_2^2 + u_2^T(z_2 - D\boldsymbol{\beta}) + \frac{\rho_2}{2}\|z_2 - D\boldsymbol{\beta}\|_2^2$$

The ADMM algorithm iteratively updates $\boldsymbol{\beta}, z_1, z_2, u_1$, and u_2 using the Augmented Lagrangian. Note that the update for $\boldsymbol{\beta}$ will be trivial, the updates for z_1 and z_2 are soft thresholding functions. For a comprehensive understanding of the KKT conditions and convex optimization, refer to Boyd and Vandenberghe's work on Convex Optimization [@boyd2004convex].

Practice question

Use the `prostate` data, compare the solution for $\alpha = 0.25$ and 0.5. Which one gives a better model fit, using cross-validation errors?

```r
enet.fit25 = cv.glmnet(data.matrix(prostate[, 1:8]),
                       prostate$lpsa, nfolds = 10, alpha = 0.25)
min(enet.fit25$cvm)
```