REGULARIZATION METHODS

Table of contents

1. Ridge regression

- In linear regression of $y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \epsilon_i$, the least squares solution to the linear regression is written as

$$\widehat{\beta}_{LS} = \left(X^T X\right)^{-1} X^T Y \tag{1}$$

that is an unbiased estimator of β. The covariance of $\widehat{\beta}_{LS}$ is

$$\mathrm{Cov}\left(\widehat{\beta}_{LS}\right) = \sigma^2 \left(X^T X\right)^{-1} \tag{2}$$

- The predictors may be (nearly) linearly dependent and hence $X^T X$ is (nearly) singular. The mean square error may become large. This problem is called multicollinearity.
- Hoerl and Kennard (1970) proposed the ridge estimator

$$\widehat{\beta}_{\lambda} = \left(X^T X + \lambda I\right)^{-1} X^T Y \tag{3}$$

- The ridge estimator minimizes the penalized sum of squared errors

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{4}$$

- It is equivalent to minimize $\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2$ under constraint $||\beta||_2^2 = \sum_{j=1}^{p} \beta_j^2 \le c$.
- $\lambda \sum_{j=1}^{p} \beta_j^2$ is called the shrinkage penalty term
- λ is called the shrinkage parameter and it is a hyperparameter
- When $\lambda = 0$, the ridge estimator is equal to the least squares estimator $\widehat{\beta}_{Ridge} = \widehat{\beta}_{LSE}$
- $If \lambda \to \infty$, then $\widehat{\beta}_{Ridge} \to 0$

- The predictor should be standardized and the response shoud be centered before applying Ridge regression.
- Suppose Y is centered and X is standardized. The ridge regression minimizes

$$(Y - X\beta)^T(Y - X\beta) + \lambda\beta^T\beta = \beta^T\left(X^TX + \lambda I\right)\beta - 2\beta^TX^TY + Y^TY \tag{5}$$

$$\widehat{\beta}_\lambda = \left(X^TX + \lambda I\right)^{-1}X^TY \tag{6}$$

$$\mathrm{Cov}(\widehat{\beta}_\lambda) = \sigma^2\left(X^TX + \lambda I\right)^{-1}X^TX\left(X^TX + \lambda I\right)^{-1} \tag{7}$$

- The ridge estimator is biased but it can have smaller variance than the least squares estimator (LSE). The MSE of the ridge estimator can be smaller than the MSE of the LSE for some λ.
- Although the regression coefficients of the ridge estimator shrink toward zero as $\lambda \to \infty$, none of the regression coefficients vanishes for a finite λ.

1.1. **Example.** Ridge regression for Boston data ::: {.cell}

```
# Ridge Regression
library(MASS)
x = model.matrix(medv ~ ., Boston)[, -1]
y = Boston$medv
library(glmnet)
grid = 10ˆseq(10, -2, length = 100)
ridge.mod = glmnet(x, y, alpha = 0, lambda = grid)
dim(coef(ridge.mod))
```

```
[1]  14 100
```

```r
ridge.mod$lambda[50]  # it is equal to grid[50]
```

```
[1] 11497.57
```

```r
coef(ridge.mod)[, 50]
```

```
  (Intercept)          crim            zn         indus          chas 
 2.255401e+01 -3.305151e-04  1.131176e-04 -5.160096e-04  5.066065e-03 
          nox            rm           age           dis           rad 
-2.697423e-02  7.256984e-03 -9.791268e-05  8.654283e-04 -3.203678e-04 
          tax       ptratio         black         lstat 
-2.033098e-05 -1.718182e-03  2.672815e-05 -7.566302e-04 
```

```r
sqrt(sum(coef(ridge.mod)[-1, 50]^2))
```

```
[1] 0.02847302
```

```r
ridge.mod$lambda[60]
```

```
[1] 705.4802
```

```r
coef(ridge.mod)[, 60]
```

```
  (Intercept)          crim            zn         indus          chas 
22.8400256987 -0.0050123699  0.0017066646 -0.0077798380  0.0810963388 
          nox            rm           age           dis           rad 
-0.4042720427  0.1138263867 -0.0014627326  0.0123500786 -0.0047694408 
          tax       ptratio         black         lstat 
-0.0003063892 -0.0266277398  0.0004075631 -0.0117325860 
```

```r
sqrt(sum(coef(ridge.mod)[-1, 60]^2))
```

```
[1] 0.4290476
```

```r
# ridge estimates when s=lambda=50
predict(ridge.mod, s = 50, type = "coefficients")[1:14, ]
```

```
 (Intercept)         crim           zn        indus         chas          nox 
23.580881779 -0.036358794  0.011707752 -0.052663644  0.914385347 -2.584648038 
          rm          age          dis          rad          tax      ptratio 
 1.136076181 -0.008821137  0.021708896 -0.026332105 -0.002040002 -0.238276456 
       black        lstat 
 0.003150038 -0.106839435 
```

```r
# Validation set approach to tune the hyperparameter
set.seed(100)
nrow(x)
```

```
[1] 506
```

```r
train = sample(1:nrow(x), 0.7 * nrow(x))
test = (-train)
y.test = y[test]
ridge.mod = glmnet(x[train, ], y[train], alpha = 0, lambda = grid)
ridge.pred = predict(ridge.mod, s = 4, newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
[1] 35.00625
```

```r
mean((mean(y[train]) - y.test)^2)  # when lambda=infinity, i.e., y^hat=const
```

```
[1] 97.92869
```

```r
ridge.pred = predict(ridge.mod, s = 1e+10, newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
[1] 97.92869
```

```
ridge.pred = predict(ridge.mod, s = 0, newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
[1] 34.11268
```

```
lm(y ~ x, subset = train)
```

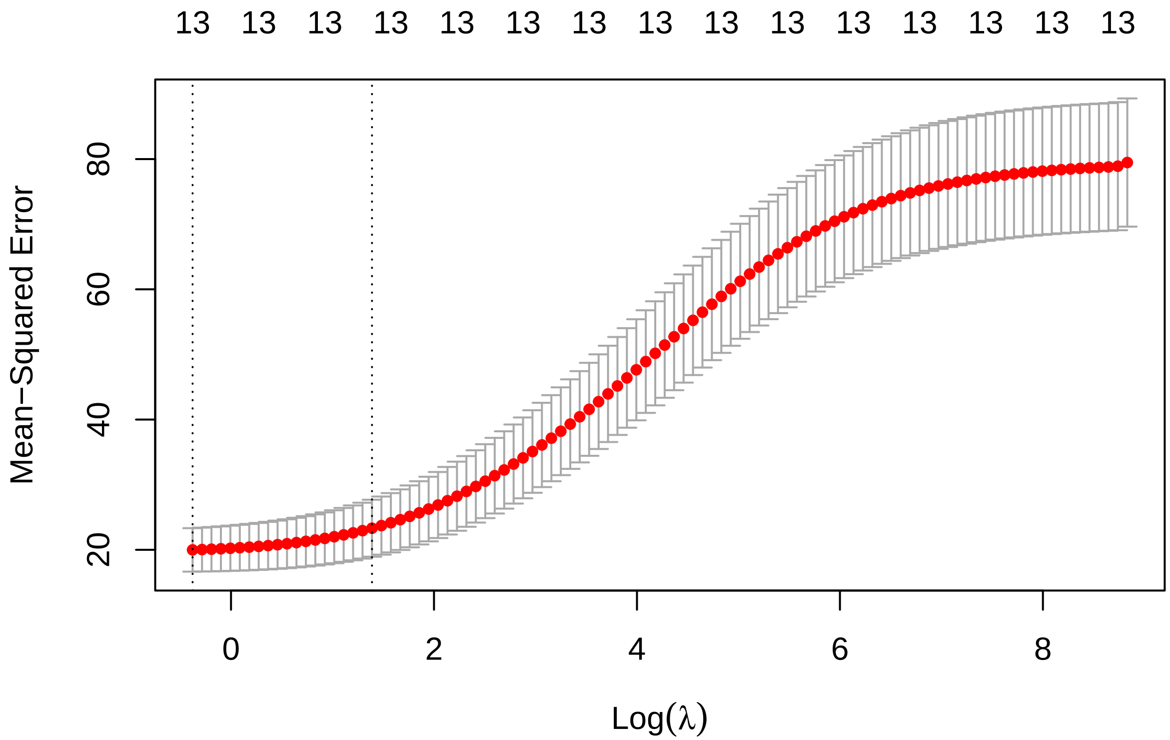
```

Call:
lm(formula = y ~ x, subset = train)

Coefficients:
(Intercept)        xcrim          xzn      xindus       xchas         xnox
  35.822004    -0.121715     0.049343    0.034271    2.812441   -11.527333
        xrm         xage         xdis        xrad        xtax     xptratio
   3.567397    -0.011754    -1.470180    0.269251   -0.010241    -0.926630
     xblack       xlstat
   0.008313    -0.640238
```

```
predict(ridge.mod, s = 0, type = "coefficients")[1:14, ]
```

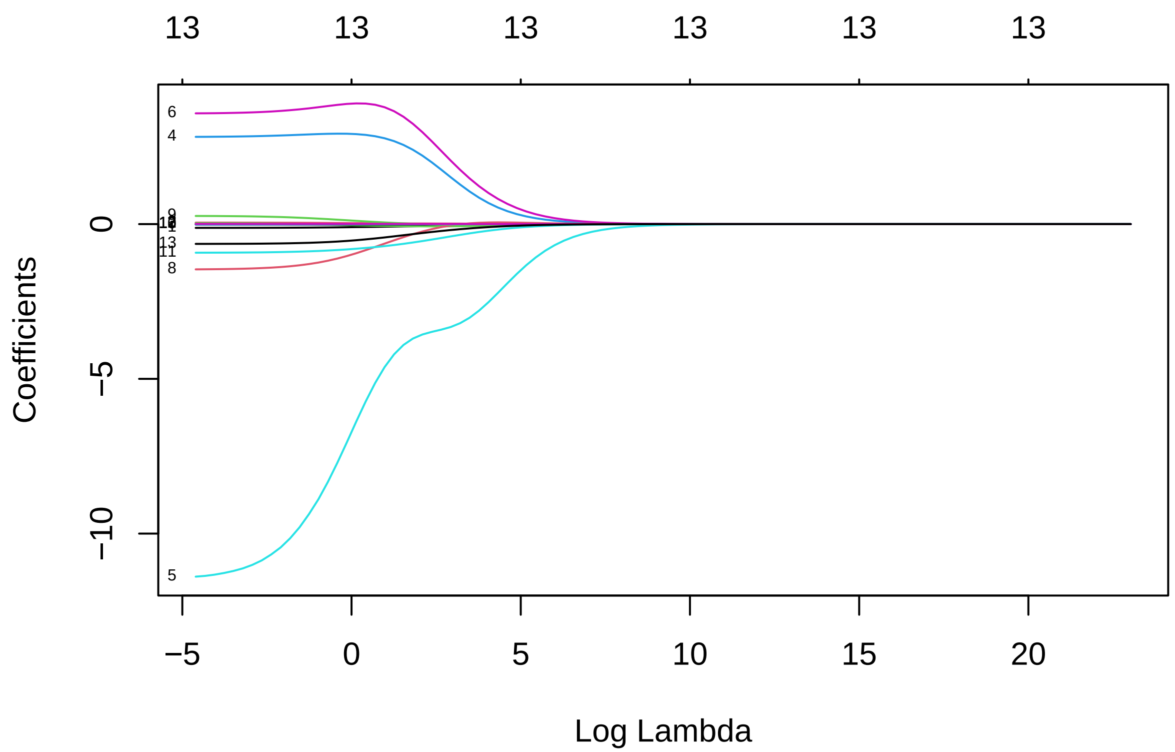
```
  (Intercept)          crim            zn         indus          chas
```

```
 35.549908703  -0.120939141   0.048862519   0.031905436   2.819839113 
          nox            rm           age           dis           rad 
-11.390432949   3.579184613  -0.011812336  -1.461821727   0.263630544 
          tax       ptratio         black         lstat 
 -0.009983265  -0.923745907   0.008314594  -0.638866945 
```

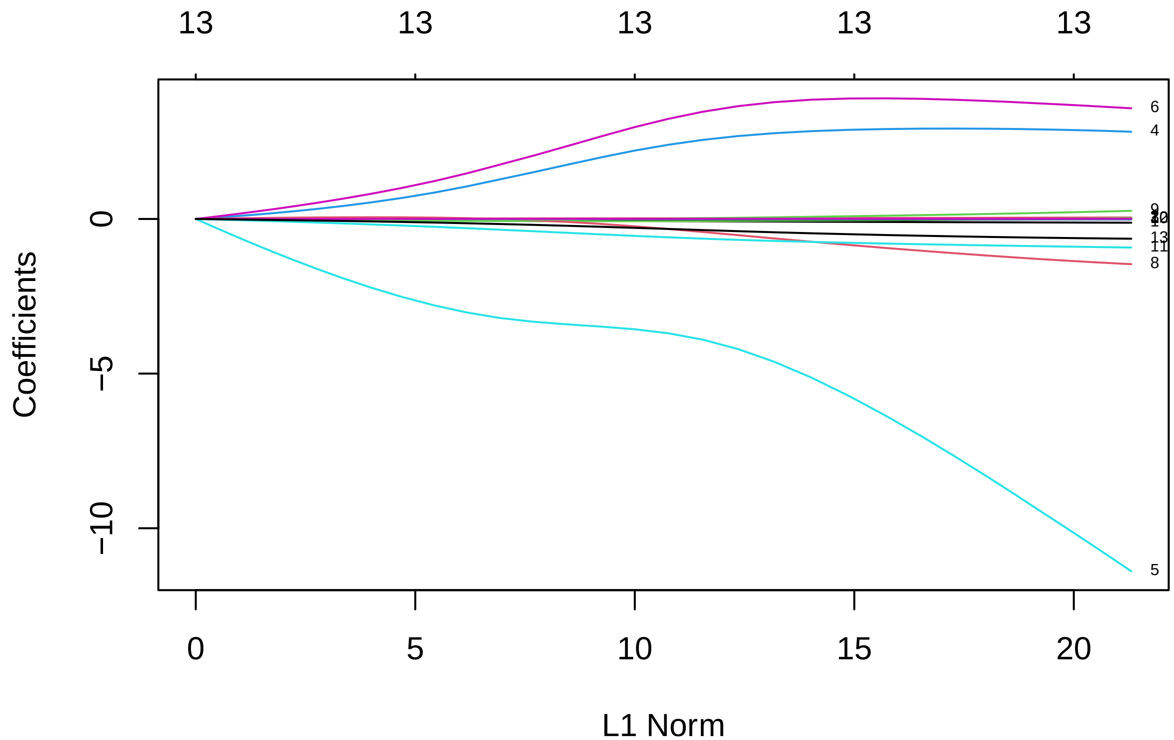
```r
set.seed(1)
cv.out = cv.glmnet(x[train, ], y[train], alpha = 0)
plot(cv.out)
```




```
plot(ridge.mod, xvar = "lambda", label = TRUE)
```




```
plot(ridge.mod, xvar = "norm", label = TRUE)
```




```
bestlam = cv.out$lambda.min
bestlam
```

```
[1] 0.6847872
```

```
ridge.pred = predict(ridge.mod, s = bestlam, newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
[1] 33.55867
```

```r
out = glmnet(x, y, alpha = 0)
predict(out, type = "coefficients", s = bestlam)[1:14, ]
```

```
  (Intercept)          crim            zn         indus          chas 
 27.951414256  -0.087459046   0.032602802  -0.038248629   2.900457357 
          nox            rm           age           dis           rad 
-11.875843374   4.011767991  -0.003755150  -1.116193399   0.152979546 
          tax       ptratio         black         lstat 
 -0.005724684  -0.854281077   0.009070677  -0.471990603 
```

:::

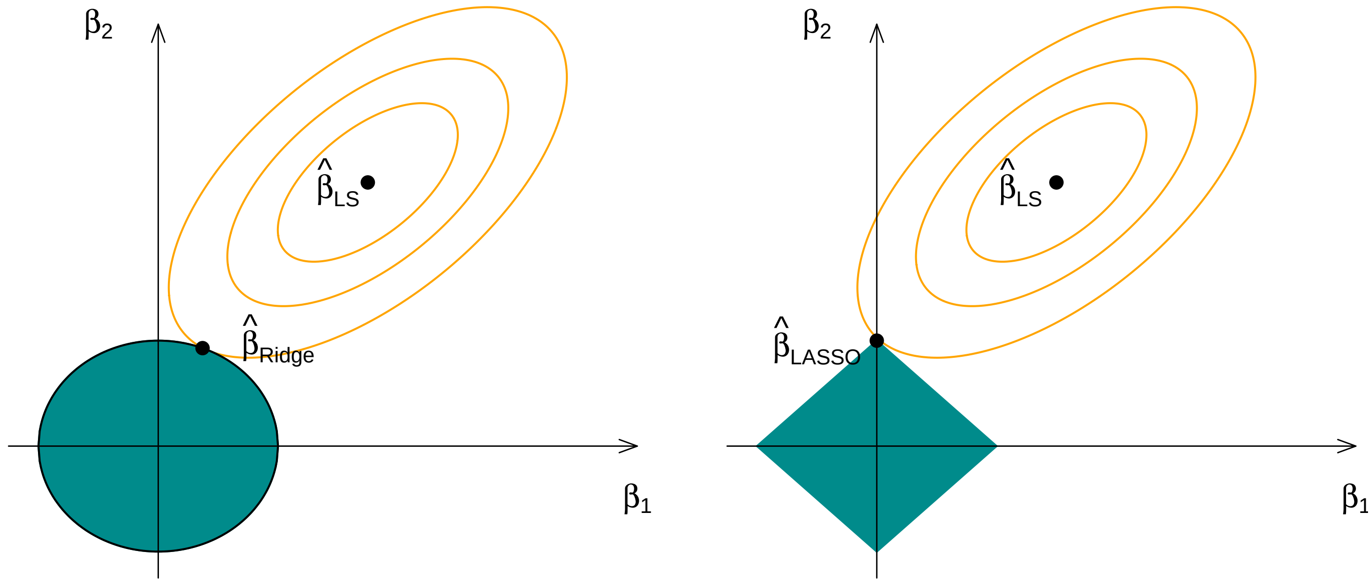
2. Least Absolute Shrinkage and Selection Operator (LASSO)

- Tibshirani (1996) considered L_1 regularization instead of L_2 regularization in the ridge regression. The LASSO solutions minimize

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| \tag{8}$$

- Unlike the ridege regularization, the LASSO solutions do not have analytic solutions and require numerical optimization.
- It forces some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large. $\rightarrow$ The lasso performs variable selection.
- When $\lambda = 0$, the LASSO estimator is equal to the LSE: $\widehat{\beta}_{LASSO} = \widehat{\beta}_{LS}$
- If $\lambda \rightarrow \infty$, then $\widehat{\beta}_{LASSO} \rightarrow 0$
- The predictor should be standardized before applying LASSO.
- Selecting the Tuning Parameter
  - Choose a grid of λ values, and compute the cross-validation error for each value of λ.
  - Select the tuning parameter value for which the cross-validation error is smallest.

Figure 1. Geometry of Ridge and LASSO




```r
# LASSO application to Boston data Validation set approach to tune
# the hyperparameter
library(MASS)
library(glmnet)
x = model.matrix(medv ~ ., Boston)[, -1]
y = Boston$medv
nrow(x)
```

2.1. **Example.**

```
[1] 506
```

```r
set.seed(100)
train = sample(1:nrow(x), 0.7 * nrow(x))
test = (-train)
y.test = y[test]
LASSO.mod = glmnet(x[train, ], y[train], alpha = 1, lambda = grid)
LASSO.pred = predict(LASSO.mod, s = 4, newx = x[test, ])
mean((LASSO.pred - y.test)^2)
```

```
[1] 61.46683
```

```r
mean((mean(y[train]) - y.test)^2)  # when lambda=infinity, i.e., y^hat=const
```

```
[1] 97.92869
```

```r
LASSO.pred = predict(LASSO.mod, s = 1e+10, newx = x[test, ])
mean((LASSO.pred - y.test)^2)
```

```
[1] 97.92869
```

```r
LASSO.pred = predict(LASSO.mod, s = 0, newx = x[test, ])
mean((LASSO.pred - y.test)^2)  #LSE
```

```
[1] 34.1495
```

```r
lm(medv ~ ., data = Boston, subset = train)
```

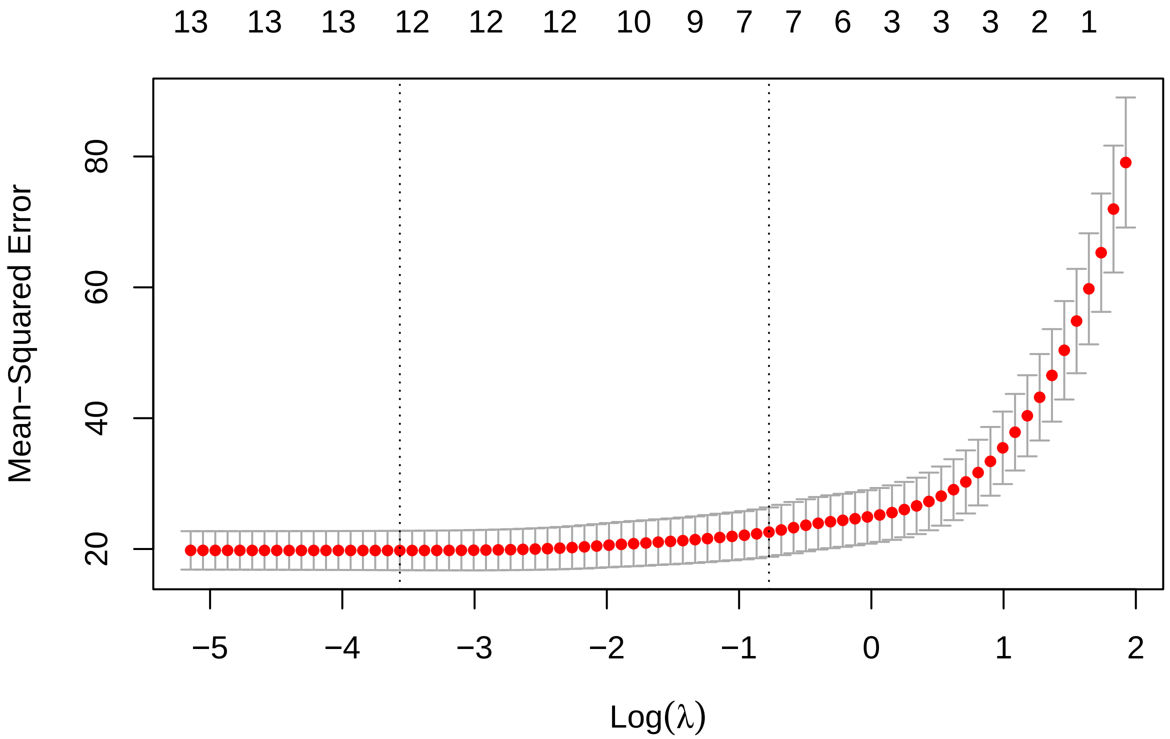
```

Call:
lm(formula = medv ~ ., data = Boston, subset = train)

Coefficients:
(Intercept)         crim           zn        indus         chas          nox
  35.822004    -0.121715     0.049343     0.034271     2.812441   -11.527333
         rm          age          dis          rad          tax      ptratio
   3.567397    -0.011754    -1.470180     0.269251    -0.010241    -0.926630
      black        lstat
   0.008313    -0.640238
```

```r
predict(LASSO.mod, s = 0, type = "coefficients")[1:14, ]
```

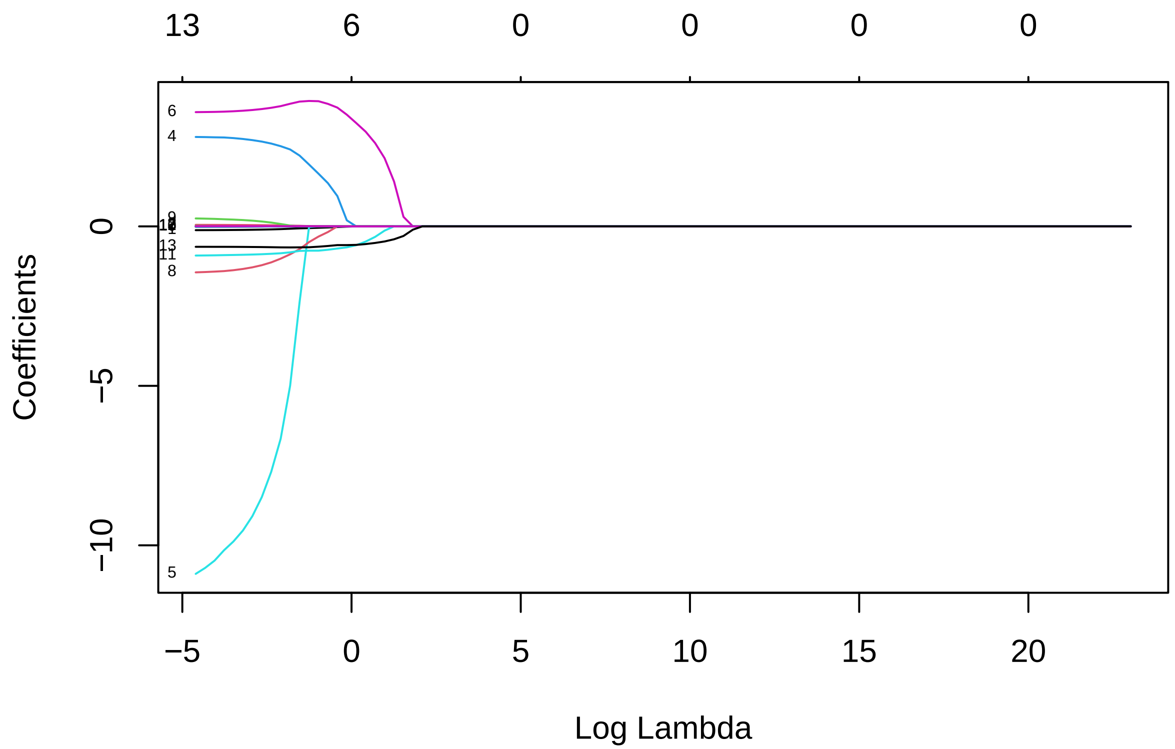
```
  (Intercept)          crim            zn         indus          chas
 34.997839277  -0.118830754   0.047426734   0.019179733   2.803618591
```

```
        nox            rm           age           dis           rad
-10.894773927   3.581606794  -0.011038824  -1.441070634   0.248876603
          tax       ptratio         black         lstat
 -0.009197514  -0.914732348   0.008191588  -0.640537015
```

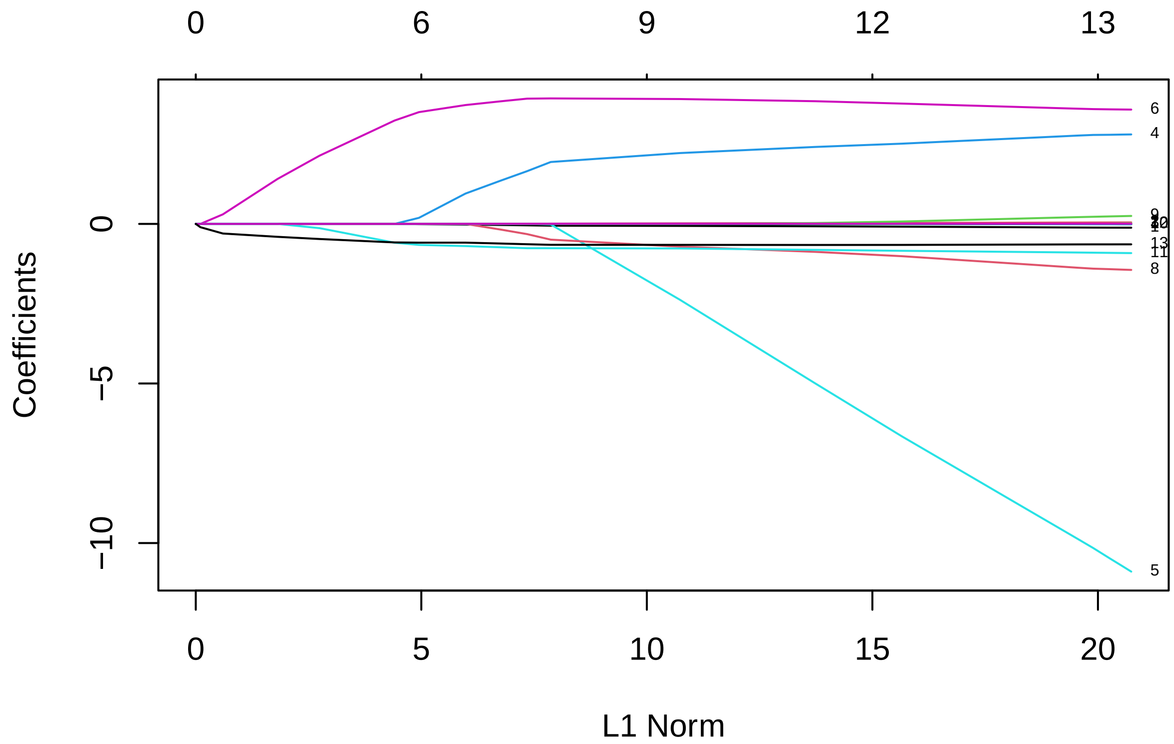
```r
set.seed(1)
cv.out1 = cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out1)
```



```r
plot(LASSO.mod, xvar = "lambda", label = TRUE)
```




```r
plot(LASSO.mod, xvar = "norm", label = TRUE)
```




```
bestlam1 = cv.out1$lambda.min
bestlam1
```

```
[1] 0.02829549
```

```
LASSO.pred = predict(LASSO.mod, s = bestlam1, newx = x[test, ])
mean((LASSO.pred - y.test)^2)
```

```
[1] 34.23123
```

```r
out = glmnet(x, y, alpha = 1)
predict(out, type = "coefficients", s = bestlam1)[1:14, ]
```

```
  (Intercept)          crim            zn         indus          chas
 34.407508409  -0.098248428   0.041398312   0.000000000   2.684795545
          nox            rm           age           dis           rad
-16.295889053   3.867565182   0.000000000  -1.395053224   0.252230309
          tax       ptratio         black         lstat
 -0.009807801  -0.929823057   0.009023176  -0.522498953
```

3. Elastic nets

- Zou and Hastie (2005) combined the ridge regression (L_2) and LASSO (L_1) as a convex combination, that is, $0 \leq \alpha \leq 1$,

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\left((1-\alpha)\sum_{j=1}^{p}\beta_j^2 + \alpha\sum_{j=1}^{p}|\beta_j|\right) \tag{9}$$

- In elastic net, there are two tuning parameters (α and λ).
- When $\alpha = 0$, it is the same as the ridge solution.
- When $\alpha = 1$, it is the same as the LASSO.

```r
# Elastic nets
enet = glmnet(x[train, ], y[train], alpha = 0.5, lambda = grid)
# plot(enet)
set.seed(1)
cv.outenet = cv.glmnet(x[train, ], y[train], alpha = 0.5)
# plot(cv.out)
bestlamenet = cv.outenet$lambda.min
enet.pred = predict(enet, s = bestlamenet, newx = x[test, ])
mean((enet.pred - y.test)^2)
```

3.1. **Example Elastic net for Boston data.**

```
[1] 34.19188
```

```r
out.enet = glmnet(x, y, alpha = 0.5, lambda = grid)
enet.coef = predict(out.enet, type = "coefficients", s = bestlam)[1:14,
    ]
enet.coef
```

```
  (Intercept)           crim             zn          indus           chas
 1.802892e+01 -2.922299e-02  2.765551e-04  0.000000e+00  2.019094e+00
          nox             rm            age            dis            rad
-4.476616e+00  4.241007e+00  0.000000e+00 -3.420123e-01  0.000000e+00
          tax        ptratio          black          lstat
-1.718986e-05 -7.891797e-01  6.747469e-03 -4.931839e-01
```

```r
enet.coef[enet.coef != 0]
```

```
  (Intercept)           crim             zn           chas            nox
 1.802892e+01 -2.922299e-02  2.765551e-04  2.019094e+00 -4.476616e+00
           rm            dis            tax        ptratio          black
 4.241007e+00 -3.420123e-01 -1.718986e-05 -7.891797e-01  6.747469e-03
        lstat
-4.931839e-01
```

```r
set.seed(100)
train = sample(nrow(x), 0.7 * nrow(x))
test = (-train)
x.train = x[train, ]
x.test = x[test, ]
y.train = y[train]
y.test = y[test]



library(glmnet)
library(doParallel)
nc = detectCores()
registerDoParallel(nc)
library(caret)
myControl = trainControl(method = "cv", number = 10, allowParallel = T,
    savePredictions = "final")

set.seed(10)
fit = train(x.train, y.train, method = "glmnet", trControl = myControl,
    tuneLength = 3, preProcess = c("center", "scale"))
fit
```

```
glmnet
```

```
354 samples
 13 predictor

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 319, 318, 318, 319, 318, 319, ...
Resampling results across tuning parameters:

  alpha  lambda      RMSE      Rsquared   MAE
  0.10   0.01369574  4.313660  0.7704306  3.160291
  0.10   0.13695744  4.301387  0.7712624  3.123830
  0.10   1.36957437  4.443096  0.7589495  3.130846
  0.55   0.01369574  4.314264  0.7703221  3.160943
  0.55   0.13695744  4.321564  0.7685076  3.120456
  0.55   1.36957437  4.733634  0.7335740  3.389508
  1.00   0.01369574  4.314116  0.7702373  3.158374
  1.00   0.13695744  4.366494  0.7628637  3.139326
  1.00   1.36957437  4.954516  0.7207622  3.543321

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were alpha = 0.1 and lambda = 0.1369574.
```

```r
fit$bestTune
```

```
  alpha    lambda
2   0.1 0.1369574
```

```r
pred = predict.train(fit, x.test)
mean((y.test - pred)^2)
```

```
[1] 34.03507
```

```r
myGrid = expand.grid(alpha = seq(0, 1, by = 0.05), lambda = seq(0, 5,
    length = 20))
set.seed(100)
fit = train(x.train, y.train, method = "glmnet", trControl = myControl,
    tuneGrid = myGrid, metric = "RMSE", preProcess = c("center", "scale"))
fit$bestTune
```

```
   alpha    lambda
22  0.05 0.2631579
```

```r
bT = fit$bestTune
pred = predict(fit, x.test)
mean((y.test - pred)^2)
```

```
[1] 33.92637
```

```r
fitb = glmnet(x.train, y.train, alpha = bT$alpha, lambda = bT$lambda)
predict(fitb, type = "coefficient")[, 1]
```

```
 (Intercept)         crim           zn        indus         chas          nox 
31.038421071 -0.109631012  0.040434389  0.000000000  2.858174938 -9.214962177 
          rm          age          dis          rad          tax      ptratio 
 3.744455117 -0.011553488 -1.268536742  0.184159429 -0.006385713 -0.877625146 
       black        lstat 
 0.008236686 -0.608371389 
```

- We can minimize

$$-\text{loglikelihood} + \lambda\left((1-\alpha)\sum_{j=1}^{p}\beta_j^2 + \alpha\sum_{j=1}^{p}|\beta_j|\right) \tag{10}$$

- When $\alpha = 0$, it is the ridge regression.
- When $\alpha = 1$, it is the LASSO.
- If $\lambda = 0$, then the elastic net is the same as logistic regression model.

```
hdata = read.csv(file = "Heart.csv")
head(hdata)
```

4.1. Example Regularization for logistic regression.

```
  X Age Sex    ChestPain RestBP Chol Fbs RestECG MaxHR ExAng Oldpeak Slope Ca
1 1  63   1      typical    145  233   1       2   150     0     2.3     3  0
2 2  67   1 asymptomatic    160  286   0       2   108     1     1.5     2  3
3 3  67   1 asymptomatic    120  229   0       2   129     1     2.6     2  2
4 4  37   1   nonanginal    130  250   0       0   187     0     3.5     3  0
5 5  41   0   nontypical    130  204   0       2   172     0     1.4     1  0
6 6  56   1   nontypical    120  236   0       0   178     0     0.8     1  0
```

```
        Thal AHD
1      fixed  No
2     normal Yes
3 reversable Yes
4     normal  No
5     normal  No
6     normal  No
```

```
str(hdata)
```

```
'data.frame':   303 obs. of  15 variables:
 $ X        : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Age      : int  63 67 67 37 41 56 62 57 63 53 ...
 $ Sex      : int  1 1 1 1 0 1 0 0 1 1 ...
 $ ChestPain: chr  "typical" "asymptomatic" "asymptomatic" "nonanginal" ...
 $ RestBP   : int  145 160 120 130 130 120 140 120 130 140 ...
 $ Chol     : int  233 286 229 250 204 236 268 354 254 203 ...
 $ Fbs      : int  1 0 0 0 0 0 0 0 0 1 ...
 $ RestECG  : int  2 2 2 0 2 0 2 0 2 2 ...
 $ MaxHR    : int  150 108 129 187 172 178 160 163 147 155 ...
 $ ExAng    : int  0 1 1 0 0 0 0 1 0 1 ...
 $ Oldpeak  : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
 $ Slope    : int  3 2 2 3 1 1 3 1 2 3 ...
```

```
 $ Ca       : int  0 3 2 0 0 0 2 0 1 0 ...
 $ Thal     : chr  "fixed" "normal" "reversable" "normal" ...
 $ AHD      : chr  "No" "Yes" "Yes" "No" ...
```

```
hdata$X = NULL  # Remove ID variable X
sum(is.na(hdata))
```

```
[1] 6
```

```
hdata[!complete.cases(hdata), ]  #listwise view of missing observations
```

```
    Age Sex    ChestPain RestBP Chol Fbs RestECG MaxHR ExAng Oldpeak Slope Ca
88   53   0   nonanginal    128  216   0       2   115     0     0.0     1  0
167  52   1   nonanginal    138  223   0       0   169     0     0.0     1 NA
193  43   1 asymptomatic    132  247   1       2   143     1     0.1     2 NA
267  52   1 asymptomatic    128  204   1       0   156     1     1.0     2  0
288  58   1   nontypical    125  220   0       0   144     0     0.4     2 NA
303  38   1   nonanginal    138  175   0       0   173     0     0.0     1 NA
          Thal AHD
88        <NA>  No
167     normal  No
193 reversable Yes
267       <NA> Yes
```

```
288 reversable   No
303     normal   No
```

```r
apply(hdata, 2, function(x) sum(is.na(x)))  # columnwise view of missing
```

```
      Age       Sex ChestPain    RestBP      Chol       Fbs   RestECG     MaxHR
        0         0         0         0         0         0         0         0
    ExAng   Oldpeak     Slope        Ca      Thal       AHD
        0         0         0         4         2         0
```

```r
hdata = na.omit(hdata)
x = model.matrix(AHD ~ ., data = hdata)[, -1]
y = hdata$AHD
n = nrow(x)

set.seed(1)
train = sample(n, n/2)
x.train = x[train, ]
x.test = x[-train, ]
y.train = y[train]
y.test = y[-train]

# Ridge
```

```r
f = glmnet(x.train, y.train, alpha = 0, lambda = 1, family = "binomial")
f.out = cv.glmnet(x.train, y.train, alpha = 0, family = "binomial")
f.out$lambda.min
```

```
[1] 0.07042912
```

```r
f.out
```

```
Call:  cv.glmnet(x = x.train, y = y.train, alpha = 0, family = "binomial")

Measure: Binomial Deviance

    Lambda Index Measure      SE Nonzero
min 0.0704    88  0.8975 0.06299      16
1se 0.3425    71  0.9589 0.04747      16
```

```r
fit0 = glmnet(x.train, y.train, alpha = 0, family = "binomial")
predict(fit0, type = "coefficients", s = f.out$lambda.min)[1:17, ]
```

```
        (Intercept)                 Age                 Sex ChestPainnonanginal
       -1.597533108        -0.012040140         0.985294280        -0.638349870
ChestPainnontypical     ChestPaintypical              RestBP                Chol
```

```
  -0.653694195   -0.996617168    0.007383391    0.002824402 
           Fbs        RestECG          MaxHR         ExAng 
  -0.331500073    0.162832960   -0.011915101    0.797386353 
       Oldpeak          Slope             Ca    Thalnormal 
   0.314804980    0.442689698    0.612323455   -0.347242751 
Thalreversable 
   0.581601059 
```

```r
pred = predict(fit0, newx = x.test, s = f.out$lambda.min, type = "response")
pcl = ifelse(pred > 0.5, "Yes", "No")
mean(y.test != pcl)
```

```
[1] 0.1342282
```

```r
# LASSO
f.out1 = cv.glmnet(x.train, y.train, alpha = 1, family = "binomial")
f.out1$lambda.min
```

```
[1] 0.01174829
```

```r
fit1 = glmnet(x, y, alpha = 1, family = "binomial")
predict(fit1, type = "coefficients", s = f.out1$lambda.min)[1:17, ]
```

```
        (Intercept)                 Age                 Sex ChestPainnonanginal
       -1.935612564         0.000000000         0.855232678        -1.296979801
ChestPainnontypical    ChestPaintypical              RestBP                Chol
       -0.609855245        -1.251826859         0.011657027         0.001557653
                Fbs             RestECG               MaxHR               ExAng
       -0.180856790         0.165744593        -0.015542794         0.673966116
            Oldpeak               Slope                  Ca          Thalnormal
        0.299774198         0.401210705         0.943606934        -0.275557142
     Thalreversable
        1.072917003
```

```
pred1 = predict(fit1, newx = x.test, s = f.out1$lambda.min, type = "response")
pcl1 = ifelse(pred1 > 0.5, "Yes", "No")
mean(y.test != pcl1)
```

```
[1] 0.1208054
```

```
mycontrol = trainControl(method = "cv", number = 10, allowParallel = T,
    classProbs = T)
set.seed(1)
hfit = train(x.train, y.train, method = "glmnet", trControl = mycontrol,
    tuneLength = 3, family = "binomial", PreProcess = c("center", "scale"))
hfit
```

```
glmnet

148 samples
 16 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 134, 133, 134, 132, 134, 133, ...
Resampling results across tuning parameters:

  alpha  lambda        Accuracy   Kappa
  0.10   0.0004612472  0.7848214  0.5627409
  0.10   0.0046124725  0.8052976  0.6027790
  0.10   0.0461247245  0.8120238  0.6156782
  0.55   0.0004612472  0.7848214  0.5627409
  0.55   0.0046124725  0.8052976  0.6027790
  0.55   0.0461247245  0.7972619  0.5870350
  1.00   0.0004612472  0.7848214  0.5627409
  1.00   0.0046124725  0.8052976  0.6027790
  1.00   0.0461247245  0.7843452  0.5572387

Accuracy was used to select the optimal model using the largest value.
```

```
The final values used for the model were alpha = 0.1 and lambda = 0.04612472.
```

```
hpred = predict(hfit, x.test)
hpred
```

```
  [1] Yes Yes No  No  No  Yes No  Yes No  No  No  No  No  No  Yes Yes No  No
 [19] Yes Yes Yes No  No  No  Yes Yes Yes No  Yes No  Yes Yes No  No  Yes Yes
 [37] Yes No  Yes Yes No  No  Yes No  No  Yes Yes Yes No  Yes Yes No  No  No
 [55] No  Yes Yes Yes No  Yes Yes No  Yes No  No  No  Yes Yes No  No  No  Yes
 [73] No  No  Yes Yes Yes Yes Yes No  Yes No  No  No  No  Yes No  Yes Yes No
 [91] No  Yes Yes Yes No  No  Yes Yes No  No  No  No  No  No  No  No  No  Yes
[109] Yes Yes No  Yes No  No  No  Yes No  No  No  No  No  No  Yes No  Yes No
[127] Yes No  No  No  No  No  No  Yes No  No  No  Yes No  No  No  No  Yes No
[145] Yes No  Yes Yes No
Levels: No Yes
```

```
mean(y.test != hpred)  #test classification error
```

```
[1] 0.1409396
```

```
mean(y.test == hpred)  #test accuracy
```

```
[1] 0.8590604
```

```r
predict(hfit, x.test, type = "prob")[1:10, ]
```

```
           No        Yes
1  0.02642216 0.97357784
2  0.02994174 0.97005826
3  0.53547380 0.46452620
4  0.93016163 0.06983837
5  0.89960407 0.10039593
6  0.20623327 0.79376673
7  0.78807341 0.21192659
8  0.08702709 0.91297291
9  0.63576416 0.36423584
10 0.85121223 0.14878777
```

```r
hfit$bestTune
```

```
  alpha     lambda
3   0.1 0.04612472
```

```r
h = glmnet(x.train, y.train, alpha = hfit$bestTune[1], family = "binomial",
    lambda = hfit$bestTune[2])
coef(h)[, 1]
```

```
        (Intercept)                 Age                 Sex ChestPainnonanginal
       -2.055151826        -0.011116266         1.131958985        -0.711472330
ChestPainnontypical    ChestPaintypical              RestBP                Chol
       -0.694273575        -1.107106809         0.007496802         0.003128645
                Fbs             RestECG               MaxHR               ExAng
       -0.298653659         0.161353192        -0.011625511         0.851453204
            Oldpeak               Slope                  Ca          Thalnormal
        0.335887925         0.467829230         0.693247063        -0.257630167
     Thalreversable
        0.659852129
```

5. Tuning hyperparameters

- Grid search
- Random search
- Other methods such as Bayesian optimization, Tree-structured Parzen estimators (TPE), Hyperband, Population-based training (PBT), and Bayesian optimization and hyperband (BOHB)

```
library(caret)
library(glmnet)
library(doParallel)
nc = detectCores()
registerDoParallel(nc)

hdata = read.csv(file = "Heart.csv")
hdata$X = NULL
hdata = na.omit(hdata)
x = model.matrix(AHD ~ ., data = hdata)[, -1]
y = hdata$AHD
n = nrow(x)

set.seed(1)
```

```r
train = sample(n, n/2)
x.train = x[train, ]
x.test = x[-train, ]
y.train = y[train]
y.test = y[-train]



# Grid search
mycontrol = trainControl(method = "cv", number = 10, allowParallel = T,
    classProbs = T, search = "grid")
myGrid = expand.grid(alpha = seq(0, 1, by = 0.2), lambda = exp(seq(-2,
    1, length = 5)))
set.seed(1)
hfit = train(x.train, y.train, method = "glmnet", trControl = mycontrol,
    tuneGrid = myGrid, family = "binomial", PreProcess = c("center",
        "scale"))
hfit
```

5.1. Example of grid and random searches for Heart data.

```
glmnet

148 samples
```

```
 16 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 134, 133, 134, 132, 134, 133, ...
Resampling results across tuning parameters:

  alpha  lambda     Accuracy   Kappa
  0.0    0.1353353  0.7919048  0.5765513
  0.0    0.2865048  0.7713690  0.5315927
  0.0    0.6065307  0.7914286  0.5686491
  0.0    1.2840254  0.8043452  0.5889134
  0.0    2.7182818  0.7625000  0.4850702
  0.2    0.1353353  0.7910119  0.5735393
  0.2    0.2865048  0.7900595  0.5646958
  0.2    0.6065307  0.7442262  0.4524892
  0.2    1.2840254  0.5610714  0.0000000
  0.2    2.7182818  0.5610714  0.0000000
  0.4    0.1353353  0.8039286  0.5965809
  0.4    0.2865048  0.7785714  0.5285347
  0.4    0.6065307  0.5610714  0.0000000
  0.4    1.2840254  0.5610714  0.0000000
```

```
  0.4    2.7182818  0.5610714  0.0000000
  0.6    0.1353353  0.7713690  0.5259513
  0.6    0.2865048  0.6769048  0.2976990
  0.6    0.6065307  0.5610714  0.0000000
  0.6    1.2840254  0.5610714  0.0000000
  0.6    2.7182818  0.5610714  0.0000000
  0.8    0.1353353  0.7571429  0.4896586
  0.8    0.2865048  0.5610714  0.0000000
  0.8    0.6065307  0.5610714  0.0000000
  0.8    1.2840254  0.5610714  0.0000000
  0.8    2.7182818  0.5610714  0.0000000
  1.0    0.1353353  0.7361310  0.4389396
  1.0    0.2865048  0.5610714  0.0000000
  1.0    0.6065307  0.5610714  0.0000000
  1.0    1.2840254  0.5610714  0.0000000
  1.0    2.7182818  0.5610714  0.0000000

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0 and lambda = 1.284025.
```

```
# Random search
mycontrol1 = trainControl(method = "cv", number = 10, allowParallel = T,
    classProbs = T, search = "random")
```

```r
set.seed(1)
hfit1 = train(x.train, y.train, method = "glmnet", trControl = mycontrol1,
    tuneLength = 10, family = "binomial", PreProcess = c("center", "scale"))
hfit1
```

```
glmnet

148 samples
 16 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 134, 133, 134, 132, 134, 133, ...
Resampling results across tuning parameters:

  alpha       lambda       Accuracy   Kappa
  0.06178627  1.076828036  0.7571429  0.4797469
  0.17655675  0.006605359  0.8052976  0.6027790
  0.20168193  1.005517688  0.5610714  0.0000000
  0.20597457  4.441872568  0.5610714  0.0000000
  0.57285336  0.476747410  0.5610714  0.0000000
  0.62911404  0.029986508  0.7914881  0.5760083
```

```
  0.66079779  7.437301367  0.5610714  0.0000000
  0.89838968  0.086574751  0.7972619  0.5820211
  0.90820779  0.031106251  0.7767857  0.5438193
  0.94467527  0.628088869  0.5610714  0.0000000

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0.1765568 and lambda
 = 0.006605359.
```