An R Markdown document converted

Lab 6

- Best subset selection, forward/backword selection
- Ridge regression
- Lasso

Generate the data

- Dataset size $n = 500$
- Generate 20 predictors $X_1, X_2, \ldots, X_{20}$ and $X_i \sim N(0,1)$ for $i = 1, \ldots, 20$.
- $Y = X_1 + X_2 + X_3 + X_4 + X_5 + \epsilon$ with $\epsilon \sim N(0,1)$

```
set.seed(1)
n = 500
p = 20
x = matrix(rnorm(n * p), nrow=n, ncol=p)
y = x[, 1] + x[, 2] + x[, 3] + x[, 4] + x[, 5] + rnorm(n)
```

```
head(x)
```

```
##            [,1]        [,2]        [,3]       [,4]        [,5]       [,6]
## [1,] -0.6264538  0.07730312  1.13496509  0.8500435 -0.88614959 -1.8054836
## [2,]  0.1836433 -0.29686864  1.11193185 -0.9253130 -1.92225490 -0.6780407
## [3,] -0.8356286 -1.18324224 -0.87077763  0.8935812  1.61970074 -0.4733581
## [4,]  1.5952808  0.01129269  0.21073159 -0.9410097  0.51926990  1.0274171
## [5,]  0.3295078  0.99160104  0.06939565  0.5389521 -0.05584993 -0.5973876
## [6,] -0.8204684  1.59396745 -1.66264885 -0.1819744  0.69641761  1.1598494
##            [,7]       [,8]       [,9]      [,10]      [,11]      [,12]
## [1,]  0.7391149  0.5205997 -1.1346302  1.5579537 -1.5163733 -1.1378698
## [2,]  0.3866087  0.3775619  0.7645571 -0.7292970  0.6291412 -0.9518105
## [3,]  1.2963972 -0.6236588  0.5707101 -1.5039509 -1.6781940  1.6192595
## [4,] -0.8035584 -0.5726105 -1.3516939 -0.5667870  1.1797811  0.1678136
## [5,] -1.6026257  0.3125012 -2.0298855 -2.1044536  1.1176545 -0.9081778
## [6,]  0.9332510 -0.7074278  0.5904787  0.5307319 -1.2377359  1.3417959
##            [,13]      [,14]      [,15]      [,16]      [,17]       [,18]
## [1,] -0.61882708  0.8871888 -1.3254177 -0.7948034  0.2637034  0.99010104
## [2,] -1.10942196 -0.3776280  0.9519797  0.6060846 -0.8294518 -0.06643542
## [3,] -2.17033523  0.1140808  0.8600044 -1.0624674 -1.4616348  0.25797379
## [4,] -0.03130307 -1.7242394  1.0607903  1.0192005  1.6839902 -0.62145348
## [5,] -0.26039848  0.7436886 -0.3505840  0.1776102 -1.5443243 -0.77645263
## [6,]  0.53443047 -0.8857317 -0.1307656 -1.0309747 -0.1908871  0.31534275
##           [,19]      [,20]
## [1,] -1.2171201  1.2980378
```

```
## [2,] -0.9462293 -1.4276760
## [3,]  0.0914098  0.2427872
## [4,]  0.7013513 -0.2107006
## [5,]  0.6734224  0.0801386
## [6,]  1.2655534  1.5460849
```

```
dim(x)
```

```
## [1] 500  20
```

```
simulated_data = data.frame(x, y)
```

```
head(simulated_data)
```

```
##           X1          X2          X3         X4          X5         X6
## 1 -0.6264538  0.07730312  1.13496509  0.8500435 -0.88614959 -1.8054836
## 2  0.1836433 -0.29686864  1.11193185 -0.9253130 -1.92225490 -0.6780407
## 3 -0.8356286 -1.18324224 -0.87077763  0.8935812  1.61970074 -0.4733581
## 4  1.5952808  0.01129269  0.21073159 -0.9410097  0.51926990  1.0274171
## 5  0.3295078  0.99160104  0.06939565  0.5389521 -0.05584993 -0.5973876
## 6 -0.8204684  1.59396745 -1.66264885 -0.1819744  0.69641761  1.1598494
##           X7         X8         X9        X10        X11        X12         X13
## 1  0.7391149  0.5205997 -1.1346302  1.5579537 -1.5163733 -1.1378698 -0.61882708
## 2  0.3866087  0.3775619  0.7645571 -0.7292970  0.6291412 -0.9518105 -1.10942196
## 3  1.2963972 -0.6236588  0.5707101 -1.5039509 -1.6781940  1.6192595 -2.17033523
## 4 -0.8035584 -0.5726105 -1.3516939 -0.5667870  1.1797811  0.1678136 -0.03130307
## 5 -1.6026257  0.3125012 -2.0298855 -2.1044536  1.1176545 -0.9081778 -0.26039848
## 6  0.9332510 -0.7074278  0.5904787  0.5307319 -1.2377359  1.3417959  0.53443047
##          X14        X15        X16        X17         X18        X19        X20
## 1  0.8871888 -1.3254177 -0.7948034  0.2637034  0.99010104 -1.2171201  1.2980378
## 2 -0.3776280  0.9519797  0.6060846 -0.8294518 -0.06643542 -0.9462293 -1.4276760
## 3  0.1140808  0.8600044 -1.0624674 -1.4616348  0.25797379  0.0914098  0.2427872
## 4 -1.7242394  1.0607903  1.0192005  1.6839902 -0.62145348  0.7013513 -0.2107006
## 5  0.7436886 -0.3505840  0.1776102 -1.5443243 -0.77645263  0.6734224  0.0801386
## 6 -0.8857317 -0.1307656 -1.0309747 -0.1908871  0.31534275  1.2655534  1.5460849
##            y
## 1 -0.2546233
## 2 -2.9053870
## 3 -1.4117623
## 4  0.2100049
## 5  1.3731671
## 6 -0.8996953
```

Fit a linear regression

```
lm_fit <- lm(y ~ ., data=simulated_data)
```

```
summary(lm_fit)
```

```
##
## Call:
## lm(formula = y ~ ., data = simulated_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.78358 -0.63078 -0.01685  0.64726  2.73856
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.0004095  0.0426961  -0.010    0.992
## X1           1.0083122  0.0422666  23.856   <2e-16 ***
## X2           0.9895599  0.0405302  24.415   <2e-16 ***
## X3           0.9851015  0.0427947  23.019   <2e-16 ***
## X4           0.9784761  0.0401118  24.394   <2e-16 ***
## X5           1.0454887  0.0411896  25.382   <2e-16 ***
## X6          -0.0243070  0.0423016  -0.575    0.566
## X7          -0.0305170  0.0408759  -0.747    0.456
## X8          -0.0293507  0.0424286  -0.692    0.489
## X9          -0.0124170  0.0442335  -0.281    0.779
## X10         -0.0340599  0.0433143  -0.786    0.432
## X11         -0.0159119  0.0461149  -0.345    0.730
## X12         -0.0276688  0.0428158  -0.646    0.518
## X13          0.0792420  0.0399635   1.983    0.048 *
## X14         -0.0160056  0.0429318  -0.373    0.709
## X15         -0.0060911  0.0432060  -0.141    0.888
## X16          0.0035515  0.0432319   0.082    0.935
## X17         -0.0001048  0.0418376  -0.003    0.998
## X18         -0.0445360  0.0449937  -0.990    0.323
## X19         -0.0362999  0.0441091  -0.823    0.411
## X20         -0.0641220  0.0443336  -1.446    0.149
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9442 on 479 degrees of freedom
## Multiple R-squared:  0.8628, Adjusted R-squared:  0.8571
## F-statistic: 150.6 on 20 and 479 DF,  p-value: < 2.2e-16
```

Looks like least squares have already done a decent job.

Let's try best subset selection first.

```
library(leaps)
```

- `regsubsets` function performs best subset selection by identifying the best model that contains a given number of predictors, where *best* is quantified using RSS.
- `summary` command outputs the best set of variables for each model size

```
regfit.full <- regsubsets(y ~ ., data=simulated_data)
```

```r
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(y ~ ., data = simulated_data)
## 20 Variables  (and intercept)
##     Forced in Forced out
## X1      FALSE      FALSE
## X2      FALSE      FALSE
## X3      FALSE      FALSE
## X4      FALSE      FALSE
## X5      FALSE      FALSE
## X6      FALSE      FALSE
## X7      FALSE      FALSE
## X8      FALSE      FALSE
## X9      FALSE      FALSE
## X10     FALSE      FALSE
## X11     FALSE      FALSE
## X12     FALSE      FALSE
## X13     FALSE      FALSE
## X14     FALSE      FALSE
## X15     FALSE      FALSE
## X16     FALSE      FALSE
## X17     FALSE      FALSE
## X18     FALSE      FALSE
## X19     FALSE      FALSE
## X20     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          X1  X2  X3  X4  X5  X6  X7  X8  X9  X10 X11 X12 X13 X14 X15 X16 X17
## 1  ( 1 ) " " " " " " "*" " " " " " " " " " " " " " " " " " " " " " " " " " "
## 2  ( 1 ) " " " " " " "*" "*" " " " " " " " " " " " " " " " " " " " " " " " "
## 3  ( 1 ) "*" " " " " "*" "*" " " " " " " " " " " " " " " " " " " " " " " " "
## 4  ( 1 ) "*" "*" " " "*" "*" " " " " " " " " " " " " " " " " " " " " " " " "
## 5  ( 1 ) "*" "*" "*" "*" "*" " " " " " " " " " " " " " " " " " " " " " " " "
## 6  ( 1 ) "*" "*" "*" "*" "*" " " " " " " " " " " " " " " "*" " " " " " " " "
## 7  ( 1 ) "*" "*" "*" "*" "*" " " " " " " " " " " " " " " "*" " " " " " " " "
## 8  ( 1 ) "*" "*" "*" "*" "*" " " " " " " " " " " " " " " "*" " " " " " " " "
##          X18 X19 X20
## 1  ( 1 ) " " " " " "
## 2  ( 1 ) " " " " " "
## 3  ( 1 ) " " " " " "
## 4  ( 1 ) " " " " " "
## 5  ( 1 ) " " " " " "
## 6  ( 1 ) " " " " " "
## 7  ( 1 ) " " " " "*"
## 8  ( 1 ) "*" " " "*"
```

`nvmax` option can be used to set the number of variables considered by the best subset selection method.

```r
regfit.full = regsubsets(y ~ ., data=simulated_data, nvmax=20)
```

```r
reg.summary <- summary(regfit.full)
```

```r
names(reg.summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```r
round(reg.summary$rsq, 3)
```

```
##  [1] 0.203 0.409 0.553 0.703 0.860 0.861 0.862 0.862 0.862 0.862 0.862 0.862
## [13] 0.863 0.863 0.863 0.863 0.863 0.863 0.863 0.863
```

Can we use R-squared/RSS to select among models with different number of predictors?

What metric can we use to select different models?

- AIC/BIC
- Cp
- Adjusted R squared

```r
par(mfrow=c(2, 2))
plot(reg.summary$rss, xlab="Number of variables", ylab = "RSS", type="l")

plot(reg.summary$adjr2, xlab="Number of variables", ylab= "Adjusted Rsq", type="l")
which.max(reg.summary$adjr2)
```

```
## [1] 7
```

```r
points(7, reg.summary$adjr2[7], col="red", cex=2, pch=20)

plot(reg.summary$cp, xlab="Number of variables", ylab= "Cp", type="l")
which.min(reg.summary$cp)
```

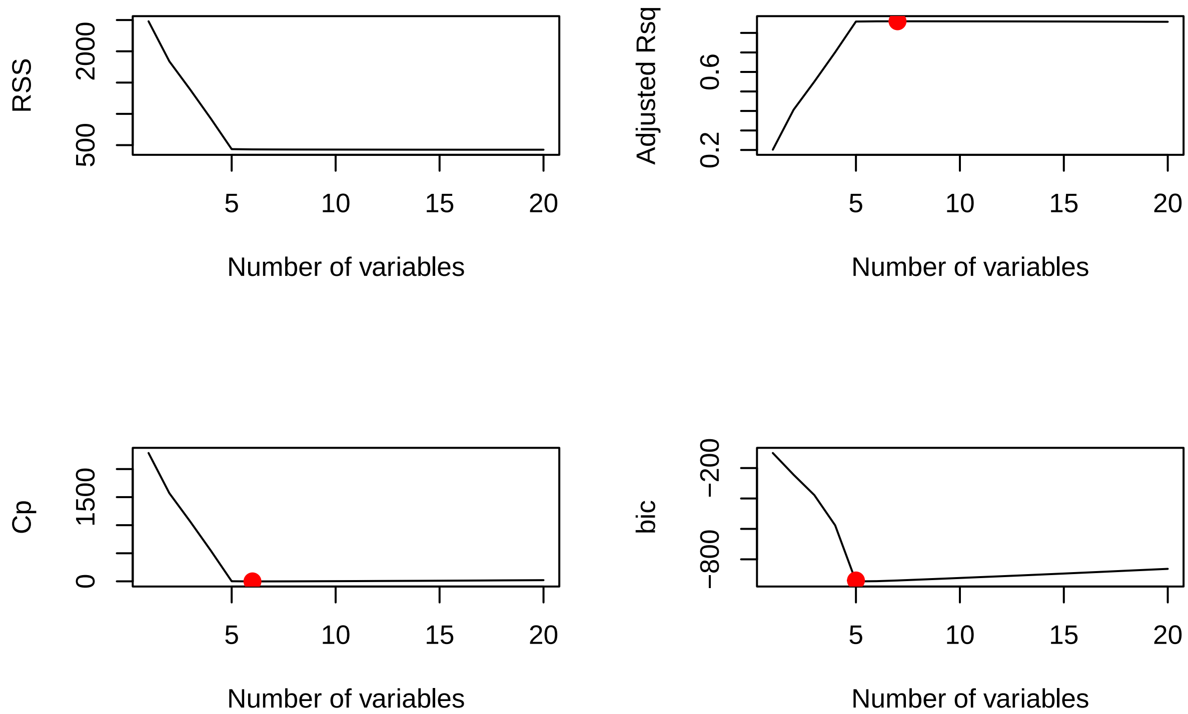
```
## [1] 6
```

```r
points(6, reg.summary$cp[7], col="red", cex=2, pch=20)

plot(reg.summary$bic, xlab="Number of variables", ylab= "bic", type="l")
which.min(reg.summary$bic)
```

```
## [1] 5
```

```r
points(5, reg.summary$bic[7], col="red", cex=2, pch=20)
```



Recall that BIC prefers simpler model

```
coef(regfit.full, 6)
```

```
##   (Intercept)            X1            X2            X3            X4            X5
## -0.001200048  1.011285561  0.987567524  0.978273655  0.974696452  1.047597425
##           X13
##   0.083728939
```

Forward and backward stepwise selection

- `regsubsets` function can also be used for forward and backward selection.
- Set argument `method="forward"` or `method="backward"`

```
regfit.fwd <- regsubsets(y ~ ., data = simulated_data, nvmax = 20, method="forward")

regfit.bwd <- regsubsets(y ~ ., data = simulated_data, nvmax = 20, method="backward")

summary(regfit.bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(y ~ ., data = simulated_data, nvmax = 20,
##     method = "backward")
## 20 Variables  (and intercept)
##     Forced in Forced out
## X1      FALSE      FALSE
```

```
## X2      FALSE      FALSE
## X3      FALSE      FALSE
## X4      FALSE      FALSE
## X5      FALSE      FALSE
## X6      FALSE      FALSE
## X7      FALSE      FALSE
## X8      FALSE      FALSE
## X9      FALSE      FALSE
## X10     FALSE      FALSE
## X11     FALSE      FALSE
## X12     FALSE      FALSE
## X13     FALSE      FALSE
## X14     FALSE      FALSE
## X15     FALSE      FALSE
## X16     FALSE      FALSE
## X17     FALSE      FALSE
## X18     FALSE      FALSE
## X19     FALSE      FALSE
## X20     FALSE      FALSE
## 1 subsets of each size up to 20
## Selection Algorithm: backward
##           X1  X2  X3  X4  X5  X6  X7  X8  X9  X10 X11 X12 X13 X14 X15 X16 X17
## 1  ( 1 )  " " " " " " "*" " " " " " " " " " " " " " " " " " " " " " " " " " "
## 2  ( 1 )  " " " " " " "*" "*" " " " " " " " " " " " " " " " " " " " " " " " "
## 3  ( 1 )  "*" " " " " "*" "*" " " " " " " " " " " " " " " " " " " " " " " " "
## 4  ( 1 )  "*" "*" " " "*" "*" " " " " " " " " " " " " " " " " " " " " " " " "
## 5  ( 1 )  "*" "*" "*" "*" "*" " " " " " " " " " " " " " " " " " " " " " " " "
## 6  ( 1 )  "*" "*" "*" "*" "*" " " " " " " " " " " " " " " "*" " " " " " " " "
## 7  ( 1 )  "*" "*" "*" "*" "*" " " " " " " " " " " " " " " "*" " " " " " " " "
## 8  ( 1 )  "*" "*" "*" "*" "*" " " " " " " " " " " " " " " "*" " " " " " " " "
## 9  ( 1 )  "*" "*" "*" "*" "*" " " " " " " " " " " " " " " "*" " " " " " " " "
## 10  ( 1 ) "*" "*" "*" "*" "*" " " "*" " " " " " " " " " " "*" " " " " " " " "
## 11  ( 1 ) "*" "*" "*" "*" "*" " " "*" " " " " "*" " " " " "*" " " " " " " " "
## 12  ( 1 ) "*" "*" "*" "*" "*" " " "*" "*" " " "*" " " " " "*" " " " " " " " "
## 13  ( 1 ) "*" "*" "*" "*" "*" " " "*" "*" " " "*" " " "*" "*" " " " " " " " "
## 14  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" " " "*" " " "*" "*" " " " " " " " "
## 15  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" " " "*" " " "*" "*" "*" " " " " " "
## 16  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" " " "*" "*" "*" "*" "*" " " " " " "
## 17  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " " "
## 18  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " "
## 19  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " "
## 20  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
##           X18 X19 X20
## 1  ( 1 )  " " " " " "
## 2  ( 1 )  " " " " " "
## 3  ( 1 )  " " " " " "
## 4  ( 1 )  " " " " " "
## 5  ( 1 )  " " " " " "
## 6  ( 1 )  " " " " " "
## 7  ( 1 )  " " " " "*"
## 8  ( 1 )  "*" " " "*"
## 9  ( 1 )  "*" "*" "*"
## 10  ( 1 ) "*" "*" "*"
## 11  ( 1 ) "*" "*" "*"
```

```
## 12  ( 1 ) "*" "*" "*"
## 13  ( 1 ) "*" "*" "*"
## 14  ( 1 ) "*" "*" "*"
## 15  ( 1 ) "*" "*" "*"
## 16  ( 1 ) "*" "*" "*"
## 17  ( 1 ) "*" "*" "*"
## 18  ( 1 ) "*" "*" "*"
## 19  ( 1 ) "*" "*" "*"
## 20  ( 1 ) "*" "*" "*"
```

Compare the results of best subset selection and forward selection

```
coef(regfit.full, 6)
```

```
##  (Intercept)           X1           X2           X3           X4           X5
## -0.001200048  1.011285561  0.987567524  0.978273655  0.974696452  1.047597425
##          X13
##  0.083728939
```

```
coef(regfit.fwd, 6)
```

```
##  (Intercept)           X1           X2           X3           X4           X5
## -0.001200048  1.011285561  0.987567524  0.978273655  0.974696452  1.047597425
##          X13
##  0.083728939
```

Recall that Cp, BIC, adjusted R-squared are all indirect estimates of test error

We can also use cross validation to directly estimate test errors and select variables

- For cross validation, we need to perform both variable selection and also estimate the test error.
- We should only use **training dataset** for both the variables selections and model fitting.
- If full data is being used to select variables, the validation set errors and cross-validation errors will not be accurate estimates of the test errors.

```
set.seed(123)
train <- sample(c(TRUE, FALSE), n, replace=T)
test <- !train
```

```
regfit.best <- regsubsets(y ~ ., data=simulated_data, nvmax=20)
```

We now compute the validation set error for the best model of each model size

- We first need to make a model matrix from the test set.
- meaning that we want to make an "X" matrix for the model.
- Use `model.matrix` function to extract the model matrix from a formula and dataset.

```r
test.mat <- model.matrix(y ~ ., data = simulated_data[test, ])
```

```r
head(test.mat)
```

```
##    (Intercept)         X1          X2         X3         X4          X5
## 4            1  1.5952808  0.01129269  0.2107316 -0.9410097  0.51926990
## 6            1 -0.8204684  1.59396745 -1.6626489 -0.1819744  0.69641761
## 7            1  0.4874291 -1.37271127  0.8108400  0.8917676  0.05351568
## 8            1  0.7383247 -0.24961093 -1.9123458  1.3292082 -1.31028350
## 11           1  1.5117812 -2.52850069 -0.5408727 -1.7997724 -0.31278658
## 12           1  0.3898432 -0.93590256 -0.2163758 -0.2627051 -1.05823571
##            X6          X7          X8         X9        X10        X11
## 4   1.0274171 -0.80355836 -0.57261053 -1.3516939 -0.5667870  1.1797811
## 6   1.1598494  0.93325097 -0.70742783  0.5904787  0.5307319 -1.2377359
## 7  -1.3332269  1.80608925  0.52120349 -1.4130700  1.6176841 -1.2301645
## 8  -0.9257557 -0.05650363  0.44818798  1.6103416  1.1845319  0.5977909
## 11 -0.5887322  0.50228462  0.02961402 -1.2555731  0.6692327 -0.8076750
## 12 -0.1153843  0.42991421  2.05234415 -1.3843471 -1.2830074  0.1145392
##            X12         X13        X14        X15         X16        X17
## 4   0.16781359 -0.03130307 -1.7242394  1.0607903  1.01920051  1.6839902
## 6   1.34179588  0.53443047 -0.8857317 -0.1307656 -1.03097474 -0.1908871
## 7   0.02204345 -0.55943937  0.3266106  0.7635859  1.13265221  1.0162117
## 8  -0.20700545  1.60837019 -1.7084777 -0.4939057  0.04034495  0.5471262
## 11  0.31403447 -1.03940831  0.8540098  0.6345931  0.61495981  1.8107821
## 12 -0.96820329 -0.36338204 -0.2764356  1.8166802  0.77404772 -0.1108024
##           X18        X19        X20
## 4  -0.6214535  0.7013513 -0.2107006
## 6   0.3153428  1.2655534  1.5460849
## 7  -0.9636937 -1.4450221 -0.2158588
## 8  -1.1563552  1.4154183 -0.7709604
## 11 -0.8005625  0.4502331  0.4181226
## 12 -0.2126575  0.9476328 -0.7995130
```

```r
dim(test.mat)
```

```
## [1] 239  21
```

```r
val.errors <- rep(0, 20)
for (i in 1:20) {
    coefi <- coef(regfit.best, id=i)
    pred <- test.mat[, names(coefi)] %*% coefi
    val.errors[i] <- mean((simulated_data$y[test] - pred)^2)
}
```

```r
round(val.errors, 3)
```

```
##  [1] 4.684 3.246 2.162 1.554 0.856 0.849 0.849 0.850 0.852 0.848 0.847 0.849
## [13] 0.850 0.848 0.848 0.848 0.847 0.847 0.847 0.847
```

```
which.min(val.errors)
```

```
## [1] 20
```

```
predict.regsubsets <- function(objects, newdata, id, ...) {
    form <- as.formula(objects$call[[2]])
    mat <- model.matrix(form, newdata)
    coefi <- coef(objects, id=id)
    xvars <- names(coefi)
    mat[, xvars] %*% coefi
}
```

Let's try 10-fold cross validation

```
k <- 10
set.seed(1)
folds <- sample(rep(1:k, length=n))
cv.errors <- matrix(NA, k, 20)
```

```
length(which(folds == 8))
```

```
## [1] 50
```

```
for (j in 1:k) {
    best.fit <- regsubsets(y ~ ., data=simulated_data[folds != j, ], nvmax=20)
    for (i in 1:20) {
        pred <- predict(best.fit, simulated_data[folds == j, ], id=i)
        cv.errors[j, i] <- mean((simulated_data$y[folds == j] - pred)^2)
    }
}
```

```
mean.cv.errors <- apply(cv.errors, 2, mean)
```

```
round(mean.cv.errors, 3)
```

```
##  [1] 5.273 3.719 3.081 2.084 0.895 0.900 0.909 0.917 0.921 0.929 0.932 0.934
## [13] 0.938 0.936 0.935 0.935 0.934 0.935 0.935 0.935
```

```
which.min(mean.cv.errors)
```

```
## [1] 5
```

Next we can perform best subset selection on the full dataset to get the 5-variable model.

- It is important that we make use of the full data set in order to obtain more accurate coefficient estimates.

```
reg.best.full <- regsubsets(y ~., data=simulated_data, nvmax=5)
```

```
coef(reg.best.full, 5)
```

```
##   (Intercept)            X1            X2            X3            X4
## -0.0009995745  1.0135772970  0.9899400896  0.9820521359  0.9747479335
##            X5
##  1.0475666292
```

Ridge regression and Lasso

- We will use the package `glmnet` to perform ridge regression and lasso.
- The main function is also called `glmnet`.
- So far we have been using the formula syntax `y ~ x`. For `glmnet` function, we pass in a `x` matrix and `y` vector.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
set.seed(1)
n = 100
p = 200
x = matrix(rnorm(n * p), nrow=n, ncol=p)
y = x[, 1] + x[, 2] + x[, 3] + x[, 4] + x[, 5] + rnorm(n)
```

Ridge regression

- `glmnet` function has an `alpha` argument that determines what type of model is fit
- `alpha = 0` means that a ridge regression model is fit
- `alpha = 1` means that a lasso model is fit.

```
grid <- 10^seq(10, -2, length=100)
ridge.mod <- glmnet(x, y, alpha=0, lambda=grid)
```

By default, `glmnet` function performs ridge regression for an automatically selected range of λ values. Here, we choose a grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$.

```
ridge.mod
```

```
##
## Call:  glmnet(x = x, y = y, alpha = 0, lambda = grid)
##
##      Df  %Dev    Lambda
## 1   200  0.00 1.000e+10
```

```
## 2   200   0.00 7.565e+09
## 3   200   0.00 5.722e+09
## 4   200   0.00 4.329e+09
## 5   200   0.00 3.275e+09
## 6   200   0.00 2.477e+09
## 7   200   0.00 1.874e+09
## 8   200   0.00 1.417e+09
## 9   200   0.00 1.072e+09
## 10  200   0.00 8.111e+08
## 11  200   0.00 6.136e+08
## 12  200   0.00 4.642e+08
## 13  200   0.00 3.511e+08
## 14  200   0.00 2.656e+08
## 15  200   0.00 2.009e+08
## 16  200   0.00 1.520e+08
## 17  200   0.00 1.150e+08
## 18  200   0.00 8.697e+07
## 19  200   0.00 6.579e+07
## 20  200   0.00 4.977e+07
## 21  200   0.00 3.765e+07
## 22  200   0.00 2.848e+07
## 23  200   0.00 2.154e+07
## 24  200   0.00 1.630e+07
## 25  200   0.00 1.233e+07
## 26  200   0.00 9.326e+06
## 27  200   0.00 7.055e+06
## 28  200   0.00 5.337e+06
## 29  200   0.00 4.037e+06
## 30  200   0.00 3.054e+06
## 31  200   0.00 2.310e+06
## 32  200   0.00 1.748e+06
## 33  200   0.00 1.322e+06
## 34  200   0.00 1.000e+06
## 35  200   0.00 7.565e+05
## 36  200   0.00 5.722e+05
## 37  200   0.00 4.329e+05
## 38  200   0.00 3.275e+05
## 39  200   0.01 2.477e+05
## 40  200   0.01 1.874e+05
## 41  200   0.01 1.417e+05
## 42  200   0.01 1.072e+05
## 43  200   0.02 8.111e+04
## 44  200   0.02 6.136e+04
## 45  200   0.03 4.642e+04
## 46  200   0.04 3.511e+04
## 47  200   0.05 2.656e+04
## 48  200   0.07 2.009e+04
## 49  200   0.10 1.520e+04
## 50  200   0.13 1.150e+04
## 51  200   0.17 8.697e+03
## 52  200   0.22 6.579e+03
## 53  200   0.29 4.977e+03
## 54  200   0.39 3.765e+03
## 55  200   0.51 2.848e+03
```

```
## 56  200   0.67 2.154e+03
## 57  200   0.89 1.630e+03
## 58  200   1.17 1.233e+03
## 59  200   1.54 9.330e+02
## 60  200   2.02 7.060e+02
## 61  200   2.66 5.340e+02
## 62  200   3.49 4.040e+02
## 63  200   4.56 3.050e+02
## 64  200   5.95 2.310e+02
## 65  200   7.72 1.750e+02
## 66  200   9.98 1.320e+02
## 67  200  12.80 1.000e+02
## 68  200  16.30 7.600e+01
## 69  200  20.53 5.700e+01
## 70  200  25.56 4.300e+01
## 71  200  31.36 3.300e+01
## 72  200  37.85 2.500e+01
## 73  200  44.88 1.900e+01
## 74  200  52.19 1.400e+01
## 75  200  59.50 1.100e+01
## 76  200  66.52 8.000e+00
## 77  200  73.00 6.000e+00
## 78  200  78.74 5.000e+00
## 79  200  83.66 4.000e+00
## 80  200  87.73 3.000e+00
## 81  200  91.00 2.000e+00
## 82  200  93.55 2.000e+00
## 83  200  95.49 1.000e+00
## 84  200  96.91 1.000e+00
## 85  200  97.93 1.000e+00
## 86  200  98.64 0.000e+00
## 87  200  99.13 0.000e+00
## 88  200  99.45 0.000e+00
## 89  200  99.66 0.000e+00
## 90  200  99.79 0.000e+00
## 91  200  99.87 0.000e+00
## 92  200  99.92 0.000e+00
## 93  200  99.95 0.000e+00
## 94  200  99.97 0.000e+00
## 95  200  99.98 0.000e+00
## 96  200  99.99 0.000e+00
## 97  200  99.99 0.000e+00
## 98  200 100.00 0.000e+00
## 99  200 100.00 0.000e+00
## 100 200 100.00 0.000e+00
```

To extract the coefficient estimates, it is stored in a matrix that can be accessed by function `coef`.

```
dim(coef(ridge.mod))
```

```
## [1] 201 100
```

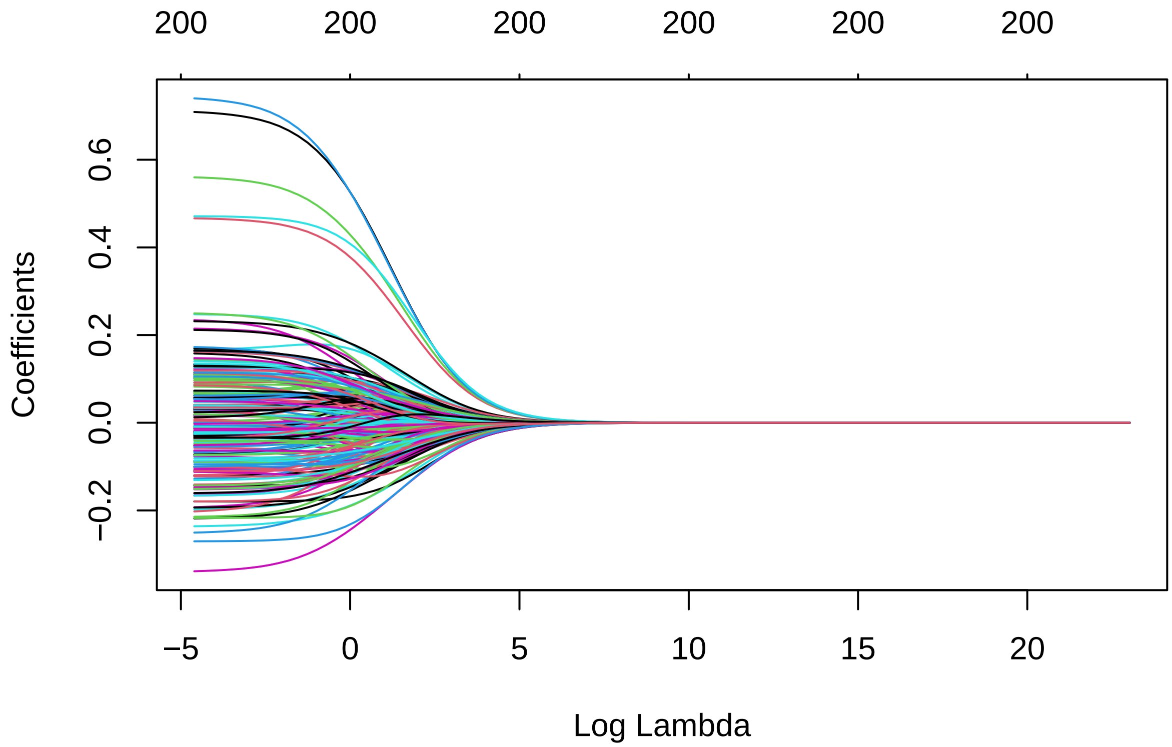
```r
coef(ridge.mod)[, 80]
```

```
##   (Intercept)            V1            V2            V3            V4 
##  0.2159537295  0.3889707319  0.2962192710  0.3309663943  0.3853699354 
##            V5            V6            V7            V8            V9 
##  0.3337783539 -0.0685354504 -0.0251793325  0.0487926601  0.0821107080 
##           V10           V11           V12           V13           V14 
##  0.0017434222 -0.1057524733  0.0011517642  0.0413562787 -0.0379104872 
##           V15           V16           V17           V18           V19 
##  0.0100188461 -0.0474594746  0.1370671883 -0.0629006653  0.0308132111 
##           V20           V21           V22           V23           V24 
## -0.0695455521  0.0640641579  0.0257579479 -0.0848988639  0.0097734724 
##           V25           V26           V27           V28           V29 
##  0.0398275669 -0.0583699585 -0.0694365445 -0.0904977472  0.0025647970 
##           V30           V31           V32           V33           V34 
## -0.0668834518  0.0067587789 -0.0678761537 -0.0302889510  0.0188291878 
##           V35           V36           V37           V38           V39 
##  0.0042341351 -0.0086840106  0.0862445346 -0.0784941947 -0.0228879729 
##           V40           V41           V42           V43           V44 
##  0.0750730754 -0.0425150949 -0.0772766992 -0.1113579377 -0.0280966154 
##           V45           V46           V47           V48           V49 
##  0.0002757242 -0.0133774578 -0.0567204327 -0.0695674525 -0.0805756737 
##           V50           V51           V52           V53           V54 
##  0.0407900807  0.0648223870 -0.0369160128  0.1329130880  0.0988308888 
##           V55           V56           V57           V58           V59 
##  0.0610663421 -0.0321340489 -0.0026645737 -0.0364281112 -0.0102564682 
##           V60           V61           V62           V63           V64 
##  0.0347290352 -0.0097022103  0.0249035656 -0.0247743557  0.0585563549 
##           V65           V66           V67           V68           V69 
##  0.0615817211  0.0691727566  0.0126174349 -0.0152064919  0.0171423892 
##           V70           V71           V72           V73           V74 
##  0.0432204967 -0.0184904353 -0.0687412226  0.0494109996  0.0563440667 
##           V75           V76           V77           V78           V79 
##  0.0182473622 -0.0853034738 -0.0255777862 -0.0304268514  0.0362958501 
##           V80           V81           V82           V83           V84 
##  0.0337394529  0.0444518316  0.0006030948 -0.1528836793  0.0475750052 
##           V85           V86           V87           V88           V89 
##  0.1431199201 -0.0366697625  0.0588236618  0.0003720083 -0.0750358819 
##           V90           V91           V92           V93           V94 
##  0.0427330267 -0.1484063589 -0.0925431820 -0.0870529247  0.0803877158 
##           V95           V96           V97           V98           V99 
##  0.0369715940 -0.1854080512  0.0242846012 -0.0935044817  0.0505622689 
##          V100          V101          V102          V103          V104 
## -0.0470796824  0.0361324647 -0.0246218998  0.0456451088  0.0345021138 
##          V105          V106          V107          V108          V109 
##  0.0327680377 -0.0072722174 -0.0530870943 -0.1016359021  0.0486483084 
##          V110          V111          V112          V113          V114 
## -0.0075463230 -0.0393603548 -0.0273332670 -0.0737774904  0.0172679980 
##          V115          V116          V117          V118          V119 
## -0.1159985426  0.0919370533  0.0033431571 -0.0078317156 -0.0337006339 
##          V120          V121          V122          V123          V124 
##  0.0935497343  0.0235372268 -0.0142433609 -0.1027724252 -0.0267511355 
##          V125          V126          V127          V128          V129
```

```
##  0.0200433251 -0.0727138656 -0.0993079281  0.0867330599  0.0597873344
##          V130          V131          V132          V133          V134
## -0.0546145076  0.0677957807  0.0572111208  0.0812754949  0.0621905120
##          V135          V136          V137          V138          V139
## -0.0145719146 -0.0992746531 -0.0065412460 -0.0626547850 -0.0313619027
##          V140          V141          V142          V143          V144
## -0.0579786196 -0.1503153513  0.0261795662  0.0207812621 -0.0596437848
##          V145          V146          V147          V148          V149
##  0.0509270049 -0.0223353957 -0.0387784481 -0.0008032294 -0.0852770235
##          V150          V151          V152          V153          V154
##  0.0059139901  0.0368928205  0.0703360619 -0.0562253698  0.0943833003
##          V155          V156          V157          V158          V159
##  0.0126104992 -0.0065115119  0.0903585203 -0.0694895137 -0.0624222289
##          V160          V161          V162          V163          V164
##  0.0339167121  0.0040008535  0.0191290521  0.0401141365  0.0098744206
##          V165          V166          V167          V168          V169
## -0.0122496454 -0.0780572245  0.0795333158 -0.0849409553  0.0965179820
##          V170          V171          V172          V173          V174
## -0.1186524116  0.0963651752  0.0671716679 -0.0002715185 -0.1003465646
##          V175          V176          V177          V178          V179
## -0.0858375370  0.0471111270 -0.0263189932 -0.0545044716 -0.0718751110
##          V180          V181          V182          V183          V184
## -0.0212774216  0.0504966349 -0.0316216053 -0.0678589608  0.0448615051
##          V185          V186          V187          V188          V189
##  0.0016154385  0.0230521796  0.0969791513  0.0259028305  0.0486893148
##          V190          V191          V192          V193          V194
## -0.1842540963  0.0500668745 -0.0088174488  0.0306105570  0.0364706419
##          V195          V196          V197          V198          V199
##  0.0646363180  0.0603672076 -0.0539495215  0.0608025137  0.0101348110
##          V200
## -0.0748688786
```

Plot the coefficient path

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



Use validation set approach to estimate test error

```
set.seed(123)
train <- sample(c(TRUE, FALSE), n, replace=T)
test <- !train
```

```
ridge.mod <- glmnet(x[train,], y[train], alpha=0, lambda=grid)
ridge.pred <- predict(ridge.mod, newx = x[test, ])
```

```
dim(ridge.pred)
```

```
## [1]  43 100
```

```
ridge.pred <- predict(ridge.mod, s = 2, newx=x[test, ])
```

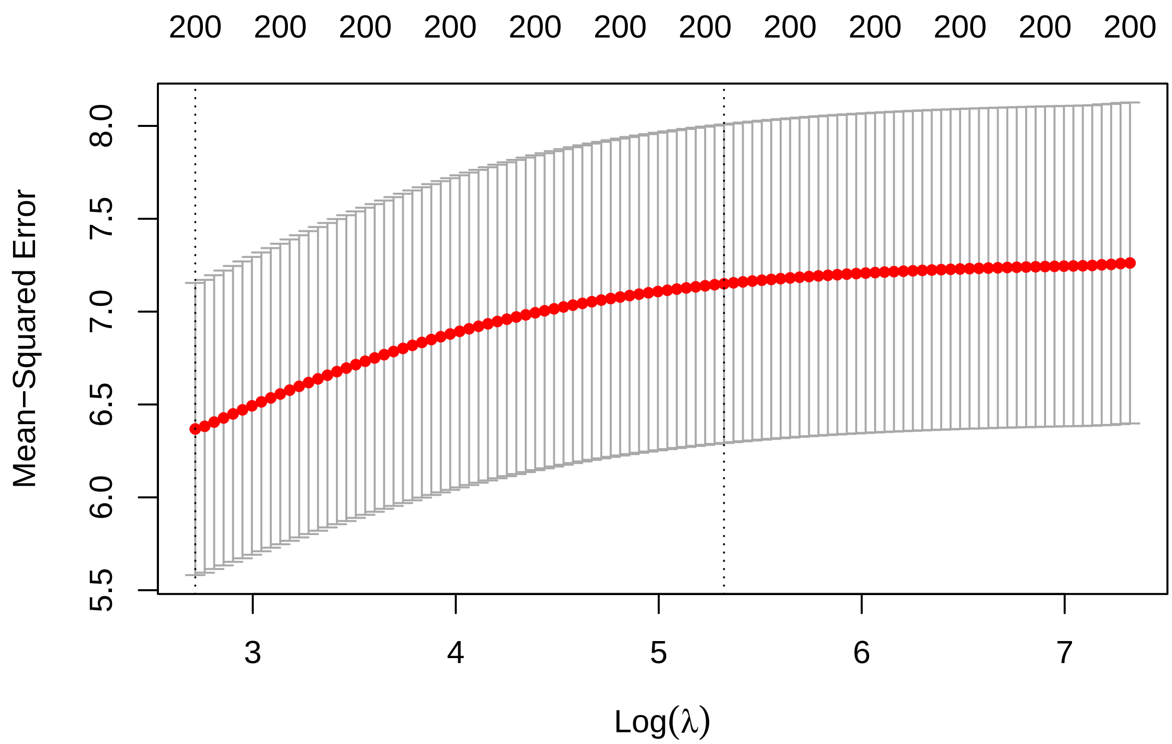
```
mean((ridge.pred - y[test])^2)
```

```
## [1] 6.591118
```

Cross validation to estimate test error

- Use a built-in cross validation function, `cv.glmnet`.
- By default, this function performs ten-fold cross validation, the number of folds K can be changed with arguments `nfolds`.

```r
set.seed(1)
cv.out <- cv.glmnet(x, y, alpha=0)
plot(cv.out)
```




```r
names(cv.out)
```

```
##  [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"
##  [6] "nzero"      "call"       "name"       "glmnet.fit" "lambda.min"
## [11] "lambda.1se" "index"
```

```r
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 15.1302
```

```r
min(cv.out$cvm) # estimated test error
```

```
## [1] 6.367945
```

```r
coef(cv.out, s=bestlam)
```

```
## 201 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept)  1.938941e-01
## V1           1.445422e-01
## V2           1.238166e-01
## V3           1.341960e-01
```

```
## V4           1.444060e-01
## V5           1.474910e-01
## V6          -4.481520e-02
## V7          -5.823097e-03
## V8           2.042892e-02
## V9           3.995440e-02
## V10         -5.760412e-03
## V11         -4.209260e-02
## V12         -1.171342e-02
## V13          7.726626e-03
## V14         -3.082724e-02
## V15         -7.855133e-03
## V16         -5.747366e-03
## V17          5.959314e-02
## V18         -3.146844e-02
## V19          9.755158e-03
## V20         -1.950678e-02
## V21          2.985730e-02
## V22          2.236713e-02
## V23         -3.239825e-02
## V24         -5.324177e-03
## V25          1.719869e-02
## V26         -2.853374e-02
## V27         -3.812120e-02
## V28         -4.932936e-02
## V29          2.791578e-03
## V30         -2.068981e-02
## V31          2.273115e-02
## V32         -4.132545e-02
## V33         -1.310593e-02
## V34          5.083164e-03
## V35          4.493674e-03
## V36         -1.203988e-02
## V37          3.852113e-02
## V38         -4.948906e-02
## V39         -1.825195e-02
## V40          2.966613e-02
## V41         -1.501537e-02
## V42         -3.697094e-02
## V43         -4.441271e-02
## V44         -5.359235e-03
## V45         -7.018050e-03
## V46          1.151597e-03
## V47         -2.651797e-02
## V48         -2.728052e-02
## V49         -3.339093e-02
## V50          2.235804e-02
## V51          2.418128e-02
## V52         -6.276546e-03
## V53          5.191952e-02
## V54          2.566216e-02
## V55          9.398619e-03
## V56          2.231501e-03
## V57         -1.089376e-03
```

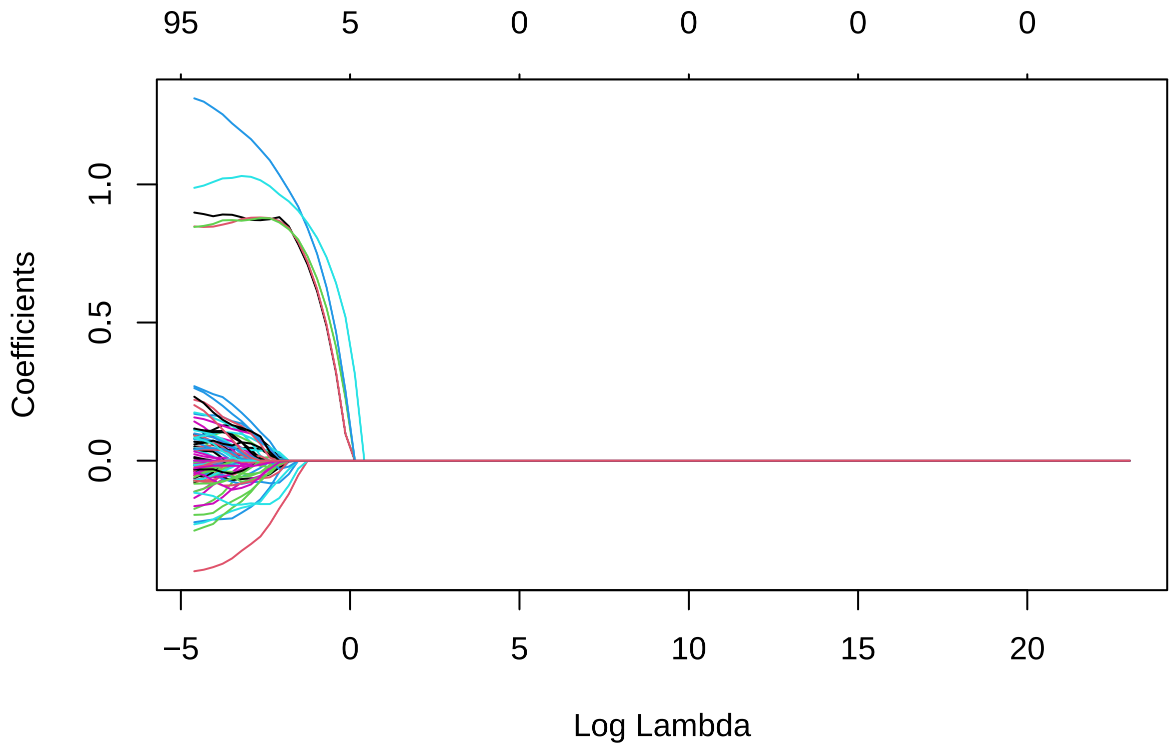
```
## V58          -1.384397e-02
## V59           2.062723e-03
## V60           4.271253e-03
## V61           5.355877e-03
## V62           1.367252e-05
## V63          -6.919221e-03
## V64           1.936364e-02
## V65           1.839920e-02
## V66           1.437023e-02
## V67           3.649225e-03
## V68          -6.834954e-03
## V69           5.568048e-03
## V70           1.457855e-02
## V71           1.519587e-03
## V72          -4.276031e-02
## V73           1.688619e-02
## V74           2.231710e-02
## V75           1.469491e-03
## V76          -4.296076e-02
## V77          -2.468672e-02
## V78          -1.725478e-02
## V79           2.199073e-02
## V80           3.374909e-02
## V81           3.277895e-02
## V82           1.086009e-02
## V83          -6.881853e-02
## V84           2.468569e-02
## V85           6.229071e-02
## V86          -1.466828e-02
## V87           2.387452e-02
## V88          -2.402534e-03
## V89          -3.673187e-02
## V90           1.600105e-02
## V91          -7.712712e-02
## V92          -4.153304e-02
## V93          -2.747004e-02
## V94           4.146573e-02
## V95           2.323669e-02
## V96          -7.913951e-02
## V97           2.175909e-02
## V98          -2.965644e-02
## V99           1.399550e-02
## V100         -1.304572e-02
## V101          1.619790e-02
## V102         -1.098055e-02
## V103          2.505644e-02
## V104          2.198065e-02
## V105          2.394907e-02
## V106         -4.899009e-03
## V107         -2.180600e-02
## V108         -4.484777e-02
## V109          2.630929e-02
## V110         -7.507805e-03
## V111         -1.782875e-02
```

```
## V112         -2.069367e-02
## V113         -3.037099e-02
## V114         -1.159231e-02
## V115         -4.755376e-02
## V116          3.378892e-02
## V117          2.250415e-04
## V118         -6.315086e-03
## V119         -2.187522e-02
## V120          3.812086e-02
## V121          1.222393e-02
## V122         -4.901935e-03
## V123         -6.090412e-02
## V124         -9.393275e-03
## V125          7.935149e-03
## V126         -3.723767e-02
## V127         -4.311174e-02
## V128          4.861385e-02
## V129          2.929446e-02
## V130         -1.396473e-02
## V131          2.431011e-02
## V132          1.365379e-02
## V133          3.632940e-02
## V134          3.516037e-02
## V135         -1.000915e-02
## V136         -2.893549e-02
## V137         -8.007429e-03
## V138         -3.233827e-02
## V139         -1.206922e-02
## V140         -3.616630e-02
## V141         -6.154890e-02
## V142          2.152247e-02
## V143          1.519355e-02
## V144         -1.131655e-02
## V145          2.307591e-02
## V146         -2.736344e-03
## V147         -2.679401e-02
## V148         -3.859175e-03
## V149         -3.271984e-02
## V150         -2.375923e-03
## V151          1.559708e-02
## V152          2.397135e-02
## V153         -3.454143e-02
## V154          3.765414e-02
## V155          7.018440e-03
## V156          4.496110e-03
## V157          2.505495e-02
## V158         -1.616339e-02
## V159         -2.073072e-02
## V160          3.730387e-03
## V161         -9.578896e-04
## V162          1.003381e-02
## V163          1.688591e-02
## V164         -5.538428e-03
## V165         -4.007004e-03
```

```
## V166        -4.202900e-02
## V167         3.054751e-02
## V168        -2.481311e-02
## V169         4.270586e-02
## V170        -6.496807e-02
## V171         2.523079e-02
## V172         3.611742e-02
## V173         7.918069e-04
## V174        -3.838620e-02
## V175        -3.857102e-02
## V176         3.169173e-02
## V177        -3.910595e-03
## V178        -2.633058e-02
## V179        -2.027471e-02
## V180        -1.271688e-02
## V181         1.878250e-02
## V182        -5.673889e-03
## V183        -1.950300e-02
## V184         9.592730e-03
## V185         4.662862e-03
## V186         1.047442e-02
## V187         3.881097e-02
## V188         1.546826e-02
## V189         1.083954e-02
## V190        -7.660967e-02
## V191         1.362593e-02
## V192        -2.094184e-03
## V193         2.587557e-03
## V194         1.115975e-03
## V195         3.285788e-02
## V196         2.259811e-02
## V197        -2.506678e-02
## V198         2.251671e-02
## V199         1.692800e-02
## V200        -3.094360e-02
```

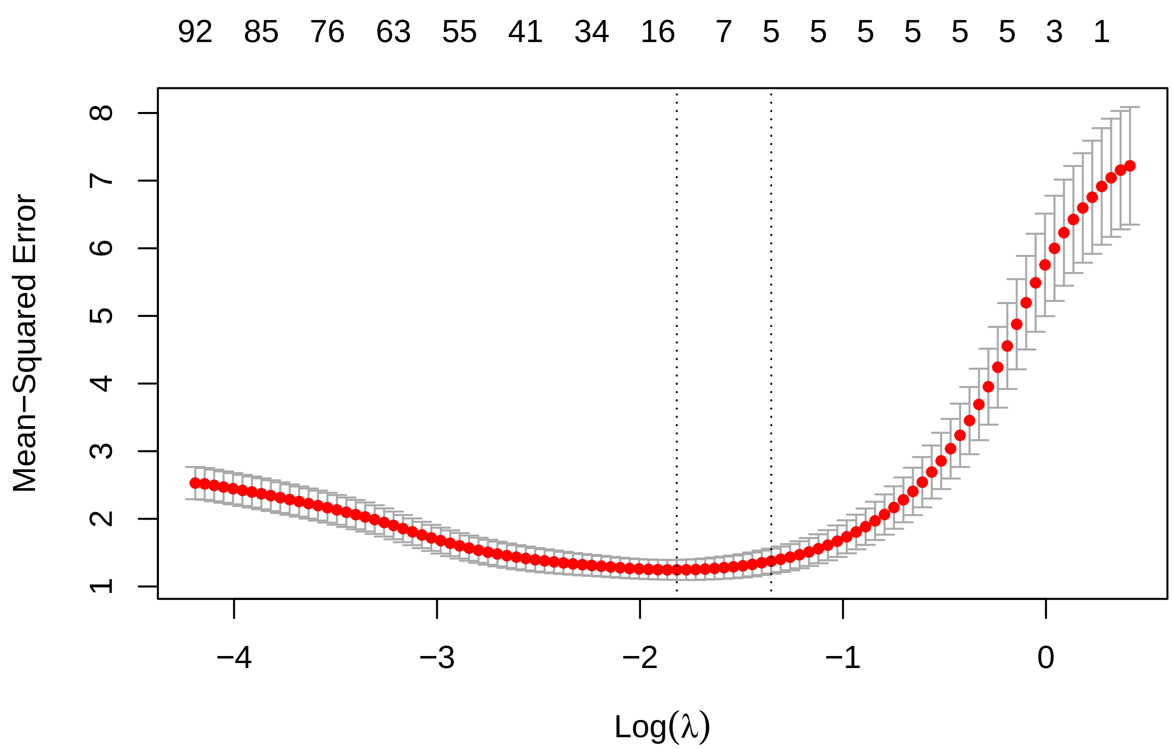
```
lasso.mod <- glmnet(x, y, alpha=1, lambda=grid)
```

```
plot(lasso.mod, xvar="lambda")
```




```
set.seed(1)
cv.out <- cv.glmnet(x, y, alpha=1)
```

```
plot(cv.out)
```



```r
bestlam <- cv.out$lambda.min
```

```r
min(cv.out$cvm)
```

```
## [1] 1.24534
```

```r
coef(cv.out, s=bestlam)
```

```
## 201 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.055152251
## V1           0.849236941
## V2           0.843118969
## V3           0.838201199
## V4           0.979870347
## V5           0.939029111
## V6           .
## V7           .
## V8           .
## V9           .
## V10          .
## V11         -0.003709171
## V12          .
## V13          .
## V14          .
## V15          .
## V16          .
## V17          .
## V18          .
## V19          .
## V20          .
## V21          .
## V22          .
## V23          .
## V24          .
## V25          .
## V26          .
## V27          .
## V28          .
## V29          .
## V30          .
## V31          .
## V32          .
## V33          .
## V34          .
## V35          .
## V36         -0.002990657
## V37          .
## V38          .
## V39          .
## V40          .
## V41          .
```

```
## V42          .
## V43          .
## V44          .
## V45          .
## V46          .
## V47          .
## V48         -0.001422773
## V49          .
## V50          .
## V51          .
## V52          .
## V53          .
## V54          .
## V55          .
## V56         -0.121982839
## V57          .
## V58          .
## V59          .
## V60          .
## V61          .
## V62          .
## V63          .
## V64          .
## V65          .
## V66          .
## V67          .
## V68          .
## V69          .
## V70          .
## V71          .
## V72          .
## V73          .
## V74          .
## V75          .
## V76          .
## V77          .
## V78          .
## V79          .
## V80          .
## V81          .
## V82          .
## V83          .
## V84          .
## V85          .
## V86          .
## V87          .
## V88          .
## V89          .
## V90          .
## V91          .
## V92         -0.049717212
## V93          .
## V94          0.000043404
## V95          .
```

```
## V96          .
## V97          .
## V98          .
## V99          .
## V100         .
## V101         .
## V102         .
## V103         .
## V104         .
## V105         .
## V106         .
## V107         .
## V108         .
## V109         .
## V110         .
## V111         .
## V112         .
## V113         .
## V114         .
## V115        -0.031923620
## V116         .
## V117         .
## V118         .
## V119         .
## V120         .
## V121         .
## V122         .
## V123         .
## V124         .
## V125         .
## V126         .
## V127         .
## V128         .
## V129         .
## V130         .
## V131         .
## V132         .
## V133         .
## V134         .
## V135         .
## V136         .
## V137         .
## V138         .
## V139         .
## V140        -0.022016187
## V141         .
## V142         .
## V143         .
## V144         .
## V145         .
## V146         .
## V147         .
## V148         .
## V149         .
```

```
## V150          .
## V151          .
## V152          .
## V153          .
## V154          .
## V155          .
## V156          .
## V157          .
## V158          .
## V159          .
## V160          .
## V161          .
## V162          .
## V163          .
## V164          .
## V165          .
## V166          .
## V167          .
## V168          .
## V169          .
## V170          .
## V171          .
## V172          .
## V173          .
## V174         -0.090645746
## V175          .
## V176          .
## V177          .
## V178          .
## V179          .
## V180          .
## V181          .
## V182          .
## V183          .
## V184          .
## V185          .
## V186          .
## V187          .
## V188          .
## V189          .
## V190          .
## V191          .
## V192          .
## V193          .
## V194          .
## V195          .
## V196          .
## V197          .
## V198          .
## V199          .
## V200          .
```

Collinearity

```
library(MASS)
```

```
Sigma = matrix(c(1, 0.99, 0.99, 1), ncol=2)
```

```
Sigma
```

```
##      [,1] [,2]
## [1,] 1.00 0.99
## [2,] 0.99 1.00
```

```
set.seed(1)
n = 500
x1 = rnorm(n)
x2 = x1
```

```
y = 0.5 + x1 + rnorm(n)
```

```
lm_fit <- lm(y ~ x1 + x2)
```

```
summary(lm_fit)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.94113 -0.74507  0.01663  0.72882  3.11131
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.45504    0.04730    9.62   <2e-16 ***
## x1           0.95692    0.04678   20.46   <2e-16 ***
## x2                NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.057 on 498 degrees of freedom
## Multiple R-squared:  0.4566, Adjusted R-squared:  0.4555
## F-statistic: 418.4 on 1 and 498 DF,  p-value: < 2.2e-16
```

```
ridge_fit <- cv.glmnet(x = cbind(x1, x2), y, alpha=0)
```

```
bestlam = ridge_fit$lambda.min
coef(ridge_fit, s=bestlam)
```

```
## 3 x 1 sparse Matrix of class "dgCMatrix"
##                    s1
## (Intercept) 0.4557394
## x1          0.4655129
## x2          0.4603055
```

```r
lasso_fit <- cv.glmnet(x = cbind(x1, x2), y, alpha=1)
```

```r
bestlam = lasso_fit$lambda.min
coef(lasso_fit, s=bestlam)
```

```
## 3 x 1 sparse Matrix of class "dgCMatrix"
##                    s1
## (Intercept) 0.4551777
## x1          0.9506259
## x2          .
```