Variable Selection in Linear Regression

Lucas Ben

In this project I will use several methods for model and variable selection in a regression context. I'll be using a simulated data set.

Data Simulation

Here I'm creating an n x p matrix, X, filled with NA where n = 25 and p = 10. I'll be filling the first nine columns with randomly generated values from a standard normal distribution.

As well, I'm creating an additional input variable that is highly correlated collinear with the other nine. I'll do this by filling the tenth column of X with a variable defined as

$$x_{i,10} = \frac{1}{9}(x_{i1} + x_{i2} + \cdots + x_{i9}) + \eta_i$$

where for each $i \in \{1,2,\ldots,n\}$, η_i is randomly generated from a normal distribution with mean 0 and standard deviation 0.005. The small standard deviation will ensure that $x_{i,10}$ is almost a linear combination of the other nine variables.

```
set.seed(2002)
# dimensions of data
n = 25
p = 10
X = matrix(NA, n, p) # creating an input matrix
X[,-p] = rnorm(n * (p - 1)) # generating standard normal random numbers
X[,p] = rowMeans(X[,1:(p-1)]) + rnorm(n, sd = 0.005)
X = scale(X) # scaling columns of X
```

Now I'll create a vector of length n for the output, y, defined as

$$y_i = (\beta_0, \ldots, \beta_{10}) = (1, 0, 0, 0.002, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 0.1)$$

and where, for each $i \in \{1,\ldots,n\}$ ϵ_i is randomly generated from a normal distribution with mean 0 and standard deviation 0.1.

```
beta = c(1, 0, 0, 0.002, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 0.1) # creating output variable
sig = 0.1
y = beta[1] + X %*% beta[-1] + rnorm(n, sd = sig)
data = as.data.frame(cbind(y, X)) # creating a data frame that has one column for the ou
tput variable and one column for each input variable
colnames(data) = c("y", paste0("x", 1:p))
head(data)
```

```
##            y         x1         x2         x3         x4         x5          x6
## 1  1.8737680 -0.2690417 -0.2013772  0.1203584 -0.3347028 -0.7725850  1.46712866
## 2  3.2453340  1.2787647  0.8235780  1.3942941  1.8723854  1.2947146  0.08331901
## 3  1.4955132  1.0753487  0.4667716  0.3829317 -0.4126904 -0.3744391 -1.42022052
## 4  2.8268730  0.4588606 -0.7039841 -0.5716212 -0.1751230 -1.8278061 -0.38265792
## 5  1.4369886  1.8570936  0.1488059  0.4610649 -2.4669313  0.3311850  0.17626444
## 6 -0.2743436 -0.3499006 -0.6973556  0.7116098  0.2819463  0.3798765 -0.36988887
##            x7         x8          x9         x10
## 1 -1.95123608  0.1302195  1.02588660 -0.37415982
## 2 -0.05560059  0.7045718  1.62053193  2.96550703
## 3  1.49663228  0.9456787 -0.19829663  0.74655753
## 4  0.42599273  1.9142277  0.85520526 -0.01584334
## 5  1.10197969  0.5744022  0.08428538  0.72582707
## 6 -0.61314179 -1.9676295 -0.18774987 -0.95752594
```

Multiple Linear Regression

Here I'll fit a multiple regression model to the data with *y* as the output and all other variables as inputs. After fitting the model, I'll use **regsubsets** from the package **leaps** for forward stepwise selection.

The best model according to the Bayesian information criterion includes 5 variables: $x_1, x_7, x_8, x_9, x_{10}$.

```
summary(lm(y ~ ., data))
```

```
##
## Call:
## lm(formula = y ~ ., data = data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.110982 -0.036341  0.004795  0.049741  0.128110
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.98969    0.01599  61.881   <2e-16 ***
## x1          -0.53344    0.50424  -1.058   0.3080
## x2          -0.43081    0.44164  -0.975   0.3459
## x3          -0.40744    0.40947  -0.995   0.3366
## x4          -0.45226    0.46507  -0.972   0.3473
## x5          -0.52865    0.51990  -1.017   0.3265
## x6          -0.32993    0.35377  -0.933   0.3668
## x7          -0.24332    0.37023  -0.657   0.5217
## x8          -0.02546    0.52148  -0.049   0.9618
## x9           0.68515    0.33153   2.067   0.0578 .
## x10          1.46270    1.35352   1.081   0.2981
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07997 on 14 degrees of freedom
## Multiple R-squared:  0.9972, Adjusted R-squared:  0.9952
## F-statistic: 503.6 on 10 and 14 DF,  p-value: 4.112e-16
```

```
library(leaps)
mods_all = regsubsets(y ~ .,
                      data = data,
                      nvmax = p,
                      method = "forward") # stepwise selection
n_var = which.min(summary(mods_all)$bic) # identifying the optimal number of variables u
sing the Bayesian information criterion
cat("The best model, according to BIC, includes", n_var, "variables.\n")
```

```
## The best model, according to BIC, includes 5 variables.
```

```
summary(mods_all)$which[n_var,] # finding variables that are included in the best model
```

```
## (Intercept)          x1          x2          x3          x4          x5
##        TRUE        TRUE       FALSE       FALSE       FALSE       FALSE
##          x6          x7          x8          x9         x10
##       FALSE        TRUE        TRUE        TRUE        TRUE
```

Ridge Regression

Here I'll fit a ridge regression model to the data using **cv.glmnet** from the package **glmnet** over the following range of values for the tuning parameter:

$\lambda \in 2^{-15}, 2^{-14}, \dots, 2^{1}, 2^{2}$

From cross-validation, the best value of is 0.0004882812.

```
library(glmnet)
```

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

```
## Loaded glmnet 4.1-8
```

```
lambdas = 2^seq(-15, 2, by = 1) # grid of lambda values
mat = as.matrix(data)
ridge = cv.glmnet(x = mat[,-1],
                  y = mat[,1],
                  alpha = 0,
                  lambda = lambdas,
                  nfolds = 5) # ridge regression
ridge$lambda.min # best lambda
```

```
## [1] 0.0004882812
```

```
coef(ridge, s = "lambda.min") # parameters for best lambda
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  0.98969499
## x1          -0.09729497
## x2          -0.04910216
## x3          -0.05332658
## x4          -0.05017719
## x5          -0.07929087
## x6          -0.02405080
## x7           0.07656281
## x8           0.42504077
## x9           0.97095291
## x10          0.29185683
```

Lasso

Here I'll be using **cv.glmnet** to fit a lasso model for the same range of values of from earlier with 5-fold cross-validation.

```
lasso = cv.glmnet(x = mat[, -1],
                  y = mat[, 1],
                  alpha = 1,
                  lambda = lambdas,
                  nfolds = 5) # lasso
```

The lasso excluded variables x_3, x_4, x_5 which have parameters equal to exactly zero in the output. As well, I checked whether ridge regression or the lasso performed better on this simulated data set. The cross-validation error for the ridge regression was slightly smaller than for the lasso so the ridge regression performed better.

```
coef(lasso, s = "lambda.min") # parameters for best lambda
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.989694991
## x1          -0.006383106
## x2           .
## x3           .
## x4           .
## x5           .
## x6           0.016447951
## x7           0.101888659
## x8           0.476048440
## x9           0.995655507
## x10          0.111122028
```

```
min(ridge$cvm) # cross-validation error for ridge regression
```

```
## [1] 0.009238531
```

```
min(lasso$cvm) # cross-validation error for lasso
```

```
## [1] 0.01313532
```