HW3: Model Selection (due 11/22 9:00)

110065508, Cheng-En Lee

Data Source

```
library(AppliedPredictiveModeling) #install package first!!
library(corrplot)  #correlation
library(leaps)
library(glmnet)
library(pls)
library(boot)
```

Problem 1: Chemical Manufacturing Process Data

1. Data Preprocessing

(1) Data discription This data set consists of 177 samples of biological material for which 58 characteristics were measured. Of the 58 characteristics, there were 12 measurements of the biological starting material; 45 measurements of the manufacturing process, and one variable of the resulting final product yield (target variable).

The objective of this problem is to predict the yield given the other variable information.

More data descriptions can be found by typing `help(ChemicalManufacturingProcess)` in r command.

```
data(ChemicalManufacturingProcess)
#View(ChemicalManufacturingProcess)  #you may view data in table
#help(ChemicalManufacturingProcess)
#view(dfSummary(ChemicalManufacturingProcess))  #view data summary in html
```

(2) Remove NA's

- There are several NA's in the data. We will only use the data with the complete variable information in this homework analysis.
- Rename the dataframe consisting of the complete variable information as **CMP**.
- Rename the variable for a shorten expression

```
dim(ChemicalManufacturingProcess)
```

```
## [1] 176  58
```

```
sum(is.na(ChemicalManufacturingProcess))
```

```
## [1] 106
```

```
which(apply(is.na(ChemicalManufacturingProcess),1,sum)>0) #check which data sample has missing
```

```
##   1   2   3   4   5   6  15  16  17  18  19  20  22  23  24  90  98 134 139 172
##   1   2   3   4   5   6  15  16  17  18  19  20  22  23  24  90  98 134 139 172
## 173 174 175 176
## 173 174 175 176
```

```
CMP <- na.omit(ChemicalManufacturingProcess) #remove missing data
dim(CMP)
```

```
## [1] 152  58
```

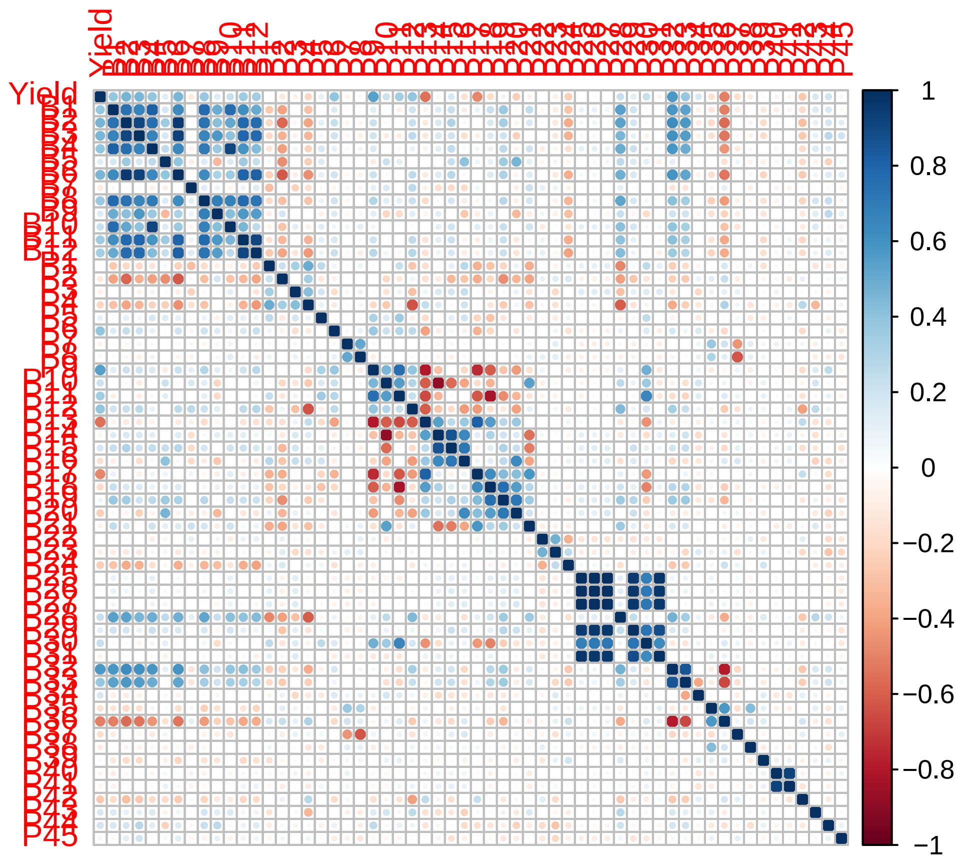
```
sum(is.na(CMP))
```

```
## [1] 0
```

```
#rename variable
B_name = c()
for (i in 1:12){
  B_name[i] = paste("B",i,sep="")
}
P_name = c()
for (i in 1:45){
  P_name[i] = paste("P",i,sep="")
}
names(CMP) <- c("Yield",B_name, P_name)
```

(3) Correlation plot Note: `pls` package also has a function named `corrplot`, which is not we want.

```r
corrplot::corrplot(cor(CMP))
```



(4) Note

- Fit regression models with various variable selection procedures.
- Go through all variable selection procedures taught in the class, to determine your best model (possibly more than one).
- Identify the important input variables (or any new features you defined) and carefully describe their impacts on the response yield.
- Evaluate your prediction performance via 5-fold cross validation.
- Give a brief analysis summary.

In this homework, I use MSE as the performance matric.

(5) Bad news: Best Subset does not work Best subset selection is not feasible for R in this case due to the size of this dataset with the following error:

```
Error in leaps.exhaustive(a, really.big) :  Exhaustive search will be S L O W, must
specify really.big=T
```

Let‘s turn to the other algorithms.

(6) Train test split The splitting proportion is set to 0.7.

```r
set.seed(48763)
train <- sample(c(TRUE, FALSE), nrow(CMP), replace=TRUE, prob=c(0.7,0.3))
test <- !train
```

2. Forward Stepwise Selection

(1) The predict function In order to conduct CV, we need a predict function since the class `regsubsets` does not have an in-build one.

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

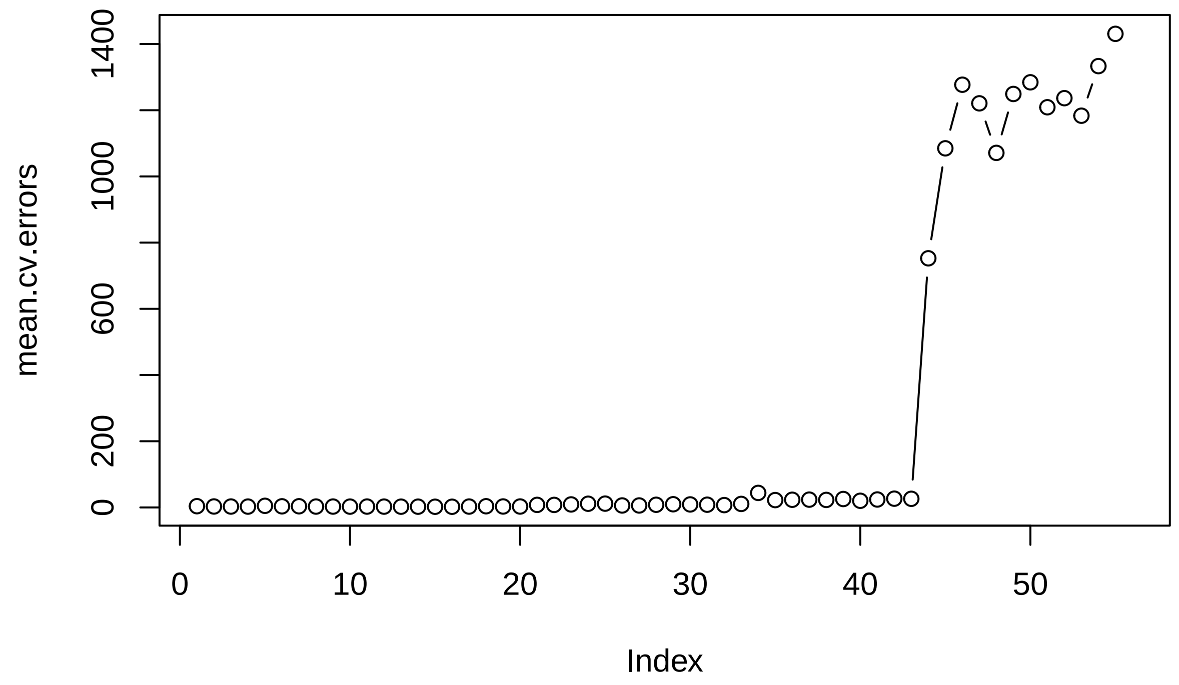
```r
# create a matrix in which we will store the results
k <- 5 # 5-fold CVs
n <- nrow(CMP[train, ])
set.seed(1)
folds <- sample(rep(1:k, length = n))
cv.errors <- matrix(NA, k, 56, dimnames = list(NULL, paste(1:56)))

# write a for loop that performs cross-validation
for (j in 1:k) {
  best.fit.fwd <- regsubsets(Yield ~ .,
                             data = CMP[train, ][folds != j, ],
                             nvmax = 56,
                             method = "forward")
  for (i in 1:55) {
    pred <- predict(best.fit.fwd, CMP[train, ][folds == j, ], id = i)
    cv.errors[j, i] <- mean((CMP[train, ]$Yield[folds == j] - pred)^2)
   }
}
```

(2) 5-fold CV

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```r
# Plot the MSE vs number of features
mean.cv.errors <- apply(cv.errors, 2, mean)
par(mfrow = c(1, 1))
plot(mean.cv.errors, type = "b")
```



Let's find out the best model contains how many features

```
which.min(mean.cv.errors) # How many features does the best model have
```

```
## 16
## 16
```

Since 16 is large enough so that the formula of the model is not suitable to type in the report, I present the coefficients of the model by the following code.

```
reg.best.fwd <- regsubsets(Yield ~ .,
                           data = CMP[train, ],
                           nvmax = 56,
                           method = "forward")
```

```
## Reordering variables and trying again:
```

```
coef(reg.best.fwd, 16) # The coefficient
```

```
##   (Intercept)            B5            B7            P2            P3
## 203.774714834   0.088596198  -1.803641141   0.013245829  -6.154026209
##            P7            P9           P17           P19           P20
##  -0.301997547   0.387444045  -0.405046784   0.012038755  -0.009036993
##           P23           P25           P29           P33           P34
##   0.005510939  -0.002618808   0.646552862   0.292800293   9.455002503
##           P38           P45
##  -0.196363005   0.531180811
```

(3) Model Performance The training performance

```
mean.cv.errors[16] # training performance
```

```
##       16
## 2.085848
```

Evaluate the model performance on the testing set

```
test_pred <- predict(reg.best.fwd, CMP[test, ], id = 16)
MSE <- mean((CMP[test, ]$Yield - test_pred)^2)
MSE
```

```
## [1] 1.815417
```

3. Backward Stepwise Selection

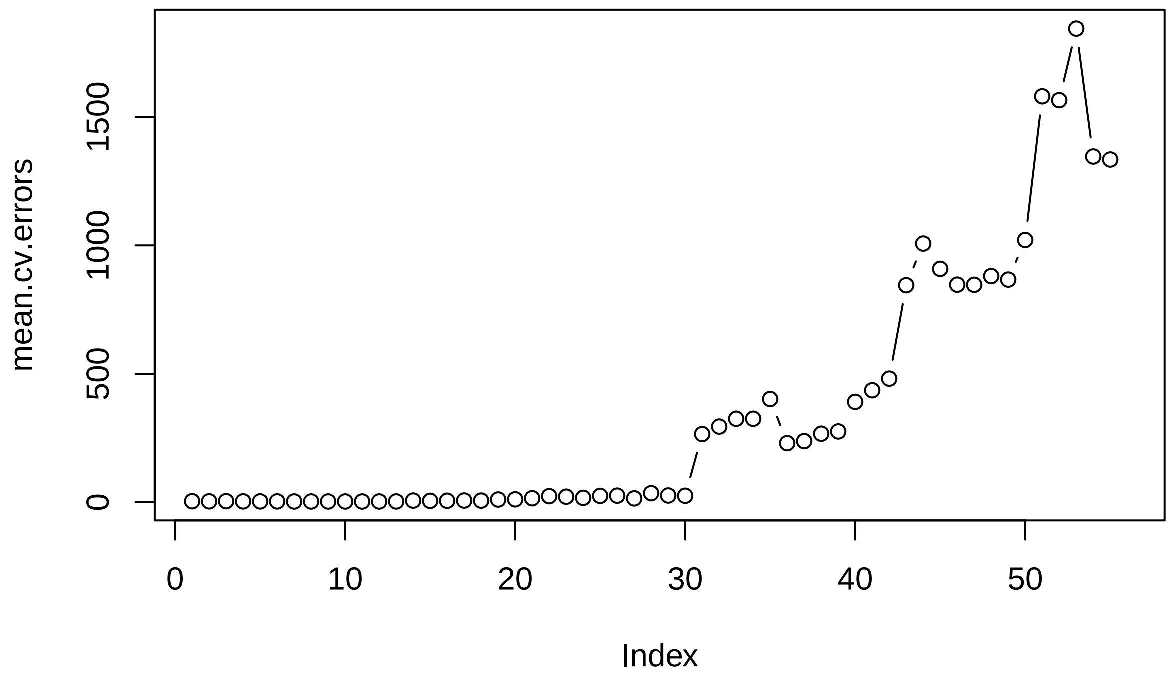
All the process in this section is parallel to 3.

```
# create a matrix in which we will store the results
k <- 5 # 5-fold CVs
n <- nrow(CMP[train, ])
set.seed(1)
folds <- sample(rep(1:k, length = n))
cv.errors <- matrix(NA, k, 56, dimnames = list(NULL, paste(1:56)))

# write a for loop that performs cross-validation
for (j in 1:k) {
  best.fit.bwd <- regsubsets(Yield ~ .,
                             data = CMP[train, ][folds != j, ],
                             nvmax = 56,
                             method = "backward")
  for (i in 1:55) {
    pred <- predict(best.fit.bwd, CMP[train, ][folds == j, ], id = i)
    cv.errors[j, i] <- mean((CMP[train, ]$Yield[folds == j] - pred)^2)
   }
}
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```
# Plot the MSE vs number of features
mean.cv.errors <- apply(cv.errors, 2, mean)
par(mfrow = c(1, 1))
plot(mean.cv.errors, type = "b")
```



```r
which.min(mean.cv.errors) # How many features does the best model have
```

```
## 11
## 11
```

```r
reg.best.bwd <- regsubsets(Yield ~ .,
                           data = CMP[train, ],
                           nvmax = 56,
                           method = "forward")
```

```
## Reordering variables and trying again:
```

```r
coef(reg.best.bwd, 11) # The coefficient
```

```
##  (Intercept)            B5            P2            P7            P9           P17
##  6.965015596  0.126224215 -0.010264969 -0.357677549  0.418829668 -0.487763889
##          P19           P20           P29           P33           P34           P38
##  0.013464565 -0.008011637 -0.013027450  0.331570014 10.950442953 -0.134254555
```

(3) Model Performance The training performance

```
mean.cv.errors[11] # training performance
```

```
##       11
## 2.785927
```

Evaluate the model performance on the testing set

```
test_pred <- predict(reg.best.bwd, CMP[test, ], id = 11)
MSE <- mean((CMP[test, ]$Yield - test_pred)^2)
MSE
```

```
## [1] 1.826801
```

Note: It looks like there are some bugs in `regsubsets` since I cannot choose every features into the model. To be precisely, the argument `nvmax` can only set to 56. However, this does not affect the results since it is really unlikely that the best model contains "every" features.

4. Ridge regression

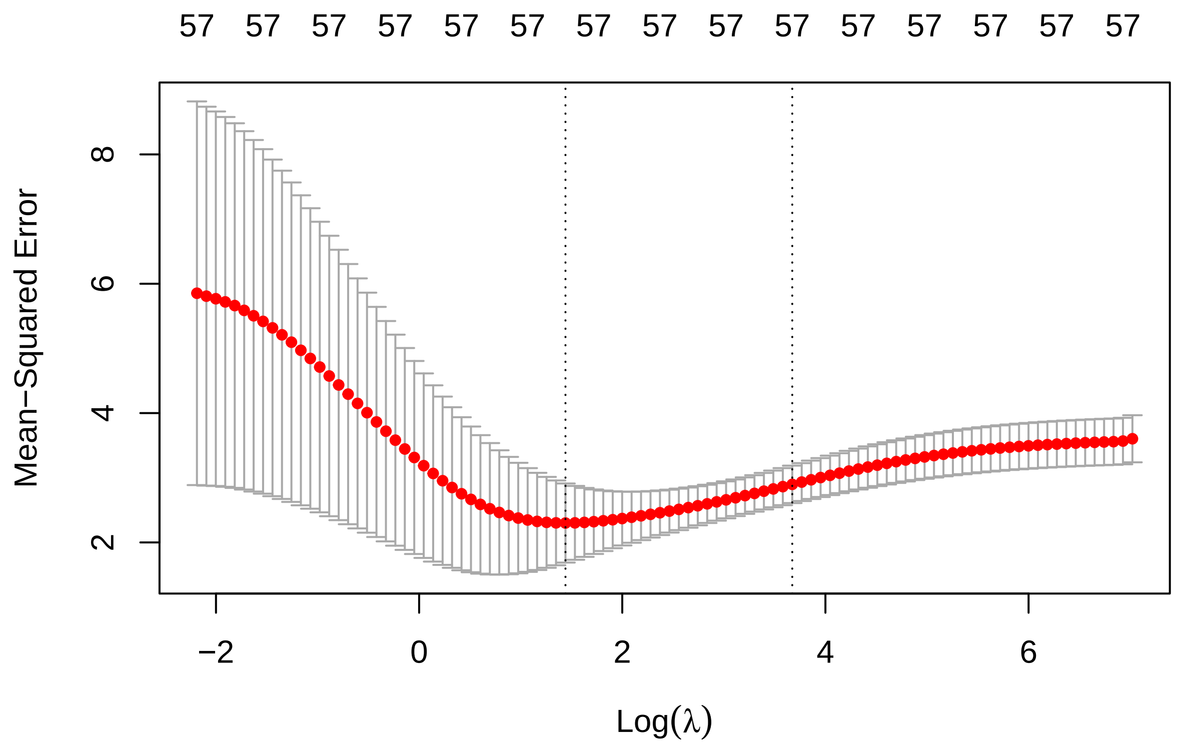
```


x.train <- model.matrix(Yield ~ ., CMP[train, ])[, -1]
y.train <- CMP[train, ]$Yield
x.test <- model.matrix(Yield ~ ., CMP[test, ])[, -1]
y.test <- CMP[test, ]$Yield
```

(1) Divide the target variable and features

(2) In-built CV In the `glmnet` function, if the argument `alpha=0` then a ridge regression model is fit, and if `alpha=1` then a lasso model is fit.

```
grid <- 10^seq(10, -2, length = 100) # use grid search to find lambda
set.seed(48763)
cv.out <- cv.glmnet(x.train, y.train, alpha = 0) # Ridge regression
plot(cv.out)
```



By the following code, we know that the best $\lambda \approx 4.635$.

```
bestlam_ridge <- cv.out$lambda.min # best lambda
bestlam_ridge
```

```
## [1] 4.223584
```

(3) Model Performance Evaluate the performance of Ridge regression.

```
# retrain the model with the best lambda
ridge.mod <- glmnet(x.train, y.train, alpha = 0, lambda = grid)

# training performance
ridge.pred <- predict(ridge.mod, s = bestlam_ridge, newx = x.train)
MSE_train <- mean((ridge.pred - y.train)^2)

# testing performance
ridge.pred <- predict(ridge.mod, s = bestlam_ridge, newx = x.test)
MSE_test <- mean((ridge.pred - y.test)^2)

# results
MSE_train
```

```
## [1] 1.370079
```

```
MSE_test
```

```
## [1] 1.557925
```

(4) Important Features Since Ridge regression does not really select variables, I show the coefficient of the first 20 features.

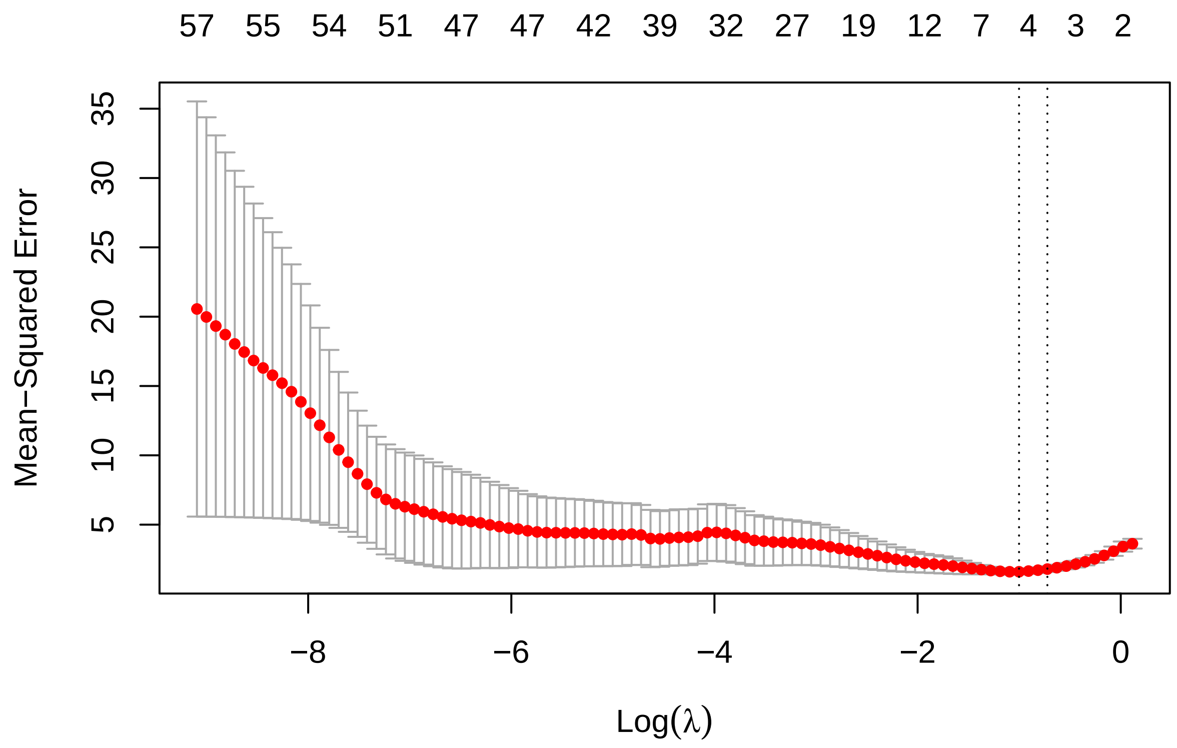
```
ridge_coeff <- predict(ridge.mod,
                       s = bestlam_ridge,
                       exact = T,
                       type = "coefficients",
                       x = x.train,
                       y = y.train)
ridge_coeff[1:20, ]
```

```
##   (Intercept)            B1            B2            B3            B4 
## 71.8476344121  0.0643033660  0.0155283234  0.0131913681  0.0254978603 
##            B5            B6            B7            B8            B9 
##  0.0119775822  0.0132245521 -0.4925201600  0.0566020940 -0.0060573688 
##           B10           B11           B12            P1            P2 
##  0.0158912180  0.0090817035  0.0368106962  0.0001256437  0.0001350158 
##            P3            P4            P5            P6            P7 
## -1.5131671132 -0.0017485545  0.0003365234  0.0595367295 -0.0841868700
```

5. Lasso

The process in this section is parallel to 5.

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



By the following code, we know that the best $\lambda \approx 0.3673$.

```
bestlam_lasso <- cv.out$lambda.min # best lambda
bestlam_lasso
```

```
## [1] 0.3673458
```

```
# retrain the model with the best lambda
lasso.mod <- glmnet(x.train, y.train, alpha = 1, lambda = grid)

# training performance
lasso.pred <- predict(lasso.mod, s = bestlam_lasso, newx = x.train)
MSE_train <- mean((lasso.pred - y.train)^2)

# testing performance
lasso.pred <- predict(lasso.mod, s = bestlam_lasso, newx = x.test)
MSE_test <- mean((lasso.pred - y.test)^2)

# results
MSE_train
```

```
## [1] 1.45914
```

```
MSE_test
```

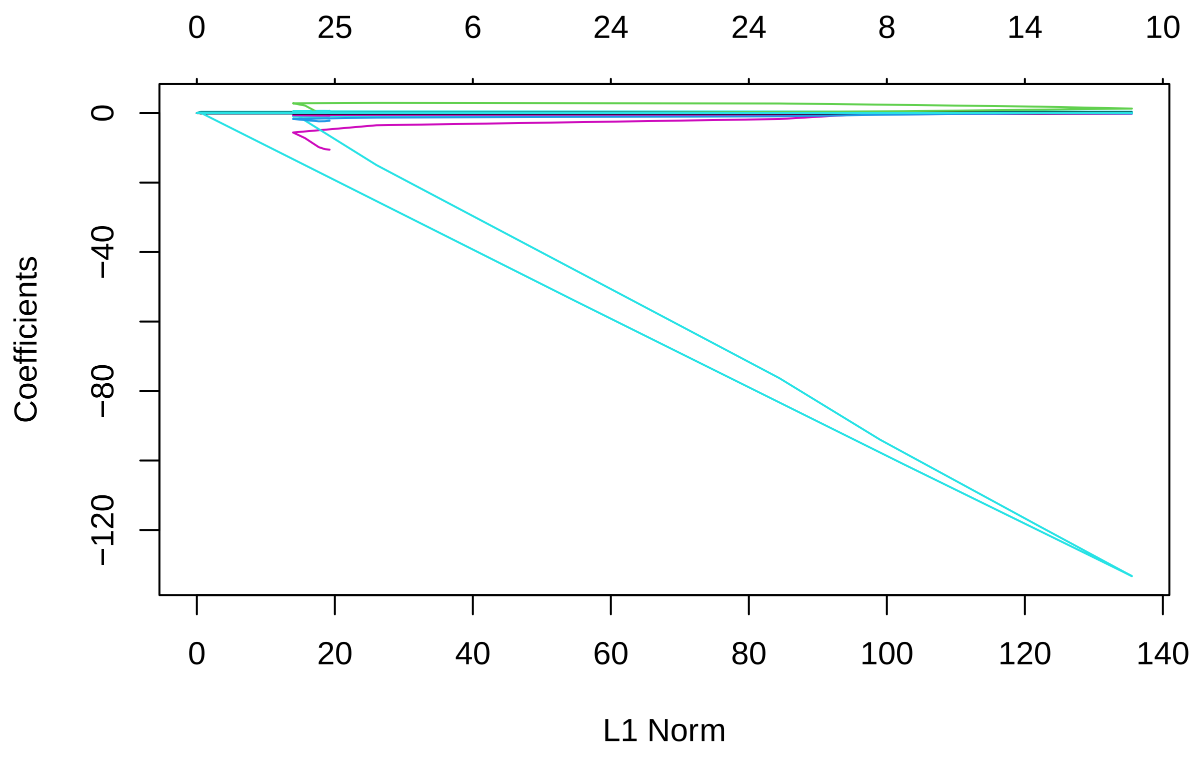
```
## [1] 1.695445
```

(4) Important Features It turns out that LASSO selects 5 features only.

```
lasso_coeff <- predict(lasso.mod,
                       s = bestlam_lasso,
                       exact = T,
                       type = "coefficients",
                       x = x.train,
                       y = y.train)
lasso_coeff[1:57,][which(lasso_coeff!=0)]
```

```
##  (Intercept)           P6           P9          P13          P17          P32
##  4.846384160  0.004737659  0.367678143 -0.097801125 -0.049232780  0.142601385
```

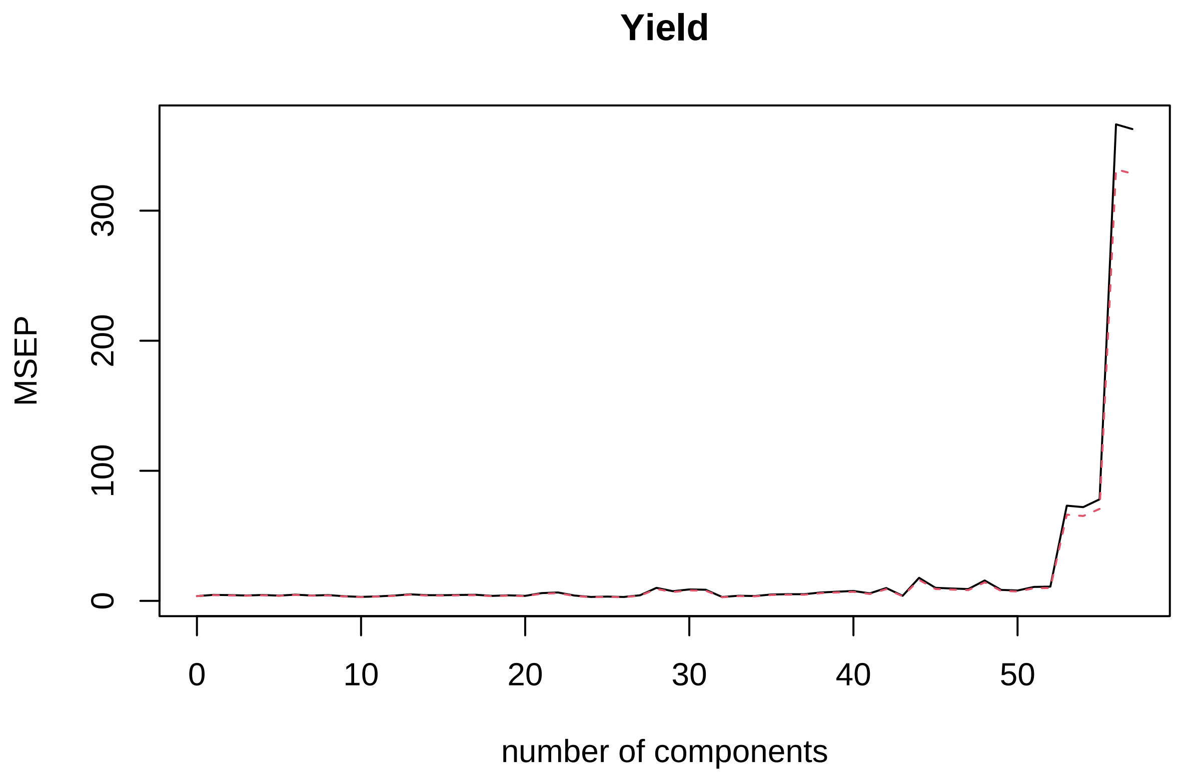
```
plot(lasso.mod)
```



6. PCA

Perform PCR on the training data.

```r
set.seed(48763)
pcr.fit <- pcr(Yield ~ .,
               data = CMP,
               subset = train,
               scale = TRUE,
               validation = "CV")
validationplot(pcr.fit, val.type = "MSEP")
```



```r
MSEP_object <- MSEP(pcr.fit)
which.min(MSEP_object$val[1,1, ])
```

```
## 32 comps
##       33
```

Hence PCR selects 33 features. Also, model performance can be evaluated

```r
# training performance
pcr.pred.train <- predict(pcr.fit, x.train, ncomp = 33)
MSE_pcr_train <- mean((pcr.pred.train - y.train)^2)

# testing performance
pcr.pred.test <- predict(pcr.fit, x.test, ncomp = 33)
MSE_pcr_test <- mean((pcr.pred.test - y.test)^2)
```

```
# results
MSE_pcr_train
```

```
## [1] 0.7671907
```

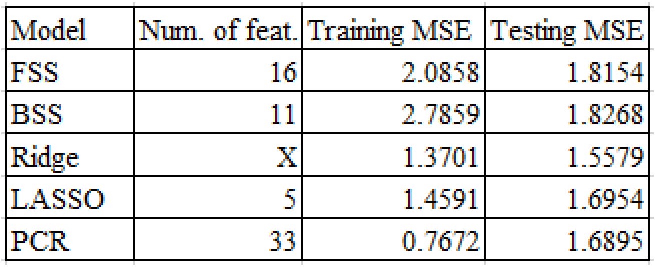
```
MSE_pcr_test
```

```
## [1] 1.689532
```

7. Summary.

(1) Model performance By the above discussion, the best model is Ridge regression, considering the testing MSE only. Both FSS and BSS are not performs as well as Ridge. However, considering the simplicity of the model, perhaps LASSO is more feasible since it only takes 5 features into consideration.

```
knitr::include_graphics("1.png")
```

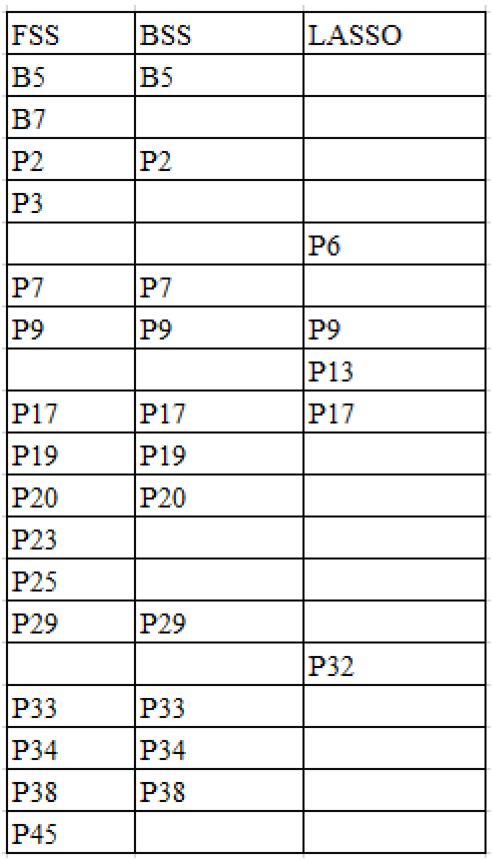


Model	Num. of feat.	Training MSE	Testing MSE
FSS	16	2.0858	1.8154
BSS	11	2.7859	1.8268
Ridge	X	1.3701	1.5579
LASSO	5	1.4591	1.6954
PCR	33	0.7672	1.6895

Though PCR performs slightly better than LASSO, the gap between training MSE and testing MSE suggests that there is an overfitting. The same problems occurs when it comes to BSS.

(2) Important variables The below table shows the important variables select by FSS, BSS, and LASSO. Only `P9` and `P17` is regarded important by all of the 3 algorithms. The coefficient in LASSO suggests that `P9` has a positive effect on `Yield`, while `P17` has a negative effect.

```
knitr::include_graphics("2.png")
```



FSS	BSS	LASSO
B5	B5	
B7		
P2	P2	
P3		
		P6
P7	P7	
P9	P9	P9
		P13
P17	P17	P17
P19	P19	
P20	P20	
P23		
P25		
P29	P29	
		P32
P33	P33	
P34	P34	
P38	P38	
P45		

Problem 2: Data with Outliers

For iid data $X_i, i = 1, 2, ..., n$ from some distribution, the sample standard deviation $\hat{\sigma}$ is often used to estimate the population deviation σ:

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2}.$$

But sometimes, the observed data are contaminated with extreme values or outliers, due to recording errors or other reasons. In order to obtain a robust estimate for σ, $\tilde{\sigma}$ is considered:

$$\tilde{\sigma} = 1.4826 \cdot med_{1\le i\le n}\{|X_i - X_{med}|\},$$

where X_{med} denotes the sample median of $\{X_i : 1 \le i \le n\}$.

Apply the nonparametric bootstrap method to evaluate the sampling distributions of $\tilde{\sigma}$ and $\hat{\sigma}$, and their estimation precision $var(\tilde{\sigma})$ and $var(\hat{\sigma})$.

The data for problem 2: **hw3_problem2.csv**

```
data <- read.table("hw3_problem2.csv", sep = ",", header=TRUE)[,2]
```

1. The function to evaluate the deviations

```
sigma_hat <- function(dat, index) {
  X <- dat[index]
  sqrt((1/(length(dat)-1)) * (sum( (X-mean(X))^2 )))
}

sigma_tilde <- function(dat, index) {
  X <- dat[index]
  1.4826*median(abs(X-median(X)))
}
```

2. Bootstarp

```
boot(data, sigma_hat, R = 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = sigma_hat, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 3.745385 -0.0273867   0.3565152
```

```
boot(data, sigma_tilde, R = 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = sigma_tilde, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 2.585234 0.05279911   0.2867246
```

Hence we have

$$\hat{\sigma} \approx 3.7454, \ \text{Var}(\hat{\sigma}) \approx 0.3698$$

and

$$\tilde{\sigma} \approx 2.5852, \ \text{Var}(\tilde{\sigma}) \approx 0.2987$$