Jin_Kweon(3032235207)_lab7

Jin Kweon

10/14/2017

The models in consideration are ordinary least squares (OLS), principal component regression (PCR), partial least squares regression (PLSR), ridge regression, and lasso.

Note that except for OLS, all the aforementioned methods require hyperparameter tuning.

For PCR and PLSR, the hyperparameter is the number of components. For ridge regression and lasso, the hyperparameter is the regularization parameter.

Q. what is grid search? ==> discretize the range of lambda for lasso and ridge.

Q. Does discretization of hyperparameter space mean make the intervals? ==> yes.

Q. How to deal with categorical variable (do i use them for shrinkage or dimension reduction?) and NA? ==> 1 and 0 .

```
hitters <- Hitters
str(hitters)
```

```
## 'data.frame':    322 obs. of  20 variables:
##  $ AtBat    : int  293 315 479 496 321 594 185 298 323 401 ...
##  $ Hits     : int  66 81 130 141 87 169 37 73 81 92 ...
##  $ HmRun    : int  1 7 18 20 10 4 1 0 6 17 ...
##  $ Runs     : int  30 24 66 65 39 74 23 24 26 49 ...
##  $ RBI      : int  29 38 72 78 42 51 8 24 32 66 ...
##  $ Walks    : int  14 39 76 37 30 35 21 7 8 65 ...
##  $ Years    : int  1 14 3 11 2 11 2 3 2 13 ...
##  $ CAtBat   : int  293 3449 1624 5628 396 4408 214 509 341 5206 ...
##  $ CHits    : int  66 835 457 1575 101 1133 42 108 86 1332 ...
##  $ CHmRun   : int  1 69 63 225 12 19 1 0 6 253 ...
##  $ CRuns    : int  30 321 224 828 48 501 30 41 32 784 ...
##  $ CRBI     : int  29 414 266 838 46 336 9 37 34 890 ...
##  $ CWalks   : int  14 375 263 354 33 194 24 12 8 866 ...
##  $ League   : Factor w/ 2 levels "A","N": 1 2 1 2 2 1 2 1 2 1 ...
##  $ Division : Factor w/ 2 levels "E","W": 1 2 2 1 1 2 1 2 2 1 ...
##  $ PutOuts  : int  446 632 880 200 805 282 76 121 143 0 ...
##  $ Assists  : int  33 43 82 11 40 421 127 283 290 0 ...
##  $ Errors   : int  20 10 14 3 4 25 7 9 19 0 ...
##  $ Salary   : num  NA 475 480 500 91.5 750 70 100 75 1100 ...
##  $ NewLeague: Factor w/ 2 levels "A","N": 1 2 1 2 2 1 1 1 2 1 ...
```

```
hitters <- na.omit(hitters)
hitters$League <- ifelse(as.numeric(hitters$League) == 2, 1, 0)
hitters$Division <- ifelse(as.numeric(hitters$Division) == 2, 1, 0)
hitters$NewLeague <- ifelse(as.numeric(hitters$NewLeague) == 2, 1, 0)
```

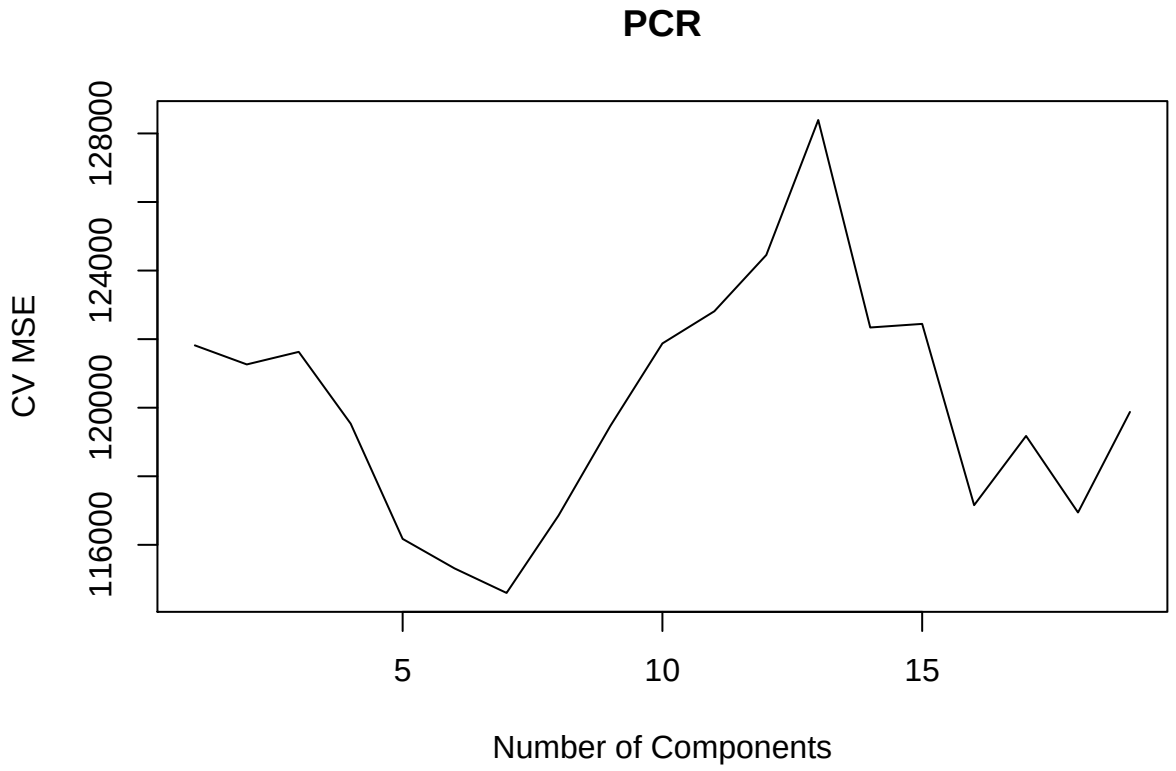
Number of components are always smaller or equal to ncol(hitters)!

Q. why is the length of pcr_fit*validation*PRESS only 19? Isnt it supposed to be ncol(hitters) = 20? ==> cuz one of them is response variable!! And, we do not have intercept here since it is standardized!

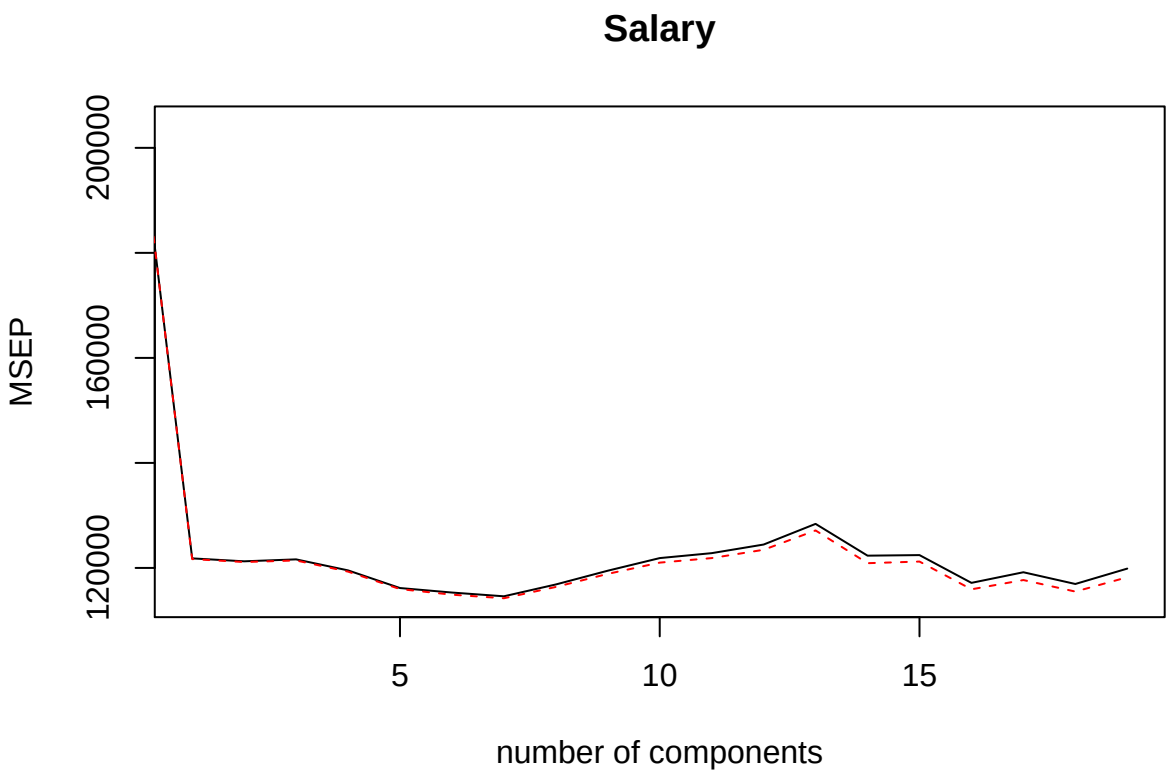
Good reference: https://en.wikipedia.org/wiki/PRESS_statistic

Q. why plot and validation plot do not look the same... although the trend looks similar? ==> They are the same but just different scale.

```
n <- nrow(hitters)
set.seed(100)

pcr_fit <- pcr(Salary ~ ., data = hitters, scale = TRUE, validation = "CV", segments=10)
plot(pcr_fit$validation$PRESS[1, ] / n, type="l", main="PCR",
     xlab="Number of Components", ylab="CV MSE")
```



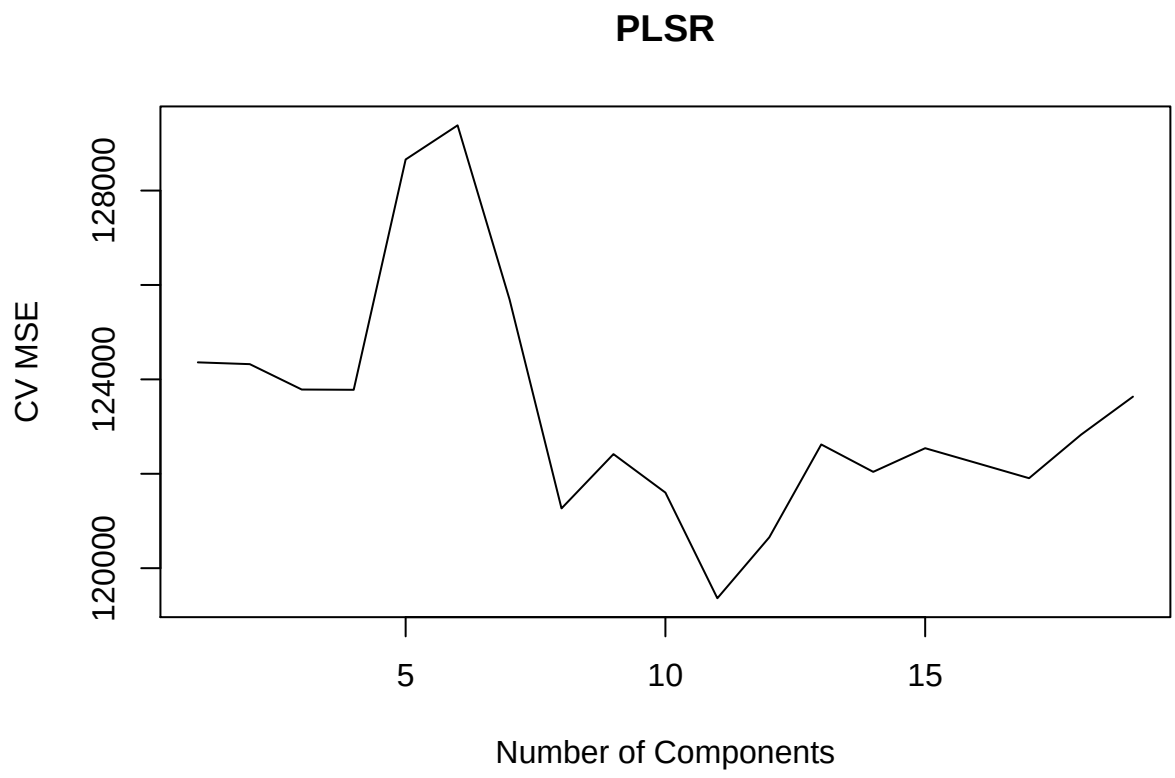

```
validationplot(pcr_fit, val.type = "MSEP", xlim = c(1,19))
```



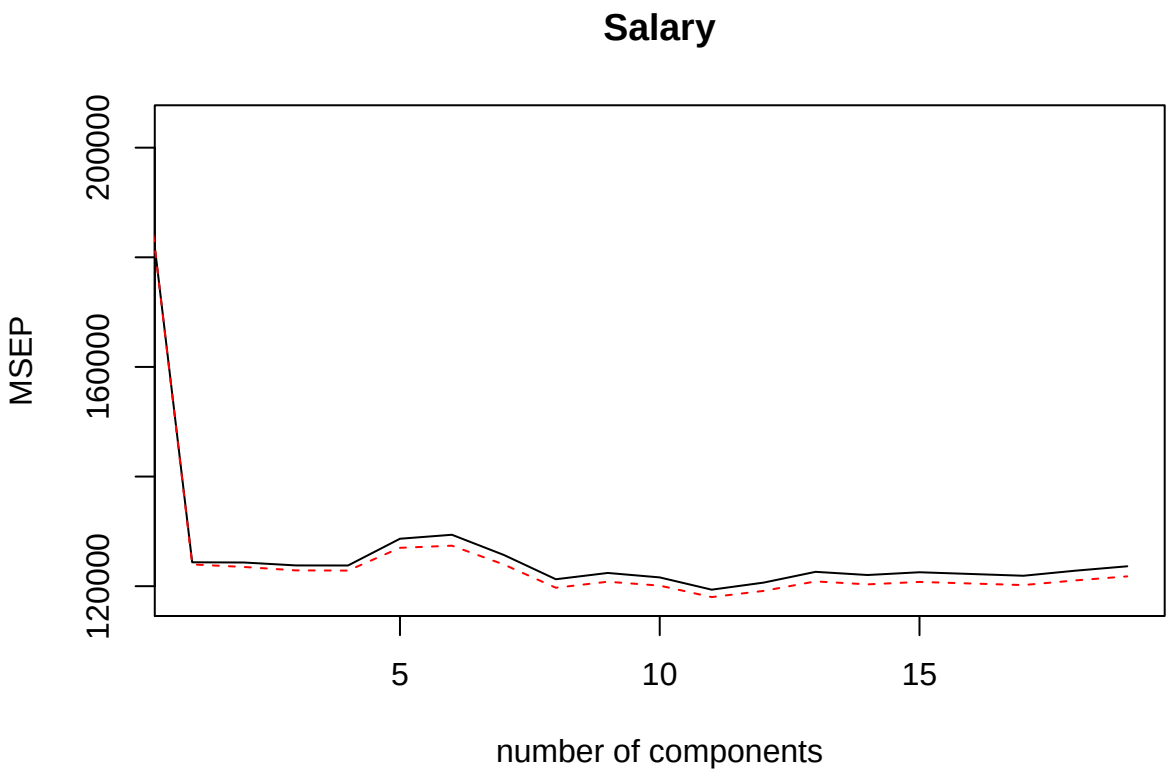

```
which.min(pcr_fit$validation$PRESS[1, ])
```

```
## 7 comps
##       7
```

```
set.seed(200)
plsr_fit <- plsr(Salary ~ ., data = Hitters, scale = TRUE,
                 validation = "CV", segments=10)
plot(plsr_fit$validation$PRESS[1, ] / n , type="l", main="PLSR",
     xlab="Number of Components", ylab="CV MSE")
```




```
validationplot(plsr_fit, val.type = "MSEP", xlim = c(1,19))
```




```
which.min(plsr_fit$validation$PRESS[1, ])
```

```
## 11 comps
```

```
##        11
```

Comment:

So, we keep 7 components for PCR, and 11 for PLSR, as they have the samllest CV-MSE.

Ridge -> α will be 0.

Lasso -> α will be 1.

Q. How can i understand that elastic net penalty formula? ==> This can give the answer betwee n ridge and lasso.

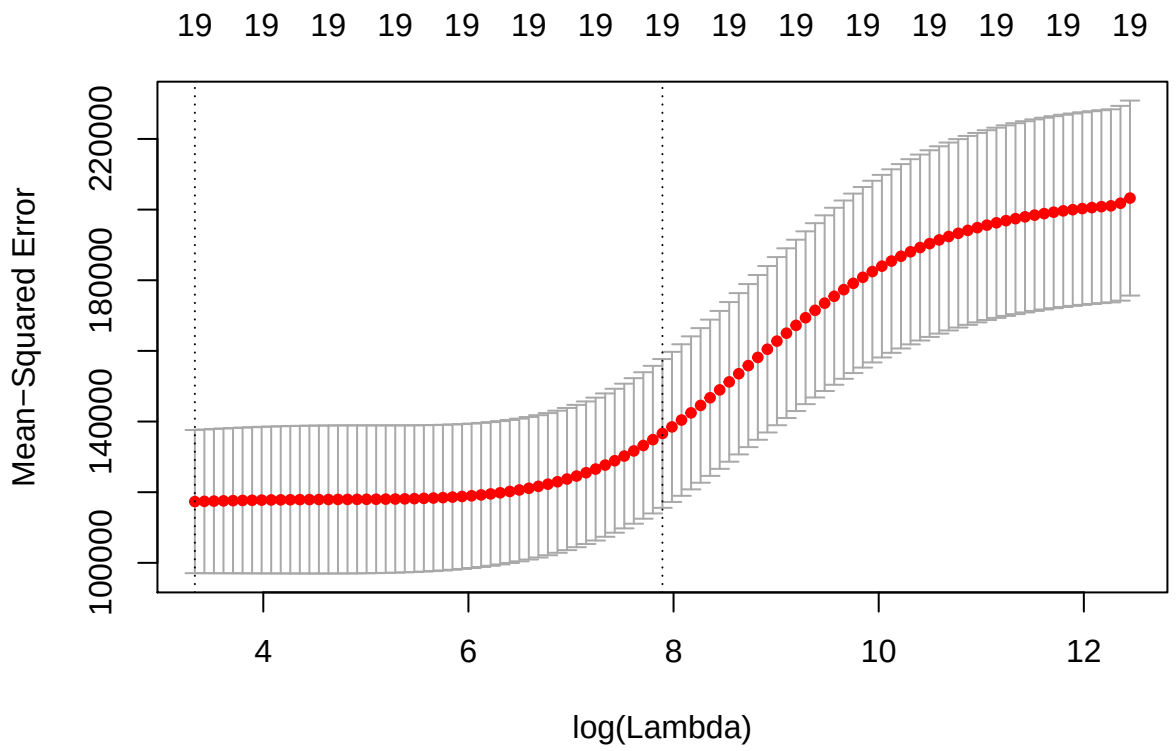
Q. My answers all weird... Why ....? 16 variables to keep and throw out only one....?? ==> So, lasso can give you zero for the coefficien, but ridge you cannot cuz it does not surface in the corner.

Q. I am curious what minimum lambda is telling me... For pcr&plsr, number of components itself is telling me how many components to keep, but how about lambda? What is lambda really telling me?... ==> lambda is giving the penalty parameter and min lambda gives the lambda that gives minimum CV MSE. Lambda is kind of constraint.

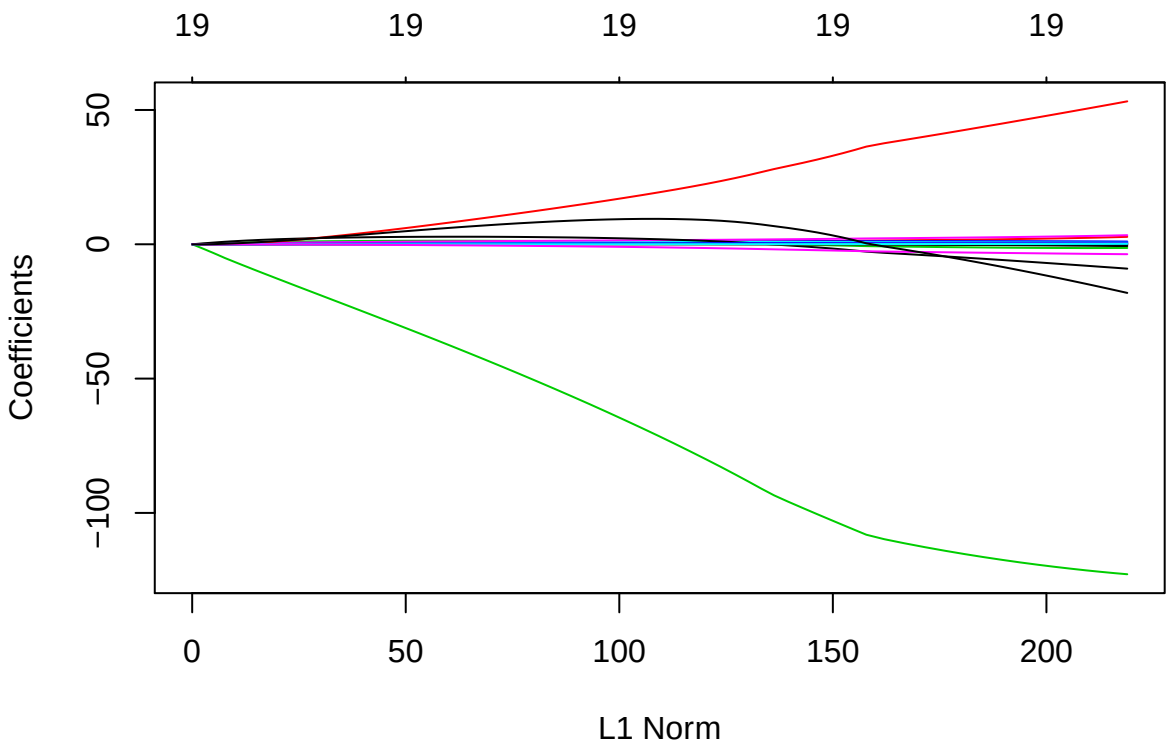
```
set.seed(300)

labridge <- cv.glmnet(as.matrix(hitters[,-19]), as.matrix(hitters[,19]), nfolds = 10, alpha = 0) #it sc
paste("Optimal regularized parameter is", labridge$lambda.min)
```

```
## [1] "Optimal regularized parameter is 28.0171785768831"
```

```
plot.cv.glmnet(labridge)
```




```
labridge2 <- glmnet(as.matrix(hitters[,-19]), as.matrix(hitters[,19]), alpha = 0)
plot.glmnet(labridge2)
```



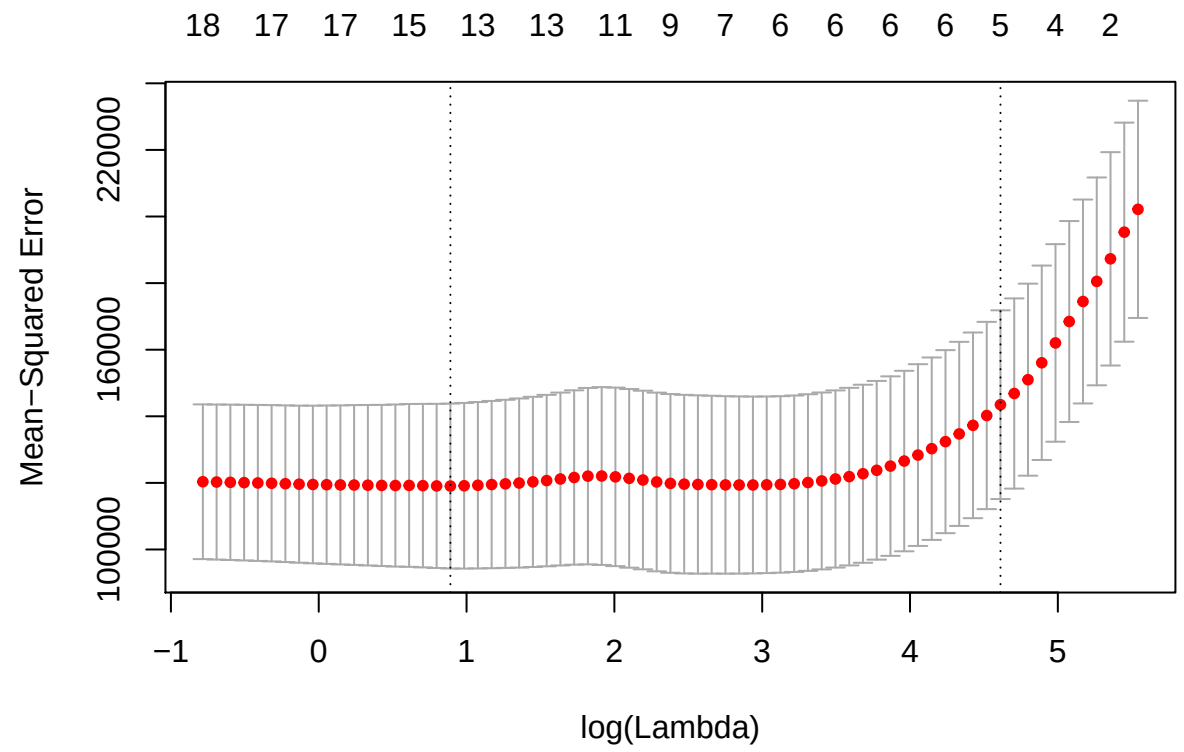

```
coef(labridge, s = "lambda.min")
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)  7.645824e+01
## AtBat       -6.315180e-01
## Hits         2.642160e+00
## HmRun       -1.388233e+00
## Runs         1.045729e+00
## RBI          7.315713e-01
## Walks        3.278001e+00
## Years       -8.723734e+00
## CAtBat       1.256354e-04
## CHits        1.318975e-01
## CHmRun       6.895578e-01
## CRuns        2.830055e-01
## CRBI         2.514905e-01
## CWalks      -2.599851e-01
## League       5.233720e+01
## Division    -1.224170e+02
## PutOuts      2.623667e-01
## Assists      1.629044e-01
## Errors      -3.644002e+00
## NewLeague   -1.702598e+01
```

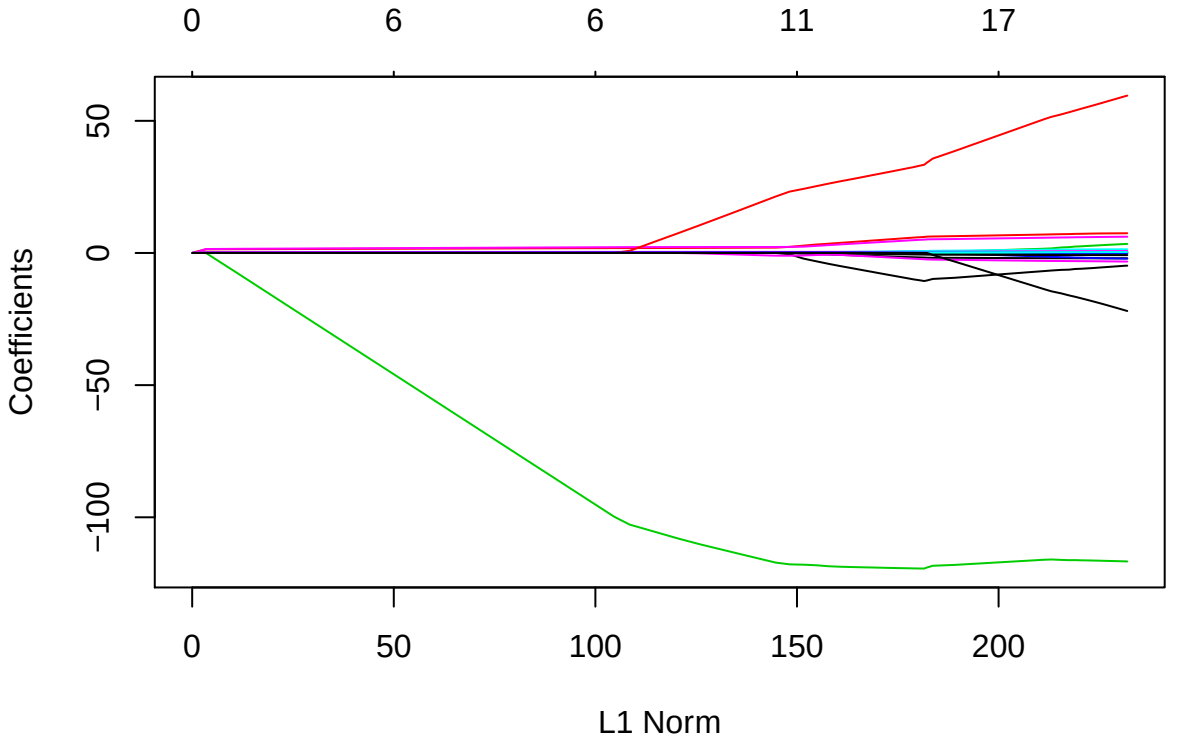
```
set.seed(400)

lablasso <- cv.glmnet(as.matrix(hitters[,-19]), as.matrix(hitters[,19]), nfolds = 10, alpha = 1)
paste("Optimal regularized parameter is",lablasso$lambda.min)
```

```
## [1] "Optimal regularized parameter is 2.43679131234084"
```

```
plot.cv.glmnet(lablasso)
```




```
lablasso2 <- glmnet(as.matrix(hitters[,-19]), as.matrix(hitters[,19]), alpha = 1)
plot.glmnet(lablasso2)
```




```
coef(lablasso, s = "lambda.min")
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  129.4155571
## AtBat         -1.6130155
## Hits           5.8058915
## HmRun          .
## Runs           .
## RBI            .
## Walks          4.8469340
## Years         -9.9724045
## CAtBat         .
## CHits          .
## CHmRun         0.5374550
## CRuns          0.6811938
## CRBI           0.3903563
## CWalks        -0.5560144
## League        32.4646094
## Division    -119.3480842
## PutOuts        0.2741895
## Assists        0.1855978
## Errors        -2.1650837
## NewLeague      .
```

For methods like PCR and ridge, we have to tune the hyperparameter(s), so an additional CV is performed within each of the CV iterations. In other words, you are asked to do a 10-fold CV (for hyperparameter tuning) within a 10-fold CV (for estimating the predictive power).

In this class, MSE is the default measure of predictive power for regression problems.

Q. How to do 10 fold csv for hyperparameter tuning for OLS? ==> no parameter tuning for OLS!!! Q. IN PCR and PLSR, summary(pcr_fit) square is little bit different with MSEP(pcr_fit) because of decimal points difference? ==> decimal or internal adjustment in R!!! Q. Using 'plsr_fit$coefficients[,,which.min(plsr_fit$validation$PRESS)]' and 'coef(pcr_fit, intercept = T)[,,which.min(plsr_fit$validation
how do we get the coefficient of smallest MSE with intercept? ==> it might be hard. I might scale x and y and use functionm, so i dont need an intercept. Q. why do i have different MSE with MSEP(pcr_fit) and MSEP(plsr_fit) for pcr and plsr? ==> plsr and pcr and OLS have the same mse when you use all components. (same y hat) Q. So, if i use predict function for pcr and plsr do they make intercept by itself? ==> YEs!!! Q. Why lasso and ridge have the same MSE? ==> when lambda is equal to zero, then lasso and ridge and OLS have the same mse!!! (same y hat)

Useful link: https://machinelearningmastery.com/linear-regression-in-r/

```r
x <- model.matrix(lm(Salary ~.-1, data = hitters))
y <- hitters$Salary
xint <- model.matrix(lm(Salary ~., data = hitters))

fold <- createFolds(1:nrow(hitters))

mse <- matrix(0, 5, 10)

for (i in 1:10){
  ytest <- y[as.vector(fold[[i]])]
  xtest <- xint[as.vector(fold[[i]]),]

  ytrain <- y[-(fold[[i]])]
  xtrain <- xint[-(as.vector(fold[[i]])),]
```

```r
#OLS
olslab <- lm(ytrain ~ xtrain)
tablelab <- summary(olslab)$coefficients[,-4]
olsbeta <- round(tablelab, 2)[,1]
mse[1,i] <- sum((ytest - (xtest %*% olsbeta))^2) / length(ytest)



#PCR
pcr_fit <- pcr(salary ~ .,
               data = as.data.frame(cbind(salary = ytrain, xtrain = xtrain)[,-2]),
               scale = TRUE, validation = "CV", segments=10)
which.min(pcr_fit$validation$PRESS[1, ])

#MSEP(pcr_fit)
#pcr_fit$validation$PRESS # predicted residual error sum of squares - proportional to MSE
#summary(pcr_fit)

#pcr_fit$coefficients[,,which.min(pcr_fit$validation$PRESS)]
#coef(pcr_fit, intercept = T)[,,which.min(pcr_fit$validation$PRESS)]
yhat <- predict(pcr_fit, xtest[,-1],
                ncomp = unname(which.min(pcr_fit$validation$PRESS[1, ])))
pcr_MSE <- mean((ytest - yhat)^2)
mse[2,i] <- pcr_MSE


#PLSR
plsr_fit <- plsr(salary ~ .,
                 data = as.data.frame(cbind(salary = ytrain, xtrain = xtrain)[,-2]),
                 scale = TRUE, validation = "CV", segments=10)
which.min(plsr_fit$validation$PRESS[1, ])

#MSEP(plsr_fit)
#plsr_fit$validation$PRESS
#summary(plsr_fit)

#plsr_fit$coefficients[,,which.min(plsr_fit$validation$PRESS)]
#coef(pcr_fit, intercept = T)[,,which.min(plsr_fit$validation$PRESS)]

yhat <- predict(plsr_fit, xtest[,-1],
                ncomp = unname(which.min(plsr_fit$validation$PRESS[1, ])))
plsr_MSE <- mean((ytest - yhat)^2)
mse[3,i] <- plsr_MSE



#Ridge
# ridge_fit <- cv.glmnet(xtrain, ytrain, nfolds = 10, alpha = 0)
# ridge_fit$lambda.min
# coef(ridge_fit, s = "lambda.min")
# sum((ytest - as.vector(xtest %*% coef(ridge_fit, s = "lambda.min")[-2,]))^2) / length(ytest)
```

```
  # --> The same as the one below

  ridge_fit2 = cv.glmnet(xtrain, ytrain, nfolds = 10, alpha = 0)
  yhat = predict(ridge_fit2, xtest, s = 'lambda.min')
  ridge_MSE = mean((ytest - yhat)^2)
  mse[4,i] <- ridge_MSE




  #Lasso
  lasso_fit <- cv.glmnet(xtrain, ytrain, nfolds = 10, alpha = 1)
  yhat = predict(lasso_fit, xtest, s = 'lambda.min')
  lasso_MSE = mean((ytest - yhat)^2)
  mse[5,i] <- lasso_MSE

}
rownames(mse) <- c("ols", "pcr", "plsr", "ridge", "lasso")

mse
```

```
##           [,1]     [,2]     [,3]      [,4]     [,5]      [,6]     [,7]
## ols   86867.28 93171.13 274218.2 102351.46 63513.48  98547.88 109282.2
## pcr   86010.41 89759.25 300569.6 105461.86 75058.56 110117.59 106107.3
## plsr  83073.27 96618.38 317519.6 105914.89 79421.58 102964.57 107058.5
## ridge 78005.92 92268.14 301383.4  95011.71 72436.93 121831.67 101125.4
## lasso 82446.12 85559.97 287668.5  96289.01 61569.08 119580.67 103210.2
##            [,8]      [,9]    [,10]
## ols    90570.87  82238.05 190037.9
## pcr    93573.94  95380.27 192244.6
## plsr   91865.31  92869.49 192244.6
## ridge 113615.68 118805.01 122791.3
## lasso  91390.69  95304.59 176625.9
```

```
apply(mse, 1, mean)
```

```
##      ols      pcr     plsr    ridge    lasso
## 119079.9 125428.3 126955.0 121727.5 119964.5
```

Comment:

For hyperparameter tuning, use a 10-fold CV; for estimating predictive power, also use a 10-fold CV.

It means that I need to get 10 folds CV and get 10 MSE for each 5 model. And, for each MSE is came from 10 folds cv and get minimum from there as well.

The best model is ridge!!!