Practical: Supervised Learning - Regression II

David Vichansky

08-12-2020

Load the packages.

```
library(ISLR)
library(glmnet)
```

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

```
## Loaded glmnet 4.0-2
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
```

```
library(eply)
library(leaps)
```

1. Prepare a dataframe baseball from the Hitters dataset where you remove the baseball players for which the Salary is missing. How many baseball players are left?

```
Hitters <- Hitters %>%
  filter(!is.na(Salary))

nrow(Hitters)
```

```
## [1] 263
```

Using 'nrow()' gives us back 263 baseball players in the dataset.

2. Create baseball_train (50%), baseball_valid (30%), and baseball_test (20%) datasets.

```
#Set seed to make your partition reproducible
set.seed(123)
#Create train dataset
train_size <- floor(0.50 * nrow(Hitters))

train_ind <- sample(seq_len(nrow(Hitters)), size = train_size, replace = FALSE)

baseball_train <- Hitters[train_ind, ]

#Create validate dataset
val_test <- Hitters[-train_ind, ]

val_size <- floor(0.3 * nrow(val_test))

val_ind <- sample(seq_len(nrow(val_test)), size = val_size, replace = FALSE)

baseball_valid <- val_test[val_ind, ]

#Create test dataset
baseball_test <- val_test[-val_ind, ]

#baseball_train <- Hitters %>%
#                     sample_frac(0.5, replace=FALSE)
#
#baseball_valid <- Hitters %>%
#                     sample_frac(0.3, replace=FALSE)
#
#baseball_test <- Hitters %>%
#                     sample_frac(0.2, replace=FALSE)
```

3. Create a function called lm_mse() with as inputs (1) a formula, (2) a training dataset, and (3) a test dataset which outputs the mse on the test dataset for predictions from a linear model.

```
lm_mse <- function(formula, train_data, valid_data) {
  y_name <- as.character(formula)[2]
  y_true <- valid_data[[y_name]]

  ##The remainder of the function here

  #Space to specify formula, now done outside of function instead
  #formula <- formula_input

```

```
  #Linear model fit, Include only subset corresponding to training set
  lm.fit <- lm(formula = formula, data=train_data)

  #Predict values, by selecting only the observations that are not in the training set
  #predict <- predict(lm.fit, data.frame(X))
  #predict_y <- predict((lm.fit, valid_data)[-train_data])
  mse <- mean((y_true - predict(lm.fit, newdata = valid_data))^2)

  #Print result of mean square error
  print(mse)

}

#mse <- function(y_true, y_pred) {
#(1/length(y_true))*sum((y_true - y_pred)^2)
#}
```

4. Try out your function with the formula Salary ~ Hits + Runs, using baseball_train and baseball_valid.

```
formula_input <- Salary ~ Hits + Runs

lm_mse(formula_input, baseball_train, baseball_valid)
```

```
## [1] 190528.5
```

```
source("generate_formulas.R")
```

5. Create a character vector of all predictor variables from the Hitters dataset. colnames() may be of help. Note that Salary is not a predictor variable.

```
column_names <- colnames(Hitters)

column_names <- column_names[!(column_names %in% "Salary")]

#I think that we should follow this and remove non-numeric variables when carrying out

#Check variable types inside Hitters data set
#lapply(Hitters,class)

#Choose only numeric variables
#column_names <- colnames(select_if(Hitters, is.numeric))

#Exclude 'Salary' from variables vector
#column_names <- column_names[!(column_names %in% "Salary")]
```

```
#Print final vector
#column_names

#Create vector which only contains the character string 'Salary
#c("Salary")
```

6. Generate all formulas with as outcome Salary and 3 predictors from the Hitters data. Assign this to a variable called formulas. There should be 969 elements in this vector.

```
formulas <- generate_formulas(3, column_names, c("Salary"))

length(formulas)
```

```
## [1] 969
```

Thus indeed our 'formulas' variable contains 969 inside of it.

7. Use a for loop to find the best set of 3 predictors in the Hitters dataset based on MSE. Use the baseball_train and baseball_valid datasets.

```
#Initialise some vector to store values
a <- numeric()

#Use for loop
for (i in 1:length(formulas)){
  result <- lm_mse(as.formula(formulas[i]), baseball_train, baseball_valid)
  a[i] <- result
}

min(a)

min_index = which.min(a)
#Return formula which minimises the mean square error
formulas[min_index]
```

8. Do the same for 1, 2 and 4 predictors. Now select the best model with 1, 2, 3, or 4 predictors in terms of its out-of-sample MSE.

```
formulas1 <- generate_formulas(1, column_names, c("Salary"))
formulas2 <- generate_formulas(2, column_names, c("Salary"))
formulas4 <- generate_formulas(4, column_names, c("Salary"))

b <- numeric()
c <- numeric()
d <- numeric()

#Use for loop for 1 variable
```

```r
for (i in 1:length(formulas1)){

  result <- lm_mse(as.formula(formulas1[i]), baseball_train, baseball_valid)
  b[i] <- result
}

min_index1 = which.min(b)

formulas1[min_index1]

#Use for loop for 2 variables
for (i in 1:length(formulas2)){

  #Remember to set.seed()
  set.seed(i)

  result <- lm_mse(as.formula(formulas2[i]), baseball_train, baseball_valid)
  c[i] <- result
}

min_index2 = which.min(c)

formulas2[min_index2]

#Use for loop for 4 variables
for (i in 1:length(formulas4)){

  #Remember to set.seed()
  set.seed(i)

  result <- lm_mse(as.formula(formulas4[i]), baseball_train, baseball_valid)
  d[i] <- result
}

min_index4 = which.min(d)

formulas4[min_index4]

#Select best number of parameters by number of variables first
selected_formula <- formulas2[min_index2]
```

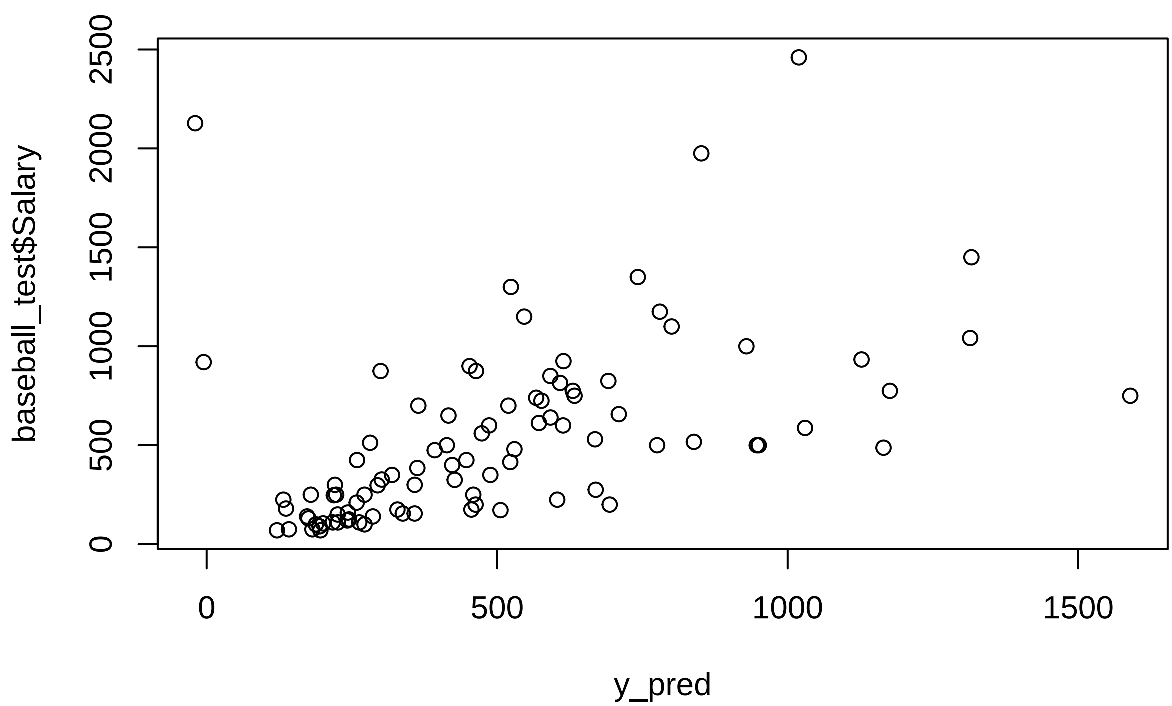
We observe that the smallest mse is for model with 1 variable used in predictor.

9. Calculate the test MSE for this model. Then, create a plot comparing predicted values (mapped to x position) versus observed values (mapped to y position) of baseball_test.

```r
selected_formula <- formulas2[min_index2]

lm_mse(as.formula(selected_formula),baseball_train,baseball_valid)
```

```
## [1] 102319.4
```

```r
#Build linear regression model with 2 predictorvariables
lm.fit_2 <- lm(as.formula(selected_formula),baseball_train)

y_pred <- predict(lm.fit_2, newdata = baseball_test)

plot(y_pred, baseball_test$Salary)
```



10. Read through the help file of glmnet. We are going to perform a linear regression with normal (gaussian) error terms. What format should our data be in?

The data needs to be in the form of x <- matrix, and y <- vector.

11. First generate the input matrix using (a variation on) the following code. Remember that the "." in a formula means "all available variables". Make sure to check that this x_train looks like what you would expect.

```
x_train <- model.matrix(Salary~. , data = baseball_train)[, -1]

#Use the formula we worked with earlier
#x_train <- model.matrix(as.formula(selected_formula), data = baseball_train)[, -1]
```

12. Using glmnet(), perform a LASSO regression with the generated x_train as the predictor matrix and Salary as the response variable. Set the lambda parameter of the penalty to 15. NB: Remove the intercept column from the x_matrix – glmnet adds an intercept internally.

```
y_train <- baseball_train$Salary

lasso.mod = glmnet(x_train, y_train, alpha=1, lambda=15)
```

13. The coefficients for the variables are in the beta element of the list generated by the glmnet() function. Which variables have been selected? You may use the coef() function.

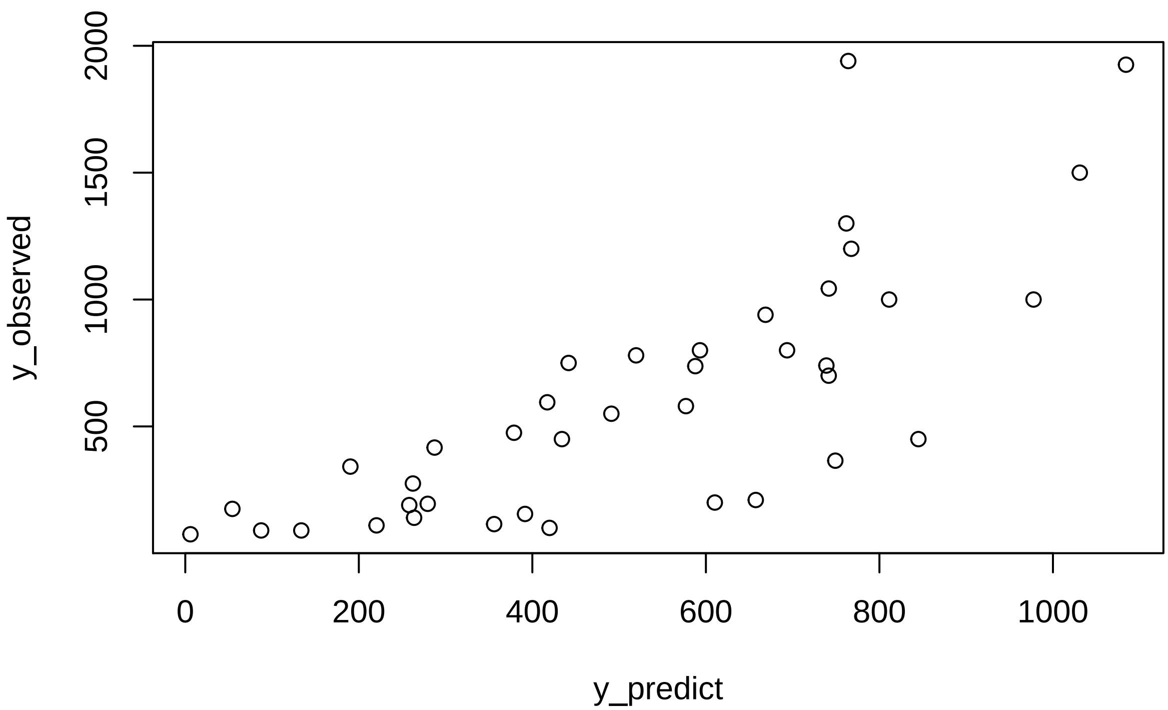
```
coef(lasso.mod)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -83.01510022
## AtBat          .
## Hits           2.00394100
## HmRun         -0.27249549
## Runs           1.81662652
## RBI            .
## Walks          2.47984353
## Years          .
## CAtBat         .
## CHits          .
## CHmRun         0.59034459
## CRuns          0.48612783
## CRBI           .
## CWalks         .
## LeagueN       14.00421528
## DivisionW    -76.65862144
## PutOuts        0.05419624
## Assists       -0.16907032
## Errors         .
## NewLeagueN     .
```

14. Create a predicted versus observed plot for the model you generated with the baseball_valid data. Use the predict() function for this! What is the MSE on the validation set?

```
x_validation <- model.matrix(Salary~. , data = baseball_valid)[, -1]
#Observed value to go on y-axis
y_observed <- baseball_valid$Salary
```

```r
#Use x_validation to predict new data points using the lasso func
y_predict <- predict(lasso.mod, x_validation)

#Plot graph
plot(y_predict, y_observed)
```




```r
#Calculate mse
mse <- mean((y_observed - y_predict)^2)
mse
```

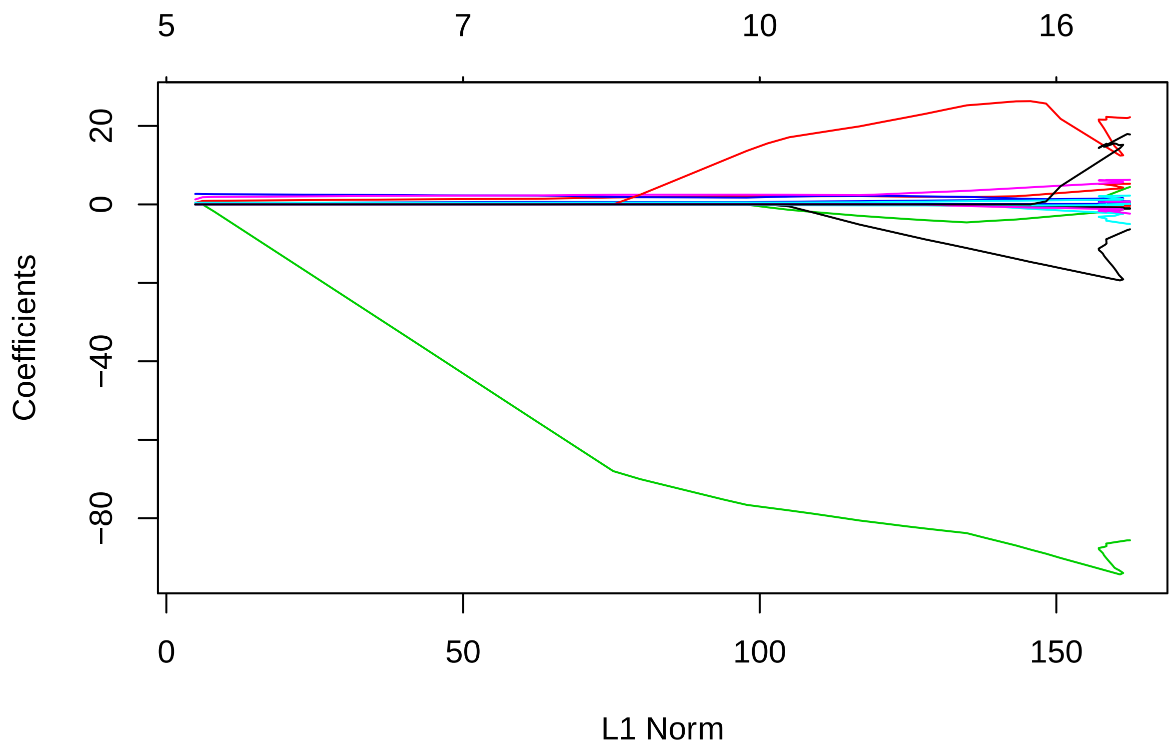
```
## [1] 109294.3
```

15. Fit a LASSO regression model on the same data as before, but now do not enter a specific lambda value. What is different about the object that is generated? Hint: use the coef() and plot() methods on the resulting object.

```r
#Create an array of different lambda functions ranging from 10^-1 and 10^2
grid = 10^seq(2,-1, length =100)

lasso.mod2 = glmnet(x_train, y_train, alpha=1, lambda=grid)

coef(lasso.mod2)
```

```
##     [[ suppressing 100 column names 's0', 's1', 's2' ... ]]
```

```r
plot(lasso.mod2)
```



We observe that the the intercept of the coefficient increase with larger values for lambda.

16. Use the cv.glmnet function to determine the lambda value for which the out-of-sample MSE is lowest using 15-fold cross validation. As your dataset, you may use the training and validation sets bound together with bind_rows(). What is the best lambda value?

```r
set.seed (1)

cv.out =cv.glmnet(x_train, y_train, alpha=1)

#Bind the training and validation data
x_value <- rbind(x_train, x_validation)
y_value <- c(y_train, y_observed)

cv_output <- cv.glmnet(x_value, y_value, alpha=1, nfolds = 15)

#Look at this output
cv_output
```

```
##
```

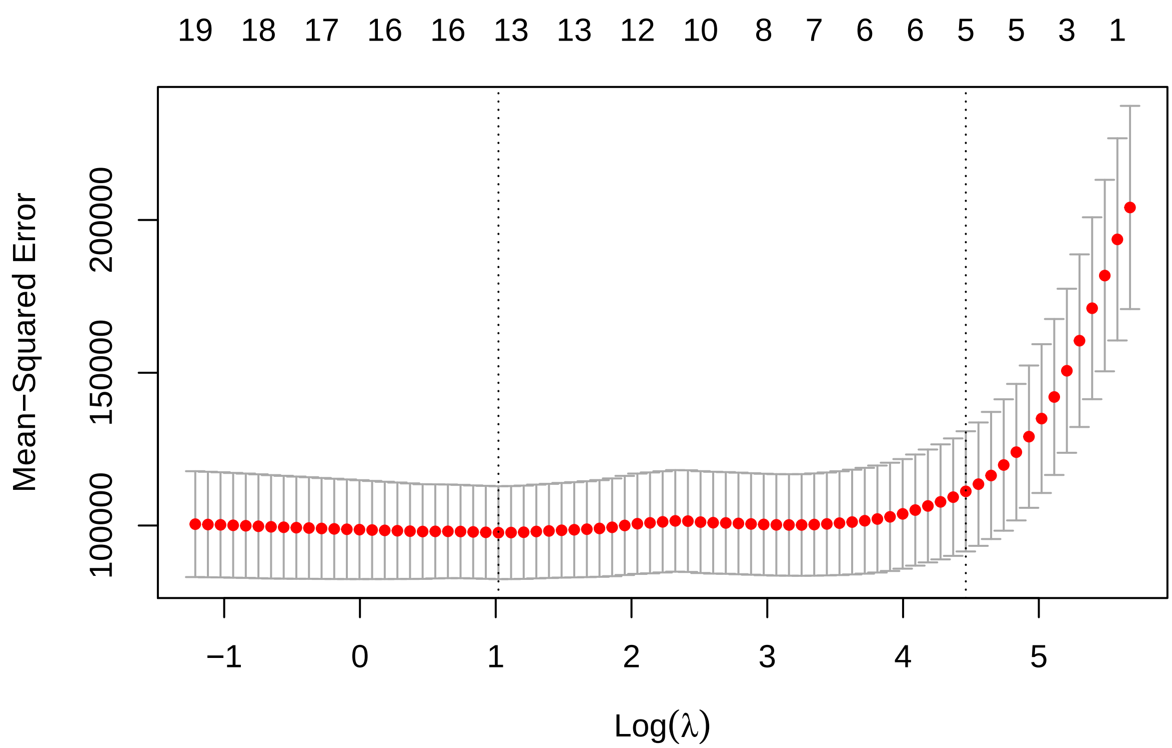
```
## Call:  cv.glmnet(x = x_value, y = y_value, nfolds = 15, alpha = 1)
##
## Measure: Mean-Squared Error
##
##     Lambda Measure    SE Nonzero
## min   2.77   97663 15204      13
## 1se  86.68  111188 19671       5
```

```r
#Obtain smallest lambda
min_lambda <- cv_output$lambda.min
min_lambda
```

```
## [1] 2.773143
```

17. Try out the plot() method on this object. What do you see? What does this tell you about the bias-variance tradeoff?

```r
plot(cv_output)
```

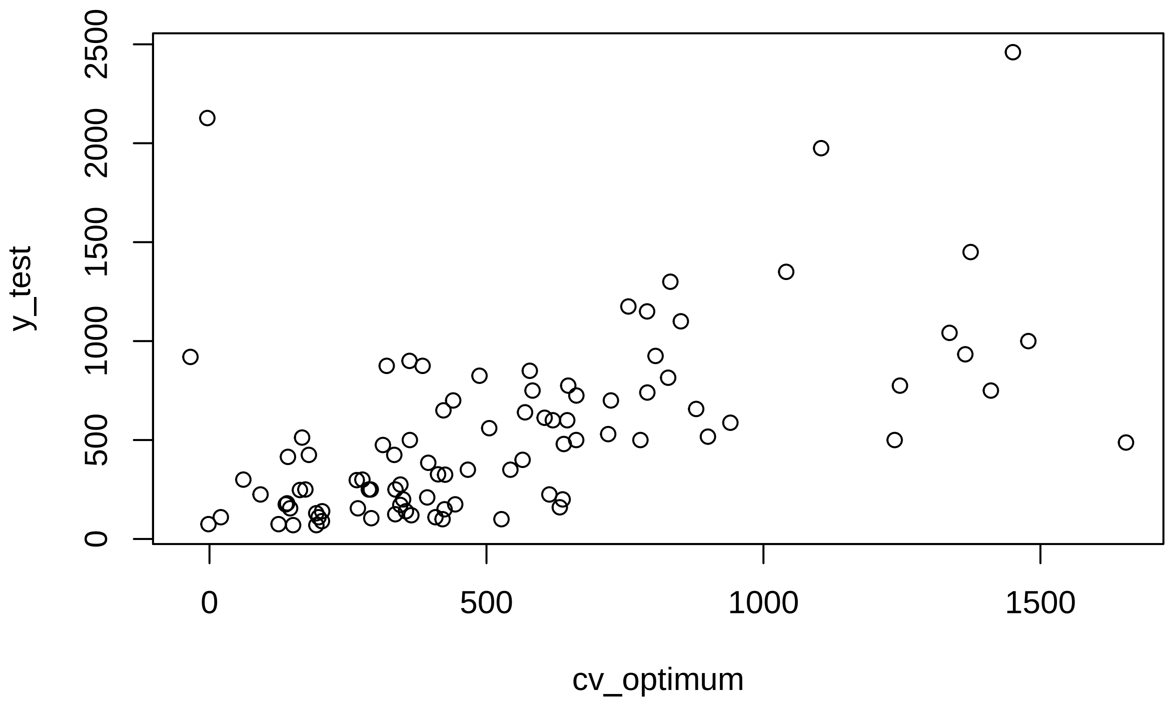


We observe that larger lambda leads to greater degree of variance (due to larger mean squared error on the y-axis). This is due to a overfitting the model.

18. Use the predict() method directly on the object you just created to predict new salaries for the baseball players in the baseball_test dataset using the best lambda value you just created (hint: you need to use the s argument, look at ?predict.cv.glmnet for help). Create another predicted-

observed scatter plot.

```
#First create lasso model variables
x_test <- model.matrix(Salary~. , data = baseball_test)[, -1]
y_test <- baseball_test$Salary

# predict(cv.object, newx = x_data, s=c())
cv_optimum = predict(cv_output, x_test, s=min_lambda)

plot(cv_optimum, y_test)
```




```
mse_optimum = mean((y_test - cv_optimum)^2)
mse_optimum
```

```
## [1] 157398.4
```