HW1

Zhaohua Chunyu

2023-02-13

In this exercise, we predict the sale price of a house using its other characteristics.

```
train = read_csv("/Users/zozochunyu/Documents/DSII/HW/DSII_HW1/housing_training.csv") %>%
  janitor::clean_names()
```

```
## Rows: 1440 Columns: 26
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (4): Overall_Qual, Kitchen_Qual, Fireplace_Qu, Exter_Qual
## dbl (22): Gr_Liv_Area, First_Flr_SF, Second_Flr_SF, Total_Bsmt_SF, Low_Qual_...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
test = read_csv("/Users/zozochunyu/Documents/DSII/HW/DSII_HW1/housing_training.csv") %>%
  janitor::clean_names()
```

```
## Rows: 1440 Columns: 26
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (4): Overall_Qual, Kitchen_Qual, Fireplace_Qu, Exter_Qual
## dbl (22): Gr_Liv_Area, First_Flr_SF, Second_Flr_SF, Total_Bsmt_SF, Low_Qual_...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# delete rows containing the missing data
train = na.omit(train)
test = na.omit(test)

xtrain = model.matrix(sale_price ~ ., train)[,-1]
ytrain = train$sale_price

xtest = model.matrix(sale_price ~ ., test)[,-1]
ytest = test$sale_price

ctrl1 = trainControl(method = "repeatedcv", number = 10, repeats = 5)
```

Least squares

```
set.seed(2023)
lm.fit <- train(xtrain,ytrain,
                method = "lm",
                trControl = ctrl1)
```

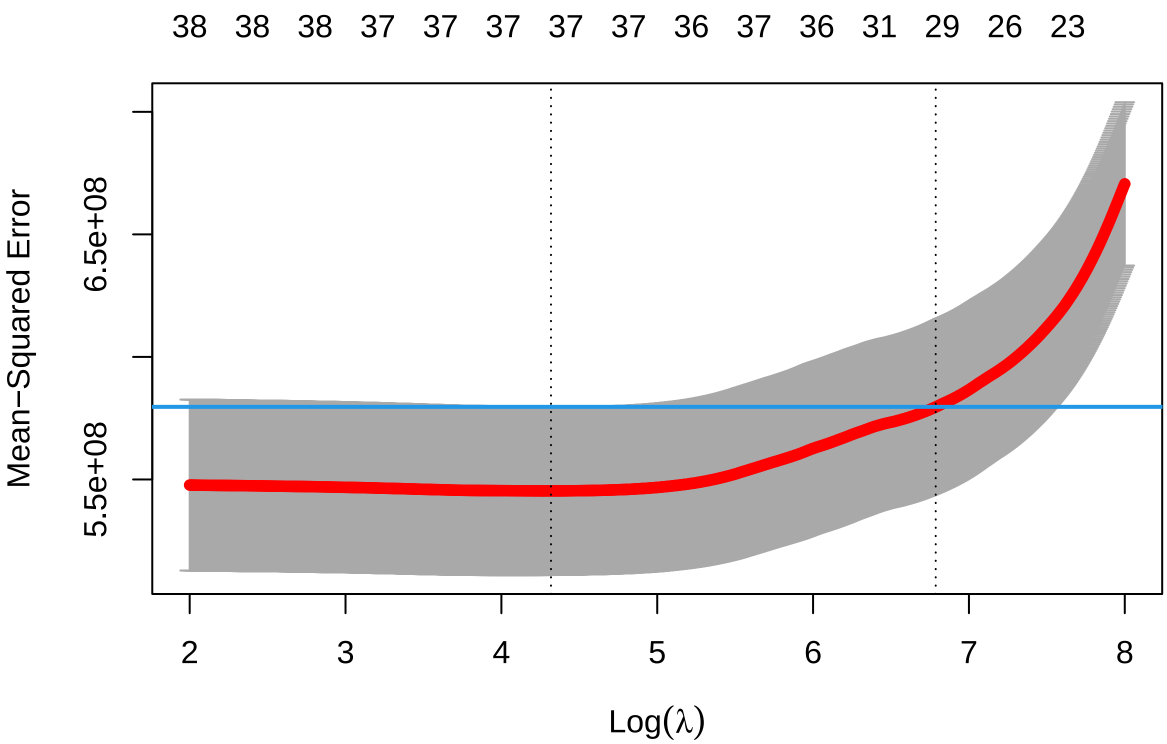
```r
pred.lm = predict(lm.fit, newx = xtest)
mse.lm = mean((ytest-pred.lm)^2)
mse.lm
```

```
## [1] 479188190
```

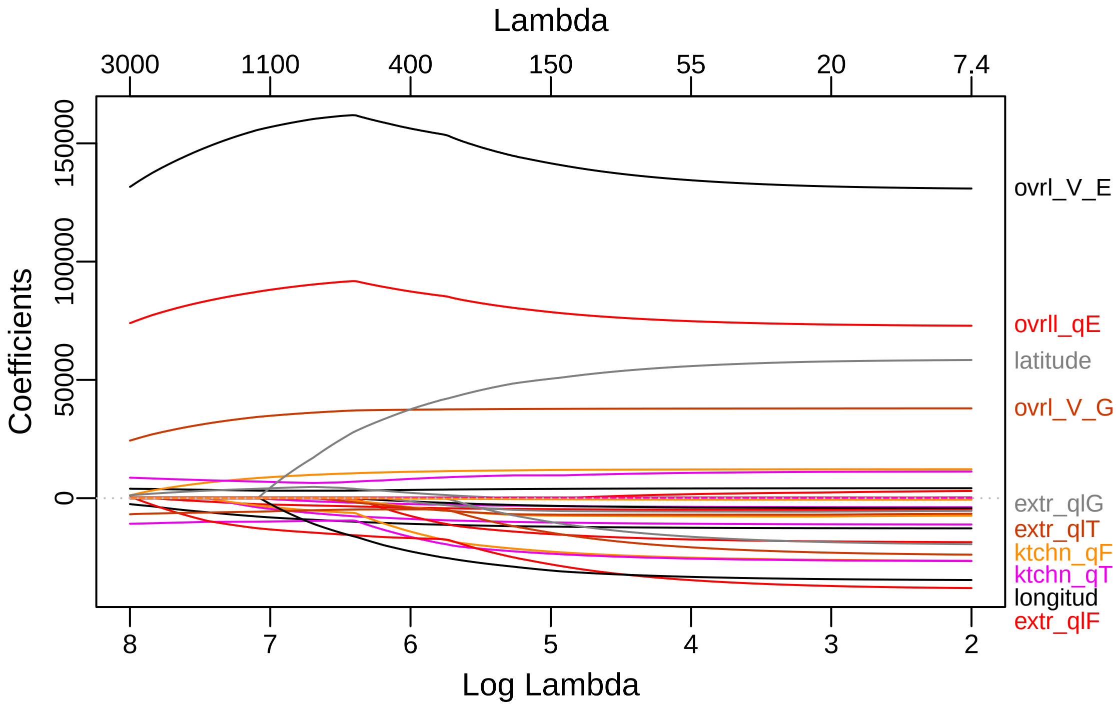
When fitting a least squares model, the test error is 4.7918819×10^{8}.

LASSO

```r
set.seed(2023)
cv.lasso = cv.glmnet(xtrain, ytrain,
                     standardize = TRUE,
                     alpha = 1,
                     lambda = exp(seq(8, 2, length = 1000)))
plot(cv.lasso)
abline(h = (cv.lasso$cvm + cv.lasso$cvsd)[which.min(cv.lasso$cvm)], col = 4, lwd = 2)
```




```r
# cv.lasso$glmnet.fit is a fitted glmnet object using the full training data
# plot(cv.lasso$glmnet.fit, xvar = "lambda", label=TRUE)
plot_glmnet(x = cv.lasso$glmnet.fit)
```




```
cv.lasso$lambda.min
```

```
## [1] 75.06229
```

```
cv.lasso$lambda.1se
```

```
## [1] 886.0619
```

```
lasso.fit.min = predict(cv.lasso, s = "lambda.min", type = "coefficients") ;  lasso.fit.min
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                lambda.min
## (Intercept)                 -4.806120e+06
## gr_liv_area                  6.524898e+01
## first_flr_sf                 8.136255e-01
## second_flr_sf                .
## total_bsmt_sf                3.543974e+01
## low_qual_fin_sf             -4.071470e+01
## wood_deck_sf                 1.157547e+01
## open_porch_sf                1.529046e+01
## bsmt_unf_sf                 -2.087834e+01
## mas_vnr_area                 1.093989e+01
## garage_cars                  4.064961e+03
## garage_area                  8.207621e+00
## year_built                   3.229258e+02
## tot_rms_abv_grd             -3.582772e+03
## full_bath                   -3.779630e+03
## overall_qualAverage         -4.824402e+03
## overall_qualBelow_Average   -1.240562e+04
## overall_qualExcellent        7.565602e+04
## overall_qualFair            -1.069627e+04
## overall_qualGood             1.208675e+04
## overall_qualVery_Excellent   1.359728e+05
```

```
## overall_qualVery_Good       3.785440e+04
## kitchen_qualFair           -2.458207e+04
## kitchen_qualGood           -1.696340e+04
## kitchen_qualTypical        -2.509214e+04
## fireplaces                  1.043171e+04
## fireplace_quFair           -7.617540e+03
## fireplace_quGood            .
## fireplace_quNo_Fireplace    1.266012e+03
## fireplace_quPoor           -5.599848e+03
## fireplace_quTypical        -6.985489e+03
## exter_qualFair             -3.319039e+04
## exter_qualGood             -1.494540e+04
## exter_qualTypical          -1.941089e+04
## lot_frontage                9.924786e+01
## lot_area                    6.041190e-01
## longitude                  -3.270618e+04
## latitude                    5.462173e+04
## misc_val                    8.124489e-01
## year_sold                  -5.504641e+02
```

```
lasso.fit.1se = predict(cv.lasso, s = "lambda.1se", type = "coefficients") ; lasso.fit.1se
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                               lambda.1se
## (Intercept)                -1.948645e+06
## gr_liv_area                 5.615314e+01
## first_flr_sf                1.142665e+00
## second_flr_sf               .
## total_bsmt_sf               3.677563e+01
## low_qual_fin_sf            -2.489128e+01
## wood_deck_sf                8.189861e+00
## open_porch_sf               7.534159e+00
## bsmt_unf_sf                -1.922749e+01
## mas_vnr_area                1.429866e+01
## garage_cars                 3.145401e+03
## garage_area                 1.130120e+01
## year_built                  3.154739e+02
## tot_rms_abv_grd            -1.046696e+03
## full_bath                   .
## overall_qualAverage        -2.969075e+03
## overall_qualBelow_Average  -8.857011e+03
## overall_qualExcellent       8.971786e+04
## overall_qualFair           -5.830308e+03
## overall_qualGood            9.599970e+03
## overall_qualVery_Excellent  1.593488e+05
## overall_qualVery_Good       3.579178e+04
## kitchen_qualFair           -4.826778e+03
## kitchen_qualGood            .
## kitchen_qualTypical        -9.694576e+03
## fireplaces                  6.544737e+03
## fireplace_quFair            .
## fireplace_quGood            4.607899e+03
## fireplace_quNo_Fireplace    .
## fireplace_quPoor            .
## fireplace_quTypical         .
```

```
## exter_qualFair              -1.419105e+04
## exter_qualGood               .
## exter_qualTypical           -5.223227e+03
## lot_frontage                 6.761224e+01
## lot_area                     5.522474e-01
## longitude                   -8.528497e+03
## latitude                     1.364228e+04
## misc_val                     .
## year_sold                    .
```

```r
pred.lasso.min = predict(cv.lasso, s = "lambda.min", newx = xtest)
mse.lasso.min = mean((ytest - pred.lasso.min)^2)
mse.lasso.min
```

```
## [1] 479950966
```

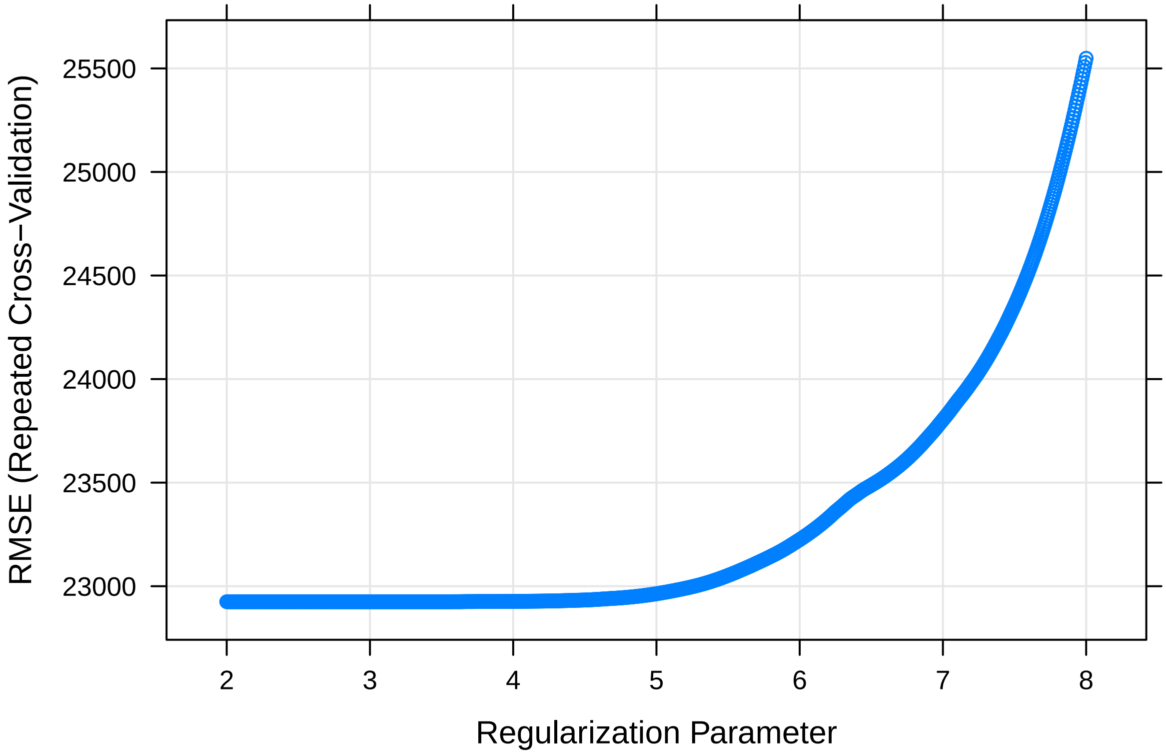
```r
pred.lasso.1se = predict(cv.lasso, s = "lambda.1se", newx = xtest)
mse.lasso.1se = mean((ytest - pred.lasso.1se)^2)
mse.lasso.1se
```

```
## [1] 520300643
```

When fitting a Lasso model, the best tuning parameter for the minimum MSE rule is 75.0622912 and the test error is 4.7995097×10^8. When the 1SE rule is applied, 29 predictors besides the intercept are included in the model.

LASSO by caret

```r
set.seed(2023)
lasso.caret.min <- train(xtrain, ytrain,
                   method = "glmnet",
                   tuneGrid = expand.grid(alpha = 1,
                                          lambda = exp(seq(8, 2, length=1000))),
                   trControl = ctrl1)
plot(lasso.caret.min, xTrans = log)
```




```
lasso.caret.min$bestTune
```

```
##     alpha   lambda
## 256     1 34.17627
```

```
coef(lasso.caret.min$finalModel, lasso.caret.min$bestTune$lambda)
```

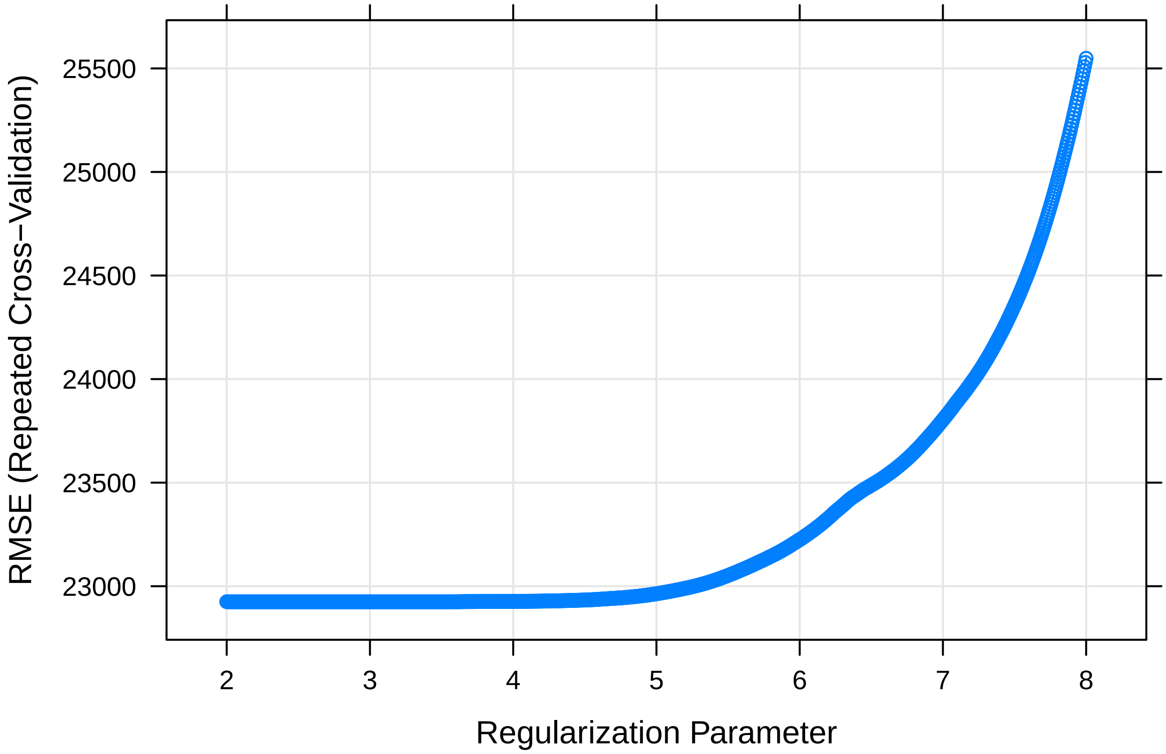
```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                        s1
## (Intercept)                 -4.891889e+06
## gr_liv_area                  6.576019e+01
## first_flr_sf                 7.854992e-01
## second_flr_sf                .
## total_bsmt_sf                3.533373e+01
## low_qual_fin_sf             -4.132375e+01
## wood_deck_sf                 1.179064e+01
## open_porch_sf                1.574960e+01
## bsmt_unf_sf                 -2.089000e+01
## mas_vnr_area                 1.072139e+01
## garage_cars                  4.139513e+03
## garage_area                  8.020923e+00
## year_built                   3.241105e+02
## tot_rms_abv_grd             -3.705645e+03
## full_bath                   -4.036604e+03
## overall_qualAverage         -4.924548e+03
## overall_qualBelow_Average   -1.260070e+04
## overall_qualExcellent        7.446458e+04
## overall_qualFair            -1.091887e+04
## overall_qualGood             1.218755e+04
## overall_qualVery_Excellent   1.337787e+05
## overall_qualVery_Good        3.793984e+04
```

```
## kitchen_qualFair           -2.556466e+04
## kitchen_qualGood           -1.785058e+04
## kitchen_qualTypical        -2.590666e+04
## fireplaces                  1.087859e+04
## fireplace_quFair           -7.738271e+03
## fireplace_quGood            .
## fireplace_quNo_Fireplace    1.978771e+03
## fireplace_quPoor           -5.709901e+03
## fireplace_quTypical        -7.015803e+03
## exter_qualFair             -3.511041e+04
## exter_qualGood             -1.674606e+04
## exter_qualTypical          -2.116559e+04
## lot_frontage                1.007469e+02
## lot_area                    6.044410e-01
## longitude                  -3.366154e+04
## latitude                    5.661433e+04
## misc_val                    8.658075e-01
## year_sold                  -5.939770e+02
```

```
pred.lasso.caret.min = predict(lasso.caret.min, s = lasso.caret.min$bestTune, newx = xtest)
mse.lasso.caret.min = mean((ytest - pred.lasso.caret.min)^2)
mse.lasso.caret.min
```

```
## [1] 479444173
```

```
ctrl2 = trainControl(method = "repeatedcv", number = 10, repeats = 5, selectionFunction = "oneSE")
set.seed(2023)
lasso.caret.1se <- train(xtrain, ytrain,
                   method = "glmnet",
                   tuneGrid = expand.grid(alpha = 1,
                                          lambda = exp(seq(8, 2, length=1000))),
                   trControl = ctrl2)
plot(lasso.caret.1se, xTrans = log)
```




```r
lasso.caret.1se$bestTune
```

```
##     alpha   lambda
## 670     1 410.7637
```

```r
coef(lasso.caret.1se$finalModel, lasso.caret.1se$bestTune$lambda)
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                       s1
## (Intercept)                -3.897085e+06
## gr_liv_area                 6.091033e+01
## first_flr_sf                9.502732e-01
## second_flr_sf               .
## total_bsmt_sf               3.629288e+01
## low_qual_fin_sf            -3.504699e+01
## wood_deck_sf                9.968877e+00
## open_porch_sf               1.195645e+01
## bsmt_unf_sf                -2.058462e+01
## mas_vnr_area                1.300620e+01
## garage_cars                 3.479595e+03
## garage_area                 9.765822e+00
## year_built                  3.149071e+02
## tot_rms_abv_grd            -2.497092e+03
## full_bath                  -1.367140e+03
## overall_qualAverage        -3.986739e+03
## overall_qualBelow_Average  -1.081091e+04
## overall_qualExcellent       8.736349e+04
## overall_qualFair           -8.718001e+03
## overall_qualGood            1.109250e+04
## overall_qualVery_Excellent  1.562134e+05
## overall_qualVery_Good       3.729801e+04
```

```
## kitchen_qualFair           -1.391060e+04
## kitchen_qualGood           -7.362904e+03
## kitchen_qualTypical        -1.618805e+04
## fireplaces                  8.135624e+03
## fireplace_quFair           -3.689784e+03
## fireplace_quGood            2.273316e+03
## fireplace_quNo_Fireplace    .
## fireplace_quPoor           -1.359037e+03
## fireplace_quTypical        -4.024181e+03
## exter_qualFair             -1.691749e+04
## exter_qualGood              .
## exter_qualTypical          -4.791414e+03
## lot_frontage                8.634683e+01
## lot_area                    5.911872e-01
## longitude                  -2.223730e+04
## latitude                    3.731517e+04
## misc_val                    2.964033e-01
## year_sold                  -1.583733e+02
```

```
pred.lasso.caret.1se = predict(lasso.caret.1se, s = lasso.caret.1se$bestTune, newx = xtest)
mse.lasso.caret.1se = mean((ytest - pred.lasso.caret.1se)^2)
mse.lasso.caret.1se
```

```
## [1] 496114286
```

We can also fit Lasso model using the `caret` package. The best tuning parameter for the minumum MSE rule is lambda = 410.7636622 and the test error is rmse.lasso.caret.min`. If we want to use the 1se rule, we can define a new resampling method`ctrl2`that specifies`selectionFunction = "oneSE"`. With the 1se rule, there are 36 predictors included in the model.

elastic net

```
set.seed(2023)
enet.fit <- train(xtrain, ytrain,
                  method = "glmnet",
                  tuneGrid = expand.grid(alpha = seq(0, 1, length = 21),
                                         lambda = exp(seq(7,-1, length = 200))),
                  trControl = ctrl1)
enet.fit$bestTune$alpha
```

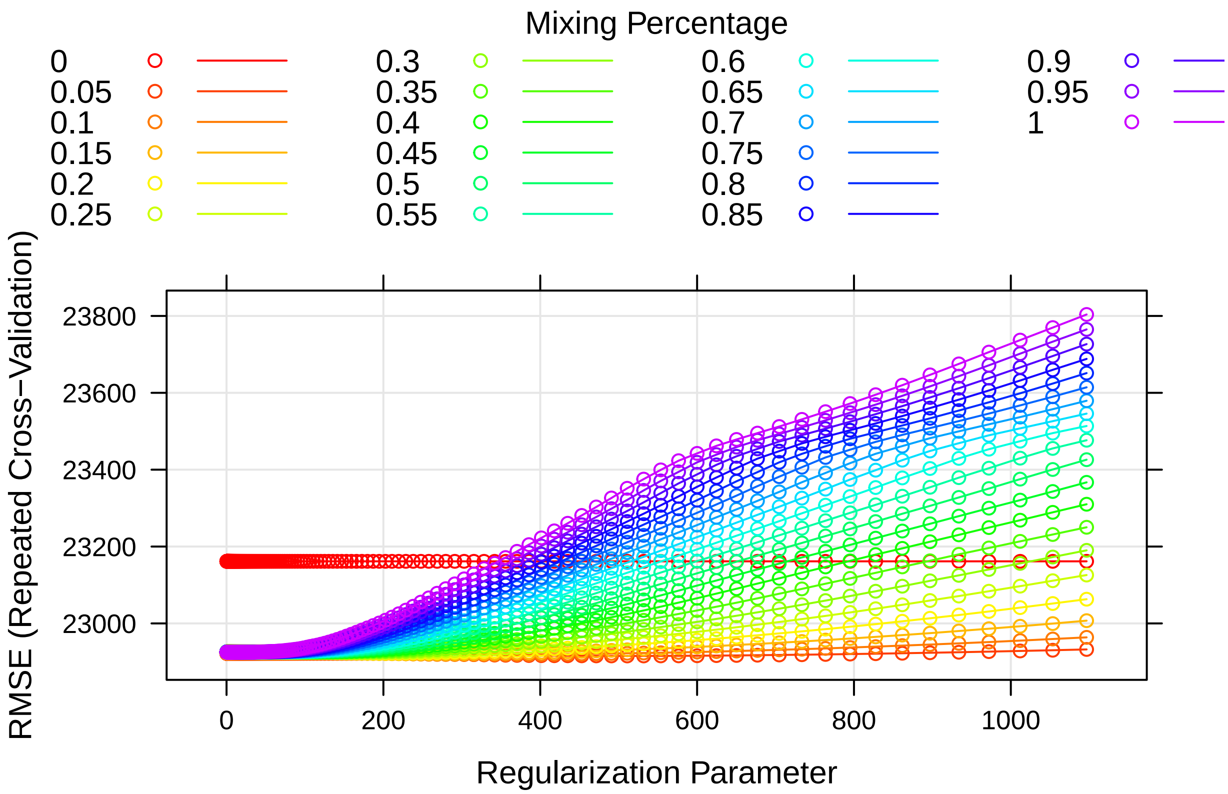
```
## [1] 0.05
```

```
enet.fit$bestTune$lambda
```

```
## [1] 531.8609
```

```
myCol= rainbow(25)
myPar = list(superpose.symbol = list(col = myCol),
                    superpose.line = list(col = myCol))

plot(enet.fit, par.settings = myPar)
```




```r
pred.enet = predict(enet.fit, s = enet.fit$bestTune, newx = xtest)
mse.enet = mean((ytest - pred.enet)^2)
mse.enet
```

```
## [1] 480063606
```

When fitting a elastic net model, the selected tuning parameter for the minumum MSE rule is alpha = 0.05 and lambda = 531.8608577. It is possible to apply the 1SE rule to select the tuning parameters by using the resampling method of `ctrl2` where `selectionFunction = "oneSE"` is specified.

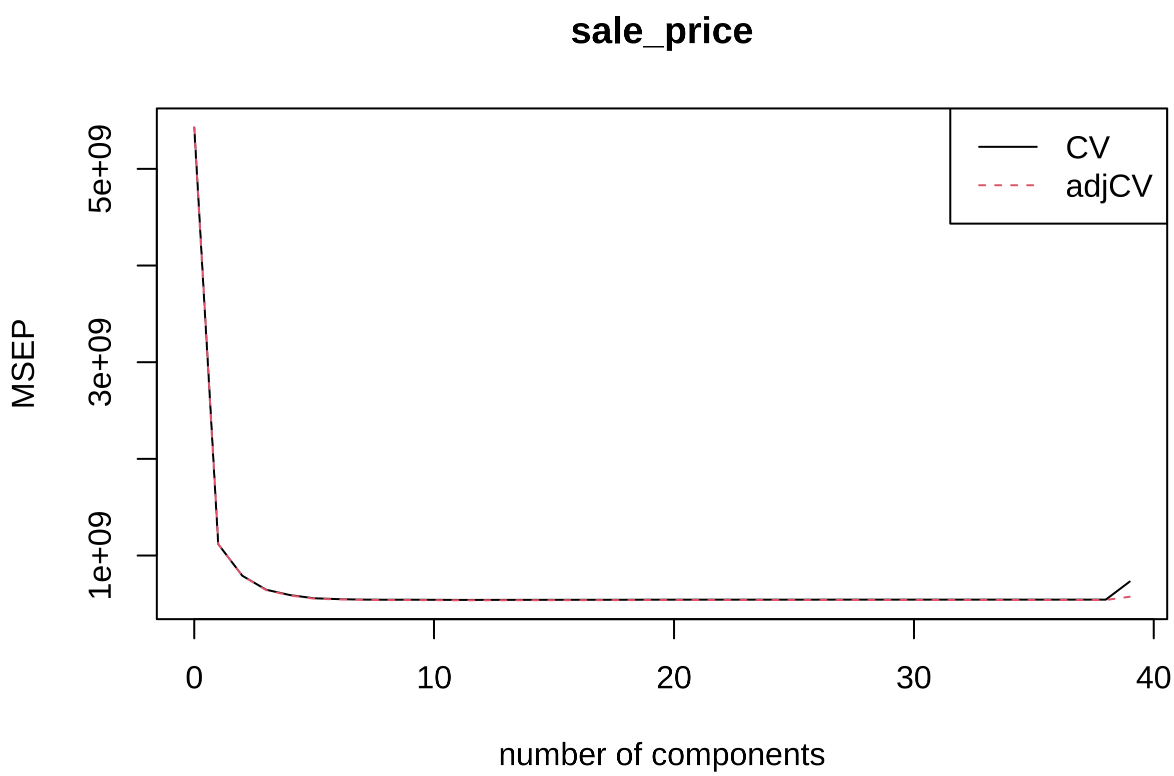
partial least squares by `pls`

```r
set.seed(2023)
pls.fit = plsr(sale_price~.,
               data = train,
               scale = TRUE,
               validation = "CV")
summary(pls.fit)
```

```
## Data:    X dimension: 1440 39
##  Y dimension: 1440 1
## Fit method: kernelpls
## Number of components considered: 39
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           73685    33432    28131    25418    24296    23613    23430
## adjCV        73685    33427    28087    25329    24210    23534    23358
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
```

```
## CV        23324    23291    23300     23275     23252     23256     23261
## adjCV     23254    23222    23228     23203     23181     23183     23188
##         14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## CV         23270     23275     23280     23283     23287     23304     23304
## adjCV      23196     23201     23206     23208     23213     23228     23228
##         21 comps  22 comps  23 comps  24 comps  25 comps  26 comps  27 comps
## CV         23309     23310     23310     23311     23312     23312     23315
## adjCV      23233     23234     23233     23234     23235     23235     23237
##         28 comps  29 comps  30 comps  31 comps  32 comps  33 comps  34 comps
## CV         23315     23315     23315     23316     23316     23316     23316
## adjCV      23238     23238     23238     23238     23238     23238     23238
##         35 comps  36 comps  37 comps  38 comps  39 comps
## CV         23316     23316     23316     23316     27032
## adjCV      23238     23238     23238     23238     23946
##
## TRAINING: % variance explained
##             1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X             20.02    25.93    29.67    33.59    37.01    40.03    42.49
## sale_price    79.73    86.35    89.36    90.37    90.87    90.99    91.06
##             8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X             45.53    47.97     50.15     52.01     53.69     55.35     56.86
## sale_price    91.08    91.10     91.13     91.15     91.15     91.16     91.16
##             15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## X              58.64     60.01     62.18     63.87     65.26     67.10
## sale_price     91.16     91.16     91.16     91.16     91.16     91.16
##             21 comps  22 comps  23 comps  24 comps  25 comps  26 comps
## X              68.44     70.12     71.72     73.35     75.20     77.27
## sale_price     91.16     91.16     91.16     91.16     91.16     91.16
##             27 comps  28 comps  29 comps  30 comps  31 comps  32 comps
## X              78.97     80.10     81.83     83.55     84.39     86.34
## sale_price     91.16     91.16     91.16     91.16     91.16     91.16
##             33 comps  34 comps  35 comps  36 comps  37 comps  38 comps
## X              88.63     90.79     92.79     95.45     97.49    100.00
## sale_price     91.16     91.16     91.16     91.16     91.16     91.16
##             39 comps
## X             100.24
## sale_price     91.14
```

```
validationplot(pls.fit, val.type="MSEP", legendpos = "topright")
```




```
cv.mse = RMSEP(pls.fit)
ncomp.cv = which.min(cv.mse$val[1,,])-1
ncomp.cv
```

```
## 11 comps
##       11
```

```
pred.pls = predict(pls.fit, newdata = xtest, ncomp = ncomp.cv)
mse.pls = mean((ytest - pred.pls)^2)
mse.pls
```

```
## [1] 480106167
```

When fitting a partial least squares model using `pls`, the test error is 4.8010617×10^{8}. There are 11 components included in the model.

partial least squares by `caret`

```
set.seed(2023)
pls.fit.caret = train(xtrain, ytrain,
                method = "pls",
                tuneGrid = data.frame(ncomp = 1:39),
                trControl = ctrl1,
                preProcess = c("center", "scale"))
pls.fit.caret$bestTune
```

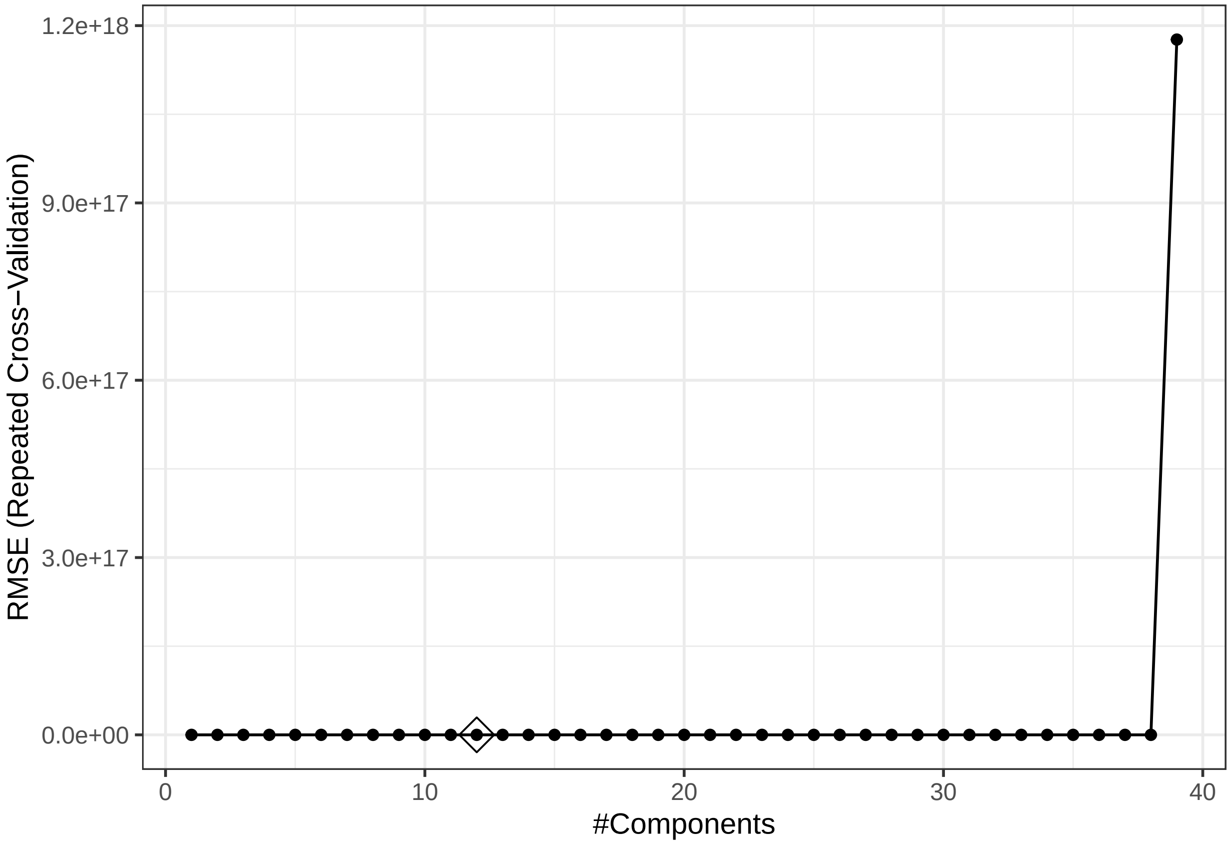
```
##    ncomp
## 12    12
```

```
pred.pls.caret = predict(pls.fit.caret, newdata = xtest)
mse.pls.caret = mean((ytest - pred.pls.caret)^2)
```

```
mse.pls.caret
```

```
## [1] 479655935
```

```
ggplot(pls.fit.caret, highlight = TRUE) + theme_bw()
```



We can also use `caret` package to fit a partial least sqaures model. We see that the number of components included in the model is different from what we got using `pls`.

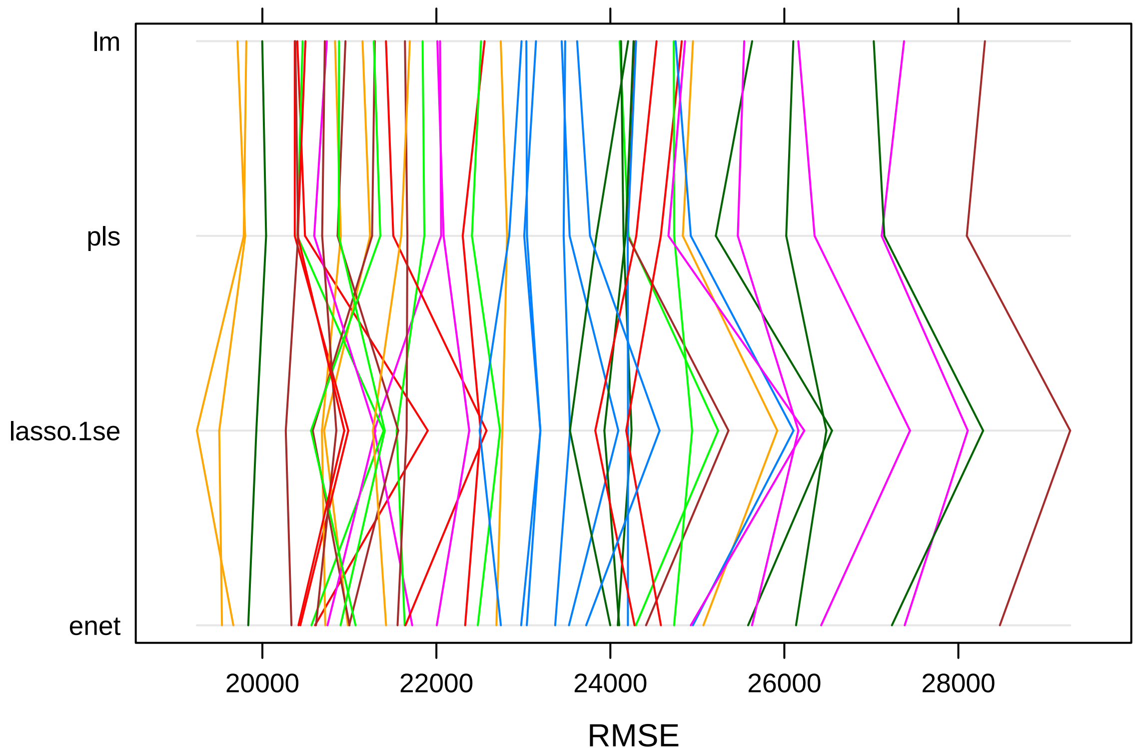
Comparing methods

```
resamp = resamples(list(lm = lm.fit, lasso.1se = lasso.caret.1se, enet = enet.fit, pls = pls.fit.caret))
summary(resamp)
```

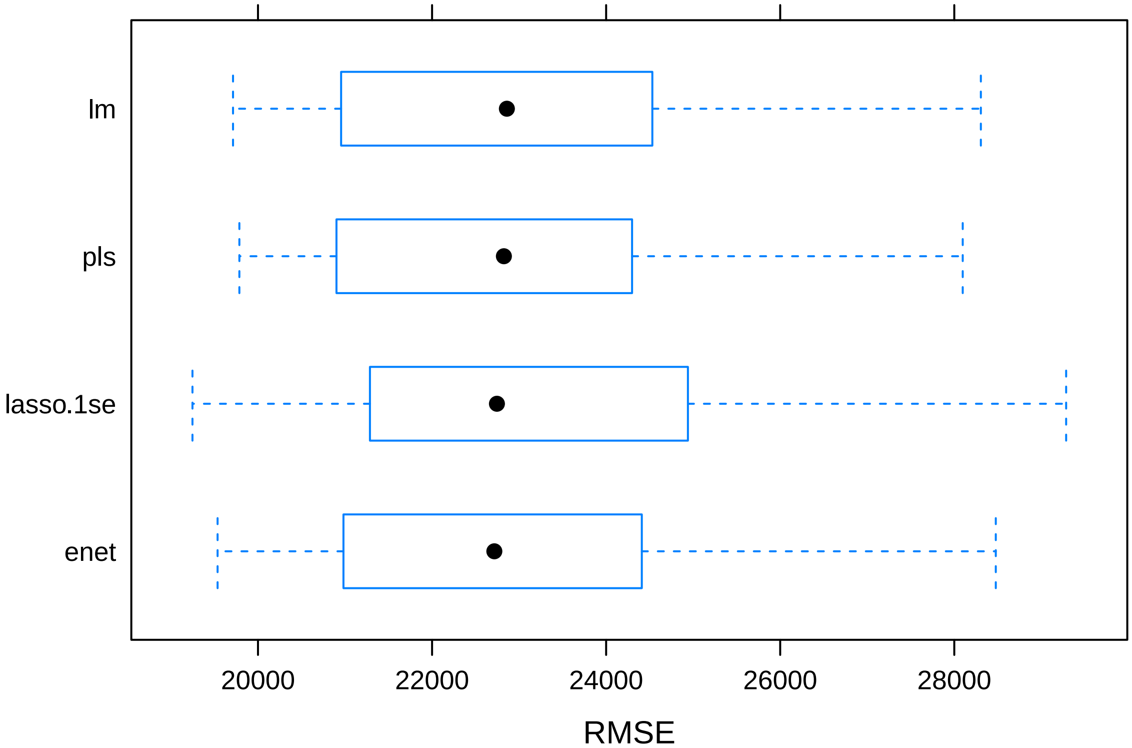
```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: lm, lasso.1se, enet, pls
## Number of resamples: 50
##
## MAE
##                Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## lm        13800.79 15933.82 16677.79 16706.93 17552.24 19577.64    0
## lasso.1se 13787.84 15756.68 16625.07 16650.15 17584.23 19299.33    0
## enet      13632.00 15849.10 16553.97 16627.25 17541.75 19493.36    0
## pls       13773.94 16021.17 16627.52 16703.52 17567.40 19558.61    0
##
```

```
## RMSE
##                 Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## lm        19713.17 21004.01 22859.59 22954.19 24472.24 28305.61    0
## lasso.1se 19247.12 21289.48 22745.25 23232.86 24846.25 29285.41    0
## enet      19535.62 20985.92 22715.59 22915.37 24380.82 28477.10    0
## pls       19785.94 20985.56 22825.16 22913.06 24276.12 28097.36    0
##
## Rsquared
##                  Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lm        0.8659887 0.8911576 0.9014198 0.9035345 0.9181673 0.9431217    0
## lasso.1se 0.8530071 0.8922338 0.9019239 0.9018264 0.9146479 0.9399871    0
## enet      0.8636811 0.8928940 0.9018478 0.9039639 0.9176192 0.9430478    0
## pls       0.8645338 0.8922100 0.9025602 0.9037972 0.9180425 0.9433645    0
```

```
parallelplot(resamp, metric = "RMSE")
```




```
bwplot(resamp, metric = "RMSE")
```



By comparing across all models built, I would select elastic net for predicting the response because it has the smallest RMSE and MSE values. The adjusted R squares is also the second highest in the four models.