p8106_hw1_wq2160

Wenshan Qu (wq2160)

2/12/2022

Contents

Import csv files.

```
train_data = read_csv("./data/housing_training.csv")
test_data = read_csv("./data/housing_test.csv")
```

Set train data and test data, create predictor matrix and changing categorial variables into dummy variables.

```
## Train Data
train_data = na.omit(train_data)
## Matrix of Predictors
x_train = model.matrix(Sale_Price ~ ., train_data)[ ,-1]
## Vector of Response
y_train = train_data$Sale_Price

## Test Data
test_data = na.omit(test_data)
## Matrix of Predictors
x_test = model.matrix(Sale_Price ~ ., test_data)[ ,-1]
## Vector of Response
y_test = test_data$Sale_Price
```

Least Squares

Fit a linear model using least squares on the training data. Is there any potential disadvantage of this model?

Cross Validation

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

## Fit Least Square Model "ls_fit"
set.seed(33)
ls_fit = train(x = x_train, y = y_train,
               preProcess = c("center", "scale"),
               method = "lm",
               trControl = ctrl1)
```

Coefficients of Final LS Model

```
coef(ls_fit$finalModel)
```

```
##                (Intercept)                Gr_Liv_Area
##                177568.5021                 11918.0894
##               First_Flr_SF              Second_Flr_SF
##                 15645.5803                 17658.0873
##              Total_Bsmt_SF            Low_Qual_Fin_SF
##                 14564.1637                         NA
##               Wood_Deck_SF              Open_Porch_SF
##                  1609.2349                  1027.1662
##                Bsmt_Unf_SF               Mas_Vnr_Area
##                 -8661.3248                  1756.9261
##                Garage_Cars                Garage_Area
##                  3056.3138                  1566.0651
```

```
##                  Year_Built              TotRms_AbvGrd
##                   9546.4284                 -5883.7090
##                   Full_Bath        Overall_QualAverage
##                  -2344.0376                 -2287.2496
##   Overall_QualBelow_Average      Overall_QualExcellent
##                  -3314.2960                 12221.9256
##            Overall_QualFair           Overall_QualGood
##                  -1367.6982                  4994.1609
## Overall_QualVery_Excellent      Overall_QualVery_Good
##                  12335.9659                 11604.5603
##            Kitchen_QualFair           Kitchen_QualGood
##                  -3410.1429                 -9158.7007
##         Kitchen_QualTypical                 Fireplaces
##                 -13332.5419                  7400.0438
##            Fireplace_QuFair           Fireplace_QuGood
##                  -1198.9860                   258.9141
##    Fireplace_QuNo_Fireplace           Fireplace_QuPoor
##                   1697.3546                  -677.4099
##         Fireplace_QuTypical             Exter_QualFair
##                  -2624.3478                 -3914.3802
##              Exter_QualGood          Exter_QualTypical
##                  -9346.0231                -11719.7870
##                Lot_Frontage                   Lot_Area
##                   3327.9966                  5015.8829
##                   Longitude                   Latitude
##                   -923.2576                  1071.8787
##                    Misc_Val                  Year_Sold
##                    541.4456                  -831.9938
```

Report Test Error

```r
## Make Prediction on Test Data
predy2_lm = predict(ls_fit, x_test)

## Test MSE
lm_test_mse = mean((y_test - predy2_lm)^2)
lm_test_mse
```

```
## [1] 447287652
```

```r
## Test RMSE
lm_test_rmse = RMSE(predy2_lm, y_test)
lm_test_rmse
```

```
## [1] 21149.18
```

Potential Disadvantages 1) OLS could be very **sensitive to outliers**; 2) Real-world data tend to be more complicated and **non-linear**; 3) May **include too many features**, and LS method may particularly prone to this problem, for as soon as the number of features used exceeds the number of training data points, the least squares solution will not be unique, and hence the least squares algorithm will fail; 4) A subset of the independent variables significantly correlated to each other (**collinearity**) may lead to poor performance of LS model (variance will be inflated).

Lasso

Fit a lasso model on the training data and report the test error. When the 1SE rule is applied, how many predictors are included in the model?

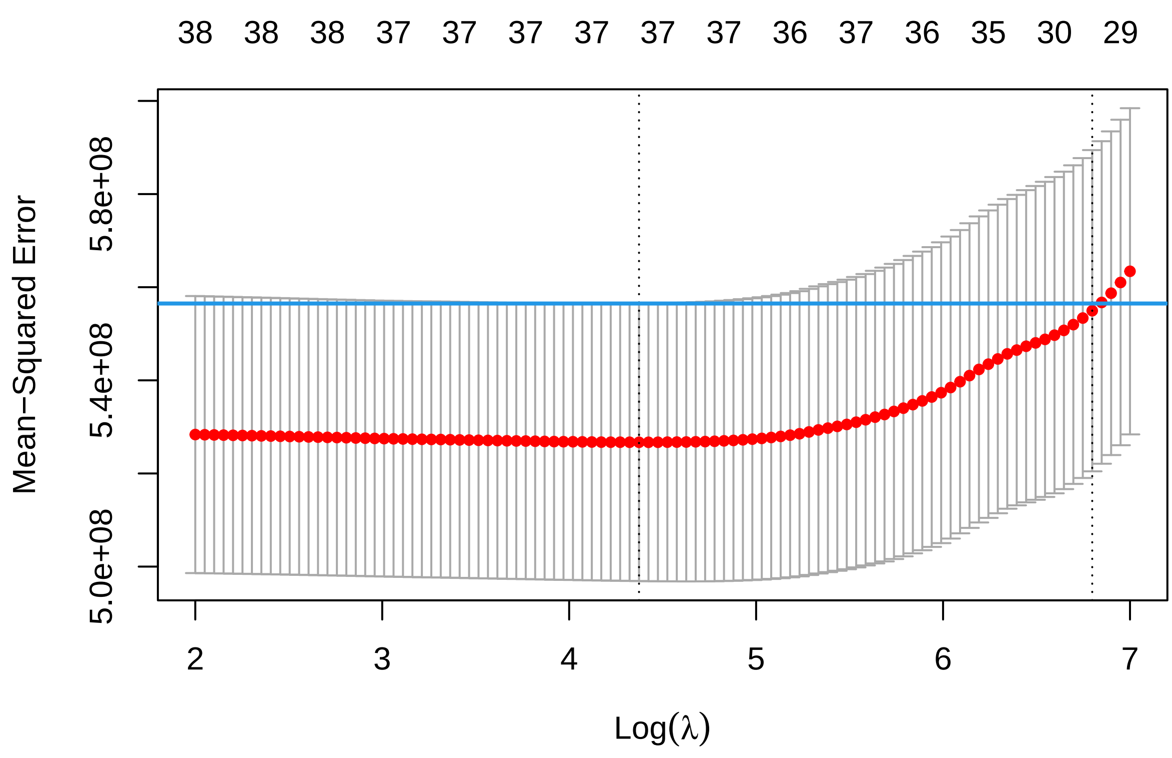
*Note: Here we use two methods **`glmnet`** and **`caret`** to fit the Lasso model. Model fitted by **`caret`** will be used in future model comparison.*

Using `glmnet`

Cross Validation

```
## Fit a Lasso Model "cv.lasso"
set.seed(33)
lasso_fit = cv.glmnet(x = x_train, y = y_train,
                      alpha = 1,
                      lambda = exp(seq(7, 2, length = 100)))

plot(lasso_fit)
abline(h = (lasso_fit$cvm + lasso_fit$cvsd)[which.min(lasso_fit$cvm)], col = 4, lwd = 2)
```




```
## Lambda Choices
## min CV MSE
lasso_fit$lambda.min
```

```
## [1] 79.3396
```

```r
## the 1SE rule (our choice in this case)
lasso_fit$lambda.1se
```

```
## [1] 896.0353
```

Coefficients of the final model

```r
## Coefficients of the Final Lasso Model (with lambda 1SE)
lasso_coeff = predict(lasso_fit, s = lasso_fit$lambda.1se, type = "coefficients")
lasso_coeff
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                     s1
## (Intercept)                 -1.921277e+06
## Gr_Liv_Area                  5.605862e+01
## First_Flr_SF                 1.146217e+00
## Second_Flr_SF                .
## Total_Bsmt_SF                3.676964e+01
## Low_Qual_Fin_SF             -2.474689e+01
## Wood_Deck_SF                 8.155223e+00
## Open_Porch_SF                7.409786e+00
## Bsmt_Unf_SF                 -1.919227e+01
## Mas_Vnr_Area                 1.428873e+01
## Garage_Cars                  3.100922e+03
## Garage_Area                  1.144968e+01
## Year_Built                   3.155287e+02
## TotRms_AbvGrd               -1.014114e+03
## Full_Bath                    .
## Overall_QualAverage         -2.958033e+03
## Overall_QualBelow_Average   -8.825529e+03
## Overall_QualExcellent        8.966701e+04
## Overall_QualFair            -5.780970e+03
## Overall_QualGood             9.577446e+03
## Overall_QualVery_Excellent   1.592569e+05
## Overall_QualVery_Good        3.576326e+04
## Kitchen_QualFair            -4.781858e+03
## Kitchen_QualGood             .
## Kitchen_QualTypical         -9.702107e+03
## Fireplaces                   6.566264e+03
## Fireplace_QuFair             .
## Fireplace_QuGood             4.584952e+03
## Fireplace_QuNo_Fireplace     .
## Fireplace_QuPoor             .
## Fireplace_QuTypical          .
## Exter_QualFair              -1.416549e+04
## Exter_QualGood               .
## Exter_QualTypical           -5.248371e+03
## Lot_Frontage                 6.719991e+01
## Lot_Area                     5.513857e-01
## Longitude                   -8.344809e+03
## Latitude                     1.339736e+04
## Misc_Val                     .
## Year_Sold                    .
```

```r
## Number of non-zero coefficients
num_lasso_coeff = length(which(lasso_coeff != 0))
num_lasso_coeff
```

```
## [1] 30
```

Here for the final lasso model with 1SE rule lambda, we got 30' predictors in the model.

Report Test Error

```r
## Make Prediction on Test Data
predy2_lasso = predict(lasso_fit, newx = x_test,
                       s = "lambda.1se", type = "response")

## Test MSE
lasso_test_mse = mean((y_test - predy2_lasso)^2)
lasso_test_mse
```

```
## [1] 421854782
```

```r
## Test RMSE
lasso_test_rmse = RMSE(predy2_lasso, y_test)
lasso_test_rmse
```

```
## [1] 20539.1
```

Comments: The test error is RMSE = 20539.1, and based on 1SE rule, 30 predictors will be included in the final model.

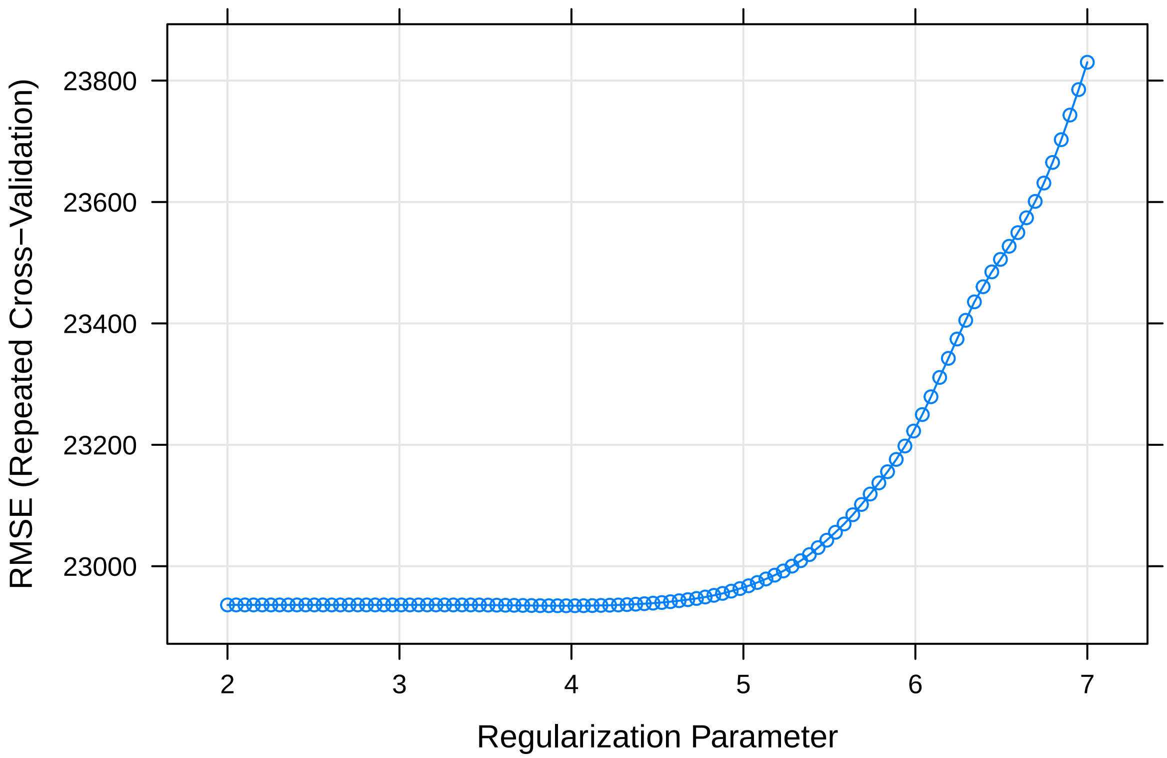
Using `caret`

To compare models (i.e., using `resample()` function later in this assignment), we have to build a lasso model using `caret`.

```r
ctrl2 = trainControl(method = "cv", selectionFunction = "oneSE")

## Build alternative Lasso Model with `caret`
set.seed(33)
lasso_caret = train(x = x_train, y = y_train,
                    method = "glmnet",
                    tuneGrid = expand.grid(alpha = 1,
                                           lambda = exp(seq(7, 2, length=100))),
                    trControl = ctrl1)

## Check Best Tune
plot(lasso_caret, xTrans = log)
```




```
lasso_caret$bestTune
```

```
##    alpha   lambda
## 40     1 52.96848
```

We could use `ctrl2 = trainControl(method = "cv", selectionFunction = "oneSE")` and replace it with "ctrl1" in `lasso_caret` to extract best tune based on 1SE rule. While to satisfy the requirement of `resample()` function, we could not make this change here.

Elastic net

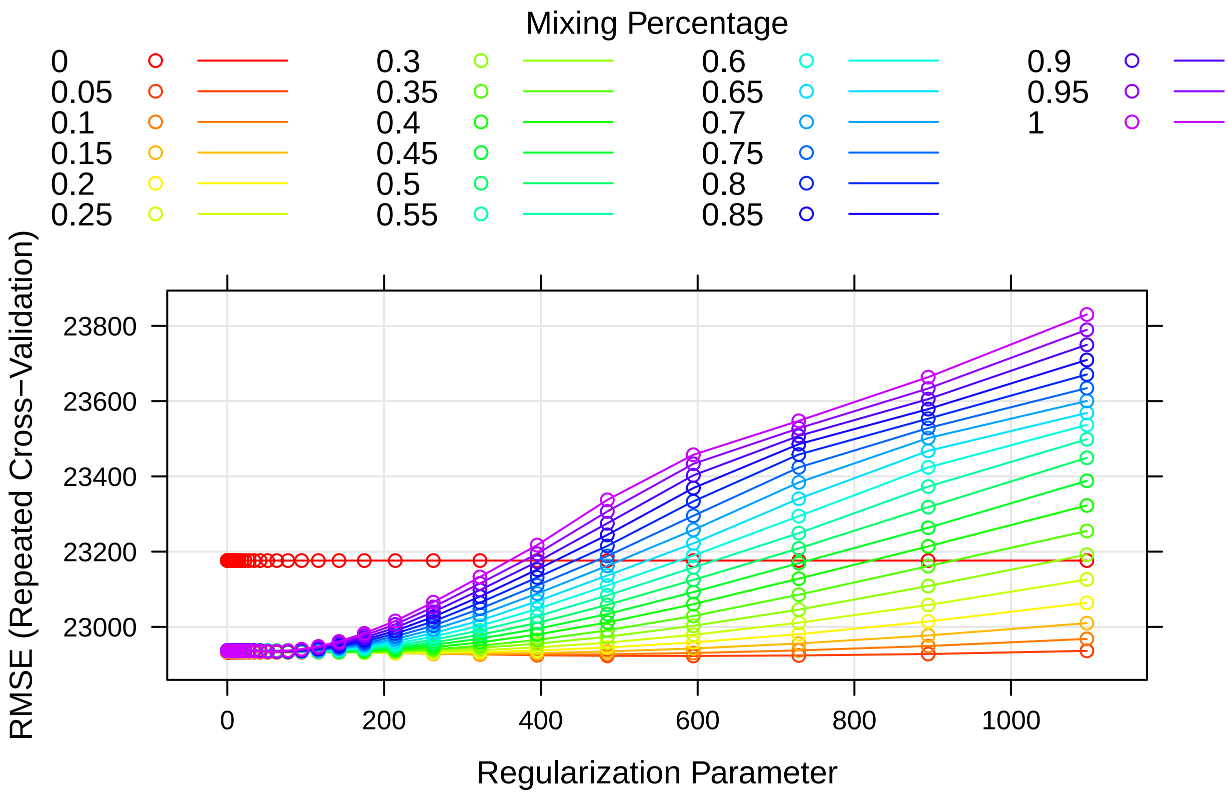
Fit an elastic net model on the training data. Report the selected tuning parameters and the test error.

Cross Validation

```
## Fit Elastic Model "enet.fit"
set.seed(33)
enet_fit = train(x = x_train, y = y_train,
                 method = "glmnet",
                 tuneGrid = expand.grid(alpha = seq(0, 1, length = 21),
                                        lambda = exp(seq(7, -3, length = 50))),
                 trControl = ctrl1)

## Plot Tune vs. RMSE (Rainbow plot)
myCol = rainbow(25)
myPar = list(superpose.symbol = list(col = myCol),
                  superpose.line = list(col = myCol))

plot(enet_fit, par.settings = myPar)
```




```
## Select Tuning Parameter
enet_fit$bestTune
```

```
##    alpha   lambda
## 97  0.05 594.5204
```

Coefficients of Final Model

```
coef(enet_fit$finalModel, enet_fit$bestTune$lambda)
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                        s1
## (Intercept)                 -5.114688e+06
## Gr_Liv_Area                  3.875517e+01
## First_Flr_SF                 2.669008e+01
## Second_Flr_SF                2.543468e+01
## Total_Bsmt_SF                3.493715e+01
## Low_Qual_Fin_SF             -1.586533e+01
## Wood_Deck_SF                 1.233184e+01
## Open_Porch_SF                1.689458e+01
## Bsmt_Unf_SF                 -2.072286e+01
## Mas_Vnr_Area                 1.167773e+01
## Garage_Cars                  4.044788e+03
## Garage_Area                  8.911280e+00
## Year_Built                   3.190772e+02
## TotRms_AbvGrd               -3.433322e+03
## Full_Bath                   -3.681927e+03
```

```
## Overall_QualAverage        -5.117537e+03
## Overall_QualBelow_Average  -1.270512e+04
## Overall_QualExcellent       7.585994e+04
## Overall_QualFair           -1.147718e+04
## Overall_QualGood            1.197478e+04
## Overall_QualVery_Excellent  1.364543e+05
## Overall_QualVery_Good       3.764655e+04
## Kitchen_QualFair           -2.363878e+04
## Kitchen_QualGood           -1.606293e+04
## Kitchen_QualTypical        -2.411455e+04
## Fireplaces                  1.081881e+04
## Fireplace_QuFair           -7.859719e+03
## Fireplace_QuGood            1.482400e+02
## Fireplace_QuNo_Fireplace    1.799177e+03
## Fireplace_QuPoor           -5.805816e+03
## Fireplace_QuTypical        -6.963002e+03
## Exter_QualFair             -3.289911e+04
## Exter_QualGood             -1.449650e+04
## Exter_QualTypical          -1.909701e+04
## Lot_Frontage                1.001255e+02
## Lot_Area                    6.031626e-01
## Longitude                  -3.516146e+04
## Latitude                    5.773725e+04
## Misc_Val                    8.673662e-01
## Year_Sold                  -5.736174e+02
```

Report Test Error

```r
## Make Prediction on Test Data
predy2_enet = predict(enet_fit, newdata = x_test)

## Test MSE
enet_test_mse = mean((y_test - predy2_enet)^2)
enet_test_mse
```

```
## [1] 438465868
```

```r
## Test RMSE
enet_test_rmse = RMSE(predy2_enet, y_test)
enet_test_rmse
```

```
## [1] 20939.58
```

Comments: Selected tune parameter is alpha = 0.05 and lambda = 594.52, and the test error is RMSE = 20939.58.

Partial Least Squares (PLS)

Fit a partial least squares model on the training data and report the test error. How many components are included in your model?

Using plsr

Cross Validation

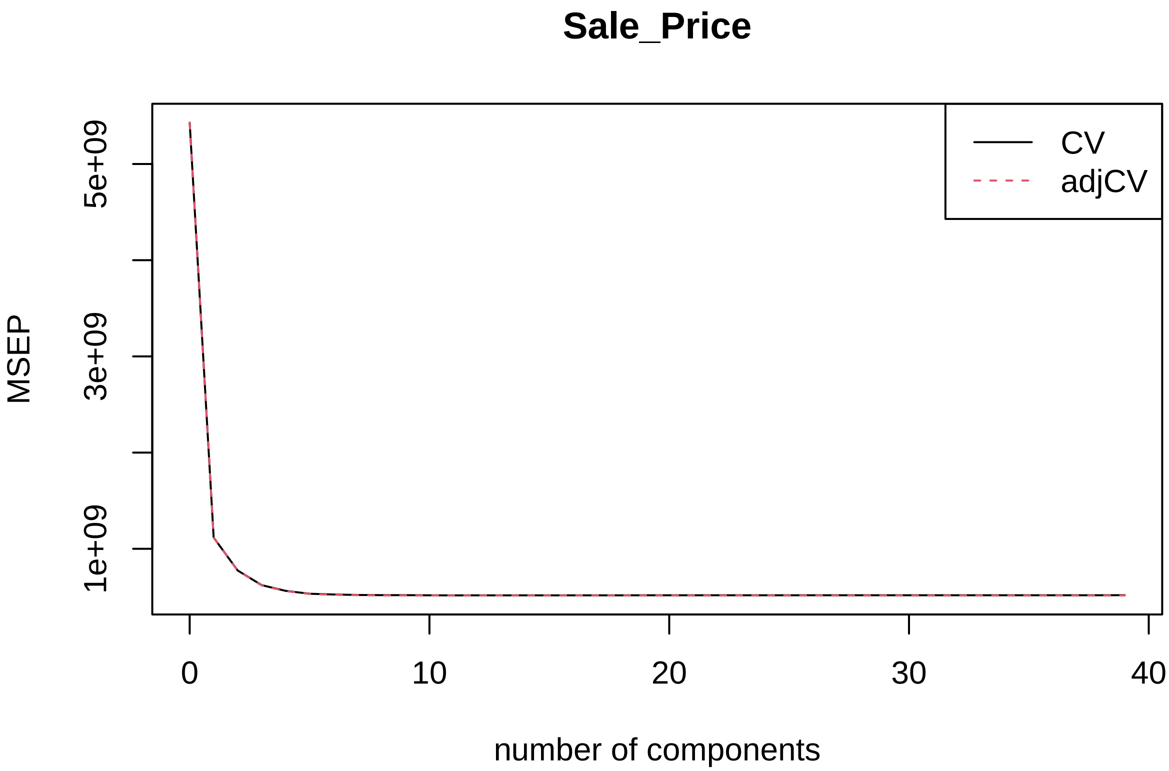
```
## Fit PLS Model "pls.fit"
set.seed(33)
pls_fit = plsr(Sale_Price ~ .,
               data = train_data,
               scale = TRUE,
               validation = "CV")

## Summary and Visualization
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    33422    27836    24950    23724    23079    22916
## adjCV        73685    33415    27806    24885    23671    23030    22871
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       22801    22766    22760     22732     22713     22725     22716
## adjCV    22760    22726    22718     22690     22671     22681     22673
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## CV        22724     22719     22724     22723     22726     22734     22736
## adjCV     22680     22676     22679     22679     22682     22689     22691
##        21 comps  22 comps  23 comps  24 comps  25 comps  26 comps  27 comps
## CV        22738     22739     22737     22738     22738     22739     22741
## adjCV     22693     22693     22692     22693     22693     22694     22695
##        28 comps  29 comps  30 comps  31 comps  32 comps  33 comps  34 comps
## CV        22742     22742     22742     22742     22742     22742     22742
## adjCV     22696     22696     22696     22696     22696     22696     22696
##        35 comps  36 comps  37 comps  38 comps  39 comps
## CV        22742     22742     22742     22742     22763
## adjCV     22696     22696     22696     22696     22702
##
## 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              100.67
## Sale_Price      91.16
```

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



Find Best Number of Components

```
cv_mse = RMSEP(pls_fit)
ncomp_cv = which.min(cv_mse$val[1,,])-1
ncomp_cv
```

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

Thus there are 11 components in the final pls model.

Report Test Error

```
## Make Prediction based on Test Data
predy2_pls = predict(pls_fit, newdata = x_test, ncomp = ncomp_cv)

## Test MSE
pls_test_mse = mean((y_test - predy2_pls)^2)
pls_test_mse
```

```
## [1] 451276530
```

```r
## Test RMSE
pls_test_rmse = RMSE(predy2_pls, y_test)
pls_test_rmse
```

```
## [1] 21243.27
```

Comments: Based on `plsr()`, 11 components are included in my model, and test error is RMSE = 21243.27.

Using `caret`

```r
## Build up a PLS Model with `caret`
set.seed(33)
pls_caret = train(x = x_train, y = y_train,
                  method = "pls",
                  tuneGrid  = data.frame(ncomp = 1:39),
                  trControl = ctrl1,
                  preProcess = c("center", "scale"))
pls_caret$bestTune
```

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

I noticed that using `caret` will lead to the result that the best number of components is 12 (instead of 11 components derived by `plsr`), while I believe this should be attributed to the underlying arithmetic difference between these two packages, combined with the fact that the RMSE of 11 and 12 components model are quite close, the different component result is reasonable.

Compare Models

Based on the mean and median of RMSE, I prefer `elastic net` model for prediction, since it has the smallest RMSE among 4 models.

```r
resamp = resamples(list(ls = ls_fit, lasso = lasso_caret, enet = enet_fit, pls = pls_caret))
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: ls, lasso, enet, pls
## Number of resamples: 50
##
## MAE
##           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## ls    14526.03 15871.73 16614.28 16700.18 17327.87 19321.64    0
## lasso 14559.04 15836.90 16543.30 16653.86 17230.78 19330.02    0
## enet  14588.48 15835.87 16446.10 16614.92 17159.71 19276.66    0
```

```
## pls   14547.29 15895.21 16551.79 16706.47 17292.83 19287.19    0
##
## RMSE
##           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## ls    19819.09 21341.55 22881.73 22965.46 24084.97 27436.71    0
## lasso 19706.42 21323.48 22851.64 22934.87 24168.38 27460.56    0
## enet  19552.99 21296.56 22761.55 22922.74 24180.36 27497.72    0
## pls   19766.03 21354.93 22896.74 22924.02 24114.27 27332.59    0
##
## Rsquared
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ls    0.8532414 0.8913140 0.9046388 0.9029611 0.9150031 0.9375877    0
## lasso 0.8536791 0.8903773 0.9050387 0.9031858 0.9159216 0.9372798    0
## enet  0.8541665 0.8905552 0.9048659 0.9033006 0.9159051 0.9370483    0
## pls   0.8538439 0.8904198 0.9048683 0.9032639 0.9158148 0.9373001    0
```

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