Evaluation Function Illustration

Robert Schmidt

Overview and basic modeling

The following example illustrates the utility of the evaluation functions contained in `eval_functions.R`. I will employ the *Hitters* dataset from the `ISLR` library.

```
source("eval_functions.R")
library(ISLR)

df = Hitters
str(df)
```

```
## '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 ...
```

```
N = nrow(df); p = ncol(df) - 1
print(paste0("There are ", N, " rows and ", p, " predictors in the Hitters dataset."))
```

```
## [1] "There are 322 rows and 19 predictors in the Hitters dataset."
```

Correlation, VIF, and significance

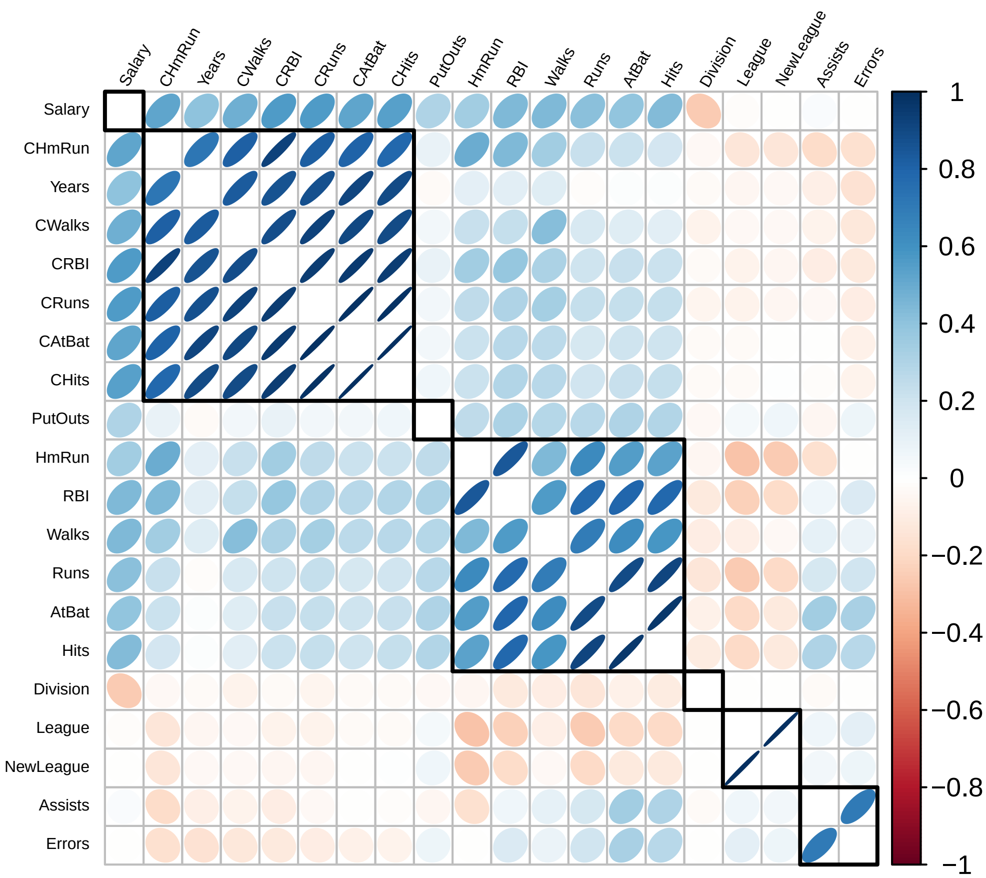
We see that the *Hitters* dataset has a variety of predictor types, including a few factor variables. Let us employ the `find_corr` function to see which predictors are the most highly correlated, accounting for the different predictor types.

```
find_corr(df, method.input = "hetcor") %>% head()
```

```
##     predictor_1 predictor_2 correlation
## 168      CAtBat       CHits   0.9950528
## 394      League   NewLeague   0.9930396
## 209       CHits       CRuns   0.9845438
## 208      CAtBat       CRuns   0.9827469
## 21        AtBat        Hits   0.9639432
## 228      CAtBat        CRBI   0.9507314
```

It is evident that a number of the predictors are highly collinear.
We can also visualize this using `make_corrPlot`.

```
make_corrPlot(df, col_labels = TRUE, var_clusters = 7)
```



Dodging this issue of collinearity for now, I will run a basic regression of Salary against all predictors in the data frame.

```
lm.fit = lm(Salary ~ ., data = df)
```

What are the most significant predictors at a 0.05 cutoff?

```
find_sig_vars(lm.fit, sig.cutoff = 0.05) %>% arrange(p_val)
```

```
##   var_names         coef         sd         z     p_val
## 1   PutOuts    0.2818925  0.0774406  3.640114 0.0003329
## 2     Walks    6.2312863  1.8285038  3.407861 0.0007662
## 3      Hits    7.5007675  2.3775341  3.154852 0.0018082
## 4     AtBat   -1.9798729  0.6339780 -3.122936 0.0020077
## 5 DivisionW -116.8492456 40.3669516 -2.894676 0.0041408
## 6    CWalks   -0.8115709  0.3280825 -2.473679 0.0140574
```

We see that a number of predictors are highly significant, although the significance is questionable given the degree of collinearity in the dataset. To address this, let's check the VIF of this basic linear regression.

```
find_VIF(lm.fit) %>% head()
```

```
##   predictor      GVIF
## 1     CHits 502.95429
## 2    CAtBat 251.56116
## 3     CRuns 162.52081
## 4      CRBI 131.96586
## 5    CHmRun  46.48846
## 6      Hits  30.28126
```

These VIFs are explosively large! Anything above 10 is considered problematic, and these are in the hundreds. Again skirting the actual issue of model integrity, how do the predictions of a linear model fare against the actual values? Some of the salaries have missing values - since this exercise is just meant to illustrate the utility of the evaluation functions, I will remove these from the dataset.

Model evaluation

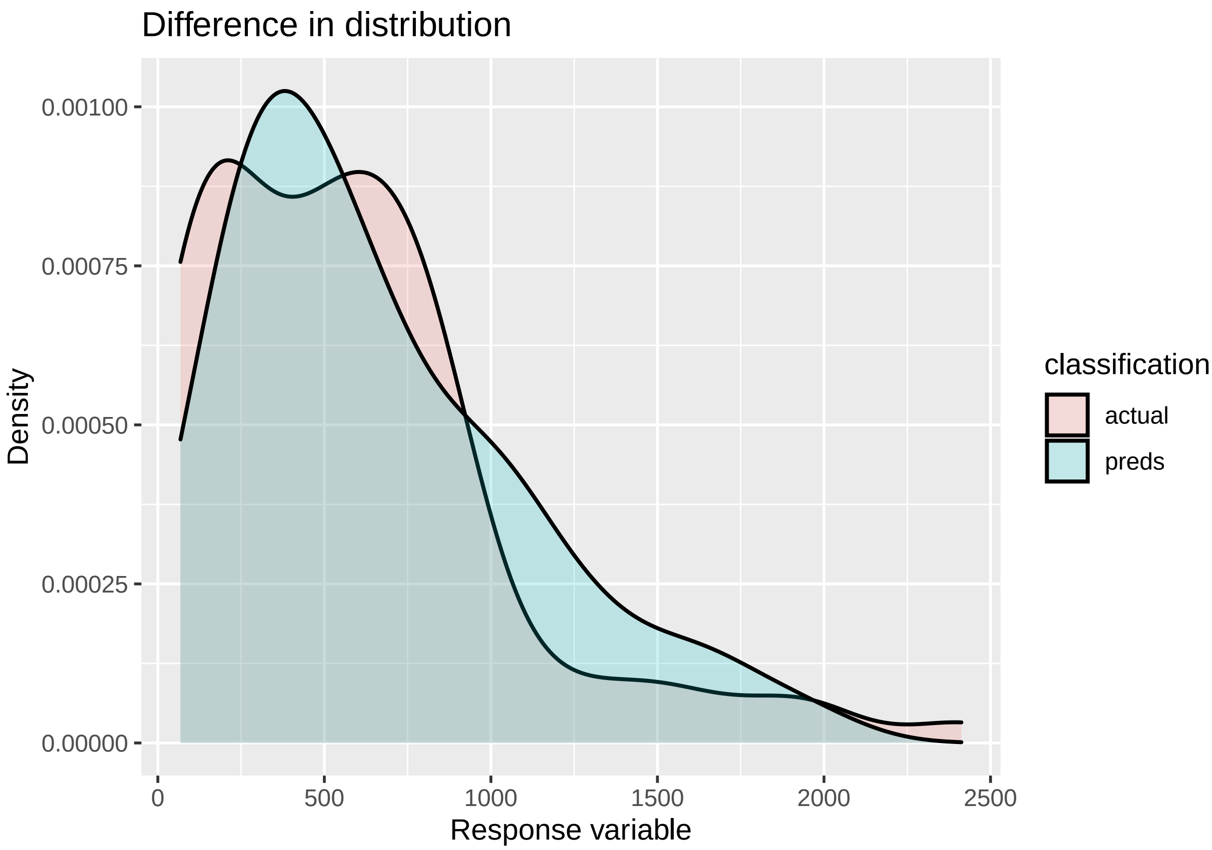
```
set.seed(1)
df.clean = df[!is.na(df$Salary), ]
N = nrow(df.clean)
data_split = sample(N, 0.7*N, replace = FALSE)
train = df.clean[data_split, ]
test = df.clean[-data_split, ]

lm.fit = lm(Salary ~ ., data = train)
preds = predict(lm.fit, test)
RMSE(preds, test$Salary)
```

```
## [1] 371.1202
```

We can also consider the distribution of the actual vs. predicted salaries:

```
dist_eval(preds, test$Salary)
```



How does the model fare at different quartiles of the actual salary?

```r
RMSE_ntile(preds, test$Salary, n_percentile = 4)
```

```
##   ntile test_RMSE
## 1     1  263.0552
## 2     2  375.1179
## 3     3  386.1865
## 4     4  441.2695
```

We see that the RMSE is worst on the highest 25% of the data.

Lasso fit

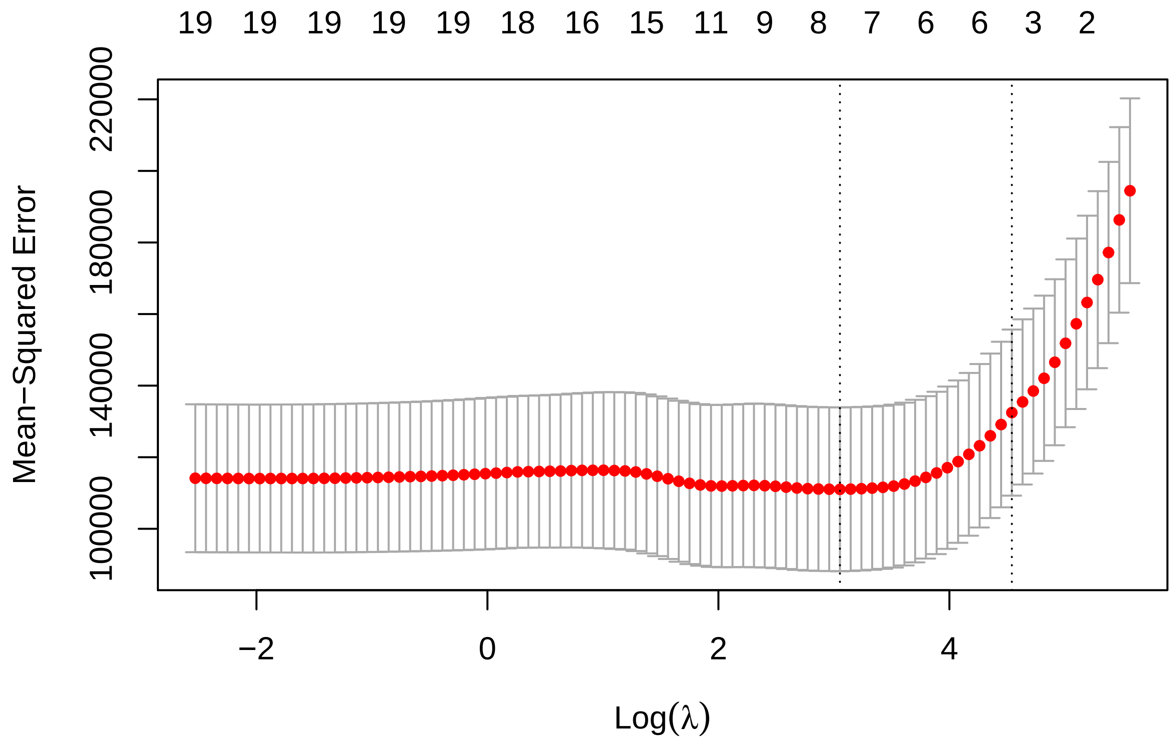
The *Hitters* dataset has a number of collinear predictors. Will a lasso regularization improve the fit?

```r
library(glmnet)

train.matrix = model.matrix(Salary ~ ., train)[, -1]
test.matrix = model.matrix(Salary ~ ., test)[, -1]
grid = 10^seq(10, -2, length = 100)

lasso.fit = glmnet(train.matrix, train$Salary, alpha = 1, lambda = grid)

# Use CV to find the best lambda
cv.out = cv.glmnet(train.matrix, train$Salary, alpha = 1)
plot(cv.out)
```




```
bestlam = cv.out$lambda.min

preds = predict(lasso.fit, s = bestlam, test.matrix)
RMSE(preds, test$Salary)
```

```
## [1] 366.6811
```

It appears that the lasso does improve the fit, if only marginally.
Of the original coefficients, which coefficients end up in the lasso fit?

```
lasso.coefs = find_lasso_coefs(lasso.fit, bestlam)
pct_lasso_coefs = round((nrow(lasso.coefs) - 1)/p, 5)*100

print(paste0("The lasso kept ", pct_lasso_coefs, " % of the original predictors."))
```

```
## [1] "The lasso kept 42.105 % of the original predictors."
```

```
lasso.coefs %>% filter(predictor != "(Intercept)")
```

```
##   predictor   coefficient
## 1 DivisionW -132.62010012
## 2   LeagueN   13.73646839
## 3     Walks    3.82004270
## 4      Hits    0.74784016
## 5      CRBI    0.46882548
## 6   PutOuts    0.22369217
## 7     CRuns    0.20531703
## 8     CHits    0.02235278
```