Northeastern University

ALY6015 Intermediate Analytics
Feature Selection Methods in R

Training and Test sets

We will use the 'mtcars' data set that is available in R for our first exercise. When modeling, it is often necessary and useful to split the data set into a training data set and a testing data set. The training data set is the data that the model will be trained on and the test set will be used to test the model. This helps us ensure that the model performs well on new data and that we are not overfitting the model. A 70/30 split where 70% of the observations are used for the training set and 30% is used for testing is quite common.

Here we use the *sample()* function to take a random sample of our data set. The *sample()* function returns a vector of indices. We can then subset 'mtcars' by the list of indices to create the training and the test set.

```
# load data
data('mtcars')
head(mtcars)

# Create Train and Test set - random sample (70/30 split)
trainIndex <- sort(sample(x = nrow(mtcars), size = nrow(mtcars) * 0.7))
sample_train <- mtcars[trainIndex,]
sample_test <- mtcars[-trainIndex,]
```

In this example, we will use the *createDataPartition()* function from the 'caret' package. This function is a bit more sophisticated than the *sample()* function, so I encourage you to look it up. A big advantage of the *createDataPartition()* function is that it can preserve the ratio or balance of the factor classes.

```
# Create Train and Test set - maintain % of event rate (70/30 split)
library(caret)
set.seed(3456)
trainIndex <- createDataPartition(mtcars$mpg, p = 0.7, list = FALSE, times = 1)
caret_train <- mtcars[ trainIndex,]
caret_test <- mtcars[-trainIndex,]
```

After running these code examples, two training and test sets will be created. Note that they may differ in size because of the way the samples are being taken.

Feature Selection Methods

As you are aware, having too many features in a model can be problematic. These methods provide a simple means of determining optimal models. While not fool proof, they can be helpful.

The first method is the forward selection method. Here we start with no predictors in the model and add the predictor that has the highest correlation with the response variable. We check to ensure that the model has actually improved by adding the variable and repeat the process if it has. The process will end when no more improvements can be made by adding variables to the model. We select the forward selection method of the *step()* function by setting the 'direction' parament equal to "forward".

In the output below, you can see that several variable combinations were attempted, with mpg ~ wt + hp being the preferred model. We can then fit that model and view the summary for verification. This task is left as an exercise for the reader.

```
# Forward selection method
step(lm(mpg ~ 1, data = mtcars), direction = 'forward', scope = ~ disp + hp + drat + wt + qsec)
model_forward <- lm(formula = mpg ~ wt + hp, data = mtcars)
summary(model_forward)
```

```
Start:  AIC=115.94
mpg ~ 1

       Df Sum of Sq     RSS     AIC
+ wt    1    847.73  278.32  73.217
+ disp  1    808.89  317.16  77.397
+ hp    1    678.37  447.67  88.427
+ drat  1    522.48  603.57  97.988
+ qsec  1    197.39  928.66 111.776
<none>              1126.05 115.943

Step:  AIC=73.22
mpg ~ wt

       Df Sum of Sq    RSS    AIC
+ hp    1    83.274 195.05 63.840
+ qsec  1    82.858 195.46 63.908
+ disp  1    31.639 246.68 71.356
<none>              278.32 73.217
+ drat  1     9.081 269.24 74.156

Step:  AIC=63.84
mpg ~ wt + hp

       Df Sum of Sq    RSS    AIC
<none>              195.05 63.840
+ drat  1   11.3659 183.68 63.919
+ qsec  1    8.9885 186.06 64.331
+ disp  1    0.0571 194.99 65.831

Call:
lm(formula = mpg ~ wt + hp, data = mtcars)
```

```
Coefficients:
(Intercept)          wt          hp
   37.22727    -3.87783    -0.03177
```

The backwards selection method starts with all predictors added to the model and removes the predictor having the least impact on the model coefficients. If this improves the model, then the process is repeated. The process stops once no improvements can be made to the model. We select the backward selection method of the *step()* function by setting the 'direction' parament equal to "backward".

```
# Backward selection method
step(lm(mpg ~ ., data = mtcars), direction = 'backward')
model_back <- step(lm(mpg ~ ., data = mtcars), direction = 'backward')
summary(model_back)
```

The output of the selection process is quite similar to that of the forward selection process, so it is not displayed to save space. Below is the summary of the model fitted with the selected variables.

```
Call:
lm(formula = mpg ~ wt + qsec + am, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4811 -1.5555 -0.7257  1.4110  4.6610

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   9.6178     6.9596   1.382 0.177915
wt           -3.9165     0.7112  -5.507 6.95e-06 ***
qsec          1.2259     0.2887   4.247 0.000216 ***
am            2.9358     1.4109   2.081 0.046716 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.459 on 28 degrees of freedom
Multiple R-squared:  0.8497,   Adjusted R-squared:  0.8336
F-statistic: 52.75 on 3 and 28 DF,  p-value: 1.21e-11
```

The stepwise selection method is considered an improvement on the forward and backward selection methods. With the forward selection method, once a variable is added to the model it cannot be removed. Similarly, with the backward selection method, once the variable is removed from the model, it cannot be added back to the model. The stepwise method starts with no variables in the model, then adds variables one at a time and tests for improvement. At each step, it also checks to see if any of the previously added variables are no longer needed in the model and removes them. It is common for the stepwise model to match the backward selection model, but this is not always true. We select the stepwise method of the *step()* function by setting the 'direction' parament equal to "both".

We can automatically save the preferred model by sending the output of the *step()* function to a variable, as demonstrated below with the model_step variable.

```
# Stepwise selection method
step(lm(mpg ~ ., data = mtcars), direction = 'both')
model_step <- step(lm(mpg ~ ., data = mtcars), direction = 'both')
summary(model_step)
```

Model Comparison

When fitting several models, it is important to know which version of the model is best. However, determining the best model is not always simple. Here are some methods that can help.

The *anova()* function may be used to test whether a significant difference between two models exists. It is important that the models being tested are nested. That means that the variables in the simpler model are also included in the improved model being tested. To interpret the ANOVA test, compare the p-value with the significance level – typically 0.05. If the p-value is larger than the significance level, then there is no significant difference between the two models so the simpler model should be selected.

In the output below, since the p-value of 0.001451 is less than 0.05 we can reject the null hypothesis and accept that adding the 'hp' variable to the fit2 model does improve the model.

```
# Compare models with ANOVA
fit1 <- lm(formula = mpg ~ wt, data = mtcars)
fit2 <- lm(formula = mpg ~ wt + hp, data = mtcars)
anova(fit1, fit2)
```

```
Analysis of Variance Table

Model 1: mpg ~ wt
Model 2: mpg ~ wt + hp
  Res.Df    RSS Df Sum of Sq      F   Pr(>F)
1     30 278.32
2     29 195.05  1    83.274 12.381 0.001451 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The Akaike Information Criterion [AIC] and the Bayesian Information Criterion [BIC] may also be used to compare two or models. Below is an example of the AIC() and BIC() functions. In the output below, model fit2 has a lower AIC value and a lower BIC value, which indicates that it is the preferred model.

```
# Compare models with AIC
AIC(fit1, fit2)

# Compare models with BIC
BIC(fit1, fit2)
```

```
> AIC(fit1, fit2)
     df      AIC
fit1  3 166.0294
fit2  4 156.6523

> BIC(fit1, fit2)
     df      BIC
fit1  3 170.4266
fit2  4 162.5153
```

The Best Subsets

The *regsubsets()* function from the 'leaps' library performs best subset selection by identifying the best model that contains a given number of predictors, where best is quantified using RSS. The syntax is the same as for *lm()*. The *summary()* command outputs the best set of variables for each model size.

In the code example below, we will use the 'Hitters' data set from the 'ISLR' library and the regsubsets() function from the leaps library to determine the best subset of 19 predictors.

```
# Using the leaps package
library(leaps)
library(ISLR)
library(dplyr)

# Review the data set
summary(Hitters)

Hitters <- Hitters %>%
  na.omit()

# Best subsets with regsubsets
best_subset = regsubsets(Salary ~ ., data = Hitters, nvmax = 19)
reg.summary <- summary(best_subset)
reg.summary
names(reg.summary)
```

The partial output below is from the *regsubsets()* function. Do not let it intimidate you. It is just showing which variables were tested in each run.

```
Subset selection object
Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19)
19 Variables  (and intercept)
           Forced in Forced out
AtBat          FALSE      FALSE
Hits           FALSE      FALSE
HmRun          FALSE      FALSE
Runs           FALSE      FALSE
RBI            FALSE      FALSE
Walks          FALSE      FALSE
Years          FALSE      FALSE
CAtBat         FALSE      FALSE
CHits          FALSE      FALSE
CHmRun         FALSE      FALSE
CRuns          FALSE      FALSE
CRBI           FALSE      FALSE
CWalks         FALSE      FALSE
LeagueN        FALSE      FALSE
DivisionW      FALSE      FALSE
PutOuts        FALSE      FALSE
Assists        FALSE      FALSE
Errors         FALSE      FALSE
NewLeagueN     FALSE      FALSE
1 subsets of each size up to 19
Selection Algorithm: exhaustive
         AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRB
I CWalks LeagueN DivisionW PutOuts Assists
1  ( 1 )  " "   " "  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
" "    " "     " "       " "      " "
2  ( 1 )  " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
" "    " "     " "       " "      " "
3  ( 1 )  " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
" "    " "     " "       "*"      " "
4  ( 1 )  " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
" "    " "     "*"       "*"      " "
5  ( 1 )  "*"   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    " "   "*"
" "    " "     "*"       "*"      " "
6  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   " "    " "   "*"
" "    " "     "*"       "*"      " "
7  ( 1 )  " "   "*"  " "   " "  " " " " "*"   " "   "*"    "*"   "*"    " "   " "
" "    " "     "*"       "*"      " "
8  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   " "    " "   "*"    "*"   " "
"*"    " "     "*"       "*"      " "
9  ( 1 )  "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
"*"    " "     "*"       "*"      " "
10  ( 1 ) "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
"*"    " "     "*"       "*"      "*"
11  ( 1 ) "*"   "*"  " "   " "  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
"*"    "*"     "*"       "*"      "*"
12  ( 1 ) "*"   "*"  " "   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
"*"    "*"     "*"       "*"      "*"
13  ( 1 ) "*"   "*"  " "   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
"*"    "*"     "*"       "*"      "*"
14  ( 1 ) "*"   "*"  "*"   "*"  " " " " "*"   " "   "*"    " "   " "    "*"   "*"
"*"    "*"     "*"       "*"      "*"
15  ( 1 ) "*"   "*"  "*"   "*"  " " " " "*"   " "   "*"    "*"   " "    "*"   "*"
"*"    "*"     "*"       "*"      "*"
```

One advantage of the *regsubsets()* function is that it provides several comparison metrics, including Mallow's Cp, Adjusted R_2 and BIC.

```
# Review evaluation metrics
reg.summary$cp
reg.summary$adjr2
reg.summary$bic
```

```
> # Review evaluation metrics
> reg.summary$cp
 [1] 104.281319  50.723090  38.693127  27.856220  21.613011  14.023870  13.12
8474   7.400719   6.158685   5.009317
[11]   5.874113   7.330766   8.888112  10.481576  12.346193  14.187546  16.08
7831  18.011425  20.000000
> reg.summary$adjr2
 [1] 0.3188503 0.4208024 0.4450753 0.4672734 0.4808971 0.4972001 0.5007849 0.
5137083 0.5180572 0.5222606 0.5225706 0.5217245
[13] 0.5206736 0.5195431 0.5178661 0.5162219 0.5144464 0.5126097 0.5106270
> reg.summary$bic
 [1]  -90.84637 -128.92622 -135.62693 -141.80892 -144.07143 -147.91690 -145.2
5594 -147.61525 -145.44316 -143.21651
[11] -138.86077 -133.87283 -128.77759 -123.64420 -118.21832 -112.81768 -107.3
5339 -101.86391  -96.30412
```

By examining the output below, we see that according to BIC, the best performer is the model with 6 variables. According to Cp 10 variables. Adjusted R_2 suggests that 11 might be best. Again, no one measure is going to give us an entirely accurate picture, but they all agree that a model with 5 or fewer predictors is insufficient, and a model with more than 12 is overfitting.

```
# Best model by Cp, Adjusted R-Squared or BIC
which.min(reg.summary$cp)
which.max(reg.summary$adjr2)
which.min(reg.summary$bic)
```

```
> # Best model by Adjusted R-Squared
> which.max(reg.summary$adjr2)
[1] 11
> # Best model by Adjusted R-Squared
> which.min(reg.summary$cp)
[1] 10
> which.max(reg.summary$adjr2)
[1] 11
> which.min(reg.summary$bic)
[1] 6
```

The regsubsets() function can also be used for forward, backward and stepwise selections. An example of a backwards selection process is provided below. The backward selection is selected by setting the 'method' parameter equal to "backward".

```
# Backward selection with regsubsets
backward = regsubsets(Salary ~ ., data = Hitters, method = "backward")
reg.summary <- summary(backward)
reg.summary
names(reg.summary)

# Best model by Adjusted R-Squared
which.max(reg.summary$adjr2)
```

In the output below, we can see that the 8 variable model is preferred, according to the Adjusted R^2.

```
Subset selection object
Call: regsubsets.formula(Salary ~ ., data = Hitters, method = "backward")
19 Variables  (and intercept)
           Forced in Forced out
AtBat          FALSE      FALSE
Hits           FALSE      FALSE
HmRun          FALSE      FALSE
Runs           FALSE      FALSE
RBI            FALSE      FALSE
Walks          FALSE      FALSE
Years          FALSE      FALSE
CAtBat         FALSE      FALSE
CHits          FALSE      FALSE
CHmRun         FALSE      FALSE
CRuns          FALSE      FALSE
CRBI           FALSE      FALSE
CWalks         FALSE      FALSE
LeagueN        FALSE      FALSE
DivisionW      FALSE      FALSE
PutOuts        FALSE      FALSE
Assists        FALSE      FALSE
Errors         FALSE      FALSE
NewLeagueN     FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: backward
         AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
CWalks LeagueN DivisionW PutOuts Assists
1  ( 1 ) " "   " "  " "   " "  " " " " " "   " "   " "    " "   " "    "*"   " "
" "    " "     " "        " "     " "
2  ( 1 ) " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    "*"   " "
" "    " "     " "        " "     " "
3  ( 1 ) " "   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    "*"   " "
" "    " "     " "        "*"     " "
4  ( 1 ) "*"   "*"  " "   " "  " " " " " "   " "   " "    " "   " "    "*"   " "
" "    " "     " "        "*"     " "
5  ( 1 ) "*"   "*"  " "   " "  " " " "*"   " "   " "    " "   " "    "*"   " "
" "    " "     " "        "*"     " "
6  ( 1 ) "*"   "*"  " "   " "  " " " "*"   " "   " "    " "   " "    "*"   " "
" "    " "     "*"        "*"     " "
7  ( 1 ) "*"   "*"  " "   " "  " " " "*"   " "   " "    " "   " "    "*"   " "
"*"    " "     "*"        "*"     " "
8  ( 1 ) "*"   "*"  " "   " "  " " " "*"   " "   " "    " "   " "    "*"   "*"
"*"    " "     "*"        "*"     " "
```

```
          Errors NewLeagueN
1  ( 1 ) " "    " "
2  ( 1 ) " "    " "
3  ( 1 ) " "    " "
4  ( 1 ) " "    " "
5  ( 1 ) " "    " "
6  ( 1 ) " "    " "
7  ( 1 ) " "    " "
8  ( 1 ) " "    " "
> names(reg.summary)
[1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
> # Best model by Adjusted R-Squared
> which.max(reg.summary$adjr2)
[1] 8
```

When modeling and working in R, there is almost always more than one way to perform a task. The methods demonstrated here are meant to provide simple examples to get you started.

In addition, there are multiple measures that are used to compare models. None of them are definitive and at times the results of two measures will point to different models. When that occurs, you need to fit and examine the models that each method suggests. If neither model displays a clear advantage, then select the simpler model, or the model that is easier to interpret.