Appendix R5: Variable Selection

Multi-Collinearity, Subset Selection and Step Methods

J. Alberto Espinosa

8/1/2021

Table of Contents

Overview

Dimensionality is an important issue in predictive modeling. As we add predictors and complex transformations in our model, the bias of the model decreases while the explanatory power of the model increases, but the model variance also increases. This is the well known bias vs. variance tradeoff in predictive modeling. This is not really a problem when the multi-collinearity of the model is tolerable, but it really affects its performance when multi-collinarity is severe. In such cases, we need to pay close attention to how we select our predictors and whether multi-collinearity is a concern. The methods described in this script can help correct for multi-collinearity when predictors causing the problem can be removed. If not, please refer to the methods described in Ch. 7, which are ideally suited for predictive modeling in the presence of multi-collinearity. I first discuss how to diagnose multi-collinearity and then illustrate how to do subset selection and run various step methods for variable selection.

1. Multi-Collinearity

High multi-collinearity causes a lot of variance in the model. Multi-collinearity is tolerable, unless it is severe, in which case it causes the the model to have high variance and become unstable to be reliable. If you drop a few observations, the resulting coefficients should not change much. But if they change substantially, chances are that your model suffers from high multi-collinearity.

Multicollinearity can be tested in two ways:

- Evaluate the multi-collinearity condition of the model as a whole, using the **Condition Index (CI)**. The CI evaluates the overall linear association of the predictors for the **entire model**.
- If the CI shows that the model is multi-collinear, you can then inspect which predictors contribute the most to collinearity by inspecting the **Variance Inflation Variable (VIF)** of **each predictor**. We discuss both methods below.

Condition Index (CI)

The Condition Index (CI) is a general collinearity statistic for the full model. It takes the covariance matrix an computes its corresponding eigenvalues. As I discuss in the main chapter, collinearity is associated with eigenvalues that vary widely because this indicates that there are one or a few dominant eigenvector directions where the predictors correlate strongly. If the eigenvalues don't vary much, it is an indication that there are no dominant linear trends among the predictors, so multi-collinearity is not a concernt. The CI is a simple statistic that measures the ratio of the largest eigenvalue of the convariance matrix to the smallest. The CI is then evaluated as follows:

CI < 30 -> Multi-Collinearity is not a concern 30 < CI < 50 -> Multi-Collinearity is a concern, is somewhat tolerable 50 < CI -> Multi-Collinearity is severe and **must** be addressed

For the examples below, we will use the `ols_eigen_cindex()` function in the **{olsrr}** package to compute the CI using the **Boston** housing data set in the **{MASS}** package. But there are other packages like **{klaR}** and **{car}** with similar diagnostic tools.

```
library(olsrr) # Tools for OLS regressions
library(MASS) # Contains the Boston housing market data set
```

Let's first analyze the Full model

```
lm.fit <- lm(medv ~ ., data = Boston)
summary(lm.fit)

##
## Call:
## lm(formula = medv ~ ., data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## chas         2.687e+00  8.616e-01   3.118 0.001925 **
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
## ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## black        9.312e-03  2.686e-03   3.467 0.000573 ***
## lstat       -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

Now let's compute the CI:

```
coll <- ols_eigen_cindex(lm.fit)
coll[, 1:2] # Eigenvalues are in column 1 and CI in column 2
```

```
##      Eigenvalue Condition Index
## 1  10.092840813        1.000000
## 2   1.594070113        2.516245
## 3   0.959881737        3.242633
## 4   0.661730799        3.905405
## 5   0.241115274        6.469852
## 6   0.166505870        7.785597
## 7   0.108356145        9.651169
## 8   0.074488723       11.640227
## 9   0.041575247       15.580784
## 10  0.025639575       19.840440
## 11  0.013189367       27.662710
## 12  0.012073860       28.912366
## 13  0.007208734       37.417710
## 14  0.001323741       87.318288
```

As I discussed earlier, if the eigenvalues change substantially, it is an indication that multi-collinearity is present. The output in the example above shows all 14 eigenvalues, from largest to smallest. For example, the CI for the second eigenvalue is the `sqrt(10.09 / 1.59) = 2.51`, which is OK. But we need to evaluate all eigenvalues relative to each other,

so we only need to focus on the largest CI, which is the CI for the model, which is the square root of the ratio of the largest eigenvalue to the lowest, or `sqrt(10.09 / 0.0013) = 87.32`. Also note that the function above computes the CI on the correlation, rather than the covariance matrix, which is equivalent to running the CI on a standardized data set with `scale=T` and `center=F`. Let's see if we can further correct this by removing non-significant predictors.

```
lm.fit.red <- lm(medv ~ crim + zn + chas + rm + dis +
                        rad + tax + ptratio + black + lstat,
                        data = Boston)
summary(lm.fit.red)
```

```
##
## Call:
## lm(formula = medv ~ crim + zn + chas + rm + dis + rad + tax +
##     ptratio + black + lstat, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.980  -2.869  -0.693   1.732  26.674
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 22.605365   4.324998   5.227 2.55e-07 ***
## crim        -0.096497   0.033446  -2.885 0.004082 **
## zn           0.052811   0.013759   3.838 0.000140 ***
## chas         2.380299   0.871150   2.732 0.006513 **
## rm           3.940596   0.414703   9.502  < 2e-16 ***
## dis         -1.054766   0.166731  -6.326 5.63e-10 ***
## rad          0.282595   0.064772   4.363 1.56e-05 ***
## tax         -0.015723   0.003351  -4.692 3.51e-06 ***
## ptratio     -0.756520   0.125988  -6.005 3.71e-09 ***
## black        0.010239   0.002729   3.753 0.000196 ***
## lstat       -0.570699   0.047475 -12.021  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.846 on 495 degrees of freedom
## Multiple R-squared:  0.7279, Adjusted R-squared:  0.7224
## F-statistic: 132.4 on 10 and 495 DF,  p-value: < 2.2e-16
```

Let's look at the CI

```
coll.red <- ols_eigen_cindex(lm.fit.red)
coll.red [, 1:2]
```

```
##     Eigenvalue Condition Index
## 1  7.437083446        1.000000
## 2  1.486050685        2.237095
## 3  0.952366861        2.794468
```

```
## 4  0.610778278        3.489471
## 5  0.229718168        5.689887
## 6  0.144696149        7.169235
## 7  0.078266220        9.747969
## 8  0.033337026       14.936124
## 9  0.017108940       20.849218
## 10 0.008779415       29.105062
## 11 0.001814812       64.015555
```

Conclusion: The reduced model did not totally solve the multi-collinearity since the CI = 64.02, which is greater than 50, so the multi-collinearity condition is severe. Note that the predictor **nox** was significant in the full model, but it became non-significant when I removed the other non-significant predictors. The fact that nox lost its significance when I removed a few predictor is evidence of the issues that multi-collinearity causes. So, I removed **nox** too. Now, let's find out which predictors contribute the most to multi-collinearity by examining the VIF's.

Variance Inflation Factors (VIFs)

The VIF helps analyze each variable's contribution to multi-collinearity. There is one VIF for each predictor, measuring how much is the standard error of that coefficient, compared to the standard error of the coefficient for the same variable, if it were modeled as a single-predictor model. It also measures how much of the variance of the predictor can be explained by the remaining predictors. The VIF of predictor i is `VIFi = 1 / (1 - Ri^2)` where `Ri^2` is the R-Squared of a regression with predictor i as an outcome and all other variables as predictors. Therefore, if `Ri^2 = 0` (no correlation, perfectly independent) then `VIFi = 1`. At the other extreme, `Ri^2 = 1` (i.e., perfectly correlated, mathematically) will yield a `VIFi = infinite`, with the `standard error = infinite`, and a model with no solution. A `Ri^2 = 0.80` and `0.90` translate to `VIFi = 5` and `10` respectively. When examining VIF's, the typical thresholds are:

- `VIF < 5` -> No concern, little or no contribution to multi-collinearity
- `5 < VIF < 10` -> Some concern, but tolerable
- `VIF > 10` -> The variable contributes significantly to multi-collinearity

To test VIF's, we can use the `vif()` function in the **{car}** (Companion to Applied Regression) package or the `ols_vif_tol()` function in the **{olsrr}** package. They are both equally good at reporting VIF's but the second function also reports the tolerance for each variable. The tolerance is simply how much variance in the predictor is not explained by the other predictors, so it is the opposite of the VIF, or the inverse, to be precise `VIFi = 1 / Tolerance i`

```
ols_vif_tol(lm.fit) # Full model
```

```
##    Variables Tolerance      VIF
## 1       crim 0.5579761 1.792192
## 2         zn 0.4350175 2.298758
## 3      indus 0.2505263 3.991596
```

```
## 4       chas 0.9311027 1.073995
## 5        nox 0.2275976 4.393720
## 6         rm 0.5171314 1.933744
## 7        age 0.3224948 3.100826
## 8        dis 0.2527841 3.955945
## 9        rad 0.1336095 7.484496
## 10       tax 0.1110056 9.008554
## 11   ptratio 0.5558384 1.799084
## 12     black 0.7415531 1.348521
## 13     lstat 0.3399636 2.941491
```

```
ols_vif_tol(lm.fit.red) # Reduced model
```

```
##    Variables Tolerance      VIF
## 1       crim 0.5618248 1.779914
## 2         zn 0.4515421 2.214633
## 3       chas 0.9497271 1.052934
## 4         rm 0.5476704 1.825916
## 5        dis 0.3772279 2.650918
## 6        rad 0.1461844 6.840676
## 7        tax 0.1457620 6.860500
## 8    ptratio 0.6250006 1.599998
## 9      black 0.7493034 1.334573
## 10     lstat 0.4045582 2.471832
```

Interestingly, the CI is severe in both models, but none of the VIF's are greater than 10 in either model. This suggests strongly that the collinearity is between one or two predictors and the constant. This issue can be resolved by centering all the variables, including the outcome, in addition to scaling, and fitting the model this way. Generally, if the CI is severe, but the VIF's are not, there is less of a concern because none of the predictor coefficients' standard errors are severely inflated.

One final note is that the function `ols_coll_diag()` of the **{olsrr}** package will run the CI and VIF test in one pass. So, if you are interested in both statistics together, you can use that instead.

2. Variable Selection

We explore two approaches to select predictors for a model with statistical methods. As discussed in the main chapter, these methods only work when the models are nested. If the model specifications are not nested (i.e., they involve different predictors), if the models are of different types, or if they are not OLS or GLM models, these methods don't work and you will need to use cross-validation to select the best models. In this section, I discuss two approaches to variable selection: (1) Subset Selection, which are applicable when you have a finite number of nested models and you want to test which one has the strongest explanatory power; and (2) Step Methods, which are useful when you have many predictors and no pre-defined idea of how to select the most appropriate ones.

Subset Comparison

The most effective way to compare the difference in explanatory power of two or more nested models is with an ANOVA F-Test. Smaller models are more biased, but large models can have dimensionality problems, particularly if the multi-collinearity condition is severe. So, this is a balancing act. The best approach is to select the best predictors based on business knowledge and then test various subsets. When comparing two models, the larger model **always** has more explanatory power than the smaller model because we have added explanatory predictors to the model. But if the increase in explanatory power is not significant, then the added complexity of the larger model is undesirable, and the smaller model is preferred. But if the difference is significant, we would select the larger model. Naturally, after a model has been selected, it is important to test for multi-collinearity. Let's use the baseball data to illustrate these concepts and fit three models: reduced, large and full.

```
library(ISLR) # Contains the Hitters baseball player salary data set
Hitters <- na.omit(Hitters) # Let's remove missing values

# Small Model, 3 predictors
lm.reduced <- lm(Salary ~ AtBat + Hits + Walks, data = Hitters)

summary(lm.reduced) # Take a look

##
## Call:
## lm(formula = Salary ~ AtBat + Hits + Walks, data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1007.5  -245.4   -70.8   165.4  2039.1
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 121.0590    73.2481   1.653  0.09960 .
## AtBat        -2.0862     0.6324  -3.299  0.00111 **
## Hits          8.9252     1.9922   4.480 1.12e-05 ***
## Walks         7.1645     1.4097   5.082 7.16e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 386.1 on 259 degrees of freedom
## Multiple R-squared:  0.2758, Adjusted R-squared:  0.2674
## F-statistic: 32.88 on 3 and 259 DF,  p-value: < 2.2e-16

# Large Model, 5 predictors

lm.large <- lm(Salary ~ AtBat + Hits + Walks + Division + PutOuts,
                        data=Hitters)
```

```r
summary(lm.large) # Take a look
```

```
##
## Call:
## lm(formula = Salary ~ AtBat + Hits + Walks + Division + PutOuts,
##     data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -907.58 -216.89  -62.37  169.87 1965.55
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  173.02126   75.85170   2.281  0.02336 *
## AtBat         -2.01116    0.62016  -3.243  0.00134 **
## Hits           8.28040    1.95398   4.238 3.15e-05 ***
## Walks          6.47059    1.38568   4.670 4.87e-06 ***
## DivisionW   -120.43396   46.83854  -2.571  0.01070 *
## PutOuts        0.26542    0.08795   3.018  0.00280 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 376.3 on 257 degrees of freedom
## Multiple R-squared:  0.3176, Adjusted R-squared:  0.3043
## F-statistic: 23.92 on 5 and 257 DF,  p-value: < 2.2e-16
```

```r
# Full Model, 6 predictors

lm.full <- lm(Salary ~ AtBat + Hits + Walks + Division + PutOuts + Errors,
                       data=Hitters)

summary(lm.full) # Take a look
```

```
##
## Call:
## lm(formula = Salary ~ AtBat + Hits + Walks + Division + PutOuts +
##     Errors, data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -875.38 -208.93  -61.53  186.63 1980.81
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  183.51751   76.18550   2.409  0.01671 *
## AtBat         -1.79757    0.64097  -2.804  0.00543 **
## Hits           7.90088    1.97334   4.004 8.17e-05 ***
## Walks          6.15433    1.40529   4.379 1.74e-05 ***
## DivisionW   -120.78703   46.77802  -2.582  0.01038 *
```

```
## PutOuts        0.26459    0.08784   3.012  0.00285 **
## Errors        -4.93036    3.81042  -1.294  0.19686
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 375.8 on 256 degrees of freedom
## Multiple R-squared:  0.322,  Adjusted R-squared:  0.3061
## F-statistic: 20.27 on 6 and 256 DF,  p-value: < 2.2e-16
```

Now, let's use the `anova()` function to compare the models with **F-Test**. You can test these models in any order, but the convention is to go from smaller to larger model in the `anova()` function.

```
anova(lm.reduced, lm.large, lm.full)
```

```
## Analysis of Variance Table
##
## Model 1: Salary ~ AtBat + Hits + Walks
## Model 2: Salary ~ AtBat + Hits + Walks + Division + PutOuts
## Model 3: Salary ~ AtBat + Hits + Walks + Division + PutOuts + Errors
##   Res.Df      RSS Df Sum of Sq      F    Pr(>F)
## 1    259 38613147
## 2    257 36385237  2   2227910 7.8888 0.0004735 ***
## 3    256 36148827  1    236409 1.6742 0.1968614
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The ANOVA output above compares 3 models. But the comparisons are pairwise. That is, there are only 2 comparisons. The first line simply shows the degrees of freedom and residual sum of squares of the first model. Line 2 has the first comparison between **lm.reduced** and **lm.large**. The F-statistic in this comparison is 7.88, which translates to a significant p-value = 0.00047. This means that the large model has more explanatory power than the reduced model and this difference is significant. Line 3 has the second comparison between **lm.large** and **lm.full**. The F-statistic is quite small at 1.674 and the p-value = 0.196 is not significant. The full model still has more explanatory power than the large model because it has more predictors, but the difference is not significant, so we retain the simpler large model. That is, the added size and complexity of the full model is not beneficial.

One important thing to note is that if the `anova()` test is significant, the added predictors will be significant too. Conversely, if the `anova() test is not significant, the added predictors will not be significant either. This property is the foundation for the Step methods, which I discuss next.

Step Methods

Step methods are excellent for exploration when you don't have pre-selected subsets you would like to test. It is important to note that Step methods are no replacement for sound predictor selection from business domain familiarity, but they are very useful when the

number of candidate predictors is quite large. For example, if you have 20 candidate predictors, there are `2^20 = 1,048,576` possible models, which is practically unfeasible to fully compare. For this purpose, we use the convenient `step()` function, which is included in the **{stats}** package that loads by default when you star R, which works with both `lm()` (OLS) and `glm()` (GLM) models. Let's illustrate its use with the **Hitters** baseball player data contained in the **{ISLR}** package.

There are three types of Step methods: forward, backward and stepwise (i.e., both). In all of these methods, there is a lower bound and an upper bound model. The Step method will add and remove predictors. The lower bound model is the minimum and thus the mandatory set of predictors to include in the model. I often use the Null model (i.e., no predictors) for the lower bound, but if there are a few predictors that you must include in the model, you can set the lower bound to some Small model that includes those predictors only. For the upper bound, you can use the Full model that contains all the candidate predictors you identified.

Let's start by specifying the lower and upper bound models

```
library(ISLR) # Contains the Hitters data set
options(scipen=4) # To minimize the display of scientific notation
```

Lower bound model:

```
Hitters.null <- lm(Salary ~ 1, data=Hitters) # No predictors
summary(Hitters.null) # Take a look

##
## Call:
## lm(formula = Salary ~ 1, data = Hitters)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -468.4 -345.9 -110.9  214.1 1924.1
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   535.93      27.82   19.27   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 451.1 on 262 degrees of freedom
```

Upper bound model:

```
Hitters.full <- lm(Salary ~., data=Hitters) # All predictors
summary(Hitters.full) # Take a look

##
## Call:
## lm(formula = Salary ~ ., data = Hitters)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -907.62 -178.35  -31.11  139.09 1877.04
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  163.10359   90.77854   1.797 0.073622 .
## AtBat         -1.97987    0.63398  -3.123 0.002008 **
## Hits           7.50077    2.37753   3.155 0.001808 **
## HmRun          4.33088    6.20145   0.698 0.485616
## Runs          -2.37621    2.98076  -0.797 0.426122
## RBI           -1.04496    2.60088  -0.402 0.688204
## Walks          6.23129    1.82850   3.408 0.000766 ***
## Years         -3.48905   12.41219  -0.281 0.778874
## CAtBat        -0.17134    0.13524  -1.267 0.206380
## CHits          0.13399    0.67455   0.199 0.842713
## CHmRun        -0.17286    1.61724  -0.107 0.914967
## CRuns          1.45430    0.75046   1.938 0.053795 .
## CRBI           0.80771    0.69262   1.166 0.244691
## CWalks        -0.81157    0.32808  -2.474 0.014057 *
## LeagueN       62.59942   79.26140   0.790 0.430424
## DivisionW   -116.84925   40.36695  -2.895 0.004141 **
## PutOuts        0.28189    0.07744   3.640 0.000333 ***
## Assists        0.37107    0.22120   1.678 0.094723 .
## Errors        -3.36076    4.39163  -0.765 0.444857
## NewLeagueN   -24.76233   79.00263  -0.313 0.754218
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 315.6 on 243 degrees of freedom
## Multiple R-squared:  0.5461, Adjusted R-squared:  0.5106
## F-statistic: 15.39 on 19 and 243 DF,  p-value: < 2.2e-16
```

A solution with any of the Step methods will yield a model in between the lower and upper bound models. The `step()` function can be set to perform forward, backward and stepwise variable selection between the two models.

The inclusion and exclusion p-value thresholds are `p < 0.15` by default. This means that predictors will only be added to the model if they are significant at the `p < 0.15` level and will only be removed if their p-values are greater than `0.15`. This default can be changed as needed with the parameter **k**. The default is `k=2`, which is approximately equivalent to `p < 0.15`. If you want lower significance levels you can use higher values of k. For example, `k=2.7` yields approximately `p < 0.10`. To find exactly how k relates to p-values, you can use the chi-square distribution function `qchisq()`:

```
qchisq(0.01, 1, lower.tail=F) # the k value for p < 0.01 is 6.634

## [1] 6.634897

qchisq(0.05, 1, lower.tail=F) # the k value for p < 0.05 is 3.841
```

```
## [1] 3.841459
```

```
qchisq(0.10, 1, lower.tail=F) # the k value for p < 0.10 is 2.705
```

```
## [1] 2.705543
```

```
qchisq(0.15, 1, lower.tail=F) # the k value for p < 0.15 is 2.072
```

```
## [1] 2.072251
```

Because Step methods are exploratory in nature, there is no particular reason to change the default k value, as you will be able to observe the significance level of each predictor included. Nevertheless, you can use p-values smaller than 0.15 or k values higher than 2 to obtain smaller models, or use larger p-values or smaller k values to obtain larger models.

Forward Selection

To run a forward step variable selection use the lower and upper bound models in the `scope=` parameter using the `list()` function (i.e., the scope is passed to the `step()` function as a list object). The first model entered in the `step()` function is the starting point in the variable selection process. Since we are using the forward method, the starting point must be the **Null** model because we will only be adding predictors. We also need to use the `direction="forward"` parameter, which will cause the `step()` function to only add predictors. Finally, we need to include the `test = F` parameter, which will cause the `step()` function to evaluate the inclusion of predictors based on the significance p-value of the added predictor, which is equivalent to an F-test. Otherwise, the default comparison criteria is the AIC (Akaike's Information Criterion), which is a generic statistic of predictive accuracy. You should get similar results, but the F-test is more robust.

```
Hitters.fwd <- step(Hitters.null,
                    scope = list(lower = Hitters.null, upper = Hitters.full),
                    direction = "forward", test = "F")
```

```
## Start:  AIC=3215.77
## Salary ~ 1
##
##             Df Sum of Sq      RSS    AIC  F value    Pr(>F)
## + CRBI       1  17139434 36179679 3115.8 123.6438 < 2.2e-16 ***
## + CRuns      1  16881162 36437951 3117.6 120.9174 < 2.2e-16 ***
## + CHits      1  16065140 37253973 3123.5 112.5518 < 2.2e-16 ***
## + CAtBat     1  14759710 38559403 3132.5  99.9052 < 2.2e-16 ***
## + CHmRun     1  14692193 38626920 3133.0  99.2744 < 2.2e-16 ***
## + CWalks     1  12792622 40526491 3145.6  82.3874 < 2.2e-16 ***
## + RBI        1  10771083 42548030 3158.4  66.0725 1.757e-14 ***
## + Walks      1  10504833 42814280 3160.1  64.0385 4.014e-14 ***
## + Hits       1  10260491 43058621 3161.6  62.1940 8.531e-14 ***
## + Runs       1   9399158 43919955 3166.8  55.8557 1.180e-12 ***
## + Years      1   8559105 44760007 3171.7  49.9090 1.463e-11 ***
## + AtBat      1   8309469 45009644 3173.2  48.1846 3.065e-11 ***
## + HmRun      1   6273967 47045145 3184.8  34.8071 1.125e-08 ***
```

```
## + PutOuts    1   4814100 48505013 3192.9  25.9041 6.871e-07 ***
## + Division   1   1976102 51343011 3207.8  10.0454  0.001709 **
## <none>                  53319113 3215.8
## + Assists    1     34497 53284615 3217.6   0.1690  0.681361
## + League     1     10876 53308237 3217.7   0.0532  0.817687
## + Errors     1      1555 53317558 3217.8   0.0076  0.930538
## + NewLeague  1       428 53318684 3217.8   0.0021  0.963511
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=3115.78
## Salary ~ CRBI
##
##             Df Sum of Sq      RSS    AIC F value    Pr(>F)
## + Hits       1   5533119 30646560 3074.1 46.9420 5.275e-11 ***
## + Runs       1   5176532 31003147 3077.2 43.4117 2.450e-10 ***
## + Walks      1   4199733 31979946 3085.3 34.1442 1.526e-08 ***
## + AtBat      1   4064585 32115095 3086.4 32.9064 2.681e-08 ***
## + RBI        1   3308272 32871407 3092.6 26.1671 6.085e-07 ***
## + PutOuts    1   3267035 32912644 3092.9 25.8086 7.204e-07 ***
## + Division   1   1733887 34445793 3104.9 13.0875 0.0003571 ***
## + Years      1   1667339 34512340 3105.4 12.5610 0.0004669 ***
## + HmRun      1   1271587 34908092 3108.4  9.4709 0.0023108 **
## + CRuns      1    354561 35825119 3115.2  2.5732 0.1099006
## + Assists    1    346020 35833659 3115.2  2.5106 0.1142958
## <none>                  36179679 3115.8
## + Errors     1    194403 35985276 3116.4  1.4046 0.2370372
## + CAtBat     1     92261 36087418 3117.1  0.6647 0.4156463
## + CHits      1     75469 36104210 3117.2  0.5435 0.4616584
## + CWalks     1     51974 36127705 3117.4  0.3740 0.5413451
## + NewLeague  1     17778 36161901 3117.7  0.1278 0.7209896
## + League     1     11825 36167855 3117.7  0.0850 0.7708606
## + CHmRun     1       515 36179165 3117.8  0.0037 0.9515485
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=3074.13
## Salary ~ CRBI + Hits
##
##             Df Sum of Sq      RSS    AIC F value    Pr(>F)
## + PutOuts    1   1397263 29249297 3063.8 12.3726 0.0005143 ***
## + Division   1   1279275 29367285 3064.9 11.2824 0.0009002 ***
## + AtBat      1    821767 29824793 3069.0  7.1363 0.0080327 **
## + Walks      1    781767 29864793 3069.3  6.7798 0.0097516 **
## + Years      1    254910 30391650 3073.9  2.1724 0.1417246
## <none>                  30646560 3074.1
## + League     1    208880 30437680 3074.3  1.7774 0.1836405
## + CRuns      1    132614 30513946 3075.0  1.1256 0.2897011
## + NewLeague  1    118474 30528086 3075.1  1.0051 0.3170083
## + Runs       1    114198 30532362 3075.1  0.9687 0.3259179
```

```
## + Errors     1     99776 30546784 3075.3  0.8460 0.3585477
## + CAtBat     1     83517 30563043 3075.4  0.7077 0.4009684
## + Assists    1     44781 30601779 3075.7  0.3790 0.5386742
## + CWalks     1     23668 30622892 3075.9  0.2002 0.6549532
## + CHmRun     1      4790 30641769 3076.1  0.0405 0.8406802
## + CHits      1      4358 30642202 3076.1  0.0368 0.8479531
## + HmRun      1      2173 30644387 3076.1  0.0184 0.8923043
## + RBI        1      1137 30645423 3076.1  0.0096 0.9219997
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Note: output was cut here to save paper. This is the last step:

## Step:  AIC=3031.26
## Salary ~ CRBI + Hits + PutOuts + Division + AtBat + Walks + CWalks +
##     CRuns + CAtBat + Assists
##
##             Df Sum of Sq      RSS    AIC F value Pr(>F)
## <none>                    24500402 3031.3
## + League     1    113056 24387345 3032.0  1.1636 0.2818
## + Runs       1     75900 24424501 3032.4  0.7800 0.3780
## + NewLeague  1     64712 24435690 3032.6  0.6647 0.4157
## + CHits      1     37564 24462838 3032.8  0.3854 0.5353
## + Errors     1     35264 24465138 3032.9  0.3618 0.5481
## + Years      1     19883 24480519 3033.0  0.2039 0.6520
## + CHmRun     1      4356 24496046 3033.2  0.0446 0.8328
## + HmRun      1      1189 24499212 3033.2  0.0122 0.9122
## + RBI        1       359 24500043 3033.2  0.0037 0.9517

summary(Hitters.fwd)

##
## Call:
## lm(formula = Salary ~ CRBI + Hits + PutOuts + Division + AtBat +
##     Walks + CWalks + CRuns + CAtBat + Assists, data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -939.11 -176.87  -34.08  130.90 1910.55
##
## Coefficients:
##               Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  162.53544   66.90784   2.429  0.015830 *
## CRBI           0.77431    0.20961   3.694  0.000271 ***
## Hits           6.91802    1.64665   4.201 0.0000369 ***
## PutOuts        0.29737    0.07444   3.995 0.0000850 ***
## DivisionW   -112.38006   39.21438  -2.866  0.004511 **
## AtBat         -2.16865    0.53630  -4.044 0.0000700 ***
## Walks          5.77322    1.58483   3.643  0.000327 ***
## CWalks        -0.83083    0.26359  -3.152  0.001818 **
```

```
## CRuns          1.40825    0.39040   3.607  0.000373 ***
## CAtBat        -0.13008    0.05550  -2.344  0.019858 *
## Assists        0.28317    0.15766   1.796  0.073673 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 311.8 on 252 degrees of freedom
## Multiple R-squared:  0.5405, Adjusted R-squared:  0.5223
## F-statistic: 29.64 on 10 and 252 DF,  p-value: < 2.2e-16
```

I cut the output above to limit the use of paper, but the `step()` outputs one model in each step, so that you can explore which variables entered the model when. In the first step of the output above you will notice two important things:

First, the predictors are ordered from smallest to largest p-value. These p-values are NOT the p-values in the model because initially there are no predictors in the Null model. Rather, these p-values indicate what the respective p-values will be if the predictor is added to the model.

Second, there is a **+** symbol next to each predictor. This is NOT a coefficient sign. Rather, it is a symbol that indicates that the predictor is a candidate for inclusion in the model, thus the **+**. Because the first step is the Null model, all predictors have a **+** symbol. Naturally, the first predictor in the output will be the first predictor to be added to the model, which in this case is **CRBI**, which will appear in the model output in the next step.

In the second step, you will notice that CRBI is no longer listed. This is so because it is already added to the model, so it is no longer a candidate for inclusion. You can also see that the model formula listed at the top of the step output includes **CRBI** in the model. Following the same logic, **Hits** will be the next predictor to be added to the model, and so on. You will notice in the very last step, that there are no more predictors with a p-value < 0.15, so no more predictors will be added to the model after this step and the process stops. The `summary()` function outputs the typical regression summary results.

Let's illustrate what happens when we increase k to 3.841. As you can see in the output below (cut to save paper), there are fewer predictors included because the p-value threshold is now 0.05. Notice that the all the final predictors included have a significance level of $p < 0.05$.

```
Hitters.fwd.3.8 <- step(Hitters.null,
                        scope = list(lower = Hitters.null, upper =
Hitters.full),
                        direction = "forward", test = "F", k = 3.841)

## Start:  AIC=3217.61
## Salary ~ 1
##
##             Df Sum of Sq      RSS    AIC  F value    Pr(>F)
## + CRBI       1  17139434 36179679 3119.5 123.6438 < 2.2e-16 ***
## + CRuns      1  16881162 36437951 3121.3 120.9174 < 2.2e-16 ***
## + CHits      1  16065140 37253973 3127.2 112.5518 < 2.2e-16 ***
```

```
## + CAtBat     1  14759710 38559403 3136.2  99.9052 < 2.2e-16 ***
## + CHmRun     1  14692193 38626920 3136.7  99.2744 < 2.2e-16 ***
## + CWalks     1  12792622 40526491 3149.3  82.3874 < 2.2e-16 ***
## + RBI        1  10771083 42548030 3162.1  66.0725 1.757e-14 ***
## + Walks      1  10504833 42814280 3163.7  64.0385 4.014e-14 ***
## + Hits       1  10260491 43058621 3165.2  62.1940 8.531e-14 ***
## + Runs       1   9399158 43919955 3170.4  55.8557 1.180e-12 ***
## + Years      1   8559105 44760007 3175.4  49.9090 1.463e-11 ***
## + AtBat      1   8309469 45009644 3176.9  48.1846 3.065e-11 ***
## + HmRun      1   6273967 47045145 3188.5  34.8071 1.125e-08 ***
## + PutOuts    1   4814100 48505013 3196.6  25.9041 6.871e-07 ***
## + Division   1   1976102 51343011 3211.5  10.0454  0.001709 **
## <none>                   53319113 3217.6
## + Assists    1     34497 53284615 3221.3   0.1690  0.681361
## + League     1     10876 53308237 3221.4   0.0532  0.817687
## + Errors     1      1555 53317558 3221.4   0.0076  0.930538
## + NewLeague  1       428 53318684 3221.4   0.0021  0.963511
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

Note: output was cut here to save paper. This is the last step:

```
## Step:  AIC=3053.73
## Salary ~ CRBI + Hits + PutOuts + Division + AtBat + Walks
##
##             Df Sum of Sq      RSS    AIC F value Pr(>F)
## <none>                   26194904 3053.7
## + CWalks     1    240687 25954217 3055.2  2.3647 0.1253
## + Years      1    184508 26010396 3055.7  1.8089 0.1798
## + CRuns      1    110695 26084209 3056.5  1.0822 0.2992
## + League     1     77974 26116930 3056.8  0.7613 0.3837
## + Assists    1     75782 26119122 3056.8  0.7399 0.3905
## + NewLeague  1     40909 26153995 3057.2  0.3989 0.5282
## + CHits      1     37304 26157599 3057.2  0.3637 0.5470
## + RBI        1     11728 26183176 3057.5  0.1142 0.7357
## + HmRun      1      4747 26190157 3057.5  0.0462 0.8299
## + Errors     1      2727 26192177 3057.6  0.0266 0.8707
## + CAtBat     1      2630 26192274 3057.6  0.0256 0.8730
## + CHmRun     1       943 26193961 3057.6  0.0092 0.9237
## + Runs       1        37 26194867 3057.6  0.0004 0.9849
```

```
summary(Hitters.fwd.3.8)
```

```
##
## Call:
## lm(formula = Salary ~ CRBI + Hits + PutOuts + Division + AtBat +
##     Walks, data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -873.11 -181.72  -25.91  141.77 2040.47
##
## Coefficients:
##               Estimate Std. Error t value   Pr(>|t|)
## (Intercept)   91.51180   65.00006   1.408   0.160382
## CRBI           0.64302    0.06443   9.979    < 2e-16 ***
## Hits           7.60440    1.66254   4.574 0.00000746 ***
## PutOuts        0.26431    0.07477   3.535   0.000484 ***
## DivisionW   -122.95153   39.82029  -3.088   0.002239 **
## AtBat         -1.86859    0.52742  -3.543   0.000470 ***
## Walks          3.69765    1.21036   3.055   0.002488 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 319.9 on 256 degrees of freedom
## Multiple R-squared:  0.5087, Adjusted R-squared:  0.4972
## F-statistic: 44.18 on 6 and 256 DF,  p-value: < 2.2e-16
```

Backward Selection

Backward selection works just like forward selection, with the same lower and upper bound models, but this time the starting model must be the Full model because we can only remove predictors. We also need to change the attribute `direction=` to `"backward"`. This will cause the process to proceed in a backward fashion. Just like in the forward process, the default k value is 2, which is approximately equivalent to `p < 0.15`. This means that any predictor that is not significant at the $p < 0.15$ level will be removed from the model. One more difference is that there is a **-** symbol next to each predictor, rather than **+**. This is so because predictors are candidates for removal, as there is no addition in backward step. The predictors are listed from least to most significance, so the first predictor to be removed is always the first one at the top.

```
Hitters.back <- step(Hitters.full,
                     scope = list(lower = Hitters.null, upper =
Hitters.full),
                     direction = "backward", test = "F")

## Start:  AIC=3046.02
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + Years +
##     CAtBat + CHits + CHmRun + CRuns + CRBI + CWalks + League +
##     Division + PutOuts + Assists + Errors + NewLeague
##
##             Df Sum of Sq      RSS    AIC F value    Pr(>F)
## - CHmRun     1      1138 24201837 3044.0  0.0114 0.9149671
## - CHits      1      3930 24204629 3044.1  0.0395 0.8427129
## - Years      1      7869 24208569 3044.1  0.0790 0.7788736
## - NewLeague  1      9784 24210484 3044.1  0.0982 0.7542178
## - RBI        1     16076 24216776 3044.2  0.1614 0.6882042
## - HmRun      1     48572 24249272 3044.6  0.4877 0.4856158
## - Errors     1     58324 24259023 3044.7  0.5856 0.4448566
```

```
## - League     1     62121 24262821 3044.7  0.6238 0.4304236
## - Runs       1     63291 24263990 3044.7  0.6355 0.4261225
## - CRBI       1    135439 24336138 3045.5  1.3599 0.2446905
## - CAtBat     1    159864 24360564 3045.8  1.6052 0.2063804
## <none>                   24200700 3046.0
## - Assists    1    280263 24480963 3047.1  2.8141 0.0947232 .
## - CRuns      1    374007 24574707 3048.1  3.7554 0.0537951 .
## - CWalks     1    609408 24810108 3050.6  6.1191 0.0140574 *
## - Division   1    834491 25035190 3052.9  8.3791 0.0041408 **
## - AtBat      1    971288 25171987 3054.4  9.7527 0.0020077 **
## - Hits       1    991242 25191941 3054.6  9.9531 0.0018082 **
## - Walks      1   1156606 25357305 3056.3 11.6135 0.0007662 ***
## - PutOuts    1   1319628 25520328 3058.0 13.2504 0.0003329 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

Note: output was cut here to save paper. This is the last step:

```
## Step:  AIC=3031.26
## Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks +
##     Division + PutOuts + Assists
##
##            Df Sum of Sq      RSS    AIC F value     Pr(>F)
## <none>                   24500402 3031.3
## - Assists   1    313650 24814051 3032.6  3.2261  0.0736726 .
## - CAtBat    1    534156 25034558 3034.9  5.4941  0.0198584 *
## - Division  1    798473 25298875 3037.7  8.2127  0.0045109 **
## - CWalks    1    965875 25466276 3039.4  9.9345  0.0018183 **
## - CRuns     1   1265082 25765484 3042.5 13.0121  0.0003731 ***
## - Walks     1   1290168 25790569 3042.8 13.2701  0.0003274 ***
## - CRBI      1   1326770 25827172 3043.1 13.6466  0.0002706 ***
## - PutOuts   1   1551523 26051925 3045.4 15.9583 0.00008504 ***
## - AtBat     1   1589780 26090181 3045.8 16.3518 0.00006996 ***
## - Hits      1   1716068 26216469 3047.1 17.6507 0.00003686 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(Hitters.back)
```

```
##
## Call:
## lm(formula = Salary ~ AtBat + Hits + Walks + CAtBat + CRuns +
##     CRBI + CWalks + Division + PutOuts + Assists, data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -939.11 -176.87  -34.08  130.90 1910.55
##
## Coefficients:
##               Estimate Std. Error t value  Pr(>|t|)
```

```
## (Intercept)  162.53544   66.90784   2.429  0.015830 *
## AtBat         -2.16865    0.53630  -4.044 0.0000700 ***
## Hits           6.91802    1.64665   4.201 0.0000369 ***
## Walks          5.77322    1.58483   3.643  0.000327 ***
## CAtBat        -0.13008    0.05550  -2.344  0.019858 *
## CRuns          1.40825    0.39040   3.607  0.000373 ***
## CRBI           0.77431    0.20961   3.694  0.000271 ***
## CWalks        -0.83083    0.26359  -3.152  0.001818 **
## DivisionW   -112.38006   39.21438  -2.866  0.004511 **
## PutOuts        0.29737    0.07444   3.995 0.0000850 ***
## Assists        0.28317    0.15766   1.796  0.073673 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 311.8 on 252 degrees of freedom
## Multiple R-squared:  0.5405, Adjusted R-squared:  0.5223
## F-statistic: 29.64 on 10 and 252 DF,  p-value: < 2.2e-16
```

Stepwise Selection

The stepwise selection process is very similar, except that predictors can be either added or removed from the model. Unless you change the default value of k, any predictor that is significant at the $p < 0.15$ level will be added, and any that is not will be removed. Another difference in each step is that you may now see either a **+** or a **-**. As before, a **+** denotes that the predictor is not in the model and, therefore, it is a candidate for inclusion. Similarly, a **-** symbol denotes that the predictor is already in the model and, therefore, it is a candidate for removal.

Two other important differences are:

First, the **direction** attribute is set to `direction="both"`, which will trigger the stepwise process of either adding or removing predictors; and

Second, the starting models can be either the Null or the Full model, your choice. If you choose the Null model, your first step will be to add predictors to the Null model in a forward fashion, which is the only direction you can go because there are no predictors in the Null model yet. If you choose the Full model, your first step will be to remove predictors from the Full model in a backward fashion. Again, that's the only direction possible. But after the first step, predictors may either be added or removed in every step. Let's illustrate this method. Notice that we start with the Full model, so our first step is backward. You can try starting with the Null model on your own, but my preference is to always start with the Full and least biased model. So, all predictors have a **-** symbol at the beginning, but notice that there is a predictor with a **+** in the second step, indicating that the predictor has already been removed, but it is now a candidate to be added back to the model.

```
Hitters.step <- step(Hitters.full,
                     scope=list(lower=Hitters.null, upper=Hitters.full),
                     direction="both", test="F")
```

```
## Start:  AIC=3046.02
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + Years +
##     CAtBat + CHits + CHmRun + CRuns + CRBI + CWalks + League +
##     Division + PutOuts + Assists + Errors + NewLeague
##
##             Df Sum of Sq      RSS    AIC F value    Pr(>F)
## - CHmRun     1      1138 24201837 3044.0  0.0114 0.9149671
## - CHits      1      3930 24204629 3044.1  0.0395 0.8427129
## - Years      1      7869 24208569 3044.1  0.0790 0.7788736
## - NewLeague  1      9784 24210484 3044.1  0.0982 0.7542178
## - RBI        1     16076 24216776 3044.2  0.1614 0.6882042
## - HmRun      1     48572 24249272 3044.6  0.4877 0.4856158
## - Errors     1     58324 24259023 3044.7  0.5856 0.4448566
## - League     1     62121 24262821 3044.7  0.6238 0.4304236
## - Runs       1     63291 24263990 3044.7  0.6355 0.4261225
## - CRBI       1    135439 24336138 3045.5  1.3599 0.2446905
## - CAtBat     1    159864 24360564 3045.8  1.6052 0.2063804
## <none>                   24200700 3046.0
## - Assists    1    280263 24480963 3047.1  2.8141 0.0947232 .
## - CRuns      1    374007 24574707 3048.1  3.7554 0.0537951 .
## - CWalks     1    609408 24810108 3050.6  6.1191 0.0140574 *
## - Division   1    834491 25035190 3052.9  8.3791 0.0041408 **
## - AtBat      1    971288 25171987 3054.4  9.7527 0.0020077 **
## - Hits       1    991242 25191941 3054.6  9.9531 0.0018082 **
## - Walks      1   1156606 25357305 3056.3 11.6135 0.0007662 ***
## - PutOuts    1   1319628 25520328 3058.0 13.2504 0.0003329 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=3044.03
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + Years +
##     CAtBat + CHits + CRuns + CRBI + CWalks + League + Division +
##     PutOuts + Assists + Errors + NewLeague
##
##             Df Sum of Sq      RSS    AIC F value    Pr(>F)
## - Years      1      7609 24209447 3042.1  0.0767 0.7820312
## - NewLeague  1     10268 24212106 3042.2  0.1035 0.7479165
## - CHits      1     14003 24215840 3042.2  0.1412 0.7074399
## - RBI        1     14955 24216793 3042.2  0.1508 0.6981310
## - HmRun      1     52777 24254614 3042.6  0.5321 0.4664293
## - Errors     1     59530 24261367 3042.7  0.6002 0.4392618
## - League     1     63407 24265244 3042.7  0.6393 0.4247563
## - Runs       1     64860 24266698 3042.7  0.6539 0.4195043
## - CAtBat     1    174992 24376830 3043.9  1.7643 0.1853360
## <none>                   24201837 3044.0
## - Assists    1    285766 24487603 3045.1  2.8811 0.0909022 .
## + CHmRun     1      1138 24200700 3046.0  0.0114 0.9149671
## - CRuns      1    611358 24813196 3048.6  6.1636 0.0137136 *
## - CWalks     1    645627 24847464 3049.0  6.5091 0.0113434 *
## - Division   1    834637 25036474 3050.9  8.4147 0.0040618 **
```

```
## - CRBI       1    864220 25066057 3051.3  8.7130 0.0034675 **
## - AtBat      1    970861 25172699 3052.4  9.7881 0.0019700 **
## - Hits       1   1025981 25227819 3052.9 10.3438 0.0014747 **
## - Walks      1   1167378 25369216 3054.4 11.7694 0.0007069 ***
## - PutOuts    1   1325273 25527110 3056.1 13.3612 0.0003145 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Note: output was cut here to save paper. This is the last step:

## Step:  AIC=3031.26
## Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks +
##     Division + PutOuts + Assists
##
##             Df Sum of Sq      RSS    AIC F value     Pr(>F)
## <none>                   24500402 3031.3
## + League     1    113056 24387345 3032.0  1.1636  0.2817552
## + Runs       1     75900 24424501 3032.4  0.7800  0.3779884
## + NewLeague  1     64712 24435690 3032.6  0.6647  0.4156749
## - Assists    1    313650 24814051 3032.6  3.2261  0.0736726 .
## + CHits      1     37564 24462838 3032.8  0.3854  0.5352800
## + Errors     1     35264 24465138 3032.9  0.3618  0.5480572
## + Years      1     19883 24480519 3033.0  0.2039  0.6520127
## + CHmRun     1      4356 24496046 3033.2  0.0446  0.8328494
## + HmRun      1      1189 24499212 3033.2  0.0122  0.9121905
## + RBI        1       359 24500043 3033.2  0.0037  0.9517012
## - CAtBat     1    534156 25034558 3034.9  5.4941  0.0198584 *
## - Division   1    798473 25298875 3037.7  8.2127  0.0045109 **
## - CWalks     1    965875 25466276 3039.4  9.9345  0.0018183 **
## - CRuns      1   1265082 25765484 3042.5 13.0121  0.0003731 ***
## - Walks      1   1290168 25790569 3042.8 13.2701  0.0003274 ***
## - CRBI       1   1326770 25827172 3043.1 13.6466  0.0002706 ***
## - PutOuts    1   1551523 26051925 3045.4 15.9583 0.00008504 ***
## - AtBat      1   1589780 26090181 3045.8 16.3518 0.00006996 ***
## - Hits       1   1716068 26216469 3047.1 17.6507 0.00003686 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

summary(Hitters.step)

##
## Call:
## lm(formula = Salary ~ AtBat + Hits + Walks + CAtBat + CRuns +
##     CRBI + CWalks + Division + PutOuts + Assists, data = Hitters)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -939.11 -176.87  -34.08  130.90 1910.55
##
## Coefficients:
##               Estimate Std. Error t value  Pr(>|t|)
```

```
## (Intercept)  162.53544   66.90784   2.429  0.015830 *
## AtBat         -2.16865    0.53630  -4.044 0.0000700 ***
## Hits           6.91802    1.64665   4.201 0.0000369 ***
## Walks          5.77322    1.58483   3.643  0.000327 ***
## CAtBat        -0.13008    0.05550  -2.344  0.019858 *
## CRuns          1.40825    0.39040   3.607  0.000373 ***
## CRBI           0.77431    0.20961   3.694  0.000271 ***
## CWalks        -0.83083    0.26359  -3.152  0.001818 **
## DivisionW   -112.38006   39.21438  -2.866  0.004511 **
## PutOuts        0.29737    0.07444   3.995 0.0000850 ***
## Assists        0.28317    0.15766   1.796  0.073673 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 311.8 on 252 degrees of freedom
## Multiple R-squared:  0.5405, Adjusted R-squared:  0.5223
## F-statistic: 29.64 on 10 and 252 DF,  p-value: < 2.2e-16
```

Best Subset Selection

I only discuss this method briefly, because the Step methods accomplish the same purpose and are superior in my opinion. But it is a useful method to know. Best subset selection works similarly to the Step method, but there are additions or removals in steps. Rather, the first step will try all possible 1-predictor models and select the best of all in terms of explanatory power. It will then fit all possible 2-predictor models, and so on. We use the `regsubsets()` function from the **{leaps}** package for this purpose. By default, `regsubsets()` stops when the model has 8 predictors, but you can change this with the `nvmax=` parameter. Let's illustrate this:

```
library(leaps) # Contains the regsubsets() function

regfit.full <- regsubsets(Salary ~ ., Hitters)
summary(regfit.full)

## Subset selection object
## Call: regsubsets.formula(Salary ~ ., Hitters)
## 19 Variables  (and intercept)

## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "
```

```
##          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 ) " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 ) " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 ) " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 ) "*"    " "     "*"       "*"     " "     " "    " "
```

The **"*"** in the output shows the predictors selected in each model. You can also view the R-Squared of each model with the `$rsq` attribute:

```
summary(regfit.full)$rsq # View the R-square of each model

## [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227
## [8] 0.5285569
```

Let's change the number of predictors in the model up to 10

```
regfit.full <- regsubsets(Salary ~ ., data = Hitters, nvmax = 10)
summary(regfit.full)

## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 10)
## 19 Variables  (and intercept)

## 1 subsets of each size up to 10
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "
## 9  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"

##          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 ) " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 ) " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 ) " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 ) "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 ) "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "
```