Feature Selection

Nate Wells

Math 243: Stat Learning

October 4th, 2021

Outline

In today's class, we will...

- Perform some exploratory data analysis on a new data set
- Investigate algorithms for selecting good subsets of predictors

Section 1

Explaratory Data Analysis

Molecular Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

Molecular Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

```
library(AppliedPredictiveModeling)
data(solubility)
```

Molecular Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

```
library(AppliedPredictiveModeling)
data(solubility)
```

- The solubility of a compound indicates how easily it dissolves in a solvent (often water), and is measured as the amount of solvent required to dissolve 1 part of the compound.
  - The less solvent required, the more soluble the compound.
  - In the dataset, the log solubility is reported, since solubility spans many orders of magnitude

Molecular Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

```
library(AppliedPredictiveModeling)
data(solubility)
```

- The solubility of a compound indicates how easily it dissolves in a solvent (often water), and is measured as the amount of solvent required to dissolve 1 part of the compound.
  - The less solvent required, the more soluble the compound.
  - In the dataset, the log solubility is reported, since solubility spans many orders of magnitude
- The data also contains 16 chemical count descriptors, such as "number of bonds" or "number of bromine atoms"

Molecular Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

```
library(AppliedPredictiveModeling)
data(solubility)
```

- The solubility of a compound indicates how easily it dissolves in a solvent (often water), and is measured as the amount of solvent required to dissolve 1 part of the compound.
  - The less solvent required, the more soluble the compound.
  - In the dataset, the log solubility is reported, since solubility spans many orders of magnitude
- The data also contains 16 chemical count descriptors, such as "number of bonds" or "number of bromine atoms"
- Finally, the data contains 4 continuous descriptors, such as "molecular weight" or "surface area"

Molecular Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

```
library(AppliedPredictiveModeling)
data(solubility)
```

- The solubility of a compound indicates how easily it dissolves in a solvent (often water), and is measured as the amount of solvent required to dissolve 1 part of the compound.
  - The less solvent required, the more soluble the compound.
  - In the dataset, the log solubility is reported, since solubility spans many orders of magnitude
- The data also contains 16 chemical count descriptors, such as "number of bonds" or "number of bromine atoms"
- Finally, the data contains 4 continuous descriptors, such as "molecular weight" or "surface area"

We are interested in determining solubility based on these 20 chemical descriptors.

Pre-Processing

- The `solubability` actually consists of 4 data sets: `solTestX`, `solTrainX`, `solTestY`, `solTrainY`

Pre-Processing

- The `solubability` actually consists of 4 data sets: `solTestX`, `solTrainX`, `solTestY`, `solTrainY`
- The `X` and `Y` indicate the data is pre-divided into separate sets for predictors and response.

Pre-Processing

- The `solubability` actually consists of 4 data sets: `solTestX`, `solTrainX`, `solTestY`, `solTrainY`
- The `X` and `Y` indicate the data is pre-divided into separate sets for predictors and response.
- Additionally, data have already been partitioned into test and training sets (25 / 75)

Pre-Processing

- The `solubability` actually consists of 4 data sets: `solTestX`, `solTrainX`, `solTestY`, `solTrainY`
- The `X` and `Y` indicate the data is pre-divided into separate sets for predictors and response.
- Additionally, data have already been partitioned into test and training sets (25 / 75)
- It will be easier to have predictors and response in the same set, so we'll bind columns together:

```
solTest <- data.frame(solTestX, Solubility = solTestY)
solTrain <- data.frame(solTrainX, Solubility =  solTrainY)
```

Pre-Processing

- The `solubability` actually consists of 4 data sets: `solTestX`, `solTrainX`, `solTestY`, `solTrainY`
- The `X` and `Y` indicate the data is pre-divided into separate sets for predictors and response.
- Additionally, data have already been partitioned into test and training sets (25 / 75)
- It will be easier to have predictors and response in the same set, so we'll bind columns together:

```
solTest <- data.frame(solTestX, Solubility = solTestY)
solTrain <- data.frame(solTrainX, Solubility =  solTrainY)
```

- The data also contains 218 binary "fingerprints" for each compound indicating presence of particular chemical substructure, each beginning with "FP"

Pre-Processing

- The `solubability` actually consists of 4 data sets: `solTestX`, `solTrainX`, `solTestY`, `solTrainY`
- The `X` and `Y` indicate the data is pre-divided into separate sets for predictors and response.
- Additionally, data have already been partitioned into test and training sets (25 / 75)
- It will be easier to have predictors and response in the same set, so we'll bind columns together:

```
solTest <- data.frame(solTestX, Solubility = solTestY)
solTrain <- data.frame(solTrainX, Solubility =  solTrainY)
```

- The data also contains 218 binary "fingerprints" for each compound indicating presence of particular chemical substructure, each beginning with "FP"
- We'll ignore these predictors.

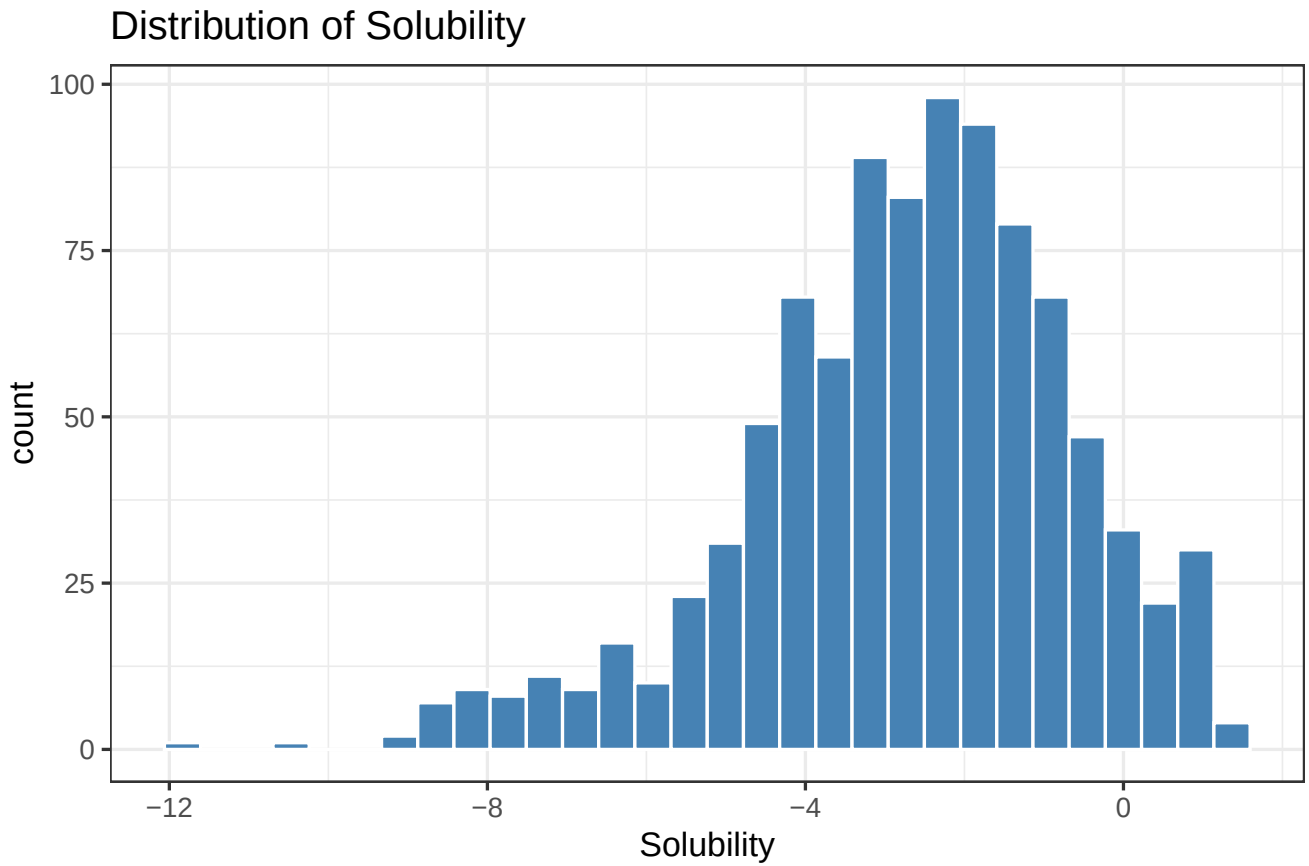
```
solTest <- solTest %>% select(!starts_with("FP"))
solTrain <- solTrain %>% select(!starts_with("FP"))
```

Distribution of Response

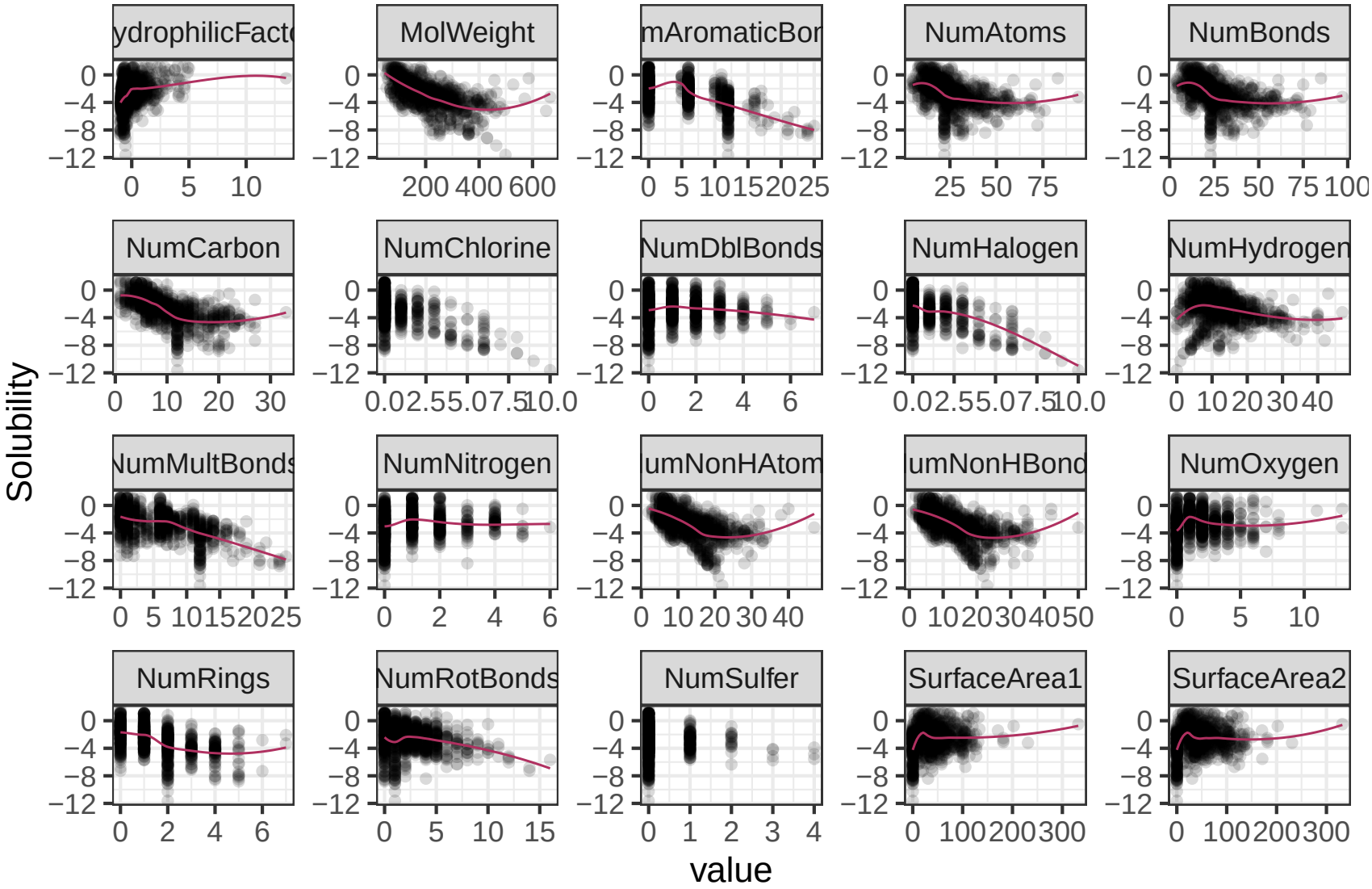
We'll take a look just at the training data for now. (Why?)

Distribution of Response

We'll take a look just at the training data for now. (Why?)



Pairwise Scatterplots



Correlation Matrix

```
library(GGally)
ggcorr(solTrain, hjust = 1, size = 2, layout.exp = 5)
```

Collinearity

- What are downsides of fitting the full model?

Collinearity

- What are downsides of fitting the full model?
- Let's do it anyway!

Model Summary

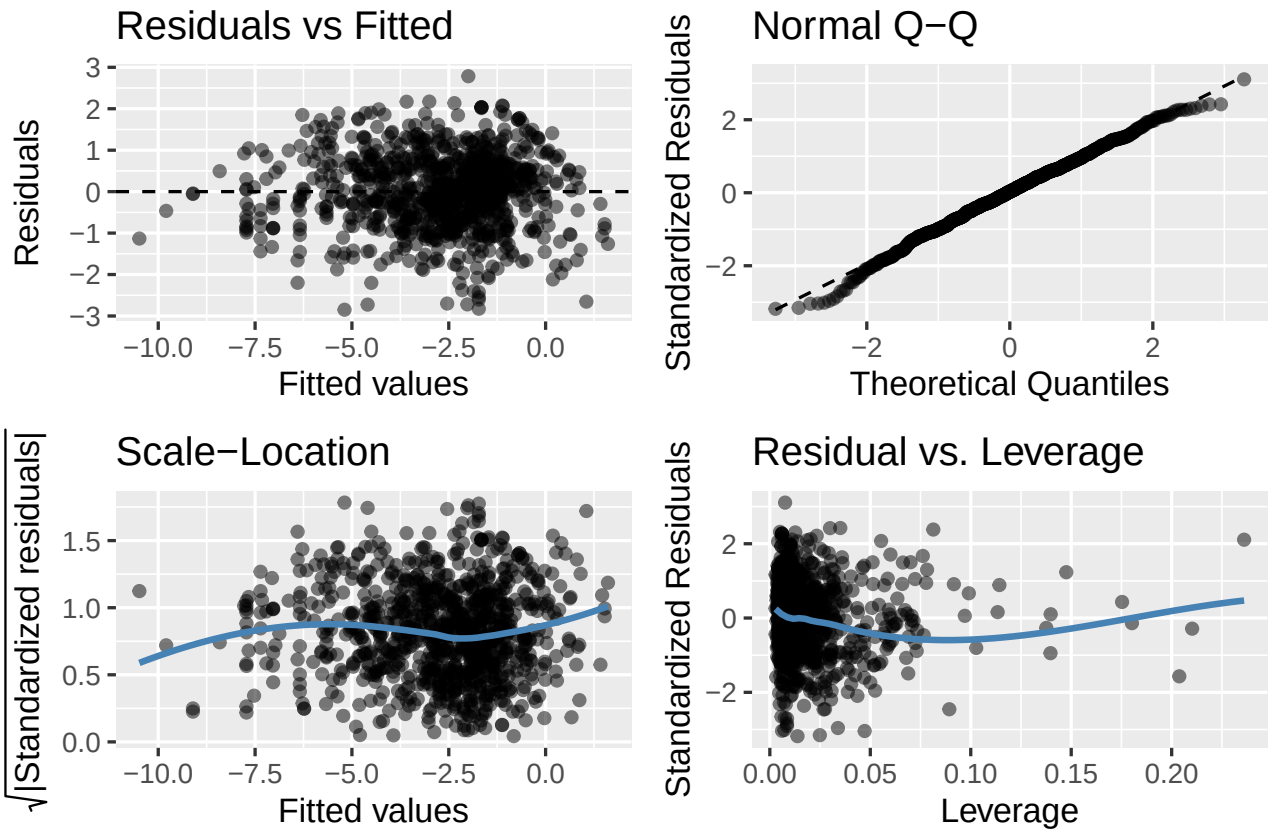
```
##
## Call:
## lm(formula = Solubility ~ ., data = solTrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8499 -0.5963  0.0232  0.5842  2.7848
##
## Coefficients: (3 not defined because of singularities)
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.344876   0.149393   2.309 0.021189 *
## MolWeight         -0.008074   0.001325  -6.093 1.61e-09 ***
## NumAtoms           0.275577   0.086182   3.198 0.001432 **
## NumNonHAtoms       1.536062   0.450948   3.406 0.000687 ***
## NumBonds          -0.612747   0.127856  -4.792 1.92e-06 ***
## NumNonHBonds             NA         NA      NA       NA
## NumMultBonds      -1.694110   0.321514  -5.269 1.70e-07 ***
## NumRotBonds       -0.147637   0.026894  -5.490 5.19e-08 ***
## NumDblBonds        0.771793   0.234853   3.286 0.001053 **
## NumAromaticBonds   1.278539   0.277614   4.605 4.69e-06 ***
## NumHydrogen              NA         NA      NA       NA
## NumCarbon         -0.650678   0.331825  -1.961 0.050187 .
## NumNitrogen       -0.222086   0.373396  -0.595 0.552140
## NumOxygen         -0.300338   0.424632  -0.707 0.479563
## NumSulfer          0.621244   0.298101   2.084 0.037432 *
## NumChlorine       -0.374042   0.061636  -6.069 1.87e-09 ***
## NumHalogen        -1.579937   0.459350  -3.440 0.000609 ***
## NumRings                 NA         NA      NA       NA
## HydrophilicFactor  0.162663   0.073229   2.221 0.026570 *
## SurfaceArea1       0.047692   0.013827   3.449 0.000587 ***
## SurfaceArea2      -0.070007   0.013245  -5.285 1.56e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9044 on 933 degrees of freedom
## Multiple R-squared:  0.8082,  Adjusted R-squared:  0.8047
```

Model Summary

```
##
## Call:
## lm(formula = Solubility ~ . - NumNonHBonds - NumHydrogen - NumRings,
##     data = solTrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8499 -0.5963  0.0232  0.5842  2.7848
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.344876   0.149393   2.309 0.021189 *
## MolWeight         -0.008074   0.001325  -6.093 1.61e-09 ***
## NumAtoms           0.275577   0.086182   3.198 0.001432 **
## NumNonHAtoms       1.536062   0.450948   3.406 0.000687 ***
## NumBonds          -0.612747   0.127856  -4.792 1.92e-06 ***
## NumMultBonds      -1.694110   0.321514  -5.269 1.70e-07 ***
## NumRotBonds       -0.147637   0.026894  -5.490 5.19e-08 ***
## NumDblBonds        0.771793   0.234853   3.286 0.001053 **
## NumAromaticBonds   1.278539   0.277614   4.605 4.69e-06 ***
## NumCarbon         -0.650678   0.331825  -1.961 0.050187 .
## NumNitrogen       -0.222086   0.373396  -0.595 0.552140
## NumOxygen         -0.300338   0.424632  -0.707 0.479563
## NumSulfer          0.621244   0.298101   2.084 0.037432 *
## NumChlorine       -0.374042   0.061636  -6.069 1.87e-09 ***
## NumHalogen        -1.579937   0.459350  -3.440 0.000609 ***
## HydrophilicFactor  0.162663   0.073229   2.221 0.026570 *
## SurfaceArea1       0.047692   0.013827   3.449 0.000587 ***
## SurfaceArea2      -0.070007   0.013245  -5.285 1.56e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9044 on 933 degrees of freedom
## Multiple R-squared:  0.8082, Adjusted R-squared:  0.8047
## F-statistic: 231.3 on 17 and 933 DF,  p-value: < 2.2e-16
```

Model Diagnostics

```
library(gglm)
gglm(sol_mod)
```



Section 2

Subset Selection

Methodology

Suppose we wish to find a linear model for Y with p predictors $X_1, \ldots, X_p$. How do we determine the optimal collection of predictors?

Methodology

Suppose we wish to find a linear model for Y with p predictors $X_1, \ldots, X_p$. How do we determine the optimal collection of predictors?

- First, determine an appropriate selection criteria.

Methodology

Suppose we wish to find a linear model for Y with p predictors $X_1, \ldots, X_p$. How do we determine the optimal collection of predictors?

- First, determine an appropriate selection criteria.
  - Cross-validation: Computationally expensive, but likely most accurate

Methodology

Suppose we wish to find a linear model for Y with p predictors $X_1, \ldots, X_p$. How do we determine the optimal collection of predictors?

- First, determine an appropriate selection criteria.
  - Cross-validation: Computationally expensive, but likely most accurate
  - Validation set: Subject to variability in test/training split (but ok for large data)

Methodology

Suppose we wish to find a linear model for Y with p predictors $X_1, \ldots, X_p$. How do we determine the optimal collection of predictors?

- First, determine an appropriate selection criteria.
  - Cross-validation: Computationally expensive, but likely most accurate
  - Validation set: Subject to variability in test/training split (but ok for large data)
  - Adjusted R^2: Penalizes non-helpful predictors, but may overestimate test error rate.

Methodology

Suppose we wish to find a linear model for Y with p predictors $X_1, \ldots, X_p$. How do we determine the optimal collection of predictors?

- First, determine an appropriate selection criteria.
  - Cross-validation: Computationally expensive, but likely most accurate
  - Validation set: Subject to variability in test/training split (but ok for large data)
  - Adjusted R^2: Penalizes non-helpful predictors, but may overestimate test error rate.
  - C_p: penalizes training RSS by typical discrepancy between test and training.

$$C_p = \frac{1}{n}(\mathrm{RSS} + 2d\hat{\sigma}^2)$$

Methodology

Suppose we wish to find a linear model for Y with p predictors $X_1, \ldots, X_p$. How do we determine the optimal collection of predictors?

- First, determine an appropriate selection criteria.
  - Cross-validation: Computationally expensive, but likely most accurate
  - Validation set: Subject to variability in test/training split (but ok for large data)
  - Adjusted R^2: Penalizes non-helpful predictors, but may overestimate test error rate.
  - C_p: penalizes training RSS by typical discrepancy between test and training.

$$C_p = \frac{1}{n}(\mathrm{RSS} + 2d\hat{\sigma}^2)$$

  - Akaike information criterion (AIC): uses method of maximum likelihood, assuming Normal errors

$$\mathrm{AIC} = \frac{1}{n\hat{\sigma}^2}(\mathrm{RSS} + 2d\hat{\sigma}^2)$$

Methodology

Suppose we wish to find a linear model for Y with p predictors $X_1, \ldots, X_p$. How do we determine the optimal collection of predictors?

- First, determine an appropriate selection criteria.
  - Cross-validation: Computationally expensive, but likely most accurate
  - Validation set: Subject to variability in test/training split (but ok for large data)
  - Adjusted R^2: Penalizes non-helpful predictors, but may overestimate test error rate.
  - C_p: penalizes training RSS by typical discrepancy between test and training.

$$C_p = \frac{1}{n}(\text{RSS} + 2d\hat{\sigma}^2)$$

  - Akaike information criterion (AIC): uses method of maximum likelihood, assuming Normal errors

$$\text{AIC} = \frac{1}{n\hat{\sigma}^2}(\text{RSS} + 2d\hat{\sigma}^2)$$

  - Bayesian information criterion (BIC): uses method of maximum likelihood and Bayes' Rule

$$\text{BIC} = \frac{1}{n\hat{\sigma}^2}(\text{RSS} + \ln n d\hat{\sigma}^2)$$

Best Subset

With p predictors, there are a total of 2^p possible MLR models.

- There are $\binom{p}{k}$ models using exactly k of p predictors

Best Subset

With p predictors, there are a total of 2^p possible MLR models.

- There are $\binom{p}{k}$ models using exactly k of p predictors

Theoretically, we can find the best model by fitting each possible model and selecting the best via appropriate selection criteria (C_p, AIC, BIC, R^2, CV)

Best Subset

With p predictors, there are a total of 2^p possible MLR models.

- There are $\binom{p}{k}$ models using exactly k of p predictors

Theoretically, we can find the best model by fitting each possible model and selecting the best via appropriate selection criteria (C_p, AIC, BIC, R^2, CV)

Downsides?

Best Subset

With p predictors, there are a total of 2^p possible MLR models.

- There are $\binom{p}{k}$ models using exactly k of p predictors

Theoretically, we can find the best model by fitting each possible model and selecting the best via appropriate selection criteria (C_p, AIC, BIC, R^2, CV)

Downsides?

- Computation time and storage grows exponentially in p

Best Subset

With p predictors, there are a total of 2^p possible MLR models.

- There are $\binom{p}{k}$ models using exactly k of p predictors

Theoretically, we can find the best model by fitting each possible model and selecting the best via appropriate selection criteria (C_p, AIC, BIC, R^2, CV)

Downsides?

- Computation time and storage grows exponentially in p
- May have low marginal improvement despite number of models fitted

Best Subset

With p predictors, there are a total of 2^p possible MLR models.

- There are $\binom{p}{k}$ models using exactly k of p predictors

Theoretically, we can find the best model by fitting each possible model and selecting the best via appropriate selection criteria (C_p, AIC, BIC, R^2, CV)

Downsides?

- Computation time and storage grows exponentially in p
- May have low marginal improvement despite number of models fitted
- We are performing a large number of *tests*, which corresponds to a relatively flexible model. Likely to overfit.

Best Subset in R

We use the `regsubsets` function in the `leaps` library.

Best Subset in R

We use the `regsubsets` function in the `leaps` library.

- `regsubsets` uses the same syntax as `lm`. The `summary` function outputs the best set of variables for the given number of predictors, across the range supplied

Best Subset in R

We use the `regsubsets` function in the `leaps` library.

- `regsubsets` uses the same syntax as `lm`. The `summary` function outputs the best set of variables for the given number of predictors, across the range supplied
- Be default, `regsubsets` only returns up to the best eight models. But `nvmax` can be used to return as many variables as desired

Best Subset in R

We use the `regsubsets` function in the `leaps` library.

- `regsubsets` uses the same syntax as `lm`. The `summary` function outputs the best set of variables for the given number of predictors, across the range supplied
- Be default, `regsubsets` only returns up to the best eight models. But `nvmax` can be used to return as many variables as desired
- The best model for each number of predictors is determined by *RSS*

Best Subset in R

We use the `regsubsets` function in the `leaps` library.

- `regsubsets` uses the same syntax as `lm`. The `summary` function outputs the best set of variables for the given number of predictors, across the range supplied
- Be default, `regsubsets` only returns up to the best eight models. But `nvmax` can be used to return as many variables as desired
- The best model for each number of predictors is determined by *RSS*
- The `regsubsets` function returns *RSS*, R^2, C_p, *AIC*, *BIC* for the best model of each number of predicts.

Best Subset in R

We use the `regsubsets` function in the `leaps` library.

- `regsubsets` uses the same syntax as `lm`. The `summary` function outputs the best set of variables for the given number of predictors, across the range supplied
- Be default, `regsubsets` only returns up to the best eight models. But `nvmax` can be used to return as many variables as desired
- The best model for each number of predictors is determined by *RSS*
- The `regsubsets` function returns *RSS*, R^2, C_p, *AIC*, *BIC* for the best model of each number of predicts.
- The **overall** best model can be selected using any of these criteria.

Best Subset in R

We use the `regsubsets` function in the `leaps` library.

- `regsubsets` uses the same syntax as `lm`. The `summary` function outputs the best set of variables for the given number of predictors, across the range supplied
- Be default, `regsubsets` only returns up to the best eight models. But `nvmax` can be used to return as many variables as desired
- The best model for each number of predictors is determined by *RSS*
- The `regsubsets` function returns *RSS*, R^2, C_p, *AIC*, *BIC* for the best model of each number of predicts.
- The **overall** best model can be selected using any of these criteria.
- Why does `regsubsets` only use *RSS* to determine best model for each number predictors?

Using regsubsets

```
library(leaps)
best_subset<-regsubsets(Solubility~.-NumNonHBonds -NumHydrogen -NumRings,
                        data = solTrain, nvmax = 17)
```

Using regsubsets

```
library(leaps)
best_subset<-regsubsets(Solubility~.-NumNonHBonds -NumHydrogen -NumRings,
                        data = solTrain, nvmax = 17)
```

- The regsubsets function itself outputs a special regsubsets object, which contains data but is not user-accessible.

Using regsubsets

```
library(leaps)
best_subset<-regsubsets(Solubility~.-NumNonHBonds -NumHydrogen -NumRings,
                        data = solTrain, nvmax = 17)
```

- The regsubsets function itself outputs a special regsubsets object, which contains data but is not user-accessible.
- We'll use the summary function, which provides the following elements:

Using regsubsets

```
library(leaps)
best_subset<-regsubsets(Solubility~.-NumNonHBonds -NumHydrogen -NumRings,
                        data = solTrain, nvmax = 17)
```

- The `regsubsets` function itself outputs a special `regsubsets` object, which contains data but is not user-accessible.
- We'll use the `summary` function, which provides the following elements:
  - `which`: a list of which predictors are in each model
  - `outmat`: a version of `which` for printing
  - Several metrics: `rsq`, `rss`, `adjr2`, `cp`, `bic`

Summary of regsubsets

- Stars indicate variable is included in model.
- For readability, I've only shown models with 5 or fewer variables

```
summary(best_subset)$outmat
```

```
##          MolWeight NumAtoms NumNonHAtoms NumBonds NumMultBonds NumRotBonds
## 1  ( 1 ) "*"       " "      " "          " "      " "          " "
## 2  ( 1 ) "*"       " "      " "          " "      " "          " "
## 3  ( 1 ) "*"       " "      " "          " "      "*"          " "
## 4  ( 1 ) " "       " "      "*"          " "      " "          " "
## 5  ( 1 ) " "       " "      "*"          "*"      " "          "*"
##          NumDblBonds NumAromaticBonds NumCarbon NumNitrogen NumOxygen NumSulfer
## 1  ( 1 ) " "         " "              " "       " "         " "       " "
## 2  ( 1 ) " "         " "              " "       " "         " "       " "
## 3  ( 1 ) " "         " "              " "       " "         " "       " "
## 4  ( 1 ) " "         " "              "*"       "*"         "*"       " "
## 5  ( 1 ) " "         " "              " "       "*"         "*"       " "
##          NumChlorine NumHalogen HydrophilicFactor SurfaceArea1 SurfaceArea2
## 1  ( 1 ) " "         " "         " "               " "          " "
## 2  ( 1 ) " "         " "         " "               "*"          " "
## 3  ( 1 ) " "         " "         " "               "*"          " "
## 4  ( 1 ) " "         " "         " "               " "          " "
## 5  ( 1 ) " "         " "         " "               " "          " "
```

Other Selection Metrics

The summary function can return selection metrics for each model.

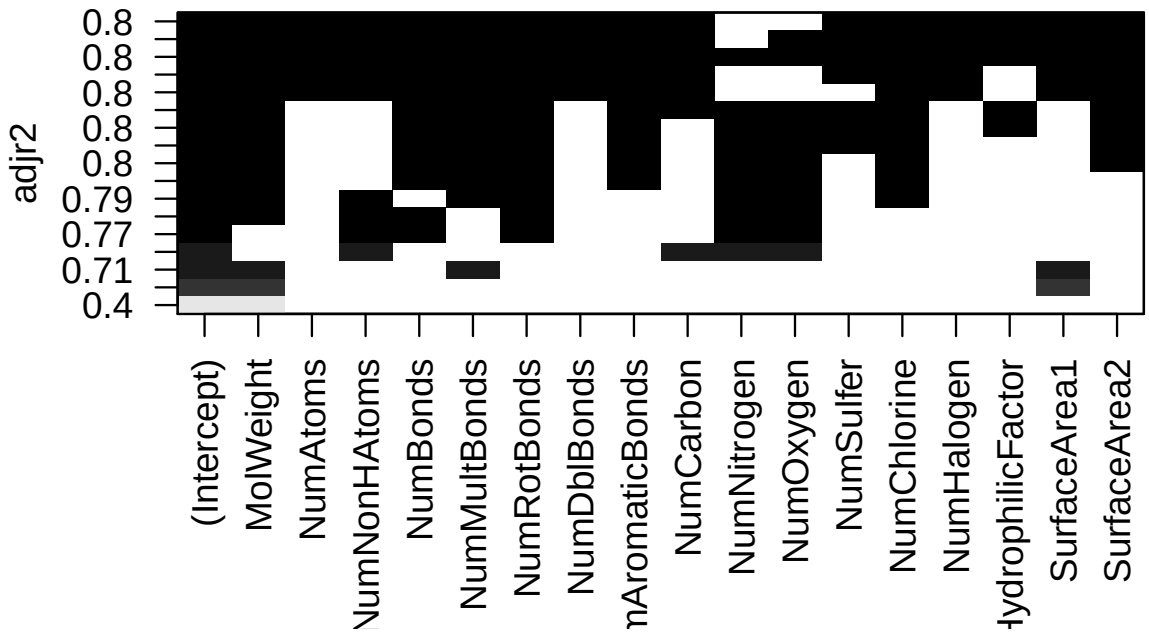
```
d <- data.frame(model = 1:17,
  adjr2 = summary(best_subset)$adjr2,
  rss = summary(best_subset)$rss,
  cp = summary(best_subset)$cp,
  bic = summary(best_subset)$bic)
d %>% head()
```

```
##   model     adjr2       rss        cp        bic
## 1     1 0.3952106 2404.1073 1992.4929  -465.5206
## 2     2 0.6590876 1353.7381  710.2104 -1004.8309
## 3     3 0.7120856 1142.0806  453.4176 -1159.6606
## 4     4 0.7447217 1011.5526  295.8216 -1268.2214
## 5     5 0.7742668  893.5334  153.5199 -1379.3431
## 6     6 0.7813296  864.6602  120.2167 -1403.7232
```

Vizualizing Variables

The variables present can also be plotted directly using `plot`:

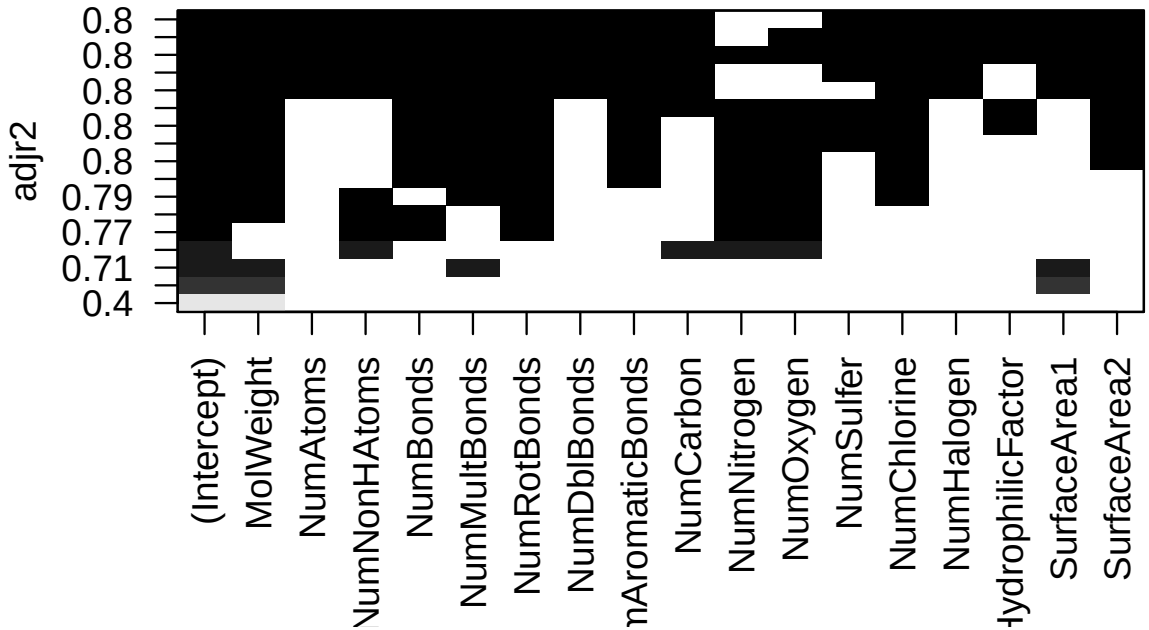
```
plot(best_subset, scale = "adjr2")
```



Vizualizing Variables

The variables present can also be plotted directly using `plot`:

```
plot(best_subset, scale = "adjr2")
```

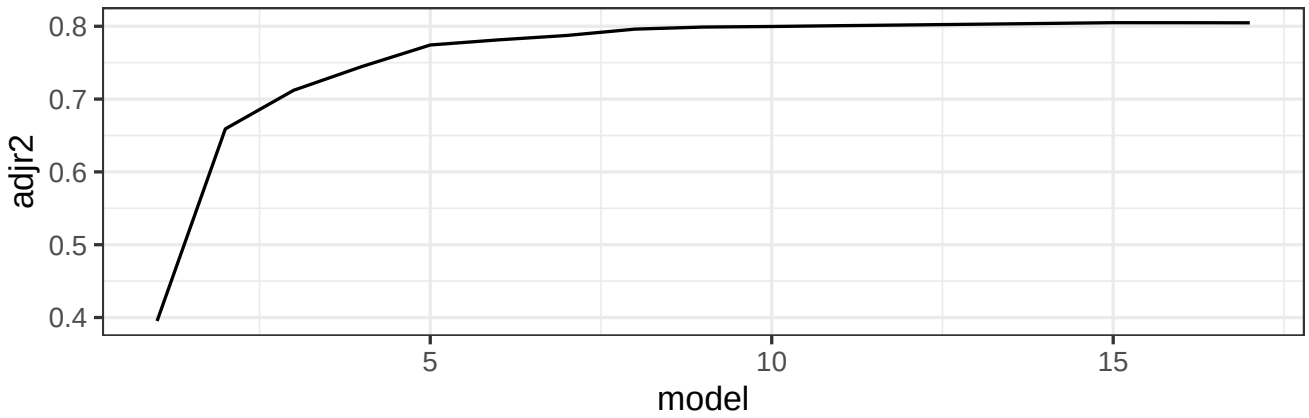


- Models are ordered by by selection statistic. Dark rectangles indicate variable presence

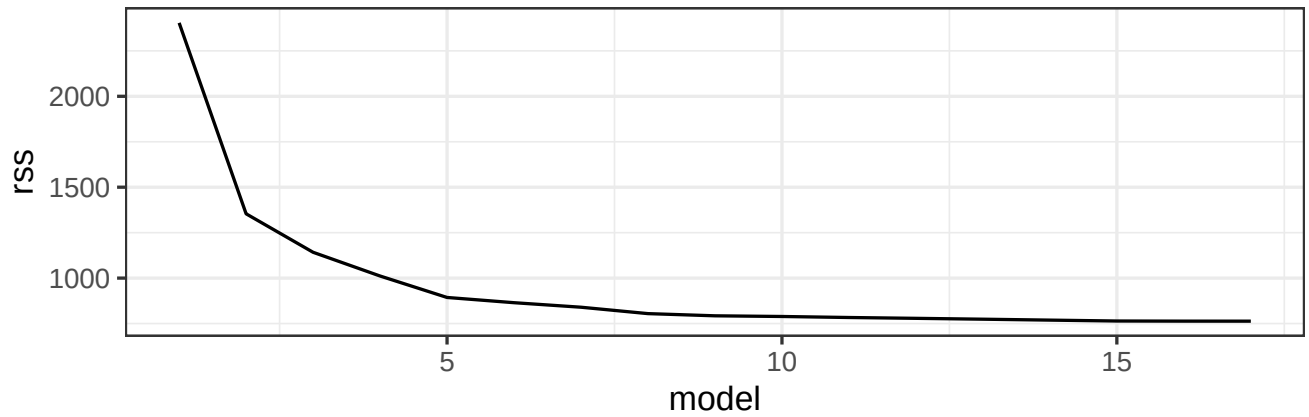
Plotting

We can use ggplot2 to visualize selection metric as a function of variable number

```
ggplot(d, aes(x = model, y = adjr2))+geom_line()+theme_bw()
```

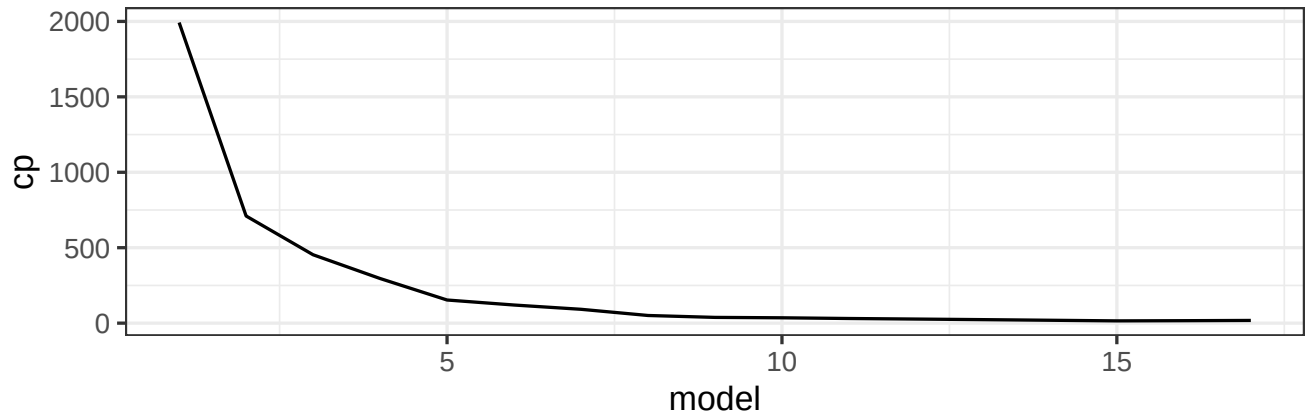



```
ggplot(d, aes(x = model, y = rss))+geom_line()+theme_bw()
```

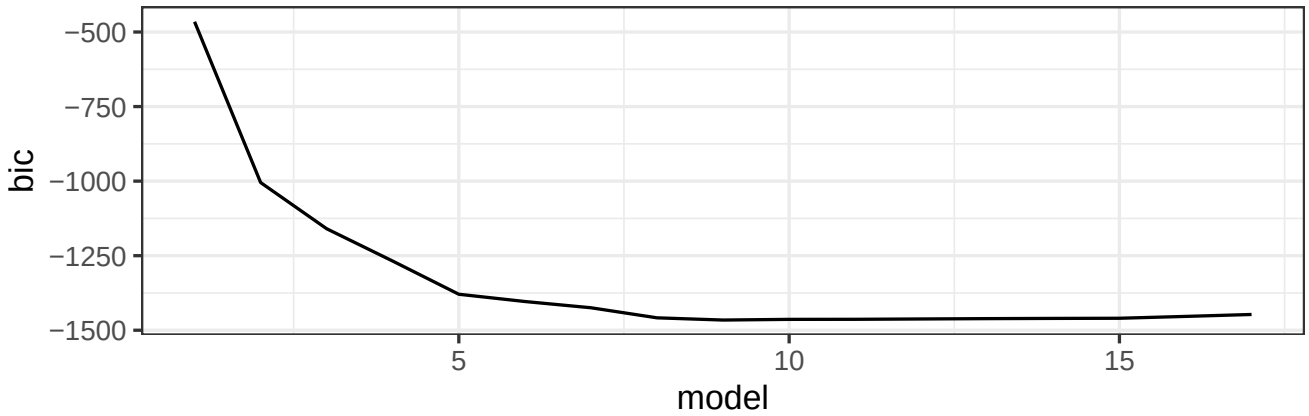


Plotting

```
ggplot(d, aes(x = model, y = cp))+geom_line()+theme_bw()
```




```
ggplot(d, aes(x = model, y = bic))+geom_line()+theme_bw()
```



Finding Best Subset

- To calculate the absolute best `cp`, `bic`, etc. we use either the `which.min` or `which.max` function

Finding Best Subset

- To calculate the absolute best `cp`, `bic`, etc. we use either the `which.min` or `which.max` function

```
adjr2.max <- which.max(summary(best_subset)$adjr2)
rss.min <- which.min(summary(best_subset)$rss)
cp.min <- which.min(summary(best_subset)$cp)
bic.min <- which.min(summary(best_subset)$bic)
data.frame(adjr2.max, rss.min, cp.min, bic.min)
```

```
##   adjr2.max rss.min cp.min bic.min
## 1        15      17     15       9
```

Finding Best Subset

- To calculate the absolute best `cp`, `bic`, etc. we use either the `which.min` or `which.max` function

```
adjr2.max <- which.max(summary(best_subset)$adjr2)
rss.min <- which.min(summary(best_subset)$rss)
cp.min <- which.min(summary(best_subset)$cp)
bic.min <- which.min(summary(best_subset)$bic)
data.frame(adjr2.max, rss.min, cp.min, bic.min)
```

```
##   adjr2.max rss.min cp.min bic.min
## 1        15      17     15       9
```

- So what model is best?

Finding Best Subset

- To calculate the absolute best `cp`, `bic`, etc. we use either the `which.min` or `which.max` function

```
adjr2.max <- which.max(summary(best_subset)$adjr2)
rss.min <- which.min(summary(best_subset)$rss)
cp.min <- which.min(summary(best_subset)$cp)
bic.min <- which.min(summary(best_subset)$bic)
data.frame(adjr2.max, rss.min, cp.min, bic.min)
```

```
##   adjr2.max rss.min cp.min bic.min
## 1        15      17     15       9
```

- So what model is best?
  - Usually the simplest model.

Model Coefficients

- To show coefficients associated with the model with lowest `bic`, use `coef`:

```
coef(best_subset, bic.min)
```

```
##      (Intercept)        MolWeight         NumBonds     NumMultBonds
##      0.179049978     -0.007776351     -0.042507435     -0.368292209
##      NumRotBonds NumAromaticBonds      NumNitrogen        NumOxygen
##     -0.138979290      0.225474767      0.628386933      0.782490751
##      NumChlorine     SurfaceArea2
##     -0.386474357     -0.008279467
```

Model Coefficients

- To show coefficients associated with the model with lowest `bic`, use `coef`:

```
coef(best_subset, bic.min)
```

```
##      (Intercept)        MolWeight         NumBonds     NumMultBonds
##      0.179049978     -0.007776351     -0.042507435     -0.368292209
##      NumRotBonds NumAromaticBonds      NumNitrogen        NumOxygen
##     -0.138979290      0.225474767      0.628386933      0.782490751
##      NumChlorine     SurfaceArea2
##     -0.386474357     -0.008279467
```

- And to get a vector of variable names, use `names`:

```
names(coef(best_subset, bic.min))
```

```
##  [1] "(Intercept)"      "MolWeight"        "NumBonds"         "NumMultBonds"
##  [5] "NumRotBonds"      "NumAromaticBonds" "NumNitrogen"      "NumOxygen"
##  [9] "NumChlorine"      "SurfaceArea2"
```

Forward Selection

Forward selection is a *computationally efficient* alternative to best subset

Forward Selection

Forward selection is a *computationally efficient* alternative to best subset

- To perform forward selection, create the best 1 variable model. Then create $p - 1$ new 2 variable models by adding each other predictor one-at-a-time to the existing 1-variable model. Repeat for 3 variables and so on.

Forward Selection

Forward selection is a *computationally efficient* alternative to best subset

- To perform forward selection, create the best 1 variable model. Then create $p-1$ new 2 variable models by adding each other predictor one-at-a-time to the existing 1-variable model. Repeat for 3 variables and so on.
- Compared to Best Subset, forward selection computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$

Forward Selection

Forward selection is a *computationally efficient* alternative to best subset

- To perform forward selection, create the best 1 variable model. Then create $p-1$ new 2 variable models by adding each other predictor one-at-a-time to the existing 1-variable model. Repeat for 3 variables and so on.
- Compared to Best Subset, forward selection computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Forward selection tends to favor parsimonous models

Forward Selection

Forward selection is a *computationally efficient* alternative to best subset

- To perform forward selection, create the best 1 variable model. Then create $p - 1$ new 2 variable models by adding each other predictor one-at-a-time to the existing 1-variable model. Repeat for 3 variables and so on.
- Compared to Best Subset, forward selection computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Forward selection tends to favor parsimonous models
- Downsides?

Forward Selection

Forward selection is a *computationally efficient* alternative to best subset

- To perform forward selection, create the best 1 variable model. Then create $p-1$ new 2 variable models by adding each other predictor one-at-a-time to the existing 1-variable model. Repeat for 3 variables and so on.
- Compared to Best Subset, forward selection computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Forward selection tends to favor parsimonous models
- Downsides?
  - Not guaranteed to find the best model (or even something close to the best model)

Forward Selection

Forward selection is a *computationally efficient* alternative to best subset

- To perform forward selection, create the best 1 variable model. Then create $p-1$ new 2 variable models by adding each other predictor one-at-a-time to the existing 1-variable model. Repeat for 3 variables and so on.
- Compared to Best Subset, forward selection computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Forward selection tends to favor parsimonous models
- Downsides?
  - Not guaranteed to find the best model (or even something close to the best model)
  - Early predictors may become redundant

Forward Selection

Forward selection is a *computationally efficient* alternative to best subset

- To perform forward selection, create the best 1 variable model. Then create $p-1$ new 2 variable models by adding each other predictor one-at-a-time to the existing 1-variable model. Repeat for 3 variables and so on.
- Compared to Best Subset, forward selection computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Forward selection tends to favor parsimonous models
- Downsides?
  - Not guaranteed to find the best model (or even something close to the best model)
  - Early predictors may become redundant
  - Can be unstable

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

- To perform backward selection, begin with full model. Then create $p-1$ new $p-1$ variable models by removing one-at-a-time each other predictor from the existing p-variable model. Repeat for $p-2$ variables and so on.

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

- To perform backward selection, begin with full model. Then create $p-1$ new $p-1$ variable models by removing one-at-a-time each other predictor from the existing p-variable model. Repeat for $p-2$ variables and so on.
- Compared to Best Subset, backward elimination computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

- To perform backward selection, begin with full model. Then create $p-1$ new $p-1$ variable models by removing one-at-a-time each other predictor from the existing p-variable model. Repeat for $p-2$ variables and so on.
- Compared to Best Subset, backward elimination computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Backward elimination tends to favor in-depth models

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

- To perform backward selection, begin with full model. Then create $p-1$ new $p-1$ variable models by removing one-at-a-time each other predictor from the existing p-variable model. Repeat for $p-2$ variables and so on.
- Compared to Best Subset, backward elimination computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Backward elimination tends to favor in-depth models
- Downsides?

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

- To perform backward selection, begin with full model. Then create $p-1$ new $p-1$ variable models by removing one-at-a-time each other predictor from the existing p-variable model. Repeat for $p-2$ variables and so on.
- Compared to Best Subset, backward elimination computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Backward elimination tends to favor in-depth models
- Downsides?
  - Not guaranteed to find the best model (or even something close to the best model)

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

- To perform backward selection, begin with full model. Then create $p-1$ new $p-1$ variable models by removing one-at-a-time each other predictor from the existing p-variable model. Repeat for $p-2$ variables and so on.
- Compared to Best Subset, backward elimination computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Backward elimination tends to favor in-depth models
- Downsides?
  - Not guaranteed to find the best model (or even something close to the best model)
  - Requires fewer predictors than observations

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

- To perform backward selection, begin with full model. Then create $p-1$ new $p-1$ variable models by removing one-at-a-time each other predictor from the existing p-variable model. Repeat for $p-2$ variables and so on.
- Compared to Best Subset, backward elimination computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Backward elimination tends to favor in-depth models
- Downsides?
  - Not guaranteed to find the best model (or even something close to the best model)
  - Requires fewer predictors than observations
  - Susceptible to multicollinearity

Backward Elimination

Backward Elimination is another *computationally efficient* alternative to best subset

- To perform backward selection, begin with full model. Then create $p-1$ new $p-1$ variable models by removing one-at-a-time each other predictor from the existing p-variable model. Repeat for $p-2$ variables and so on.
- Compared to Best Subset, backward elimination computation time grows polynomially in p: $\text{Num. Models} = 1 + \frac{p(p+1)}{2}$
- Backward elimination tends to favor in-depth models
- Downsides?
  - Not guaranteed to find the best model (or even something close to the best model)
  - Requires fewer predictors than observations
  - Susceptible to multicollinearity
  - Can be unstable

Forward/Backward Selection in R

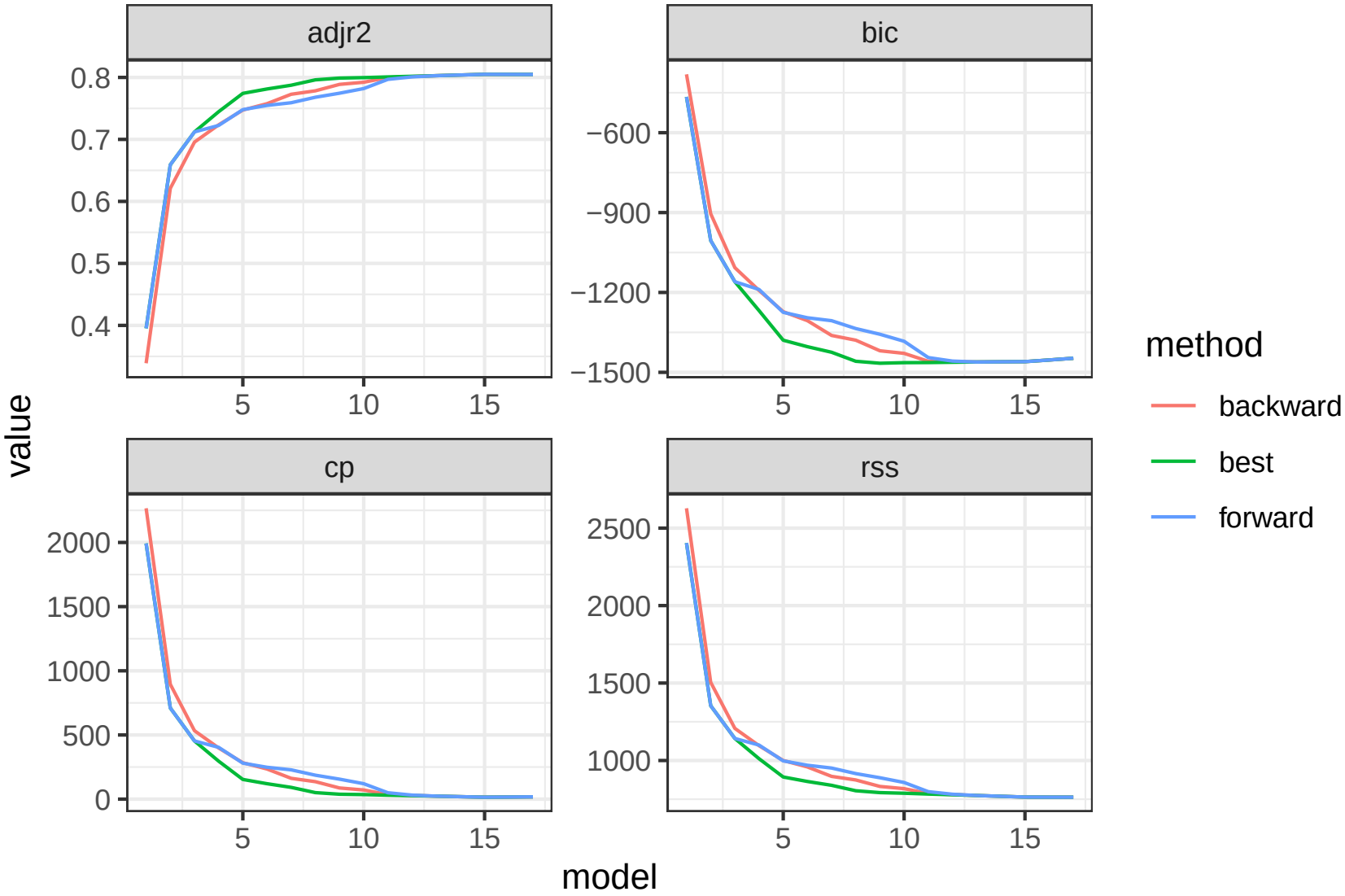
We again use the `regsubsets` function in the `leaps` library.

```
forward_select<-regsubsets(Solubility~.-NumNonHBonds -NumHydrogen -NumRings,
                           data = solTrain, nvmax = 17, method = "forward")

backward_elim<-regsubsets(Solubility~.-NumNonHBonds -NumHydrogen -NumRings,
                           data = solTrain, nvmax = 17, method = "backward")
```

- All of the same tools used for best subsets are available for forward and backward selection

Comparison of Models



Model Testing

- Let's go with 4 models, based on best subset (since we have it)
  - 5 variables (elbow of metric plots)
  - 9 variables (best bic)
  - 15 variables (best adjusted R^2)
  - 17 variables (the full model)
- We'll build each model on the training data, and then compute MSE on the test data.

```
## # A tibble: 4 x 2
##   model      mse
##   <chr>    <dbl>
## 1 model_15 0.928
## 2 model_9  0.966
## 3 model_5  1.13
## 4 model_17 4.31
```