Rule–Based Models for Regression and Classification

Meeting

Max Kuhn

Pfizer R&D

Outline

- Trees and rules
- Regression Models: M5 and Cubist
- Classification Models: C5.0, random forest and others

The notes and code are at `https://github.com/topepo/odsc_rules`.

Packages used *directly* in the notes are: C50, car caret, Cubist, ggmap, inTrees, pROC, and randomForest.

Tree–based Regression Models

Classification and Regression Trees (CART) are a framework for machine learning models.

A CART searches through each predictor to find a value of a single variable that best splits the data into two groups.

- typically, the best split minimizes the RMSE of the outcome in the resulting data subsets.

For the two resulting groups, the process is repeated until a hierarchical structure (a tree) is created.

- in effect, trees partition the X space into rectangular sections that assign a single value to samples within the rectangle.

To demonstrate, we'll walk through the first two iterations of this process.

Example Data

The data used to illustrate the models are sale prices of homes in Sacramento CA.

The original data were obtained from the website for the SpatialKey software. From their website:

> *The Sacramento real estate transactions file is a list of 985 real estate transactions in the Sacramento area reported over a five-day period, as reported by the Sacramento Bee.*

Google was used to fill in missing/incorrect data.

Example Data

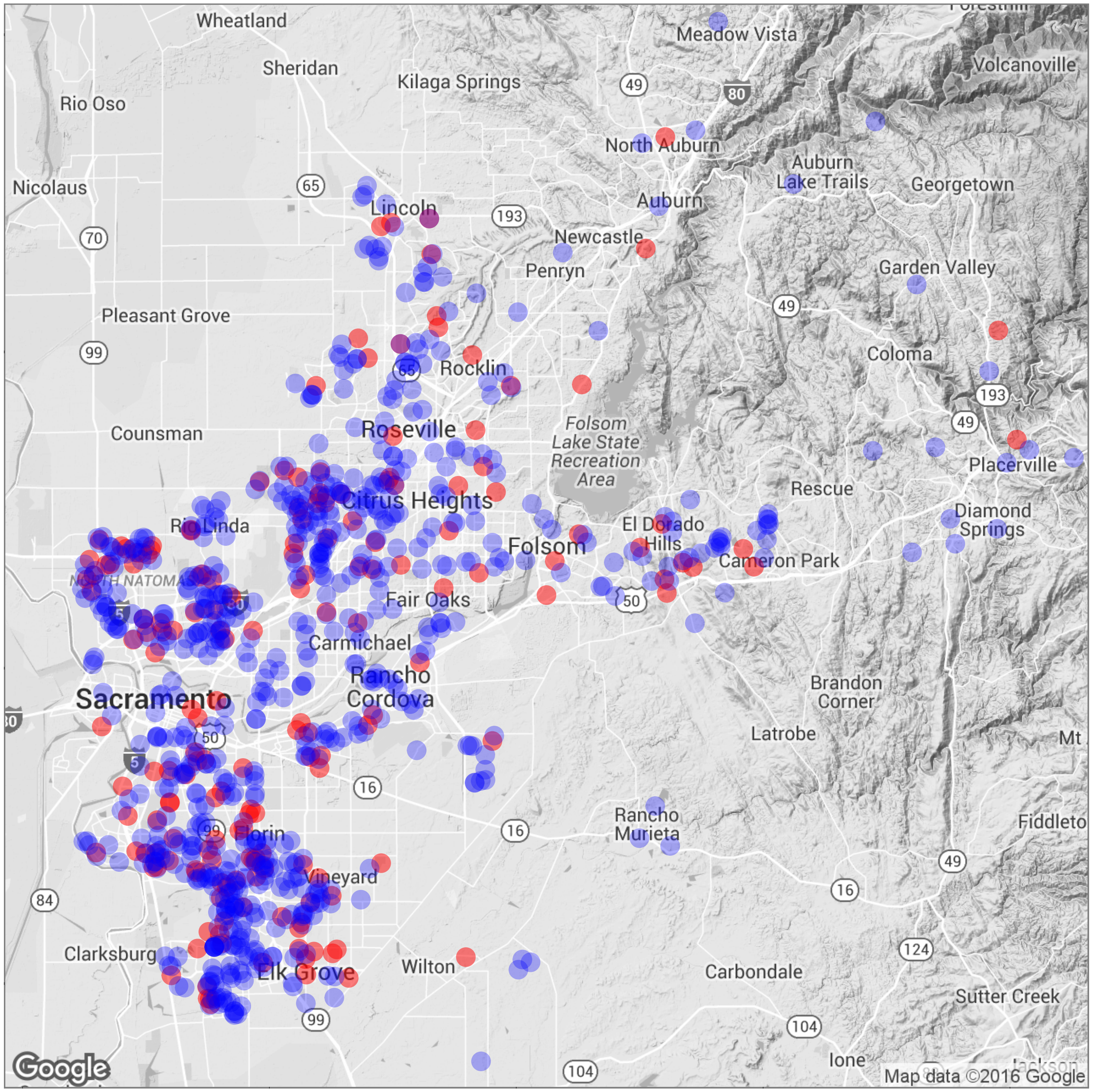
```
> library(caret)
> data(Sacramento)
> str(Sacramento, vec.len = 1)

'data.frame': 932 obs. of  9 variables:
 $ city     : Factor w/ 37 levels "ANTELOPE","AUBURN",..: 34 34 ...
 $ zip      : Factor w/ 68 levels "z95603","z95608",..: 64 52 ...
 $ beds     : int  2 3 ...
 $ baths    : num  1 1 ...
 $ sqft     : int  836 1167 ...
 $ type     : Factor w/ 3 levels "Condo","Multi_Family",..: 3 3 ...
 $ price    : int  59222 68212 ...
 $ latitude : num  38.6 ...
 $ longitude: num  -121 ...
```

Example Data

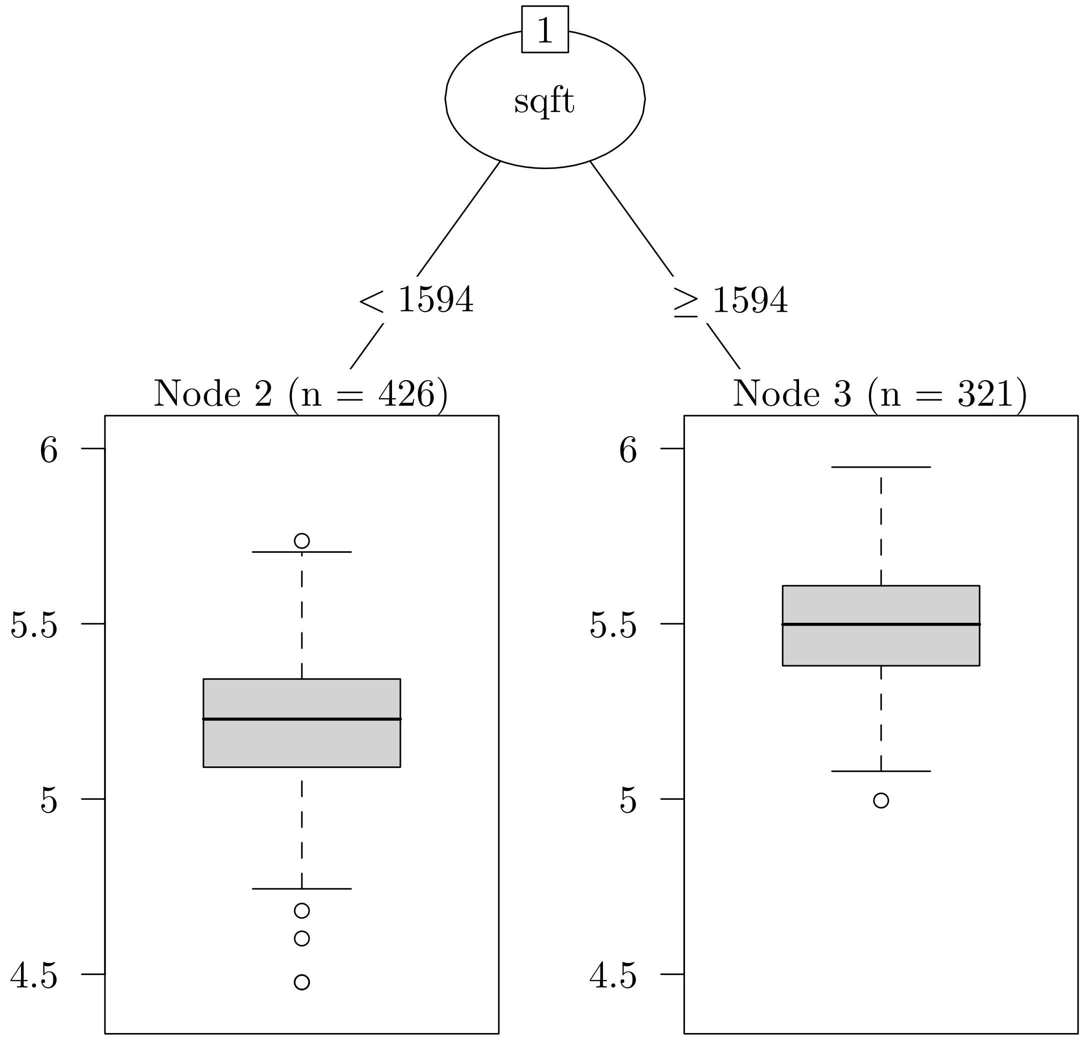
A random split was used to create a test set with 20% of the data. The data are:

```
> set.seed(955)
> in_train <- createDataPartition(log10(Sacramento$price), p = .8, list = FALSE)
>
> training <- Sacramento[ in_train,]
> testing  <- Sacramento[-in_train,]
> nrow(training)

[1] 747

> nrow(testing)

[1] 185
```

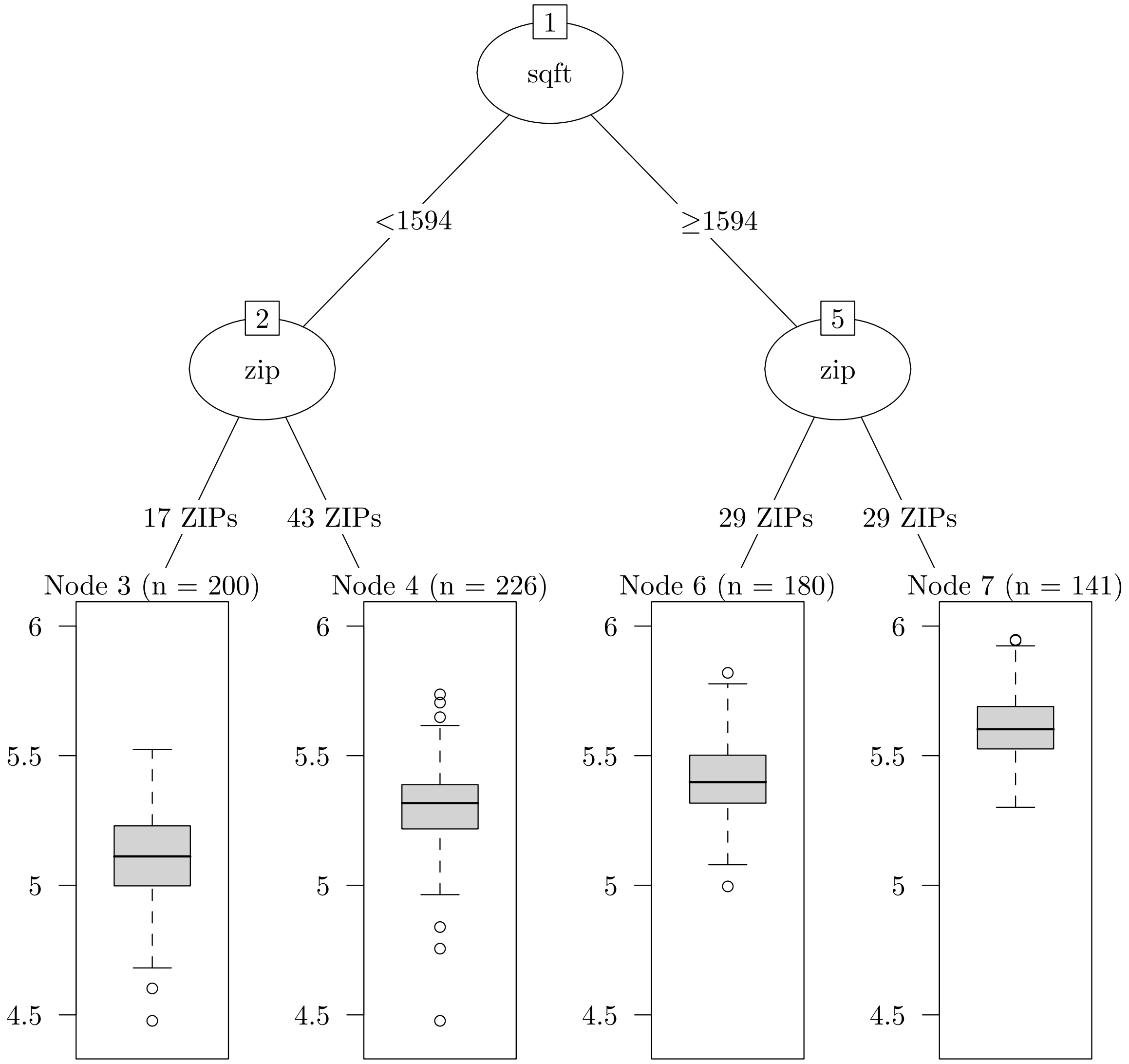
Training in Blue, Testing in Red



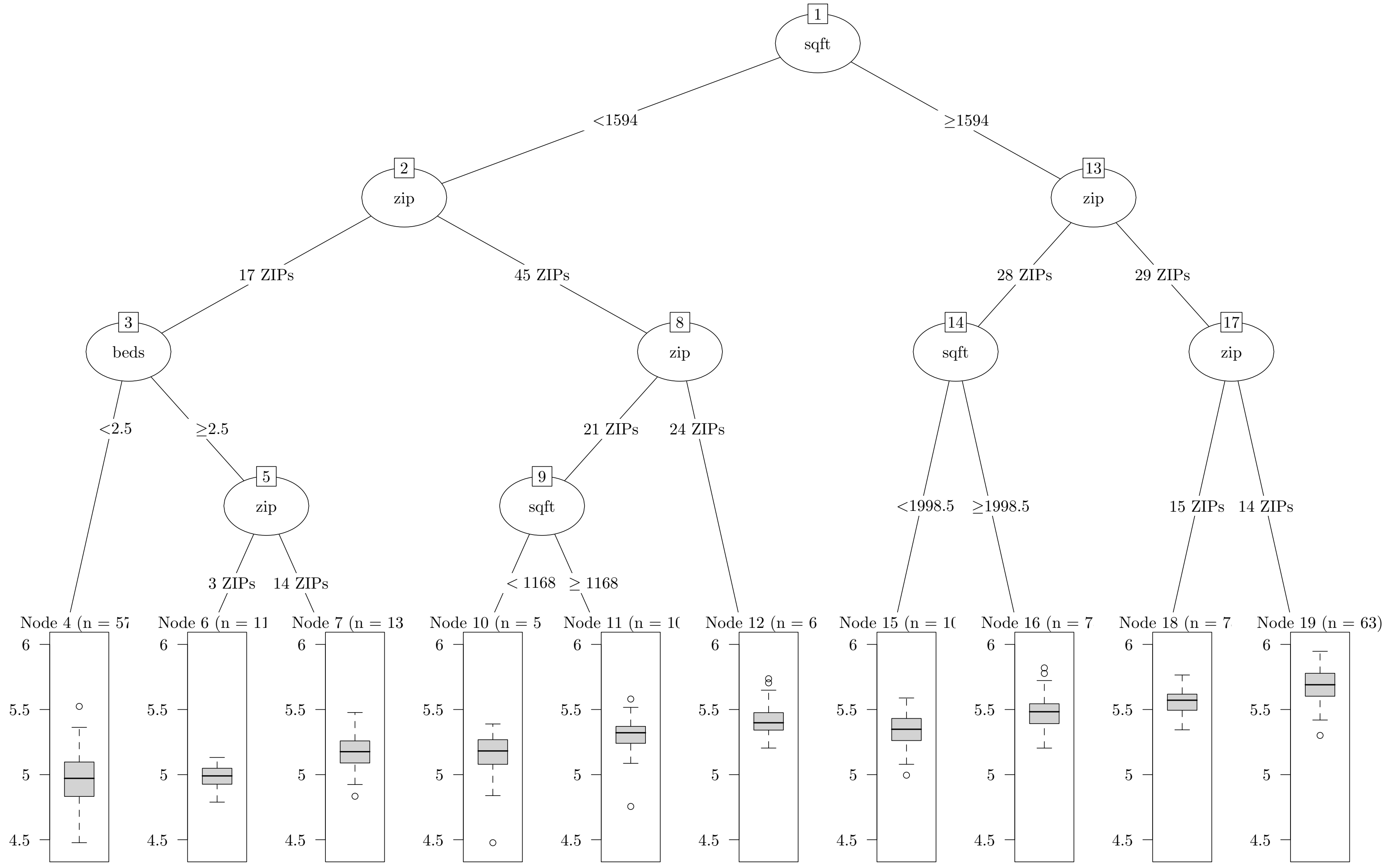
First Split of a CART Tree



Second Split



Full Tree



1
sqft
<1594
≥1594
2
zip
13
zip
17 ZIPs
45 ZIPs
28 ZIPs
29 ZIPs
3
beds
8
zip
14
sqft
17
zip
<2.5
≥2.5
21 ZIPs
24 ZIPs
5
zip
9
sqft
<1998.5
≥1998.5
15 ZIPs
14 ZIPs
3 ZIPs
14 ZIPs
< 1168
≥ 1168
Node 4 (n = 57
Node 6 (n = 11
Node 7 (n = 13
Node 10 (n = 5
Node 11 (n = 1(
Node 12 (n = 6
Node 15 (n = 1(
Node 16 (n = 7
Node 18 (n = 7
Node 19 (n = 63)
6
5.5
5
4.5

The Good and Bad of Trees

Trees can be computed very quickly and have simple interpretations.

Also, they have built-in feature selection; if a predictor was not used in any split, the model is completely independent of that data.

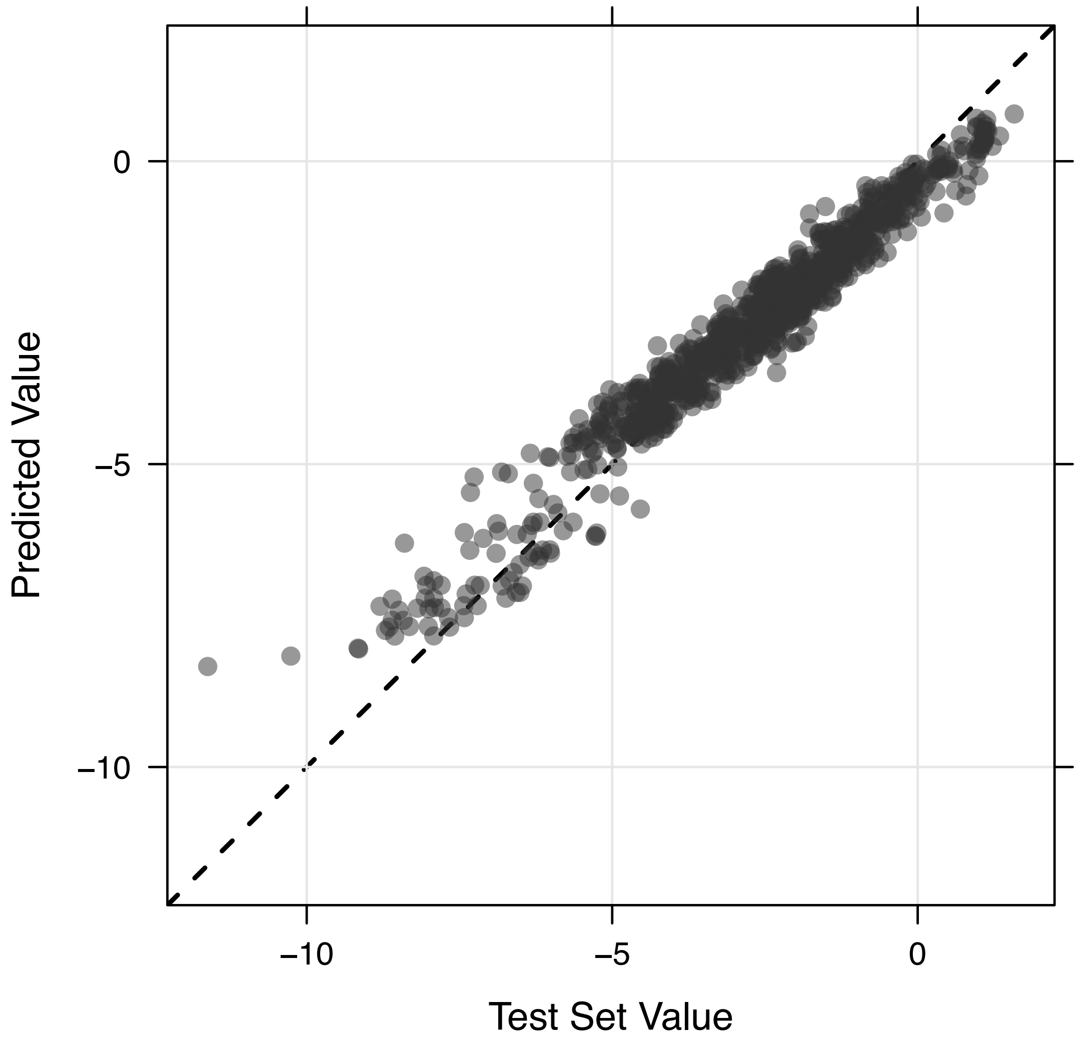
Unfortunately, trees do not usually have optimal performance when compared to other methods.

Also, small changes in the data can drastically affect the structure of a tree.

This last point has been exploited to improve the performance of trees via ensemble methods where many trees are fit and predictions are aggregated across the trees. Examples are bagging, boosting and random forests.

Trees may not fit the data well in the extremes of the outcome range.

Poor Fits in the Tails



Model Trees

The *model tree* approach described in Quinlan (1992) called M5, which is similar to regression trees except:

- the splitting criterion is different,
- the terminal nodes predict the outcome using a linear model (as opposed to the simple average), and
- when a sample is predicted, it is often a combination of the predictions from different models along the same path through the tree.

The main implementation of this technique is a "rational reconstruction" of this model called M5', which is described by Wang and Witten (1997) and is included in the Weka software package.

Model Tree Structure

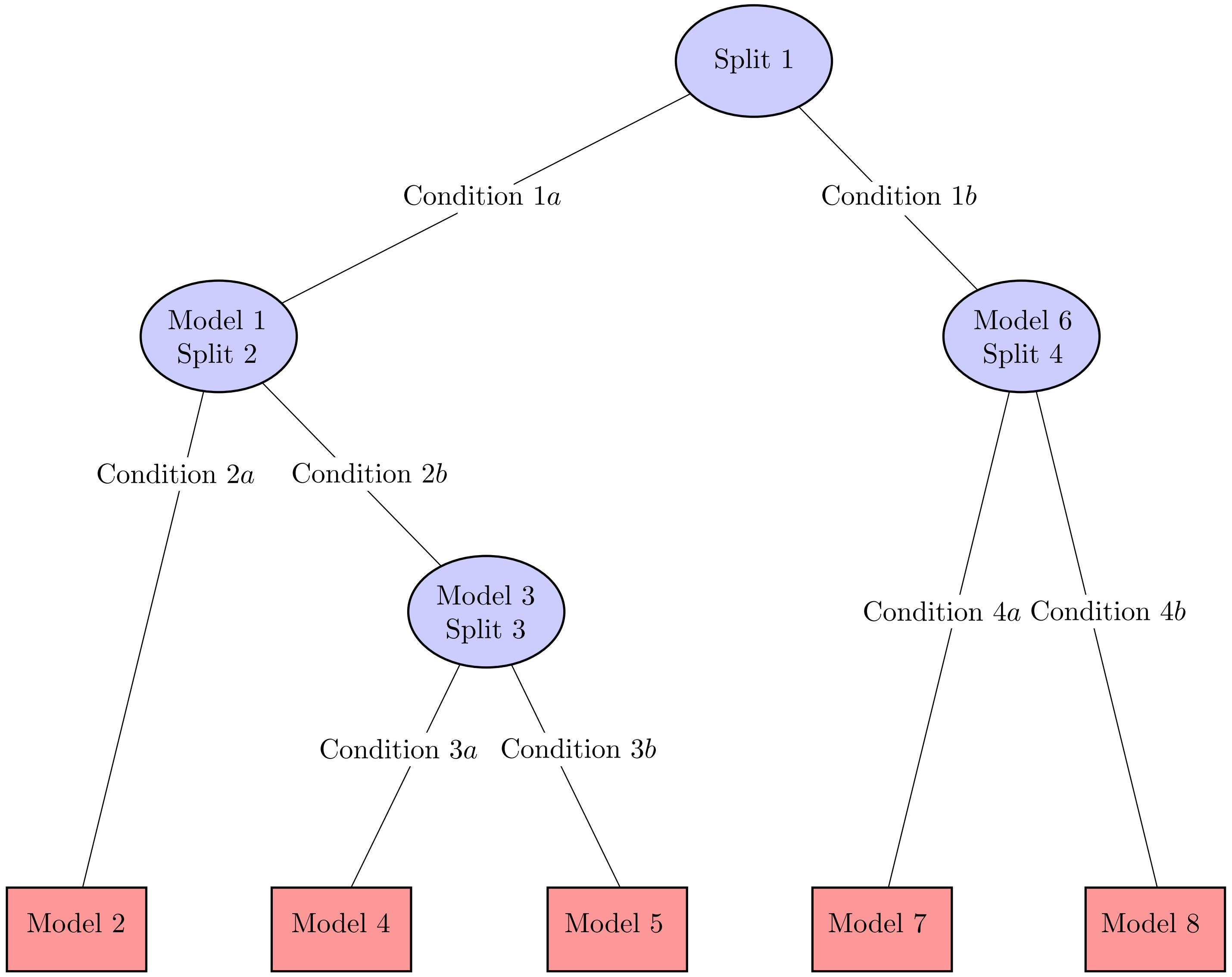
When model trees make a split of the data, they fit a linear model to the current subset using all the predictors involved in the splits along the path.

This process proceeds until there are not enough samples to split and/or fit the model.

A *pruning* stage is later used to simplify the model.

Note: Many of the models here are fit with and without encoding categorical predictors as dummy variables. Tree– and rule–based models usually do not require dummy variables.

Model Tree Structure



Model Tree Predictions

When a sample is predicted, all of the linear models along the path are combined using:

$$\widehat{y}_{par} = \frac{n_{kid}\,\widehat{y}_{kid} + c\,\widehat{y}_{par}}{n_{kid} + c}$$

$\widehat{y}_{kid}$ is the prediction from the child node
n_{kid} is the number of training set data points in the child node
$\widehat{y}_{par}$ is the prediction from the parent node
c is a constant with a default value of 15.

For the example data, the unpruned model had 81 paths through the tree and the pruned version used 2 paths.

From Trees to Rules

Tree–based models consist of one or more nested if-then statements for the predictors that partition the data.

Within these partitions, a model is used to predict the outcome.

For example, a very simple tree could be defined as:

```
if >= 1.7 then
|  if X2 >= 202.1 then Y = 1.3
|  else Y = 5.6
else Y = 2.5
```

From Trees to Rules

Notice that the if-then statements generated by a tree define a unique route to one terminal node for any sample.

A *rule* is a set of if-then conditions (possibly created by a tree) that have been collapsed into independent conditions.

For the example above, there would be three rules:

```
if X1 >= 1.7 & X2 >= 202.1 then Y = 1.3
if X1 >= 1.7 & X2 <  202.1 then Y = 5.6
if X1 <  1.7               then Y = 2.5
```

Rules can be simplified or pruned in a way that samples are covered by multiple rules, eg.

```
if X1 >= 1.7
```

Rule–Based Models

One path to a terminal node in an unpruned model is

```
sqft <= 1594 &
zip not in {z95631, z95833, z95758, z95670, 45 others} &
beds <= 2.5 &
latitude >  38.543 &
latitude >  38.615 &
latitude >  38.637 &
latitude <= 38.688 &
latitude <= 38.673
```

We can convert our model tree to a rule–based model. Many conditions can be simplified

"Separate and Conquer" Approach to Rules

First, an initial model tree is created and only the rule with the largest coverage is saved from this model.

The samples covered by the rule are removed from the training set and another model tree is created with the remaining data.

Again, only the rule with the maximum coverage is retained.

This process repeats until all the training set data has been covered by at least one rule.

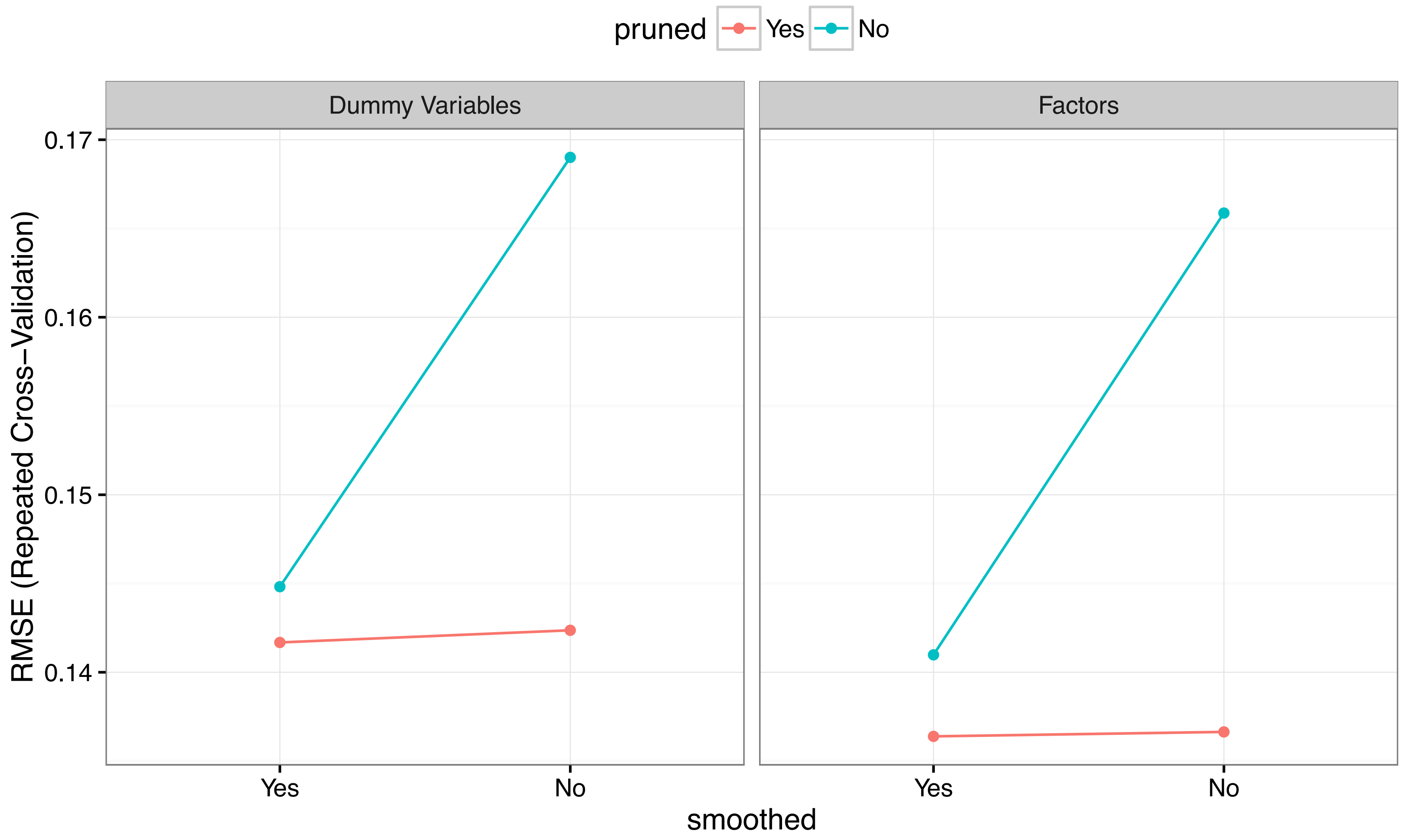
A new sample is predicted by determining which rule(s) it falls under then applies the linear model associated with the largest coverage.

For our data, the unpruned model has 81 and can be reduced shown to two rules based on `sqft` $\leq$ 1594.

An Example of a Terminal Node Model

```
log10(price) =
   - 6.3388
   - 0.0032 * city in {GALT, POLLOCK_PINES, ..., GRANITE_BAY}
   + 0.0209 *  zip in {z95820, z95822, z95626, ..., z95746}
   + 0.0015 *  zip in {z95673, z95832, z95621, ..., z95746}
   + 0.0098 *  zip in {z95631, z95833, z95758, ..., z95746}
   + 0.0091 *  zip in {z95818, z95608, z95662, ..., z95746}
   + 0.0033 *  zip in {z95814, z95765, z95667, ..., z95746}
   + 0.0005 * beds
   + 0.0001 * sqft
   + 0.3097 * latitude
   + 0.0056 * longitude
```

Effect of Smoothing and Pruning Results



Model Trees in R

```
> library(RWeka)
> model_tree <- M5P(log10(price) ~ ., data = training,
+                   ## Make the minimum number of instances per
+                   ## leaf higher than the default of 4
+                   control = Weka_control(M = 15))
>
> model_tree_unpruned <- M5P(log10(price) ~ ., data = training,
+                            control = Weka_control(M = 15, N = TRUE))
```

Note that the formula method is used but factors are *not* converted to dummy variables.

Tuning Model Trees in R

```
> ctrl <- trainControl(method = "repeatedcv", repeats = 5)
>
> mt_grid <- expand.grid(rules = "Yes",
+                        pruned = c("No", "Yes"),
+                        smoothed = c("No", "Yes"))
>
> ## will use dummy variables:
> set.seed(139)
> mt_tune_dv <- train(log10(price) ~ ., data = training,
+                     method = "M5",
+                     tuneGrid = mt_grid,
+                     trControl = ctrl)
> ## will not:
> set.seed(139)
> mt_tune <- train(x = training[, -7], y = log10(training$price),
+                  method = "M5",
+                  tuneGrid = mt_grid,
+                  trControl = ctrl)
```

Setting the seed prior to each call ensures that the same resamples are used.

Cubist

Some specific differences between Cubist and the previously described approaches for model trees and their rule–based variants are:

- the specific techniques used for linear model smoothing, creating rules and pruning are different,
- an optional boosting–like procedure called *committees* can be used, and
- the predictions generated by the model rules can be adjusted using nearby points from the training set data.

We are indebted to the work of Chris Keefer, who extensively studied the Cubist source code to figure out the details.

Cubist

Cubist does not use the Separate and Conquer approach to creating rules from trees.

A single tree is created then "flattened" into a set of rules.

The pruning and smoothing procedures are similar to those implemented in M5, but . . .

Smoothing Models in Cubist

Cubist has a different formula for combining models up the tree:

$$\widehat{y}_{par} = a \times \widehat{y}_{kid} + (1 - a) \times \widehat{y}_{par}$$

where

$$a = \frac{Var(\widehat{y}_{par}) - b}{Var(\widehat{y}_{par}) + Var(\widehat{y}_{kid}) - 2b}$$

$$b = \frac{S_{11} - \frac{1}{n} S_1 S_2}{n - 1}$$

$$S_1 = \sum_{i=1}^{n} (y_i - \widehat{y}_{i\,par})$$

$$S_2 = \sum_{i=1}^{n} (y_i - \widehat{y}_{i\,kid})$$

$$S_{12} = \sum_{i=1}^{n} (y_i - \widehat{y}_{i\,kid})(y_i - \widehat{y}_{i\,par})$$

Cubist in R

```
> library(Cubist)
> cb <- cubist(x = training[, -7], y = log10(training$price))
> ## To see the rules + models
> summary(cb)
```

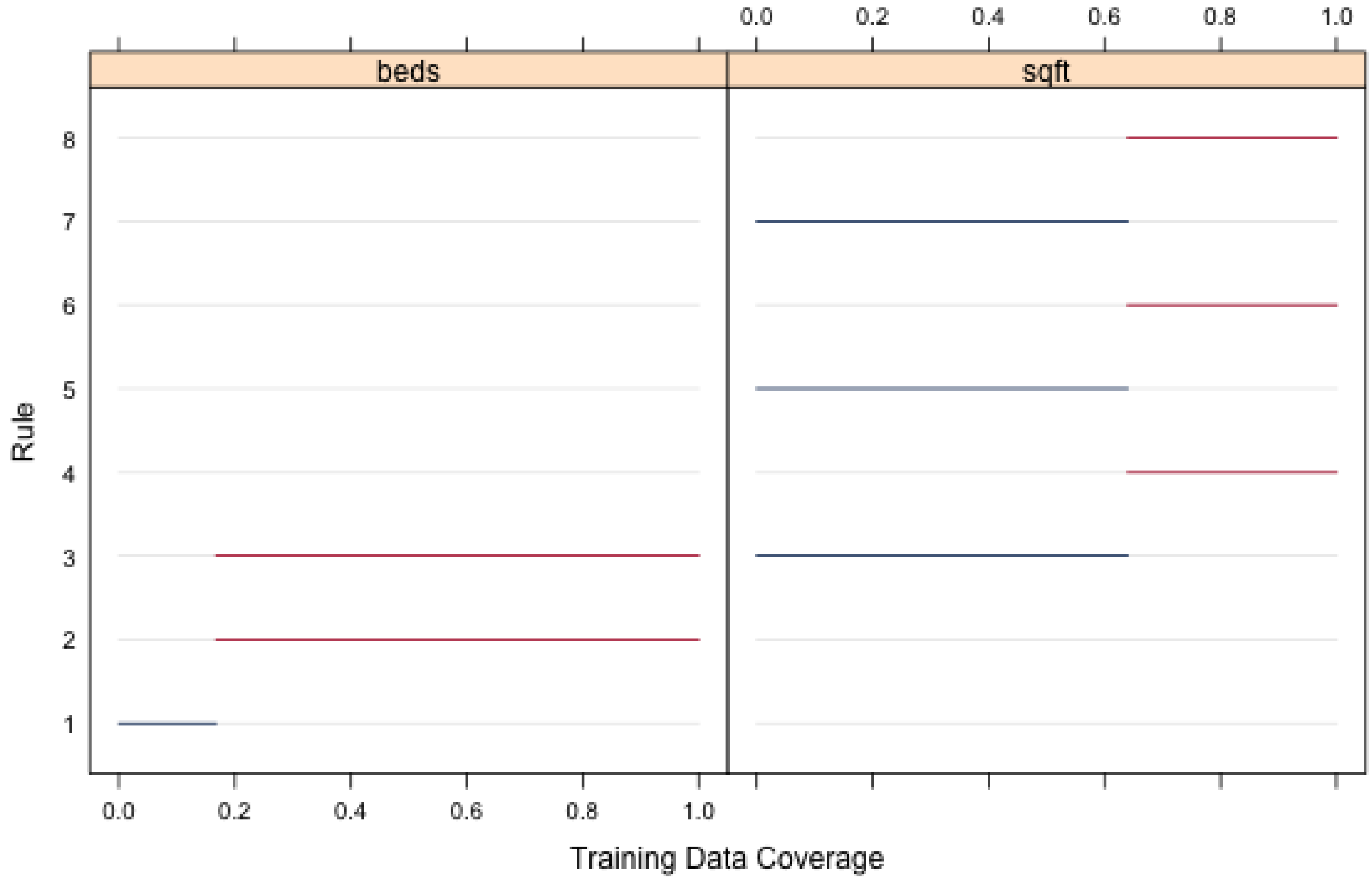
Cubist Base–Model Results

A basic cubist model resulted in 8 rules. For example:

```
Rule 1: [58 cases, mean 4.980517, range 4.477121 to 5.523746, est err 0.152796]

if zip in {z95621, z95626, z95660, z95673, z95683, z9581, z95817, z95820,
           z95822, z95823, z95824, z95826, z95827, z95828, z95832, z95838
           z95841, z95842, z95843} and
   beds <= 2 then
outcome = 7.944631 + 0.323 beds + 4e-05 sqft + 0.03 longitude

Rule 2: [126 cases, mean 5.200466, range 4.788875 to 5.662758, est err 0.090147]

if zip in {z95626, z95660, z95683, z95815, z95823, z95824, z95827, z95832,
           z95838, z95841} and
   beds > 2 then
outcome = 8.524561 - 0.056 beds + 0.000342 sqft + 0.03 longitude + 0.003 baths
```

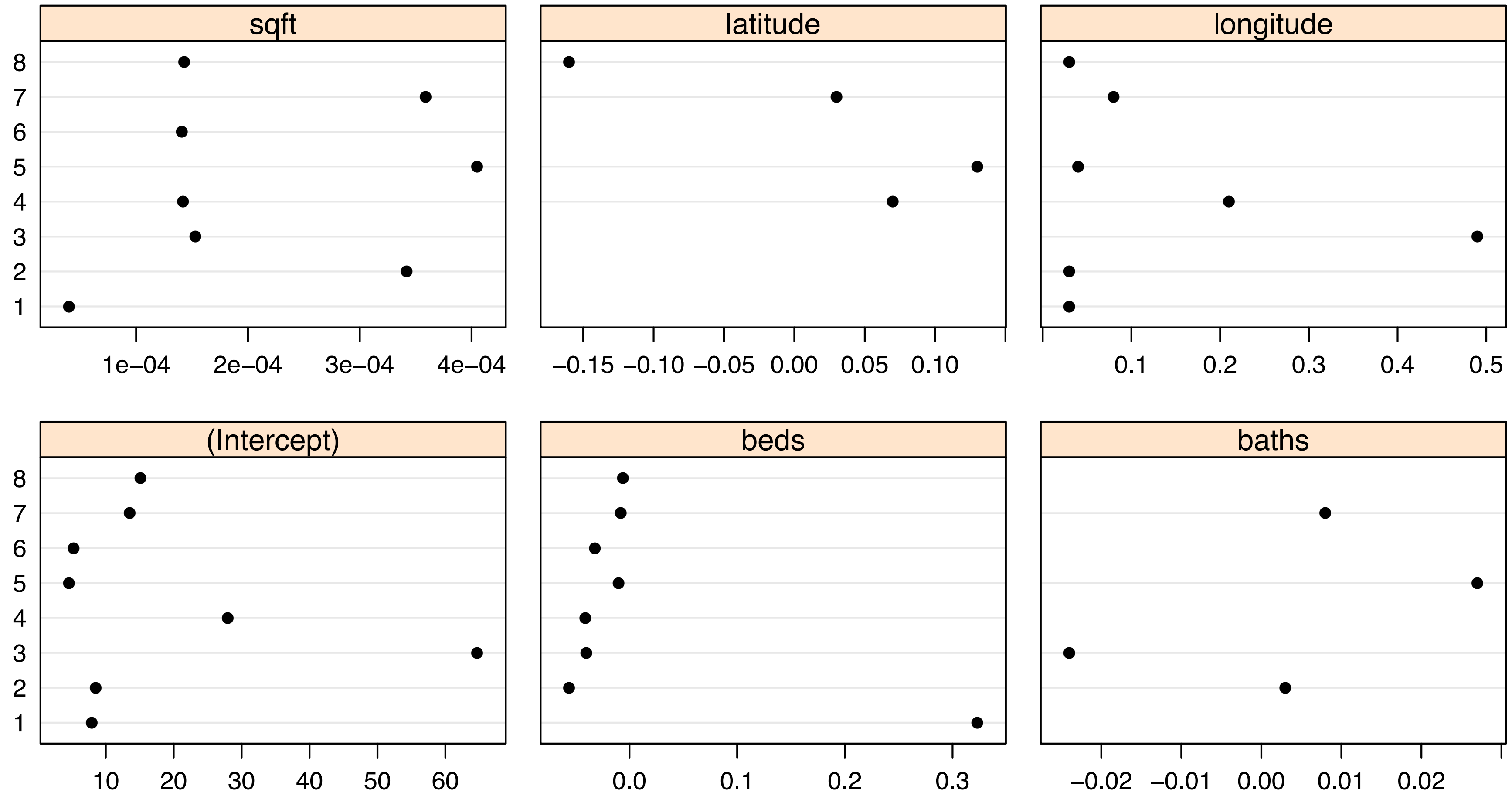
Plotting the Splits

```
> dotplot(cb)
```



Plotting the Slopes

```
> dotplot(cb, what = "coefs")
```



Why is the slopes for bed and bath negative in rule 3?

```
> r3_data <- subset(training, beds > 2 &  sqft <= 1712 &
+                     zip %in% c("z95621", "z95673", "z95817", "z95820",
+                                "z95822", "z95826", "z95828", "z95842",
+                                "z95843"))
>
> (mod1 <- lm(log10(price) ~ beds+baths, data = r3_data))
```

```
Call:
lm(formula = log10(price) ~ beds + baths, data = r3_data)

Coefficients:
(Intercept)         beds        baths
    5.36584     -0.02008     -0.04177
```

Adding ft^2 Changes the Estimate

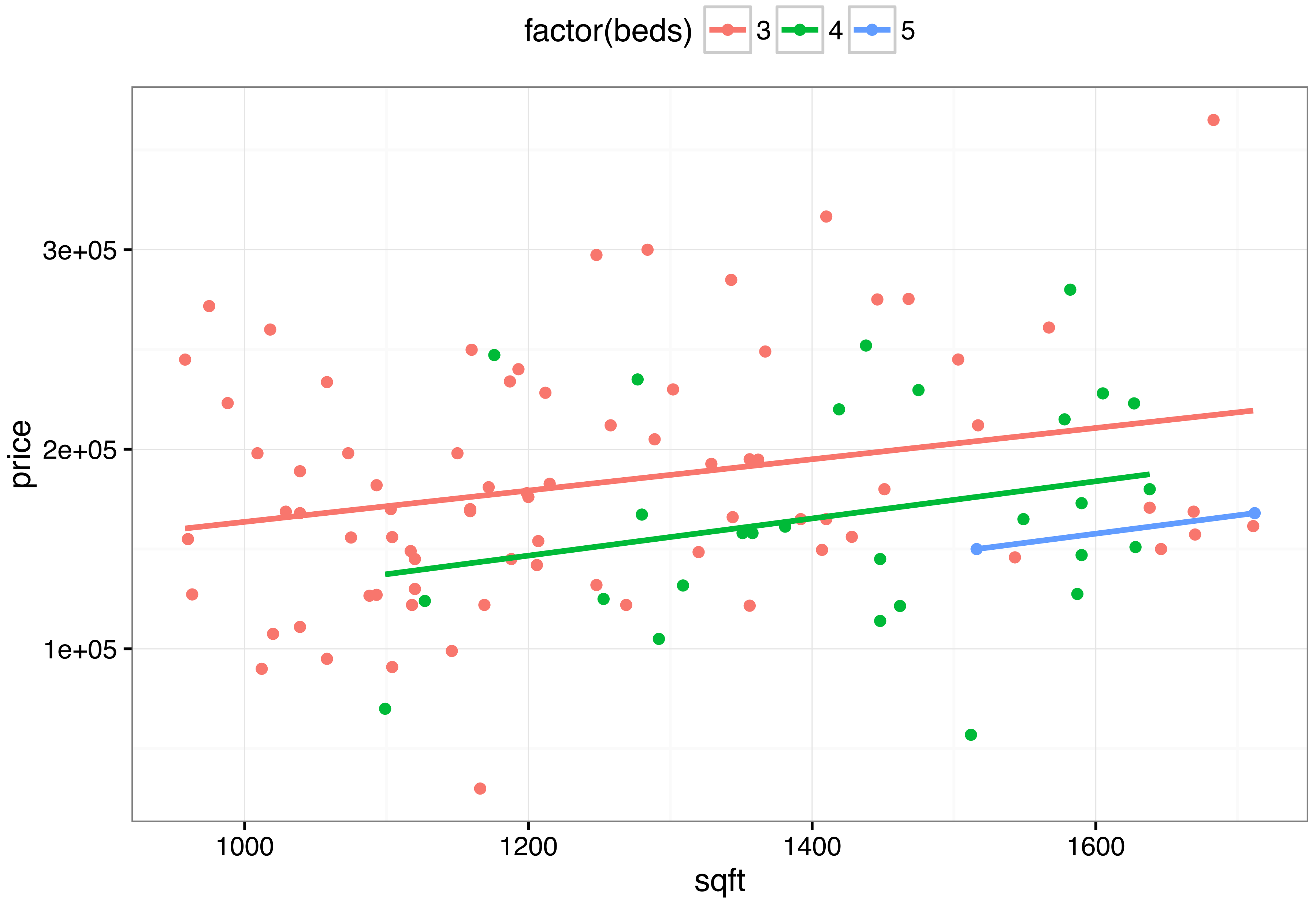
```
> (mod2 <- lm(log10(price) ~ beds+baths+sqft, data = r3_data))


Call:
lm(formula = log10(price) ~ beds + baths + sqft, data = r3_data)

Coefficients:
(Intercept)         beds        baths         sqft
    5.21540     -0.05959     -0.08134      0.00027

> (mod3 <- lm(log10(price) ~ beds+baths+sqft+longitude, data = r3_data))


Call:
lm(formula = log10(price) ~ beds + baths + sqft + longitude,
    data = r3_data)

Coefficients:
(Intercept)         beds        baths         sqft    longitude
 57.1758991   -0.0565129   -0.0944042    0.0002572    0.4277928
```

Collinearity?

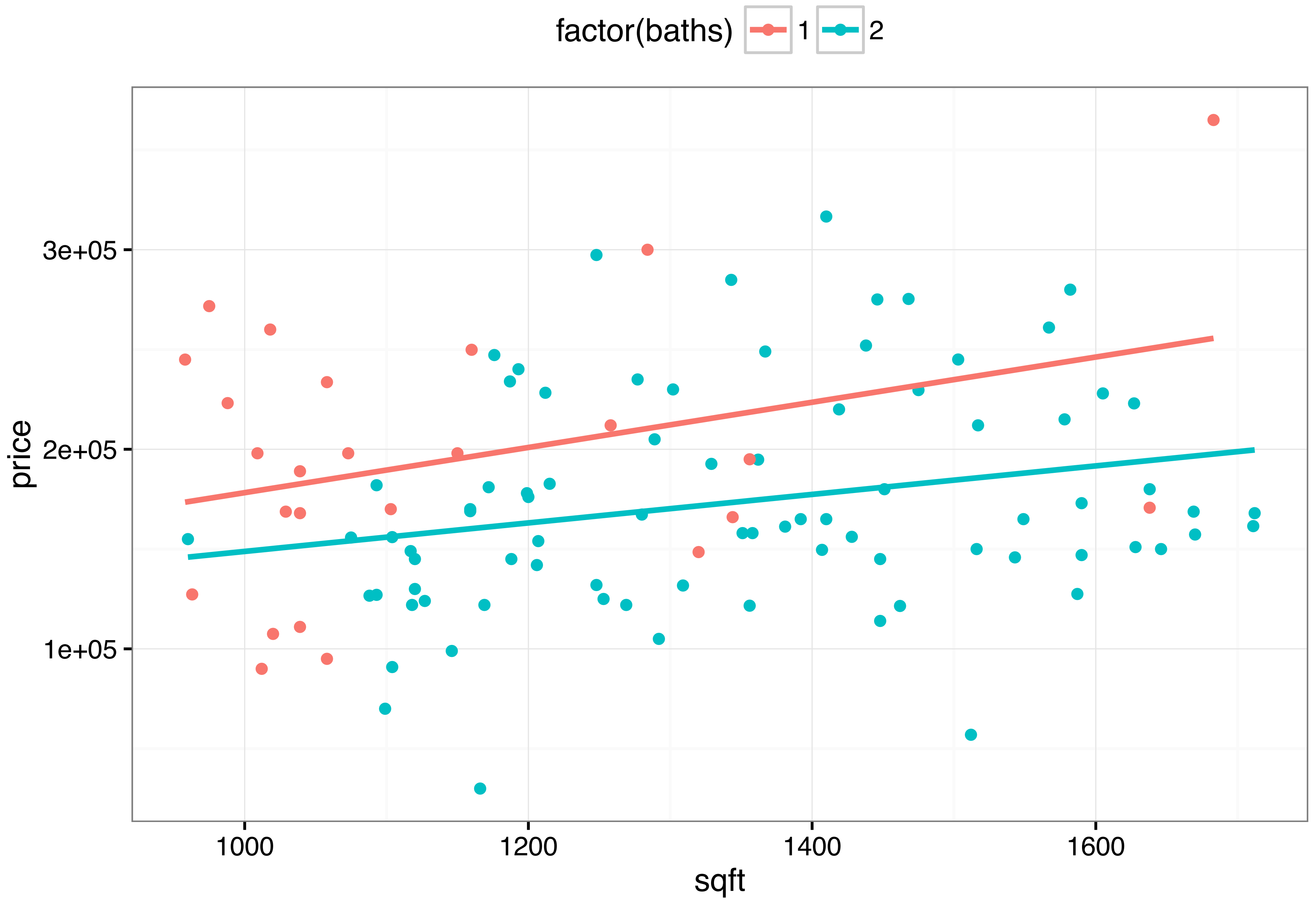
Variance Inflation Factors are fairly low

```
> ## the VIFs are not too bad:
> library(car)
> vif(mod3)

    beds     baths      sqft longitude
1.300475  1.303686  1.407447  1.075937

> ## fold increase in standard error:
> sqrt(vif(mod3))

    beds     baths      sqft longitude
1.140384  1.141791  1.186359  1.037274
```

The slopes should be negative!



The slopes should be negative!



Cubist Committees

Model committees can be created by generating a sequence of rule–based models (similar to boosting).

The training set outcome is adjusted based on the prior model fit and then builds a new set of rules using this pseudo–response.

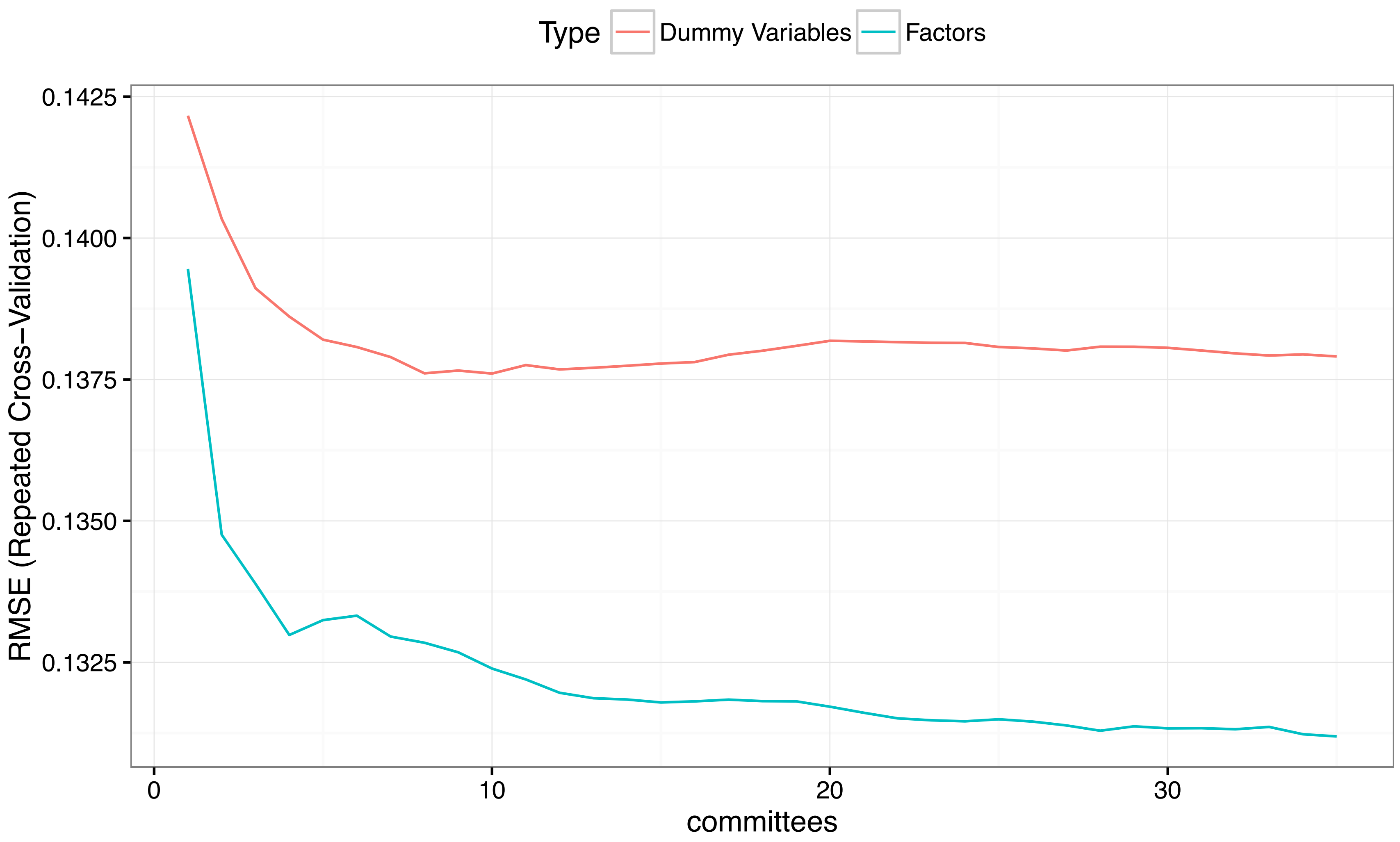
Specifically, the k^{th} committee model uses an adjusted response:

$$y_{i(k)} = 2y_{i(k-1)} - \widehat{y}_{i(k-1)}$$

Once the full set of committee models are created, new samples are predicted using each model and the final rule–based prediction is the simple average of the individual model predictions.

```
> cb <- cubist(x = training[, -7], y = log10(training$price), committees = 17)
```

Committee Results



Neighbor–Based Adjustments

Cubist has the ability to adjust the model prediction using samples from the training set (Quinlan 1993).

When predicting a new sample, the K most similar neighbors are determined from the training set.

$$\widehat{y} = \frac{1}{K}\sum_{\ell=1}^{K} w_\ell \left[\left(t_\ell - \widehat{t}_\ell\right) + \widehat{y}\right]$$

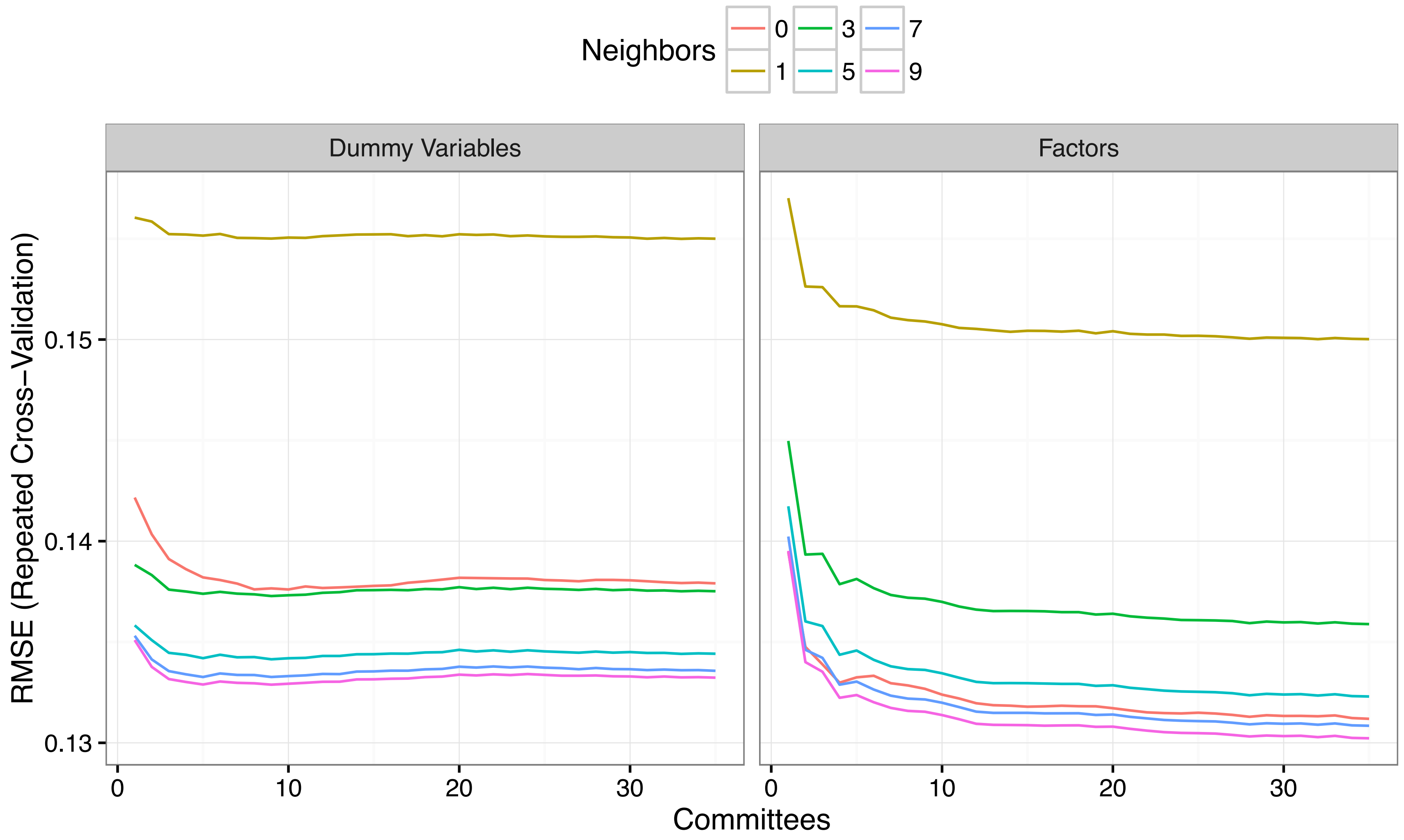
t_ℓ is the observed outcome for a training set neighbor,
$\widehat{t}_\ell$ is the model prediction of that neighbor and
w_ℓ is a weight calculated using the distance of the training set neighbors to the new sample.

```
> predict(cb, newdata = testing, neighbors = 4)
```

Tuning Model Trees in R

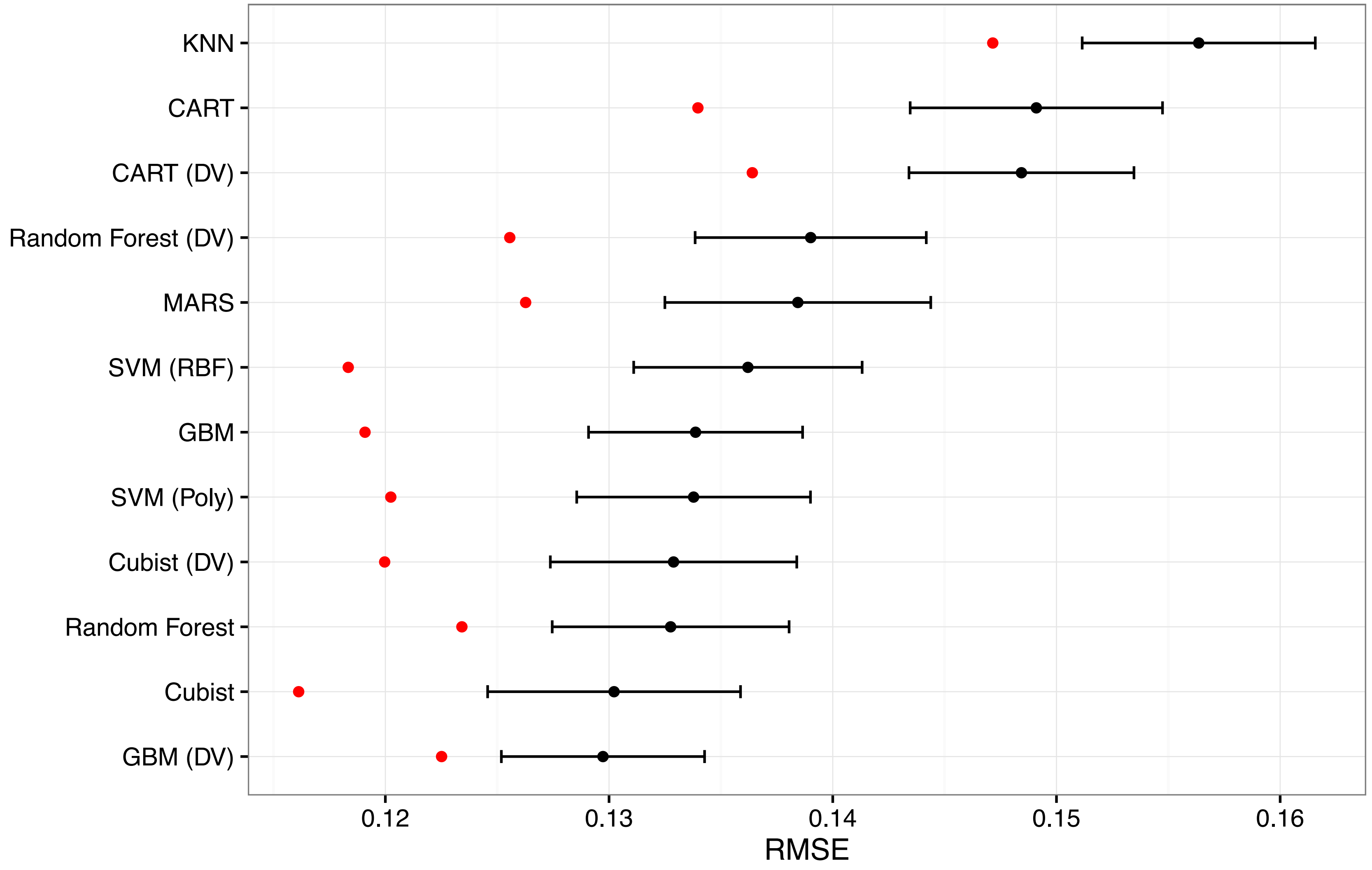
```
> cb_grid <- expand.grid(committees = c(1:35), neighbors = c(0, 1, 3, 5, 7, 9))
> set.seed(139)
> cb_tune_dv <- train(log10(price) ~ ., data = training,
+                     method = "cubist",
+                     tuneGrid = cb_grid,
+                     trControl = ctrl)
> set.seed(139)
> cb_tune <- train(x = training[, -7], y = log10(training$price),
+                  method = "cubist",
+                  tuneGrid = cb_grid,
+                  trControl = ctrl)
> ggplot(cb_tune) ## to see the profiles
```

Results with Neighbor Correction

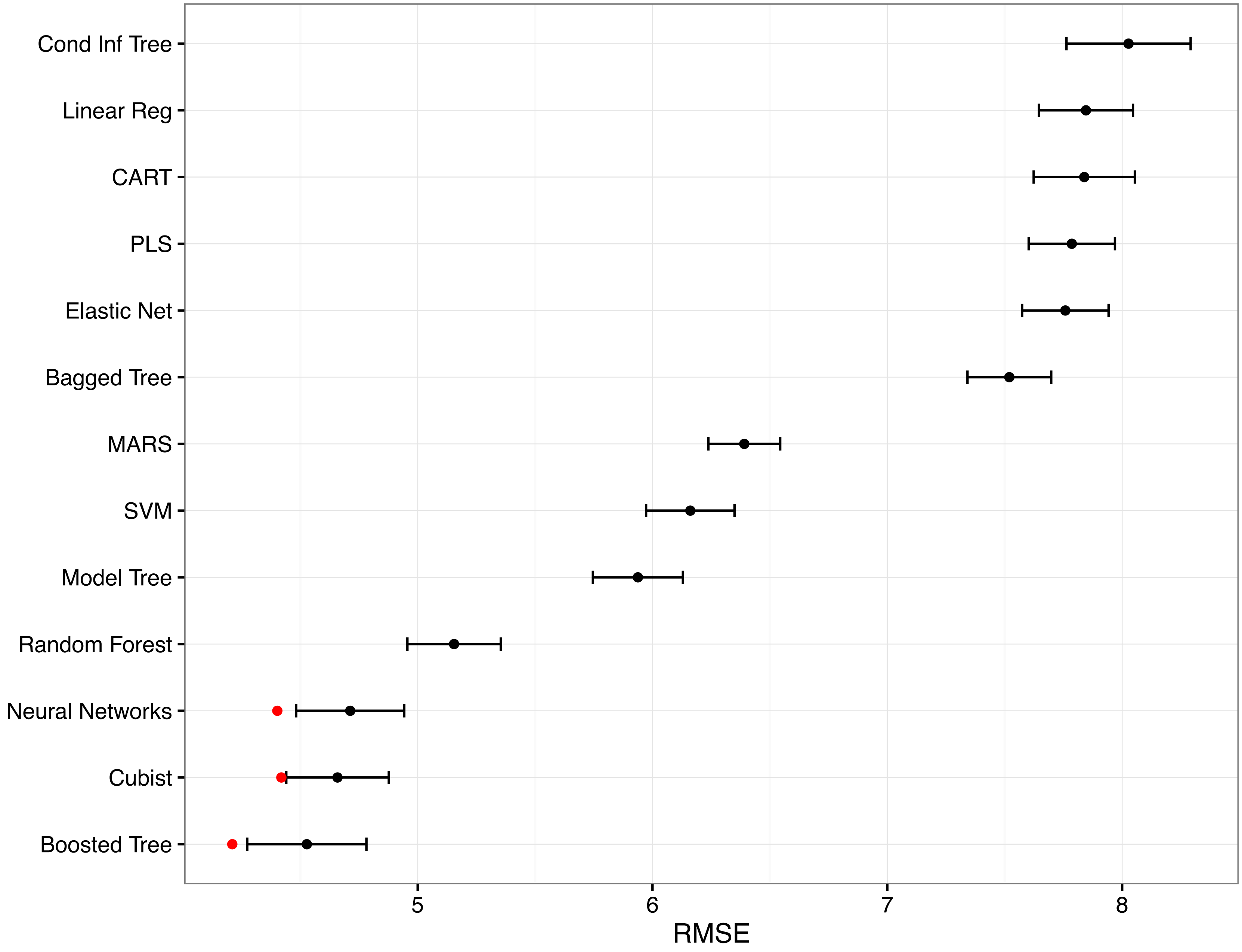


Comparisons with Other Models

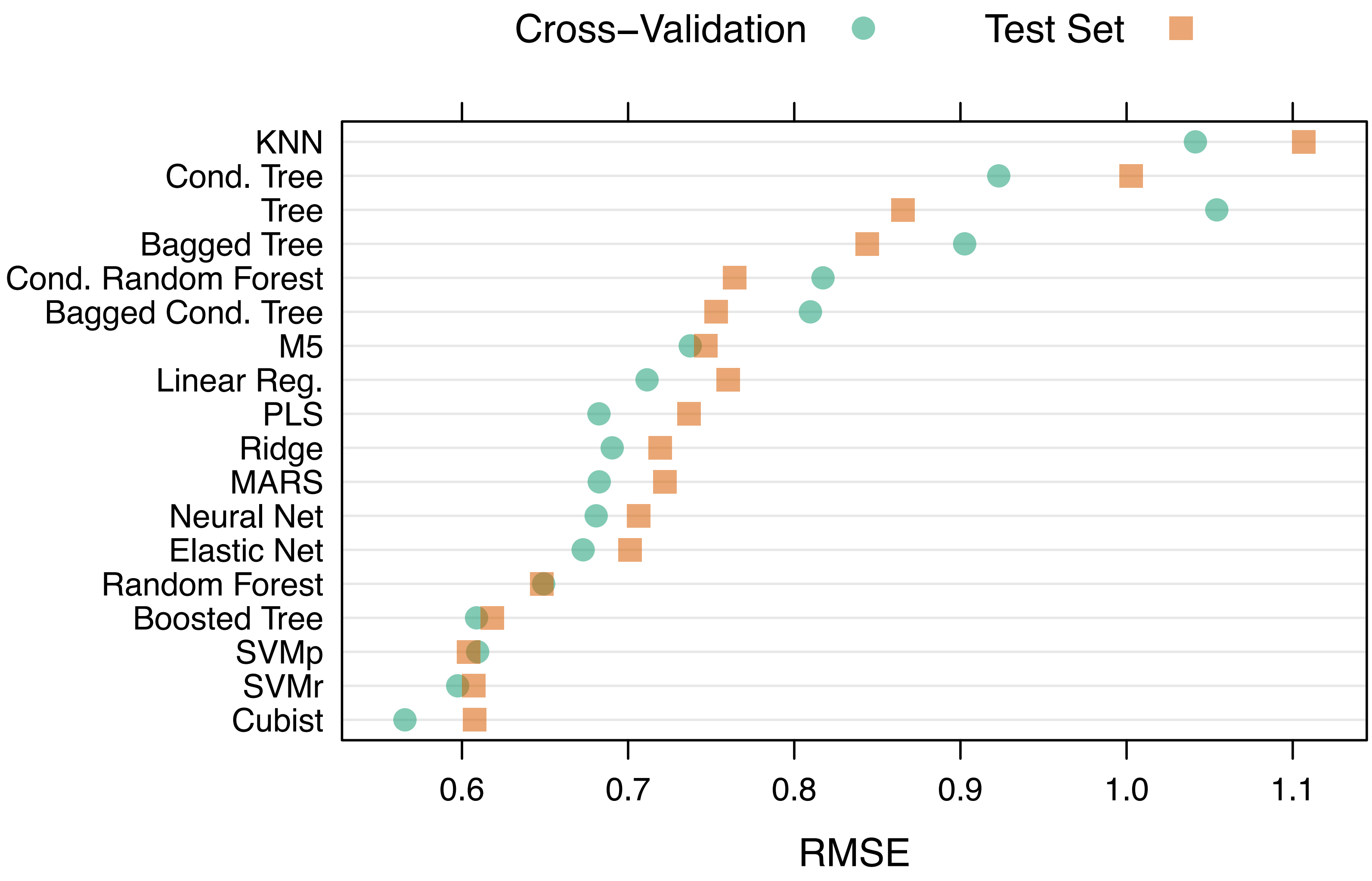
Test results in red



Results with APM's Concrete Data Analysis



Results with APM's Solubility Data Analysis



Classification Rules

Many Rule–Based Models

Classification rules have been around for a long time. Some models historical models are:

- C4.5 rules
- Repeated Incremental Pruning to Produce Error Reduction (ripper)
- PART

The last two are "separate and conquer" approaches available in the RWeka package.

There are also fuzzy rule–based models, such as the Adaptative–Network-Based Fuzzy Inference System (ANFIS). See the frbs andanfis packages.

C5.0

C5.0 is a classification tree model. It is dissimilar from the more well-known CART model in that

- a different (unbiased) splitting criterion is used
- multi-way splits are allowed
- different pruning methods and missing value contingencies are used

C5.0 also has a boosting procedure that is more similar to adaBoost than to stochastic gradient boosting.

C5.0 Rules

C5.0 follows a rule creation algorithm similar to the one used by Cubist

1. create an initial tree
2. convert the tree to rules
3. prune and simplify the rules
4. determine class probabilities per rule (with a Laplace–like correction)

Predictions are created using *all* active rules; each votes for a class proportional to the corresponding class probability.

Example Data

For illustration, I'll use some credit scoring data from Gaston Sanchez by way of this blog post by Sergio Venturini:

```
> library(RCurl)
> url <- "https://raw.githubusercontent.com/gastonstat/CreditScoring/master/CleanCr
> cs_data <- getURL(url)
> cs_data <- read.csv(textConnection(cs_data))
> ## Remove some predictors that are discrete versions of existing fields
> cs_data <- cs_data[, !grepl("R$", names(cs_data))]
```

Example Data

```
> str(cs_data)

'data.frame': 4446 obs. of  16 variables:
 $ Status   : Factor w/ 2 levels "bad","good": 2 2 1 2 2 2 2 2 2 1 ...
 $ Seniority: int  9 17 10 0 0 1 29 9 0 0 ...
 $ Home     : Factor w/ 6 levels "ignore","other",..: 6 6 3 6 6 3 3 4 3 4 ...
 $ Time     : int  60 60 36 60 36 60 60 12 60 48 ...
 $ Age      : int  30 58 46 24 26 36 44 27 32 41 ...
 $ Marital  : Factor w/ 5 levels "divorced","married",..: 2 5 2 4 4 2 2 4 2 2 ...
 $ Records  : Factor w/ 2 levels "no_rec","yes_rec": 1 1 2 1 1 1 1 1 1 1 ...
 $ Job      : Factor w/ 4 levels "fixed","freelance",..: 2 1 2 1 1 1 1 1 2 4 ...
 $ Expenses : int  73 48 90 63 46 75 75 35 90 90 ...
 $ Income   : int  129 131 200 182 107 214 125 80 107 80 ...
 $ Assets   : int  0 0 3000 2500 0 3500 10000 0 15000 0 ...
 $ Debt     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Amount   : int  800 1000 2000 900 310 650 1600 200 1200 1200 ...
 $ Price    : int  846 1658 2985 1325 910 1645 1800 1093 1957 1468 ...
 $ Finrat   : num  94.6 60.3 67 67.9 34.1 ...
 $ Savings  : num  4.2 4.98 1.98 7.93 7.08 ...
```

Data Spitting

The data were split using a stratified random sample:

```
> set.seed(1987)
> inTrain <- createDataPartition(cs_data$Status, p = .8, list = FALSE)
>
> credit_train <- cs_data[ inTrain,]
> credit_test  <- cs_data[-inTrain,]
>
> table(credit_test$Status)/nrow(credit_test)


      bad      good
0.2804054 0.7195946

> table(cs_data$Status)/nrow(cs_data)


      bad      good
0.2809267 0.7190733
```

A Single C5.0 Ruleset and Tree

To fit the models:

```
> library(C50)
>
> c5_tree  <- C5.0(Status ~ ., data = credit_train)
> c5_rules <- C5.0(Status ~ ., data = credit_train, rules = TRUE)
```

The `summary` function will show the rules and details. Here, there were 38 pruned rules of the initial set of 95 terminal tree nodes.

C5.0 Rulesets

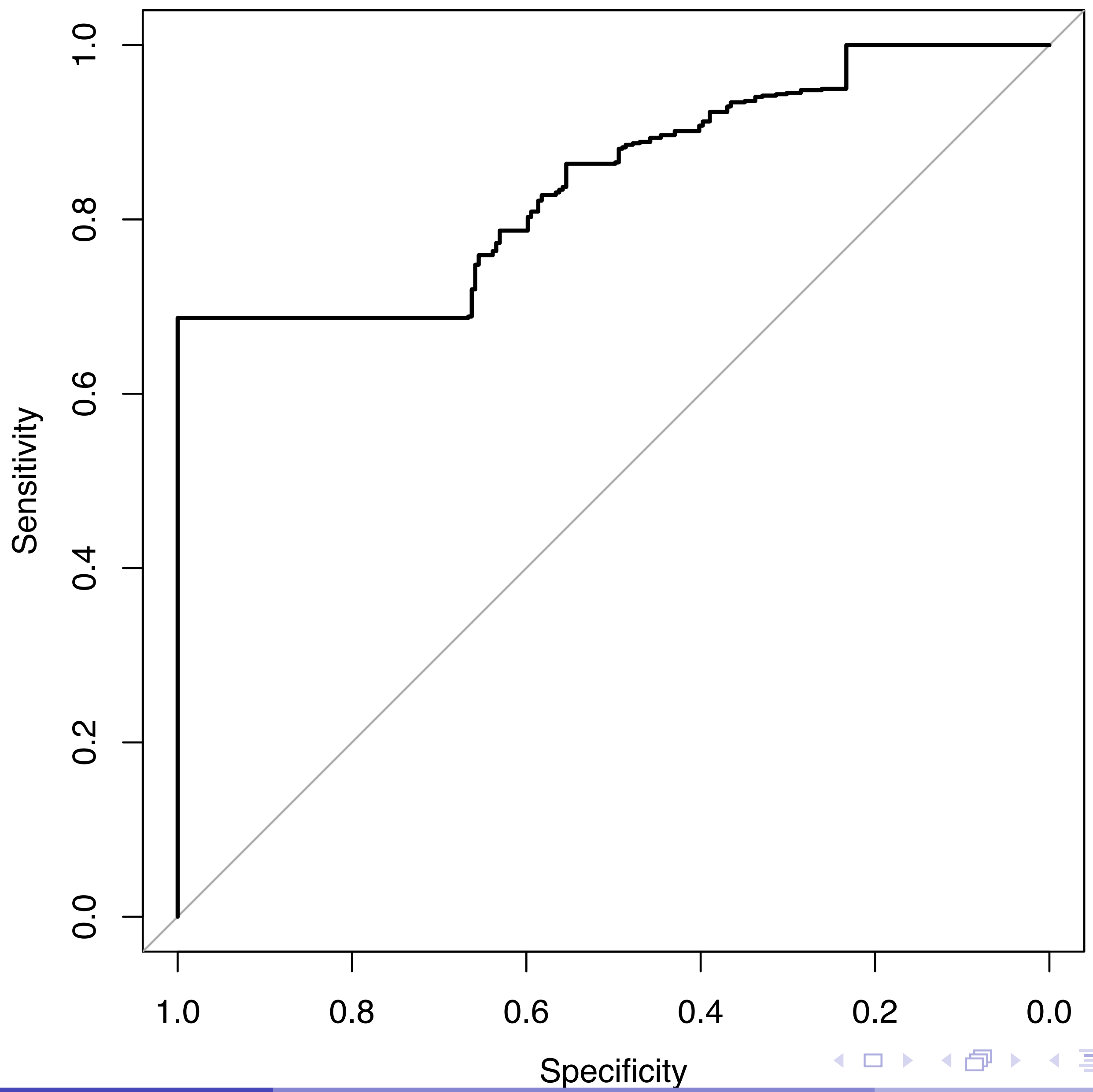
For example:

```
> summary(c5_rules)
> ## snip!


Rule 1: (15, lift 3.3)
Seniority <= 5
Home = rent
Records = no_rec
Job = freelance
Expenses <= 45
Assets <= 3750
->  class bad  [0.941]

Rule 2: (27/1, lift 3.3)
Home = rent
Job in {fixed, freelance, others}
Assets <= 2250
Price > 1651
Finrat > 69.80273
Savings > -0.9230769
Savings <= 1.708235
->  class bad  [0.931]
```

Test Set Performance

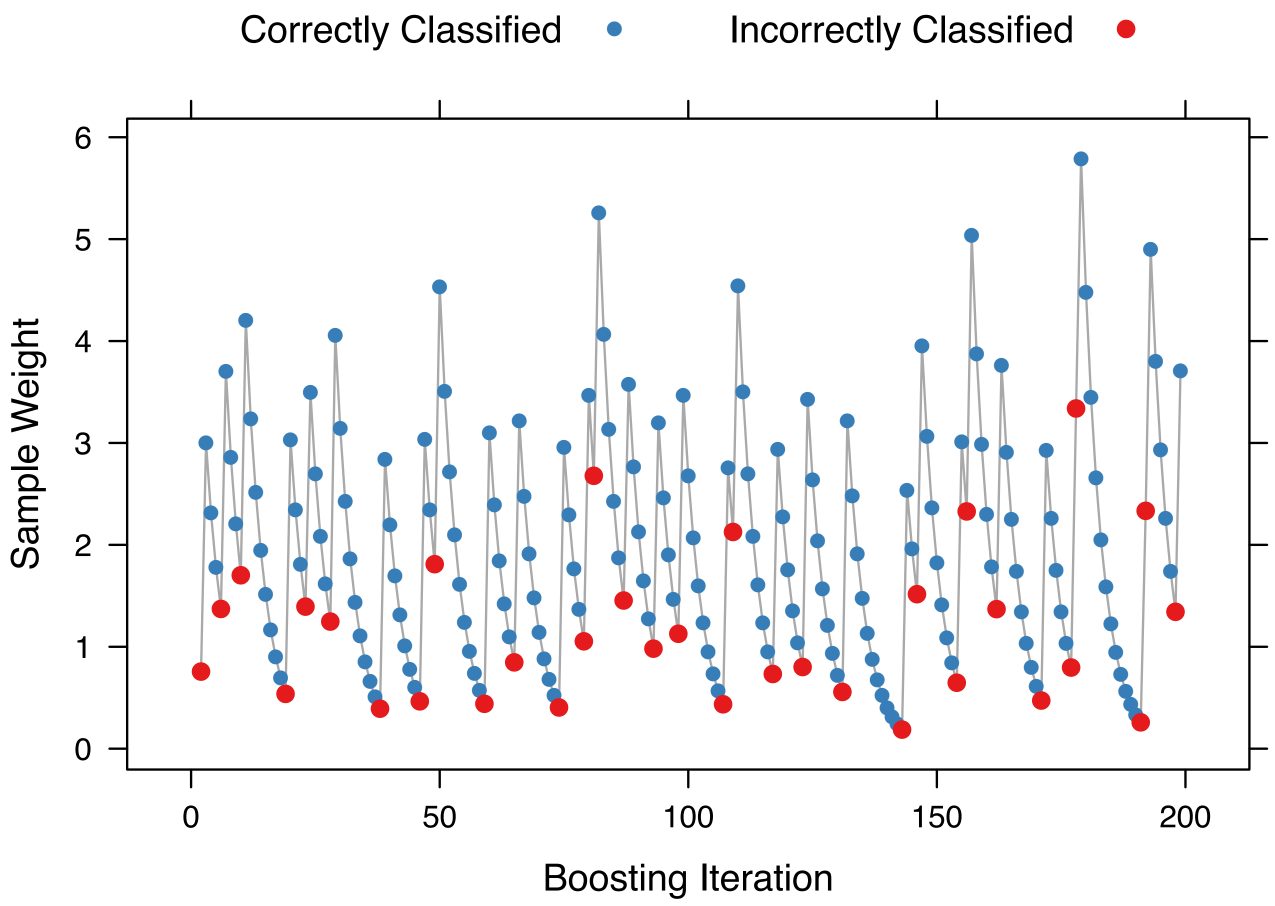
```
> library(pROC)
> c5_rules_pred <- predict(c5_rules, credit_test, type = "prob")
> c5_rules_roc <- roc(credit_test$Status, c5_rules_pred[, "good"])
> c5_rules_roc


Call:
roc.default(response = credit_test$Status, predictor = c5_rules_pred[,     "good"])

Data: c5_rules_pred[, "good"] in 249 controls (credit_test$Status bad) < 639 cases
Area under the curve: 0.7217

> ## plot(c5_rules_roc, type = "s")
```

Test Set Performance



Boosted C5.0 Rules

C5.0 uses an iterative weighting scheme for boosting where errors increase the weights additively and correctly predicted training set samples are decreased multiplicatively:



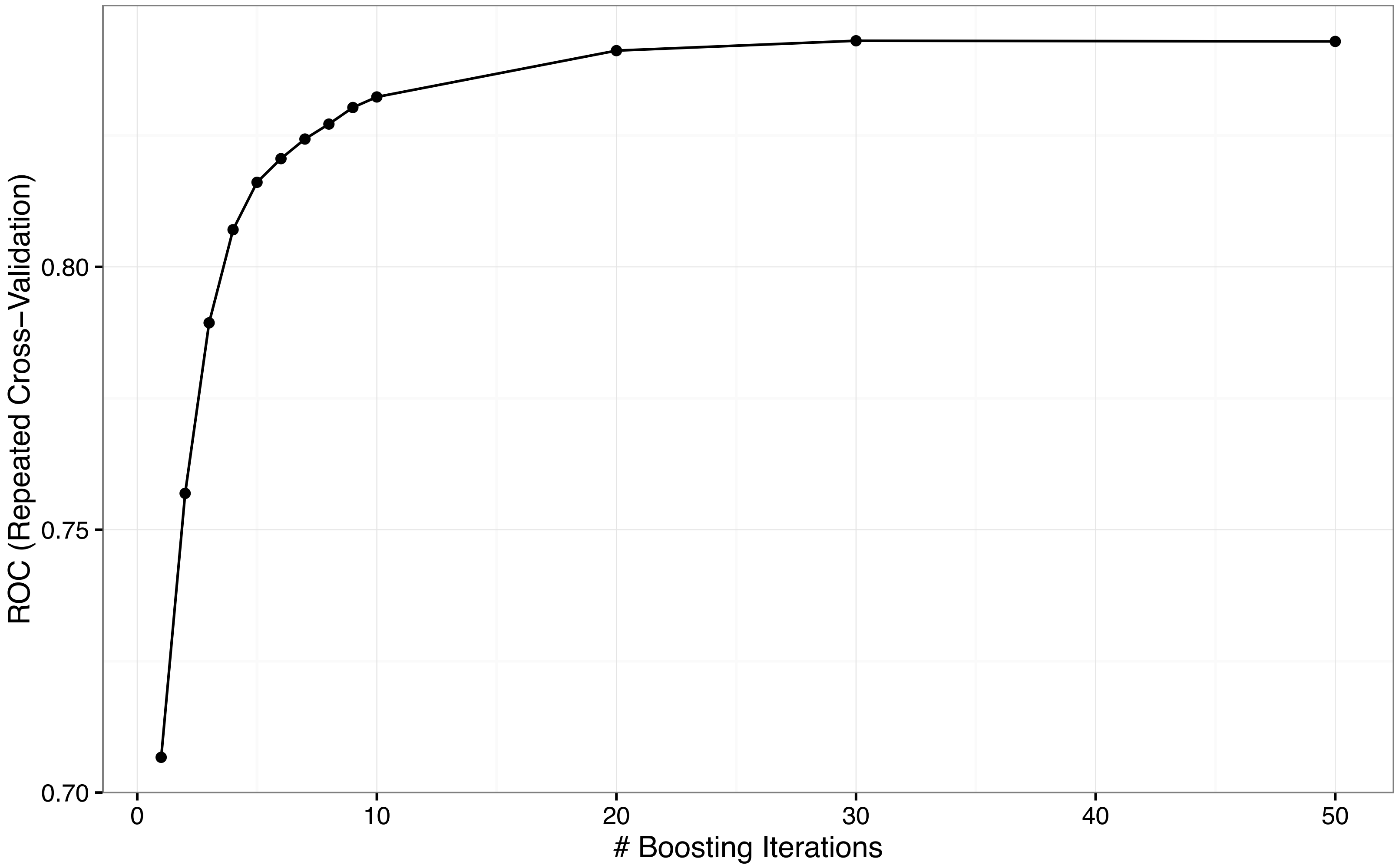
Boosted C5.0 Rulesets

The `trials` argument can be used to specify the amount of boosting, or:

```
> set.seed(1658)
> rule_boost <- train(Status ~ ., data = credit_train,
+                     method = "C5.0",
+                     metric = "ROC",
+                     tuneGrid = data.frame(trials = c(1:10, 20, 30, 50),
+                                           model = "rules",
+                                           winnow = FALSE),
+                     trControl = trainControl(method = "repeatedcv",
+                                              repeats = 5,
+                                              classProbs = TRUE,
+                                              summaryFunction = twoClassSummary))
```

Test Set Performance

```
> ggplot(rule_boost)
```



Bagged Tree Rules

Houtao Deng describes methods for post–pruning of rules created by a random forest fit into a classifier. His package (inTrees) can be used. Here, we use a small set of unpruned CART trees:

```
> library(inTrees)
> set.seed(910)
> bag_fit <- randomForest(x = credit_train[,-1],
+                         y = credit_train$Status,
+                         mtry = ncol(credit_train) - 1,
+                         ntree = 25) # make larger for RF
> bag_fit_list <- RF2List(bag_fit)
> bag_rules <- extractRules(bag_fit_list,
+                           X = credit_train[,-1],
+                           maxdepth = 10,
+                           ntree = 25)

5686 rules (length<=10) were extracted from the first 25 trees.

> bag_rules[1]

[1] "X[,1]<=2.5 & X[,3]<=33 & X[,6] %in% c('no_rec') & X[,7] %in% c('fixed','freela
```

Bagged Tree Rules

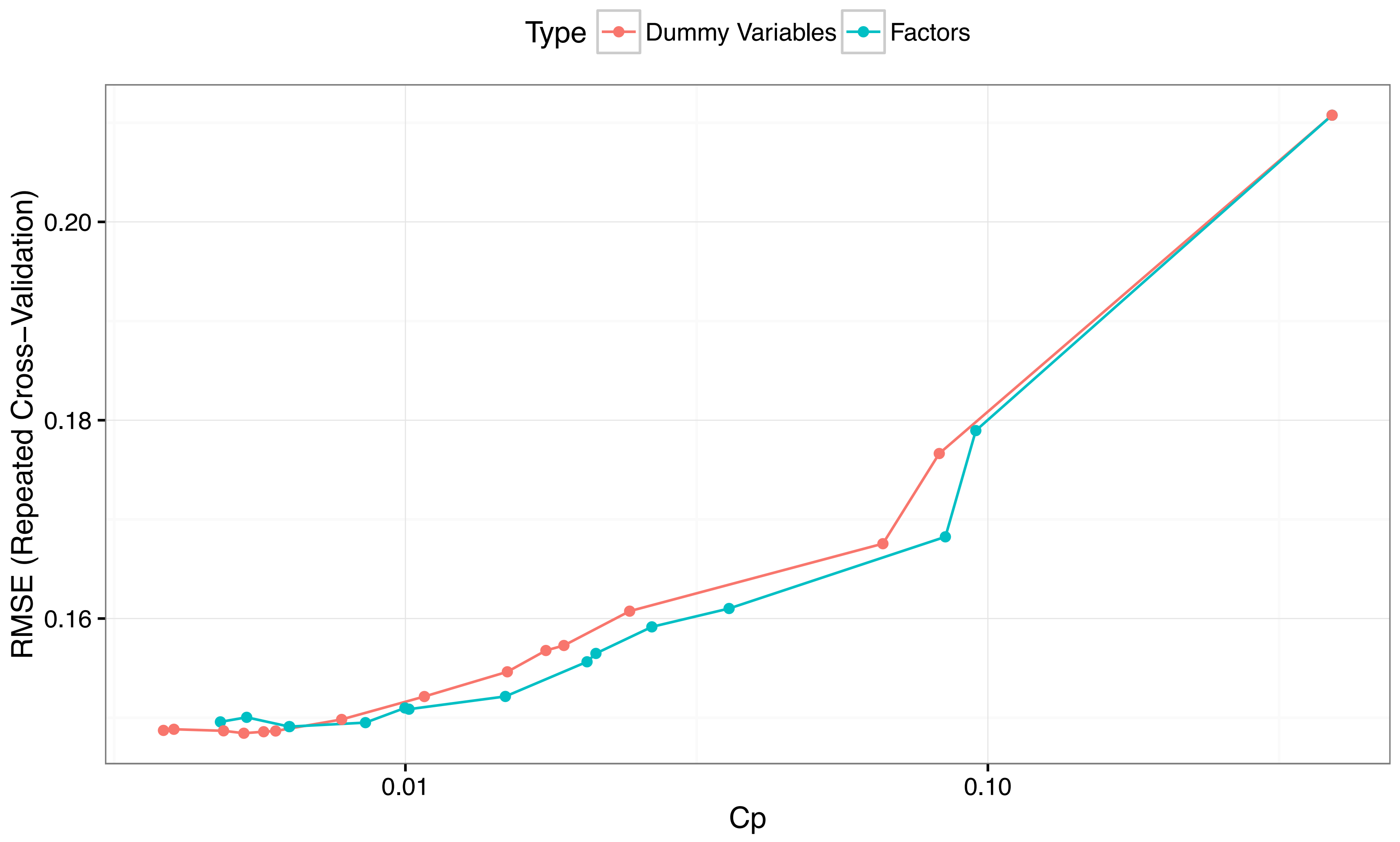
```
> rule_metric <- getRuleMetric(bag_rules, X = credit_train[,-1],
+                              target = credit_train$Status)
> pruned <- pruneRule(rule_metric, X = credit_train[,-1],
+                     target = credit_train$Status)
> filtered <- selectRuleRRF(pruned, X = credit_train[,-1],
+                           target = credit_train$Status)
> bag_rule_mod <-  buildLearner(filtered, X = credit_train[,-1],
+                               target = credit_train$Status)
> applyLearner(bag_rule_mod, credit_test[1:4,-1])

[1] "good" "good" "bad"  "good"

> ## no class probabilities can be generated
```

Random Forest Rules

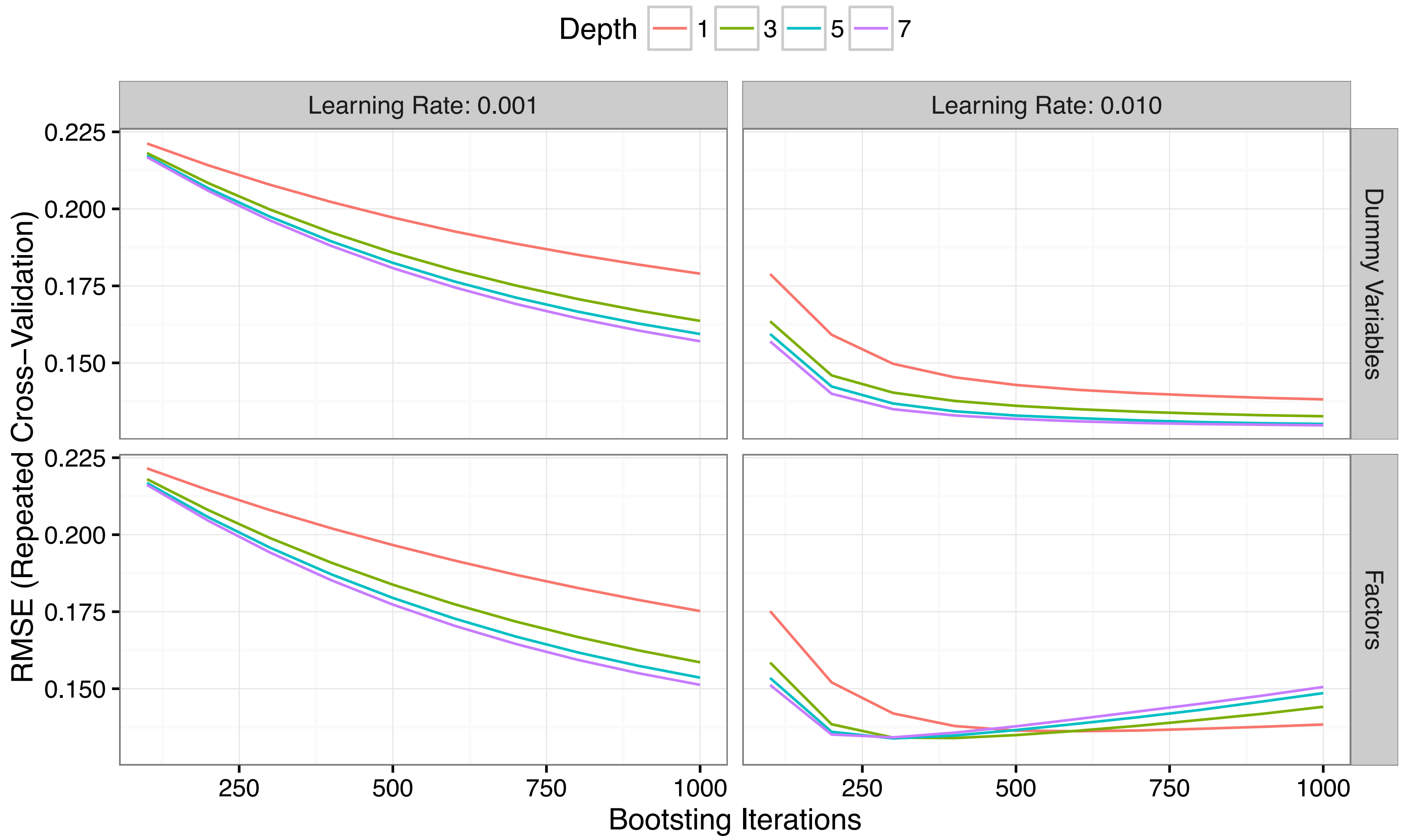
The number of trees, m_{try}, the tree depth and other factors may impact performance for this approach. caret's `train` function has a method (`rfRules`) that can be used to tune this process.

Backup Slides – Regression

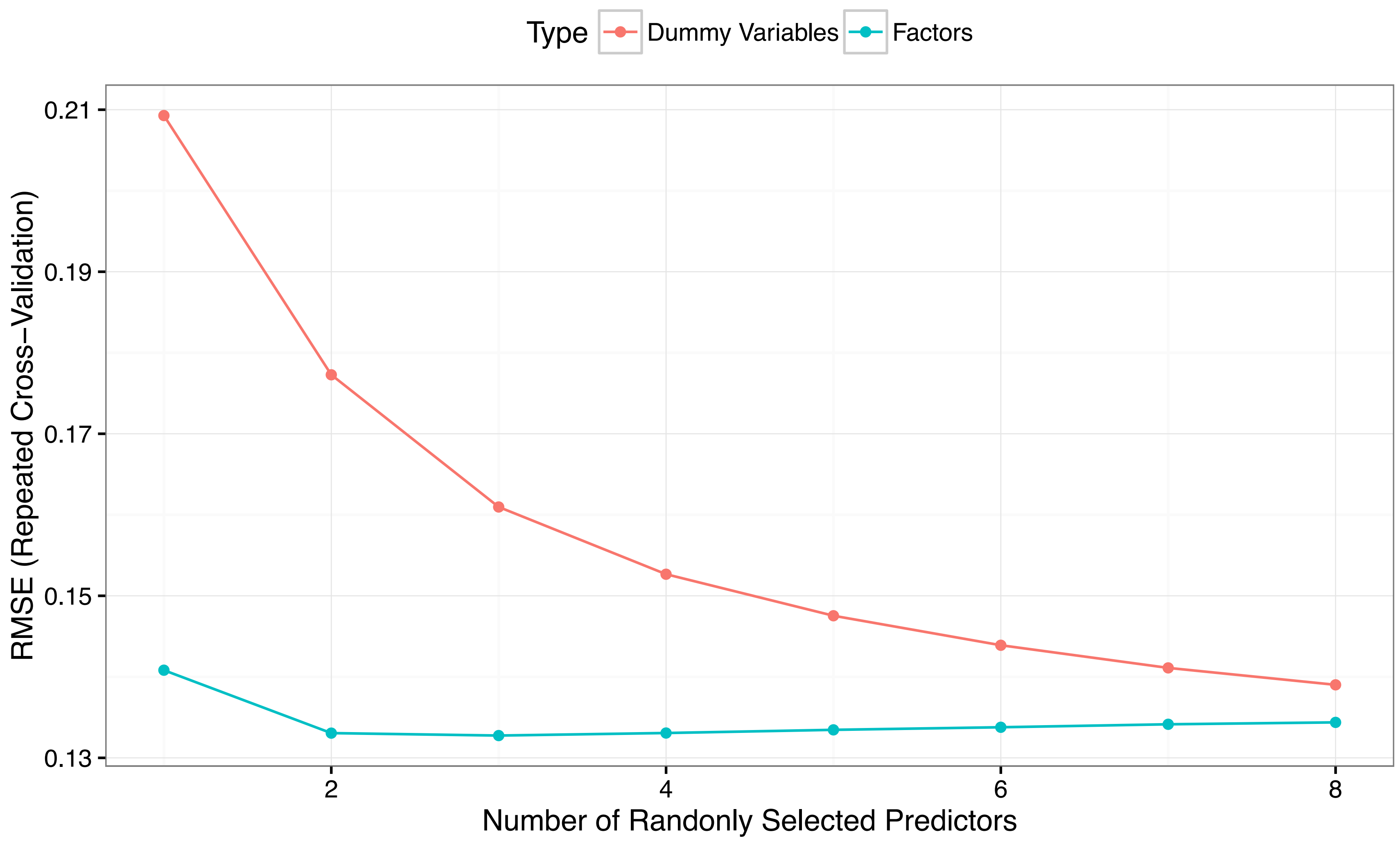
CART Profiles



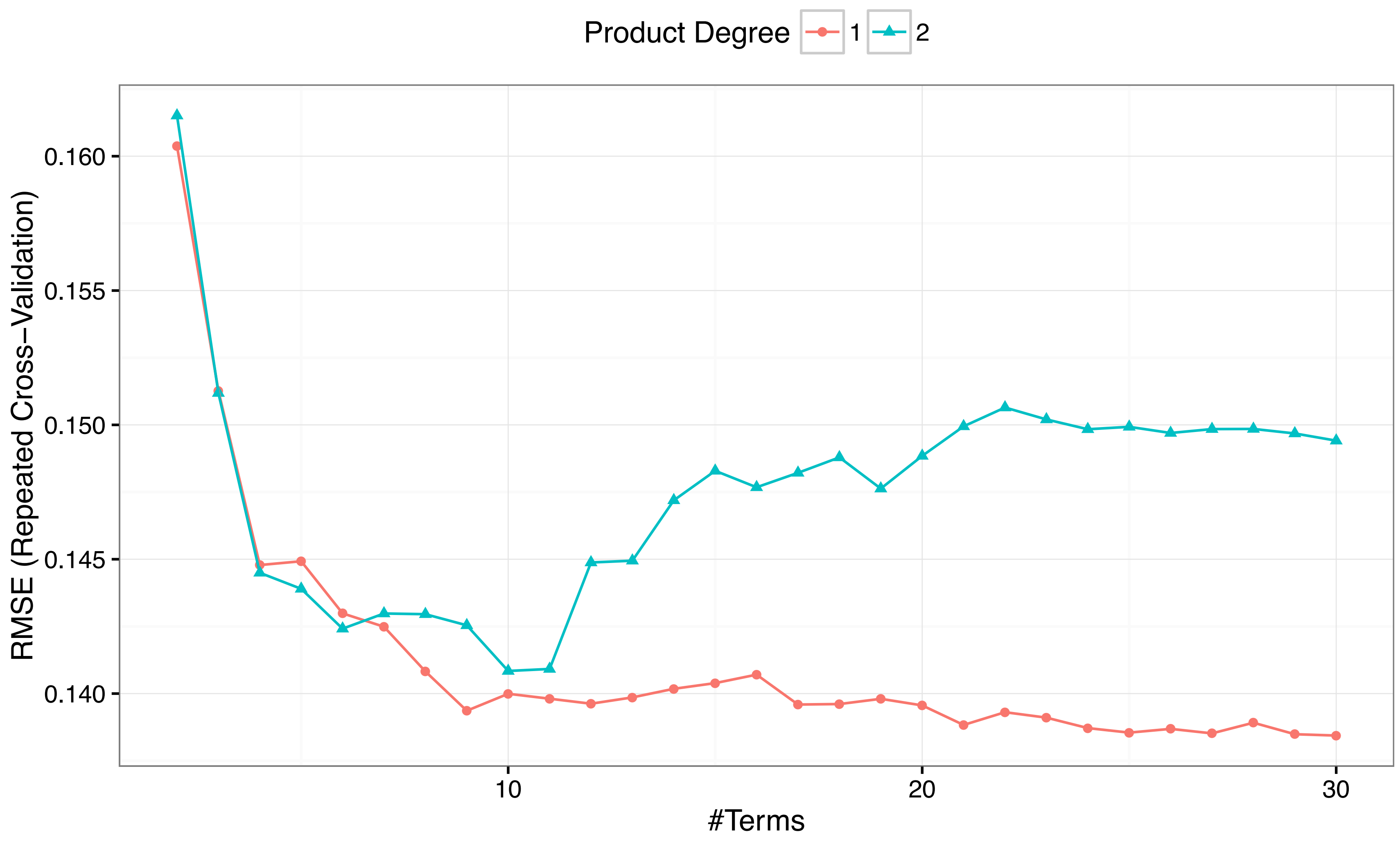
Boosted Tree Profiles



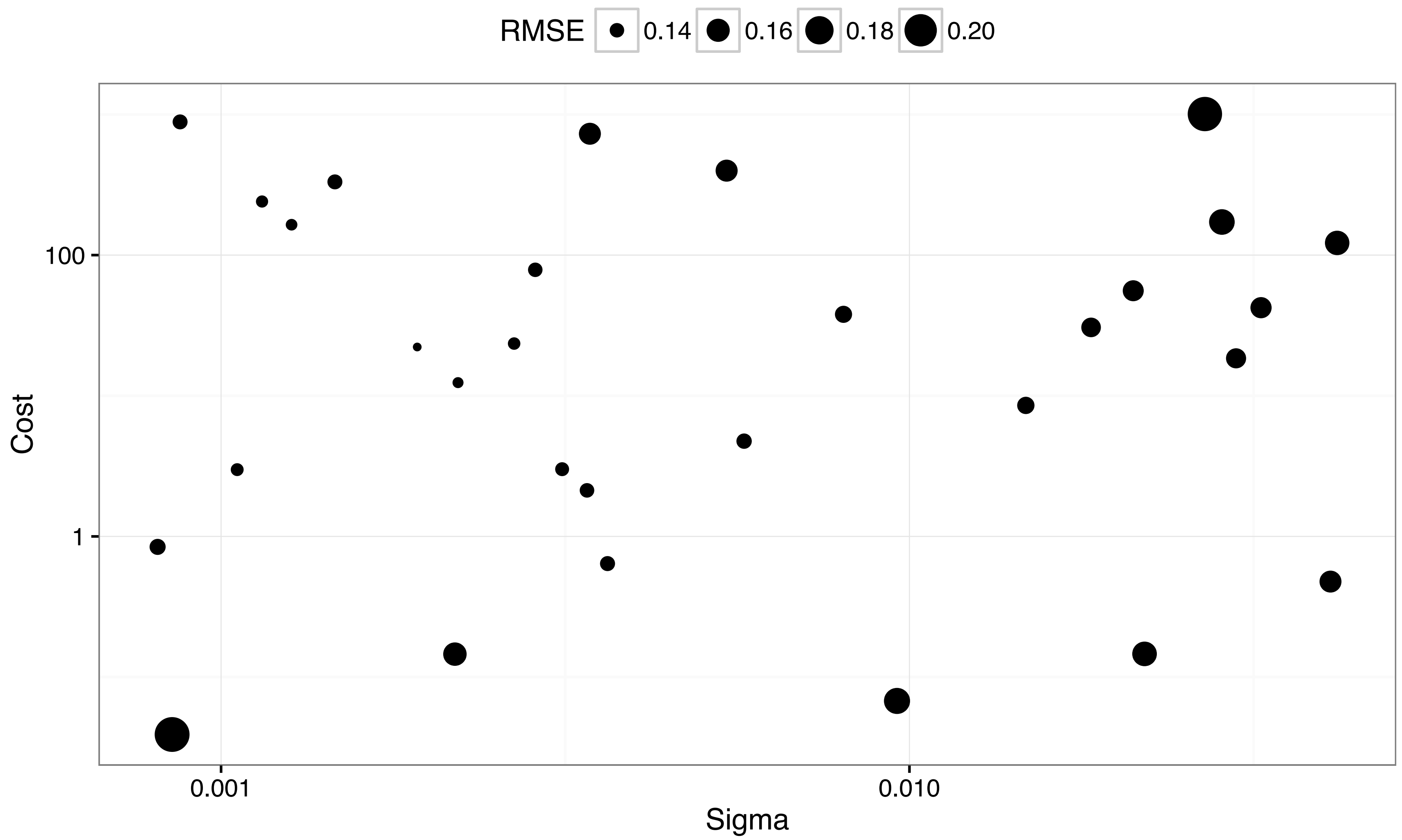
Random Forest Profiles



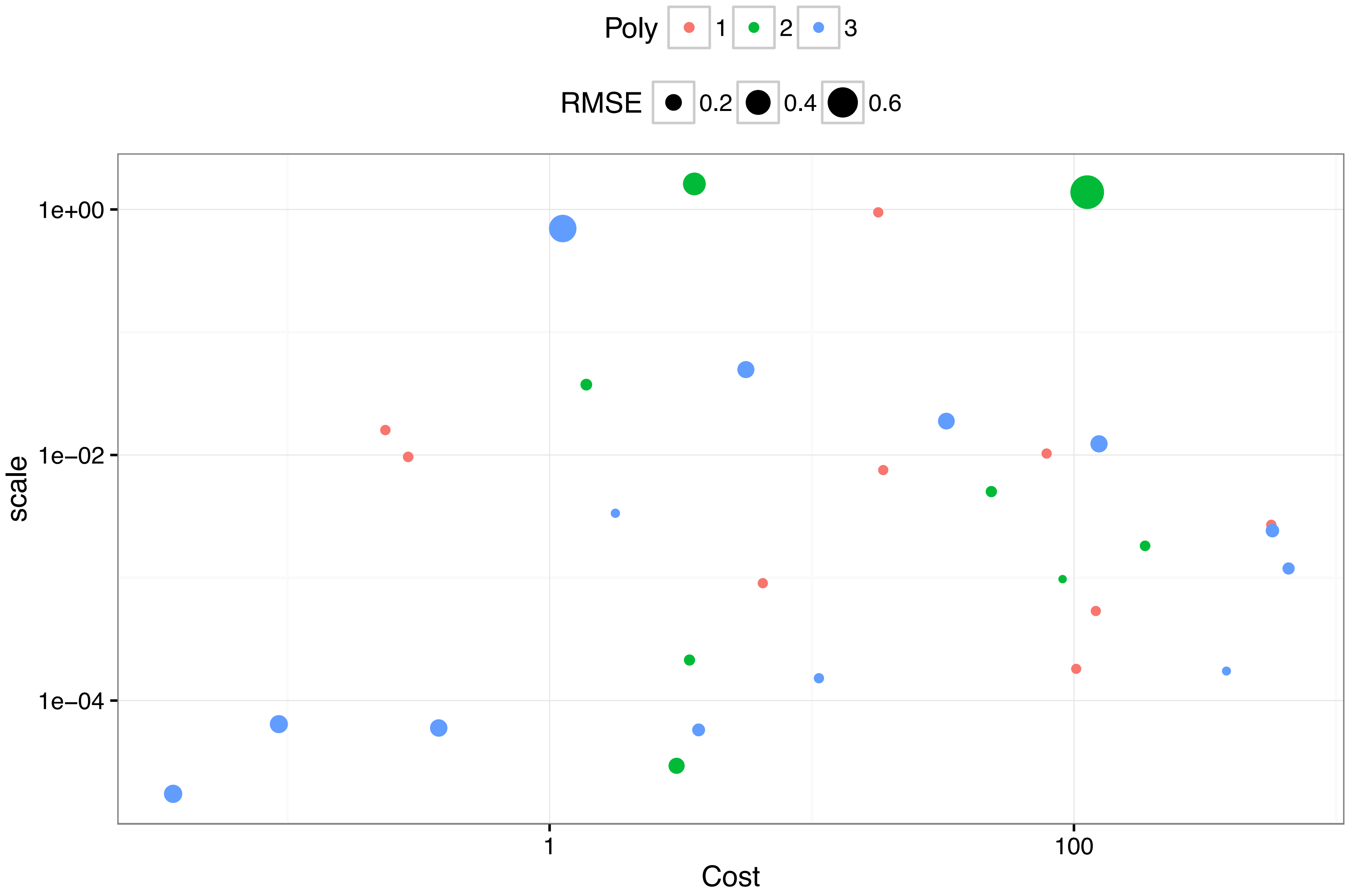
MARS Profiles



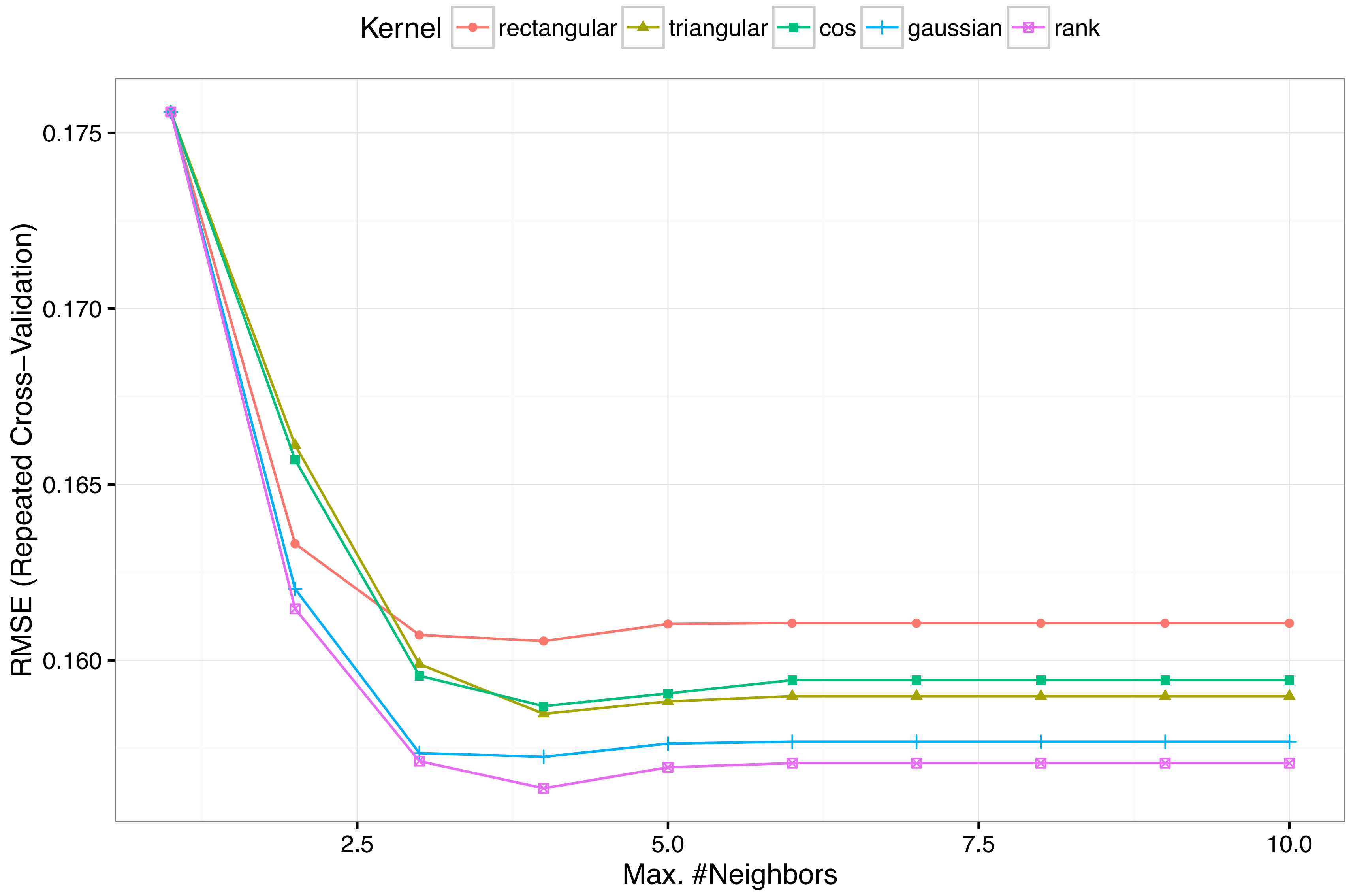
SVM (RBF) Profiles using Random Search



SVM (Poly) Profiles using Random Search



KNN Profiles



Thanks

Chris Keefer for his work with Cubist

Steve Weston, Chris Keefer and Nathan Coulter for adapting the Cubist C code to R.

References

Kuhn M and Johnson K (2013). *Applied Predictive Modeling*

Quinlan R (1992). "Learning with Continuous Classes." Proceedings of the 5th Australian Joint Conference On Artificial Intelligence, pp. 343-348.

Quinlan R (1993). "Combining InstanceBased and ModelBased Learning." Proceedings of the Tenth International Conference on Machine Learning, pp. 236-243.

Wang Y, Witten I (1997). "Inducing Model Trees for Continuous Classes." Proceedings of the Ninth European Conference on Machine Learning, pp. 128-137.

Witten I, Frank E and Hall, M (2011). *Data Mining: Practical Machine Learning Tools and Techniques*