Statistics 149 – Classification and regression trees (CART), and random forests

CART and random forests:

- Recursive partitioning algorithm
- Classification trees
- Regression trees
- Tree pruning via cross-validation
- Extending trees to random forests

Classification and regression trees: CART

Distinction between classification trees, regression trees:

- Classification tree: A tree-based model with a categorical response variable (binary, multinomial). Usually interested in modeling the probability of the different response levels.
- Regression tree: A tree-based model with a quantitative response variable. Usually interested in modeling the mean.

Tree-based models:

Goal: Same as most regression problems

- Given a data set of n observations with J predictors $x_{i1}, \ldots, x_{iJ}$ and response y_i for observation i, estimate μ_Y given the predictor variable values (if Y is quantitative), or the probability of Y being in level k (if Y is categorical).
- Unlike linear or additive models, the final result of a tree-based model is a partitioning of the n observations into M distinct groups, with the groupings determined by ranges of values of the predictor variables.
- The final fitted model is (usually) the average y_i within each grouping. The estimated model is not smoothly connected – it is a set of disconnected flat surfaces as a function of the predictor variables.

Basic partitioning method: Assume a quantitative response Y

1. Start with full set of n observations, compute the mean for all Y, and compute the deviance at the mean (acting as if the data were from a normal distribution).
2. For each variable x_j, consider the split of the data set into observations for which $x_j < t$ and $x_j \geq t$ for some t.

(a) Compute the mean of Y within each group, and the deviance within each group at the within-group means.

(b) Record the drop in deviance from step 1.

3. Partition the data into two groups by $x_j < t$ and $x_j \geq t$ that resulted in the largest drop in deviance.

This is the first step in the process.

- We now have a partition into two groups.
- Now apply the same procedure separately to the two groups. This will result in partitioning one of the groups in two, resulting in a total of three groups.

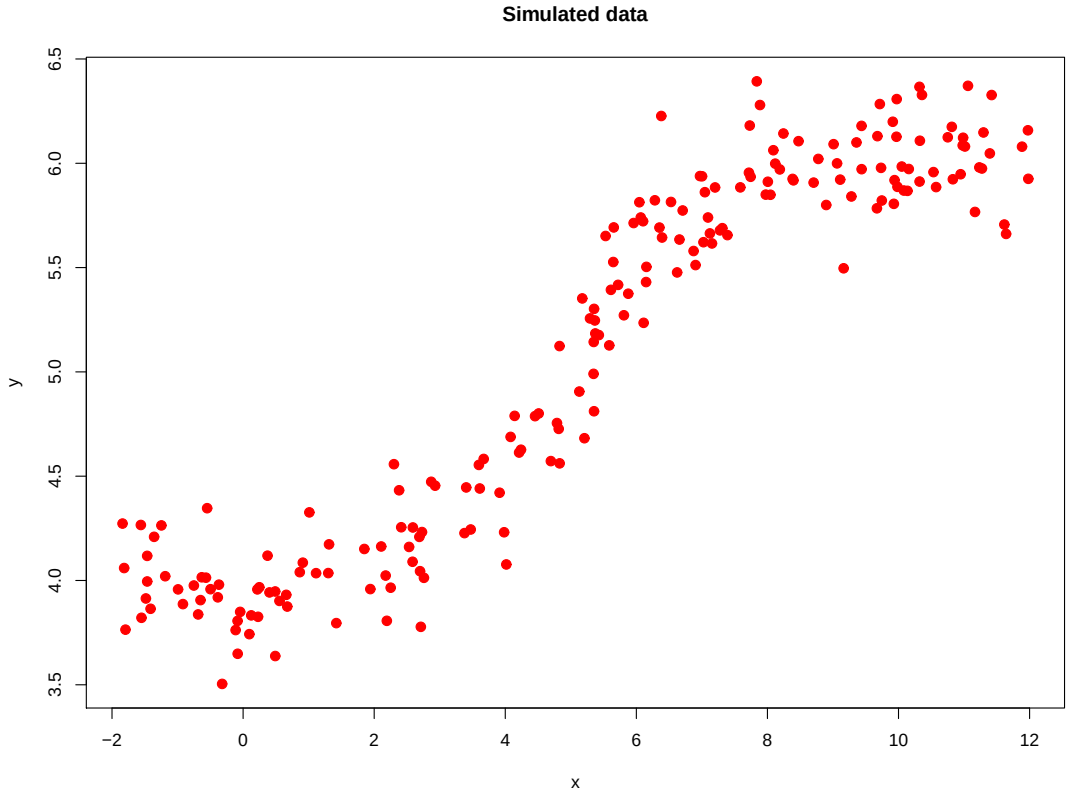
  Note: The same variable can be involved in splitting in this step!
- Repeat this procedure until drops in deviance are not so large (will discuss in more detail later).

This process is sometimes called recursive partitioning.

For categorical variables, sample proportions for each response level are computed at each step of the partitioning, along with the Bernoulli/multinomial deviance.

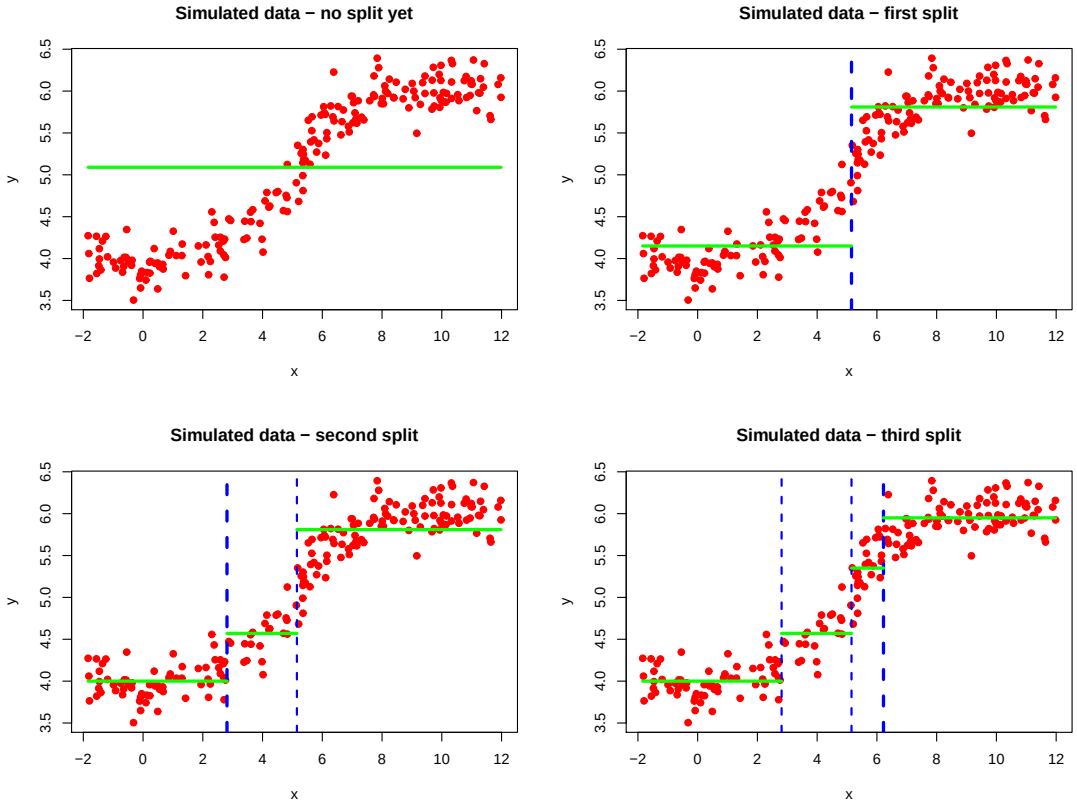
Simple one-variable example:

- Quantitative data were generated as a function of a single predictor variable.
- The relationship between the mean of Y is clearly non-linear with respect to x.



Partitioning steps:

- Determine the mean of the y_i and compute the deviance (sum of squared deviations from the mean).
- Consider all the possible splits in the x variable, compute the mean within the groups, and then compute the drop in total deviance.
- Partition the data at the optimizing split.
- Recursively partition following the same procedure.

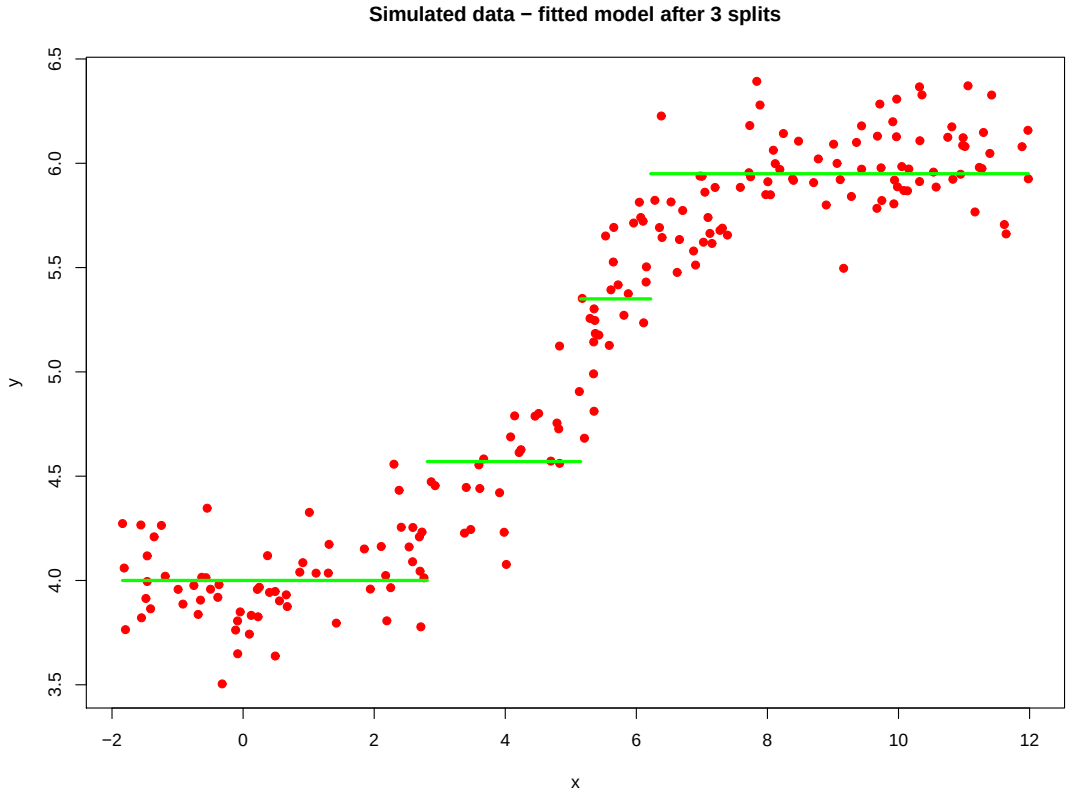


Final estimated model: Based on three splits

- Final estimated model is a step function based on ranges of the x variable.

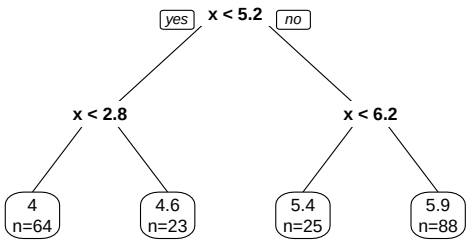
In particular:

$$\hat{\mu}_Y = \begin{cases} 4.00 & \text{if } x < 2.81 \\ 4.57 & \text{if } 2.81 \leq x < 5.15 \\ 5.35 & \text{if } 5.15 \leq x < 6.22 \\ 5.95 & \text{if } 6.22 \leq x \end{cases}$$



Alternative representation of the fitted model: Tree

- The tree represents the partitioning decisions at each step in the algorithm.
- The terminal nodes of the tree are the final partitions.
- This structure is why these approaches are called "tree-based" models.



Example classification tree with multiple predictors: Stage C prostate cancer

It is of interest to know whether prostate cancer returns after surgical removal of the prostate. Data were collected on 146 stage C prostate cancer patients to address this question.

The response is a binary variable indicating whether the cancer has progressed.

Goal: Produce a classification tree for the probability of cancer progression.

Variables in the data frame:

`progressed` - (response) Whether the cancer has progressed (no/yes)
`age` - age (years) at cancer diagnosis
`eet` - early endocrine therapy (no/yes)
`ploidy` - diploid/tetraploid/aneuploid DNA pattern
`g2` - percent of cells in "G_2" phase
`grade` - tumor grade (1-4)
`gleason` - Gleason grade (3-10)

Summaries:

```
> summary(sc)
      age        eet            g2              grade          gleason
 Min.   :47   no  :108   Min.   : 2.400   Min.   :1.00   Min.   : 3.00
 1st Qu.:59   yes : 36   1st Qu.: 9.215   1st Qu.:2.00   1st Qu.: 5.00
 Median :63   NA's:  2   Median :13.010   Median :3.00   Median : 6.00
 Mean   :63              Mean   :14.275   Mean   :2.61   Mean   : 6.35
 3rd Qu.:67              3rd Qu.:16.715   3rd Qu.:3.00   3rd Qu.: 7.00
 Max.   :75              Max.   :54.930   Max.   :4.00   Max.   :10.00
                         NA's   :7                       NA's   :3

        ploidy   progressed
 diploid   :67   no :92
 tetraploid:68   yes:54
 aneuploid :11
```

Splitting criterion for classification trees:

At each stage in the recursive partitioning, compute the sample proportions of the binary response at each terminal node in the tree (i.e., within each partition).

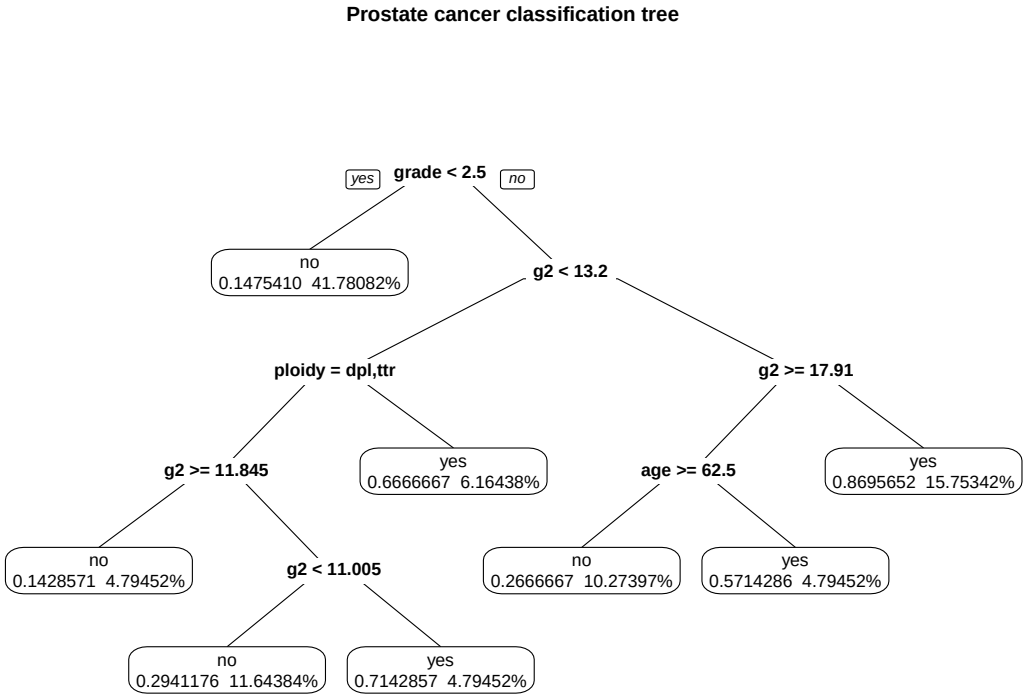
Then, to measure the impact of further splitting on each variable at a specified cut-point, can compute either

- the reduction in the (binomial) deviance statistic, or
- the reduction in the "Gini" statistic (defined as $1 - \sum_k \hat{p}_k^2$ where k indexes partitions).

Choose the variable and cut-point on that variable that produces the largest drop in the deviance (or Gini) statistic in growing the tree.

After loading library `rpart`, fit the model:

```
> sc1.fit = rpart(progressed ~ age+eet+g2+grade+gleason+ploidy,
    data=sc, method="class", parms=list(split="information"))
> sc1.fit


n= 146

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 146 54 no (0.6301370 0.3698630)
   2) grade< 2.5 61  9 no (0.8524590 0.1475410) *
   3) grade>=2.5 85 40 yes (0.4705882 0.5294118)
     6) g2< 13.2 40 17 no (0.5750000 0.4250000)
      12) ploidy=diploid,tetraploid 31 11 no (0.6451613 0.3548387)
        24) g2>=11.845 7  1 no (0.8571429 0.1428571) *
        25) g2< 11.845 24 10 no (0.5833333 0.4166667)
          50) g2< 11.005 17  5 no (0.7058824 0.2941176) *
          51) g2>=11.005 7  2 yes (0.2857143 0.7142857) *
      13) ploidy=aneuploid 9  3 yes (0.3333333 0.6666667) *
     7) g2>=13.2 45 17 yes (0.3777778 0.6222222)
      14) g2>=17.91 22  8 no (0.6363636 0.3636364)
        28) age>=62.5 15  4 no (0.7333333 0.2666667) *
        29) age< 62.5 7  3 yes (0.4285714 0.5714286) *
      15) g2< 17.91 23  3 yes (0.1304348 0.8695652) *
```

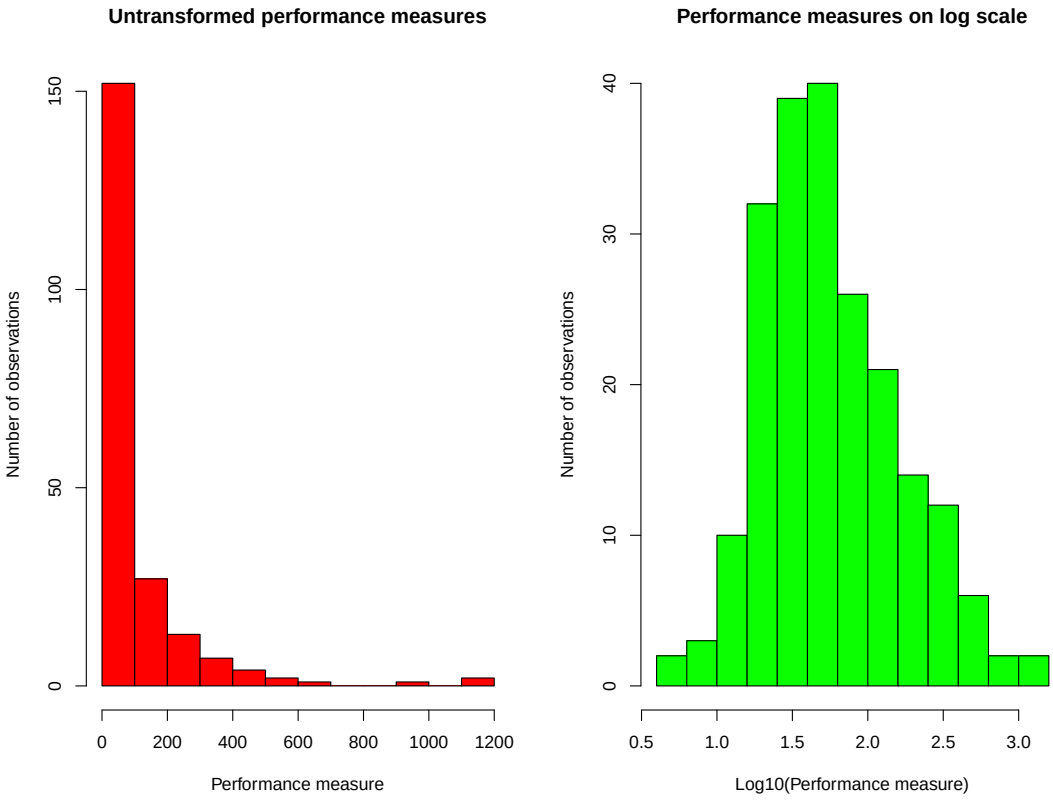


Example regression tree with multiple predictors: CPU performance

A study from 1987 measured the relative performance and characteristics of 209 CPUs.

The response variable is the performance on a benchmark mix relative to an IBM 370/158-3.

Regression trees use sum of squared deviations as the splitting criterion, so worth checking the distribution of the response variable.



Variables in the data frame:

`logperf` - (response) Log (base 10) of CPU performance measure

`syct` - cycle time (nanoseconds)

`mmin` - minimum main memory in Kbytes

`mmax` - maximum main memory in Kbytes

`cach` - cache size in Kbytes

`chmin` - minimum number of channels

`chmax` - maximum number of channels

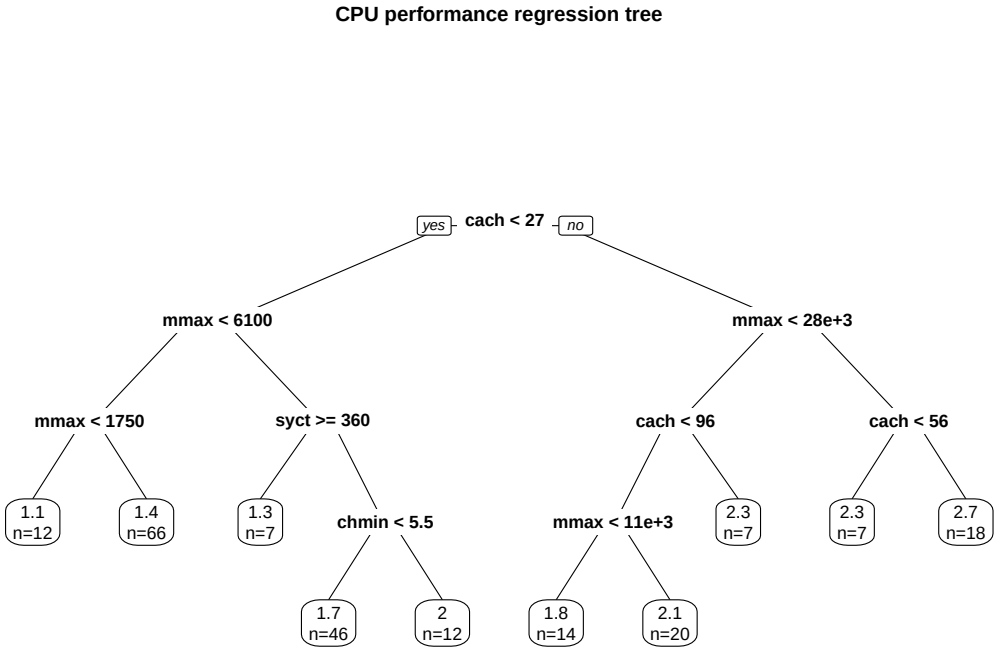
Splitting criterion:

At each stage in the recursive partitioning, compute the sample mean of the response at each terminal node in the tree (within each partition).

Then, to measure the impact of further splitting on each variable at a specified cut-point, compute the sum of squared deviations of each value from the mean within each partition (i.e., the normal deviance statistic).

Choose the variable and cut-point on that variable that produces the largest drop in the deviance statistic in growing the tree.

```
> cpus1.fit = rpart(logperf ~ syct+mmin+mmax+cach+chmin+chmax,
    data=cpus, method="anova")
> cpus1.fit


node), split, n, deviance, yval
      * denotes terminal node

 1) root 209 43.1155400 1.753333
   2) cach< 27 143 11.7908500 1.524647
     4) mmax< 6100 78  3.8937440 1.374824
       8) mmax< 1750 12  0.7842516 1.088732 *
       9) mmax>=1750 66  1.9487330 1.426840 *
     5) mmax>=6100 65  4.0452030 1.704434
      10) syct>=360 7  0.1290809 1.279749 *
      11) syct< 360 58  2.5012470 1.755690
        22) chmin< 5.5 46  1.2262290 1.698613 *
        23) chmin>=5.5 12  0.5507131 1.974483 *
   3) cach>=27 66  7.6426350 2.248821
     6) mmax< 28000 41  2.3414170 2.061986
      12) cach< 96.5 34  1.5919510 2.008124
        24) mmax< 11240 14  0.4246237 1.826635 *
        25) mmax>=11240 20  0.3834013 2.135166 *
      13) cach>=96.5 7  0.1717302 2.323601 *
     7) mmax>=28000 25  1.5228630 2.555230
      14) cach< 56 7  0.0692943 2.268365 *
      15) cach>=56 18  0.6535127 2.666788 *
```



When to stop growing the tree:

Problem: Deviance always decreases when growing a tree.

Some ad hoc aspects of controlling tree growth:

- Ensure a minimum number of observations in a terminal node (6 by default in `rpart`)
- Ensure a minimum number of observations in a node before the algorithm will even consider partitioning (20 by default in `rpart`)

But there is a more fundamental concept to incorporate.

Comparison to smoothers:

- For smoothers, without controlling the neighborhood size (or penalty parameter for wiggliness), the more the model fit would "connect the dots."
- For tree models, the larger the tree is grown (more partitioning of the data), the fewer observations in the terminal nodes:

  This is just like connecting the dots!

For smoothers, we solved the problem through penalties for being too wiggly, and used cross-validation to estimate the penalty parameter.

Can we use the same principle here?

YES!

Penalty idea in trees: Penalize based on the number of partitions

Let α be a (positive-valued) penalty parameter. Rather than computing the deviance of a tree prior to splitting, compute

$$D_\alpha = D + \alpha \cdot \text{number of partitions}$$

Notice that

- if $\alpha = 0$, then a tree would be grown without stopping (except based on ad hoc rules)
- if $\alpha = \infty$ (or just $\alpha \geq D^{(0)}$, the deviance of the root tree), then a tree would not grow beyond the root ("null model")

So α has a similar interpretation to λ in smoothing splines.

Estimating α: Cross-validation

Consider a set of candidate values of α.

- For each α, grow a tree leaving out observation i, but use penalized deviance to stop growing the tree.
- Predict the mean response (or probability, if categorical) for the withheld observation, and compute a predictive deviance for that value.
- Repeat for all i in the data set, and sum the predictive deviances.

Choose the α that minimizes the sum of predictive deviances.

More common to withhold larger collections of data rather than one at a time, e.g., 10-fold cross-validation.

Tree fitting process:

Given that we need to penalize large trees, the accepted process is

1. Fit a tree without a penalty – this will eventually be pruned.
2. Independently perform cross-validation along with the penalized deviance criterion to estimate α.
3. Prune the original tree from step 1 using the deviance criterion with the estimated penalty.

In `rpart`, the process is operationalized in the following way.

Complexity parameter: `cp`

Let $D^{(0)}$ be the deviance of the null model (i.e., full data set without partitioning), and $\hat{\alpha}$ the estimated penalty parameter.

Then

$$\texttt{cp} = \hat{\alpha}/D^{(0)}$$

Because deviances for fitted trees that involve any partitioning will be less than $D^{(0)}$, the value of `cp` will be between 0 and 1.

In other words, `cp` is a rescaling of $\hat{\alpha}$ to the interval of $[0, 1]$. For R, we want to estimate `cp` rather than α.

Key information to determine `cp`: The `printcp` command

For prostate cancer example:

```
> printcp(sc1.fit)

        CP nsplit rel error  xerror    xstd
1 0.104938      0   1.00000 1.00000 0.10802
2 0.055556      3   0.68519 0.94444 0.10668
3 0.027778      4   0.62963 0.88889 0.10511
4 0.018519      6   0.57407 0.88889 0.10511
5 0.010000      7   0.55556 0.83333 0.10332
```

These are the results of 10-fold cross-validation for different `CP` values.

Columns of `printcp` output:

`CP` - candidate complexity parameter (`cp`)

`nsplit` - number of times the data are split (add 1 to get the number of partitions)

`rel error` - deviance measure divided by $D^{(0)}$

`xerror`, `xstd` - 10-fold cross-validated deviance statistic and its standard error based on growing the tree by `nsplit` splits, relative to the value obtained for the null model

Choosing the `cp` value:

Two typical options:

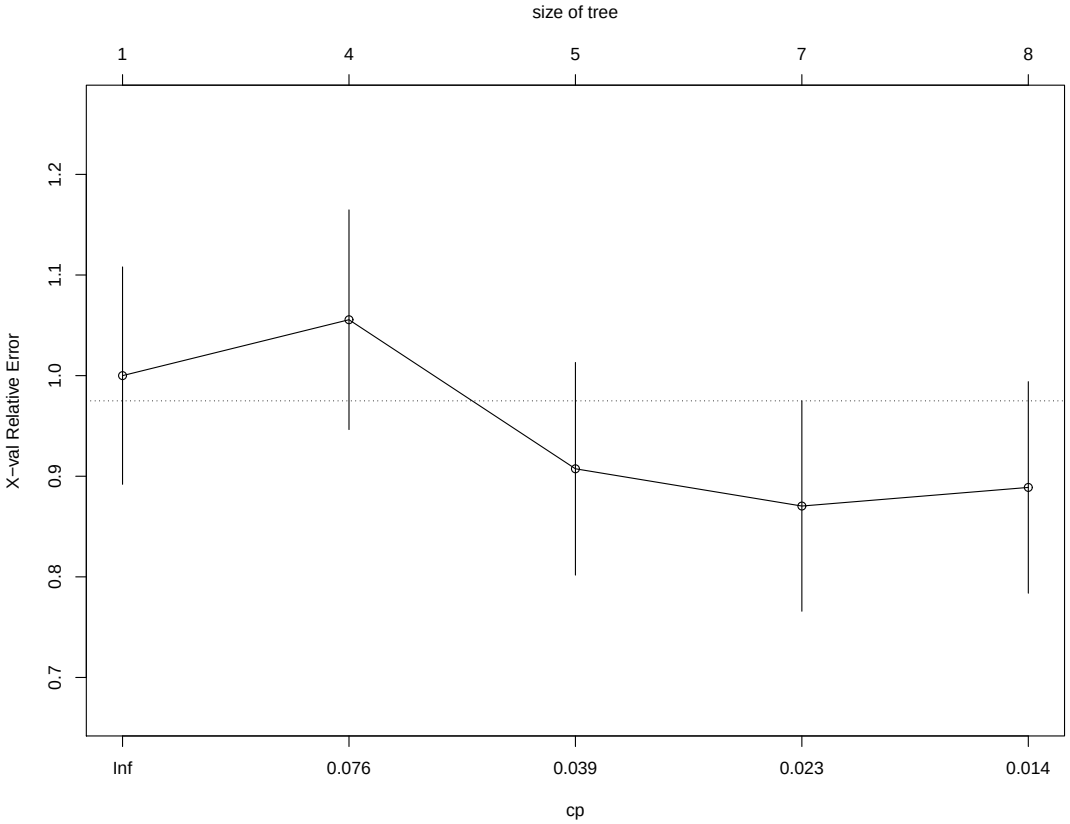
- Choose the value of `CP` where `xerror` is the smallest

- Preferred method: The "1-SE rule."

  Identify the smallest `xerror`. Now add the corresponding `xstd` to `xerror` and choose the largest `CP` that has its associated `xerror` less than this sum.

Easy to see this using the `plotcp` command. Warning – the `cp` values are likely to be different!

```
> plotcp(sc1.fit)
```



Results of 1-SE rule:

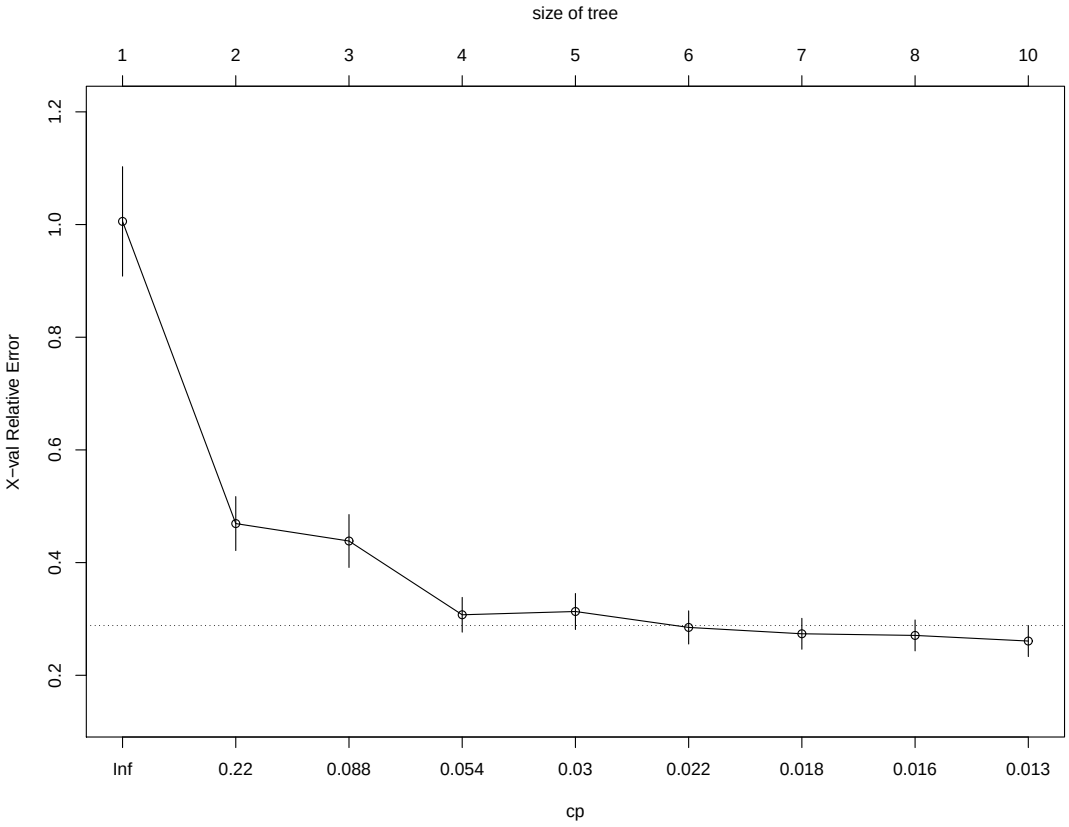
Choose `cp = 0.039`.

Now prune the tree with this value of cp:

```
> sc2.fit = prune(sc1.fit, cp=0.039)
> print(sc2.fit)

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 146 54 no (0.6301370 0.3698630)
   2) grade< 2.5 61  9 no (0.8524590 0.1475410) *
   3) grade>=2.5 85 40 yes (0.4705882 0.5294118)
     6) g2< 13.2 40 17 no (0.5750000 0.4250000)
      12) ploidy=diploid,tetraploid 31 11 no (0.6451613 0.3548387) *
      13) ploidy=aneuploid 9  3 yes (0.3333333 0.6666667) *
     7) g2>=13.2 45 17 yes (0.3777778 0.6222222)
      14) g2>=17.91 22  8 no (0.6363636 0.3636364) *
      15) g2< 17.91 23  3 yes (0.1304348 0.8695652) *
```

Optimizing the cp parameter for CPU performance:

```
> printcp(cpus1.fit)

        CP nsplit rel error  xerror     xstd
1 0.549270      0   1.00000 1.01213 0.097707
2 0.089339      1   0.45073 0.47851 0.048547
3 0.087633      2   0.36139 0.45352 0.045812
4 0.032816      3   0.27376 0.32970 0.031125
5 0.026922      4   0.24094 0.31007 0.030019
6 0.018556      5   0.21402 0.27865 0.027265
7 0.016799      6   0.19546 0.27612 0.026548
8 0.015791      7   0.17866 0.27454 0.027829
9 0.010000      9   0.14708 0.26733 0.028672
```



The 1-SE rule suggests choosing `cp = 0.022`.

```
> cpus2.fit = prune(cpus1.fit, cp=0.022)
> print(cpus2.fit)
node), split, n, deviance, yval
      * denotes terminal node

 1) root 209 43.1155400 1.753333
   2) cach< 27 143 11.7908500 1.524647
     4) mmax< 6100 78  3.8937440 1.374824
       8) mmax< 1750 12  0.7842516 1.088732 *
       9) mmax>=1750 66  1.9487330 1.426840 *
     5) mmax>=6100 65  4.0452030 1.704434
      10) syct>=360 7  0.1290809 1.279749 *
      11) syct< 360 58  2.5012470 1.755690 *
   3) cach>=27 66  7.6426350 2.248821
     6) mmax< 28000 41  2.3414170 2.061986 *
     7) mmax>=28000 25  1.5228630 2.555230 *
```

Prediction:

After fitting and pruning the tree, predicted means/probabilities for new observations can be read off the fitted tree.

Example: For a new CPU with `cach=24`, `mmax=9000`, and `syct=400`, then based on the above pruned tree the estimated log-performance is $\boxed{1.2797}$.

Within R:

```
> cpus.new = data.frame(cach=24,mmax=9000,syct=400,mmin=2000,chmin=2,chmax=5)
> predict(cpus2.fit, cpus.new)
       1
1.279749
```

Need to include all predictor variables, though the last three will be irrelevant to the predictions (unless any of the first three had missing values).

Prediction for classification tree:

Suppose a new patient who had surgery for prostate cancer had `grade=3.0`, `g2=14.0`, and `ploidy=aneuploid`. Then according to the final pruned tree the probability is $\boxed{0.8696}$ the patient's cancer has progressed.

In R:

```
> sc1new = data.frame(grade=3.0, g2=14.0, ploidy="aneuploid",
    age=63, eet="no",gleason=6.0)
> predict(sc2.fit,sc1new)
         no       yes
1 0.1304348 0.8695652
```

Missing values:

An interesting feature of `rpart` is in how it handles missing values.

When encountering a missing value on a variable (denoted `NA`), a surrogate variable is used on which to split (though not reported).

Example in R:

```
> sc2new = sc1new
> sc2new$grade = as.numeric(NA)  # force the value to be missing
> predict(sc2.fit,sc2new)
         no       yes
1 0.1304348 0.8695652
```

Comments on tree models:

- In addition to categorical and (approximately) normal responses, can also fit Poisson data and time-to-response data (see `rpart` help page).
- One use of tree models is to perform variable selection. The predictors that make it into the final pruned tree can then be used for more conventional modeling.
- A side-benefit of tree models is that they incorporate predictor interactions automatically. Again, this can be helpful if the fitted tree is used as a first step in conventional modeling (e.g., GLM).
- Recursive partitioning is a greedy algorithm. Each step in the recursion, the best split is sought. The final tree is a therefore "local" optimum, but not necessarily a "global" optimum (but this is at the price of computational expense).

Random forests:

Extension of tree models.

Attempts to address problems associated with

- the greediness of the CART algorithm, and
- the instability/high variance of CART solutions.

The solution is to generate many "random" trees and average their contribution to construct an ensemble prediction.

Algorithm for constructing random forests:

Assume a sample of size n.

1. Draw `ntree` "bootstrap" samples from the original data set of size n. A bootstrap sample of size n is a random sample with replacement of the original data. On average, about 37% of the original data will not be part of each bootstrap sample.

   The value of `ntree` should be large (in R, the default is 500).
2. For each bootstrap sample, fit a classification or regression tree without pruning (ad hoc minimums per final partitions are okay for regression trees), but at each node choose the best split among `mtry` randomly selected predictors (rather than all predictors).
3. To make predictions, average the results across the `ntree` trees.

This procedure is an example of

- model averaging
- "bagging" (bootstrap aggregating)

With random forests, noteworthy that

- prediction algorithm is a "black box" – very difficult to visualize.
- only requires specifying `ntree` and `mtry` to obtain final predictions, and these parameters are not required to be estimated through cross-validation.

Random forest for classification: Stage C cancer

(in R, `randomForest` implements the Gini index criterion, not the deviance criterion)

```
> sc.rf = randomForest(progressed ~ age+eet+g2+grade+gleason+ploidy,
    data=sc, na.action=na.roughfix)
> sc.rf
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 36.99%
Confusion matrix:
    no yes class.error
no  66  26   0.2826087
yes 28  26   0.5185185
```

Rows in the confusion matrix are the "truth" and the columns are the majority estimates.

Interpreting the output:

Classification predictions are discrete (e.g., binary) for each tree, not estimated probabilities.

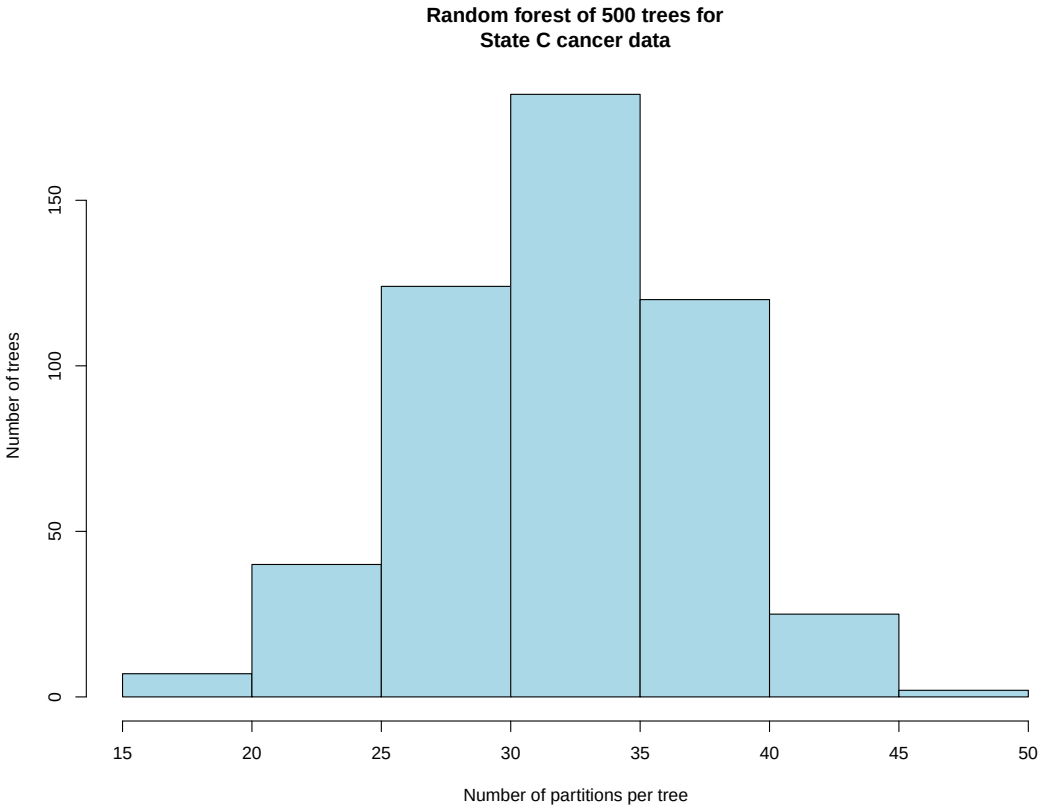
- `Confusion matrix`: Because each of the `ntree` trees have roughly 37% of the data excluded, these left-out observations ("out-of-bag", OOB) can be used for a predictive error measure.

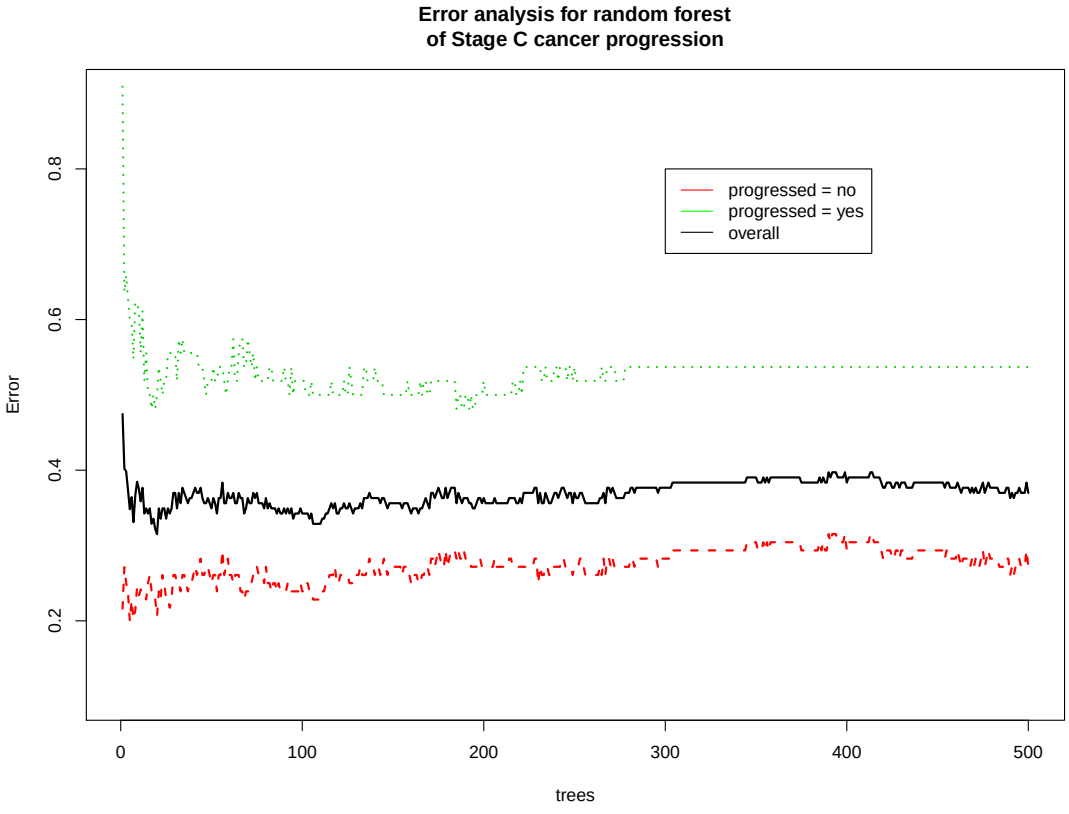
  For classification problems, the reported entries in the confusion matrix is determined by first making the (binary) prediction for each tree, and then use majority rule for all `ntree` trees.

- `OOB estimate of error rate`: Overall misclassification error rate.

Other details:

- The default number of trees is 500, and the default number of (randomly selected) predictors to try at each tree node is `mtry`$= \sqrt{J}$, with J the number of predictors.
- We used the defaults, but these can be specified explicitly.
- There are many other "flavors" of random trees that involve growing small trees, different tree algorithms, etc. This is an active area of research.


Random forest of 500 trees for
State C cancer data
Number of trees
0
50
100
150
15
20
25
30
35
40
45
50
Number of partitions per tree


Error analysis for random forest
of Stage C cancer progression
Error
0.2
0.4
0.6
0.8
progressed = no
progressed = yes
overall
0
100
200
300
400
500
trees

Prediction:

Interested in only the first observation, but need to trick the `predict` command so it recognizes the levels of the factor variables.

```
> sc1new = data.frame(age=c(63,60,60), eet=c("no","yes","yes"),
  g2=c(14,13,12), grade=c(3,2,4),
  gleason=c(6,7,8),ploidy=c("aneuploid","diploid","tetraploid"))
> predict(sc.rf,sc1new,type="prob")

     no    yes
1 0.626 0.374       <---------
2 0.614 0.386
3 0.444 0.556
```

These are all proportions out of `ntree=500`.

From `rpart`:

```
> sc1new = data.frame(grade=3.0, g2=14.0, ploidy="aneuploid",
    age=63, eet="no",gleason=6.0)
> predict(sc2.fit,sc1new)

         no       yes
1 0.1304348 0.8695652
```

Wildly different!

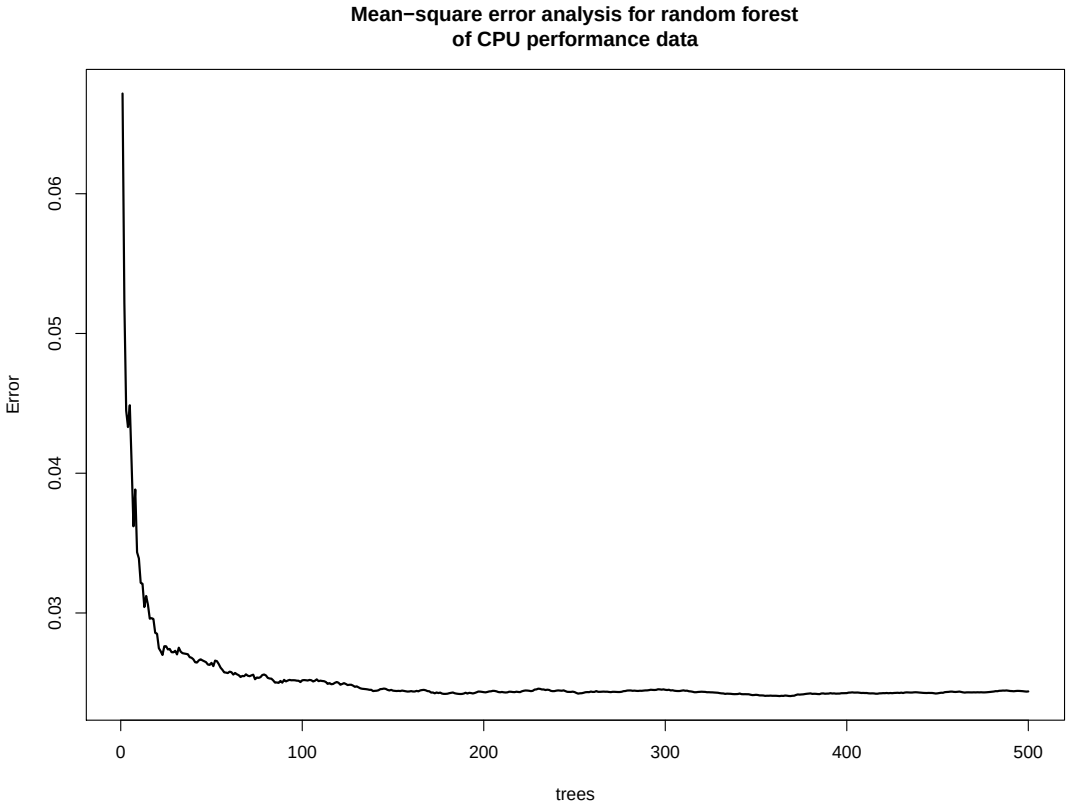
Random forest for quantitative responses: CPU data

```
> cpus.rf = randomForest(logperf ~
   syct+mmin+mmax+cach+chmin+chmax,
   data=cpus)
> cpus.rf

               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 2

          Mean of squared residuals: 0.02440099
                    % Var explained: 88.17
```

By comparison, R^2 for least-squares regression is 81.3%.

Details and interpretations:

- For each observation i, we can compute an "out-of-bag" prediction $\hat{y}_i^{OOB}$ by averaging each tree-based prediction in which observation i is not in the bootstrap sample.
- A (predictive) residual is then $(y_i - \hat{y}_i^{OOB})$.
- R^2 is defined as
$$1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i^{OOB})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$
- Default value of `mtry` is $J/3$ for quantitative response random forest models (as opposed to $\sqrt{J}$).
- By default every tree node must contain at least 5 observations (for classification, 1 is the minimum).

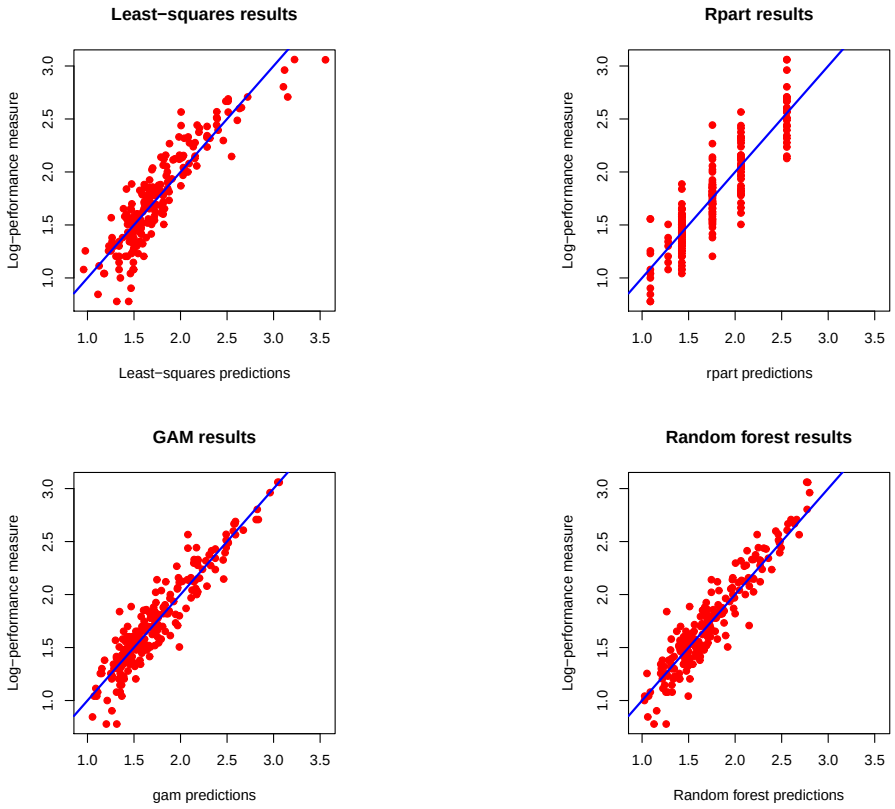


Prediction:

```
> cpus.new = data.frame(cach=24,mmax=9000,
   syct=400,mmin=2000,chmin=2,chmax=5)
> predict(cpus.rf, cpus.new)
       1
1.671448
```

From `rpart`:

```
1.279749
```



Comments on random forests:

- The more predictor variables you have, the more trees you should include in your forest.
- If you have a large number of variables but only expect a few to be predictive, setting `mtry` to a larger value may give better performance.
- The algorithm is intrinsically parallel. For large data sets, ideally one can run the same algorithm on several machines/cores and aggregate when the runs complete.