Machine Learning

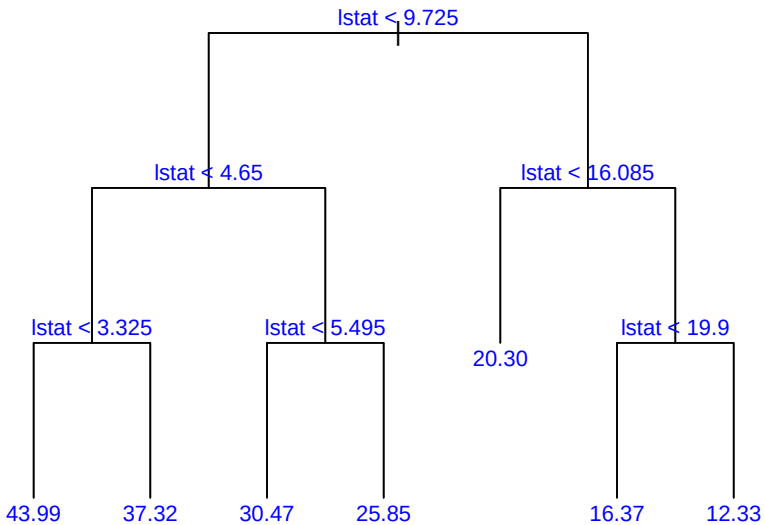
Week 3

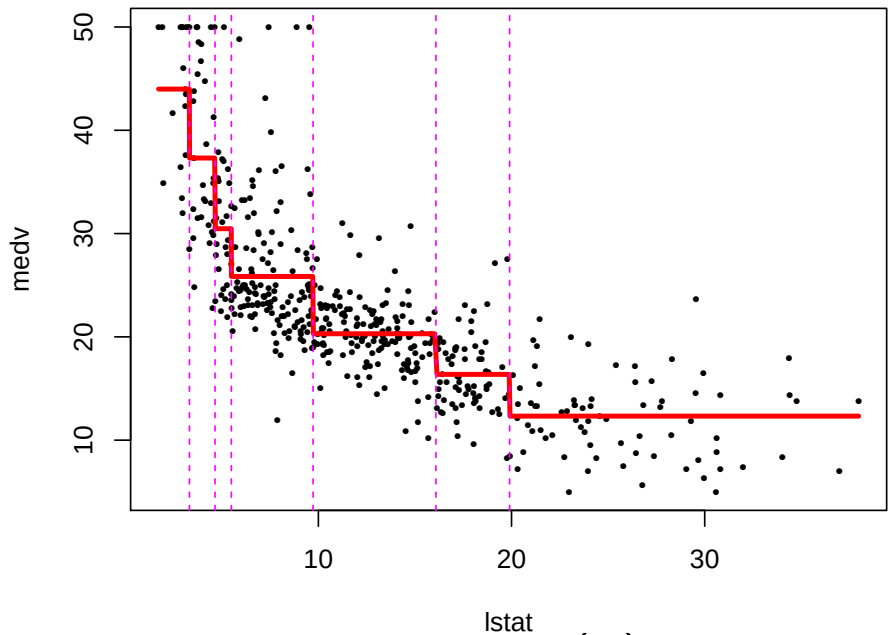
Mladen Kolar (mkolar@chicagobooth.edu)

More on Trees

Example: Boston housing

At left is the tree fit to the data. At each interior node there is a decision rule of the form $\{x < c\}$. If $x < c$ you go left, otherwise you go right. Each observation is sent down the tree until it hits a bottom node or leaf of the tree.

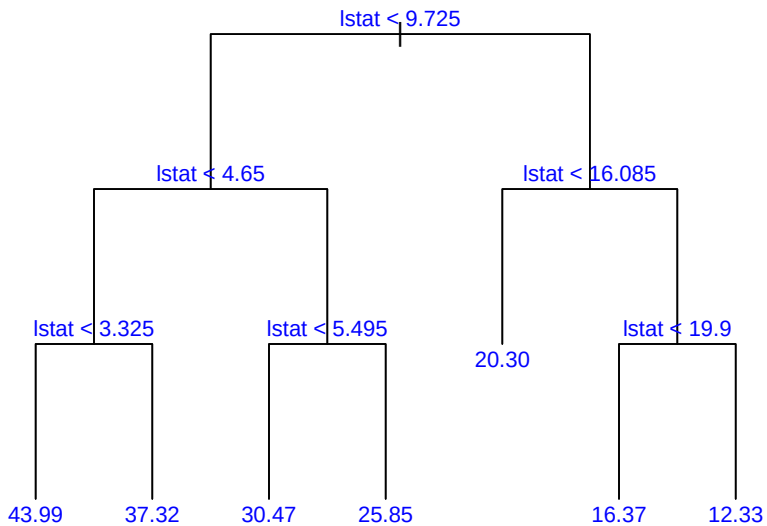


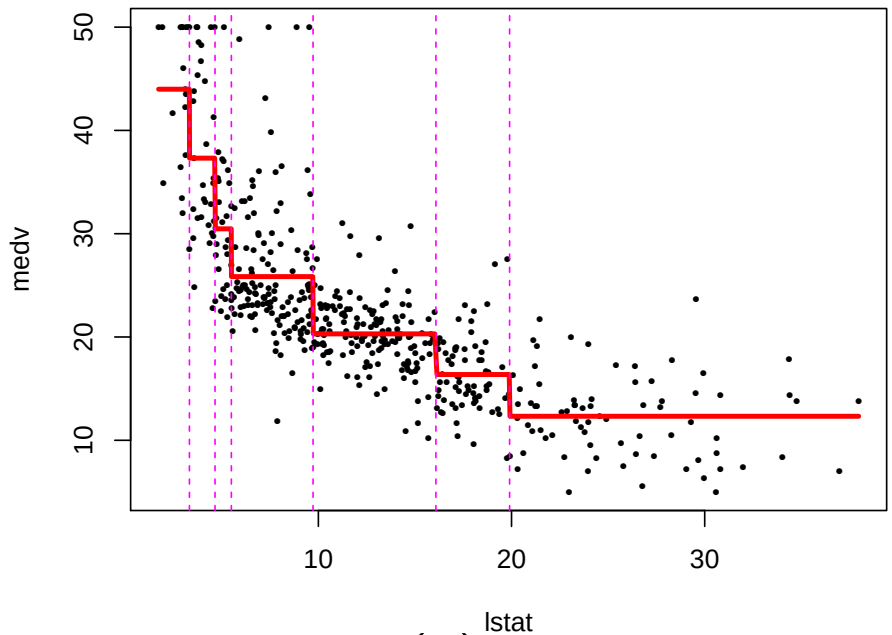


The set of bottom nodes gives us a partition of the predictor (x) space into disjoint regions. At right, the vertical lines display the partition. With just one x, this is just a set of intervals.

Example: Boston housing

Within each region (interval) we compute the average of the y values for the subset of training data in the region. This gives us the step function which is our $\hat{f}$. The $\bar{y}$ values are also printed at the bottom nodes (left plot).



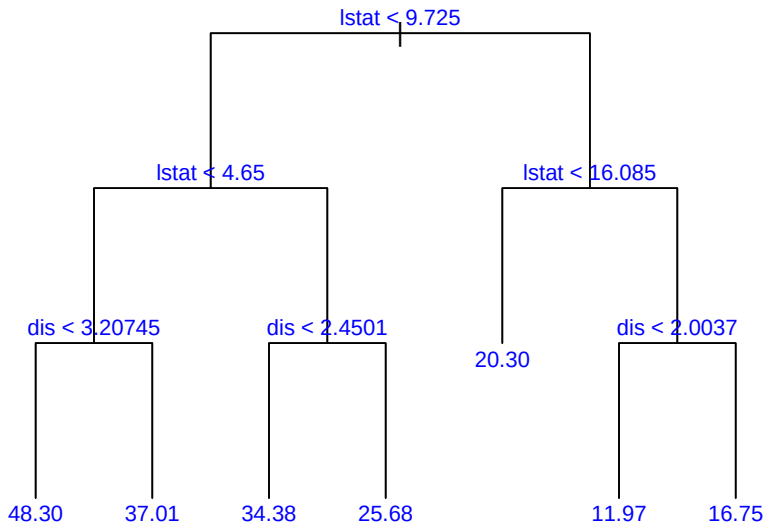


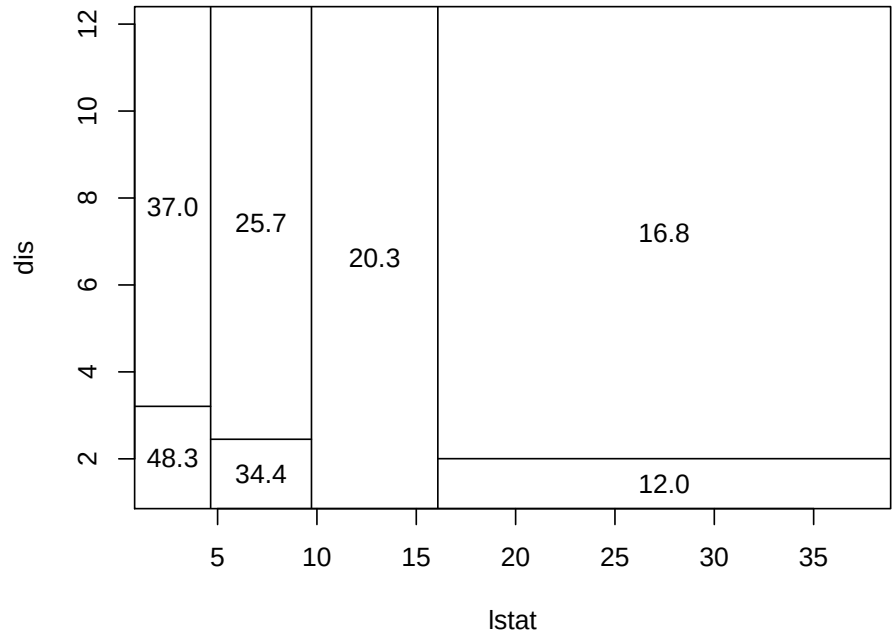
To predict, we just use our step function estimate of $f(x)$.

Equivalently, we drop x down the tree until it lands in a leaf and then predict the average of the y values for the training observations in the same leaf.

Example: Boston housing — two explanatory variables

Here is a tree with $x = (x_1, x_2)$ = (lstat,dis) and y=medv.





At right is the *partition* of the x space corresponding to the set of bottom nodes (leaves).

The average y for training observations assigned to a region is printed in each region and at the bottom nodes.

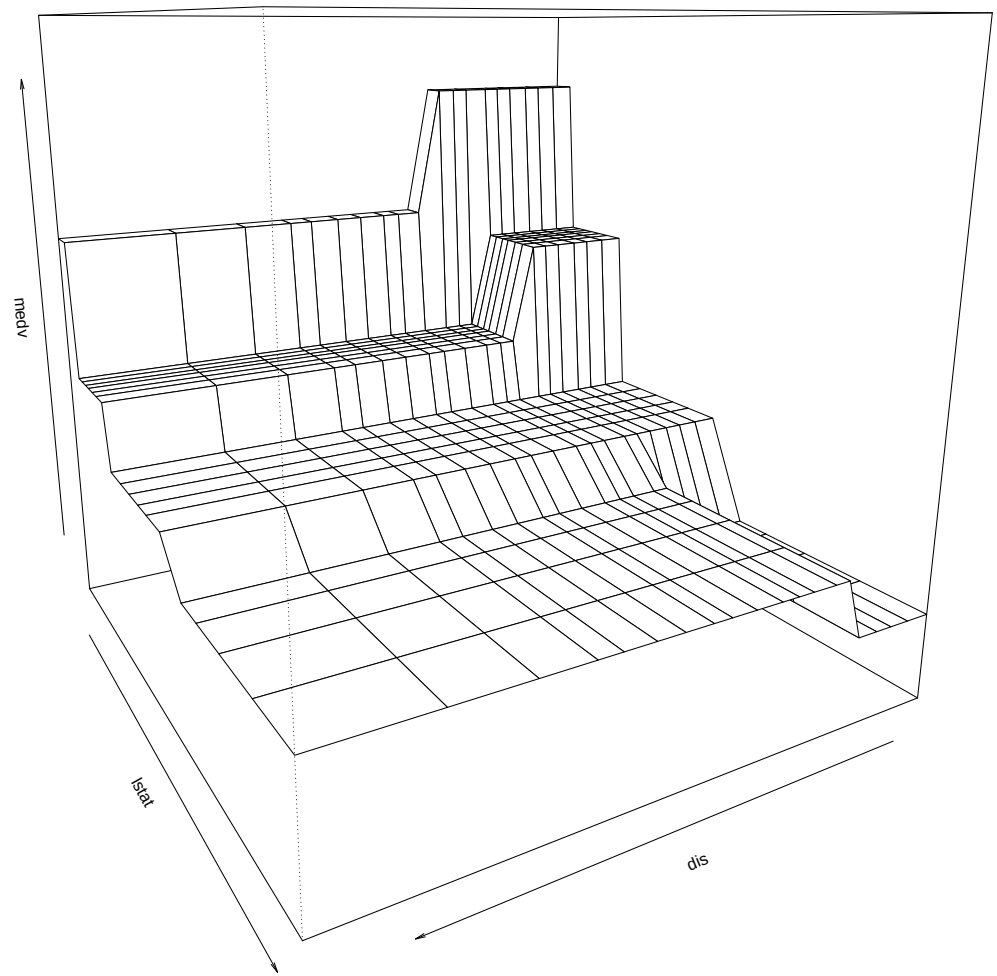
This is the regression function given by the tree.

It is a step function which can seem dumb, but it delivers non-linearity *and* interactions in a simple way and works with a lot of variables.

Notice the interaction.

The effect of `dis`

depends on `lstat`!!



More on prunning

For any tree T, let $|T|$ denote its number of leaves. We define

$$C_\alpha(T) = \sum_{j=1}^{|T|}(1 - \hat{p}_j(R_j)) + \alpha \cdot |T|$$

We seek the tree $T \subseteq T_0$ that minimizes $C_\alpha(T)$.

It turns out that this can be done efficiently using dynamic programming.

Note that α is a tuning parameter, and a larger α yields a smaller tree. CART picks α by 5- or 10-fold cross-validation.

- example in `rpart`

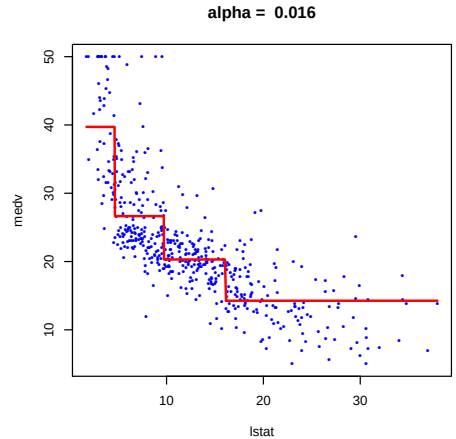
Choosing α

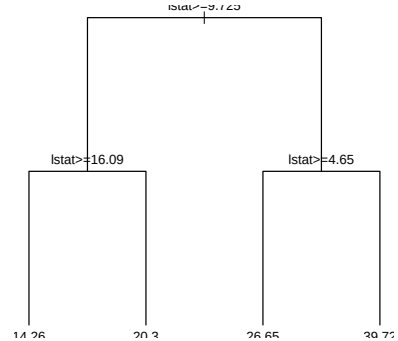
At right are three different tree fits we get from three different α values (using all the data).

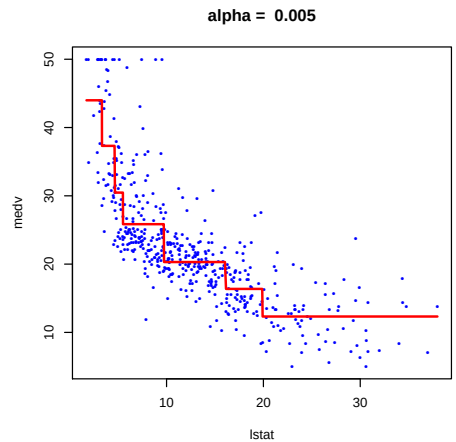
The smaller α is, the lower the penalty for complexity is, the bigger tree you get.

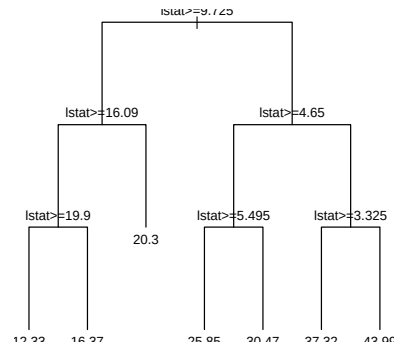
The top tree is a sub-tree of the middle tree, and the middle tree is a sub-tree of the bottom tree.

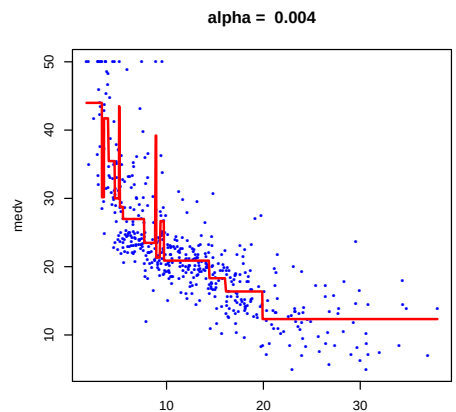
The middle α is the one suggested by CV.

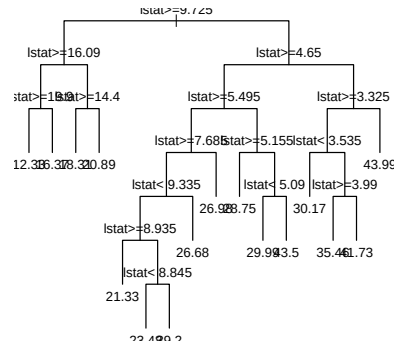






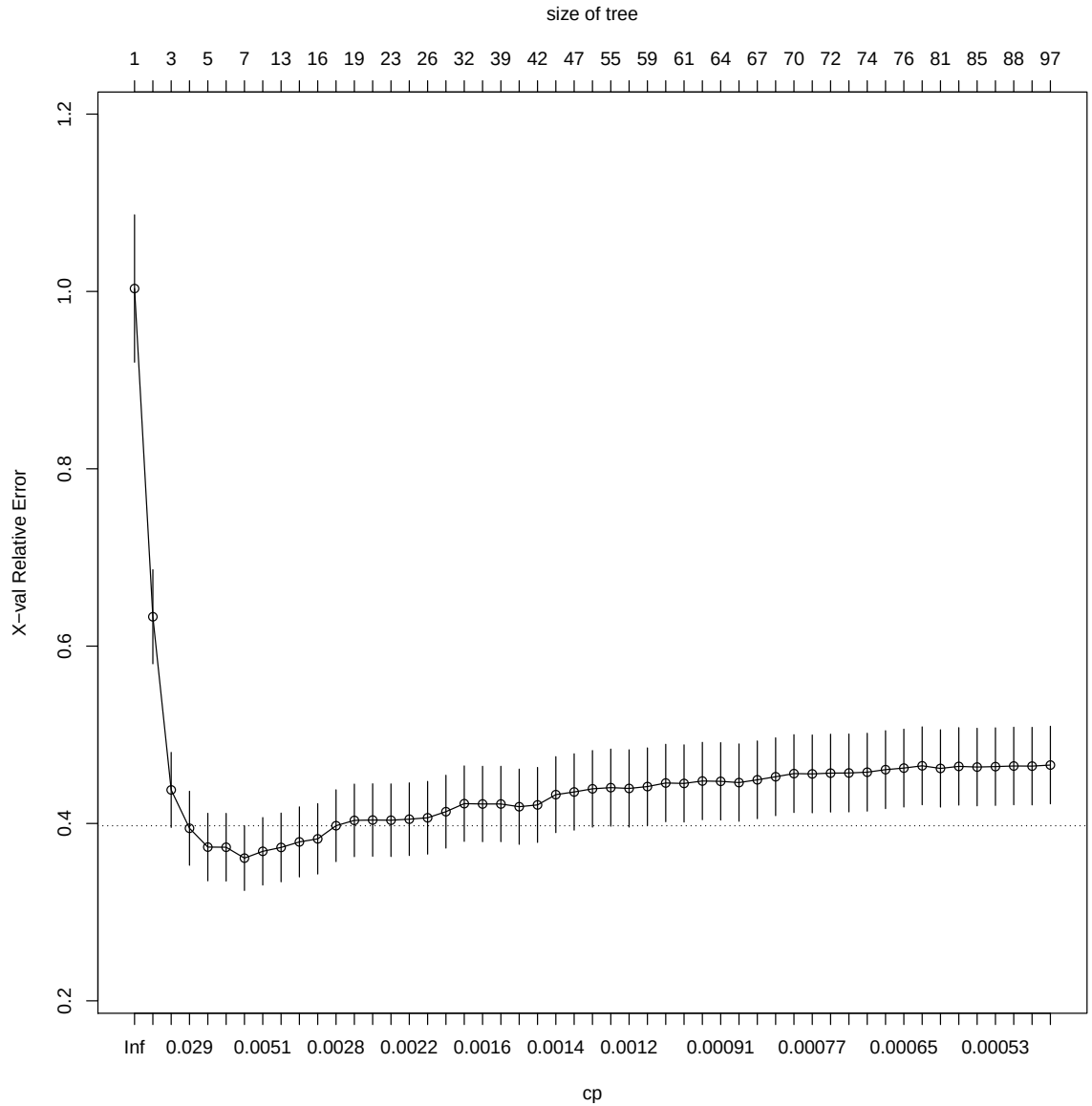






This is the CV plot giving by the R package `rpart` for `y=medv x=lstat`.

The error is relative to the error obtained with a single node (fit is $y = \bar{y}$, $\alpha = \infty$).



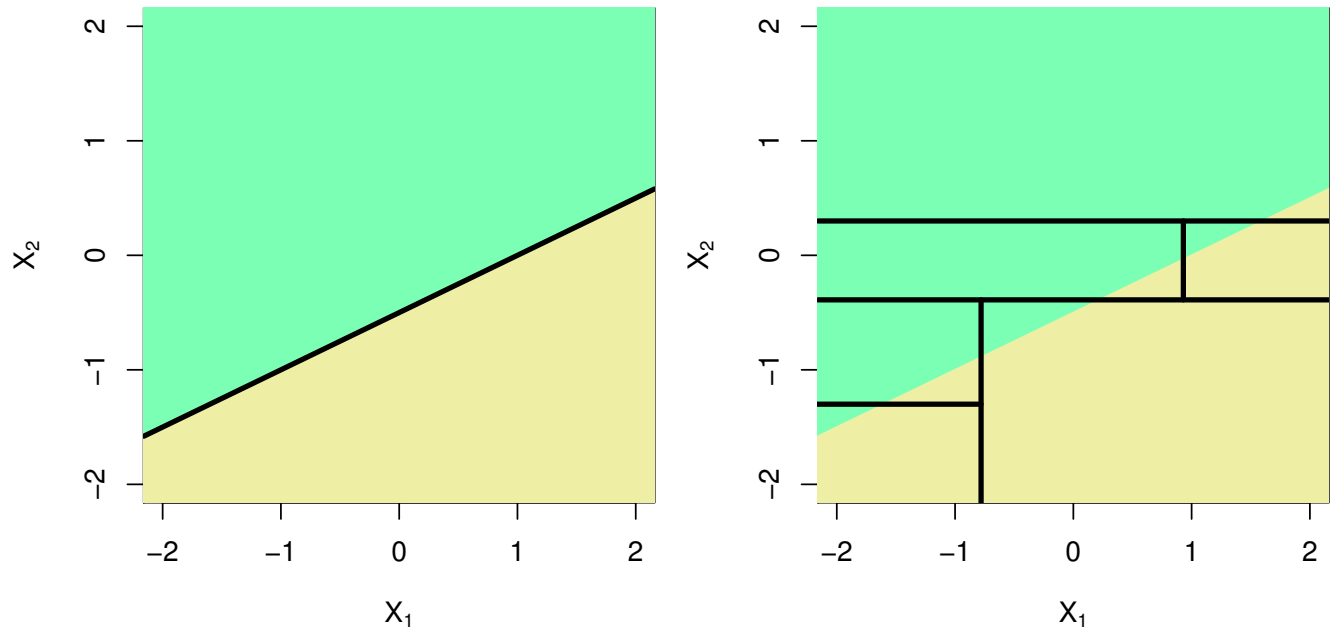
Tree problems?

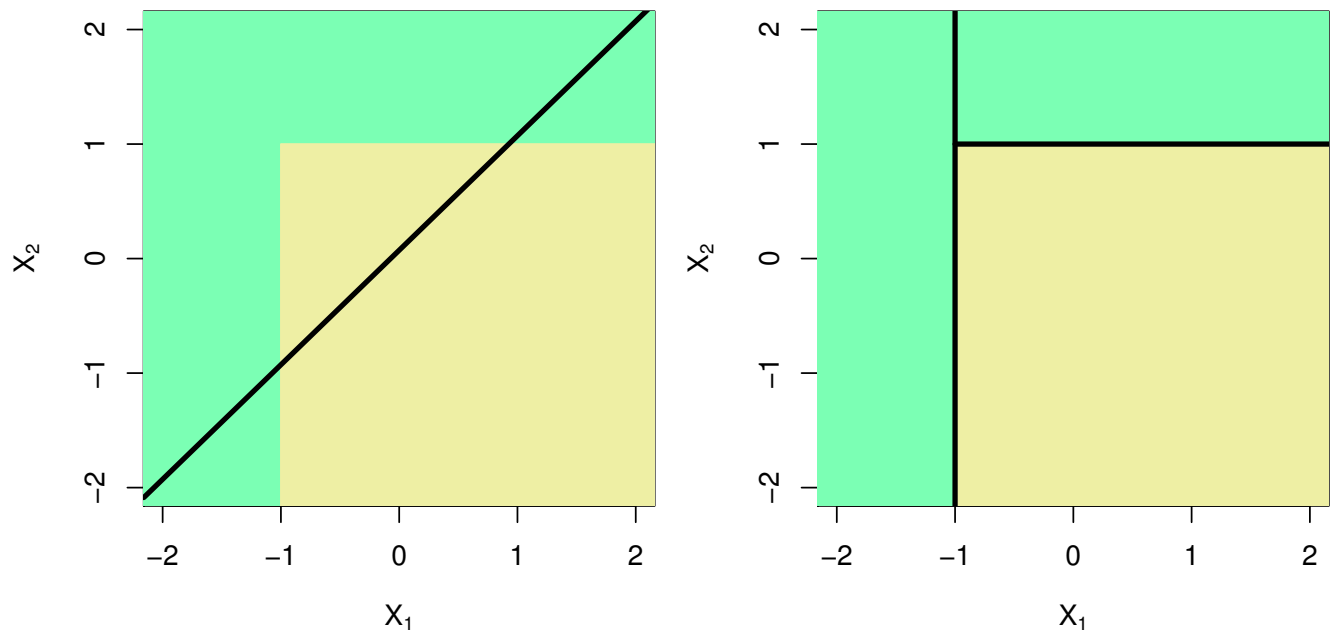
Although they have much to recommend them:

- automatic learning of non-linear response functions with interactions and interpretability,

there are certainly some drawbacks.

- Easy to over-fit, even with CV: deep tree structure creates stability problems, making optimal depth hard to assess.
- No theory to fall back on.
- When regression response surfaces are smooth, the piece-wise constant nature of fits actually strains interpretability as many splits are required to mimic linear and low degree polynomial relationships.


X2
X1
−2
−1
0
1
2


X2
X1
−2
−1
0
1
2

How well do trees predict?

Unfortunately, the answer is not great. Of course, at a high level, the prediction error is governed by bias and variance, which in turn have some relationship with the size of the tree (number of nodes). A larger size means smaller bias and higher variance, and a smaller tree means larger bias and smaller variance.

But trees generally suffer from high variance because they are quite instable: a smaller change in the observed data can lead to a dramatically different sequence of splits, and hence a different prediction. This instability comes from their hierarchical nature; once a split is made, it is permanent and can never be "unmade" further down in the tree.

We'll learn some variations of trees have much better predictive abilities. However, their predictions rules aren't as transparent.

Summary: Trees

Pros:

- easy to explain
- closely mirror human decision-making
- can handle both discrete and continuous variables
- fast

Cons:

- not competitive in terms of accuracy
- non-robust — small changes in data can cause a large change in the estimated tree

Bootstrap and Quantification of uncertainty (detour)

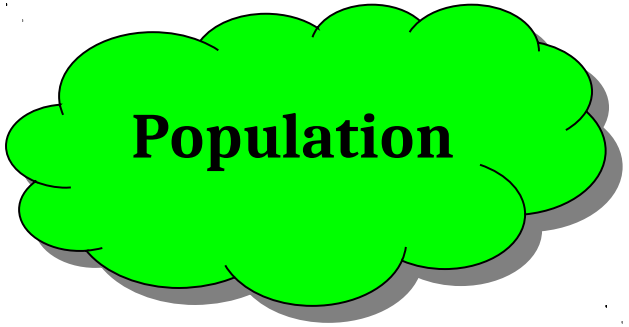
Standard Error

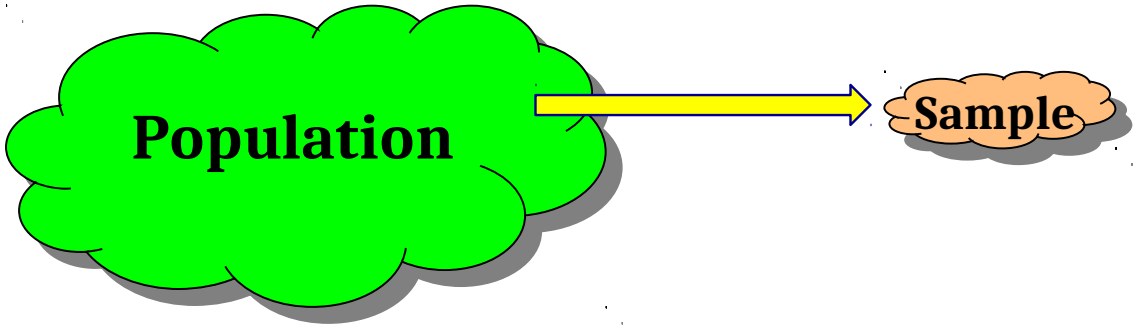
Whenever we compute or estimate a quantity from data, we understand that our estimate is a random quantity.

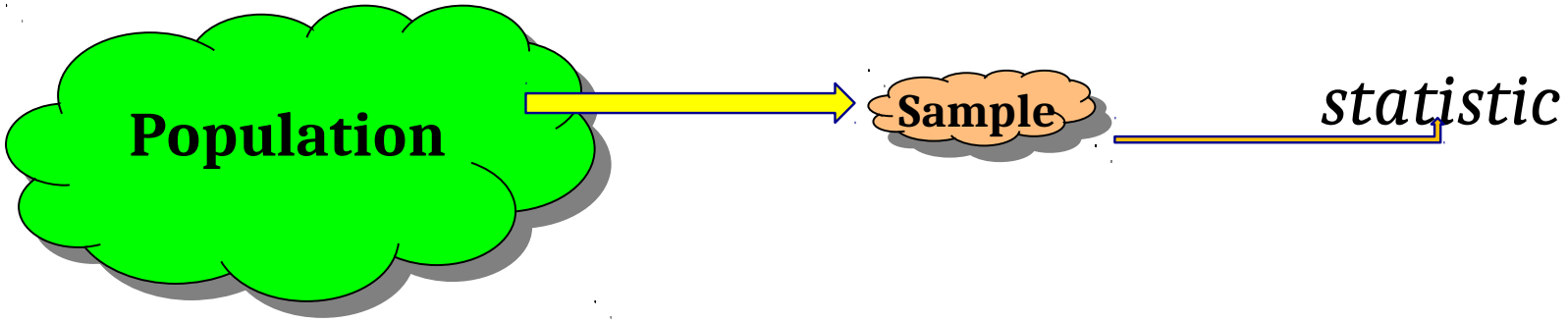
We would like to know is how variable our estimate and also what is the plausible range of values for the unknown parameter.

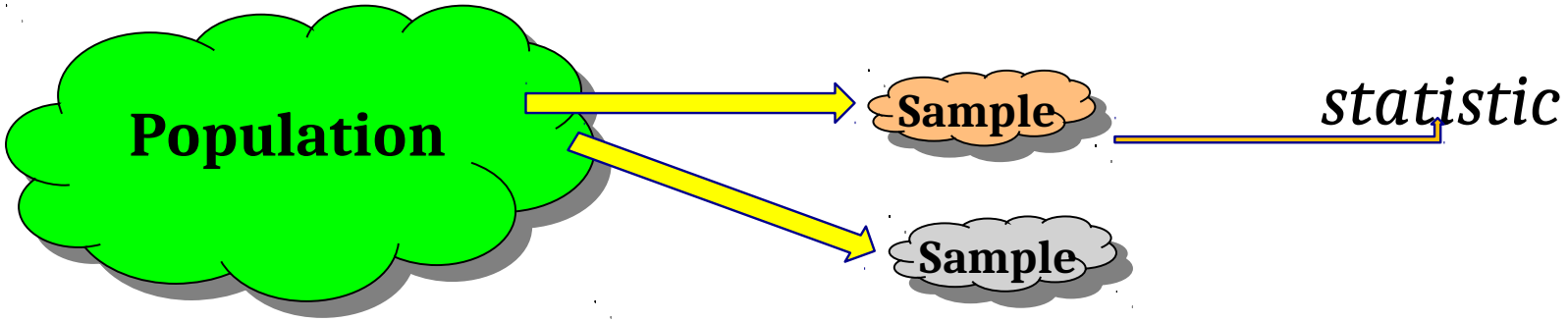
The **standard error** of an estimator, SE, is the standard deviation of the estimator.

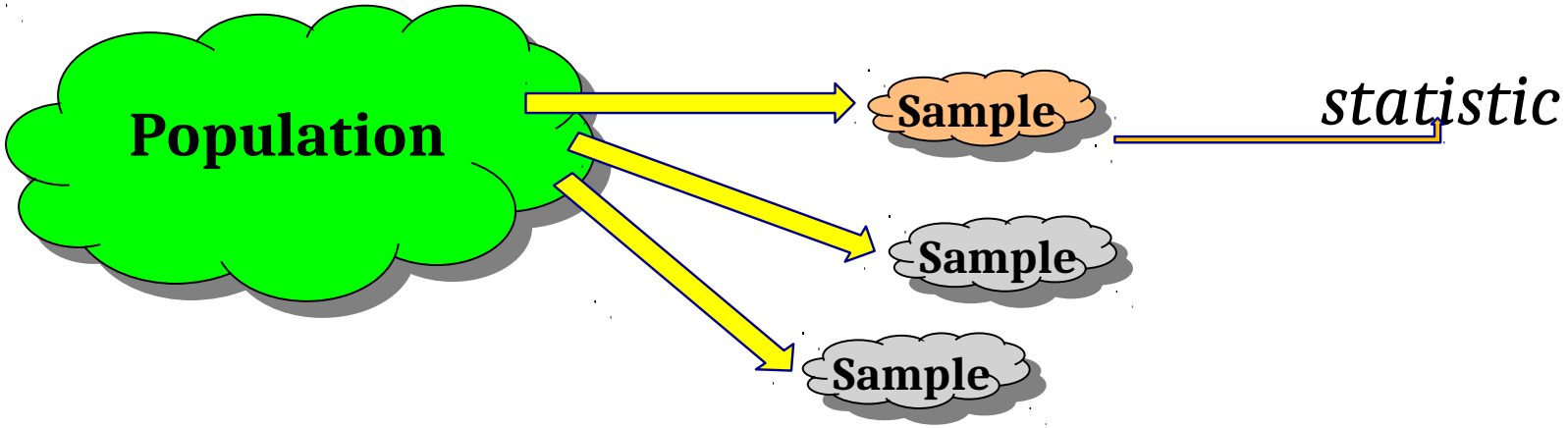
- The standard error can be calculated as the standard deviation of the sampling distribution.

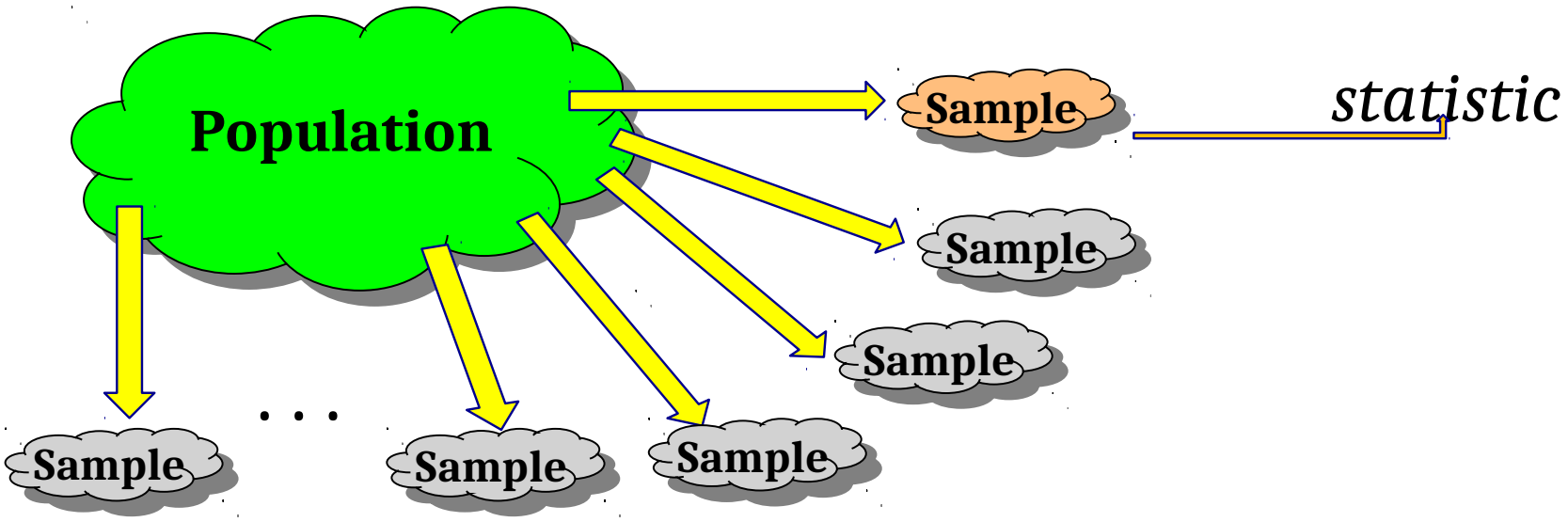

Population

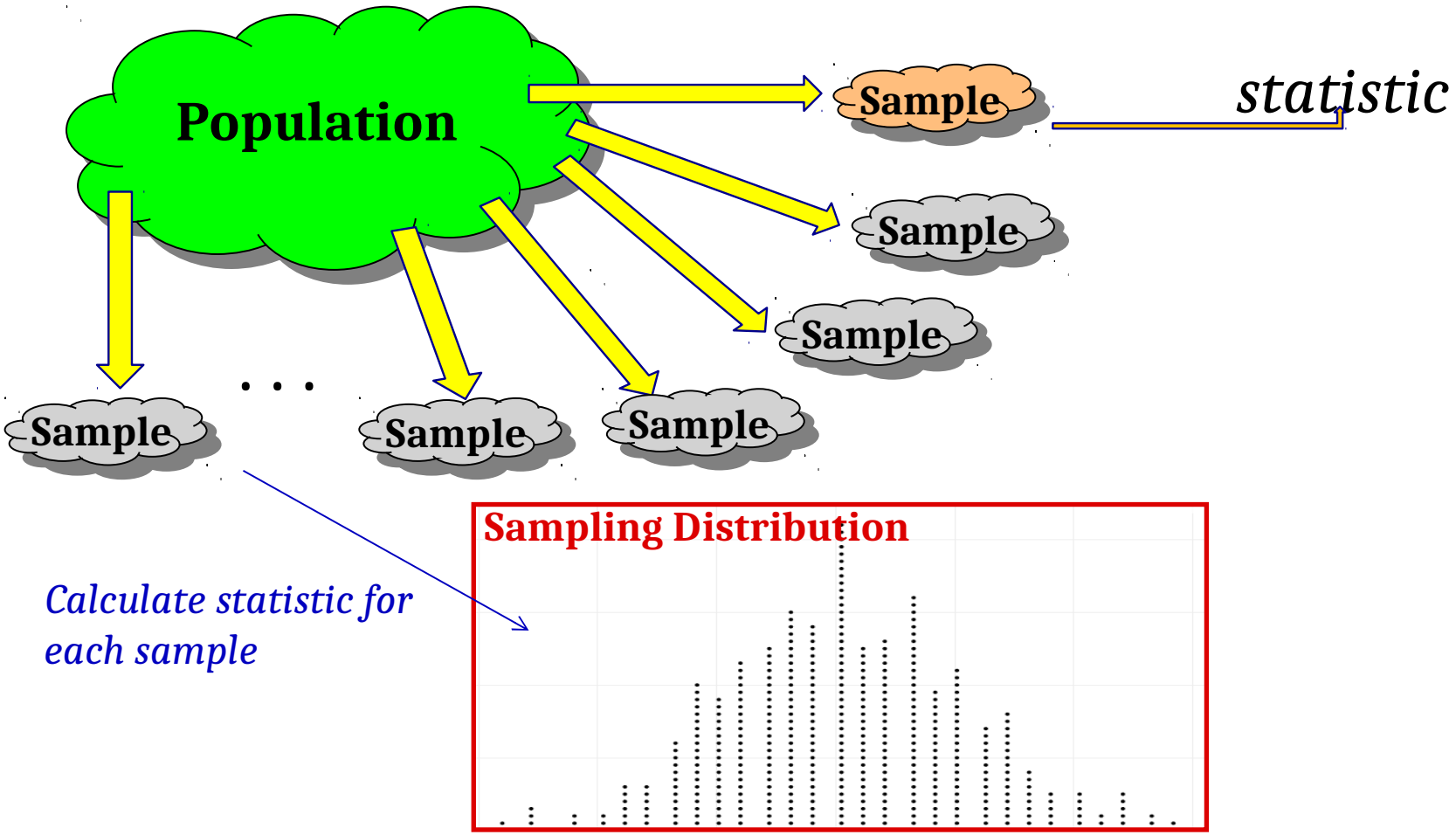

Population
Sample

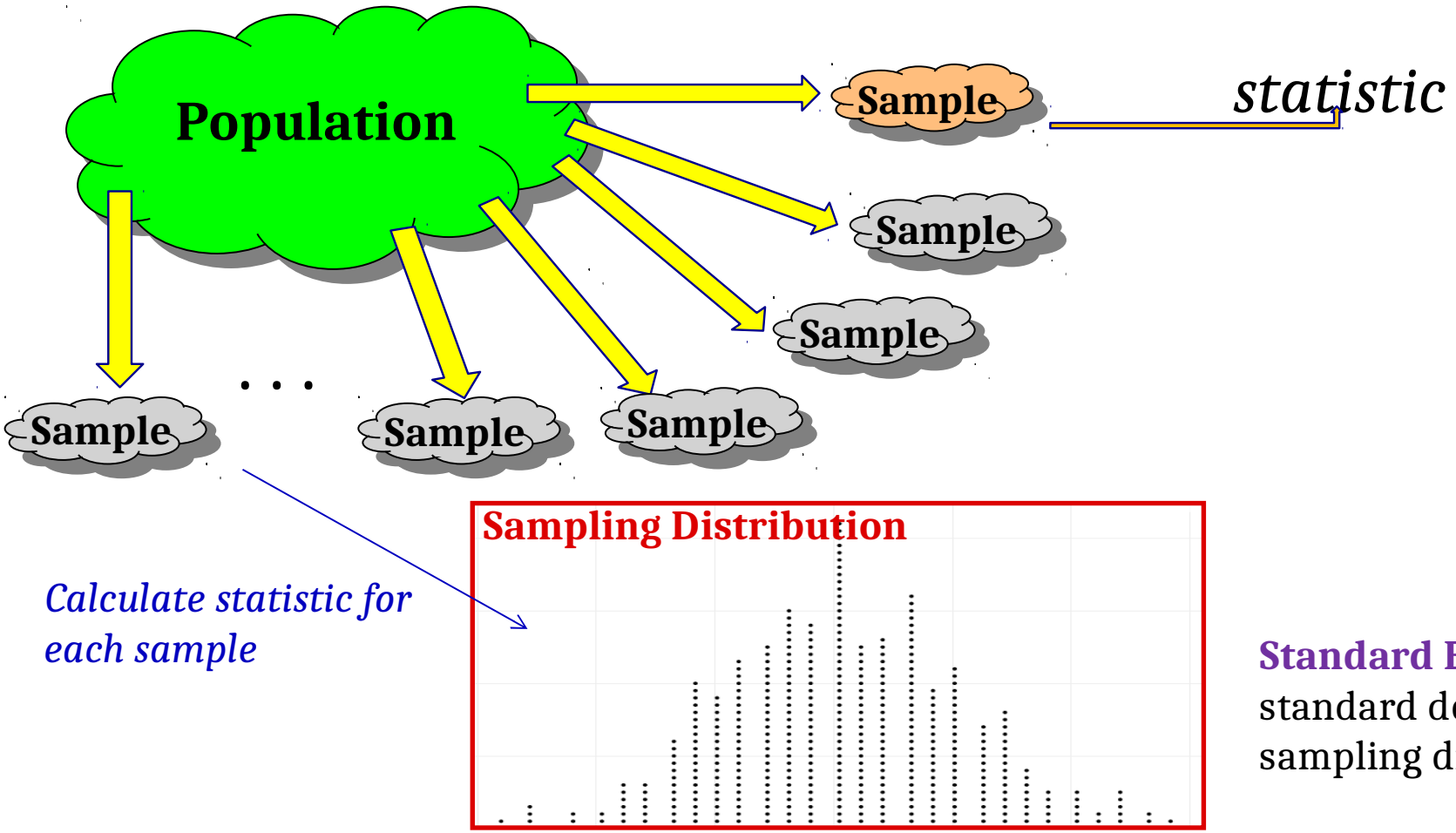

Population
Sample
statistic


Population
Sample
Sample
statistic

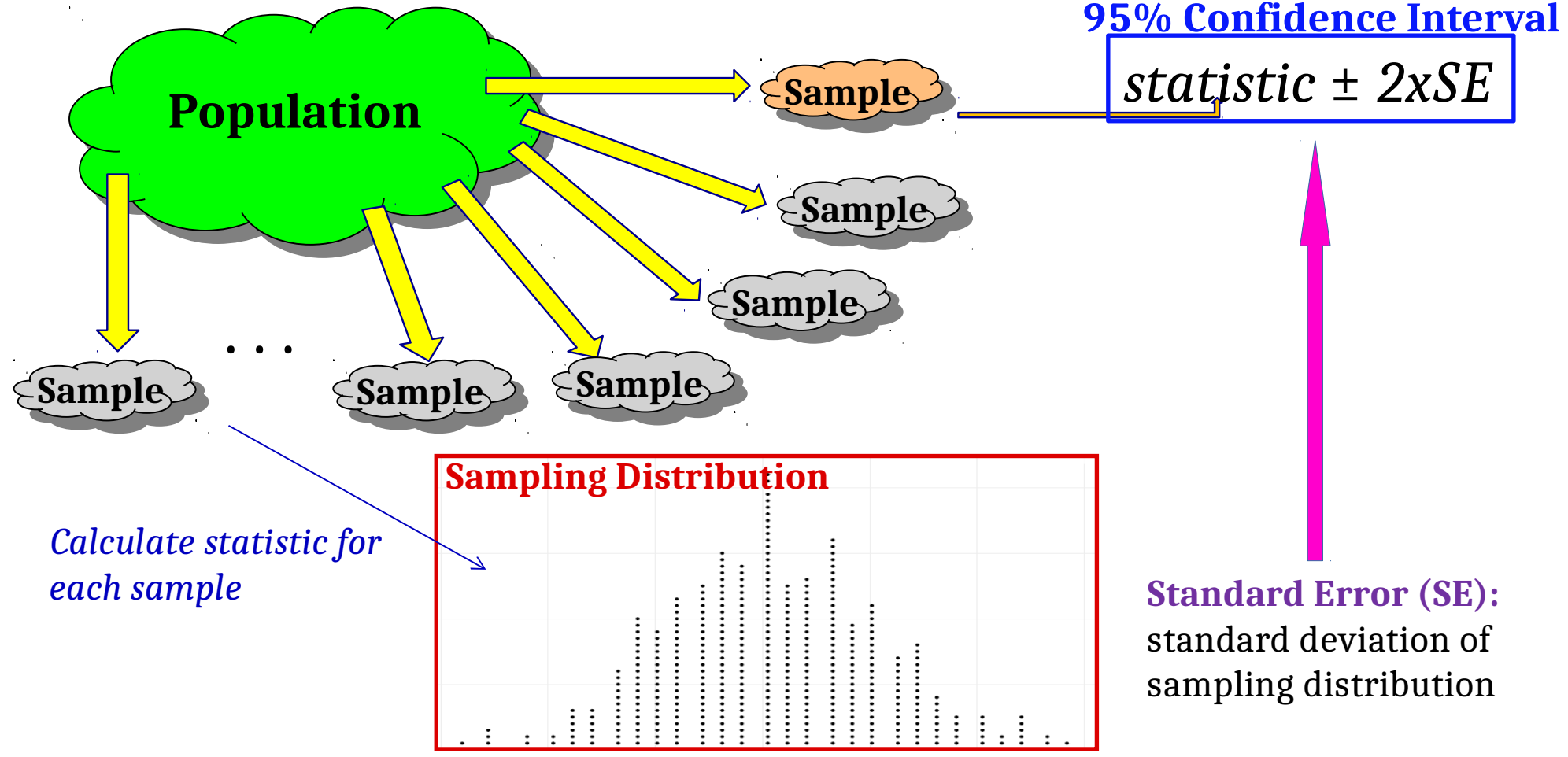

Population
Sample
Sample
Sample
statistic


Population
Sample
statistic
Sample
Sample
Sample
Sample
Sample
Sample


Population
Sample
statistic
Sample
Sample
Sample
Sample
Sample
Sample
Calculate statistic for each sample
Sampling Distribution



Standard Error (SE): standard deviation of sampling distribution


Population
Sample
Sample
Sample
Sample
Sample
Sample
Sampling Distribution
Calculate statistic for each sample
95% Confidence Interval
statistic ± 2xSE
Standard Error (SE):
standard deviation of sampling distribution

To create a plausible range of values for a parameter:

- Take many random samples from the population, and compute the sample statistic for each sample
- Compute the standard error as the standard deviation of all these statistics
- Use $\text{statistic} \pm 2 \cdot \text{SE}$

One small problem... WE ONLY HAVE ONE SAMPLE

The bootstrap

A key idea in modern statistics is the bootstrap.

- it can provide an estimate of the standard error of a coefficient, or a confidence interval for that coefficient.
- we can use the bootstrap to make tree **much** better predictors !!!!

The term bootstrap derives from the phrase to pull oneself up by one's bootstraps

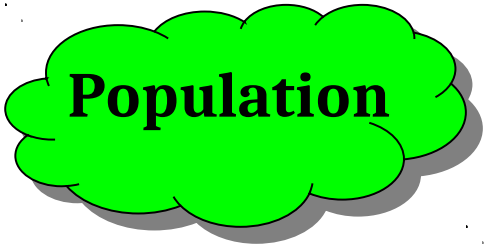
"The Baron had fallen to the bottom of a deep lake. Just when it looked like all was lost, he thought to pick himself up by his own bootstraps."

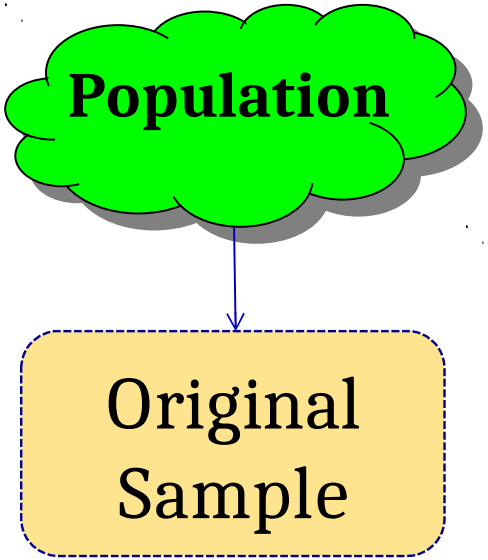
from *The Surprising Adventures of Baron Munchausen* by *Rudolph Erich Raspe*

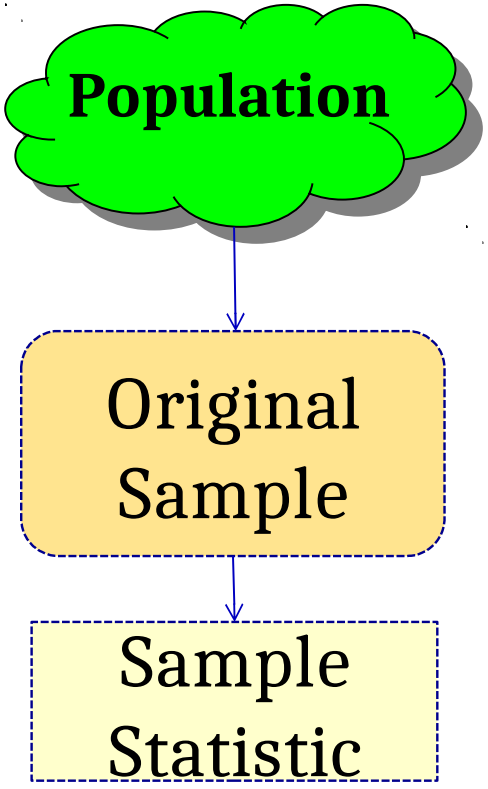
The bootstrap

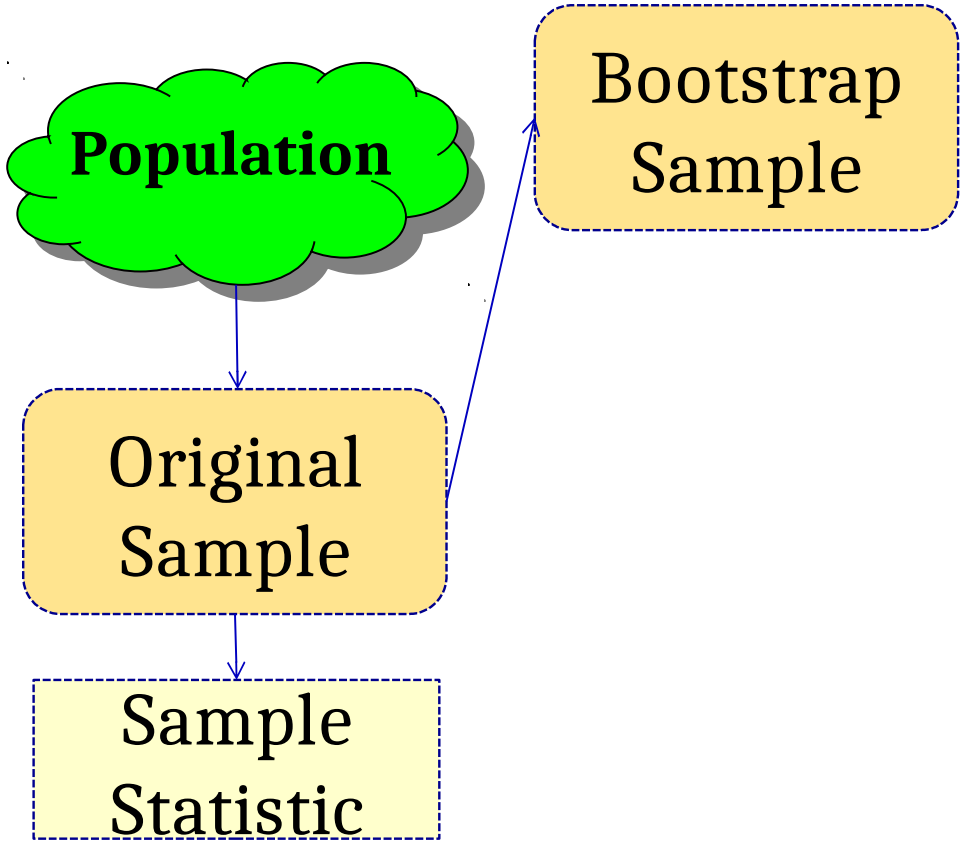
Suppose that we observe training data (x_i, y_i), $i = 1, \ldots, n$, which represents n independent draws from some unknown probability distribution $\mathcal{F}$.

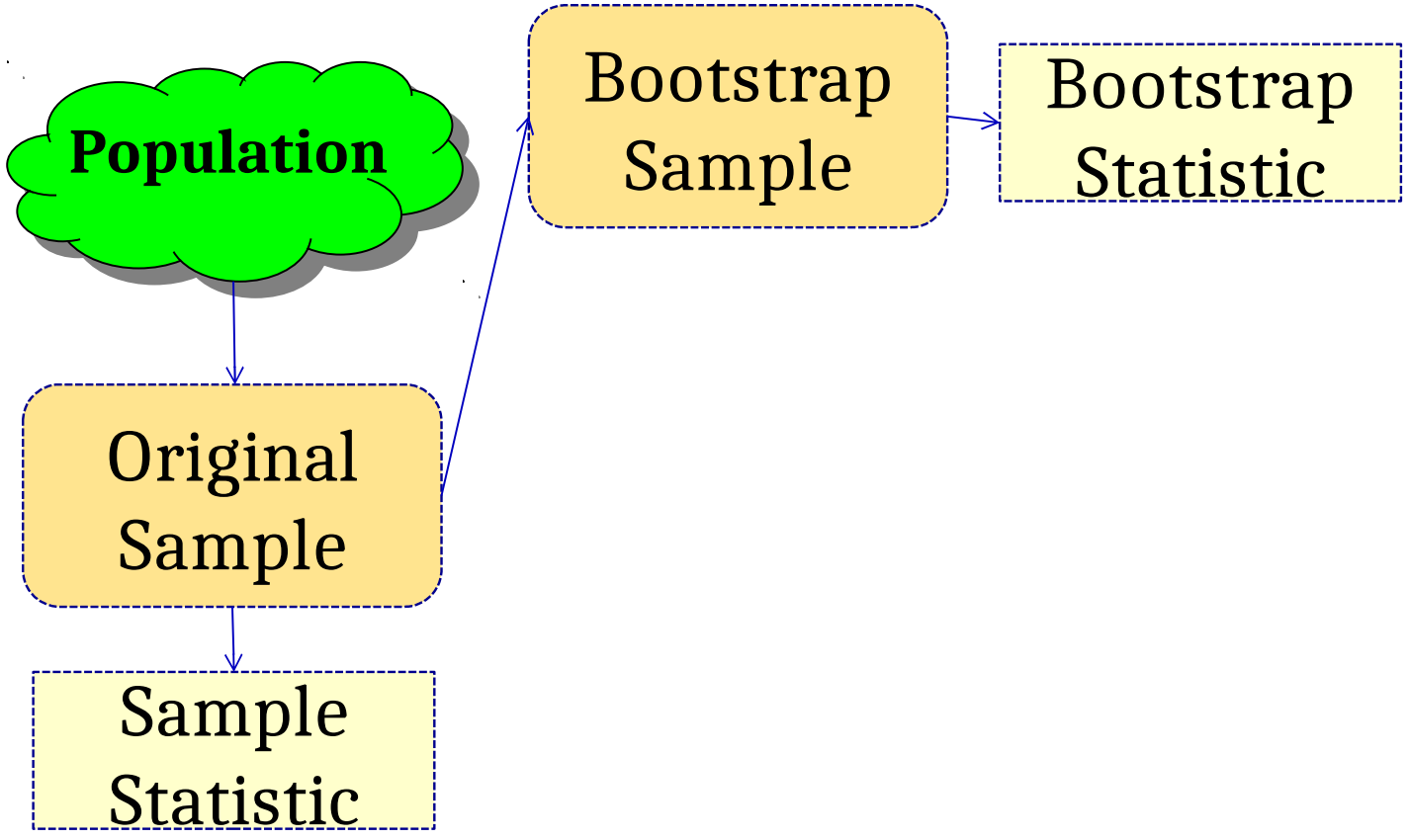
The bootstrap is a fundamental resampling tool in statistics. The basic idea underlying the bootstrap is that we can estimate the true $\mathcal{F}$ by the so-called empirical distribution $\hat{\mathcal{F}}$.

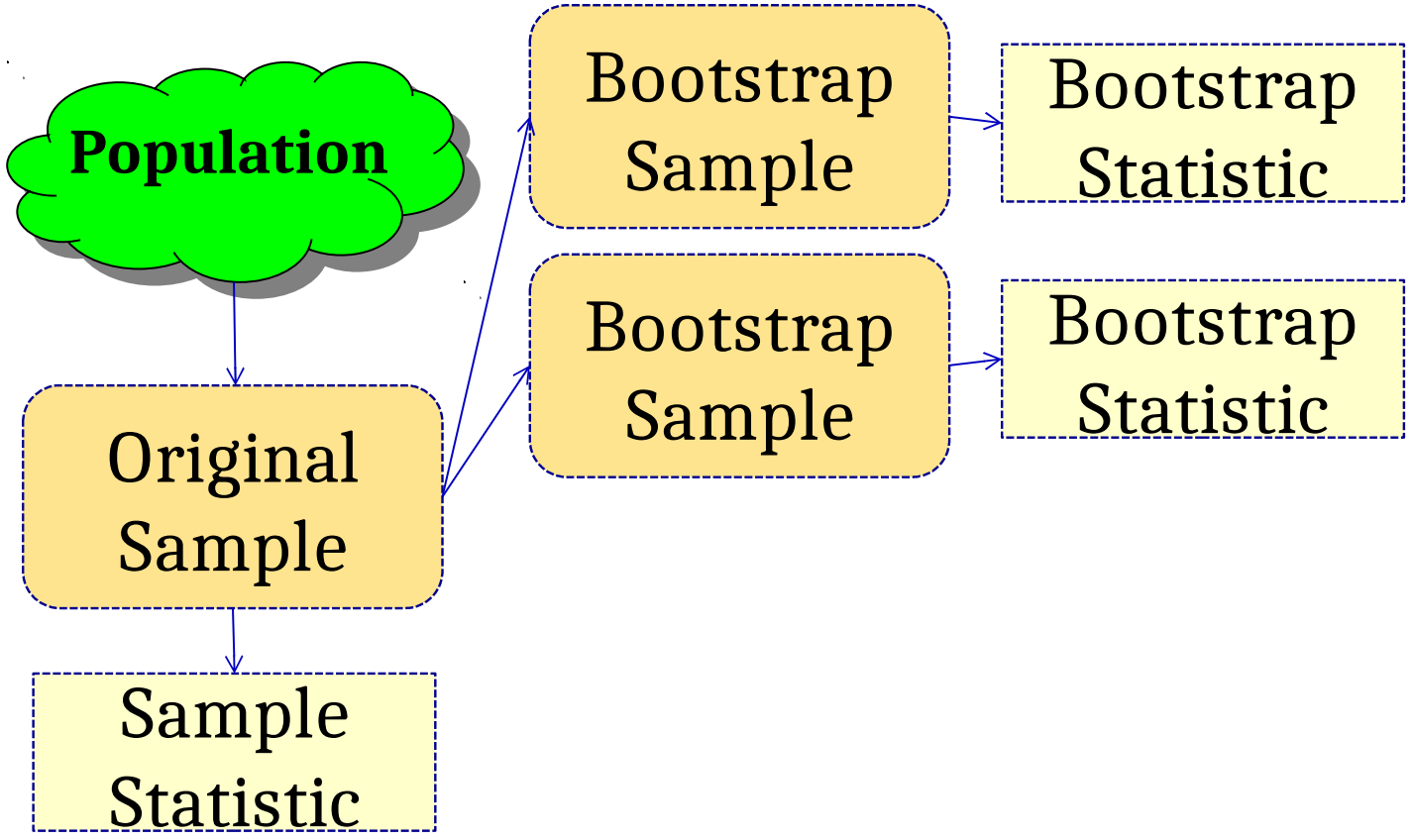

Population

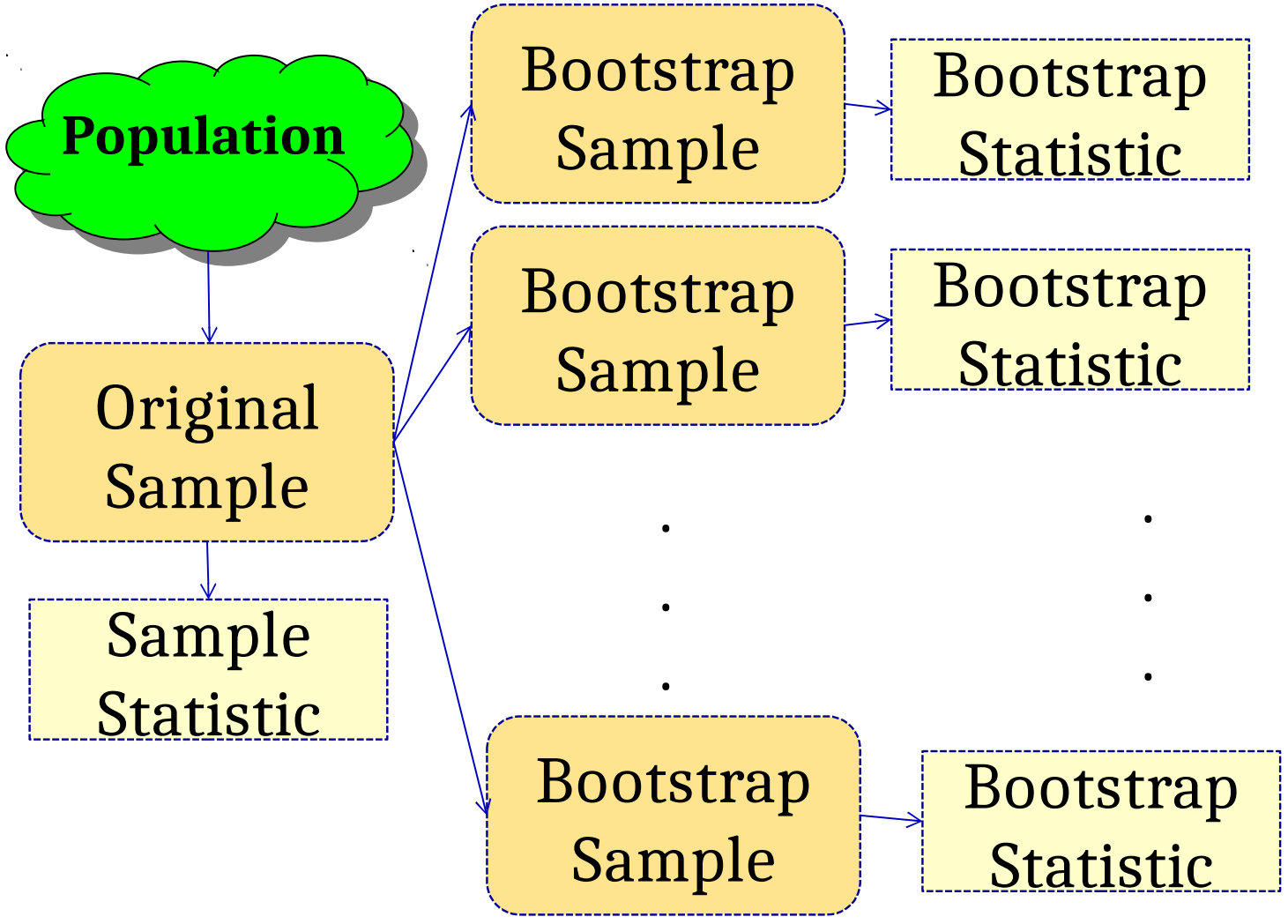

Population
Original Sample

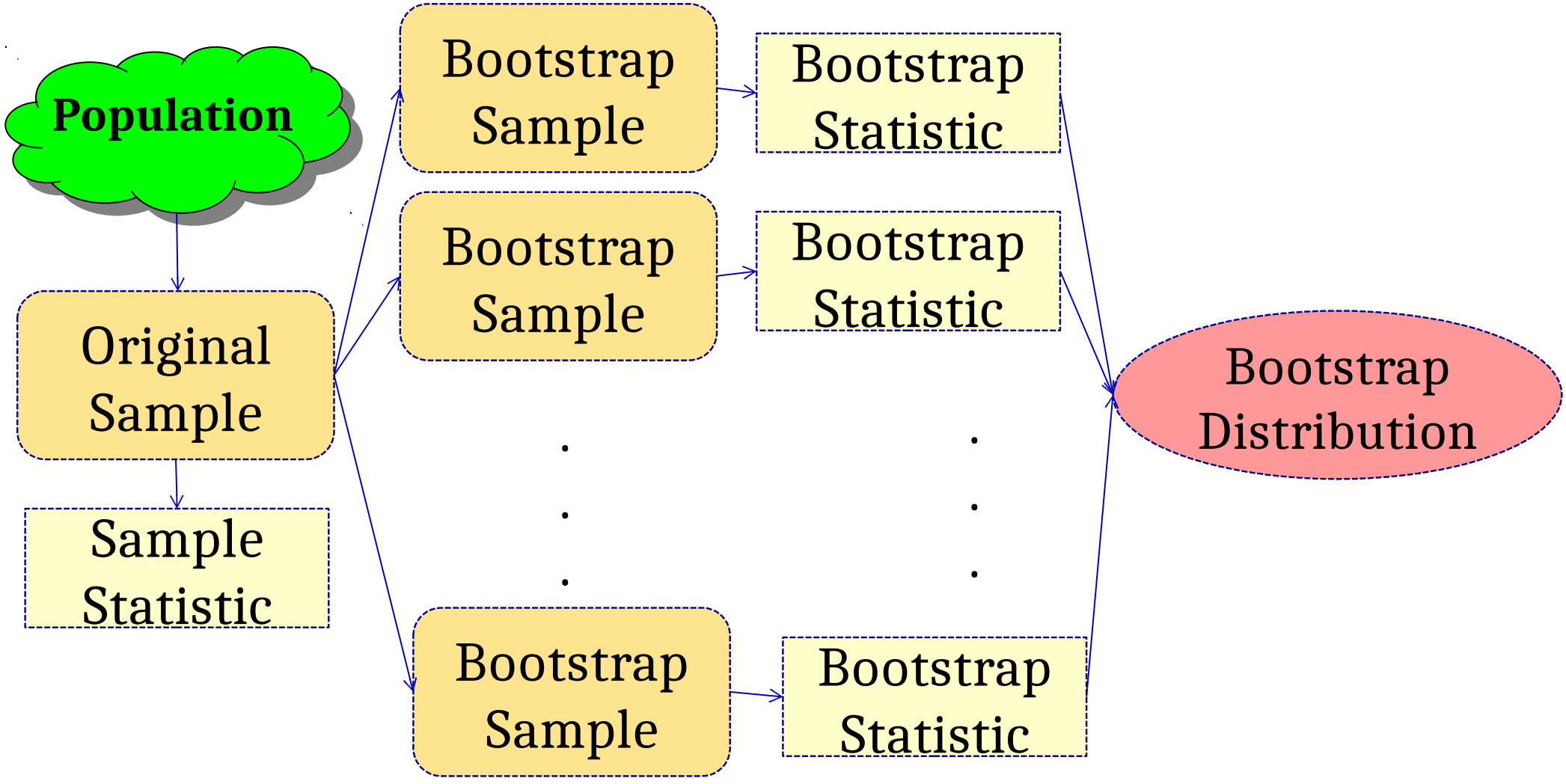

Population
Original Sample
Sample Statistic


Population
Bootstrap Sample
Original Sample
Sample Statistic


Population
Bootstrap Sample
Bootstrap Statistic
Original Sample
Sample Statistic


Population
Original Sample
Sample Statistic
Bootstrap Sample
Bootstrap Statistic
Bootstrap Sample
Bootstrap Statistic


Population
Original Sample
Sample Statistic
Bootstrap Sample
Bootstrap Statistic
Bootstrap Sample
Bootstrap Statistic
Bootstrap Sample
Bootstrap Statistic


Population
Original Sample
Sample Statistic
Bootstrap Sample
Bootstrap Sample
Bootstrap Sample
Bootstrap Statistic
Bootstrap Statistic
Bootstrap Statistic
Bootstrap Distribution

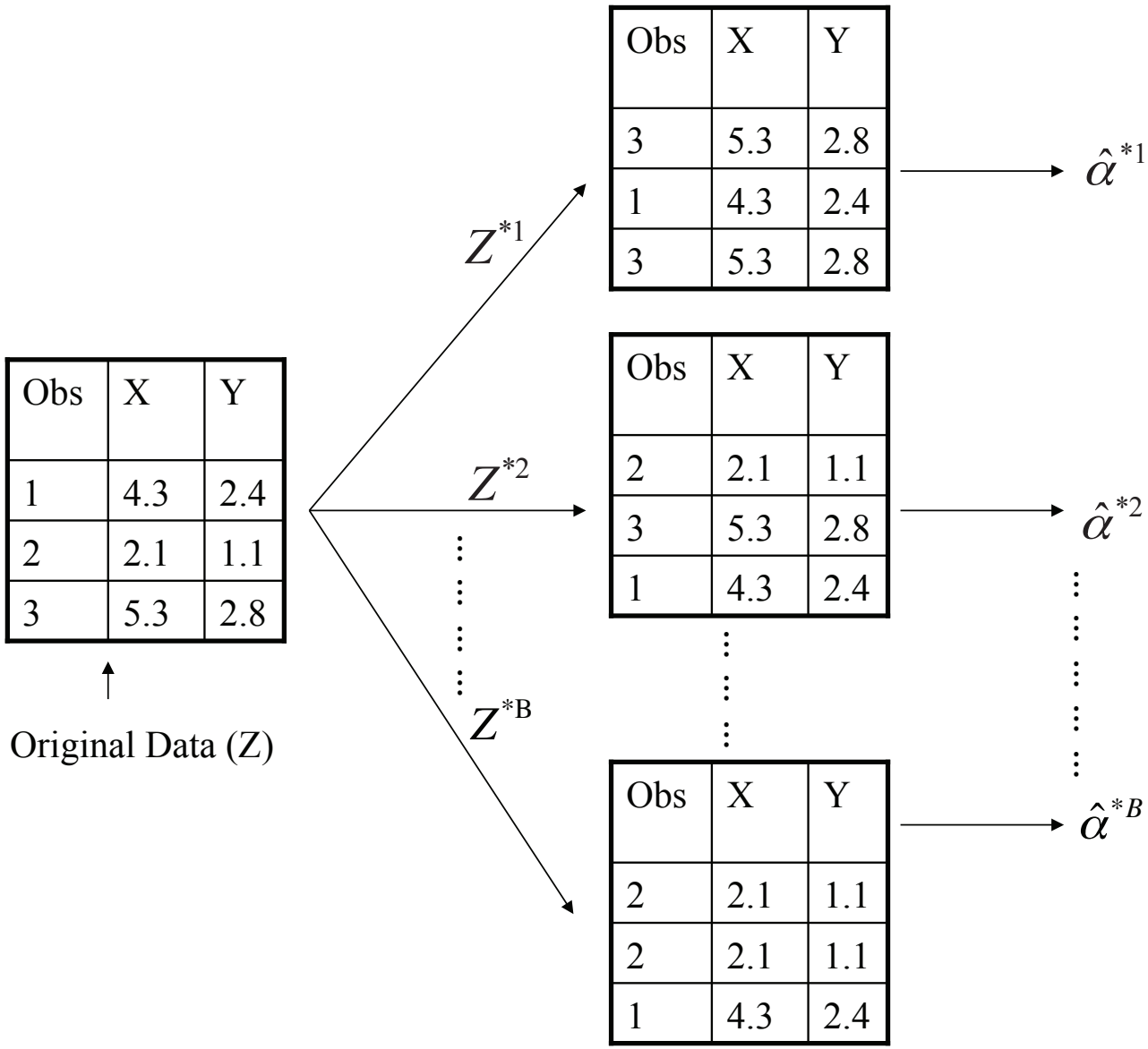
The bootstrap

Treat the sample as if it were the population and then take iid draws.

A bootstrap sample of size m from the training data is (x_i^*, y_i^*), $i = 1, \ldots, m$, where each (x_i^*, y_i^*) are drawn from uniformly at random from $(x_1, y_1), \ldots, (x_n, y_n)$, with replacement.

This corresponds exactly to m independent draws from $\hat{\mathcal{F}}$. Hence it approximates what we would see if we could sample more data from the true $\mathcal{F}$. We often consider $m = n$, which is like sampling an entirely new training set.

Note: not all of the training points are represented in a bootstrap sample, and some are represented more than once. When $m = n$, about 36.8% of points are left out, for large n.

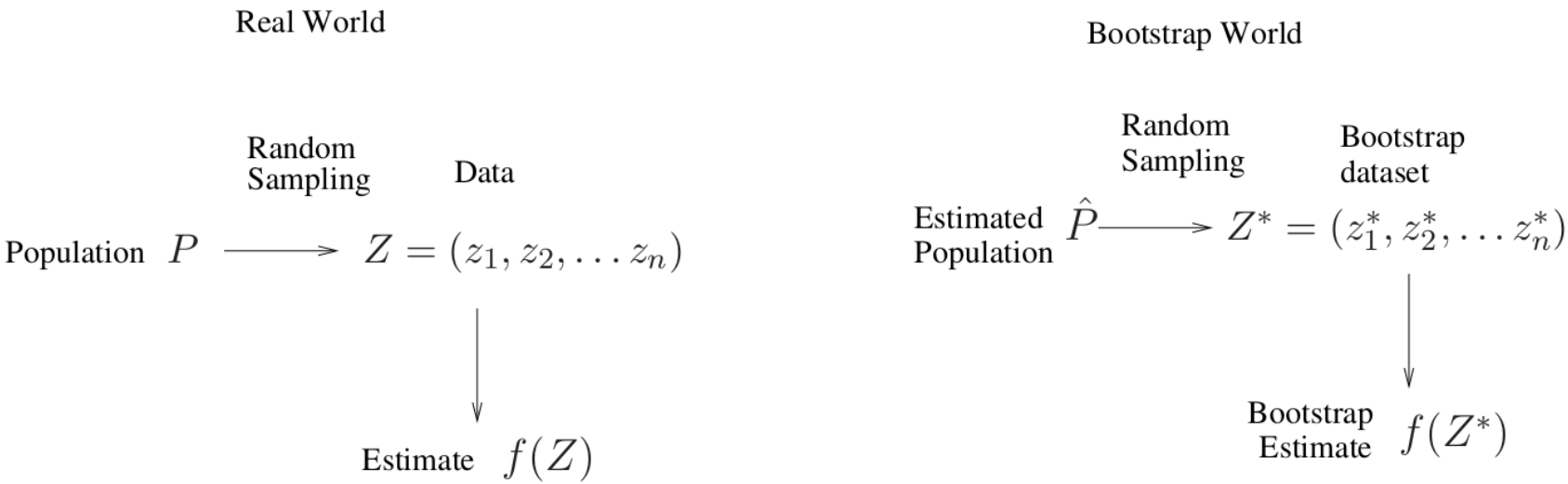

Obs X Y
1 4.3 2.4
2 2.1 1.1
3 5.3 2.8
Original Data (Z)
Z^{*1}
Z^{*2}
Z^{*B}
Obs X Y
3 5.3 2.8
1 4.3 2.4
3 5.3 2.8
Obs X Y
2 2.1 1.1
3 5.3 2.8
1 4.3 2.4
Obs X Y
2 2.1 1.1
2 2.1 1.1
1 4.3 2.4
$\hat{\alpha}^{*1}$
$\hat{\alpha}^{*2}$
$\hat{\alpha}^{*B}$

R Code

Let us see an example of bootstrap in action.

- confidence intervals
- standard deviation of the estimator

Bootstrap Summary



Bagging and Random Forest

Classification trees are popular since they are highly **interpretable**. They are also model-free (don't assume an underlying distribution for the data).

But they don't generally give a **prediction accuracy** competitive with that of other classifiers (logistic regression, k-nearest neighbors, etc.) The reason: trees have somewhat inherently **high variance**. The separations made by splits are enforced at all lower levels of the tree, which means that if the data is perturbed slightly, the new tree can have a considerably different sequence of splits, leading to a different classification rule.

Now we'll learn of two ways to control the variance, or **stabilize** the predictions made by trees. Of course these solutions aren't perfect: in doing so, we can greatly improve prediction accuracy but we suffer in terms of interpretability.

Bagging

Bootstrap aggregation (**bagging**) is a general-purpose procedure for reducing the variance of a machine learning method

- we will use it in the context of classification and regreession trees;
- but it is useful for other types of algorithms as well.

Suppose that you have a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, then $\mathrm{Var}(\bar{Z}) = \frac{\sigma^2}{n}$.

Averaging observations reduces variance

- unfortunately, we only have one sample.

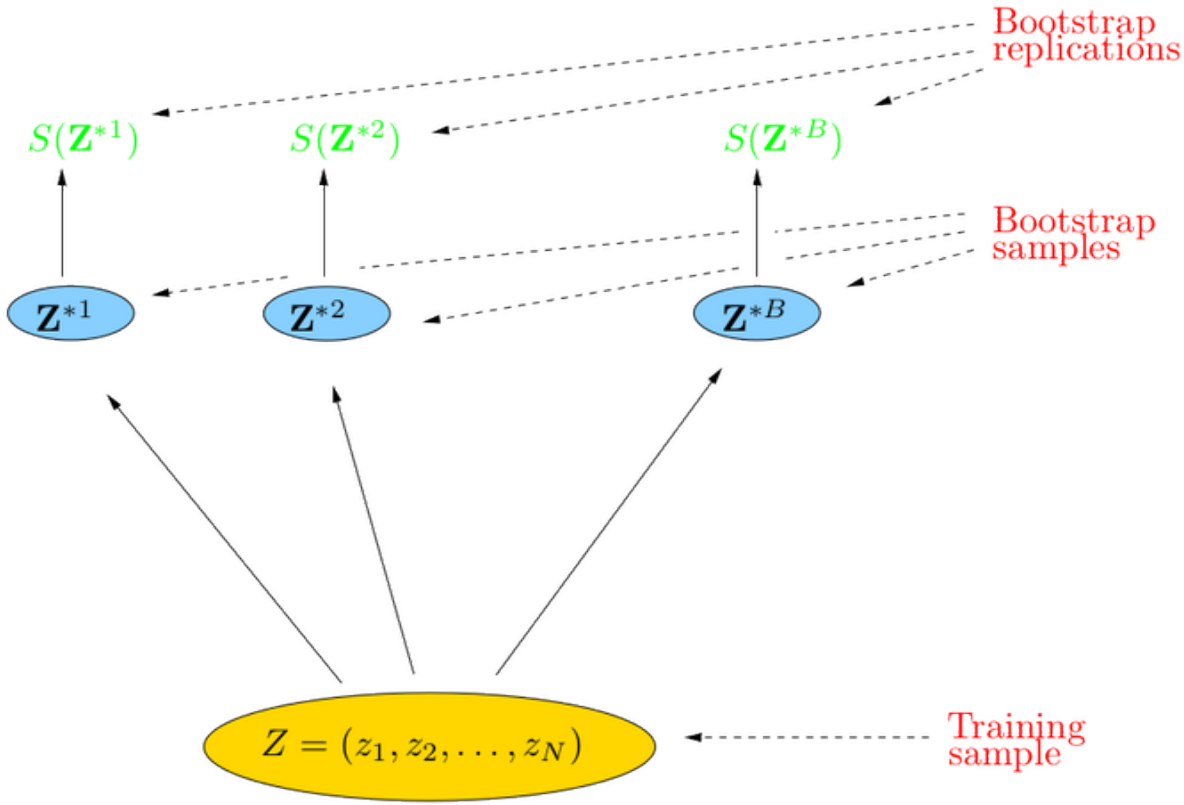
Bagging

Use the bootstrap to learn a collection of trees.

- Repeatedly re-sample the data, with replacement, to get a *jittered* data set of n observations.
- For each bootstrap re-sample, fit a CART tree. Each tree will be slightly different due to the randomness of the sampling.

When you want to predict $Y(x)$ for some new x:

- **regression**: take the average prediction from this collection of trees
- **classification**:
  1. majority vote across all trees; or
  2. average estimated probabilites

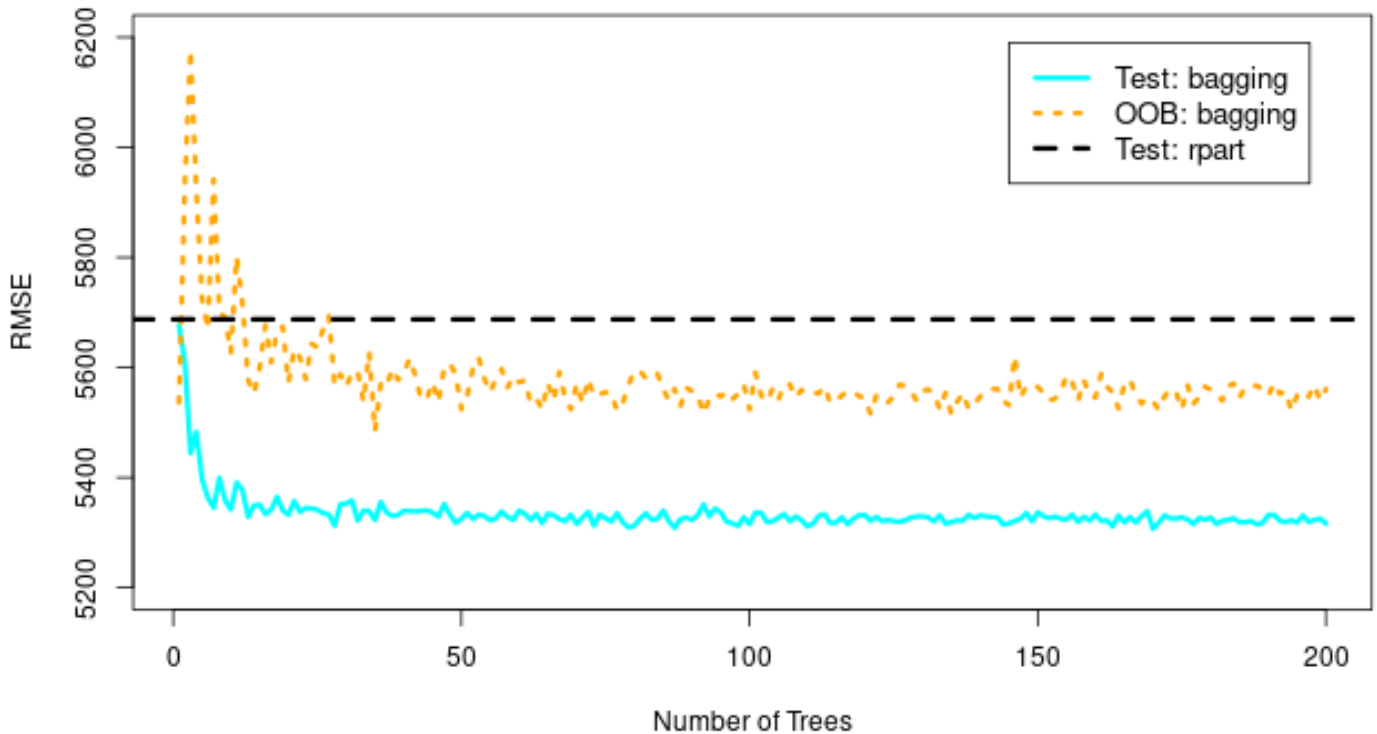

Bootstrap replications
$S(\mathbf{Z}^{*1})$
$S(\mathbf{Z}^{*2})$
$S(\mathbf{Z}^{*B})$
Bootstrap samples
$\mathbf{Z}^{*1}$
$\mathbf{Z}^{*2}$
$\mathbf{Z}^{*B}$
$Z = (z_1, z_2, \ldots, z_N)$
Training sample

Example: Breiman's bagging

Example from the original Breiman paper on bagging: comparing the misclassification error of the CART tree (pruning performed by cross-validation) and of the bagging classifier (with $B = 50$):

Data Set	$\bar{e}_S$	$\bar{e}_B$	**Decrease**
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

Example: Used Cars



Out-of-Bag Error Estimation

Straightforward way to estimate the test error of a bagged model.

- trees are repeatedly fit to bootstrapped subsets of the observations
- on average, each bagged tree makes use of around $2/3$ of the observations

Roughly one-third of the observations are not used to fit a given bagged tree are

- **out-of-bag** (OOB) observations.

Predict the response for the ith observation using trees in which that observation was OOB.

- we get around $B/3$ predictions for the ith observation, which we average.

This estimate is essentially the LOO cross-validation error for bagging, if B is large.

How should we choose the number of trees B?

You need B big enough to get the averaging to work, but it does not seem to hurt if you make B bigger than that.

The cost of having very large B is in computational time.

- We can build trees fast, but if you start building thousands of really big trees on large data sets, it can end taking a while.

Why is bagging working?

The justification for bagging is based on a bias-variance argument.

Recall how CART is used in practice.

- Split to lower deviance until leaves hit a minimum size.
- Create a set of candidate trees by pruning back from this deep fit.
- Choose the best among those trees by cross validation.

Trees are low bias, high variance models.

- Can change a lot from sample-to-sample—high variance.
- Especially true for deep trees.

Averaging trees is a way to reduce variance!

Why is bagging working?

The thinking is that

- real structure that persists across bootstrapped data sets will "show up" in the average;
- noisy/useless signals will average out with no effect.

Random forests

Random Forests is very similar to bagging with **one key innovation**.

- build a number of trees on bootstrapped samples;
- however, when attempting to create a split, a random selection of m predictors are considered as split candidates;
- the split is allowed to use only one of those m predictors, rather than choosing among the full set of p predictors.

Dramatic improvement in predictive performance. Why?

Imagine there is one strong predictor and a number of moderate predictors.

- Then each bagged tree will always split on the strong predictor first.
- This results in the trees being highly correlated.
- Random predictor selection allows for decorrelation of the trees, lower variance, and better predictive performance.

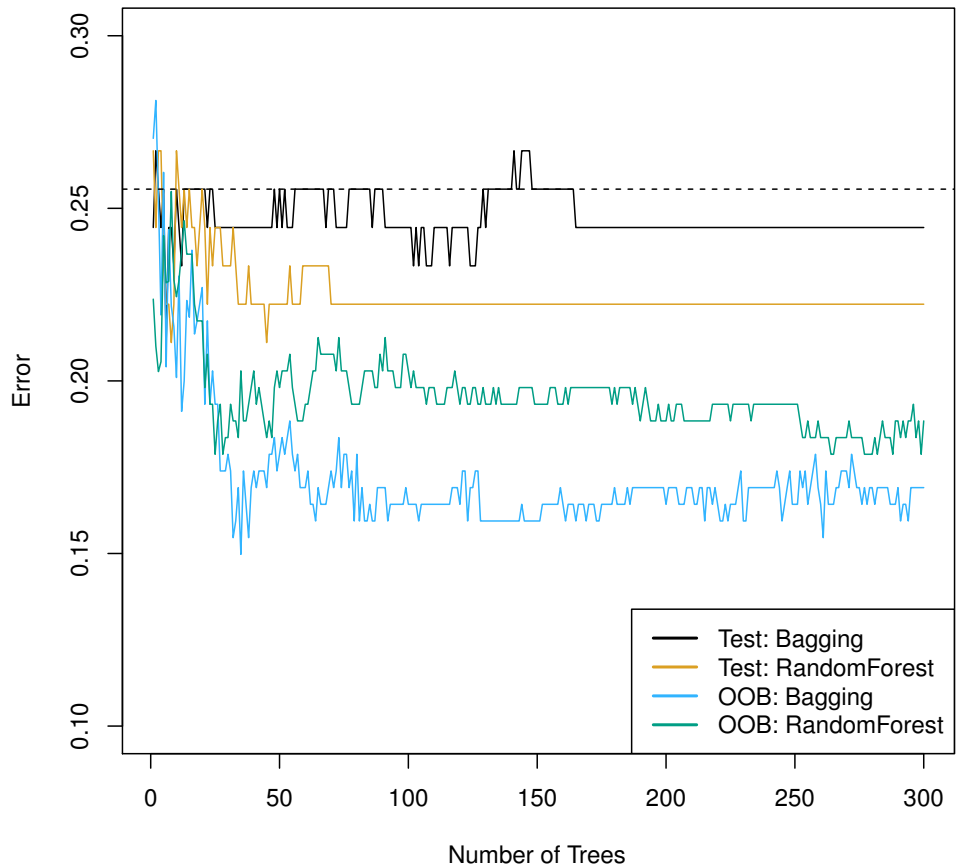
Prediction depends on the leaf model.

Tuning parameters?

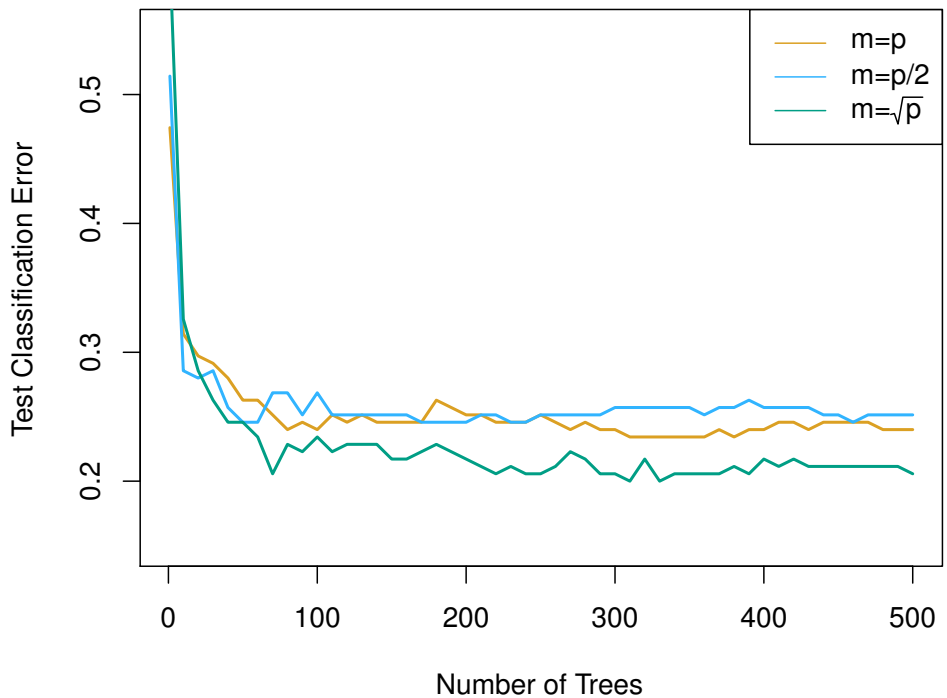
Have to choose:

- B: number of Bootstrap samples (hundreds, thousands)
- m: number of variables to try at each split
  - regression: $m = p/3$
  - classification: $m = \sqrt{p}$
- n_{min}: minimum number of observations required in a node—roughly how deep should trees be grown
  - regression: 5
  - classification: 1
  - very deep trees are desired for low bias

Example: The Heart Data



Example: Gene expression data



Disadvantages

It is important to discuss some disadvantages of bagging:

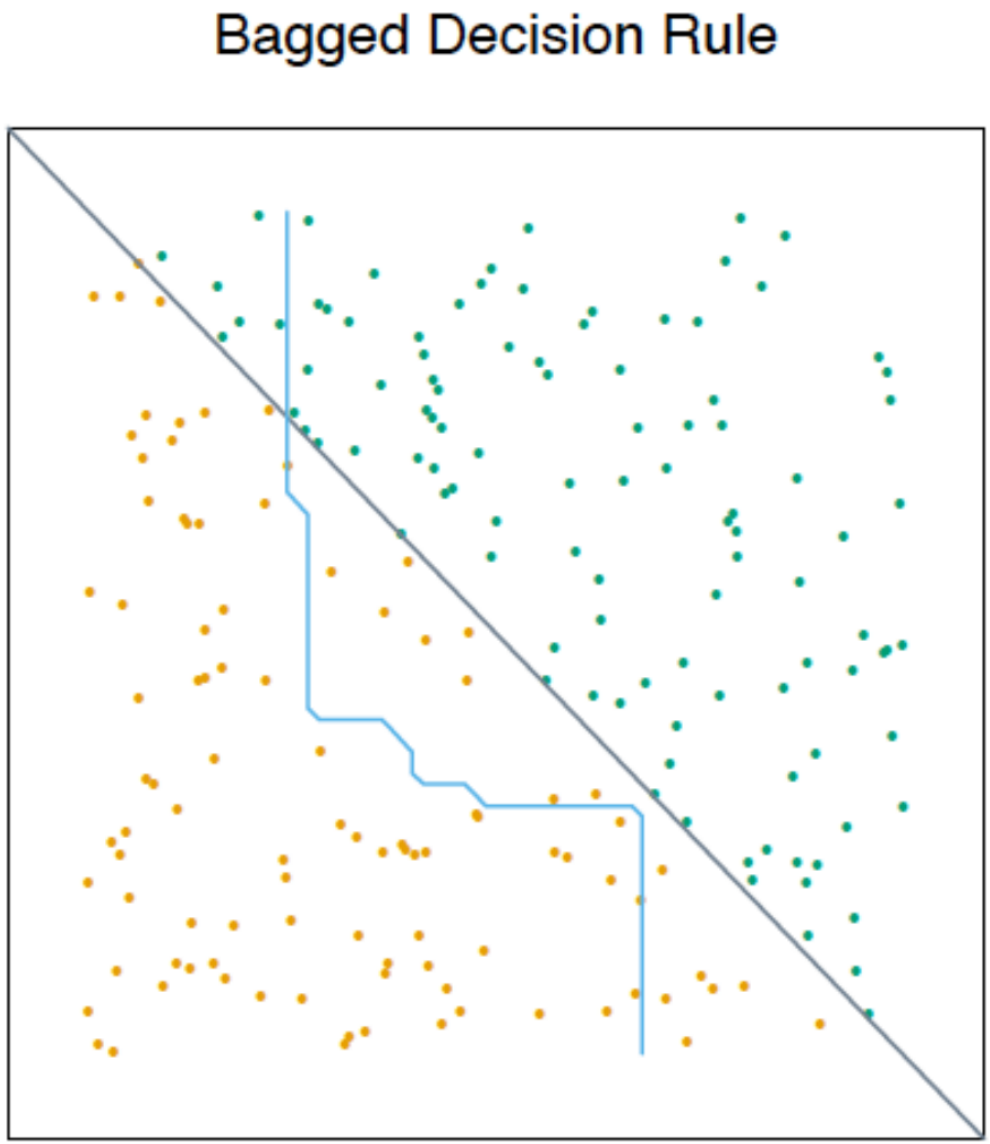
- Loss of interpretability: the final bagged classifier is not a tree, and so we forfeit the clear interpretative ability of a classification tree
- Computational complexity: we are essentially multiplying the work of growing a single tree by B (especially if we are using the more involved implementation that prunes and validates on the original training data)

You can think of bagging as extending the space of models. We go from fitting a single tree to a large group of trees. Note that the final prediction rule cannot always be represented by a single tree.

Sometimes, this enlargement of the model space isn't enough, and we would benefit from an even greater enlargement.

Example: limited model space

Bagging cannot really represent a diagonal decision rule

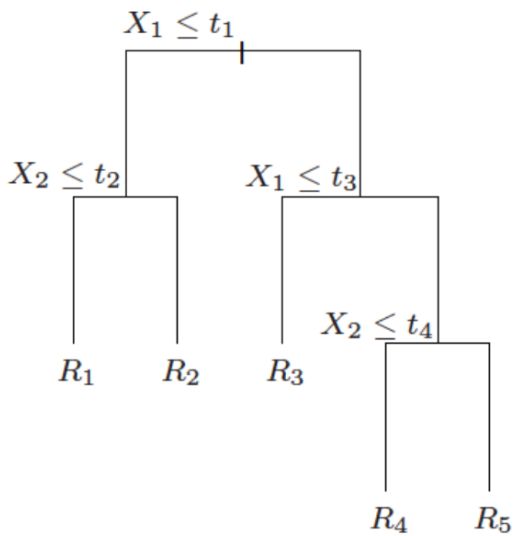


Variable importance for trees

For a single decision tree $\hat{f}^{\text{tree}}$, we define the squared importance for variable j as

$$\text{Imp}_j^2(\hat{f}^{\text{tree}}) = \sum_{k=1}^{m} \hat{d}_k \cdot 1\{\text{split at node } k \text{ is on variable } j\}$$

where m is the number of internal nodes (non-leaves), and $\hat{d}_k$ is the improvement in training misclassification error from making the kth split



Variable importance for random forests

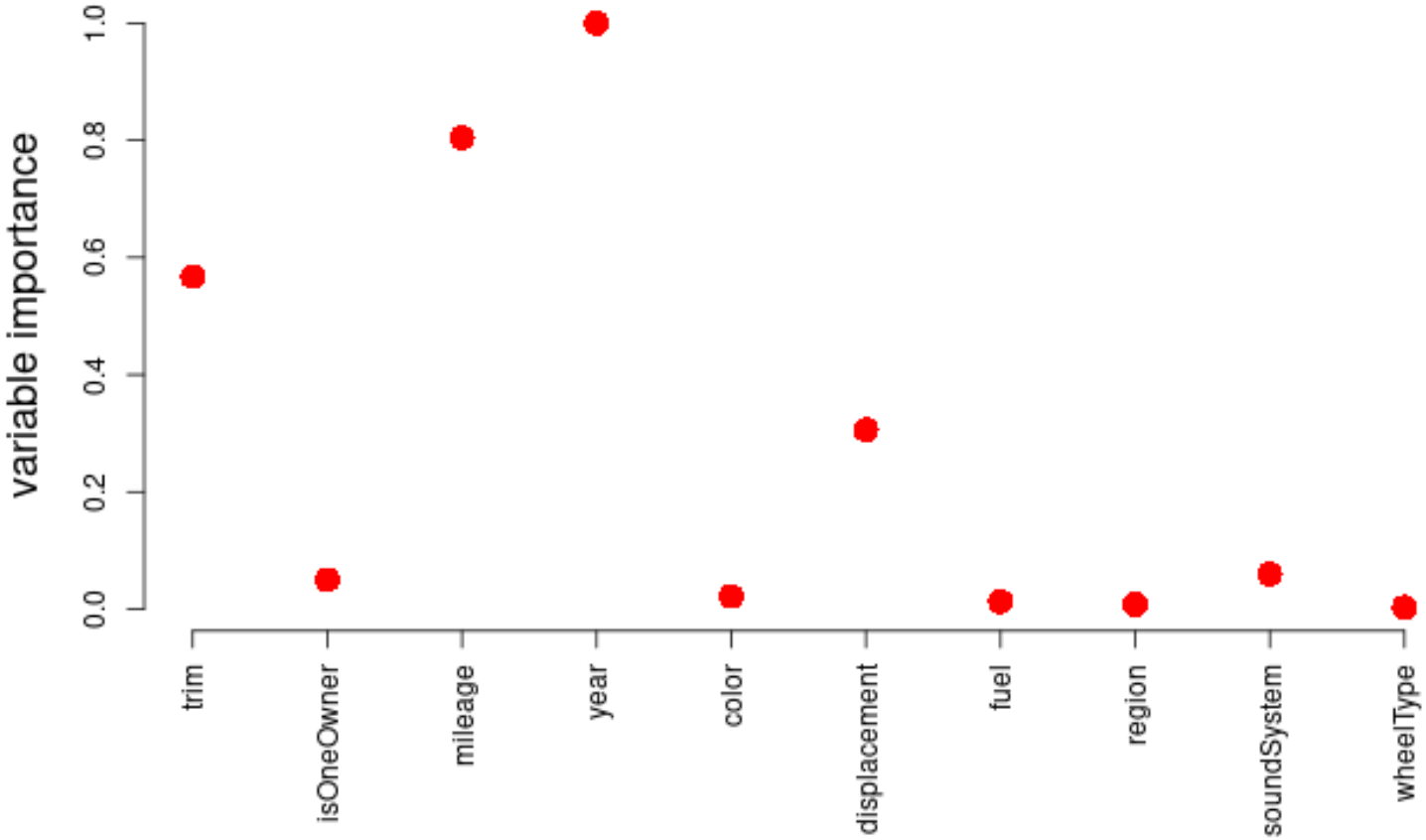
We define the squared importance for a variable j by simplying averaging the squared importance over all of the fitted trees:

$$\text{Imp}_j^2(\hat{f}^{\text{rf}}) = \frac{1}{B}\sum_{b=1}^{B} \text{Imp}_j^2(\hat{f}^{\text{tree},b})$$

(To get the importance, we just take the squared root.) We also usually set the largest importance to 100, and scale all of the other variable importances accordingly, which are then called relative importances.

This averaging stabilizes the variable importances, which means that they tend to be much more accurate for an ensemble of trees than they do for any single tree.

Example: Used Cars



Recap: Bagging/Random Forests

Random forests are a technique in which we draw many bootstrap-sampled data sets from the original training data, train on each sampled data set individually, and then aggregate predictions at the end. We applied bagging to classification trees.

There were two strategies for aggregate the predictions: taking the class with the majority vote (the **consensus** strategy), and averaging the estimated class probabilities and then voting (the **probability** strategy). The former does not give good estimated class probabilities; the latter does and can sometimes even improve prediction accuracy.

The method works if the base classifier is **not bad** (in terms of its prediction error) to begin with. Combining bad classifiers can degrade their performance even further. Combining still can't represent some basic decision rules.

Boosting

Boosting

Boosting is similar to bagging in that we combine the results of several classification trees.

However, boosting does something fundamentally different, and can work a lot better.

Boosting builds trees **sequentially**

- each tree is grown using information from previously grown trees.

Boosting

Boosting is similar to bagging in that we combine the results of several classification trees.

However, boosting does something fundamentally different, and can work a lot better.

Boosting builds trees **sequentially**

- each tree is grown using information from previously grown trees.

Boosting for regression

Algorithm:

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

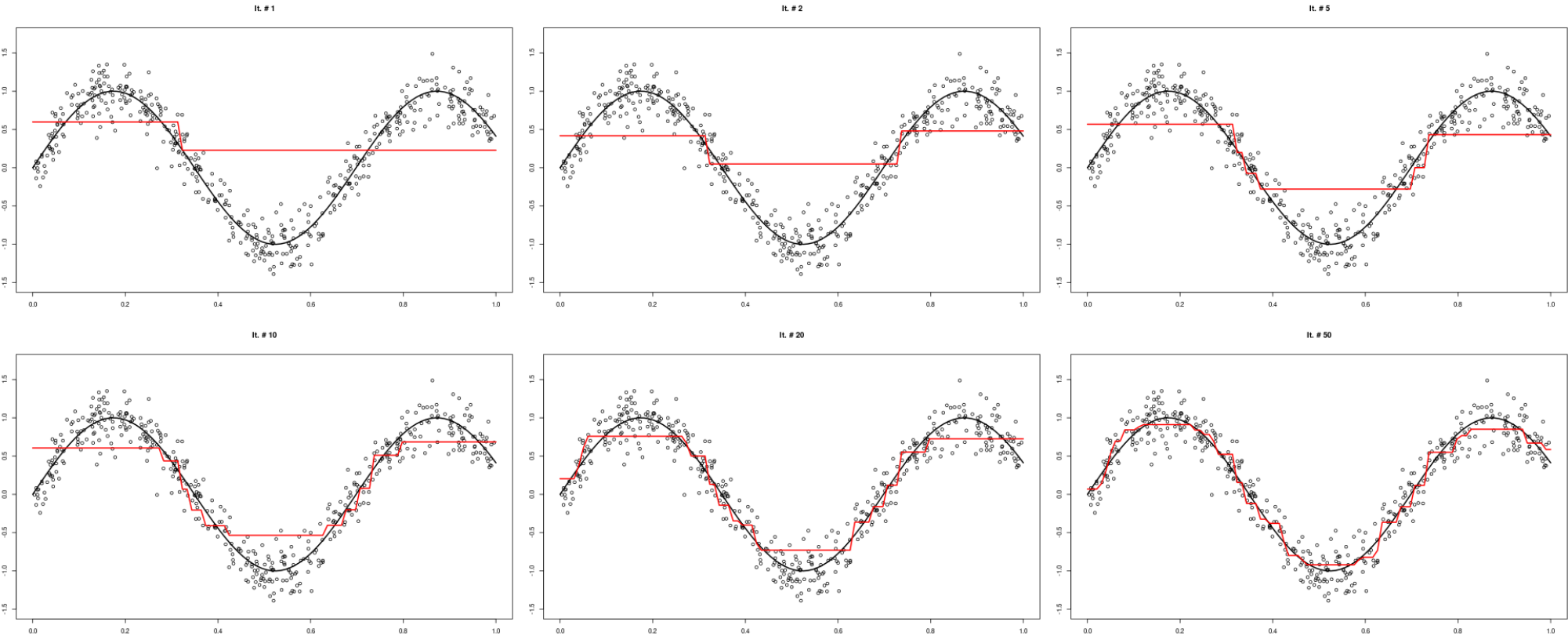
   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Example



What is the idea?

Given the current model, we fit a tree to the residuals, i.e., where the current model does not perform well

- We add the new tree into the model to update the residuals

Typically, each tree is small

- Prevents overfitting, more cautious
- Governed by the number of terminal nodes

Learns slowly

- By fitting small trees to the residuals, we improve the model where it does not perform well
- The shrinkage parameter λ slows the process down even further
- We are allowing more and different shaped trees to capture the residuals

Tuning parameters for boosting

B, number of trees

- Boosting can overfit if B is too large

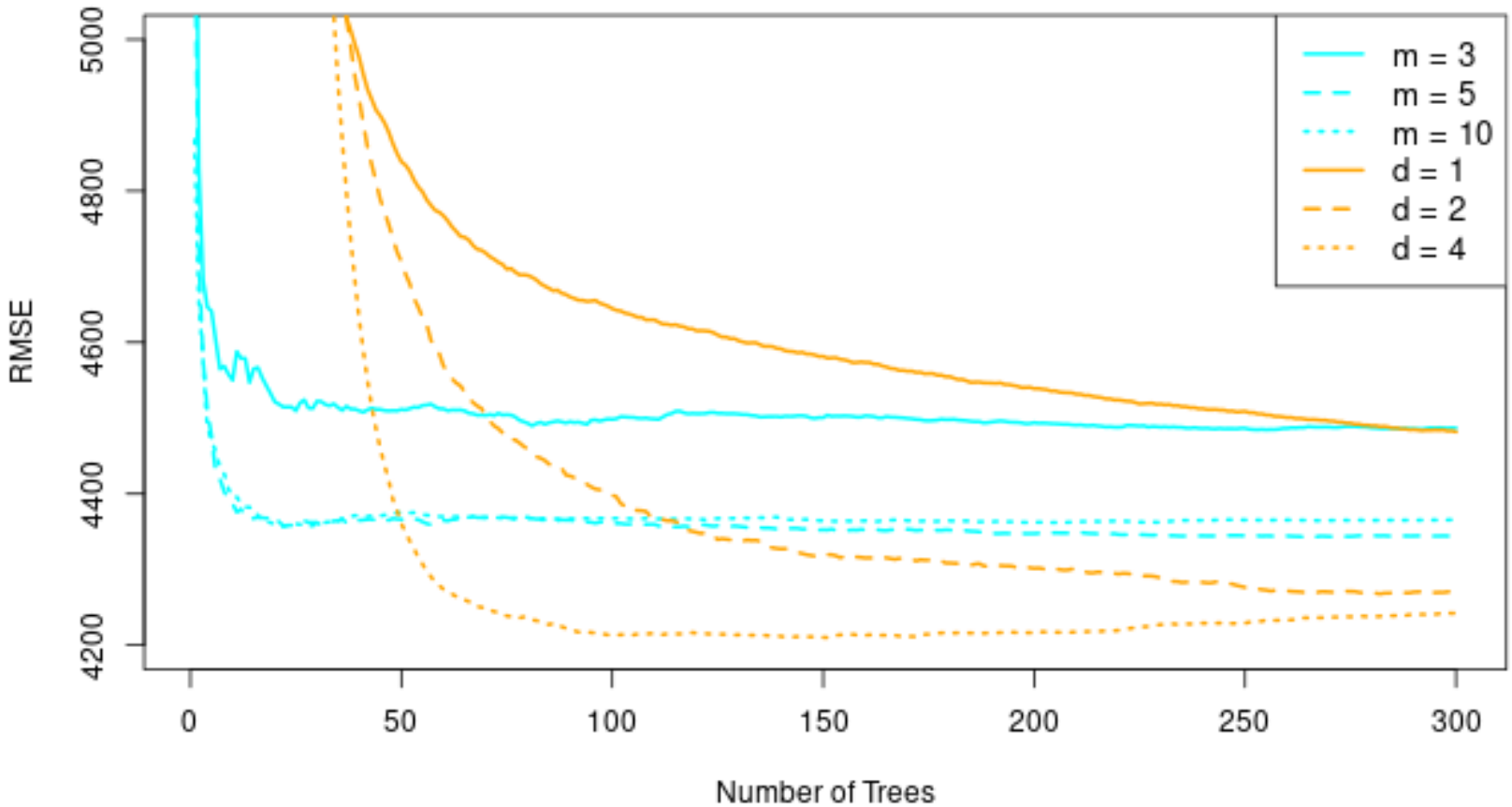
λ, the "shrinkage" parameter

- controls the rate at which boosting learns
- typical values are between 0.01 and 0.001 (depends on the problem)
- a very small λ can require using a very large value of B in order to achieve good performance

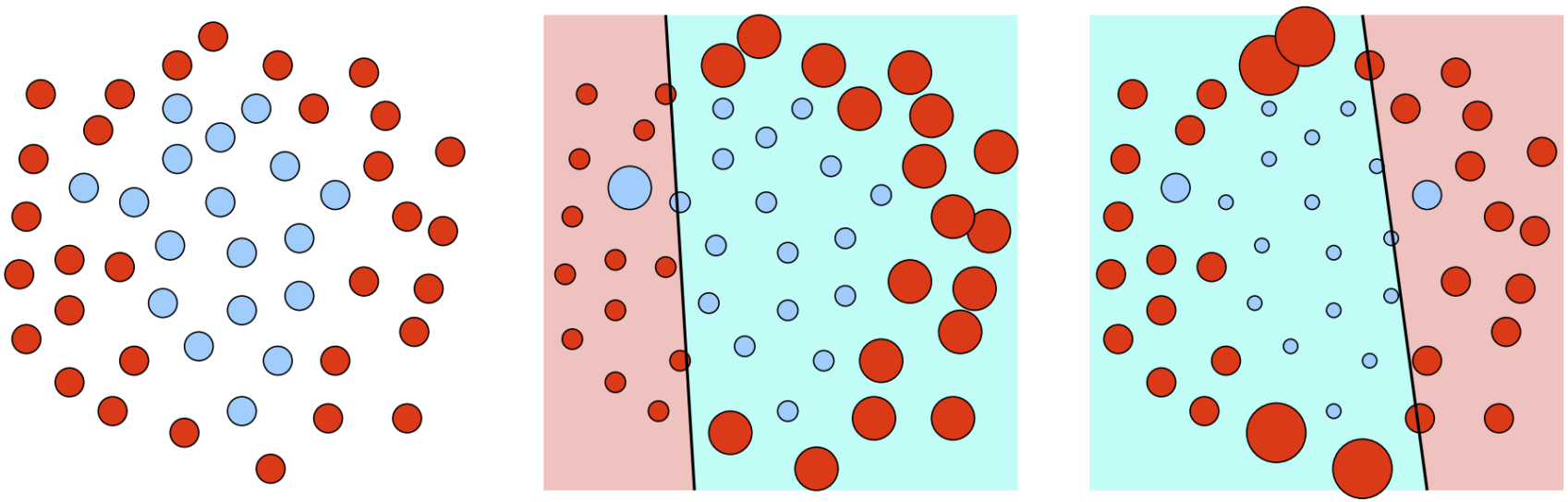
d, the number of splits in each tree

- $d = 1$, results in a decision stump—tree with a single split; gives us an additive model
- more generally d is the interaction depth
- controls the interaction order of the boosted model, since d splits can involve at most d variables.
- typically d is between 1 and 10

Example: Used Cars



Boosting for classification



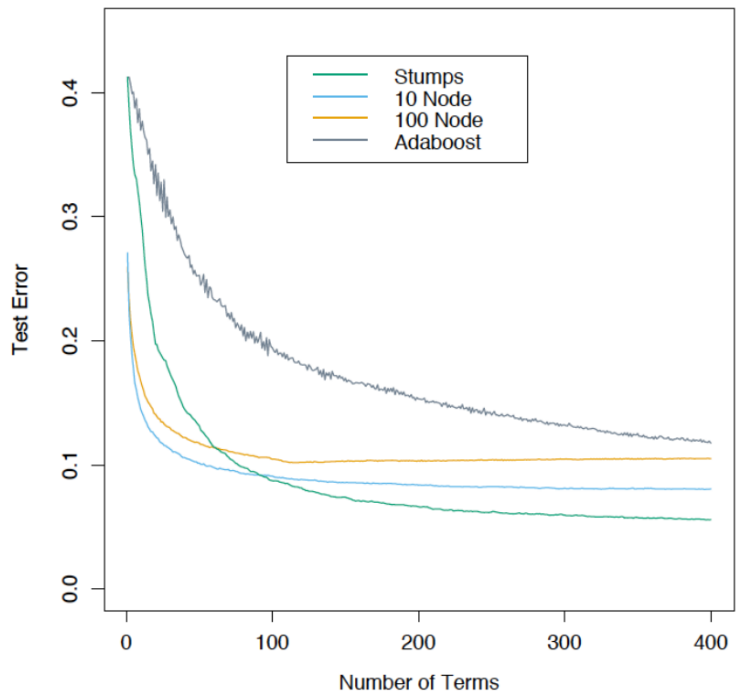
See **Elements of Statistical Learning**, *chapter 10*.

Choosing the size of the trees

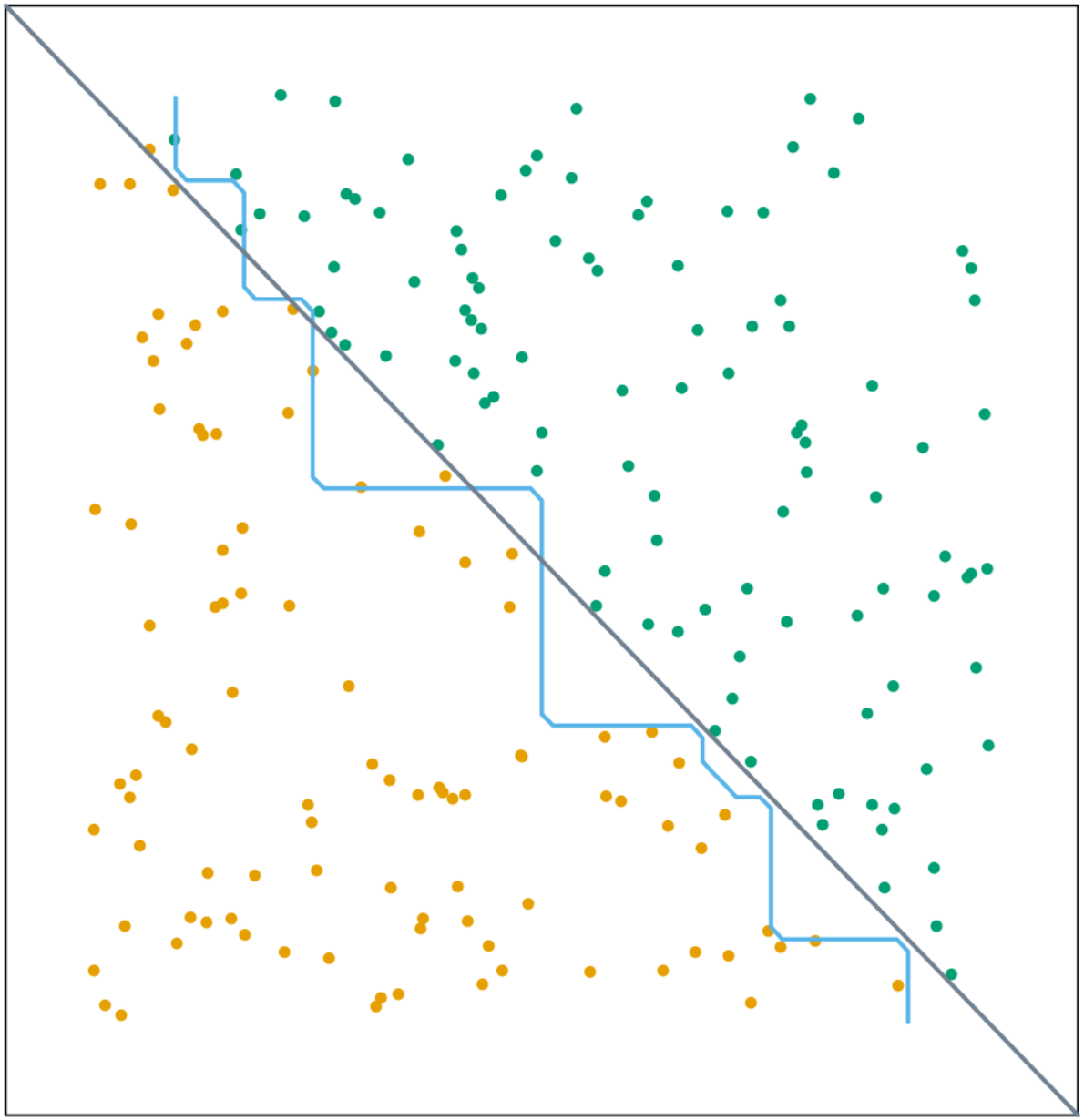
How big should the trees be used in the boosting procedure?

Remember that in bagging we grew large trees, and then either stopped (no pruning) or pruned back using the original training data as a validation set. In boosting, actually, the best methods if to grow small trees with no pruning

In ESL 10.11, it is argued that right size actually depends on the level of the iteractions between the predictor variables. Generally trees with 2–8 leaves work well; rarely more than 10 leaves are needed



Decision boundary



Disadvantages

As with bagging, a major downside is that we lose the simple interpretability of classification trees. The final classifier is a weighted sum of trees, which cannot necessarily be represented by a single tree.

Computation is also somewhat more difficult, though quite straightforward using packages in R. Further, because we are growing small trees, each step can be done relatively quickly (compared to say, bagging).

To deal with the first downside, a measure of variable importance has been developed for boosting (and it can be applied to bagging, also).

Roundup on trees

We've seen three techniques for building tree models.

- **CART**: recursive partitions, pruned back by CV.
- **RF**: average many simple CART trees.
- **Boosted trees**: repeatedly fit simple trees to residuals (from earlier fits).
  - Fast, but tough to avoid over-fit (requires CV): `brt`, `gbm`

There are many other tree-based learning algorithms.

- **Bayesian additive regression trees (BART)**: a sum of trees model.
  - Robust prediction, but suffers with non-constant variance: `BayesTree`
- **Dynamic trees**: grow sequential `particle` trees.
  - Good online, but fit depends on data ordering: `dynaTree`

Trees can be poor in high dimensions,

- but fitting to low dimensional factors (principal components) is a good option.

Credit

Some of the figures have been taken from the ISLR and ESL book.