Trees, Bagging and Random Forests, Boosting

Book Chapter 8

Jared S. Murray
The University of Texas McCombs School of Business

1. Trees
2. Regression Trees
3. Trees: A Summary
4. Fitting Trees: the Bias Variance Trade Off Again
5. Bagging and Random Forests
6. Boosting Trees
7. Variable Importance Measures
8. Bayesian Additive Regression Trees (BART)
9. Trees, Random Forests, Boosting: The California Data

1. Trees

Tree based methods are a major player in data-mining.

Good:

- flexible fitters, capture non-linearity and interactions.
- do not have to think about scale of variables.
- handles categorical and numeric y and x very nicely.
- fast.
- interpretable (when small).

Bad:

Not the best in out-of-sample predictive performance
(*but not bad!!*).

But,

If we **bag** or **boost** trees, we can get the best off-the-shelf prediction available.

Bagging and Boosting are *ensemble methods* that combine the fit from many (hundreds, thousands) of tree models to get an overall predictor.

2. Regression Trees

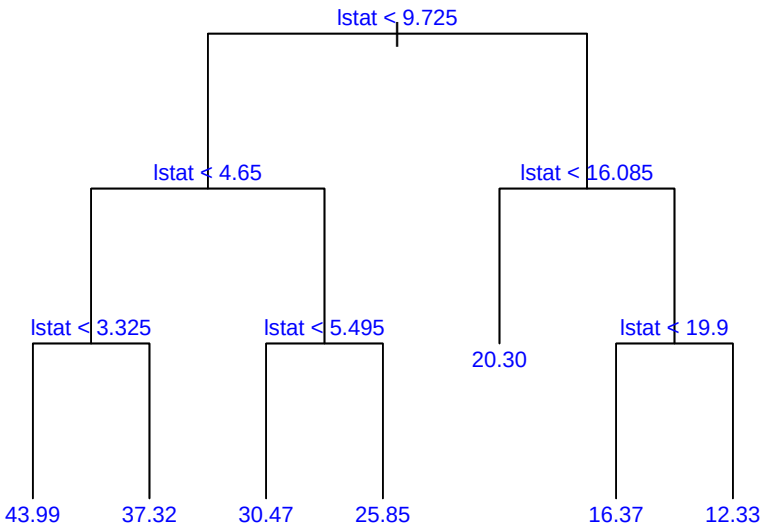
Let's look at a simple 1-dimensional example so that we can see what is going on.

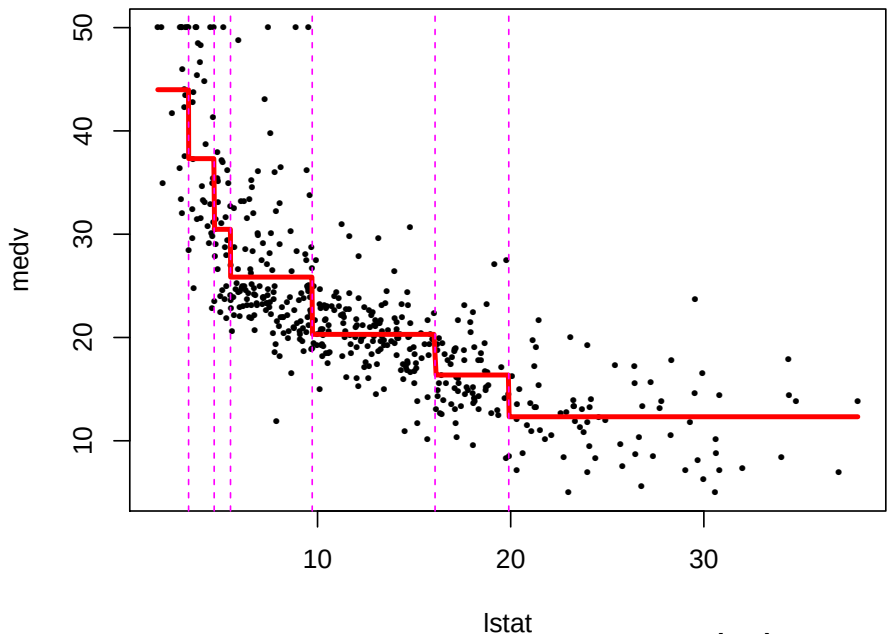
We'll use the Boston housing data and relate x=lstat to y=medval.

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.

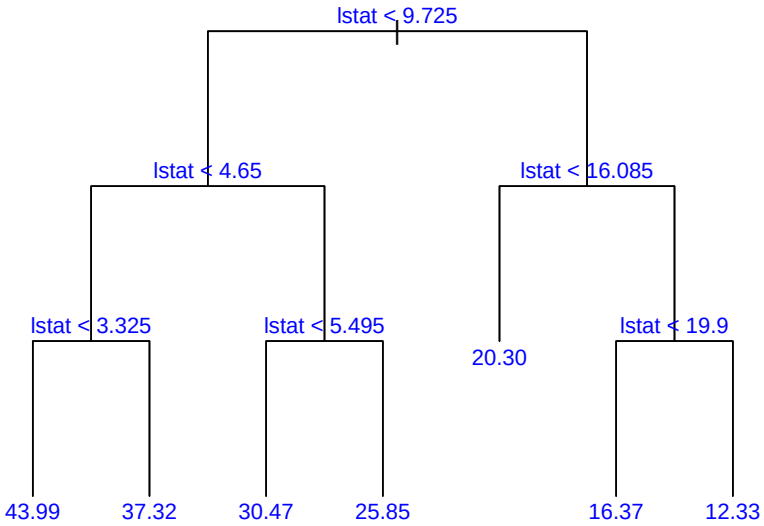
Each observation is sent down the tree until it hits a bottom node or *leaf* of the tree.

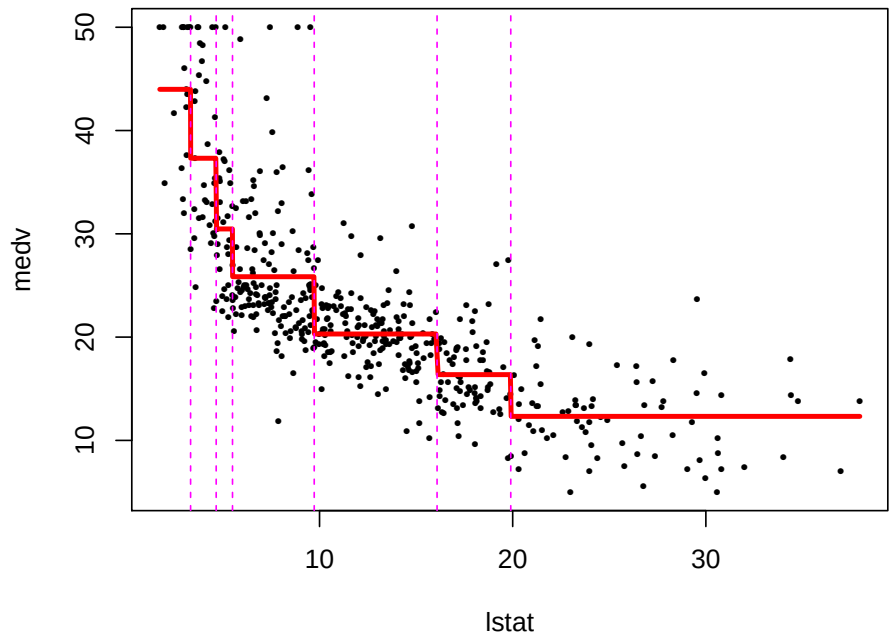




The set of bottom nodes gives us a partition of the predictor (x) space into disjoint regions. At right, the vertical lines display the partition. With just one x, this is just a set of intervals.

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 left at the bottom nodes.





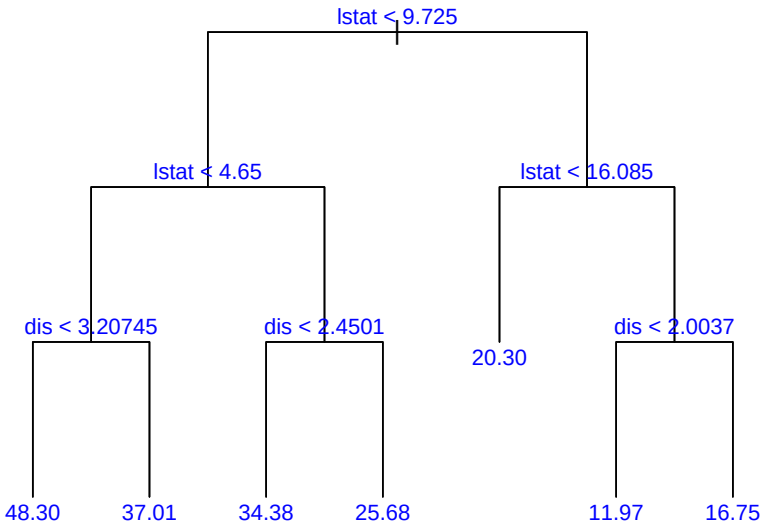
To predict, we just use our step function estimate of $f(x)$.

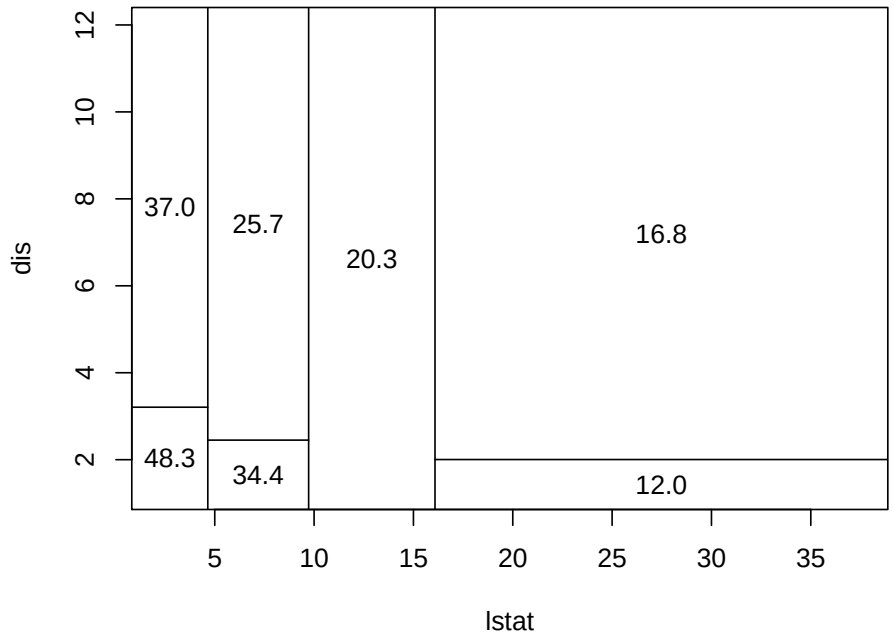
Equivalently, we drop x down the tree until it lands in a leaf and then predict the average of the y values for the training observations in the same leaf.

A Tree with Two Explanatory Variables

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

Now the decision rules can use either of the two x's.





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

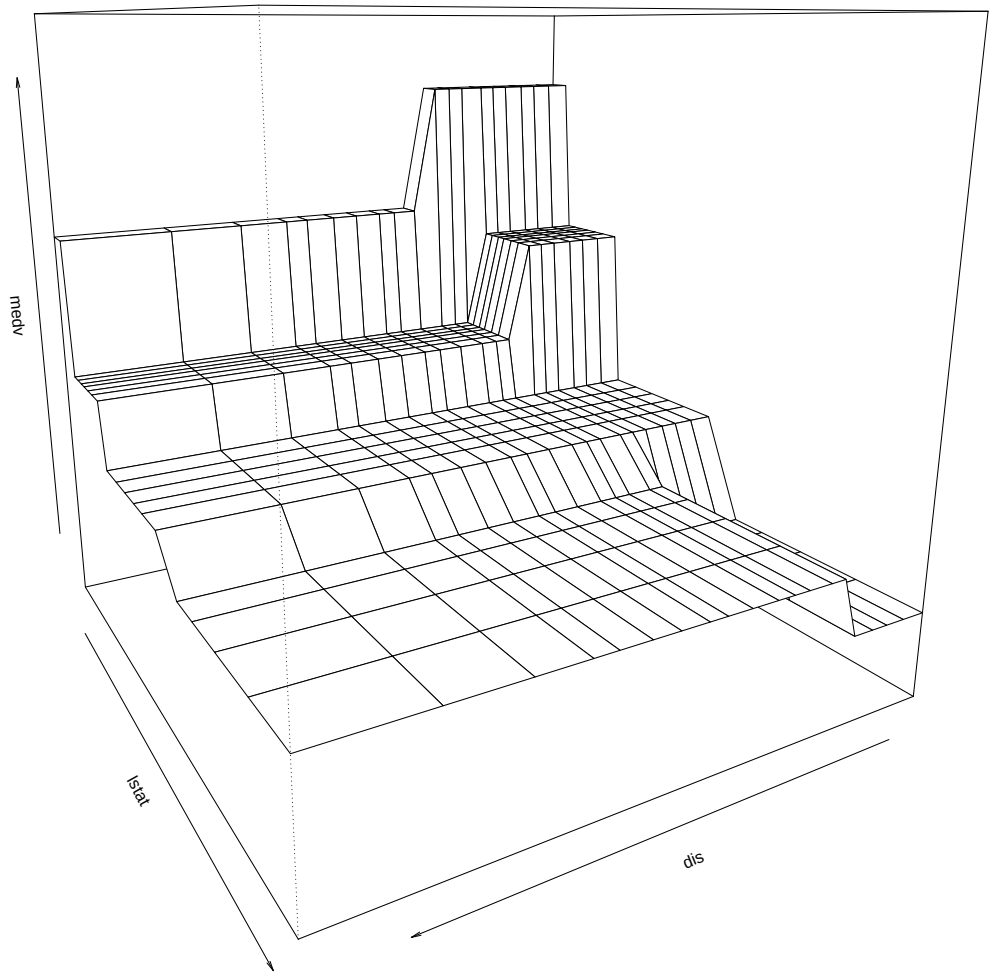
The average y for training observations assigned to a region is printed in each region and at the bottom nodes.

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.

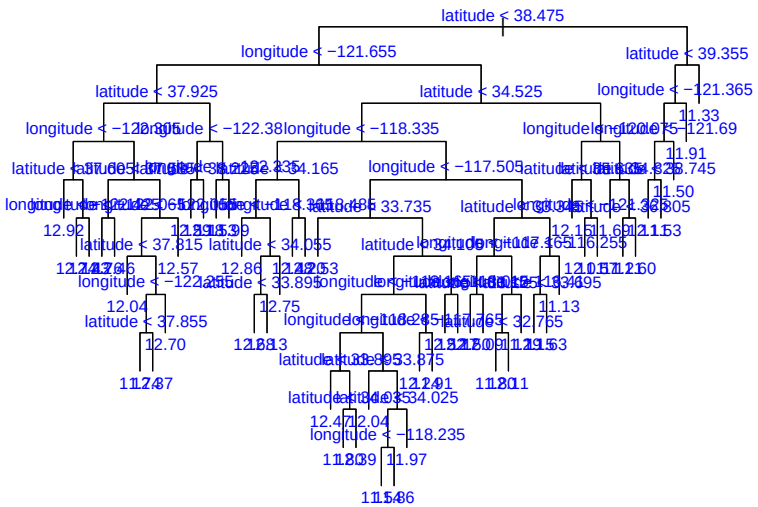
Notice the interaction.

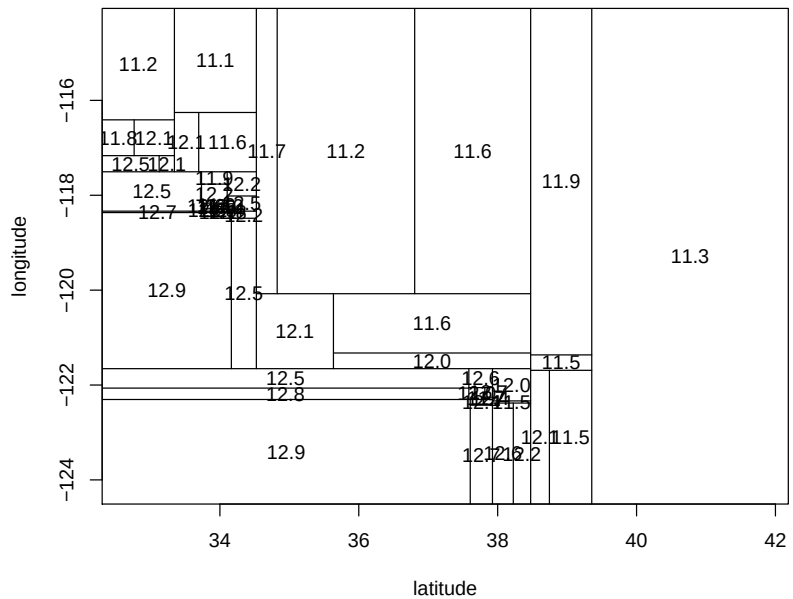
The effect of `dis` depends on `lstat`!!



The California Housing Data

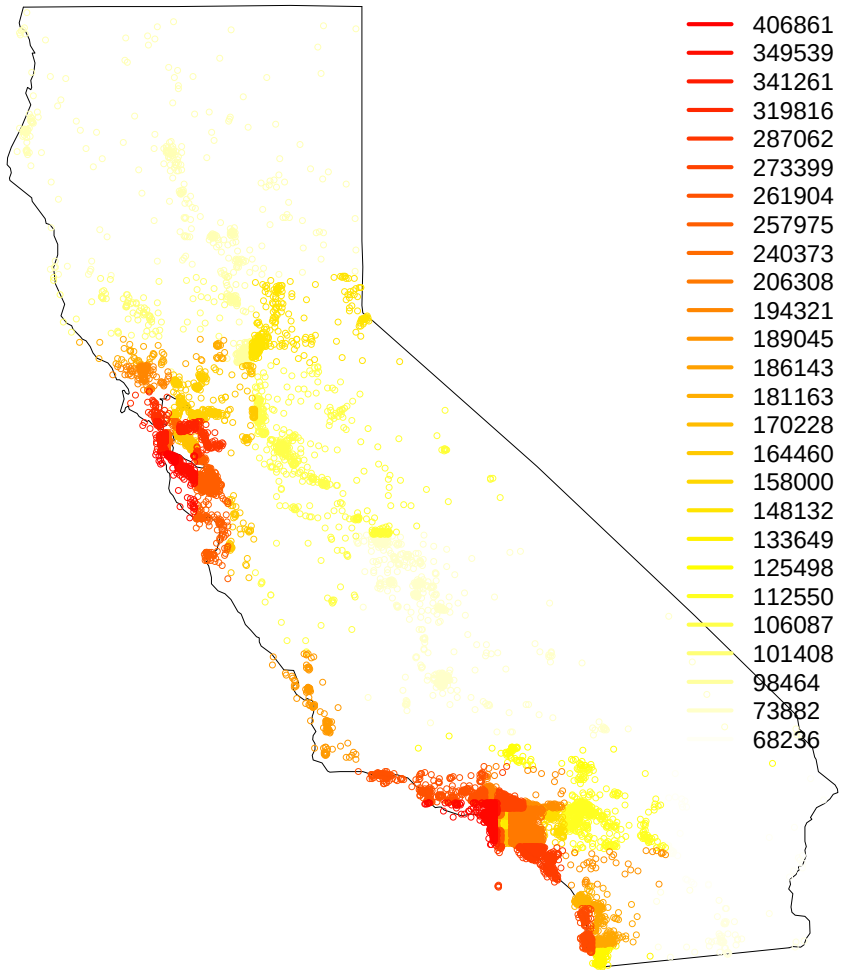
Here is a tree with 50 bottom nodes fit to the California Housing data using only `longitude` and `latitude`.





Don't extrapolate into the ocean!

Here is a view of the fit using the map of the state.
(units are dollars, the logMedVal was exponentiated for the labels).



3. Trees: A Summary

Trees:

- Trees use recursive binary splits to partition the predictor space.
- Each binary split consists of a decision rule which sends x left or right.
- For numeric x_j, the decision rule is of the form if $x_j < c$.
- For categorical x_j, the rule lists the set of categories sent left.
- The set of bottom nodes (or leaves) give a partition of the x space.
- To predict, we drop an out-of-sample x down the tree until it lands in a bottom node.
- For numeric y, we predict the average y value for the training data that ended up in the bottom node.
- For categorical y we use the category proportions for the training data that ended up in the bottom node.

Good:

- Handles categorical/numeric x and y nicely.
- Don't have to think about the scale of x's !!!
- Computationally fast ("scales").
- Small trees are interpretable.
- Variable selection.

Bad:

- Step function is crude, does not give the best predictive performance.
- Hard to assess uncertainly.
- Big trees are not interpretable.

4. Tree Models and the Bias Variance Trade Off

How do we fit trees to data??

We want to fit the observed data well without overfitting (so that we predict well on new data)

In knn this was simple: Check each value of k using cross validation.

With trees (and other methods) the space of possible prediction functions $\hat{f}$ is much larger, and we need more efficient ways to find good candidates

4. Optimizing for the Bias Variance Trade Off

The key idea: Choose $\hat{f}$ to minimize the objective function

$$C_\alpha(f, y) = \underbrace{L(f, y)}_{\text{Loss/Prediction error}} + \underbrace{\alpha\, g(f)}_{\text{complexity penalty}}$$

- L measures prediction error for the *observed* data y
- $g(f)$ is a measure of "complexity" of f
- α is a (positive) number that "trades off" in-sample prediction error (bias) and complexity (variance)

In many cases this is equivalent to a *constrained* minimization problem: Minimize $L(f, y)$ subject to $g(f) \leq d_\alpha$

We usually choose α to minimize out-of-sample loss via cross validation, the same way we picked k in knn.

Tree Models and the Bias Variance Trade Off

To fit a tree, we try to minimize:

$$C(T, y) = L(T, y) + \alpha |T|$$

where,

- $L(T, y)$ is our loss in fitting data y with tree T.
- $|T|$ is the number of bottom nodes in tree T.

For numeric y our loss is usually sum of squared errors, for categorical y we can use the log-loss/deviance or the miss-classification rate (more soon!)

How do we do the minimization ??!!

Now we have a problem.

While trees are simple in some sense, once we view them as variables in an optimization they are large and complex.

A key to tree modeling is the success of the following heuristic algorithm for fitting trees to training data.

(I. Grow Big)

Use a greedy, recursive forward search to build a big tree.

(i)

Start with the tree that is a single node.

(ii)

At each bottom node, search over all possible decision rule to find the one that gives the biggest decrease in loss (increase in fit).

(iii)

Grow a big tree, stopping (for example) when each bottom node has 5 observations in it.

(II. Prune Back)

(i)

Recursively, prune back the big tree from step (I).

(ii)

Give a current pruned tree, examine every pair of bottom nodes (having the same parent node) and consider eliminating the pair.

Prune the pair the gives the biggest decrease in our criterion C.

This is give us a sequence of subtrees of our initial big tree.

(iii)

For a given α, choose the subtree of the big tree that has the smallest C.

So,

Give training data and α we get a tree.

How do we choose α ??

As usual, we can leave out a validation data set and choose the α the performs best on the validation data, or use k-fold cross validation.

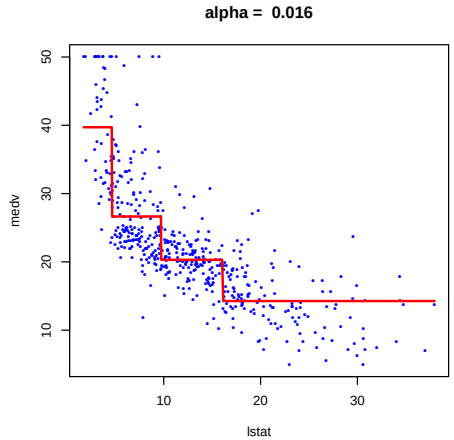
Boston Data, lstat and medv:

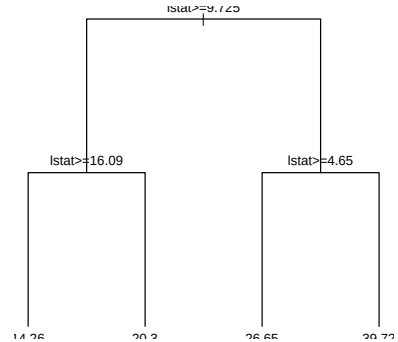
On the right are three different tree fits we get from three different α values (using all the data).

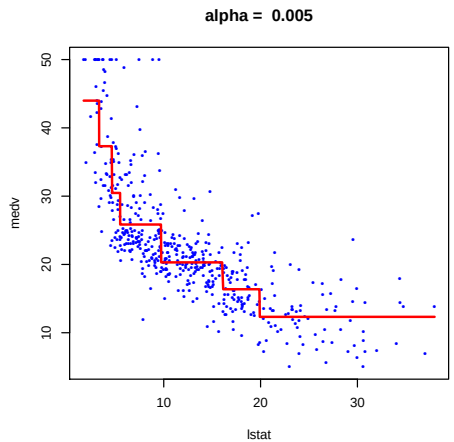
The smaller α is, the lower the penalty for complexity is, the bigger tree you get.

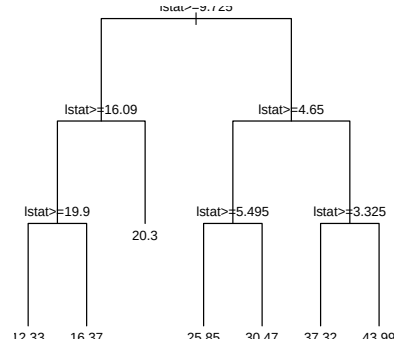
The top tree is a sub-tree of the middle tree, and the middle tree is a sub-tree of the bottom tree.

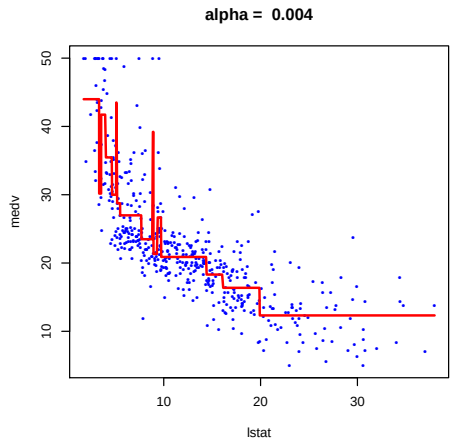
The middle α is the one suggested by CV.

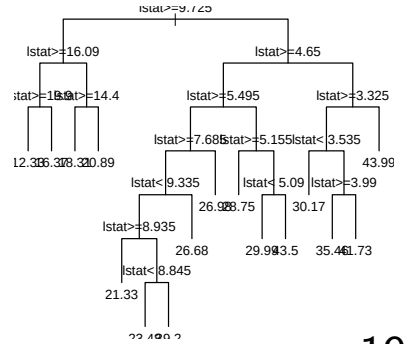








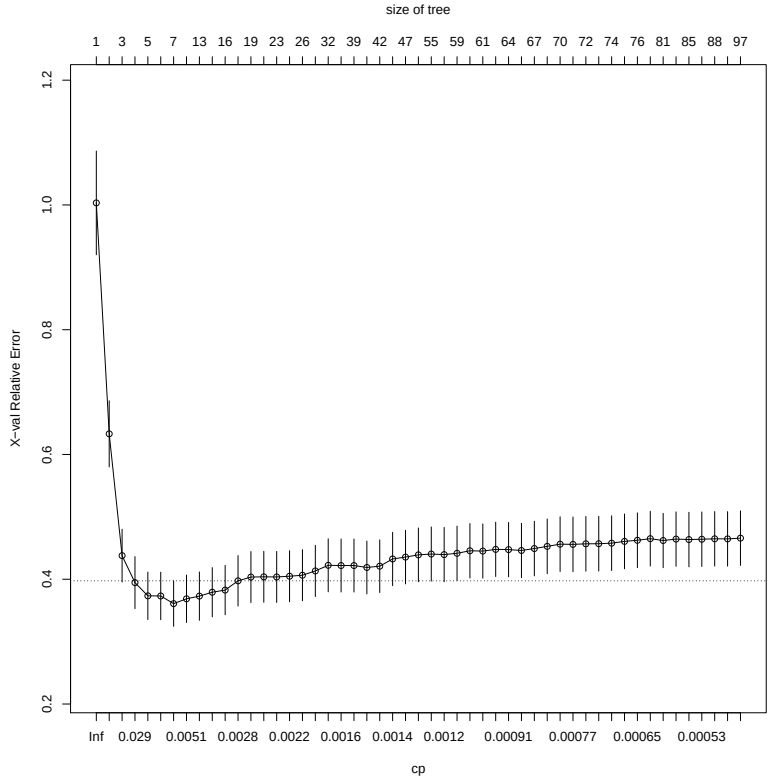




This is the CV plot giving by the R package rpart for y=medv x=lstat.

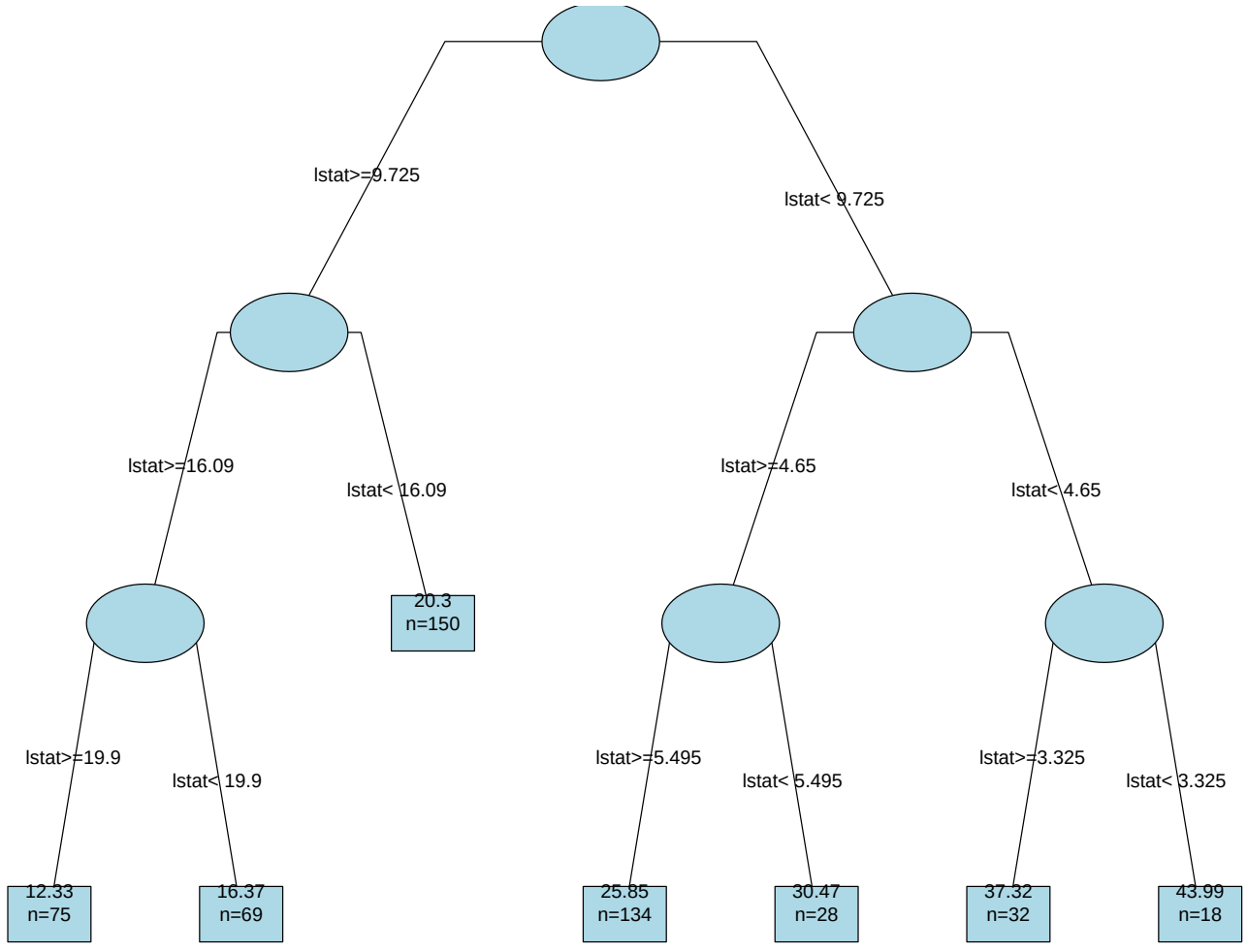
Tree sizes at top of plot, and (a transformation of) α (the "cost-complexity" parameter) on the bottom.

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



Caution: the bottom axis is a transformation of α.

Here is the best CV tree as plotted by rpart.



5. Bagging and Random Forests

Trees are usually at their best when multiple are combined or ensembled.

Our first tree ensembles come from bootstrap aggregating ("bagging") trees.

To **B***ootsrap* **Ag***gregate (Bag) we:*

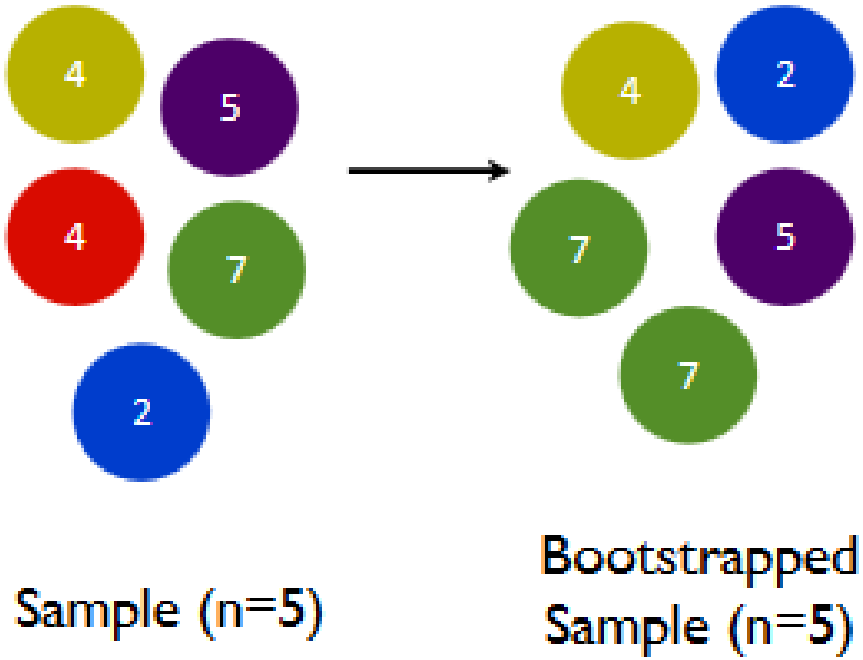
- Take B bootstrap samples from the training data
- Fit a *large* tree to each bootstrap sample (we know how to do this fast!). This will give us B trees.
- Combine the results from each of the B trees to get an overall prediction.

Bootstrap (re)sampling

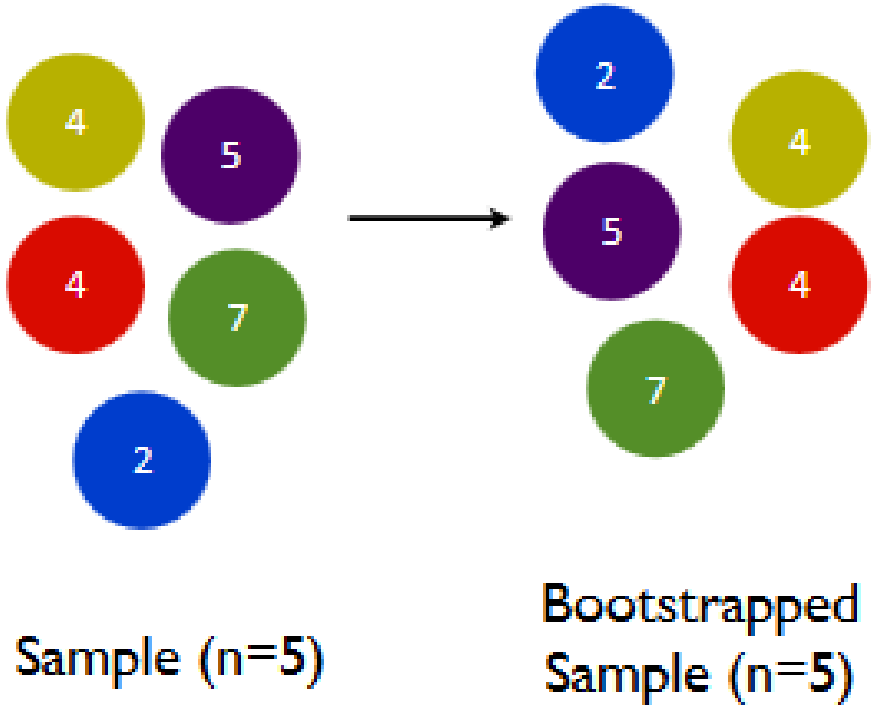
The bootstrap was introduced in statistics to mimic repeated random sampling from a population (think public opinion polling).

By averaging statistics – or predictions – computed using slightly different versions of the original data, we identify patterns that are stable over small random changes in the data. Separate signal from noise!

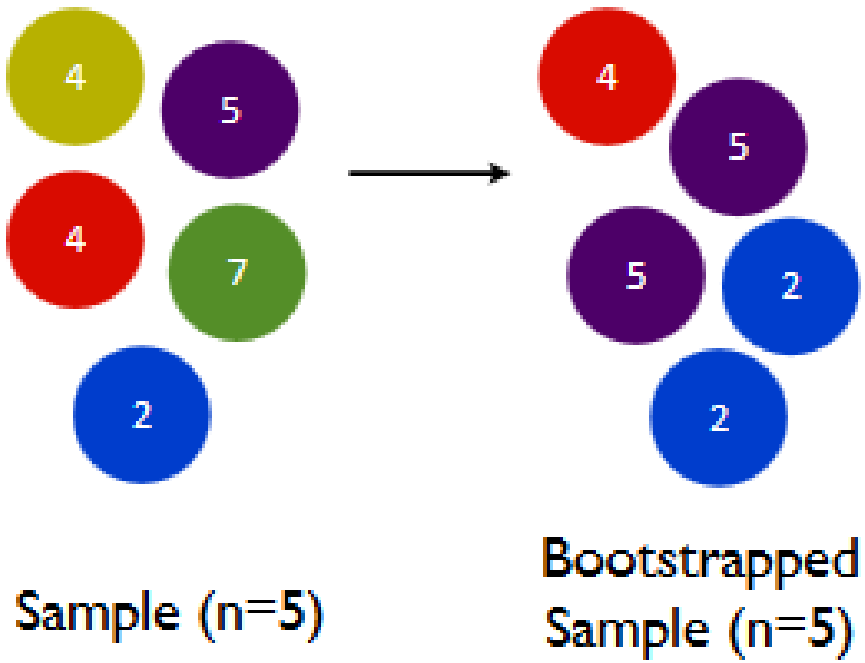
Bootstrap (re)sampling



Bootstrap (re)sampling



Bootstrap (re)sampling



Bootstrap (re)sampling

(Almost) every bootstrap resample will omit some observations entirely – about 35% when n is large – and have multiple copies of others.

Omitted observations are said to be "out-of-bag".

The bootstrap resamples are all similar, however – "representative" of our original sample, at least on average.

Aggregating

For numeric y we can combine the results easily by making our overall prediction the average of the predictions from each of the B trees.

For categorical y, it is not quite so obvious how you want to combine the results from the different trees.
Often people let the trees vote: given x get a prediction from each tree and the category that gets the most votes (out of B ballots) is the prediction.

Alternatively, you could average the $\hat{p}$ from each tree. Most software seems to follow the vote plan.

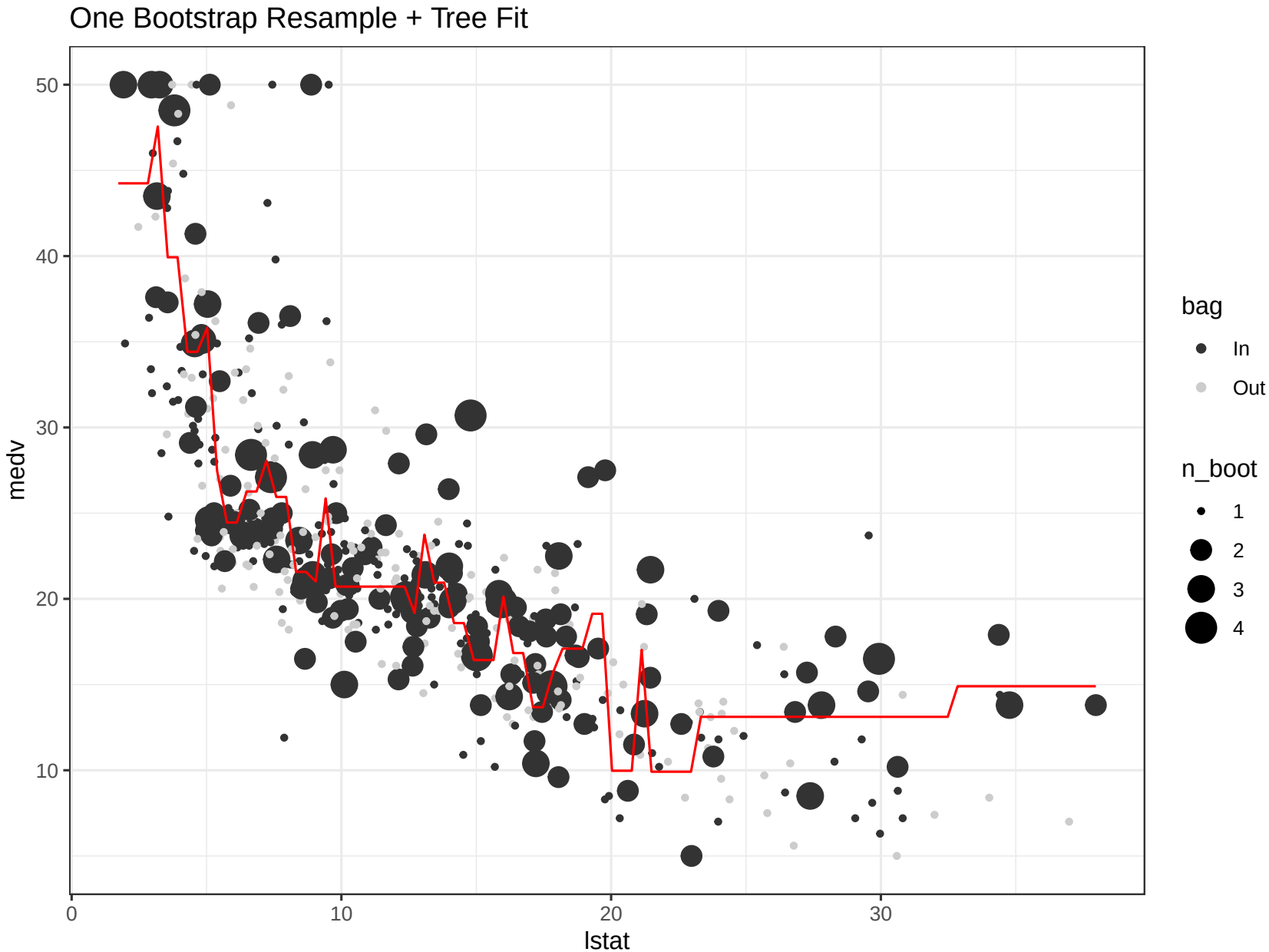
Why on earth would this work??!!

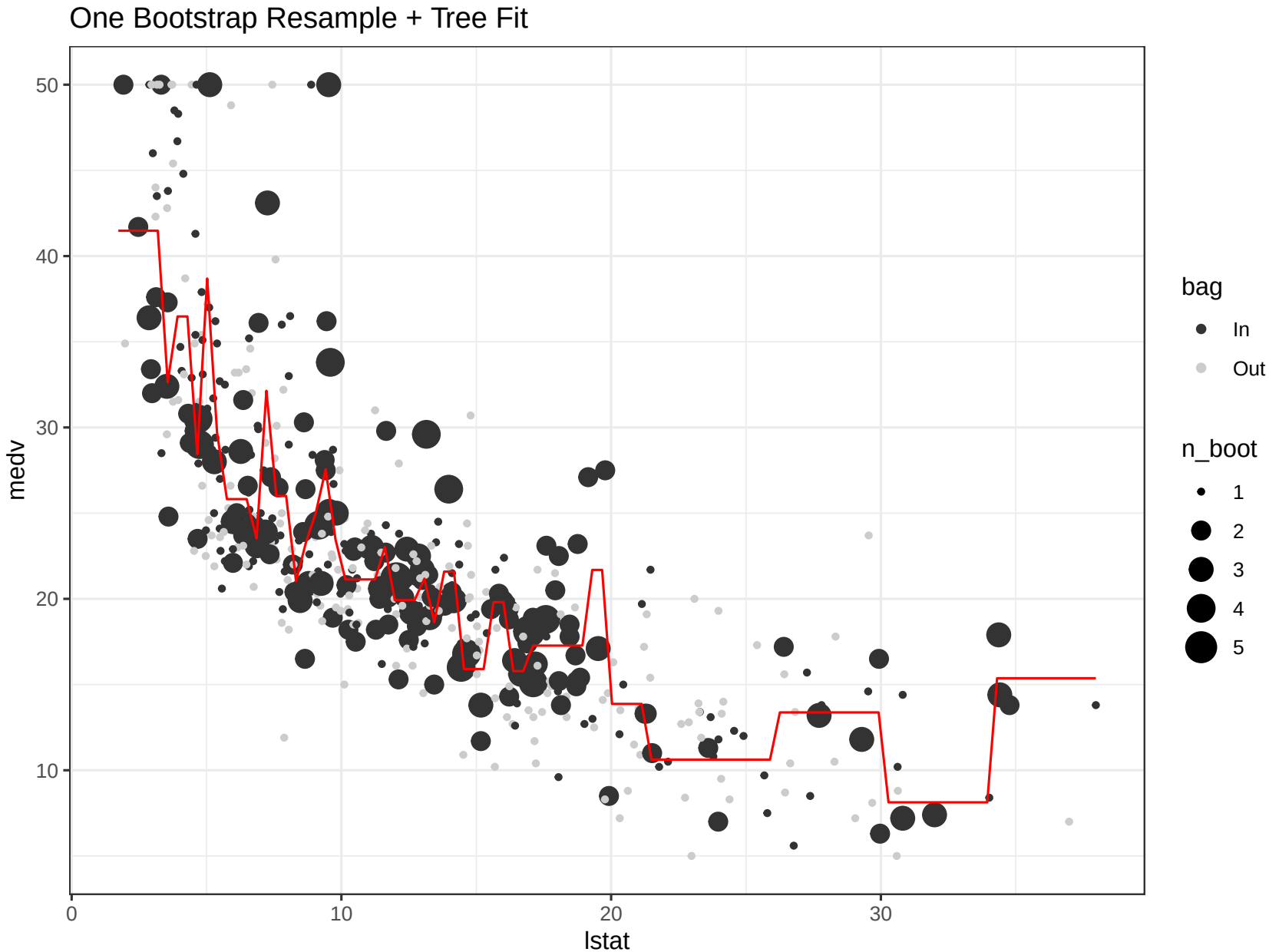
We randomize our data and then build a lot of big (and hence noisy!) trees.

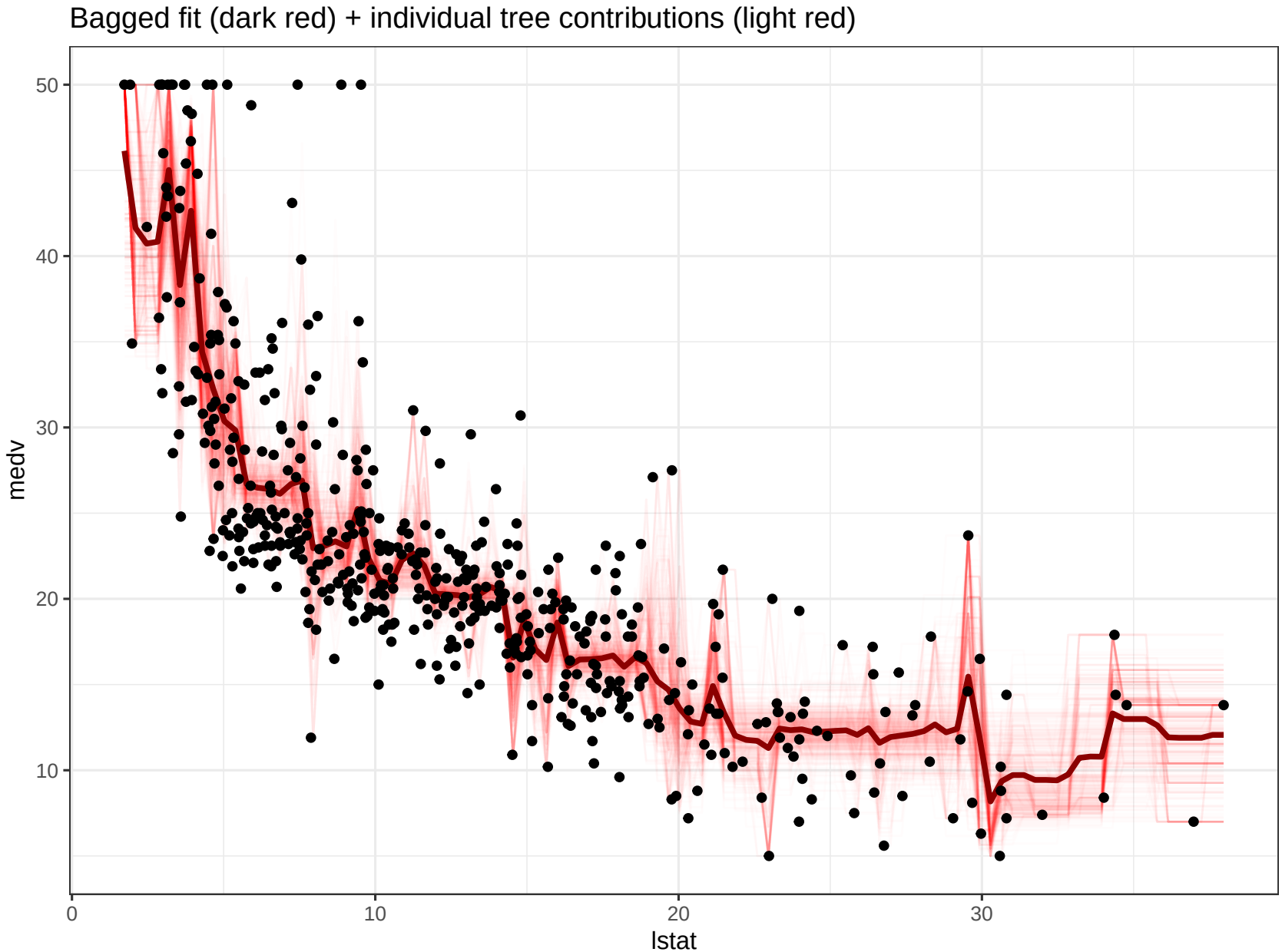
The relationships which are real get captured in a lot of the trees.

The noise from overfitting *to the bootstrap resamples* (eventually) washes out when we take the average: Stuff that happens "by chance" is idiosyncratic to one (or a few) trees/bootstrap resamples.

Brilliant. **Leo Brieman**.


One Bootstrap Resample + Tree Fit
medv
lstat
50
40
30
20
10
0
10
20
30
bag
In
Out
n_boot
1
2
3
4


One Bootstrap Resample + Tree Fit
medv
lstat
50
40
30
20
10
0
10
20
30
bag
In
Out
n_boot
1
2
3
4
5


Bagged fit (dark red) + individual tree contributions (light red)
medv
lstat
50
40
30
20
10
0
10
20
30

Random Forests:

Random Forests starts from Bagging and adds another kind of randomization.

Rather than searching over all the individual variables in x when we do our greedy build of the big trees, we randomly sample a subset of m variables to search over.

This makes the big trees "move around more" so that we explore a rich set of trees, *but the important variables will still shine through!!*.

Have to choose:

- B: number of Bootstrap samples (hundreds, thousands).
- m: number of variables to sample.

  A common choice is $m = \sqrt{p}$,
  where p is the dimension of x.

Note:

Bagging is Random Forests with $m = p$.

Note:

There is no explicit regularization parameter as in the lasso and single tree prediction.

OOB Error Estimation:

OOB is "Out of Bag".

For a bootstrap sample, the observations chosen are "in the bag" and the rest are out.

There is a very nice way to estimate the out-of-sample error rate when bagging.

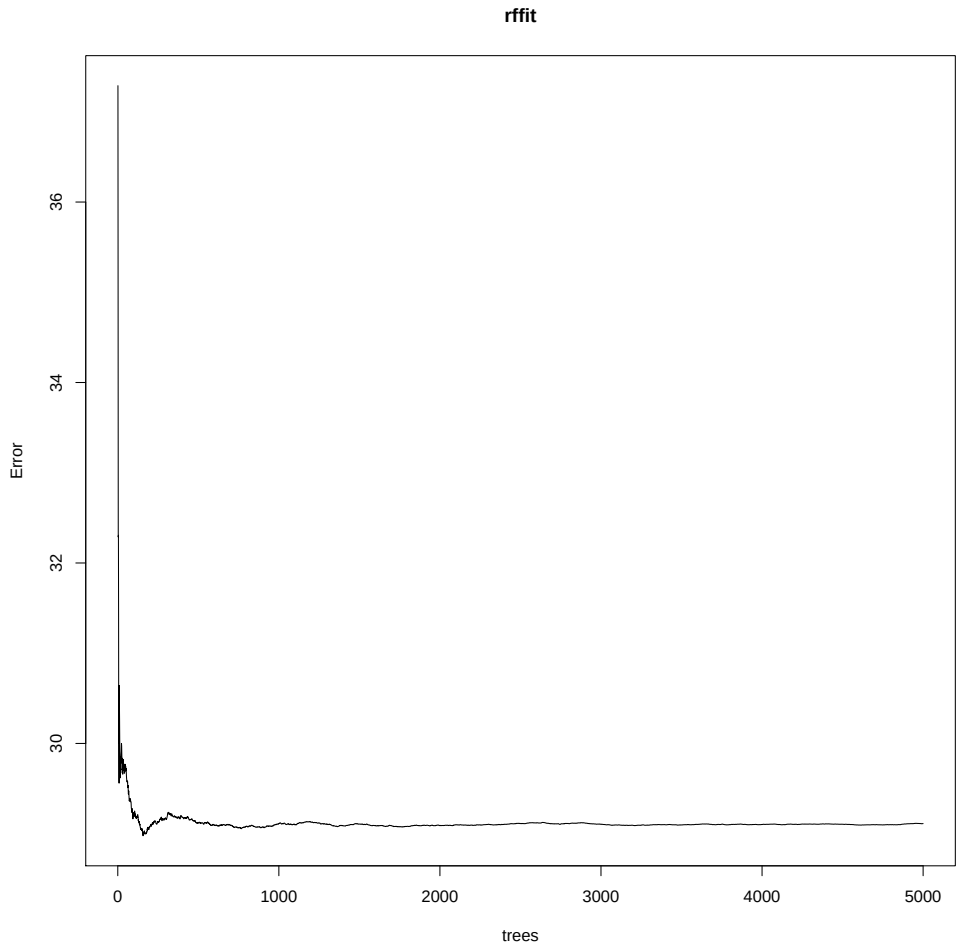
One can show that, on average, each bagged tree makes use of about 2/3 of the observations.

By carefully keeping track of which bagged trees use which observations you can get out-of-sample predictions.

Bagging for Boston: y=medv, x=lstat.

Here is the error estimation as a function of the number of trees based on OOB.

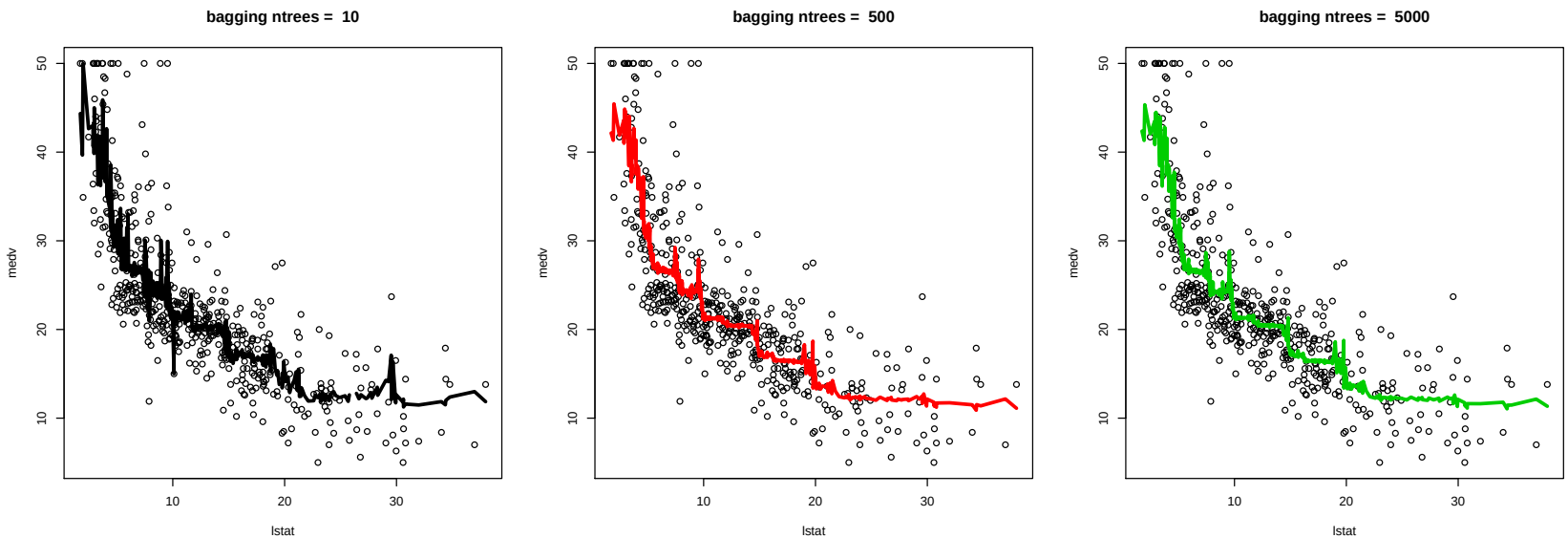
Suggests you just need a couple of hundred trees.



Bagging for Boston: y=medv, x=lstat.

With 10 trees our fit is too jumbly.

With 1,000 and 5,000 trees the fit is not bad and very similar.



6. Boosting Trees

Like Random Forests, boosting is an *ensemble method* is that the overall fit it produced from many trees.

The idea however, is totally different!!

In Boosting we:

- Fit the data with a single tree.
- Crush the fit so that it does not work very well.
- Look at the part of y not captured by the crushed tree and fit a new tree to what is "left over".
- Crush the new tree. Your new fit is the sum of the two trees.
- Repeat the above steps iteratively. At each iteration you fit "what is left over" with a tree, crush the tree, and then add the new crushed tree into the fit.
- Your final fit is the sum of many trees.

This one is actually made clearer by the mathematical notation. This is Algorithm 8.2 (page 322) in the book.

For Numeric y:

(i) Set $\hat{f}(x) = 0$. $r_i = y_i$ for all i in the training set.

(ii) for $b = 1, 2, \ldots B$, repeat:

- Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
- Update $\hat{f}$ by adding in a shrunken version of the new tree: $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$.
- Update the residuls: $r_i \leftarrow r_i - \lambda \hat{f}^b(x)$.

(iii) Output the boosted model:

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

Note:

λ is the "crushing" or "shrinkage" parameter.

It make each new tree a *weak leaner* in that is only does a little more fitting.

Have to choose:

- B, number of interations (the number of trees in the sum) (hundreds, thousands).
- d, the size of each new tree.
- λ, the crush factor.

Note:

Boosting for categorical y works in an analogous manner but it is more messy how you define "the part left over", you can't just use residuals.

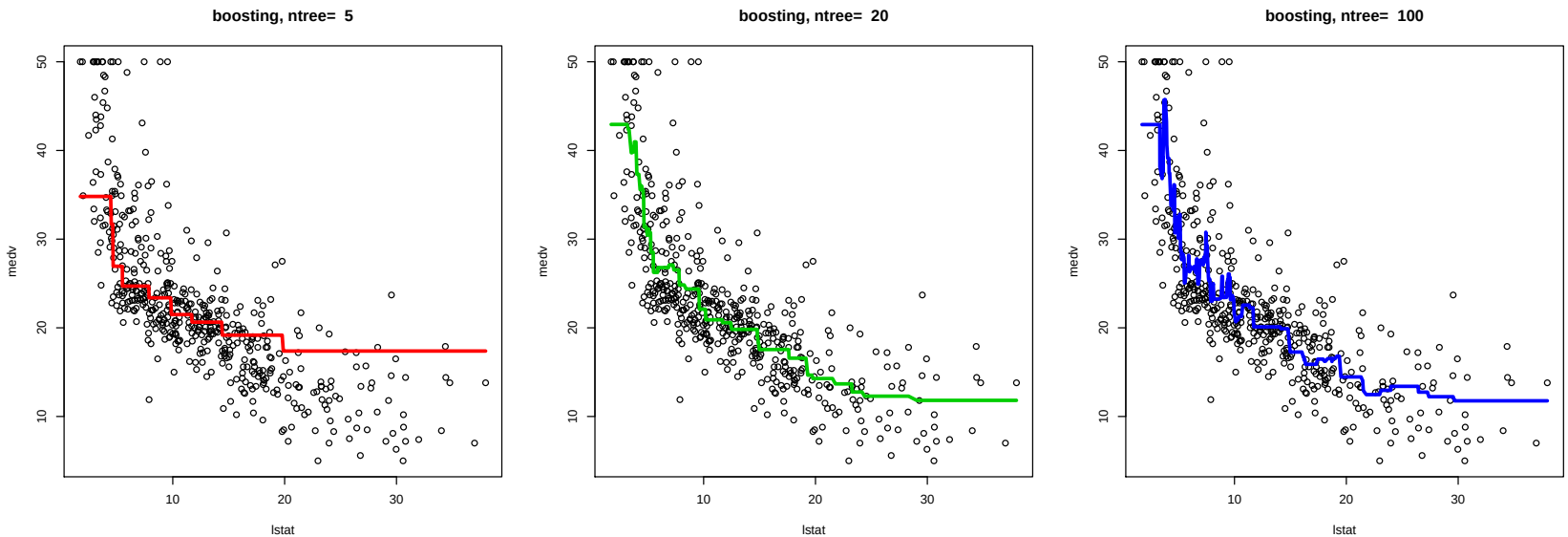
Also you can't just add up the fit.

But, it is the same idea:

- fit.
- crush fit.
- fit what is left over.
- aggregate crushed fits.

Boosting for Boston: y=medv, x=lstat:

Here are some boosting fits where we vary the number of trees, but fix the depth at 2 (suitable with 1 x) and shrinkage = λ at .2.



Again, this ensemble method gets away from the crude step function given by a single tree.

7. Variable Importance Measures

The ensemble methods Random Forests and Boosting can give dramatically better fits than simple trees. Out-of-sample, they can work amazingly well. They are a breakthrough in statistical science.

However, they are certainly not interpretable!!

You cannot look at hundreds or thousands of trees.

Nonetheless, by computing summary measures, you can get some sense of how the trees work.

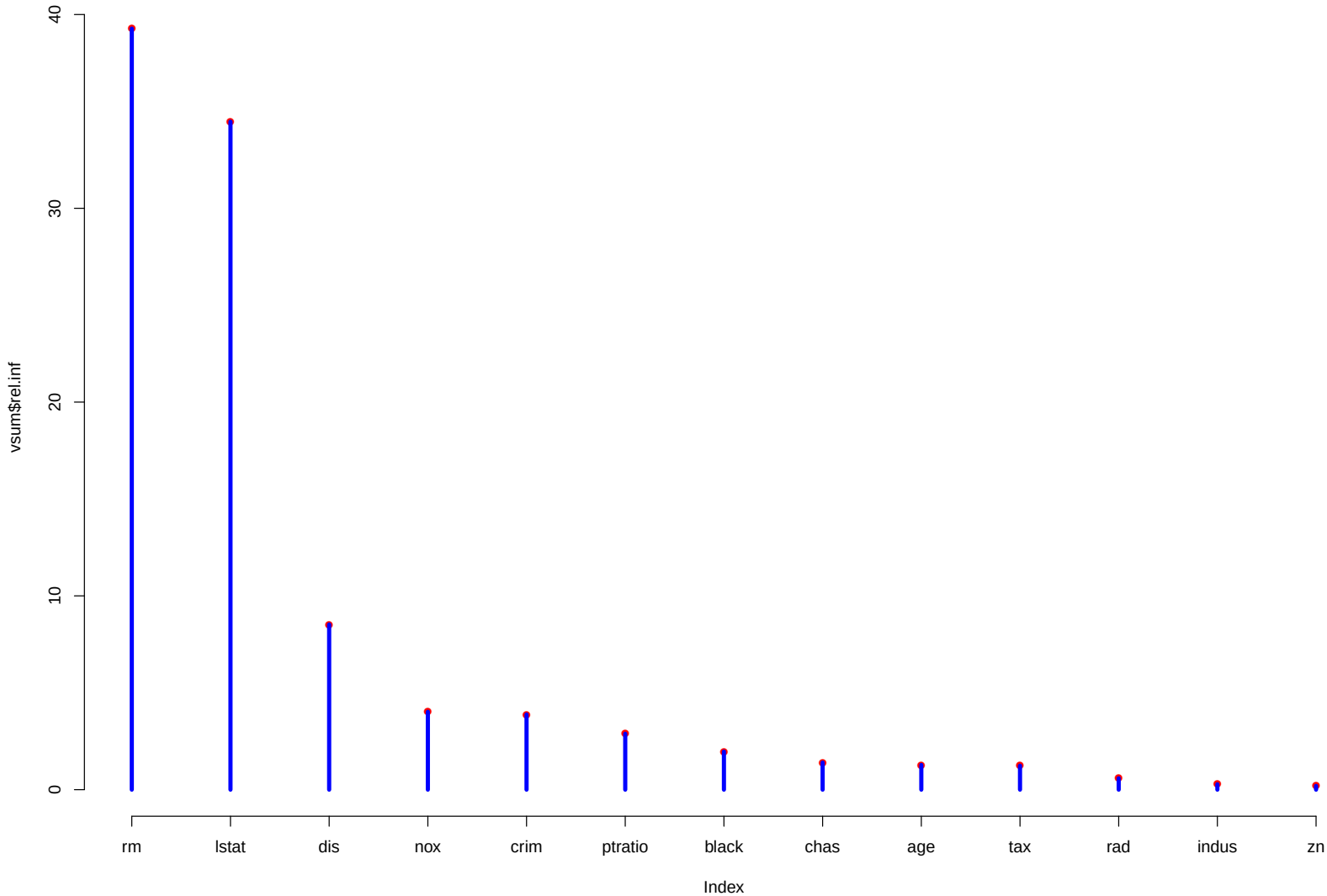
In particular, we are often interested in which variables in x are really the "important" ones.

What we do is look at the splits (decision rules) in a tree and pick out the ones that use a particular variable. Then we can add up the reduction in loss (eg residual sum of squares) due to the splits using the variable.

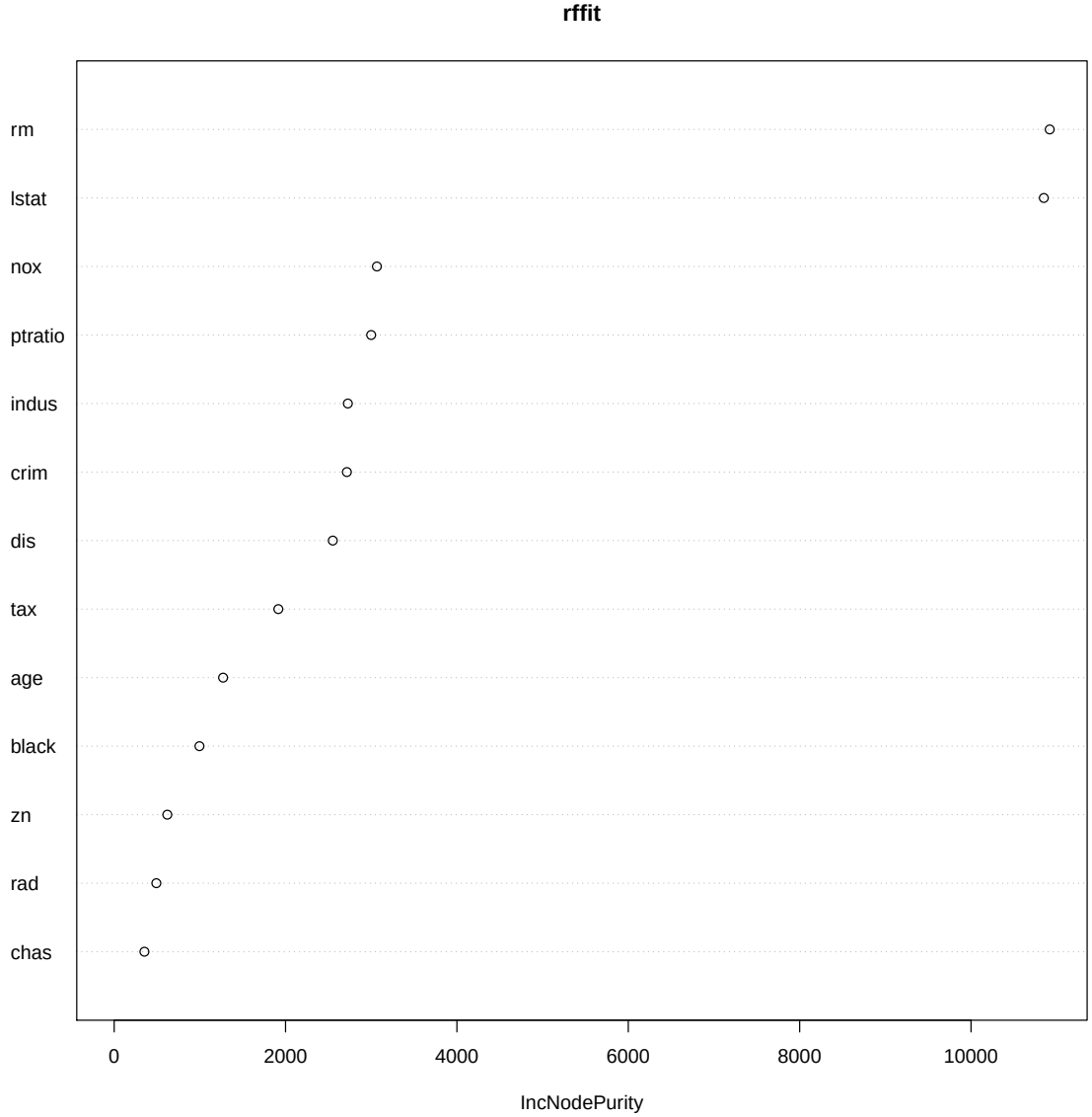
For a single tree we are done.

For bagging we can average the effect of a variable over the B trees and for Boosting we can sum the effects.

Here is the variable importance for the Boston data with all the variables obtained from a Boosting fit.



Here is the variable importance for the Boston data with all the variables obtained from a Random Forests fit.



8. Bayesian Additive Regression Trees (BART)

$$y_i = f(x_i) + \epsilon_i$$

$$f(x) = \sum_{h=1}^{m} g(x, T_h, M_h)$$

- each T_h is a tree with parameter M_h
- the idea is to have lots of trees, i.e., big m with very small, *randomly short* trees
- Question: what does this look like?
- fitting process uses a Markov Chain Monte Carlo algorithm
- *consistently a top performer in a variety of situations!!*
- BART package example...

9. Trees, Random Forests, Boosting: The California Data

Let's try all this stuff on the California Housing data.

That is, we'll try trees, Random Forests, and Boosting.

How will they do !!!

We'll do a simple three set approach since we have a fairly large data set.

We randomly divide the data into three sets:

Train: 10,320 observations.

Validation: 5,160 observations.

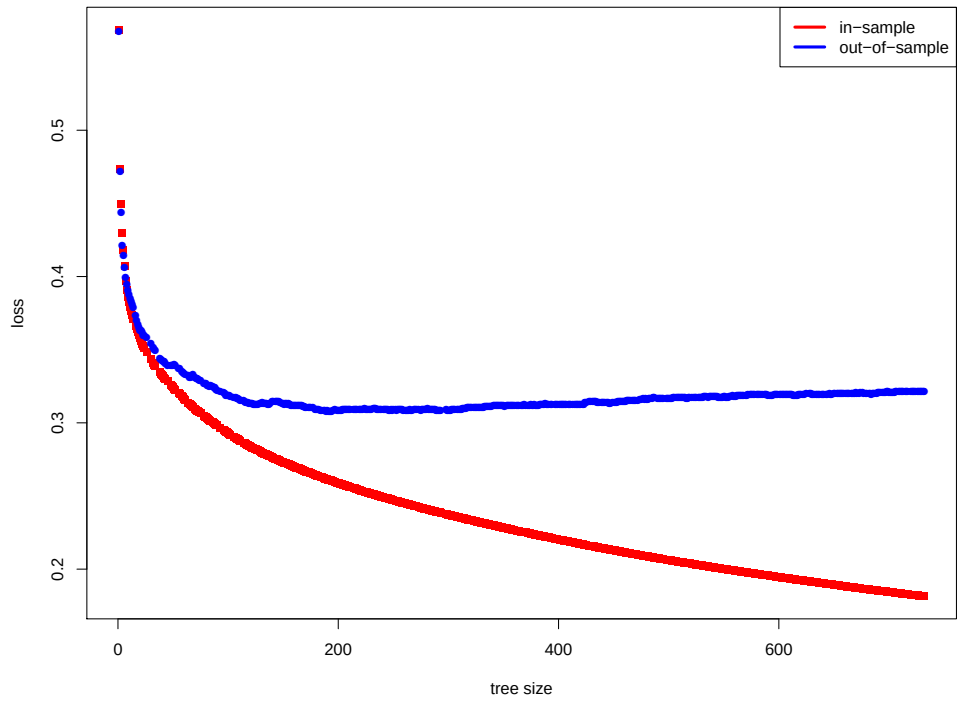
Test: 5,160 observations.

We,

- Try various approaches using the Training data to fit and see how well we do out-of-sample on the Validation data set.
- After we pick an approach we like, we fit using the combined Train+Validation and then predict on the test to get a final out-of-sample measure of performance.

Trees:

- Fit big tree on train.
- For many cp=α, prune tree, giving trees of various sizes.
- Get in-sample loss on train.
- Get out-of-sample loss on validation.



The loss is RMSE.

We get the smallest out-of-sample loss (.307) at a tree size of 194.

Boosting:

Let's try:

- maximum depths of 4 or 10.
- 1,000 or 5,000 trees.
- $\lambda = .2$ or .001.

`olb`:
out-of-sample loss

`ilb`:
in-sample loss.

min loss of .231 is quite a bit better than trees!

	tdepth	ntree	lam	olb	ilb
1	4	1000	0.001	0.414	0.416
2	10	1000	0.001	0.378	0.380
3	4	5000	0.001	0.279	0.282
4	10	5000	0.001	0.252	0.250
5	4	1000	0.200	0.232	0.164
6	10	1000	0.200	0.233	0.098
7	4	5000	0.200	0.231	0.081
8	10	5000	0.200	0.233	0.014

Random Forests:

Let's try:

- m equal 3 and 9 (Bagging).
- 100 or 500 trees.

`olrf` is the out-of-sample loss and `ilrf` is the in-sample loss.

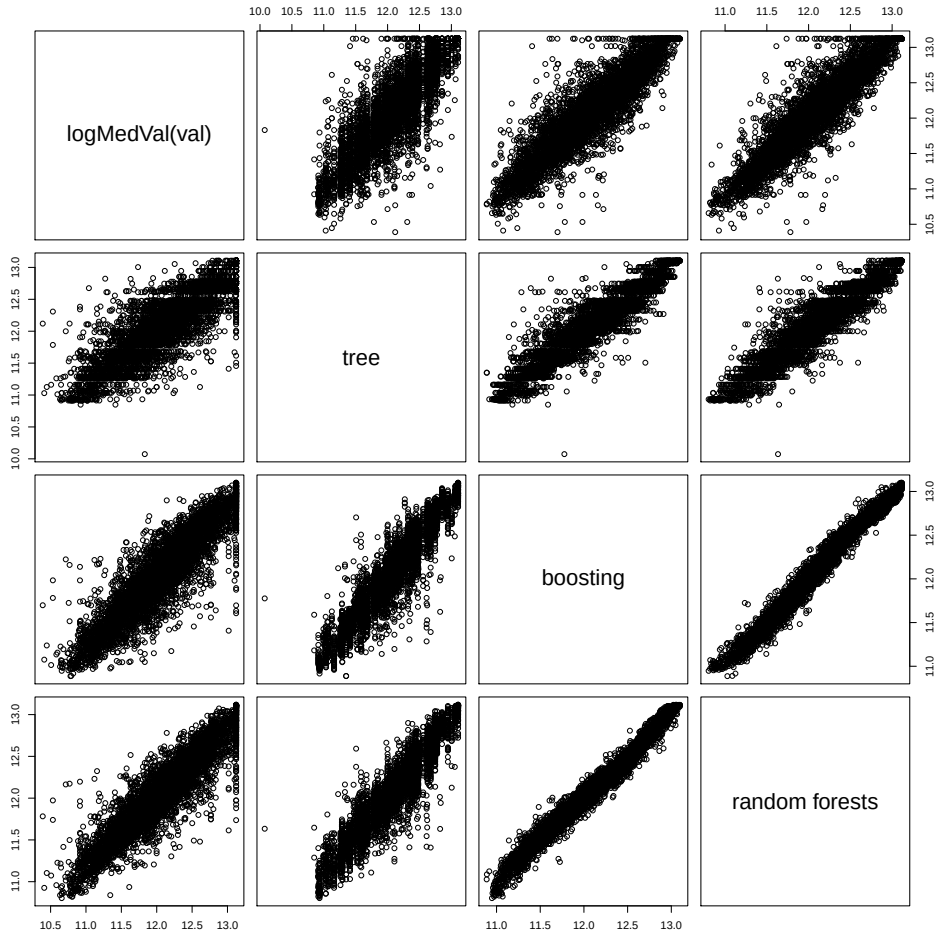
	mtry	ntree	olrf	ilrf
1	9	100	0.241	0.255
2	3	100	0.236	0.250
3	9	500	0.241	0.253
4	3	500	0.233	0.245

Minimum loss is comparable to boosting.

Let's compare the predictions on the Validation data with the best performing of each of the three methods.

It does look like Boosting and Random Forests are a lot better than a single tree.

The fits from Boosting and Random Forests are not too different (this is not always the case).

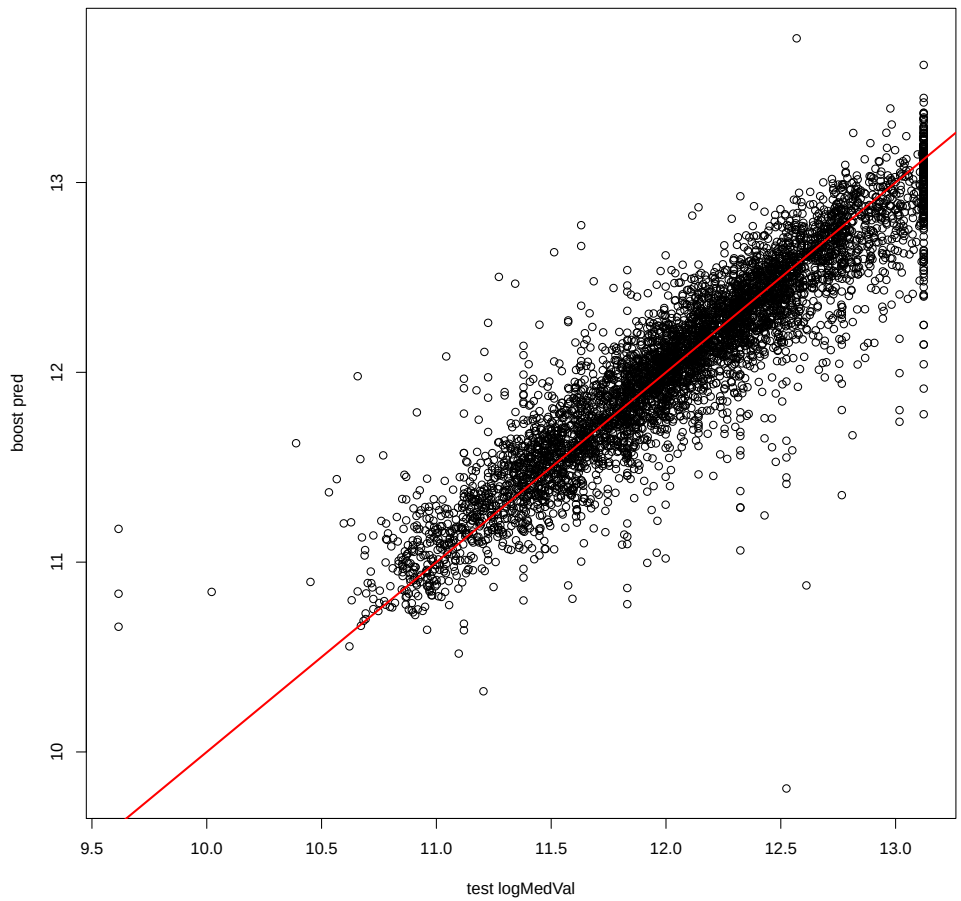


Test Set Performance, Boosting

Let's fit Boosting using depth=4, 5,000 trees, and shrinkage = λ=.2 on the combined train and validation data sets.

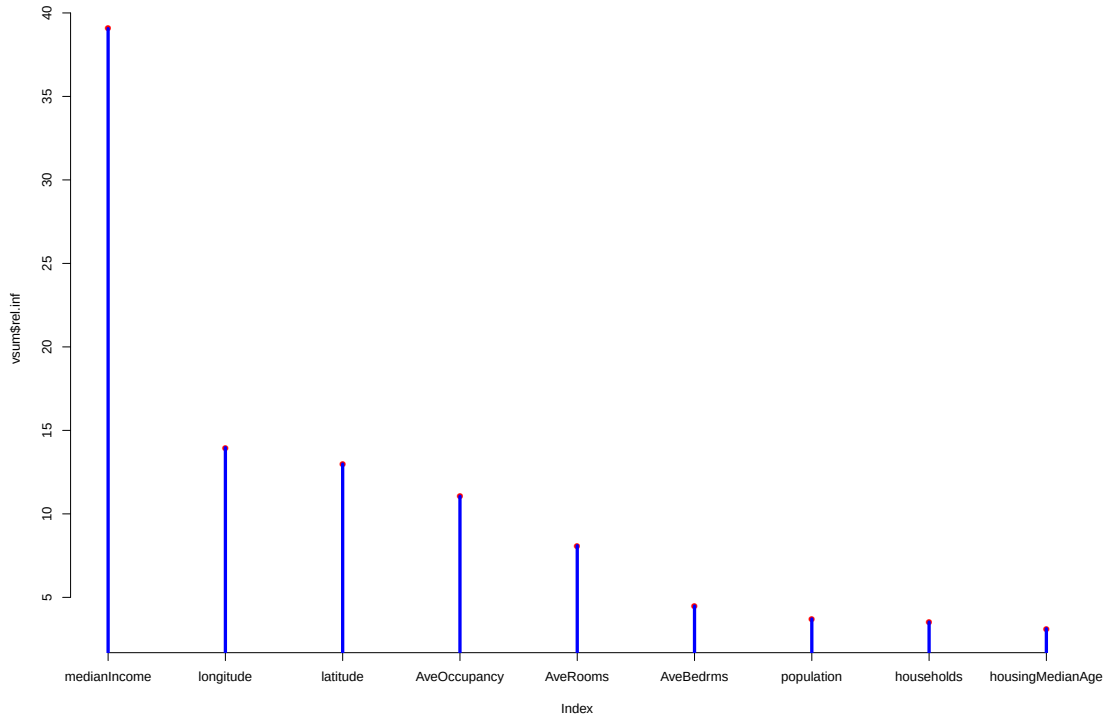
The RMSE on test data is .231.

This is consistent with what we had before from the train-validation data.



Boosting gives us a measure of variable importance:

	var	rel.inf
1	medianIncome	39.065051
2	longitude	13.965980
3	latitude	12.985643
4	AveOccupancy	11.055079
5	AveRooms	8.093967
6	AveBedrms	4.480044
7	population	3.708594
8	households	3.520058
9	housingMedianAge	3.125583

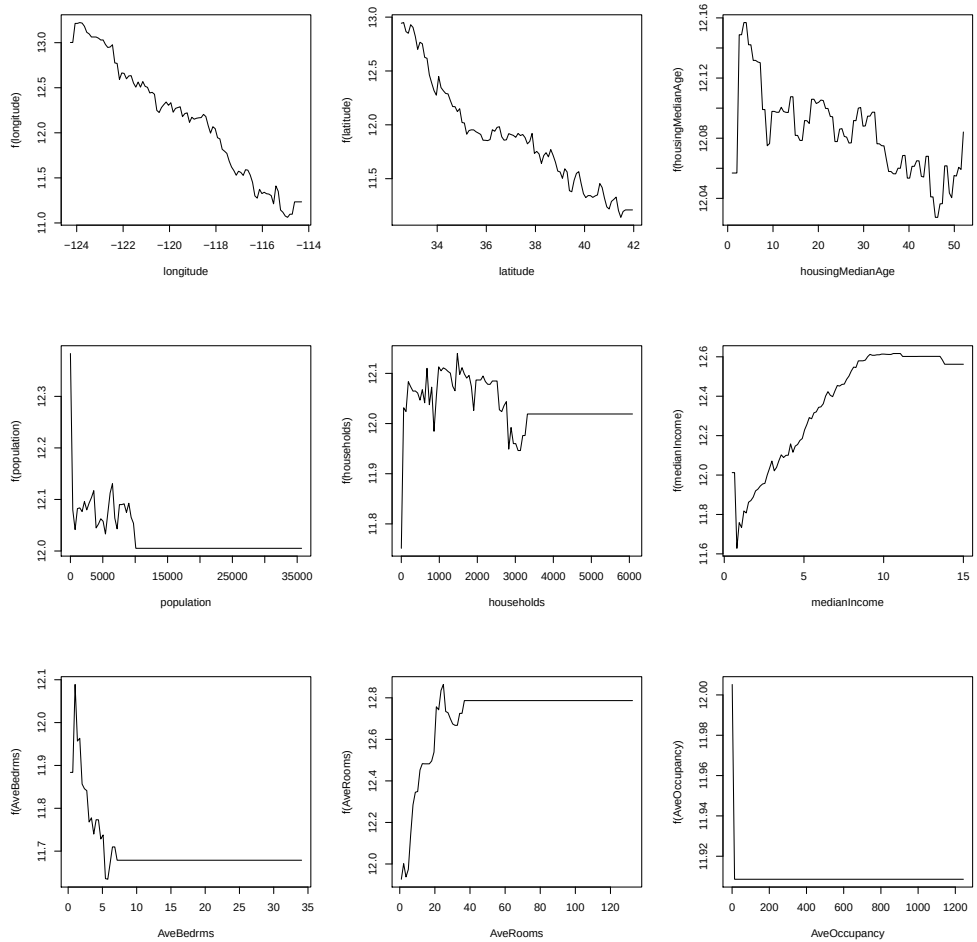


medianIncome is by far the most important variable.
After that, it is location - *makes sense*.

The boosting package also generated a "partial dependence plot" (see here) which is supposed to show the plot of x_j vs. y for each variable while averaging over the values of other x's.

This is supposed to be a plot of the "marginal" effect of x_j on the predicted y=logMedVal for each $j = 1, 2, \ldots, 9$.

This can be misleading when the x's are correlated and/or there are interactions!



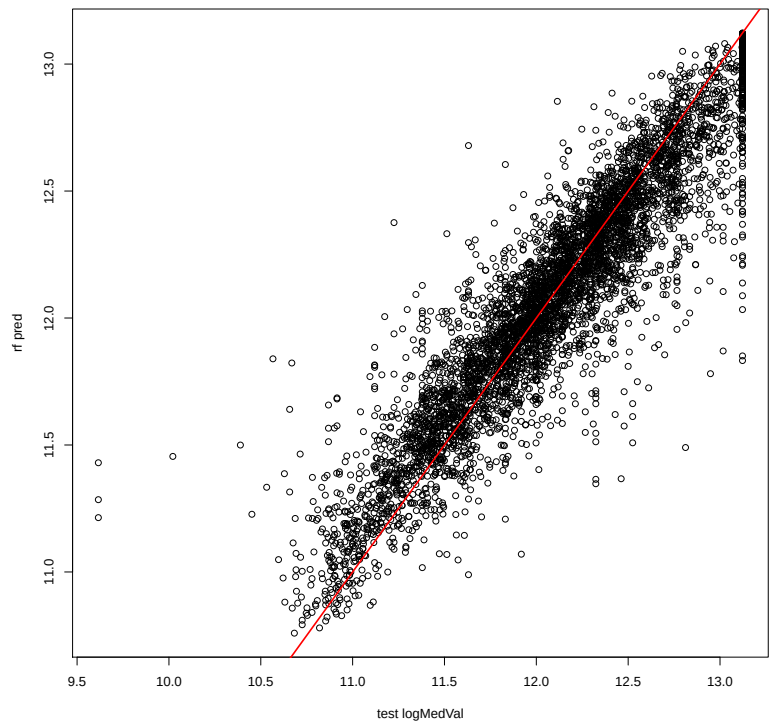
Test Set Performance, Random Forests

Let's fit Random Forests using m=3 and 500 trees on the combined train and validation data sets.

Let's see how the predictions compare to the test values.

Not too bad!!

The RMSE is .23,
so our train-validation results hold up.



Random Forests: Variable Importance:

Random Forests give a measure of variable importance. It just adds up how much the loss decreases every time a variable is used in a split.

Not suprisingly, medianIncome is by far the most important variable.