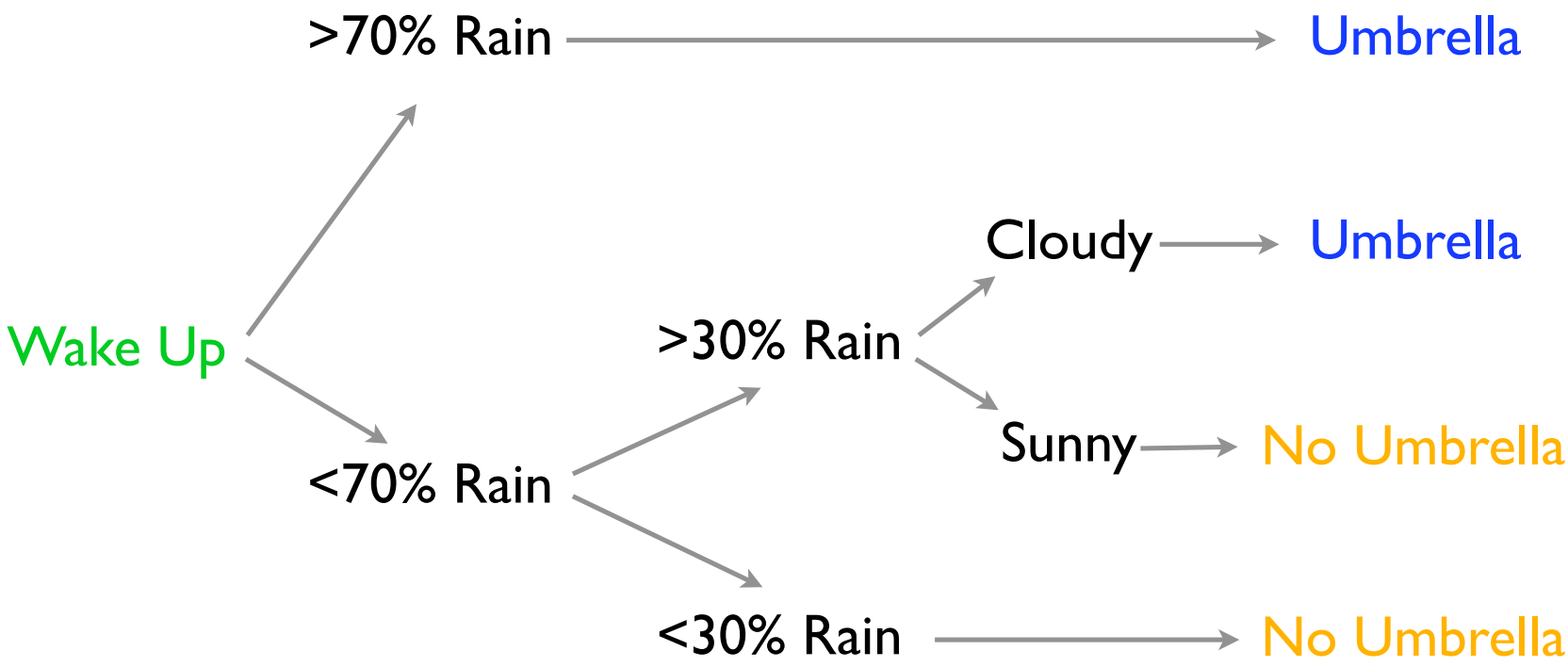
Empirical Bayesian Forests

Matt Taddy, Chicago Booth

with Chun-Sheng Chen, Jun Yu, and Mitch Wyle at eBay

faculty.chicagobooth.edu/matt.taddy/research

What is a Decision Tree?



Tree-logic uses a series of steps to come to a conclusion.

The trick is to have mini-decisions combine for good choices.

Each decision is a node, and the final prediction is a leaf node

Decision Trees are a Regression Model

You have inputs **x** (forecast, current conditions)
and an output of interest y (need for an umbrella).

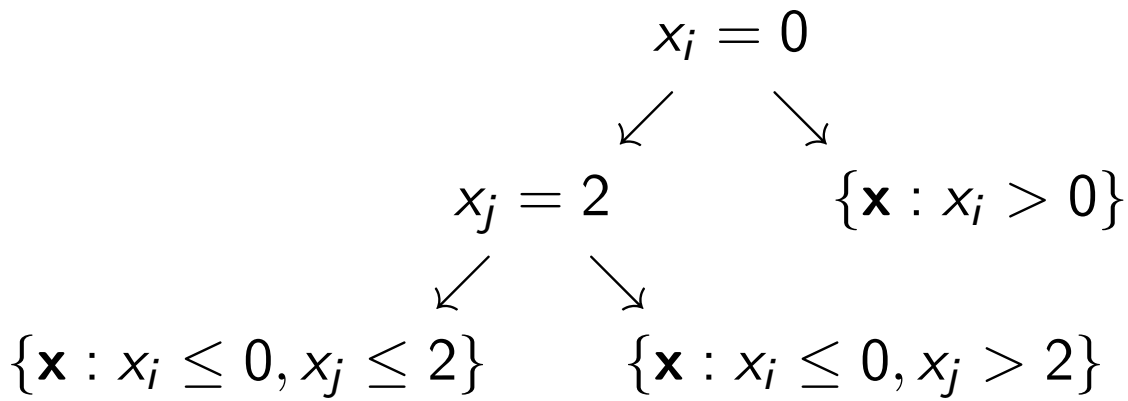
Based on previous data, the goal is to specify branches of choices that lead to good predictions in new scenarios.
In other words, you want to estimate a Tree Model.

Instead of linear coefficients, we need to find 'decision nodes':
split-rules defined via thresholds on some dimension of **x**.

Nodes have a parent-child structure: every node except the root has a parent, and every node except the leaves has two children.

Decision trees are like a game of mousetrap

You drop your **x** covariates in at the top, and each decision node bounces you either left or right. Finally, you end up in a leaf node which contains the data subset defined by these decisions (splits).



The prediction rule at each leaf (a class probability or predicted $\hat{y}$) is the average of the sample y values that ended up in that leaf.

Given a parent set of data $\{\mathbf{x}_i, y_i\}_{i=1}^n$, the optimal split is that location x_{ij} on some dimension j on some observation i,
so that the child sets

$$\text{left: } \{\mathbf{x}_k, y_k : \ x_{kj} \leq x_{ij}\} \ \text{ and } \ \text{right: } \{\mathbf{x}_k, y_k : \ x_{kj} > x_{ij}\}$$

are as homogeneous in response y as possible.

For example, we will minimize the sum of squared errors

$$\sum_{k \in \text{left}} (y_k - \bar{y}_{\text{left}})^2 + \sum_{k \in \text{right}} (y_k - \bar{y}_{\text{right}})^2$$

for *regression trees*, or gini impurity for *classification trees*
(e.g., the sum across children 'c' of $n_c \bar{y}_c (1 - \bar{y}_c)$ if $y \in \{0, 1\}$).

We estimate decision trees by being recursive and greedy

CART grows the tree through a sequence of splits:

- Given any set (node) of data, you can find the optimal split (the error minimizing split) and divide into two child sets.
- We then look at each child set, and again find the optimal split to divide it into two homogeneous subsets.
- The children become parents, and we look again for the optimal split on their new children (the grandchildren!).

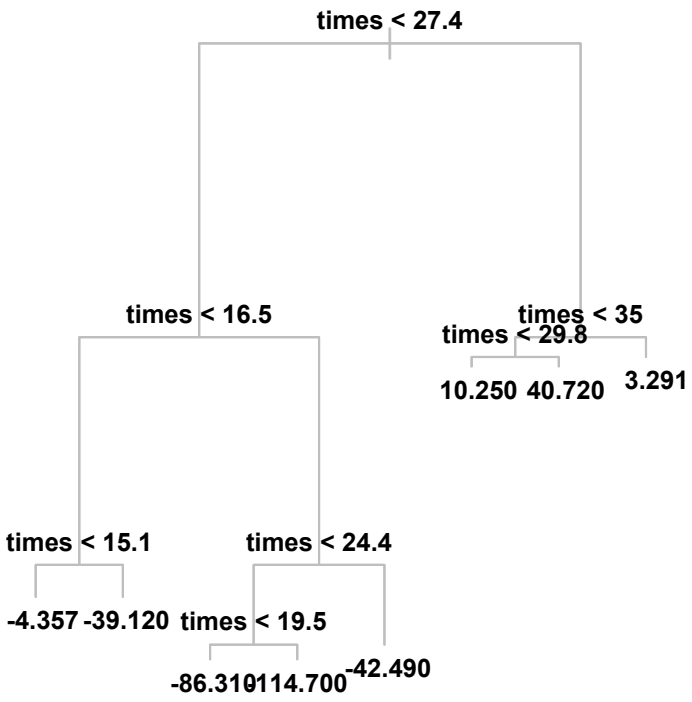
You stop splitting and growing when the size of the leaf nodes hits some minimum threshold (e.g., say no less than 10 obsv per leaf).

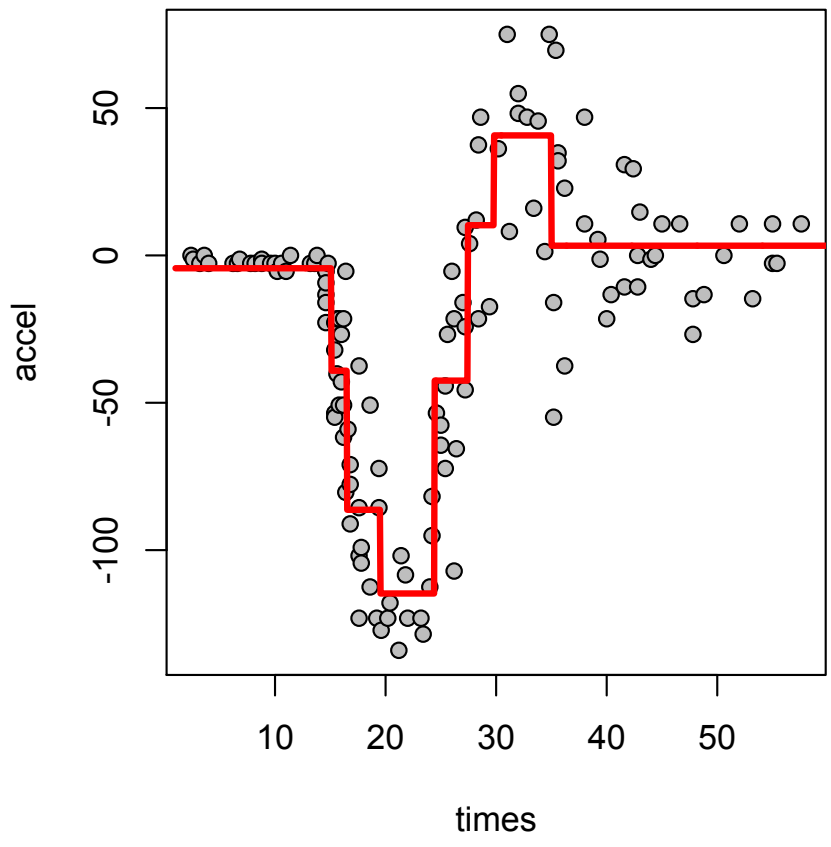
Trees are awesome

They automatically learn non-linear response functions and will discover interactions between variables.

Example: Motorcycle Crash Test Dummy Data

x is time from impact, y is acceleration on the helmet.





Unfortunately, it is tough to avoid overfit with CART:

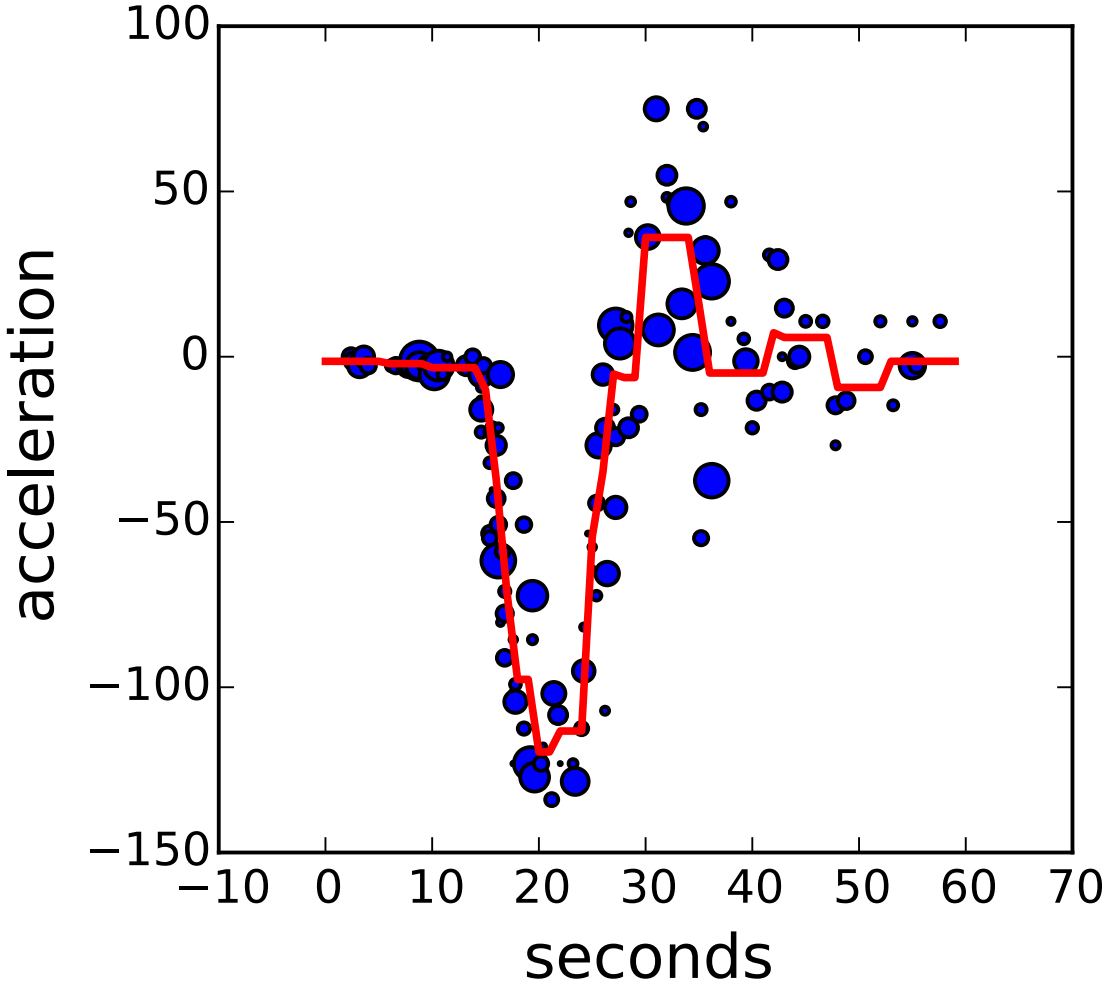
Deep tree structure is so unstable that optimal depth is not easily chosen via cross validation, and there's no theory to fall back on.

Instead, we can average over a bootstrapped sample of trees:

- repeatedly re-sample the data, with-replacement, to get a 'jittered' dataset of n observations.
- for each resample, fit a CART tree.
- when you want to predict y for some $\mathbf{x}$, take the average prediction from this forest of trees.

Real structure that persists across datasets shows up in the average. Noisy useless signals will average out to have no effect.

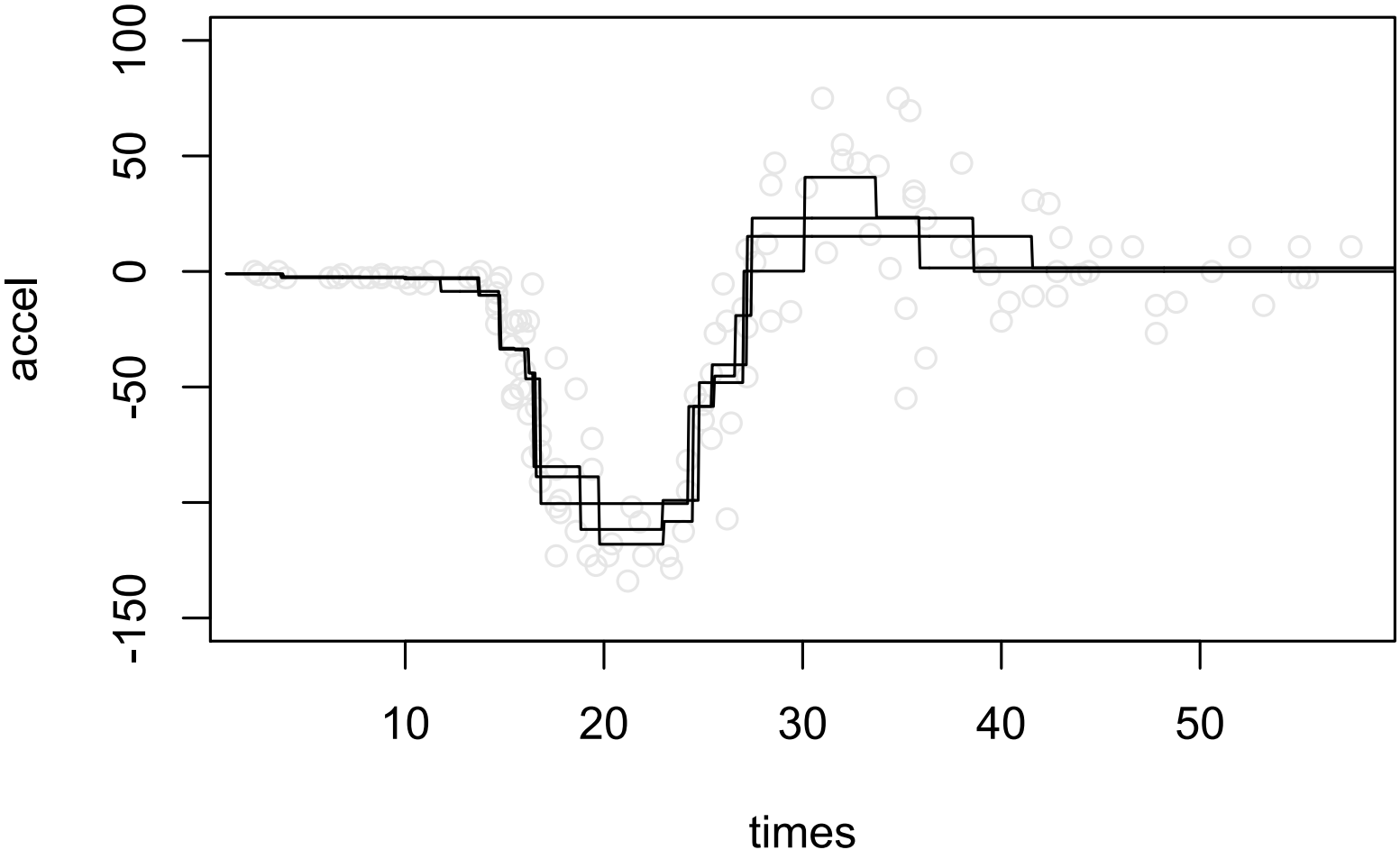
This is a Random Forest (RF)



Fitting CART to sampled-with-replacement data is equivalent to randomly weighting your observations in the deviance calculations.

(size proportional to weight in this picture).

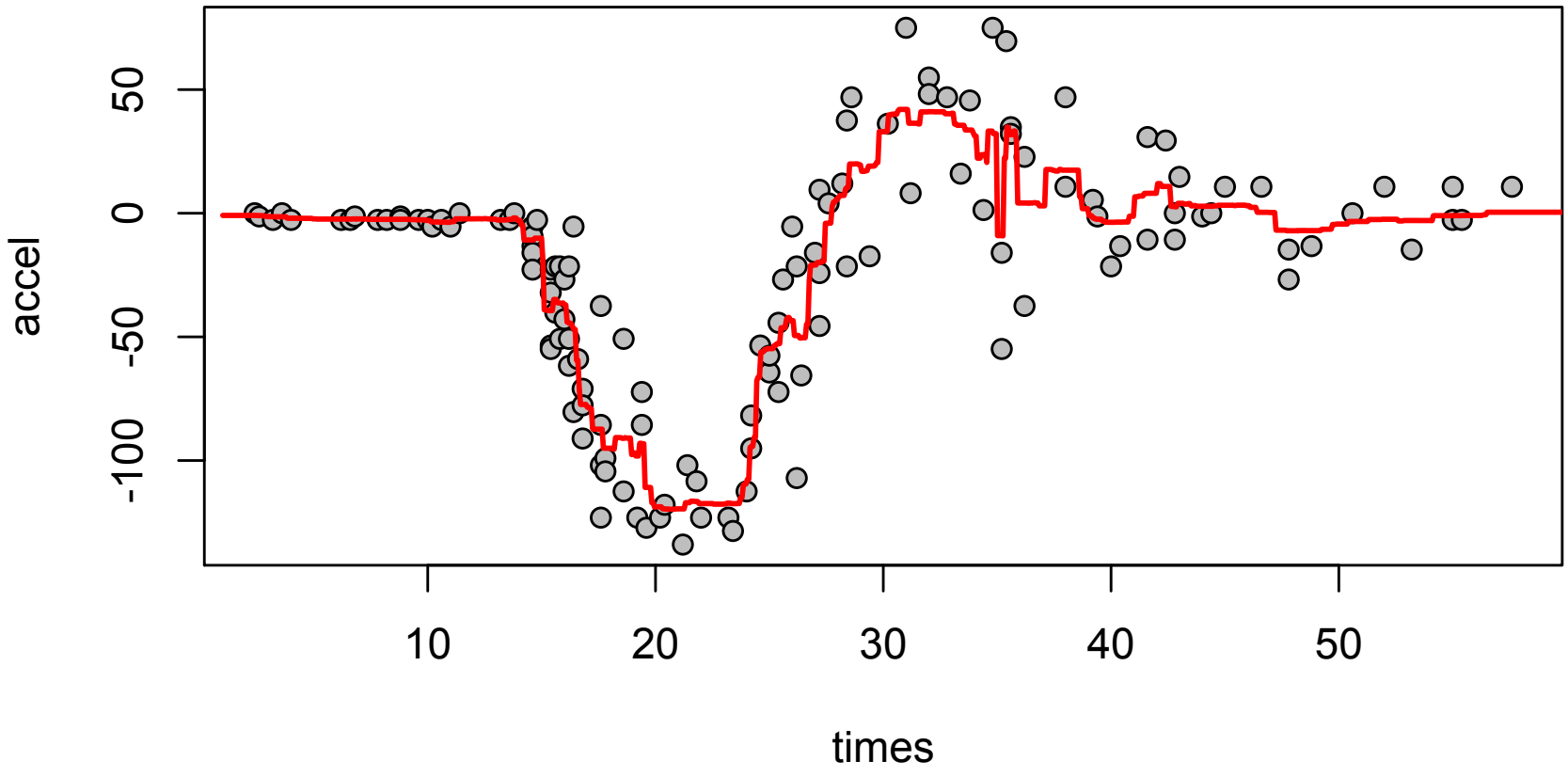
Random Trees for the Motorcycle Data



If you fit to random subsets of the data,

you get a slightly different tree each time.

Model Averaging with Random Forests



Averaging many trees yields a single [high quality] response surface.

Random Forests are really awesome

They have all the nice properties of trees, while avoiding overfit.

Unfortunately, search for *optimal splits* is expensive on Big data.
And if one tree is costly, then many is infeasible (you want many!).

Sub-Sampling Forests: for each tree, sample without-replacement only as many observations as are easy to fit a single tree.

Sadly, performance suffers: deep structure *needs* big samples.

Think about a rare data segment of 50 observations that all look and act the same (e.g., all users in some class commit fraud).

With 1/50 of the data, you'll never find this group.

Efficient Big Data analysis

To cut computation without hurting performance, we need to think about what portions of the tree are hard or easy to learn.

Once we figure this out, we can use a little bit of the data to learn the easy stuff and direct our full data at the hard stuff.

FWIW this is the future for Big Data.

Bayesian Forests

Our framework for thinking about easy vs hard learning derives Random Forests as a Bayesian posterior over the optimal tree.

Imagine you see all data and fit the best 'population' CART tree.

The Bayesian/Random Forest represents your uncertainty about this optimal tree. It is a sample of equally likely possible best trees.
(you are uncertain because you don't actually get to see all data.)

BFs and RFs are slightly different: a BF fits CART to randomly weighted data, and RF fits to randomly re-sampled data.
But you can think of an RF as an approximation to the BF.

Theoretical trunk stability

Given forests as a posterior, we can start talking about *variance*.

We are able to derive theoretically that the earliest structure in the tree – the trunk – should be very stable for large samples.

For the data at a given node, the probability that the optimal split on resampled data matches the true optimal split is

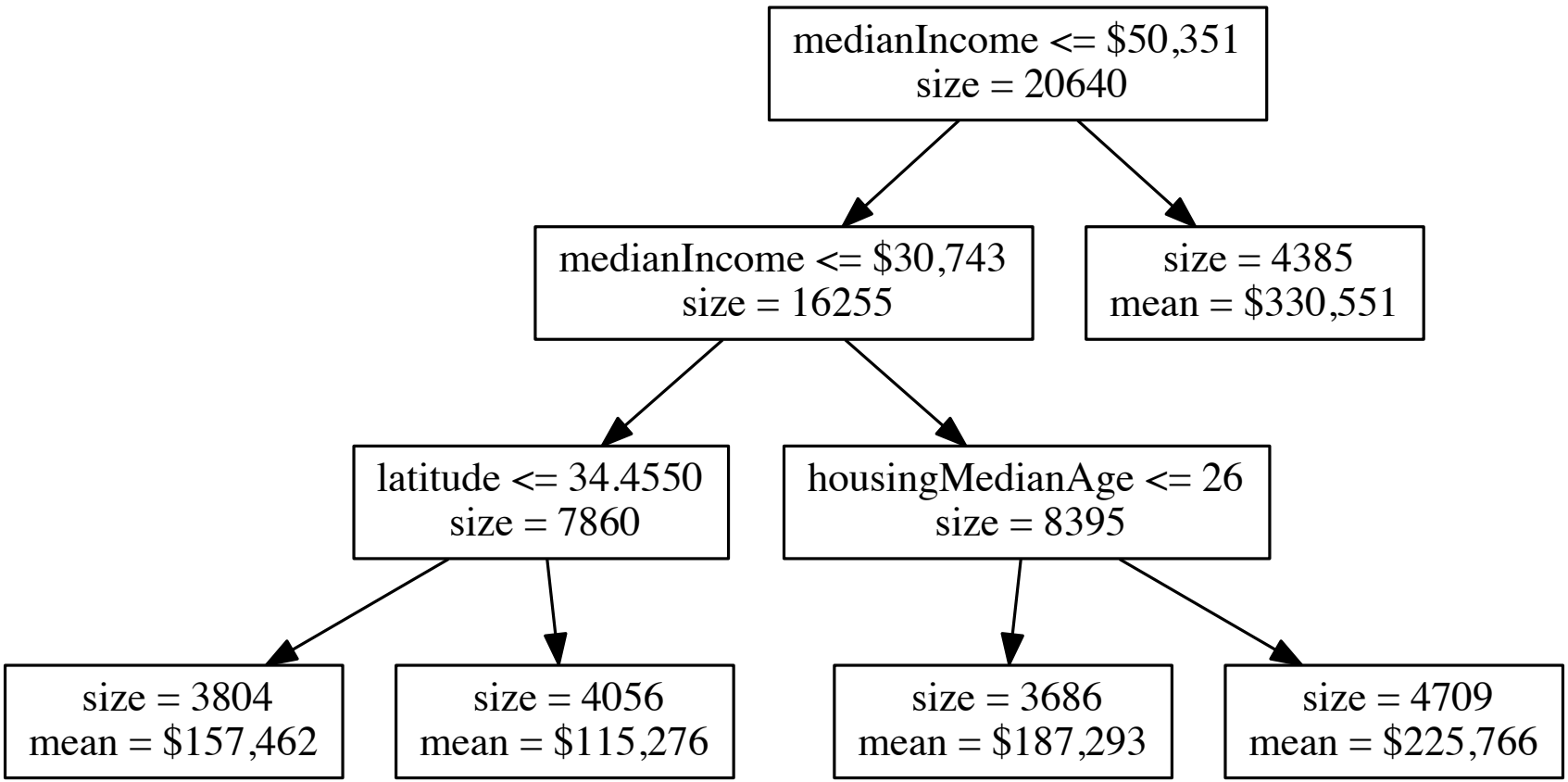
$$\mathrm{p}\left(\text{split matches sample CART}\right) \gtrsim 1 - \frac{p}{\sqrt{n}} e^{-n},$$

where p is the number of possible split locations and n the number of observations on the current node.

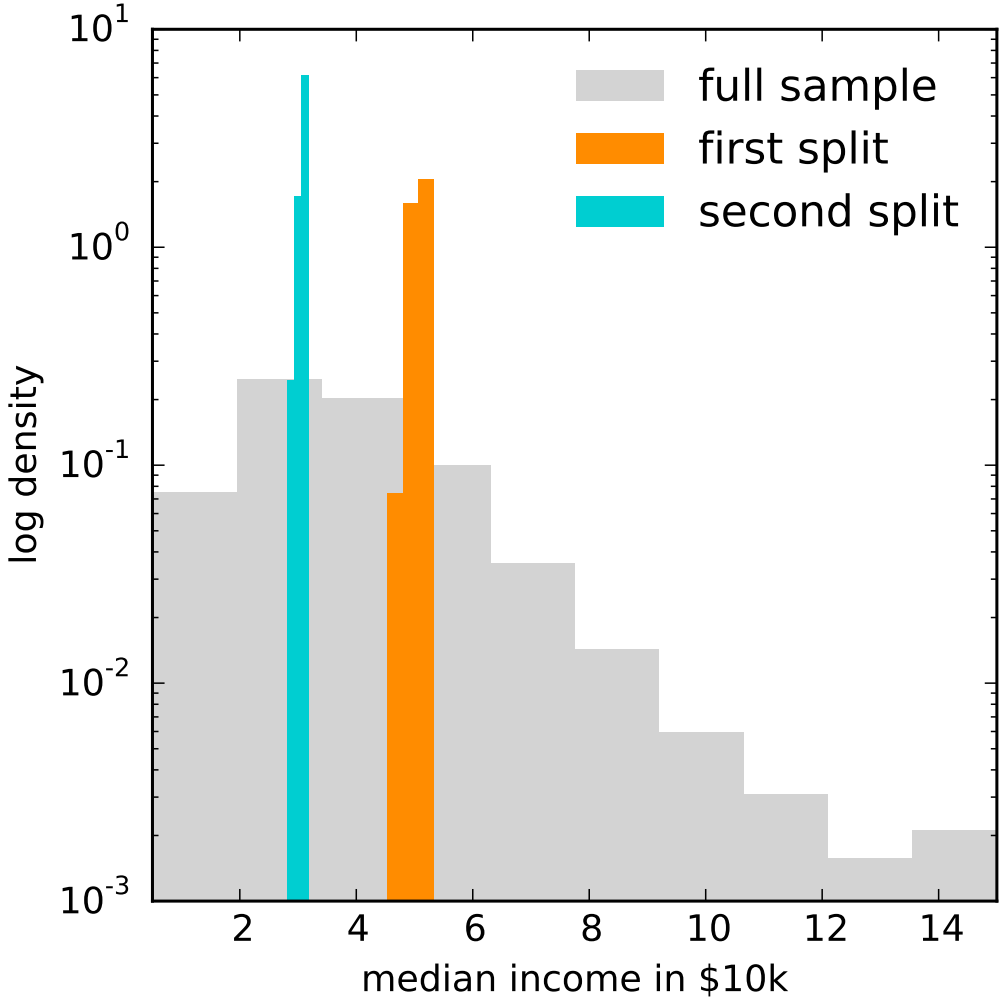
Things are pretty stable, until they aren't: as the tree grows, node sizes get smaller and chance of a non-optimal split multiplies.

California Housing Data

20k observations on median home prices in zip codes.



Above is the trunk you get setting min-leaf-size of 3500.



- sample tree occurs 62% of the time.
- 90% of trees split on income twice, and then latitude.
- 100% of trees have 1st 2 splits on median income.

Empirically and theoretically: trees are stable, at the trunk.

RFs are expensive when data is too big to fit in memory.

Subsampling forests lead to a big drop in performance.

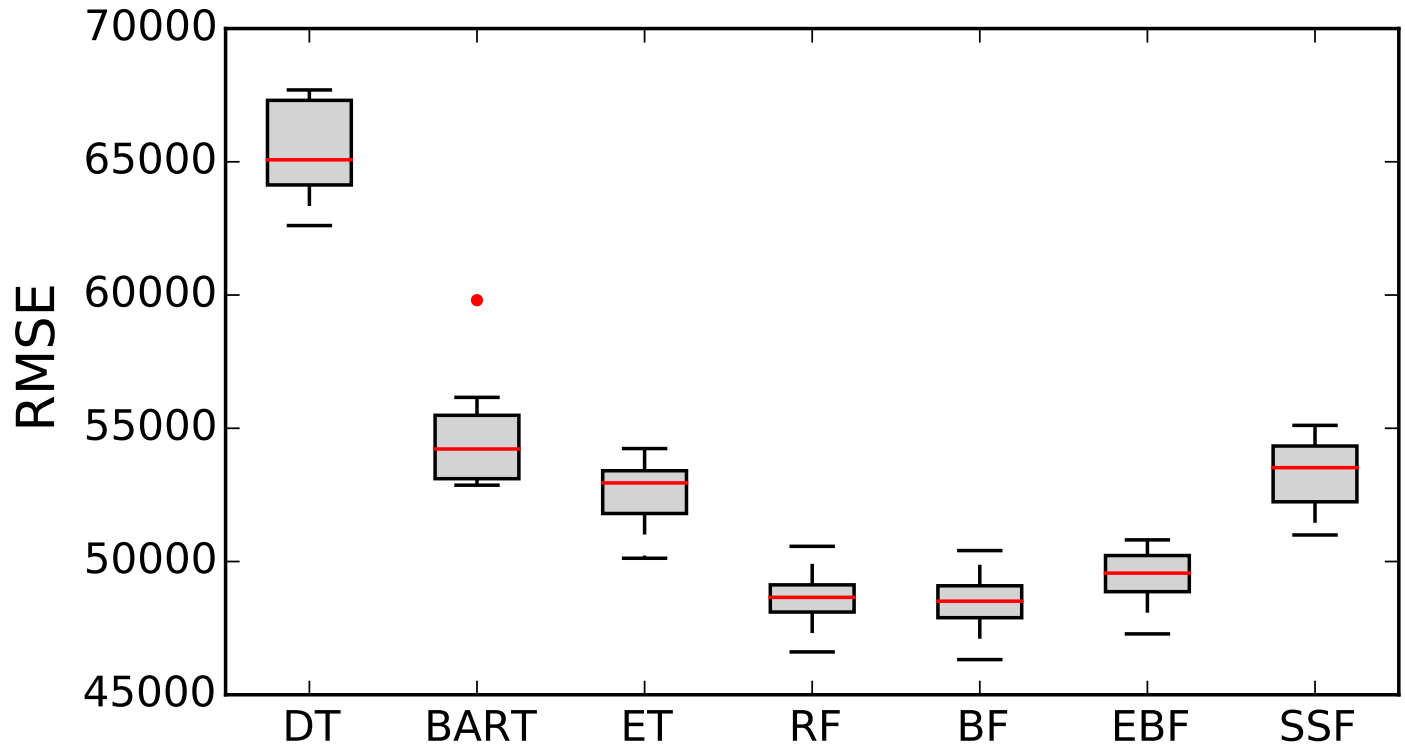
But wait: if the trunks are stable, can we just fit that once and then fit forests at each branch? Yes!

Empirical Bayesian Forests (EBF):

- fit a single tree to a shallow trunk.
- Use this as a mapper to direct full data to each branch.
- Fit a full forest on the smaller branch datasets.

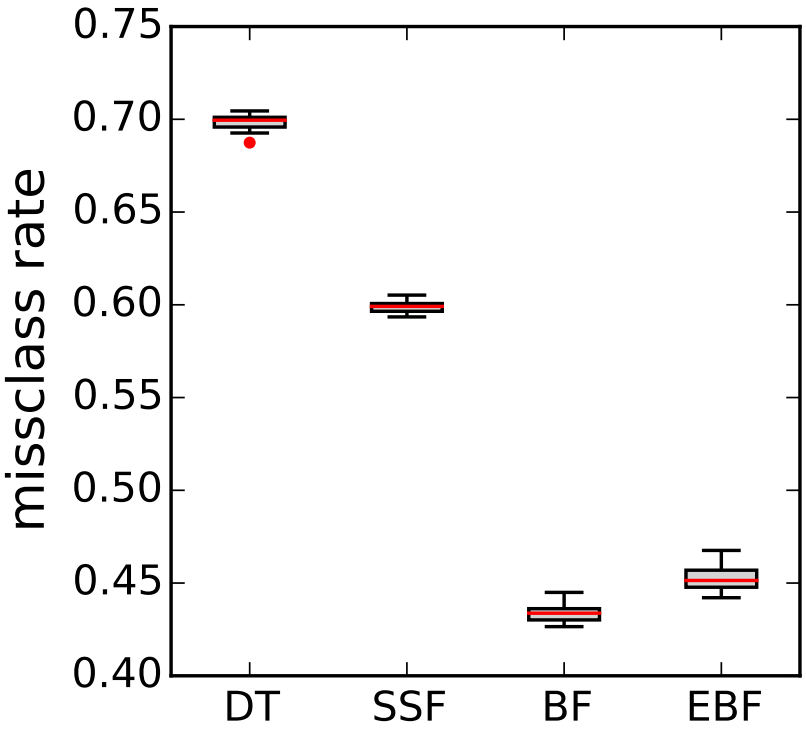
This is classic Empirical Bayes: fix higher levels in a *hierarchical model*, and direct your machinery+data at learning the hard bits.

Since the trunks are all the same for each tree in a full forest, our EBF looks nearly the same at a fraction of computational cost.



Here EBF and BF give nearly the same results. *SSF does not.*

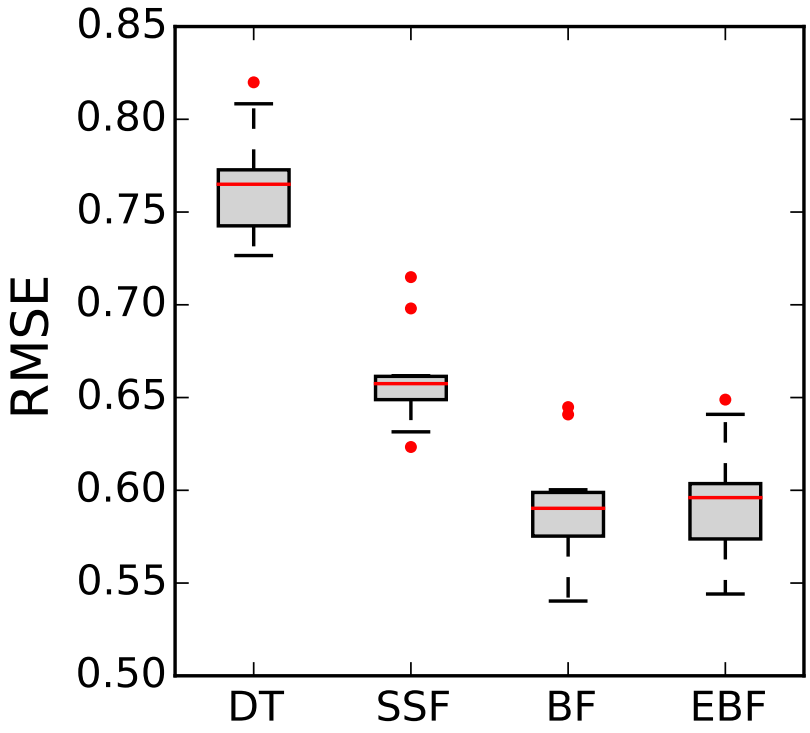
EBFs work all over the place



$\overline{\text{MCR}}$	% WTB	
0.4341	0.0	BF
0.4531	4.4	EBF
0.5989	38.0	SSF
0.6979	60.8	DT

Predicting beer choice from demographics

EBFs work all over the place



$\overline{\text{RMSE}}$	% WTB	
0.5905	0.0	BF
0.5953	0.8	EBF
0.6607	11.9	SSF
0.7648	29.5	DT

or wine rating from chemical profile

Choosing the trunk depth

Distributed computing perspective: fix only as deep as you must!

How big is each machine? Make that your branch size.

	CA housing			Wine			Beer		
Min Leaf Size in 10^3	6	3	1.5	2	1	0.5	20	10	5
% Worse Than Best	1.6	2.4	4.3	0.3	0.8	2.2	1.0	4.4	7.6

Still, open questions. e.g., more trees vs shallower trunk?

A key point: we do not think that EBFs are better than forests fit to all the data. But EBFs allow you to fit to much more data in less time without hurting performance too much.

Big Data axiom: more data beats fancy model.

EBFs at EBay: predicting Bad Buyer Experiences

A BBE could be receiving something that is 'significantly not as described', or shipping delays, or any of many other problems.

The models are updated frequently, and information about p(BBE) is an input to search rankings and more.

The best thing to improve predictions is more data.
With millions of daily transactions, there's little limit on data.

EBFs at EBay

Full random forest runs take too long on full data
(even using distributed tree algorithms).

Subsampling led to a noticeable and big drop in performance.

So: EBFs!

- trunk can be fit in distribution using Spark `MLLib`.
- this trunk acts as a sorting function to map observations to separate locations corresponding to each branch.
- Forests are then fit on a machine for each branch.

EBFs at EBay

On 12 million transactions, EBF with 32 branches yields a 1.3% drop in misclassification over the SSF alternatives.

This amounts to more than 20,000 extra detected BBE occurrences over this short time window.

Putting it into production requires some careful engineering, but this really is *a very simple algorithm*.

If you already fit RFs and are hitting time/space constraints, then an EBF is lots of gain for little pain.

The key to big data

Use plug-in estimates for the stuff that is easy to measure.

Partition conditional on these plug-ins.

Direct the full data towards the stuff that is tough to learn.

Thanks!