Empirical Bayesian Forests

Matt Taddy @ Chicago Booth

Chun-Sheng Chen, Jun Yu, and Mitch Wyle @ eBay Trust Science

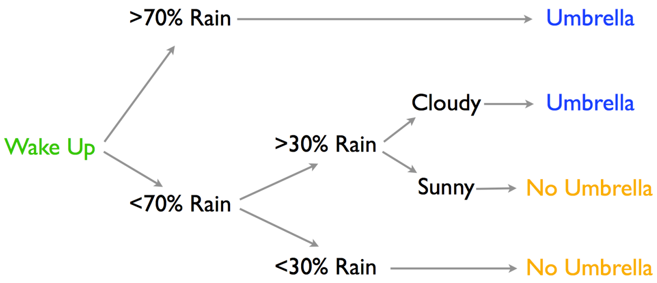
Outline

- Decision Tree and Random Forests
- Bayesian Forests
- Empirical Bayesian Forests
- Experimental Result

What is a Decision Tree?

Decision Tree (a.k.a CART[1]) is a predictive model, which maps from observations ($\mathbf{X}$) to a conclusion (Y) using a series of rules.

Example: Predict whether or not to bring an umbrella given the forecast and the current condition.



[1]CART: Classification And Regression Trees

How to decide the variables to split on?

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 $\mathbf{x}$.

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

How to construct the decision tree?

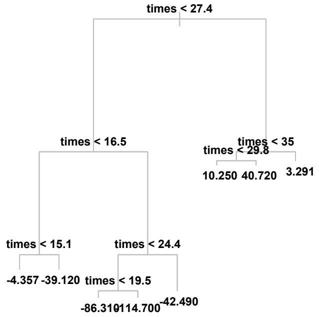
CART grows the tree through a sequence of splits:

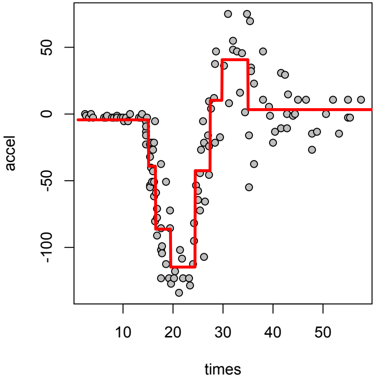
- Given the data in a node, find the optimal split (the error minimizing split) and divide data into two "child" nodes.
- The same process is applied to each "child" node (i.e. it is a recursive procedure).
- Splitting stops when CART detects no further gain can be made, or some pre-set stopping criterions (the size of the leaf nodes hits some minimum threshold) are met.

Nice properties of decision trees

Decision trees automatically learn **non-linear** response functions and will discover **interactions** between variables.

Example: Motorcycle Crash Test Data where x is time from impact, y is acceleration on the helmet.





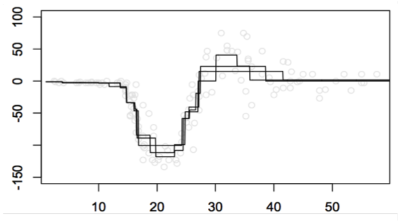
Random Forests

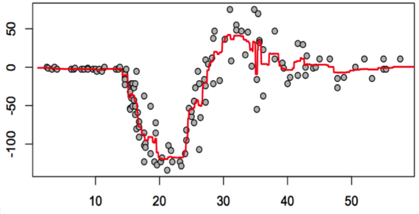
Unfortunately, CART has high variance and can overfit because deep tree structure is unstable.

Instead, we can construct an ensemble of decision trees (Random Forests):

- Bootstrap the original dataset by sampling with replacement.
- For each bootstrapped data, fit a CART tree.
- For a testing data point x, take the average prediction from this forest of trees.

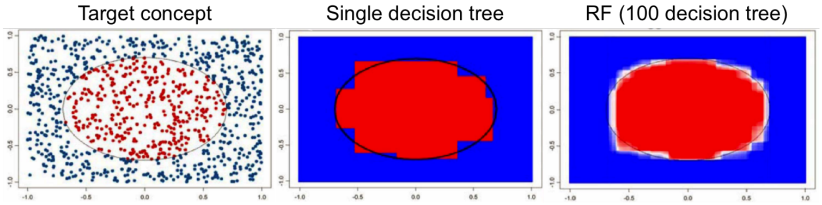
Random Trees for the Motorcycle Data





- If you fit to random subsets of the data, you get a slightly different tree each time.
- Averaging many trees yields a single [high quality] response surface.

Random Forests help



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

Random Forests are awesome, BUT ...

- Search for *optimal splits* is expensive on Big data.
- Sub-Sampling Forests hurts the performance because deep structure *needs* big samples.

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.

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

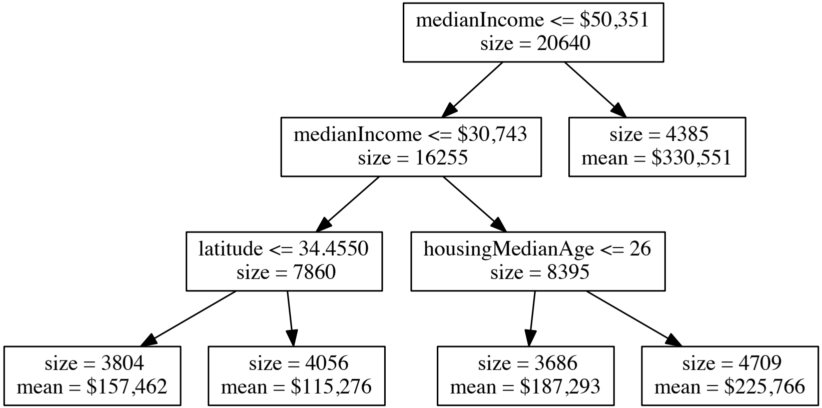
$$\mathrm{p}\left(\text{split matches sample CART}\right) \gtrsim 1 - \frac{p}{\sqrt{n}} e^{-n},$$

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.

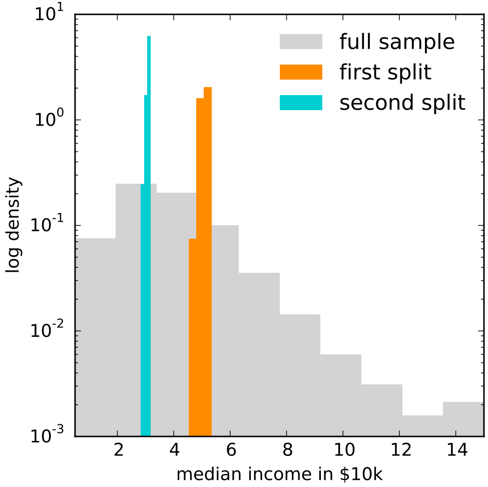
California Housing Data

20k observations on median home prices in zip codes.



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

California Housing Data



- 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.

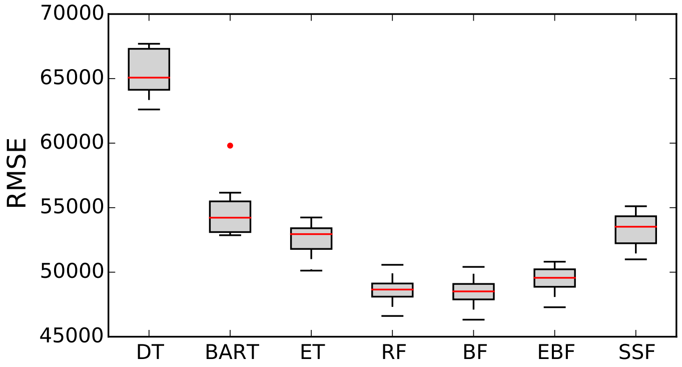
But wait: if the trunks are stable, can we just fit that once and then fit forests at each branch? Yes!

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.

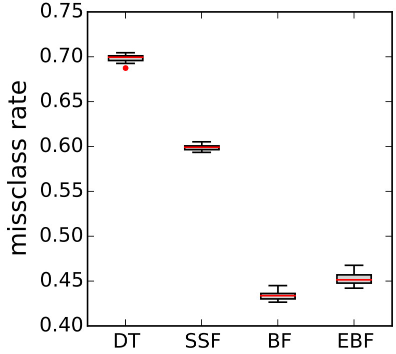
This is classic Empirical Bayes: fix higher levels in a *hierarchical model*, and direct your machinery+data at learning the hard bits.

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.*

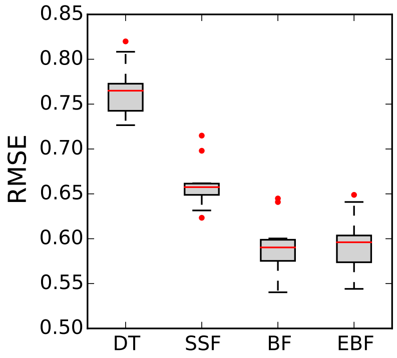
EBFs work all over the place



$\overline{\text{MCR}}$	% 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 Defect / Bad Buyer Experiences

Defect/BBE include various complains from eBay buyers regarding a transaction, including item not received, significantly not as described, negative feedback, shipping delays, and etc.

The defect model learns from seller's historical behaviors, and predicts the likelihood of a seller causing defects on eBay. Defect model score is fed an input to search rankings and demote sellers more likely to cause defects.

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

EBFs at eBay: predicting Defect / Bad Buyer Experiences

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: predicting Defect / Bad Buyer Experiences

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.

Thanks!

Q & A