E pluribus, veritum: An Introduction to Ensemble Models via Random Forests

Renzo (Ren) Massari (they / them)
2025-01-08

E pluribus, veritum: An Introduction to Ensemble Models via Random Forests

Source: http://www.publicdomainpictures.net/pictures/30000/velka/tree-1332664495LMO.jpg

Renzo (Ren) Massari (they / them)
2024-01-11

E pluribus, veritum: An Introduction to Ensemble Models via Random Forests

Source: http://www.publicdomainpictures.net/pictures/30000/velka/tree-1332664495LMO.jpg

Renzo (Ren) Massari (they / them)
2024-01-11

E pluribus, veritum: An Introduction to Ensemble Models via Random Forests

Source: http://freebigpictures.com/wp-content/uploads/2009/09/forest-path.jpg

Renzo (Ren) Massari (they / them)
2024-01-11

Purpose of this chat:

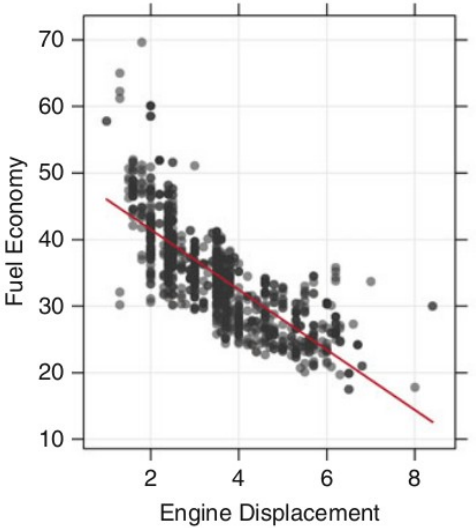
- Get to know random forests as a tool
- Like all tools, they have strengths and weaknesses
- Get a sense of when to use them and when to try something else
- It’s usually a good first-approach to data when you don’t have too much background to impose more structured models on it; a favorite for data mining and for multiple imputation of missing data
- But keep in mind the mantra of statistical learning: there is no one tool to rule them all

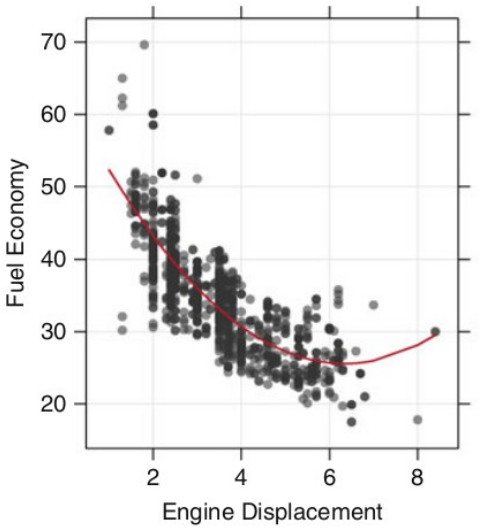
Things I assume you “know”:

- Linear models of this kind $Y = \mathrm{E}(Y|X_1,\ldots,X_p) + \varepsilon = \beta_0 + \sum_{j=1}^{p} X_j\beta_j + \varepsilon$
  - Regression
  - Classification
- K-nearest neighbors (KNN)
- Cross validation (CV)
- Overfitting
- Bootstrapping
- The relationship between the variance (and sd) of a distribution and that of the sampling distribution of its mean: $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$

(Binary) Decision trees:

- They are linear models, though they don't assume linearity
- Linear models are wonderful things (and they don't assume linearity)
- You probably know them from OLS or logit models (linear-ish)





Source: Kuhn and Johnson, Applied predictive modeling

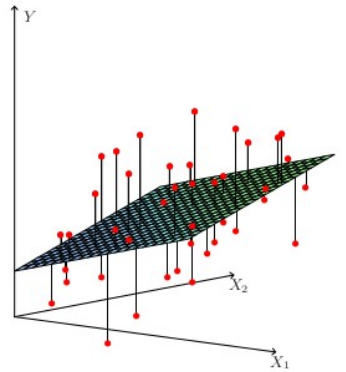


FIGURE 3.1. *Linear least squares fitting with $X \in \mathbb{R}^2$. We seek the linear function of X that minimizes the sum of squared residuals from Y.*

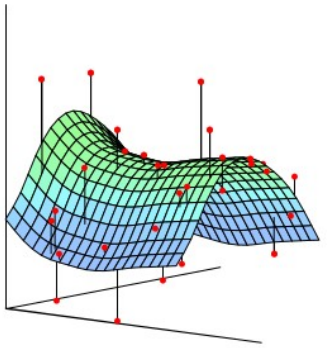
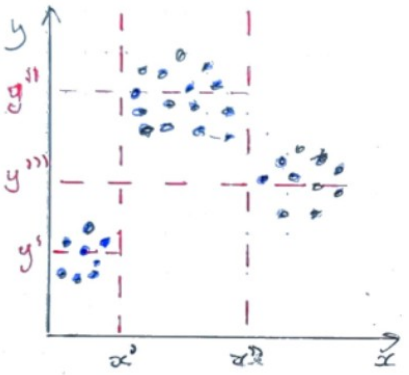


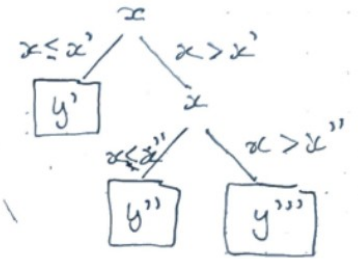
FIGURE 2.10. *Least squares fitting of a function of two inputs. The parameters of $f_\theta(x)$ are chosen so as to minimize the sum-of-squared vertical errors.*

Source: Hastie et al, ESL II

(Binary) Decision trees:

They are linear models





Tree!

Rule 1: $x \leq x' \Rightarrow y'$

2: $x > x' \;\&\; x \leq x'' \Rightarrow y''$

3 $x > x' \;\&\; x > x''' \Rightarrow y'''$

Redundancy

Rules-based model

$$d_1 = \begin{cases} 1 & \text{if } x \leq x' \\ 0 & \text{o.w} \end{cases}$$

$$d_2 = \begin{cases} 1 & \text{if } x > x' \;\&\; x \leq x'' \\ 0 & \text{o.w.} \end{cases}$$

$$d_3 = \begin{cases} 1 & \text{if } x > x'' \\ 0 & \text{o.w} \end{cases}$$

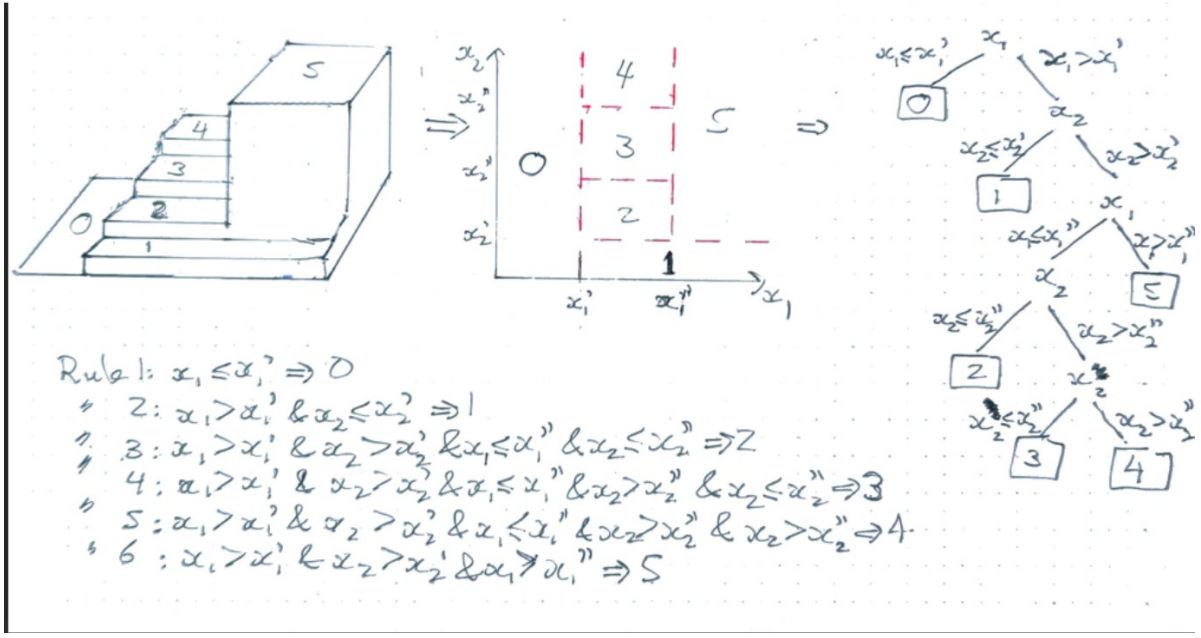
$$\rightarrow \quad y = y' d_1 + y'' d_2 + y''' d_3 + \varepsilon$$

homoscedasticity

Linear model

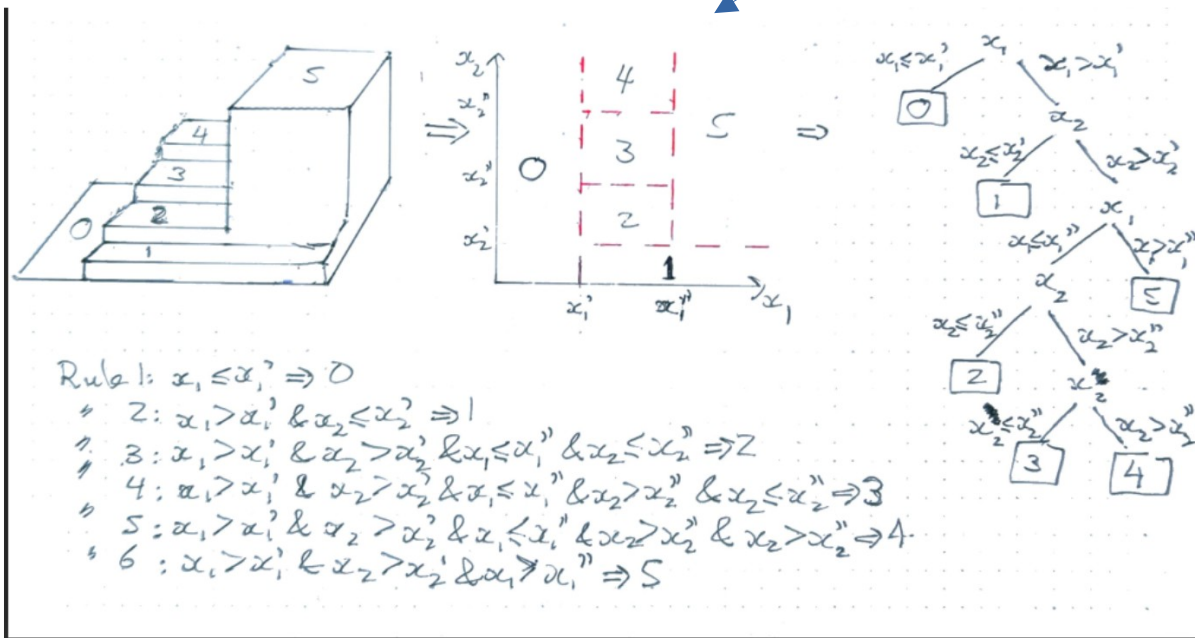
Decision trees:

Any number of dimensions… K-dimensional rectangles!



Decision trees:

Any number of dimensions… K-dimensional rectangles!



Can you see how, if you squint, this looks like KNN, but with rectangles?

Decision trees:

- But where do we stop? Overfitting is a thing :(
- In a linear model, you could go all the way to assigning one dummy to each observation; you will fit the training data perfectly… but it will be useless for other data

Nick	1
Richard	2
Roger	3
David	4
Syd	5

- Your rule now sets a dummy equal to 1 if name is “Nick” and your “estimated parameter” will be 1, etc…
- But try predicting anything when you face John, Paul, Ringo, and George, huh?

Decision trees:

- We can replicate this with OLS, so why are we doing this?
- It’s the algorithm and the learning it does for you
  - Imagine you have not one or two, but K predictors (and K is large). Can you visually inspect this K+1 dimensional graph and find the cutoff points for each variable in the right sequence
  - More generally, imagine you have no clue as to which predictors to include in your linear model and where to introduce “cuts” (the rules for dummy variable creation)
- Pick your algorithm based on considerations like your theoretical understanding of how the data was generated (plus practical considerations of the dataset like dimensions and patterns of missingness and many etc.)
- If you know what variables you need to use and how to bin them or where to introduce changes in slope or intercept or some such, use OLS; if you’re more agnostic, start with a decision tree? Data doesn’t “speak”… but you can let it express itself!

A rough guide to CART

- The Classification And Regression Tree (CART) algorithm is the OG of decision tree algorithms, introduced by Breiman et al (1984) (*)

- This is just one of the algorithms you can use to grow your tree (to train it, but we are nerds who like to stay metaphorically consistent)

- The algorithm must decide:

    1) Which variable to split on
    2) At what value of that variable to split on
    3) How deep to grow the tree (I know, growing downwards)
    4) What to use a a predictor in the terminal node

(*) Breiman, Friedman, Olshen, and Stone (1984), “Classification and regression trees,” Chapman and Hall, NY

Growing the tree!

1) Which variable to split on
2) At what value of that variable to split on

Say this is a regression problem and we want to minimize the MSE. We start here:

$$SSE = \sum_{i \in S} (y_i - \bar{y})^2$$

For each of your K variables:
- Grid them, if continuous; decide how to group them / rank them if categorical (there IS data engineering; we can talk about it at the end if there's time)
- Pick the one variable and split such that: $\min_{S_1, S_2} SSE = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$ across all values of all K variables
- Take one of those two splits and repeat the exercise within it, ie:

$$SSE = \sum_{i \in S} (y_i - \bar{y})^2 \leftarrow \text{mean as prediction}$$

$$\min_{S_1, S_2} SSE = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

$$\downarrow$$

$$\min_{S_{1,1}, S_{1,2} \mid S_1, S_2} SSE = \sum_{i \in S_{1,1}} (y_i - \bar{y}_{1,1})^2 + \sum_{i \in S_{1,2}} (y_i - \bar{y}_{1,2})^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

(*) Breiman, Friedman, Olshen, and Stone (1984), "Classification and regression trees," Chapman and Hall, NY

Growing the tree!

3) How deep to grow the tree

- One way is to set a minimum number of observations at each “leaf” (terminal node) (sounds akin to one of those “do not use regression if you have less than 20 observations”)

- Combined with the more mathy approach of **cost-complexity tuning**

  - You grow the tree as far as it goes (given min number of obs at leaves)…
  - Then you prune back (yes, “prunning”: love your metaphors or you’re in the wrong place)
  - You do this by penalizing your SSE with the size of the tree:

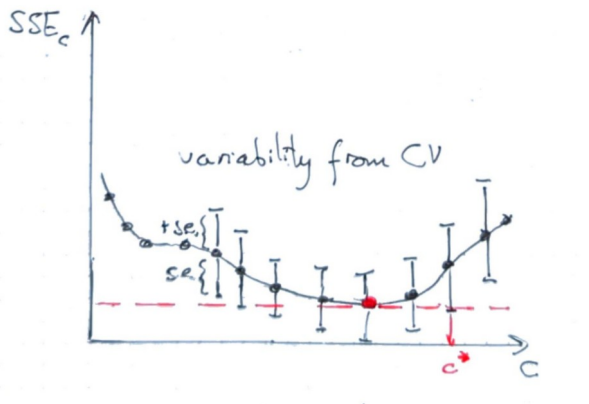
$$SSE_c = SSE + c * (\text{number of terminal nodes})$$

  - How do you choose that c (the cost complexity parameter)?
  - You could just go with the tree that results in the smallest SSE_c; or...

(*) Breiman, Friedman, Olshen, and Stone (1984), “Classification and regression trees,” Chapman and Hall, NY

Growing the tree!

Breiman et al (1984) suggest using CV and the one standard-error rule (pick the smallest tree that contains the smallest cost-complexity-corrected SSE within one standard error of its own cost-complexity-corrected SSE):



A note on c:

- A lower value of c leads to larger models
- Larger models are at a higher risk of overfitting
- They are also harder to interpret

(*) Breiman, Friedman, Olshen, and Stone (1984), “Classification and regression trees,” Chapman and Hall, NY

Growing the tree!

4) What to use a a predictor in the terminal node

- So far, we've used the mean of all the values in the leaves
- You can use the median, or an OLS linear model (smoothing)
- Or anything you can think of. In principle, you could have different predictors at different nodes.
- We will return to this… at the end, if we have time.

(*) Breiman, Friedman, Olshen, and Stone (1984), "Classification and regression trees," Chapman and Hall, NY

We love decision trees!

- They’re quick to estimate (if you have a lot of parallelism)
- They’re so easy to interpret
- They automate feature selection
- They’re good with missing data (**surrogate splits**, we can talk later)
- They’re good with sparse, skewed, continuous, categorical…
- They do NOT assume linearity (even though they are linear models: they do not assume a linear relation between the X and the Y, but can be expressed as linear models of rules to Y)

But trees, omg, not so good, you trees!

- Model instability (perturb the data a bit and you can get a very different tree) = **HIGH VARIANCE**

- Not so good predictive performance:
  - Are hyper-rectangles the correct way to model your data?
  - Finite number of predicted outcomes if using something like the mean (one value per terminal node)
  - Large range of values in each of the leaves (unless you overfit a lot by having tiny leaves)
    - Gives large SSE (in other words, since we're just using the mean of that large range, probably bad at predicting more extreme values)
  - Selection bias: continuous predictors are preferred over more "granular" ("chunky") predictors
    - This becomes a problem specially if your less-granular predictors are noisier

- Interpretability (and stability) are affected negatively by correlation between predictors
  - Choice of split becomes unstable
  - More predictors may be selected than necessary
  - Their individual importance is divided between the correlated predictors (think t-test vs F-test)

WHAT CAN WE DO TO MAKE THINGS BETTER?!?!?!

Hint:
E pluribus, veritum

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

BAGGINS!

Bootstrap AGGregation (Bagging):

- An ensemble of models (not just trees) created by bootstrapping:
  - Train your model on a lot of bootstrapped samples, then get the average prediction of them all
  - And the average prediction's s.e. drops by the square root of the number of predictions!
  - So a good technique to use with unstable models (like trees)

- Applies to regression and classification, developed by Breiman (1996)

- In the case of trees: do not prune your models

- Bagging allows a sort of CV "for free": test on "out-of-bag" samples (those not included in the bootstrapped sample)
  - Each trained model will have its own performance metric
  - Performance of averaged predictions correlates with what's observed from CV and from test-sets

Breiman (1996), "Bagging Predictors," Machine Learning, 24(2), 123–140

Bagging:

- How many bootstraps? (Hint: think of formula for s.e. of average!)
  - Benefits decrease rapidly with number of bootstraps
    - Suggestion is to start around 10
    - Then keep adding bootstrapped samples while you see improvements in performance
    - If by about 50 bootstraps, you don’t see improvements, maybe don’t use trees (or, before abandoning them, try a different ensemble technique)
  - Pretty fast thanks to parallel computing
  - Understanding:
    - Interpretability is pretty low
    - But variable importance can be explored (eg. add reductions in SSE for each variable)

But Bagging!

- Not random enough because of correlation between trees!
  - $\sigma_{\bar{x}}^2 = \rho\sigma_x^2 + (1 - \rho)(\sigma_x^2 / n)$ ← The second term disappears with growing n; but unless rho, the correlation between bagged trees, is zero, the first term remains.
- All models use the same predictors, so some tree-structure carries across all trees, leading to potential overfitting (think very large samples from an infinite population)
- So how do we de-correlate trees?

Glad you asked!

Random Forests:

- Proposed by (guess)… Breiman (2001)
- Select m bootstrapped samples, just like with bagging…
- But train them in a special way:
  - At each split in each tree, select k < K predictors
- That's it; from then on, you continue as with bagging:
  - do not prune your trees
  - average your predictions

Breiman (2001), "Random Forests," Machine Learning, 45, 5–32

A little bit of random forestry:

- Tuning parameters: m (bootstrapped samples) and k (how many variables at each split)
- Breiman (2001) suggests k = K/3; probably better: grid a few values between 2 and K and tune
- m >= 1000 (these trees are not correlated so you can keep improving with little risk of overfitting!)

- The average of the predictions is a linear combination of many independent learners
  - Strong learners
  - Complex learners
  - Low-bias learners
- Random forests tend to be robust to noise in responses because of the drop in the s.e. due to averaging (but may underfit non-noisy responses)

- Estimation costs: each tree uses only some predictors, so cheaper than bagging with the same m… but random forests will use a much larger m

Breiman (2001), “Random Forests,” Machine Learning, 45, 5–32

Understanding predictors in ensembles:

- An idea by… Breiman (2000) allows to measure importance (but not relationship):
  - Randomly permute the values of each predictor (one predictor at a time) for each tree's out-of-bag sample
  - Record the difference in performance between the permuted and not permuted predictions for each predictor and tree
  - Aggregate these differences across forest

- More recently, Shapley (1953) values, derived from cooperative game theory
  - Shapley values assign payouts to players as a function of their contribution to the total payout of a game (an incentive mechanism to encourage cooperation)
  - Here, features are the players of the game and the performance of the predictor is the payout
  - Beyond the scope of this talk, but look it up!

Breiman (2000), "Randomizing Outputs to Increase Prediction Accuracy," Machine Learning, 40, 229–242
Shapley (1953), "A value for n-person games," Contributions to the Theory of Games 2.28: 307-317

Issues with understanding the importance of predictors in random forests:

- Same issues of bias and dilution-through-correlation as found in decision trees
  - Bias against granular predictors
  - m has an effect on apparent importance in the presence of correlation
  - Correlated but uninformative predictors can appear to be important and reduce the apparent importance of the correlated and informative predictors

Bonus points: Boosting

- An offshoot of learning theory; not just for decision trees
- So far, we have used strong learners, which risk overfitting
- Instead, let’s switch to weak predictors
  - Weak predictors are those that predict only a little better than random
  - By combining them in an ensemble, we **boost** them and get better predictive performance than even ensembles of strong predictors!
  - It performs similarly to random forests in many instances. Why one or the other?

Boosting

- The idea didn’t find a good implementation until the AdaBoost algorithm came along:

  1) Choose a tree depth and the number of iterations
  2) Choose an initial predictor: the mean of y!
  3) Repeat the number of iterations:
     1) Find residuals u = true value y - prediction from previous iteration
     2) Fit residuals u using a tree of chosen depth
     3) Predict the value of u of each observation in the sample using the tree
     4) Update your predicted values of y by adding the previous iteration’s predicted value (of y) to the current prediction (of u)

- Individual observations can carry individual weights, adjusted each time, so that harder-to-predict cases end up having a greater presence through the iterations
- Each iteration prediction can also carry an evolving weight, with the weight a function of its performance, so that stronger iterations carry more weight

Boosting

- Now, we use XGBoost (at least, last time I checked)
  - Similar in “spirit,” but uses Newton-Raphson (2nd order approx) instead of gradient descent
  - Adds some tricks to improve performance
  - Packages:
    - scikit-learn for Python
    - caret for R

Other topics:

- (Unordered) Categorical predictors:
  - separate dummies for each category or aggregations? 2^{k-1}-1 possible combinations (where k is the number of categories)
  - One way: ordering them on mean of y (regression) or majority vote (classification)

- Other forms of data-engineering?

- Missing data: surrogate splits!
  - Say variable X_i splits the tree in a given node given the split-criteria applied to observations where it is not missing; however, it is missing from some other observations.
  - Look for another variable that most closely imitates the split at that point; that will be the first surrogate split variable.
  - Repeat this so you have an ordered set of a few surrogates at each split
  - Now split your training, test, and any future datasets following that ordering in that split if you come across observations with a missing value for X_i

- Regression model trees and smoothing

- Rules-based models