Heterogeneous treatment effect

Han Zhang

Outline

Heterogeneous treatment effect by subgroups

- Broadly speaking, heterogeneous treatment effect (HTE) just means that treatment effect varies
- For cluster-randomized experiment, Neyman estimator naturally gives heterogeneous treatment effect for each subgroup (e.g., each school)

Heterogeneous treatment effect by subgroups

- Regression estimator:
  - Fixed effect regression gives an overall ATE, but does not estimate heterogeneous treatment effect for each group
  - To estimate group-specific ATE, we fit linear regression with interactions between group dummy and treatment D (and no fixed effects)
  - (Interaction coefficients + the coefficient on treatment) captures the treatment effect for each subgroup
    - But this interaction model does not give us overall ATE

Heterogeneous treatment effect by propensity score

- The second type of treatment effect heterogeneity is effect difference by treatment propensity score: $p(D = 1|X = x)$
- That is, we are comparing treatment effects among units that are more likely to be treated, and units that are less likely to be treated.

Jennie E. Brand and Yu Xie, *Who Benefits Most from College?: Evidence for Negative Selection in Heterogeneous Economic Returns to Higher Education*, American Sociological Review **75** (2010), no. 2, 273–302

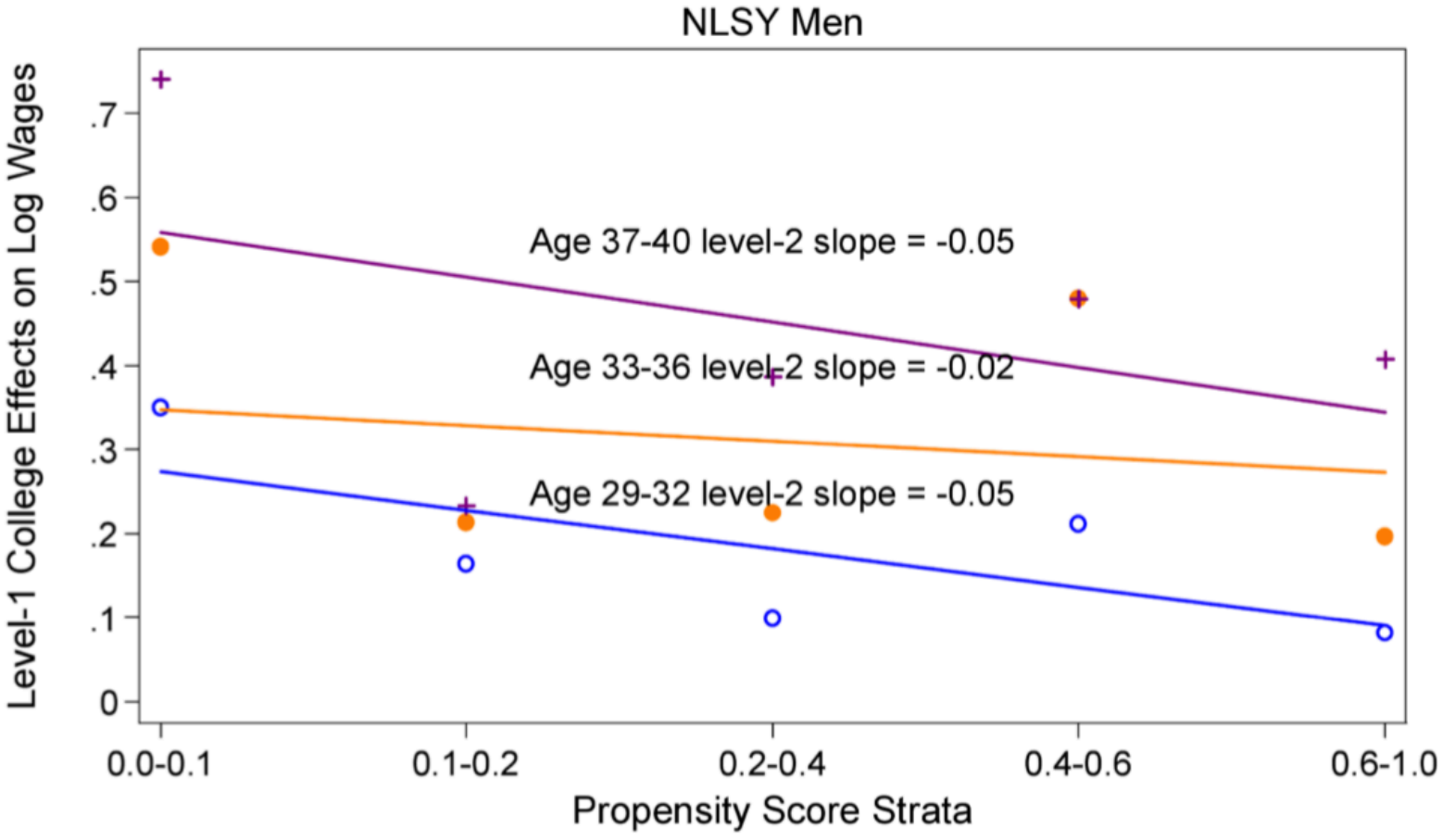
Heterogeneous treatment effect by propensity score

- Treatment: college education
- Outcome: future earnings
- HTE: corresponding to two different hypothesis
  - individuals who are most likely to select into college benefit the most from college
    - Treatment effect is high among units whose treatment propensity $P(D = 1|X = x)$ is high
  - individuals who are least likely to obtain a college education benefit the most from college
    - Treatment effect is high among units whose treatment propensity $P(D = 1|X = x)$ is low

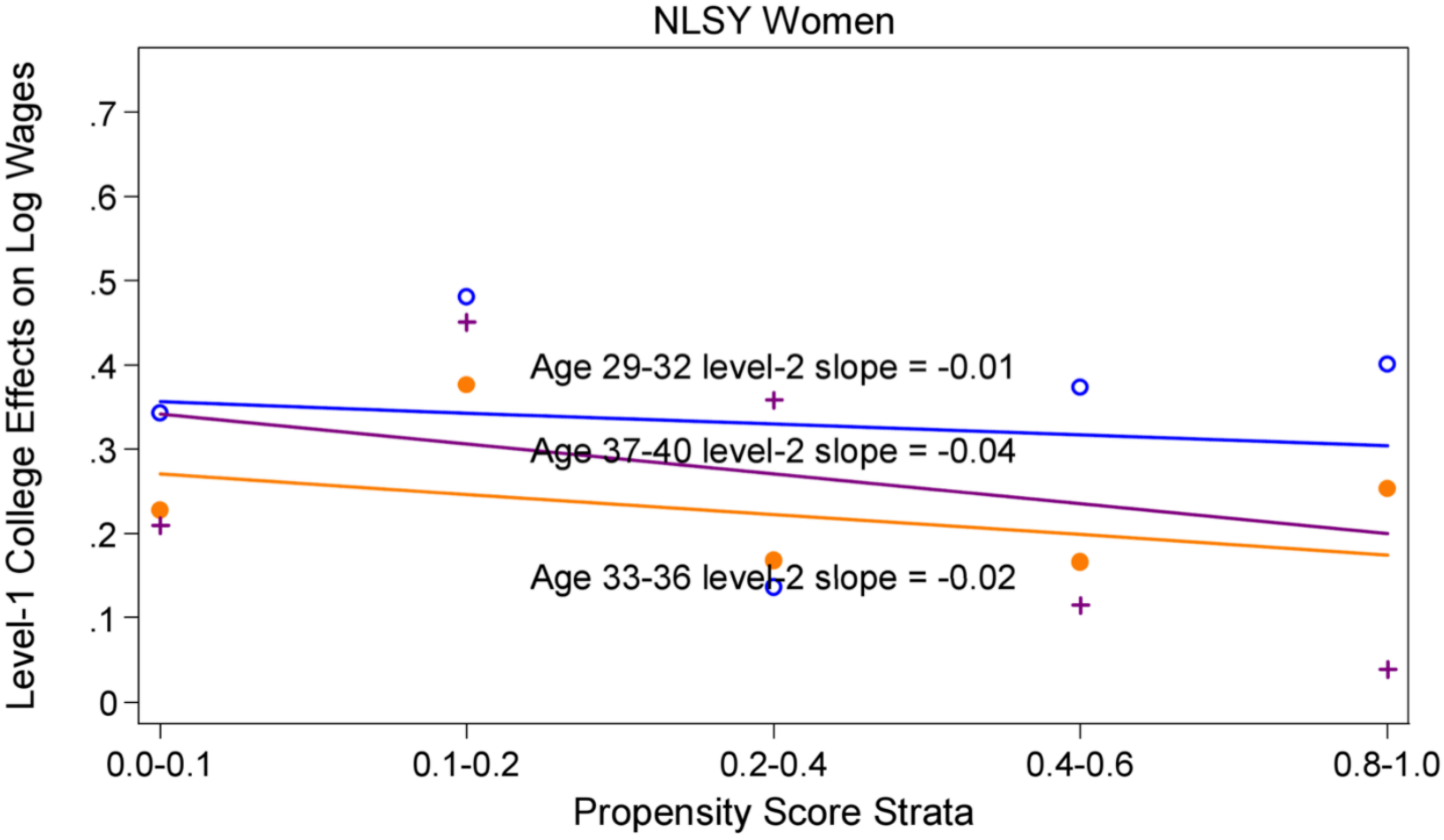
Heterogeneous treatment effect by propensity score

- Divide respondents into groups based on their propensity score
- Use regression to estimate treatment effect for each group
- Plot the effects by propensity score
- Caution: this article is not a randomized experiment; it uses a survey dataset
  - rigorously what it estimates is not ATE, unless we add strong assumptions
  - More on the assumptions next week
  - I am using it to illustrate the idea of HTE

Heterogeneous treatment effect by propensity score



Heterogeneous treatment effect by propensity score



Heterogeneous treatment effect by covariates

- The third, and probably the most common type of HTE you will see in research articles: treatment heterogeneity by covariates
- Formal notation: $\tau(x) = E(Y^1 - Y^0|X = x)$
- $\tau(x)$ is usually referred as conditional average treatment effect (CATE)
- In Project STAR example:
  - We think the important variation of treatment heterogeneity come from whether teacher is more experienced or not

Using interaction model to capture CATE

- To capture heterogeneity by covariate (CATE), the easiest approach is to add interaction between treatment and covariate:

$$Y_{ij} = \alpha_j + D_{ij}\rho + \text{experienced}_i\beta + D_{ij}\text{experienced}_i\gamma + \epsilon_{ij}$$

- γ and ρ together captures the treatment effect heterogeneity
  - ρ: *CATE* for classes with inexperienced teacher
  - $\gamma + \rho$: *CATE* for treated classes with experienced teacher

Using interaction model to capture CATE: shortcomings

- Using interaction model to capture CATE is the dominant approach in applied literature now
- The interaction model assumes constant effect within covariate levels
  - It slightly relax the overall constant effect assumption, but still can be unrealistic
  - Read Hainmueller, Mummolo and Xu (2019) for how to see if this assumption is correct

Using interaction model to capture CATE

- If the covariate is beyond binary and has more levels, or it is continuous variable, we are implicitly adding another assumption: linear interaction effect
- That is, CATE grows linearly in the covariate
- Is this realistic? certainly not.
- If time permits, at the end of semester, we can talk some advances of estimating heterogeneous treatment effects using machine learning
- The basic idea is to predict each individual's counterfactual outcomes
- So the need to split population into subgroups by propensity score or by covariates is decreased

Problems

- Use theory to guide you to add some interaction terms and obtain CATE
- This approach may miss important treatment effect heterogeneity
  - There may be heterogeneity for another variable, but you did not add its interaction into regression
  - Or, there are higher-order interactions
    - like it requires combinations of three covariates to have a causal effect
  - Or if the relationship cannot be captured by a linear model (nonlinear relationship)
    - If the relationship is U-shaped, then interaction term may not significant

Worse: P-hacking

- Say if you have specified an X and Y
- And you have 20 control variables
- Search one by one, and see which control variable has a significant interaction term
- And invent a theory about that variable.
- `StataOneClick`
- Problem:

Motivating example: strong interactions

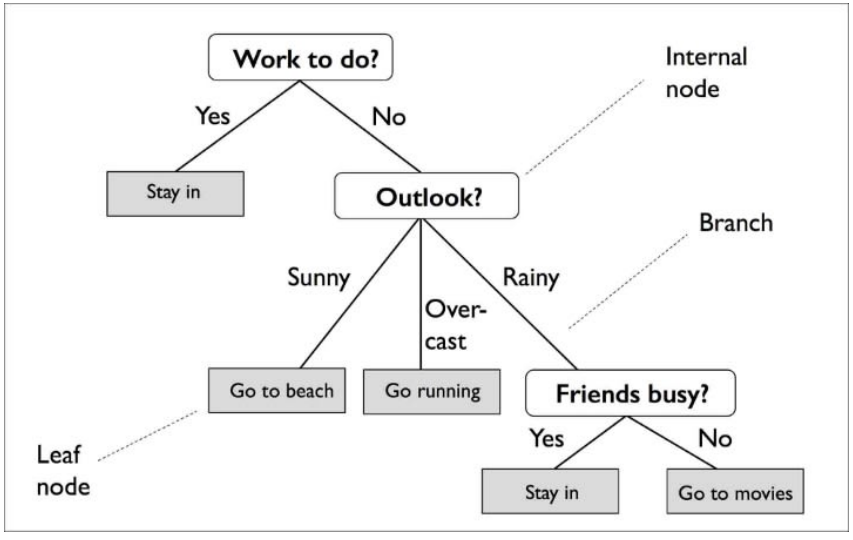
- Let us assume that we have a dataset with 1000 observations and 20 continuous predictors
- Say, each pair of the 20 predictors can have an interaction effect
- That means $20 + \binom{20}{2} = 190$ parameters
- What about we assume triple interactions between each triplet of predictors?
- $20 + \binom{20}{2} + \binom{20}{3} = 1350$ parameters
- OLS estimator cannot find a unique solution

Decision Tree

- Decision tree is a simple method to handle complex data with lots of interactions
  - classification tree: binary/categorical outcomes
  - regression tree: continuous outcomes
- A different name: CART:
  - Classification And Regression Tree

Decision Tree Example

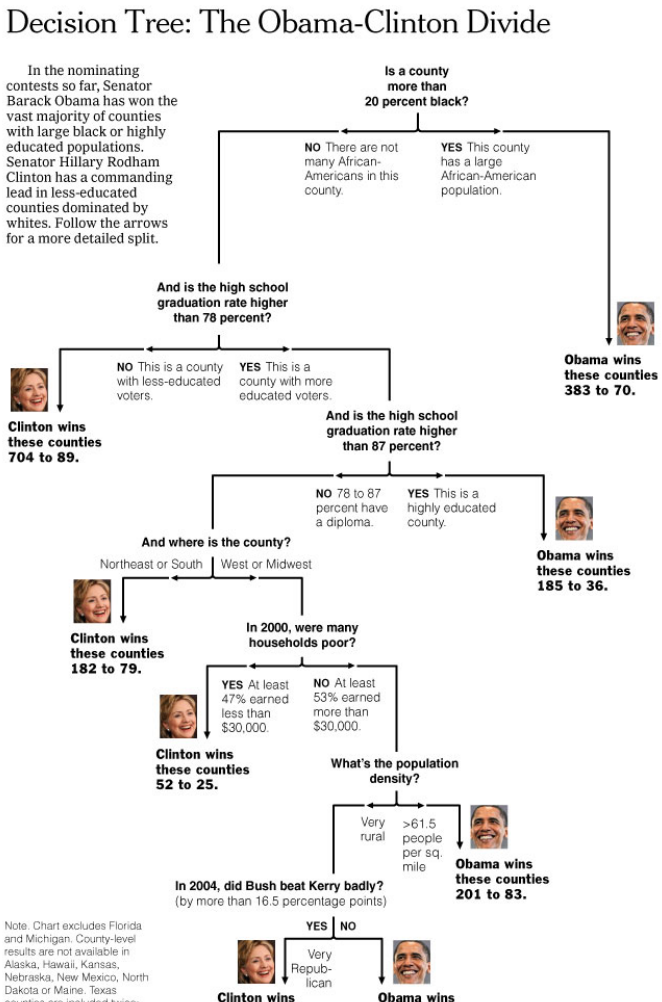
- Intuitively, decision tree visualizes one's sequential decisions process (Y), based on some predictors



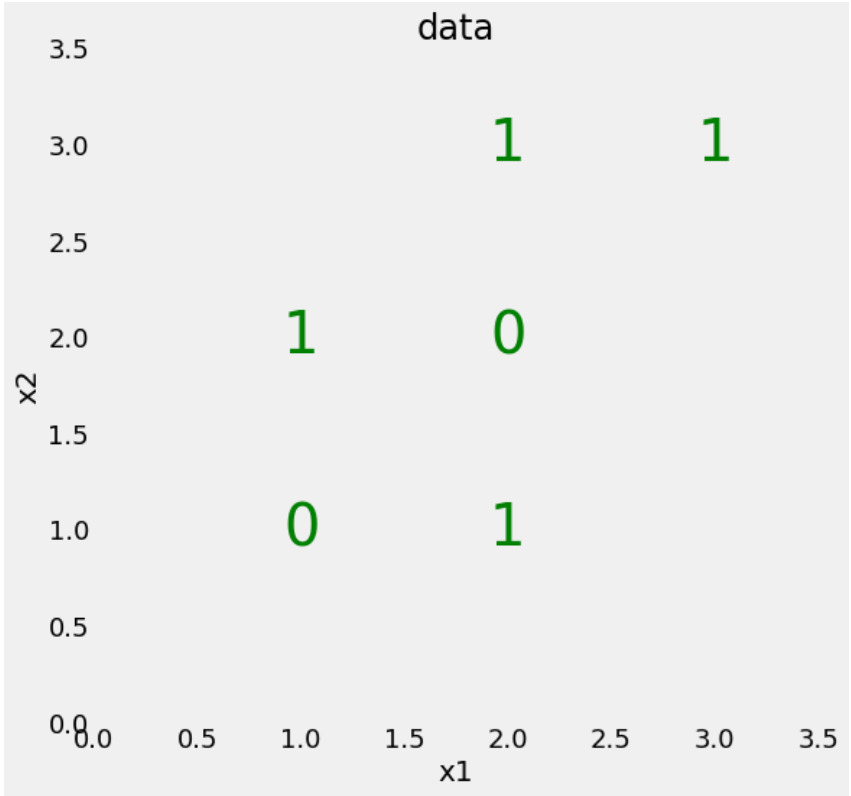
- Decisions Y are located at leaves
- The rightmost branch has a triple interaction

Decision Tree Example

https://archive.nytimes.com/www.nytimes.com/imagepages/2008/04/16/us/20080416_OBAMA_GRAPHIC.html?emc=polb1&nl=pol



Growing a Tree



- The above data cannot easily be separated by drawing a straight line (i.e., simplest linear regression)
- Let us draw a tree by ourselves to distinguish $Y = 0$ vs. $Y = 1$, based on X_1 and X_2
  - We want a binary tree: split into two branches

Some principles

- If you want to teach computers to draw a tree, here are some principles people usually follow:
- There are usually multiple ways to grow a tree. How do we pick the order we draw branches?
  - We should select a variable that best split data into two groups
  - In other words, it predicts the outcomes the best
- Machine is silly; we force it to be binary tree: each time split into two branches.
  - This means that for categorical and continuous X, we force them to make binary splits
- What if we there are multiple outcomes on a same leaf?
  - For continuous outcomes, the prediction is the mean
  - For categorical outcomes, the prediction is the mode
- One predictor can be used multiple times

Desision Tree Algorithms

- Decision Tree Algorithms help you to draw a tree from more complex data
- What are the steps we should take
- Let us first work with continuous outcome Y: regression tree and J continuous predictors X_1 to X_J
- There are two questions to consider:
  - Which X_j to choose first?
  - We will split X_j into $X_j < s$ and $X_j \geq s$. How do we choose s?
- And the intuitive answer is that:
  - You choose choose X_j and s that best separates Y (thus predicts Y the best)

Decision Tree Algorithms: formal math

- If we write this intuition down mathematically:
- We have p predictors: $X_1, \cdots, X_p$
- For each predictor X_j , calculate its minimum MSE:
  - Consider all it possible cutoffs s. A particular cutoff s will split the data into two regions:

$$R_1(j,s) = \{X|X_j < s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\} \quad (1)$$

  - We should select a s that minimizes the MSE

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \quad (2)$$

  - $\hat{y}_{R_1}$ is the mean response for the training observations in region $R_1(j,s)$
- Finally, select X_j and its s that yields the smallest MSE

Some details

- The above procedure tells you how to make the first split
- We repeat this procedure to grow sub-trees for each branch the procedure is the same, expect that you only use observations that belong to this branch to calculate MSE
  - This is known as *greedy recursive binary splitting*
  - It is possible that a particular choice of X_j may not be the current best, but may turn out to be the best choice in the future (locally best vs. globally best)
  - We do not consider the above possibility; just *greedily* choose the best partition and do not look back
- Decision Tree is a non-parametric model
  - Intuitively, whenever we grow a new branch, we are adding more interactions, using regression analogy.
  - But we do not write down parameters explicitly
  - MLE is not applicable

Classification Tree

- Very similar to regression trees.
- With one exception: we use cross-entropy loss instead of MSE
  - Also called log-loss or entropy loss
- For binary classification:

$$-\sum_{i=1}^{N} y_i \cdot \log P(\hat{y}_i = 1) + (1 - y_i) \cdot \log\left(1 - P(\hat{y}_i = 1)\right)$$

Pros and Cons of Decision Trees

- Pros:
  - easy to understand with visualization
  - easy to recognize which feature is more important (top of a tree)
  - handles complex interactive data with ease
  - handles categorical variables easily: no need for dummies
- Cons:
  - If your data are not that complex, tree easily overfit
  - It's not a parametric model: cannot summarize with several parameters (like linear regression do)
  - Confidence interval?
    - There are no explicit parameters, so surely no confidence intervals for parameters
    - The first analytical results is by Wager, Hastie and Efron (2014); JMLR

From One Tree to Many Trees

- Bagging tree (or ensemble of trees): averaging the predictions of many trees
  1. From the original training data, draw a sample with replacement of equal size
  2. Fit a tree for each sample
  3. Repeate 1 and 2 for some times
- Take the mean of estimates of each tree to produce a single estimate for each test data point

Random Forest

- Random Forests further extend the idea of bagging
- The key innovation of random forests:
- For each sample from the original training data, randomly select m variables (not using all p variables), and grow a tree;
  - A common choice: $m = \sqrt{p}$
- In other words, we just force $p - m$ predictors to be non-relevant each time
- Why? avoid over fitting

Boosting trees

- Bagging tree and Random Forest create many trees and average them together
  - Each of the tree is independent of the others
- A different idea is to create a sequence of trees that gradually improve over each other

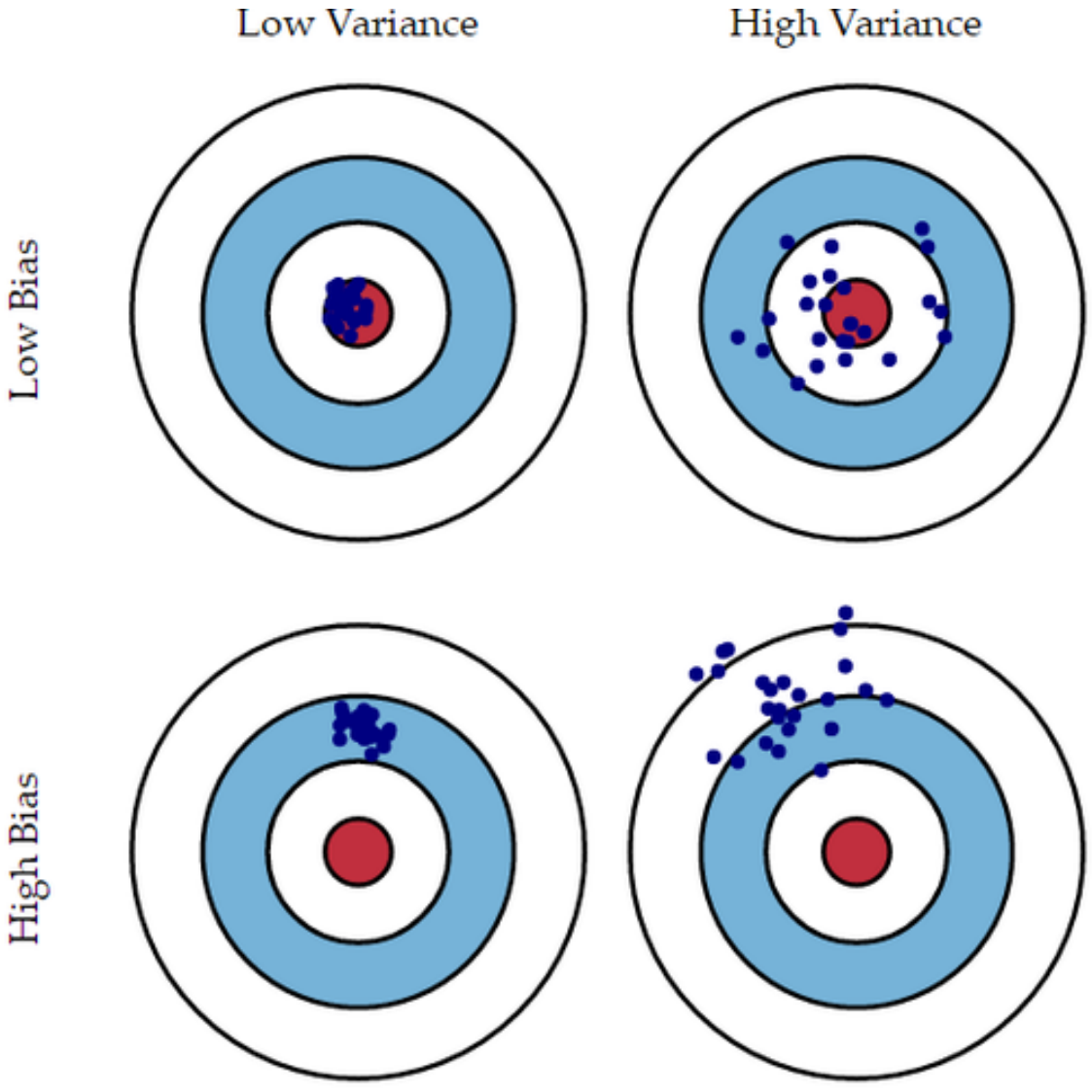
Boosting trees

- Assume you first fit a decision tree $G_1(X)$
- Bagging trees and random forests: fit another decision $G_2(X)$, totally independent of G_1
  - Then final prediction $Y = \frac{G_1(X)+G_2(X)}{2}$
- Boosting trees: find $G_2(X)$ based on prediction error of the first tree
  1. Learn a second tree $G_2(X)$ to predict $Y - G_1(X)$
  2. Then final prediction $Y = G_1(X) + G_2(X)$
  3. Repeat Step 1 and 2: learn a new tree $G_3(X) = Y - G_1(X) - G_2(X)$, and so on.
- Essentially, boosting trees find data points that previous algorithms are most likely to be wrong, and improve the algorithm on these points.

Boosting trees vs Random Forests

- You man hear may different variants of trees
  - AdaBoost is the first and Gradient Boosting Tree is the most successful
- Gradient Boosting Tree (GBT) and Random Forests (RF) are typically the two best prediction methods you can get
  - GBT typically works well then the dimension is not that high
  - RF works well when the dimension is very high

Bias Variance Trade-Off



Decision Tree: Bias vs Variance

- The tree grown using the above procedure can be quite complex
- We can always make a very complex tree by:
  - Try your best to make every single leaf contains only one Y
- Bias-variance trade-off:
  - Complex trees fit the data nearly perfect (low predictor MSE)
  - But has high predictor variance
    - Each time you have a new observation, the tree may look entire different
- We need to regularize to make tree simpler
  - regularize = avoid overfitting
  - explicitly make a very complex model less complex

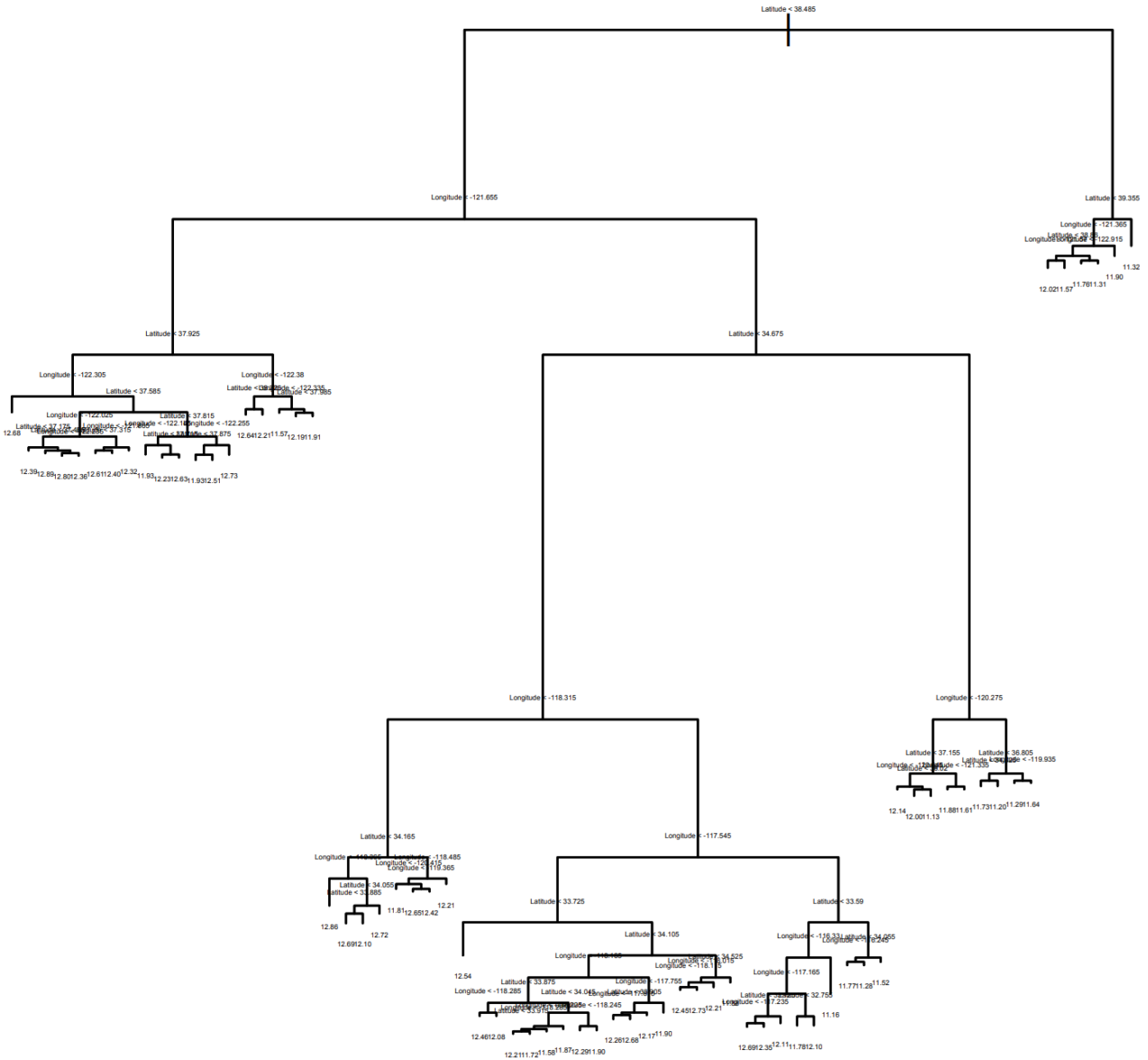
Regularization methods

- There are model-specific and model-agnostic ways to perform regularization
- Model-specific: tied to a specific model
  - For tree, it's called pruning
  - For regression, it's LASSO
  - For deep learning, it's max pooling
- Model-agnostic: train/test split

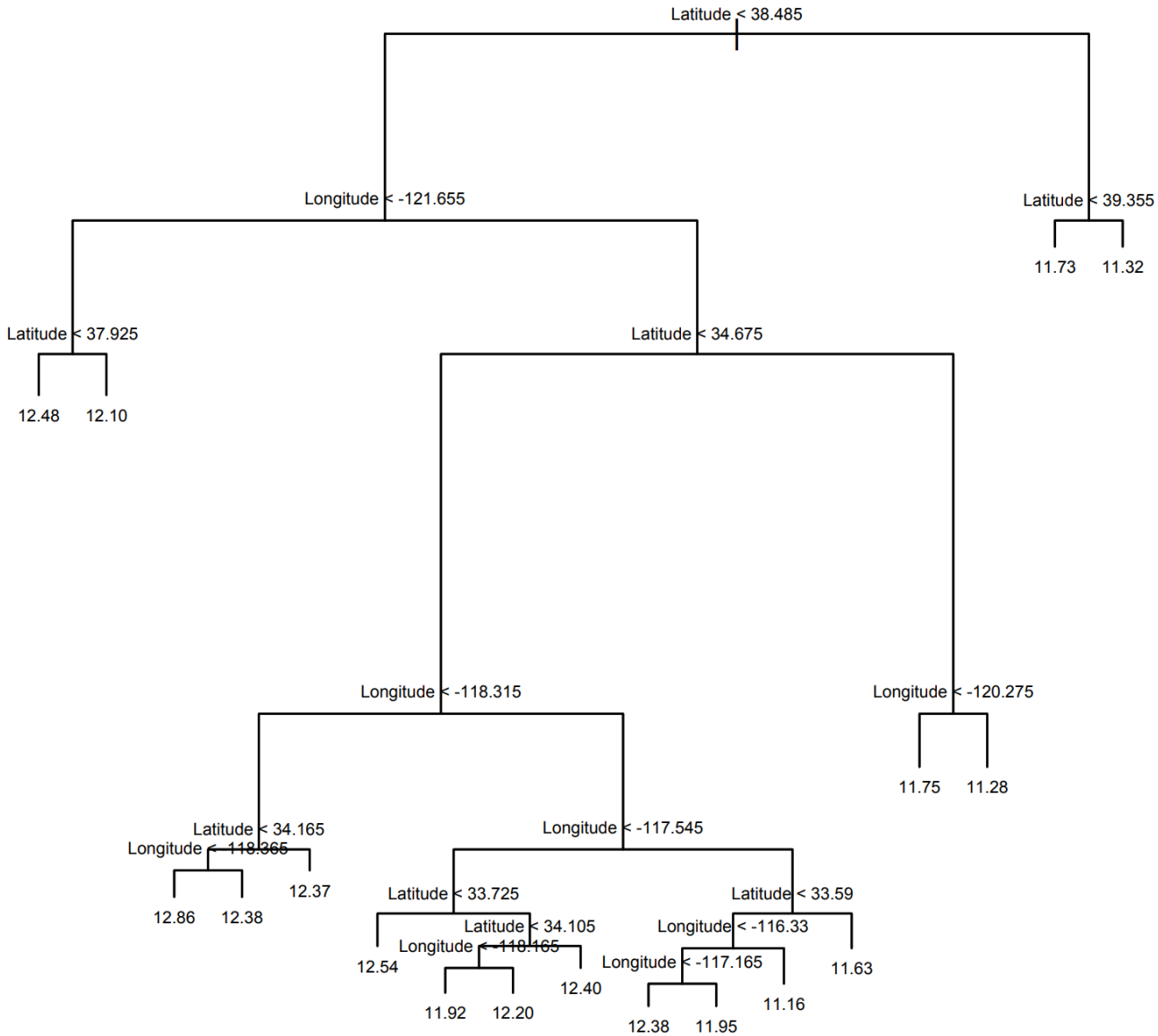
Decision tree pruning

- In decision trees, people often call regularization as pruning
- Intuitively, we are pruning leaves that are too small
  - So that in each leaf, there may be multiple observations and thus prediction errors for the training data
  - But it improves generalization performance
- Many different ways:
  - For the leaf node to have at least k observations
  - Force the tree to have no more than k leaves
  - Force the tree to have no more than k - level deep

Decision Tree: un-pruned



Decision Tree: pruned



Tuning parameters

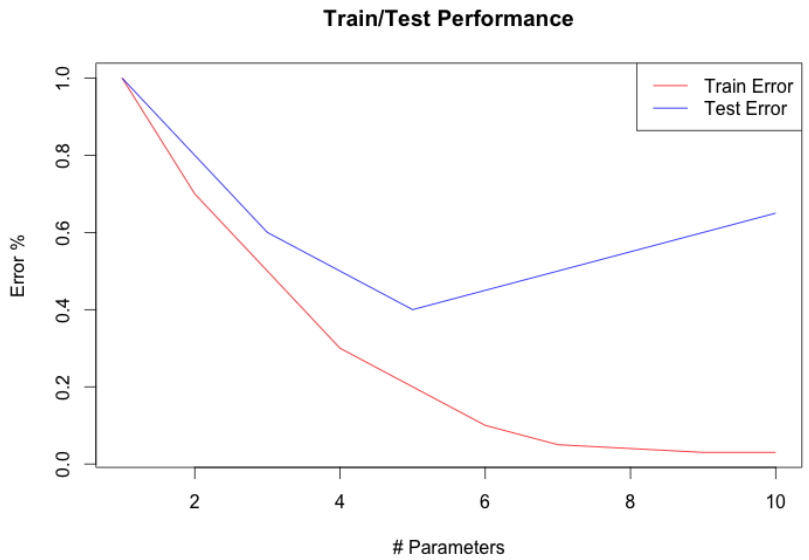
- Pruning is not enough
- E.g., the original tree is 15 levels; should we made it 5 level deep or 4 level deep>
- How can we distinguish between setting the min number of observation for leaf node to be 10 versus 15?
- These parameters, whose values need to be chosen, are called tuning parameters

Training and Test data

- Split data into two sets: training and test data (say, 80% vs. 20%)
- We only use training data to fit the model
- And we test our predictions on the test data
- Our best model should minimize the MSE on test data (also called test error)

Train/test split

- Test MSE is usually larger than training MSE
- When test error begins to increase, overfitting occurs
  - the predictor variances begin to increase
- Error/performance metrics based on test data gives more faithful evaluation of how the algorithm will perform in real-world, unseen new data

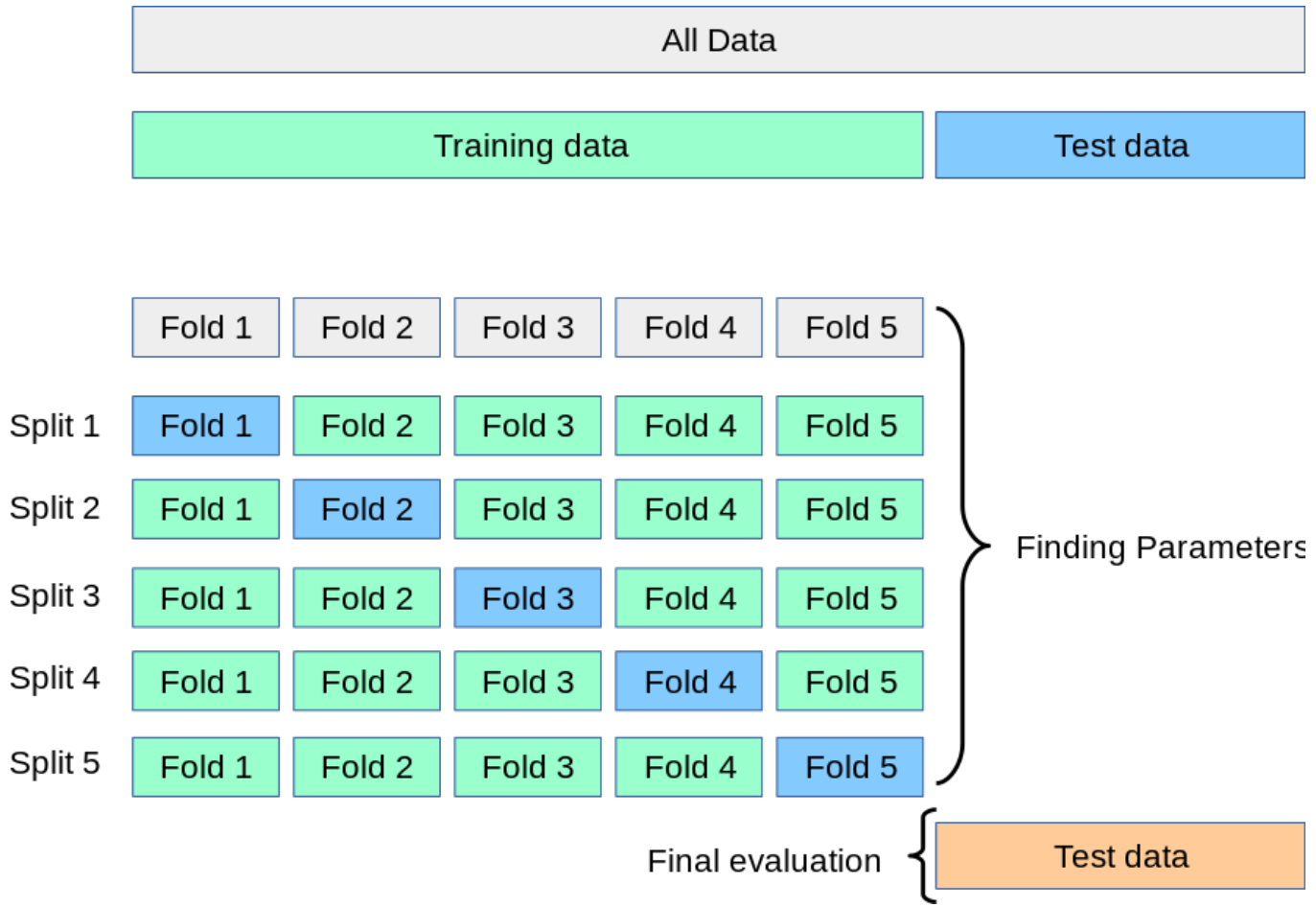


Cross-validation

- Imagine you have chosen a particular two-way split:
  - 80% as training
  - 20% as validation
- Sometimes the concern is that you the particular 80% may be different from the entire sample
- A more complex and popular approach is to use K-fold cross validation

Cross-validation

- K-fold cross validation (below example shows $K = 5$)



Select Tuning Parameters: cross-validation

- K-fold Cross-Validation:
  1. Divide your data into K subsets (i.e., folds) of roughly the same size (K is often 5 or 10)
  2. Select one subset k as the test data, and use the rest $K-1$ subsets as the training data
  3. Calculate the prediction error and save it
  4. repeat and take the average for k prediction errors
- For each possible value of your tuning parameters (e.g., how many observations a leaf node must have), we calculate its average prediction error over K folds
- Then select the choice that made the prediction error the smallest

Causal forests

- Causal forests: a series of articles such as Athey and Imbens, 2016, PNAS; Wager and Athey, 2018, JASA; Athey, Tibshirani, and Wager, 2019, Ann. Stat.
- Setup: we already know treatment D and outcome Y; we know many ways to estimates ATT
- Causal tree finds a split (variable) to maximize the differences in treatment effects
  - the first split reveals the most important causal heterogeneity, and so on so forth

Causal tree

- On each leaf, observations are a mix of treated and control units
- The estimated causal effect for each leaf is the differences in means of treated and control units
- depending where each unit belongs to each unit, the unit level causal effect equals to that of the leaf's causal effect

Avoid p-hacking

- Athey et al. extends the two-way split idea to honest subsampling to alleviate the concern of overfitting / p-hacking
    - you should not use the same data for training and for prediction
- Spliting sample: Use 50% to grow a tree (avoid overfitting by using fewer data)
- In this way, the split that maximize the treatment differences on the training sample may not give you the largest treatment difference on the test data (to calculate true estimates)
    - This is good!

Causal forest

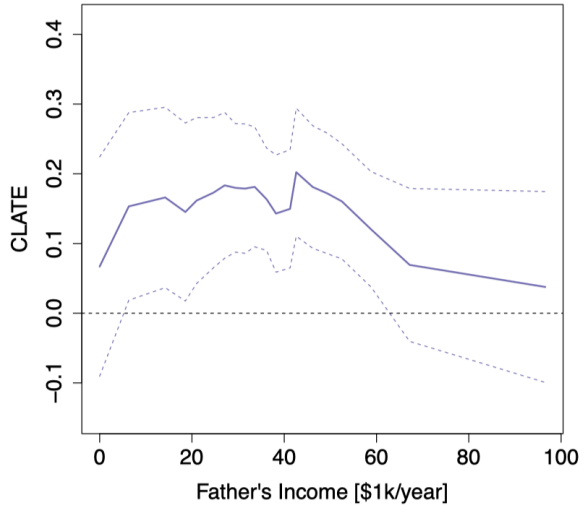
- Estimation sample: Use the rest 50% to calculate treatment effects for each
- Last, the average treatment effect is calculated the average of the estimation sample
- The above procedure can be repeated many times; from causal tree to causal forests

Causal forest

- Athey, Tibshirani, and Wager, 2019, Ann. Stat.
- Questions: child penalty? more children decrease mother's tendency to work?
- Treatment is whether the mother had 3 or more children at census time
- Outcome: whether the mother did not work in the year preceding the census
- HTE question: does treatment effect varies by other covariates?
  - e.g., by fathers' income?
  - Regression approach: just interact father's income with treatment

Causal forest

- Causal forest: predict individual treatment effect τ_i; then plot τ_i against father's income
- Found an inverse-U shape HTE; not easy to obtain if you just interact father's income with treatment status



Causal forests for observational data

- The default causal forest works with experiment data
- If you want to use that for observational data
  - Has to add ignorability assumption (more shortly)
  - And then weight observations by their treatment propensity
  - Called propensity trees
- R package called `grf` to achieve the above