

CSC 4020 Fundamentals of Machine Learning: Decision Trees, Bagging, & Random Forests

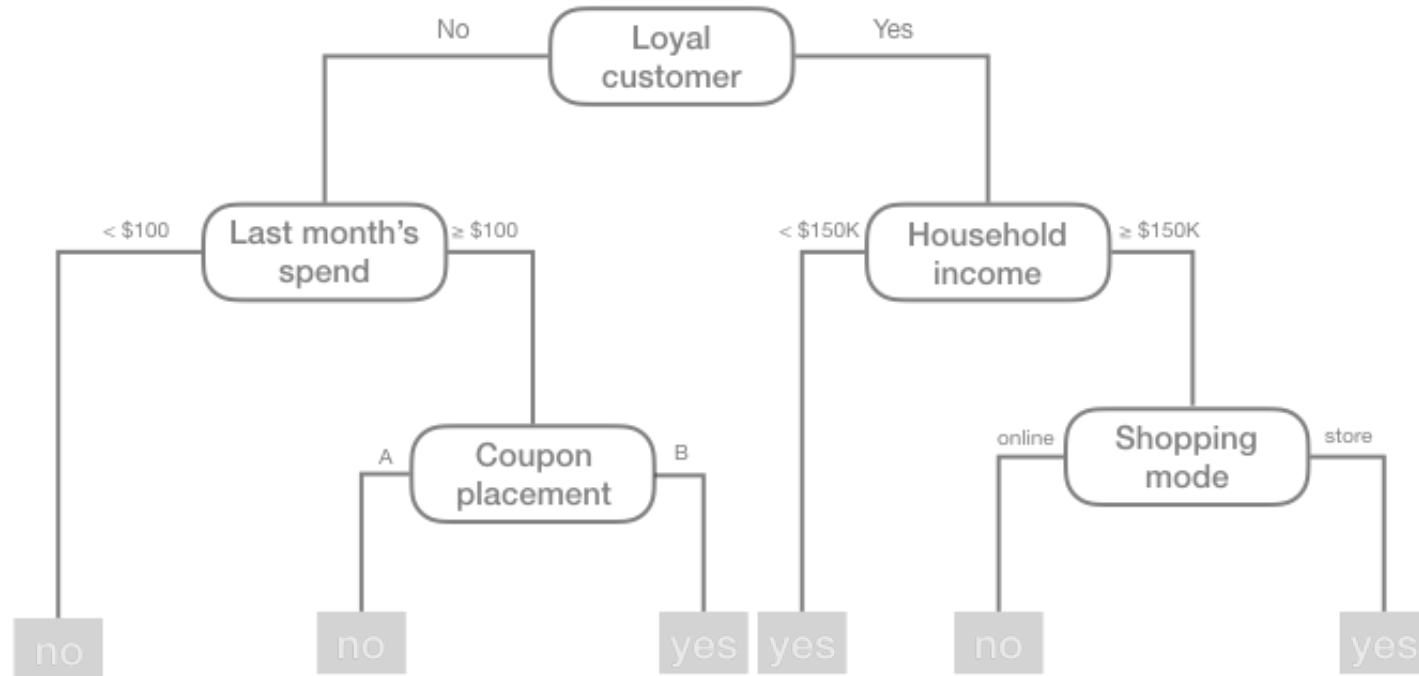
Baoyuan Wu

March 29/31

Source: <https://bradleyboehmke.github.io/random-forest-training/slides-source.html#>

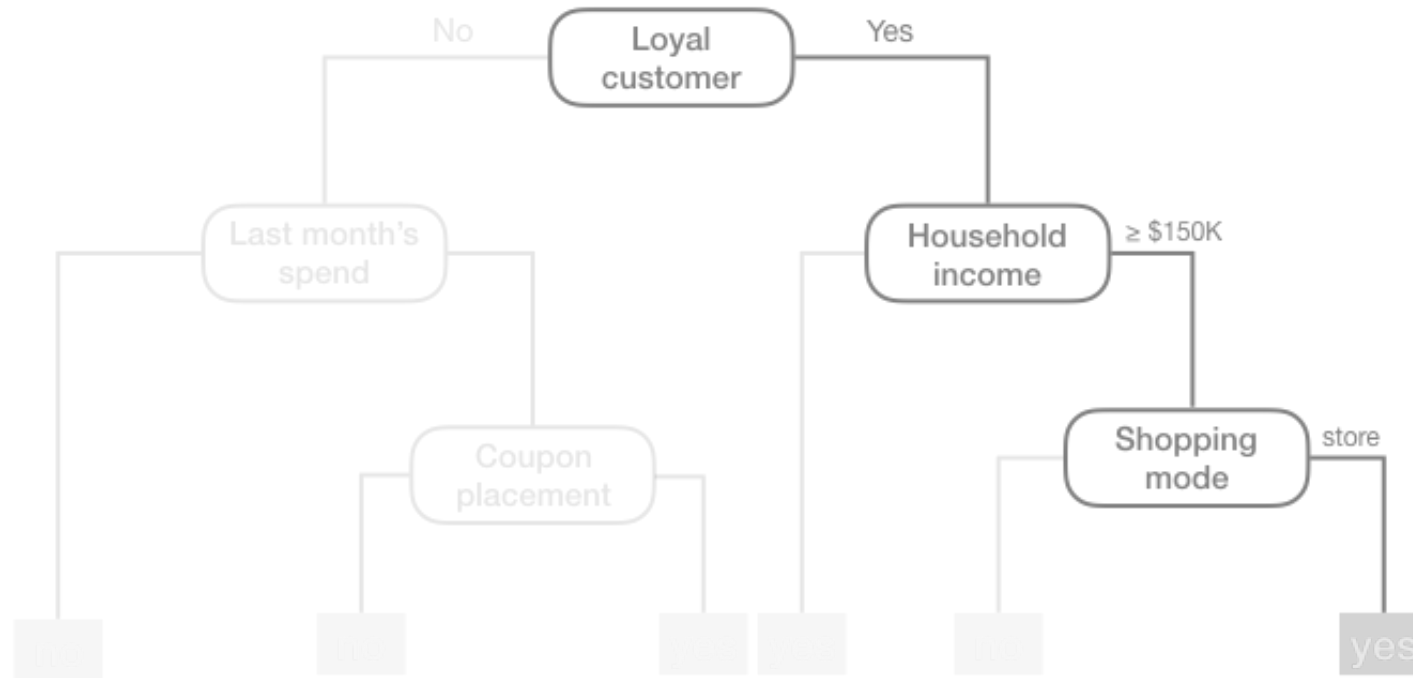
Decision trees

Basic Idea



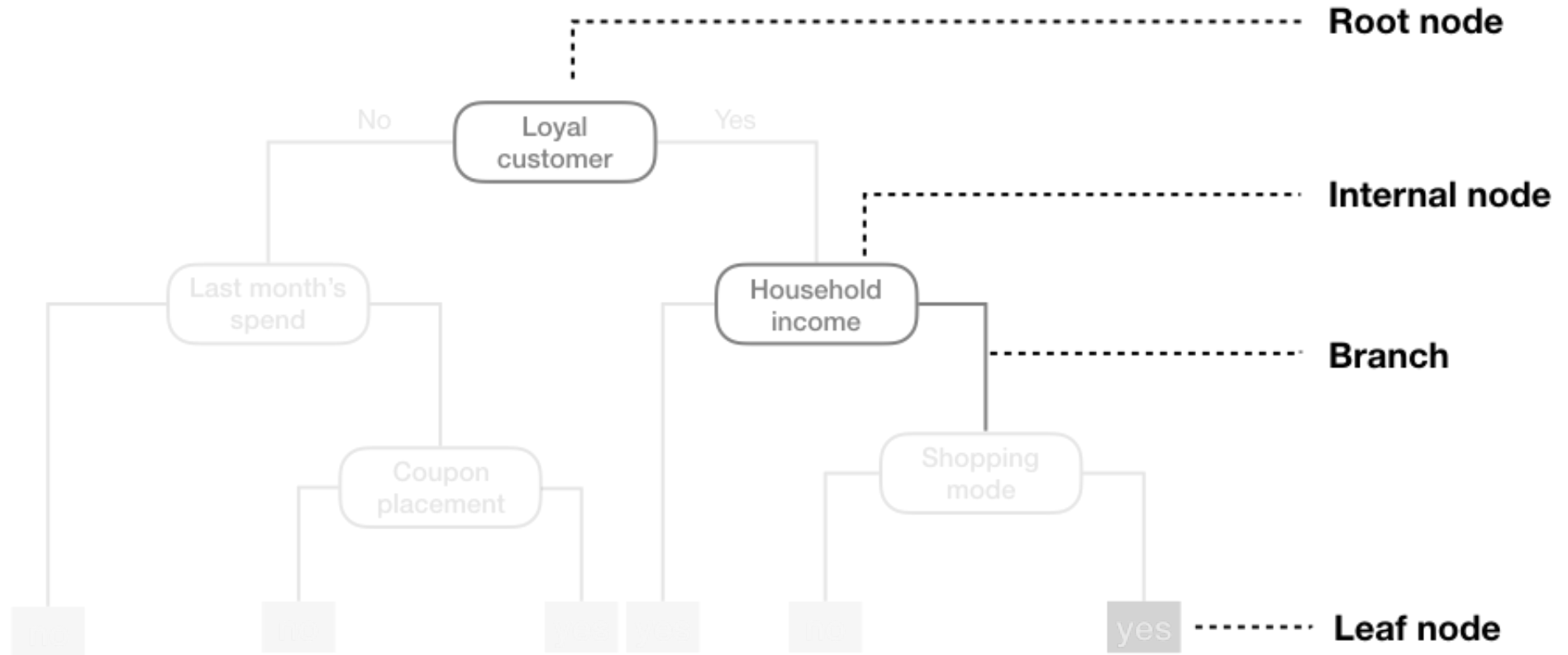
Will a customer redeem a coupon

A ruleset model



if Loyal Customer = Yes and Household income >= \$150K and Shopping mode = store then coupon redemption = Yes

Terminology



Growing the tree

Algorithms


- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)
- CHAID (CHi-squared Automatic Interaction Detector)
- MARS: (Multivariate Adaptive Regression Splines)
- Conditional Inference Trees
- and more...

Growing the tree

Algorithms

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- **CART (Classification And Regression Tree)**
- CHAID (CHi-squared Automatic Interaction Detector)
- MARS: (Multivariate Adaptive Regression Splines)
- Conditional Inference Trees
- and more...

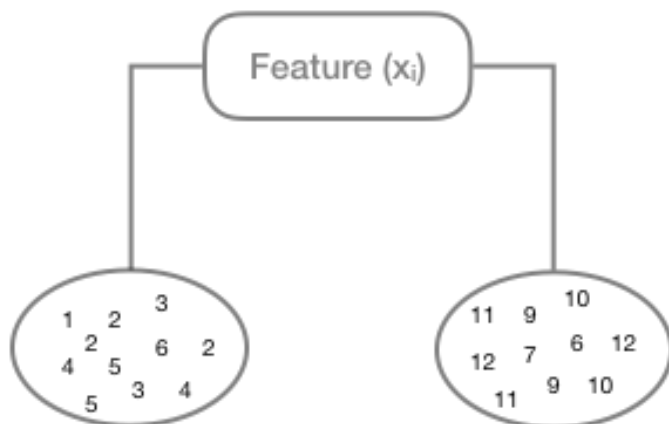
CART Features

- Classification and regression trees
- Continuous and discrete features
- Partitioning
 - Greedy top-down
 - Strictly binary splits (tends to produce tall/deep trees)
 - Variance reduction in regression trees
 - Gini impurity in classification trees
- Cost complexity pruning
-  (Breiman, 1984)

Most common decision tree algorithm

Best Binary Partitioning

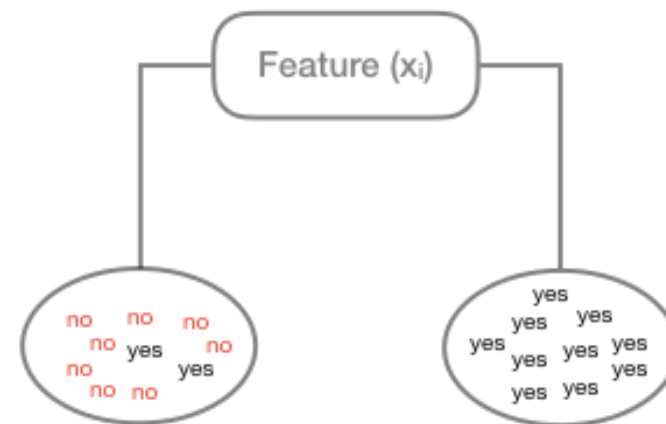
Regression tree



Prediction: 3.36
SSE: 24.55

9.7
36.1

Classification tree



Prediction: no
Gini: 0.32

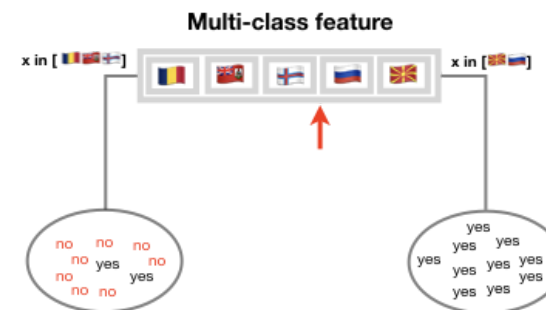
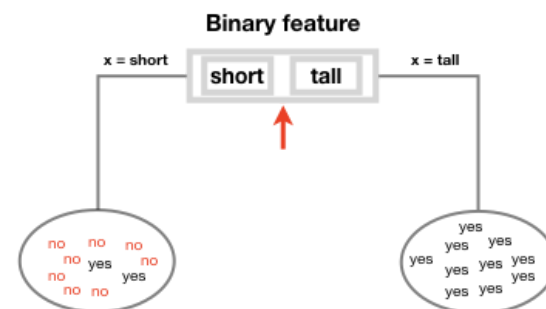
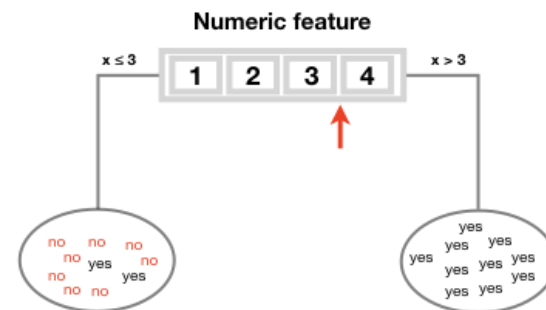
yes
0

$$I_G(D) = 1 - \sum_{i=1}^m p_i^2$$

Objective: Minimize dissimilarity in terminal nodes

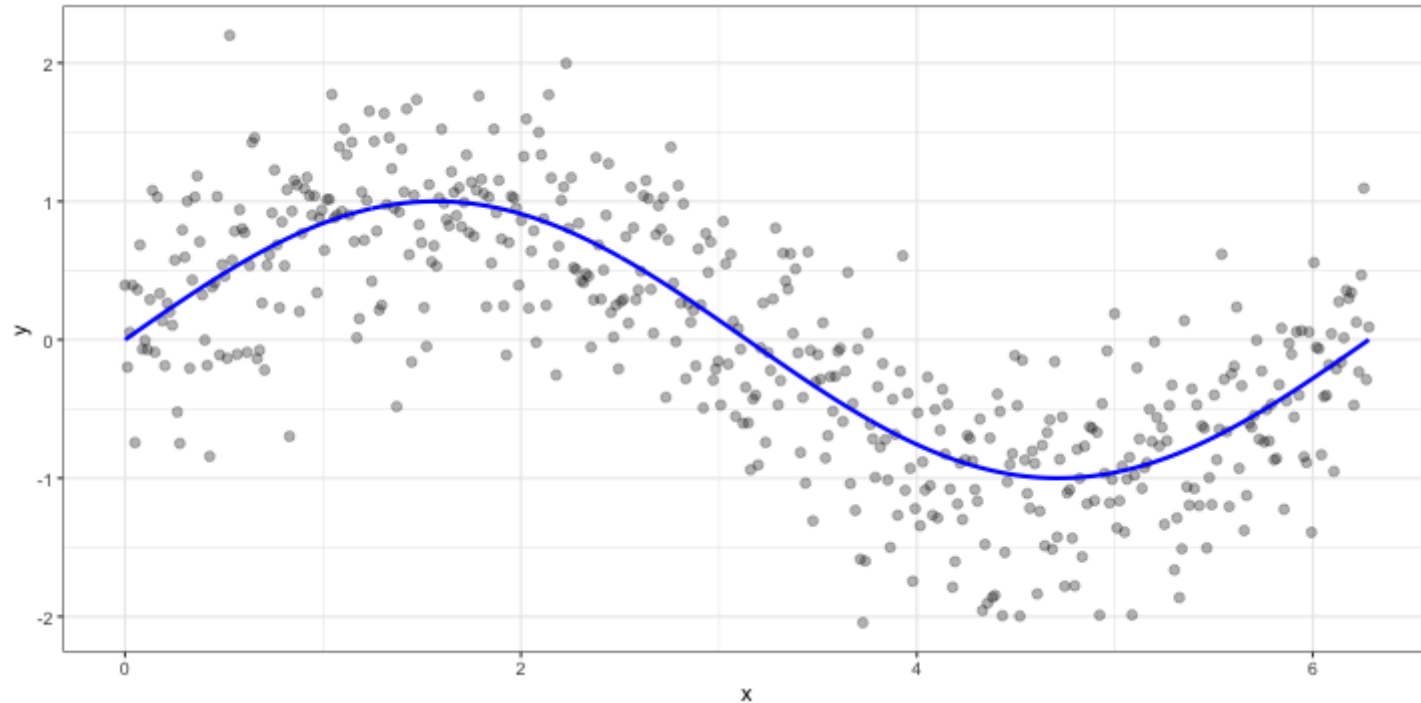
Best **Binary** Partitioning

- **Numeric feature:** Numeric split to minimize loss function
- **Binary feature:** Category split to minimize loss function
- **Multiclass feature:** Order feature classes based on mean target variable (regression) or class proportion (classification) and choose split to minimize loss function (🔗 See ESL, section 9.2.4 for details).

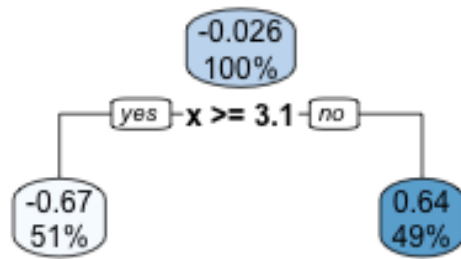


How deep to grow a tree?

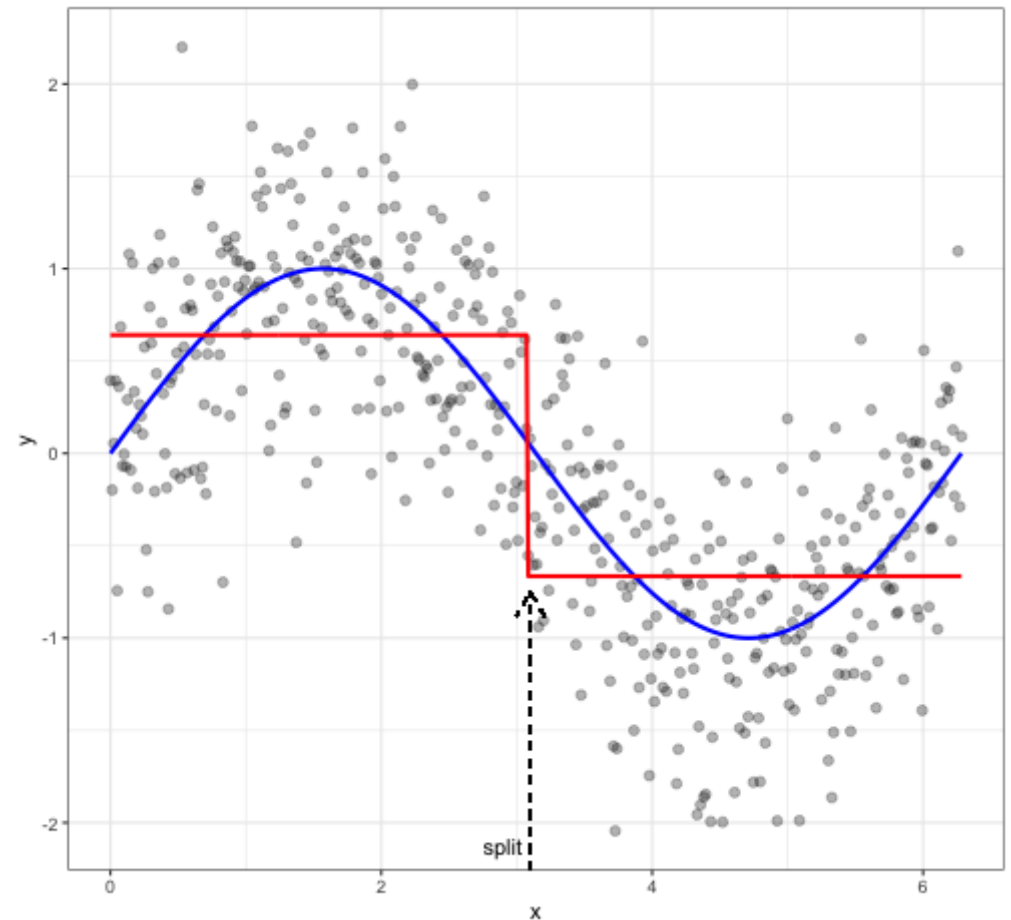
Say we have the given data generated from the underlying "truth" function



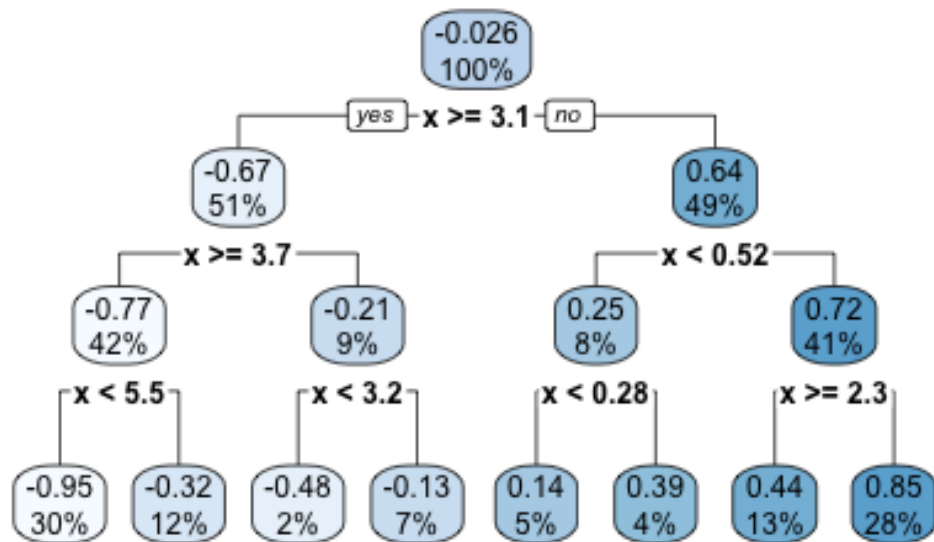
Depth = 1 (decision **stump** 🪵)



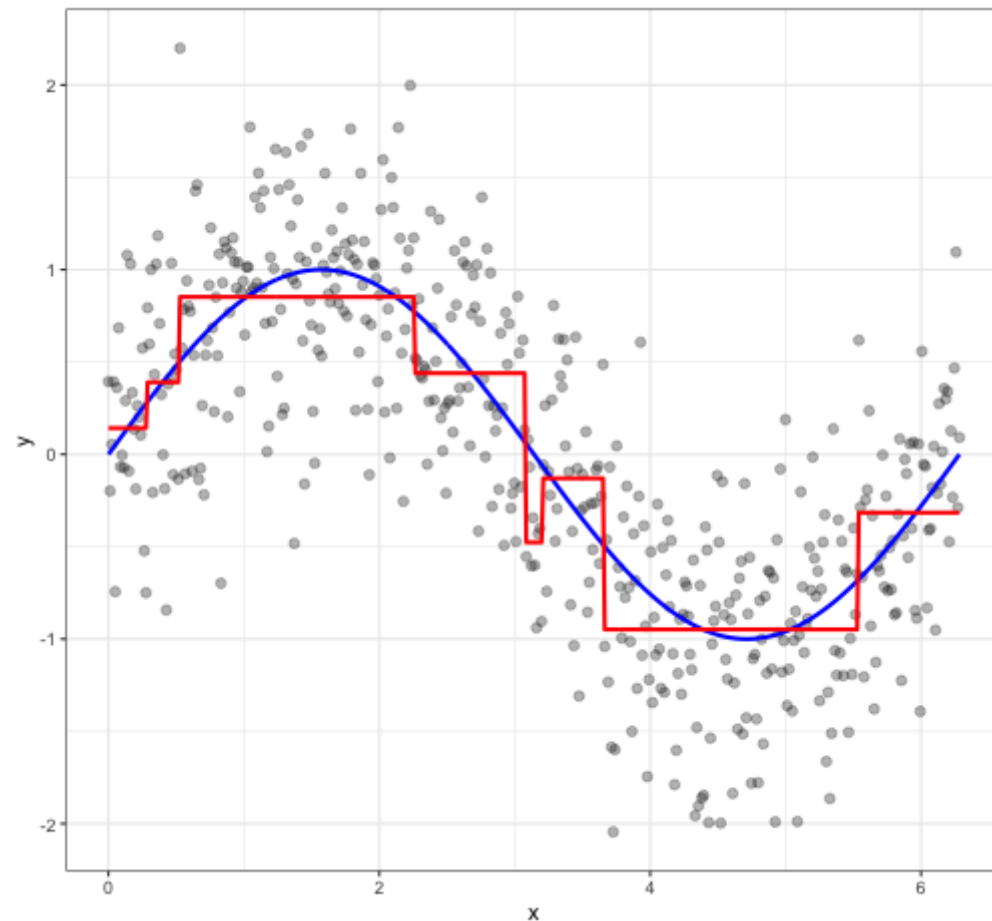
```
##  
## Model formula:  
## y ~ x  
##  
## Fitted party:  
## [1] root  
## | [2] x >= 3.07863: -0.665 (n = 255, err = 95.5)  
## | [3] x < 3.07863: 0.640 (n = 245, err = 75.9)  
##  
## Number of inner nodes: 1  
## Number of terminal nodes: 2
```



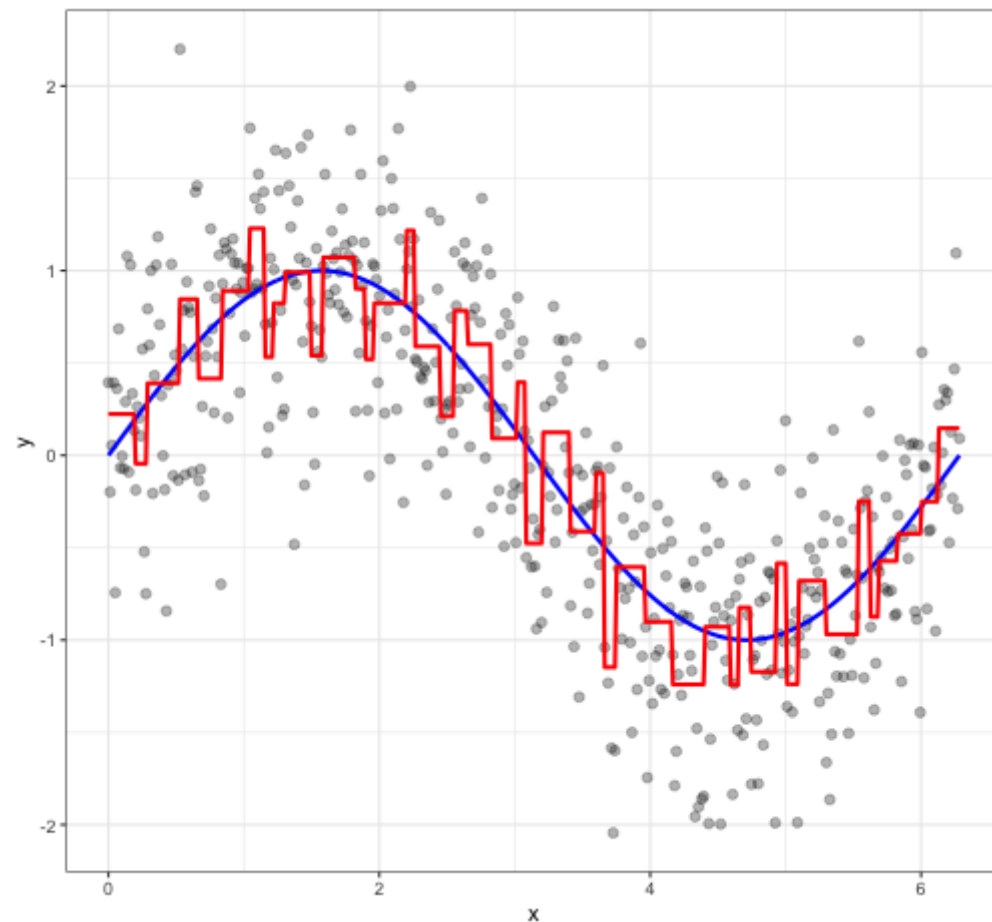
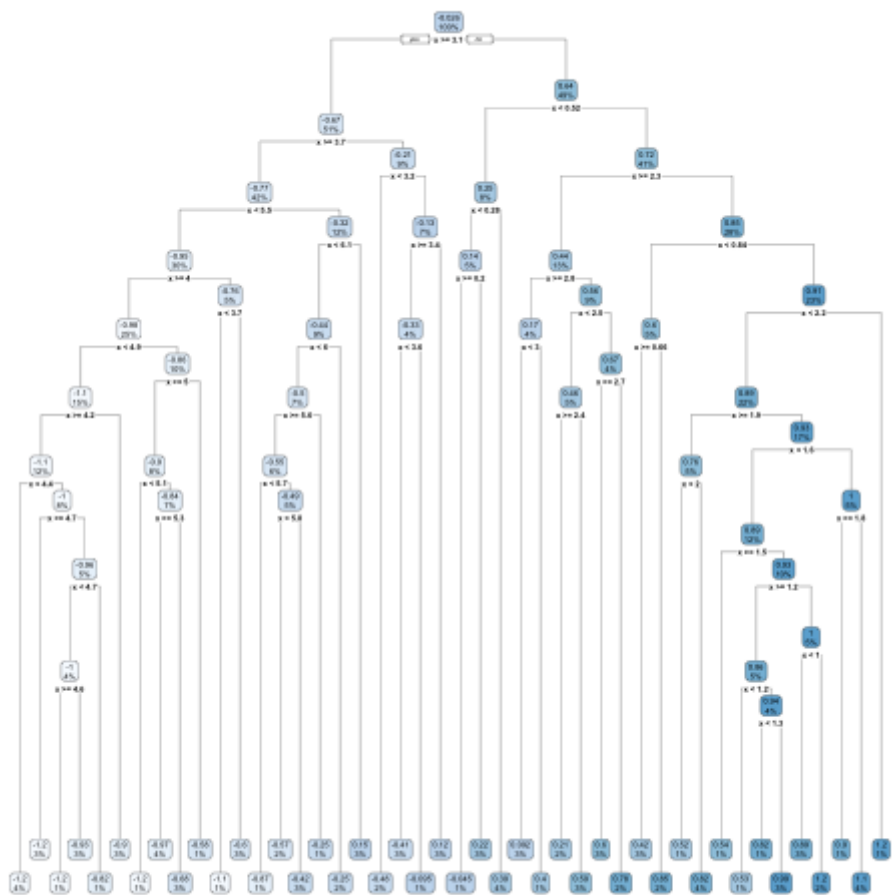
Depth = 3



```
##
## Model formula:
## y ~ x
##
## Fitted party:
## [1] root
```

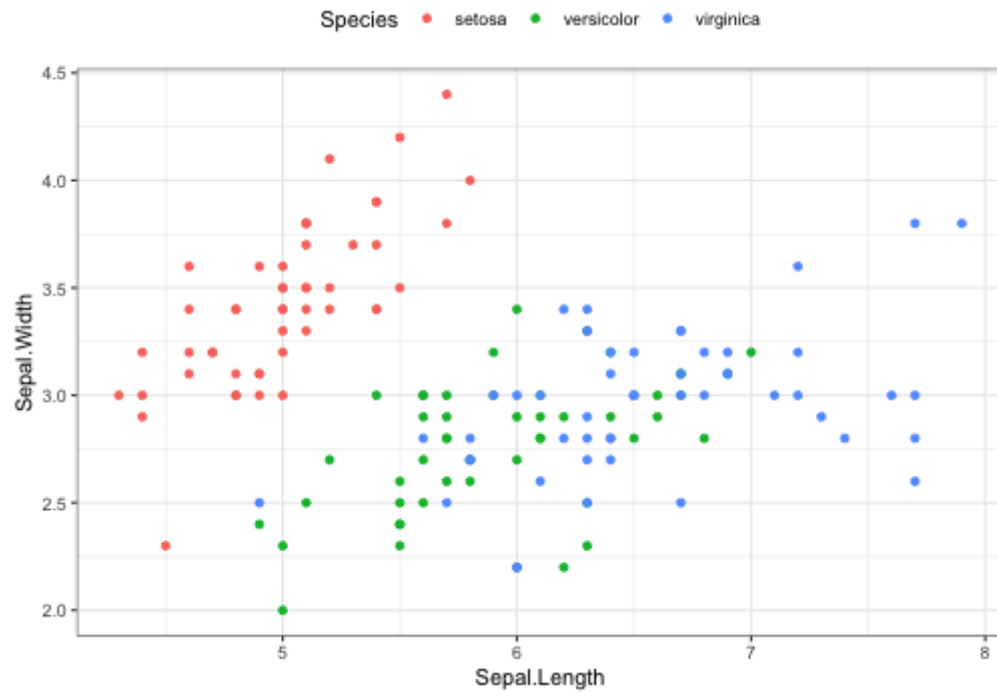


Depth = 20 (complex tree)



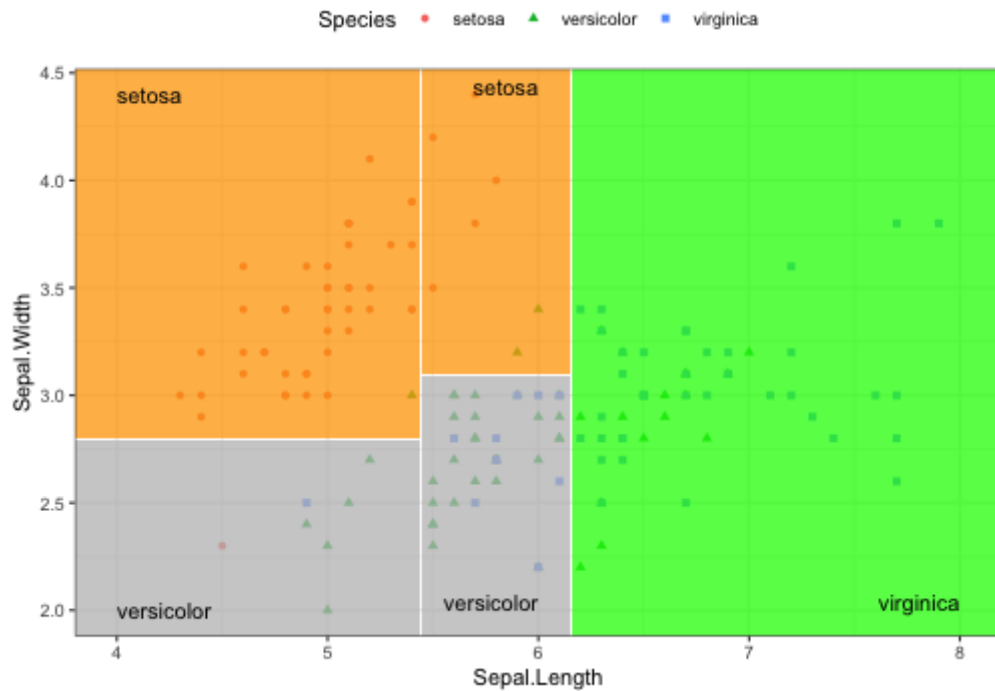
Two Predictor Decision Boundaries

Classification problem: Iris data

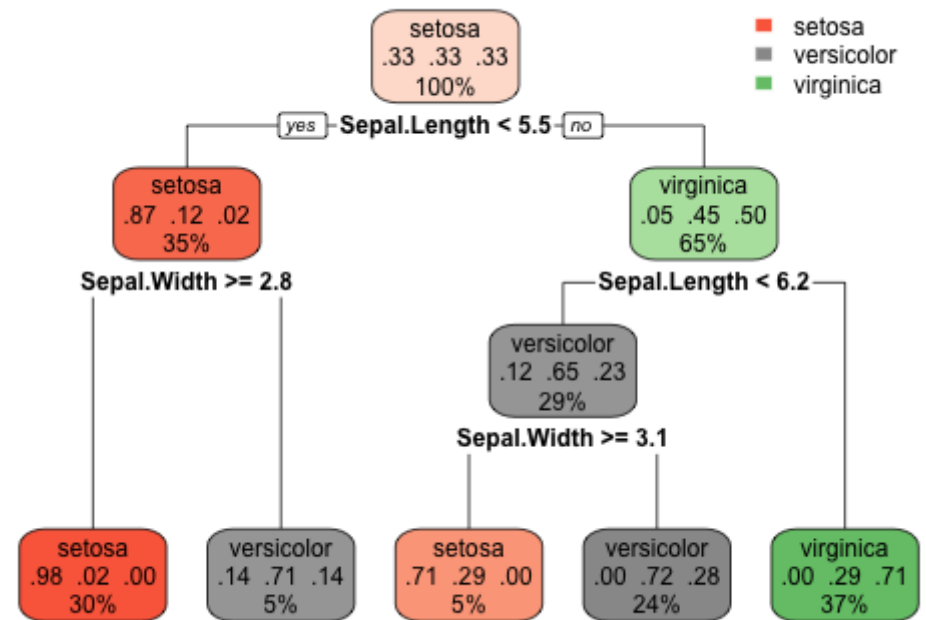


Two Predictor Decision Boundaries

Classification problem: Iris data



Classification tree

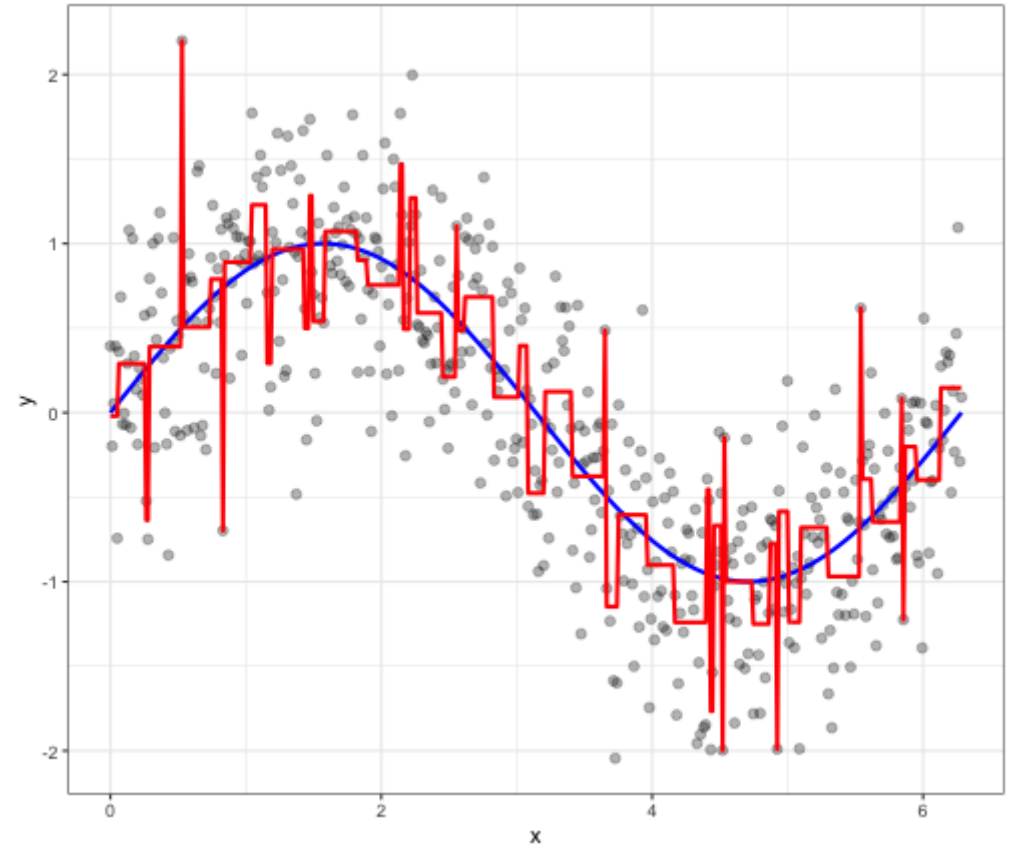


Minimize overfitting

Must balance the depth and complexity of the tree to **generalize** to unseen data

2 main options:

- Early stopping
 - Restrict tree depth
 - Restrict node size
- Pruning



Trees have a tendency to overfit

Minimize overfitting: Early stopping

Limit tree depth: Stop splitting after a certain depth

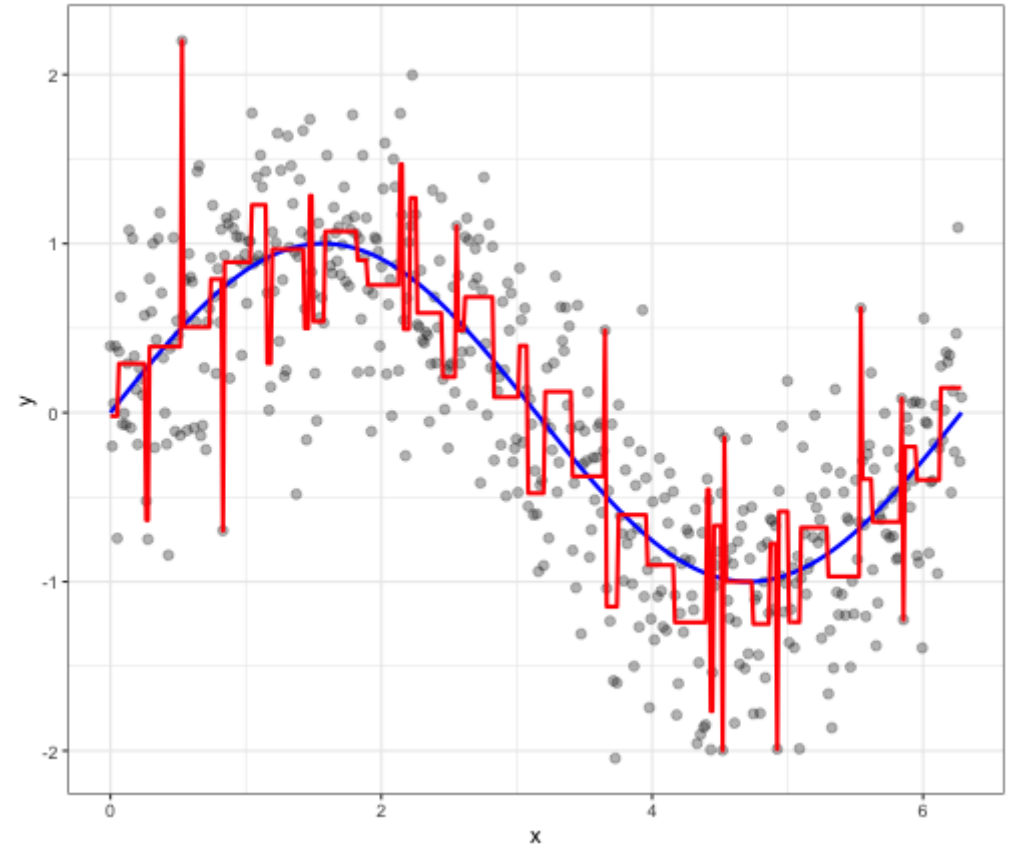
Minimize overfitting: Early stopping

Limit tree depth: Stop splitting after a certain depth

Minimum node “size”: Do not split intermediate node which contains too few data points

Minimize overfitting: Pruning ✂

1. Grow a very large tree

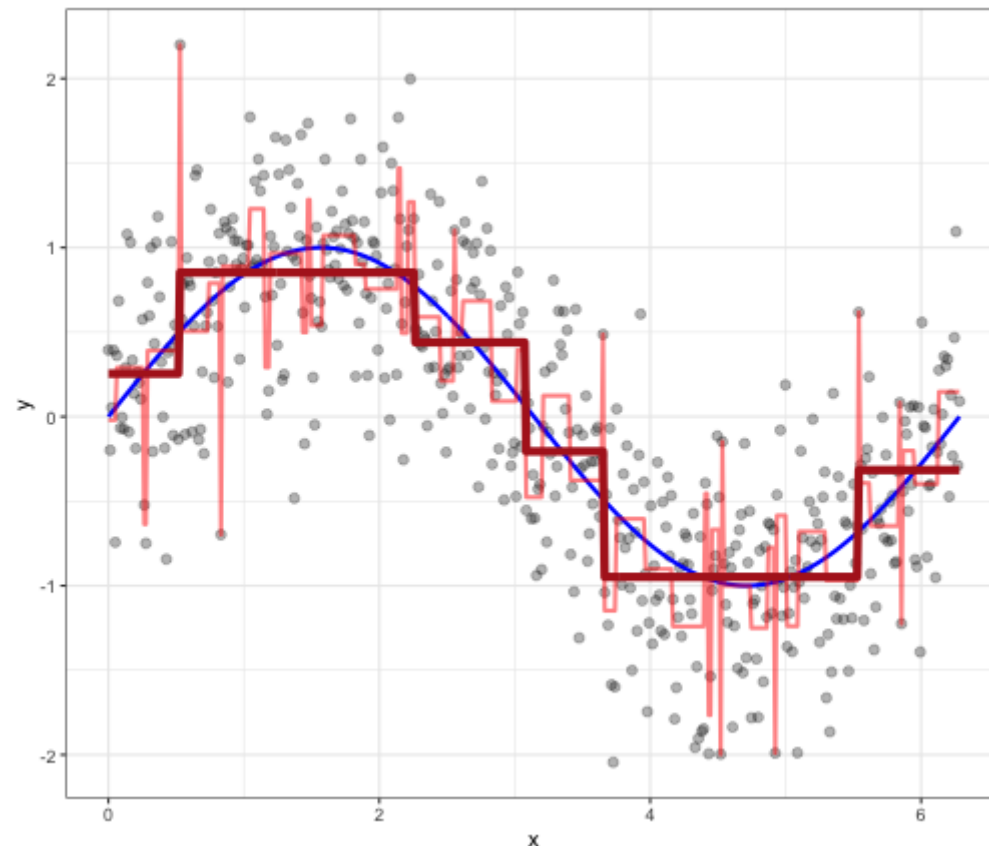


Deep trees overfit

Minimize overfitting: Pruning ✂

1. Grow a very large tree
2. Prune it back with a *cost complexity parameter* (α) \times number of terminal nodes (T) to find an optimal subtree:
 - Very similar to lasso penalty in regularized regression
 - Large α = small tree
 - Small α = large tree
 - Find optimal α with cross validation

minimize: loss function + $\alpha|T|$



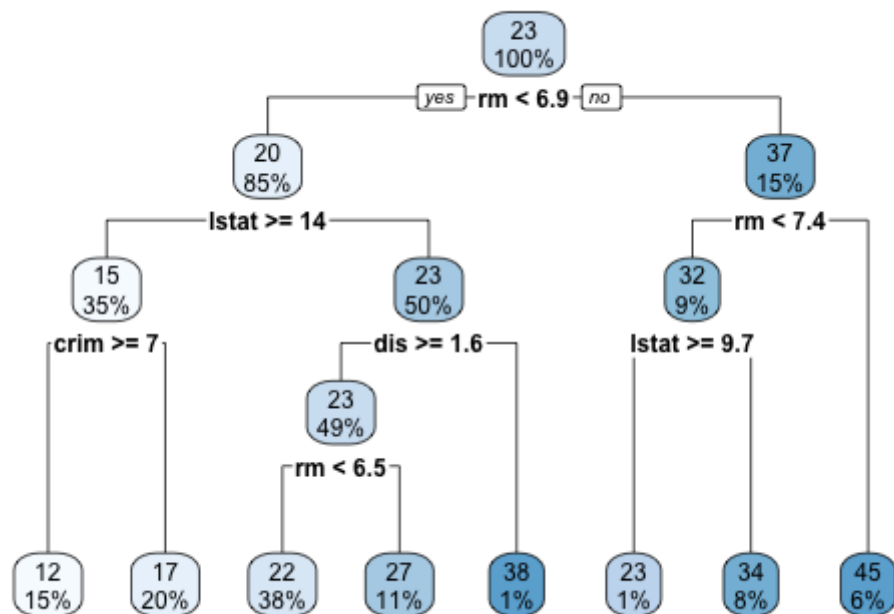
Penalize depth to generalize

Feature/Target Pre-processing Considerations

- **Monotonic transformations** (i.e. log, exp, sqrt): **Not required** to meet algorithm assumptions as in many parametric models; only shifts the optimal split points.
- **Removing outliers**: **unnecessary** as the emphasis is on a single binary split and outliers are not going to bias that split.
- **One-hot encoding**: **unnecessary** and actually forces artificial relationships between categorical levels. Also, by increasing p , we reduce the probability that influential levels and variable interactions will be identified.
- **Missing values**: **unnecessary** as most algorithms will 1) create new "missing" class for categorical variables, 2) auto-impute for continuous variables, or 3) use *surrogate* splits

Variable importance

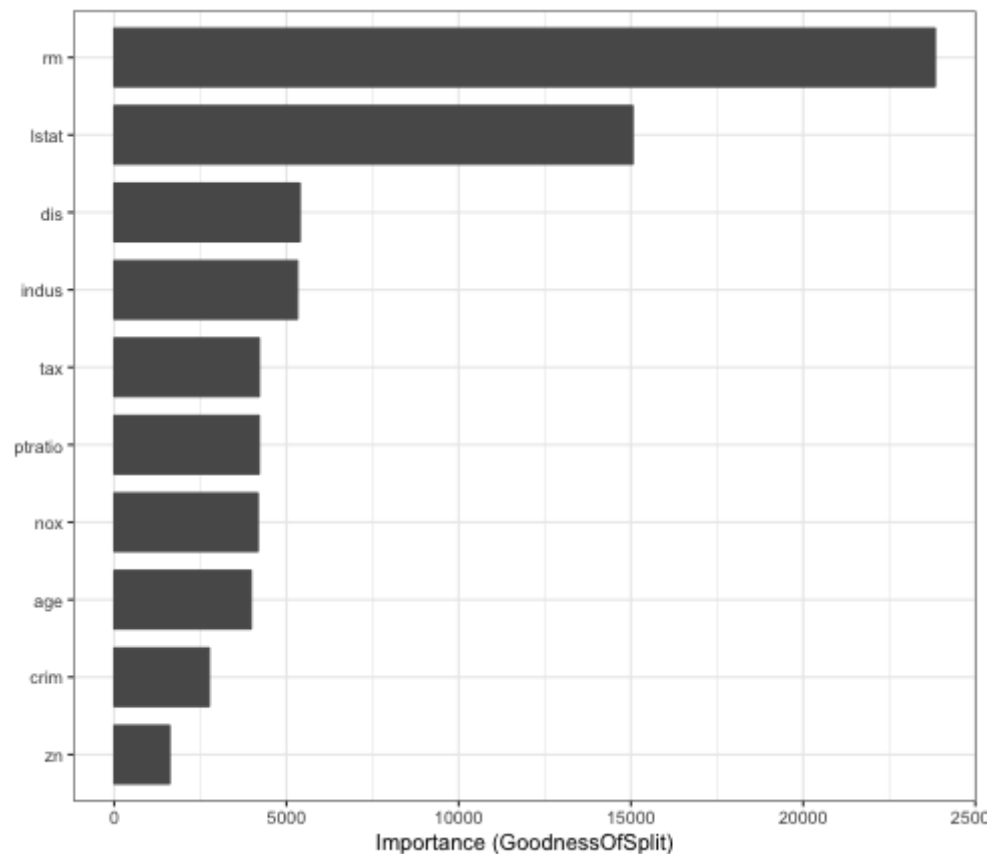
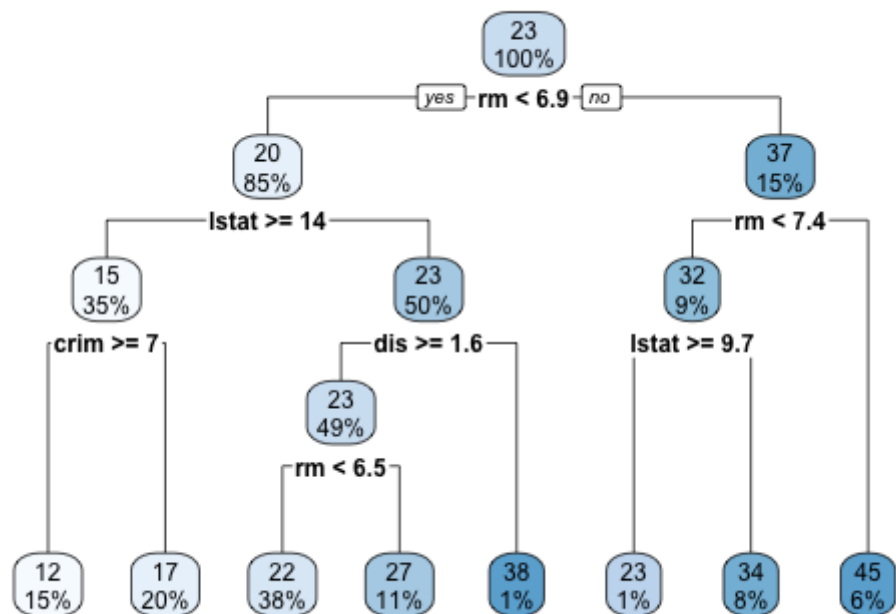
Once we have a final model, we can find the most **influential variables** based on those that have the **largest reduction** in our loss function:



##	Variable	Importance
## 1	rm	23825.9224
## 2	lstat	15047.9426
## 3	dis	5385.2076
## 4	indus	5313.9748
## 5	tax	4205.2067
## 6	ptratio	4202.2984
## 7	nox	4166.1230
## 8	age	3969.2913
## 9	crim	2753.2843
## 10	zn	1604.5566
## 11	rad	1007.6588
## 12	black	408.1277

Variable importance

Once we have a final model, we can find the most **influential variables** based on those that have the **largest reduction** in our loss function:



Strengths & Weaknesses

Strengths 🙌

- Small trees are easy to interpret
- Trees scale well to large N (fast!!)
- Can handle data of all types (i.e., requires little, if any, preprocessing)
- Automatic variable selection
- Can handle missing data
- Completely nonparametric

Strengths & Weaknesses

Strengths 🤔

- Small trees are easy to interpret
- Trees scale well to large N (fast!!)
- Can handle data of all types (i.e., requires little, if any, preprocessing)
- Automatic variable selection
- Can handle missing data
- Completely nonparametric

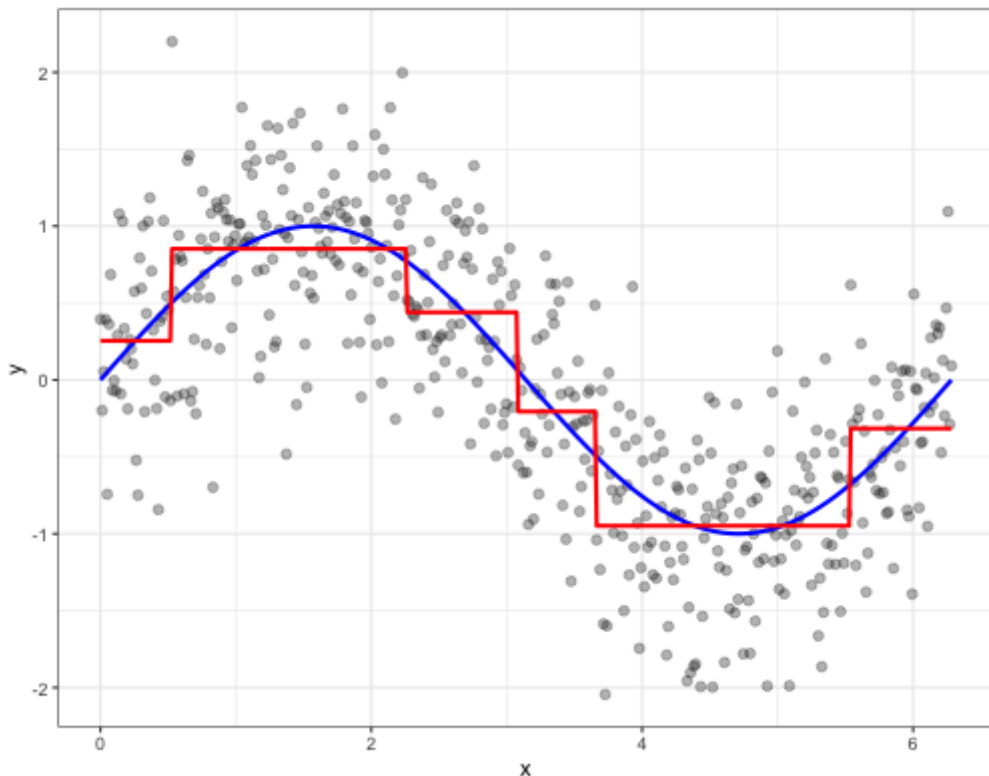
Weaknesses 😞

- Large trees can be difficult to interpret
- All splits depend on previous splits (i.e. capturing interactions 👍; additive models 👎)
- Trees are step functions (i.e., binary splits)
- Single trees typically have poor predictive accuracy
- Single trees have high variance (easy to overfit to training data)

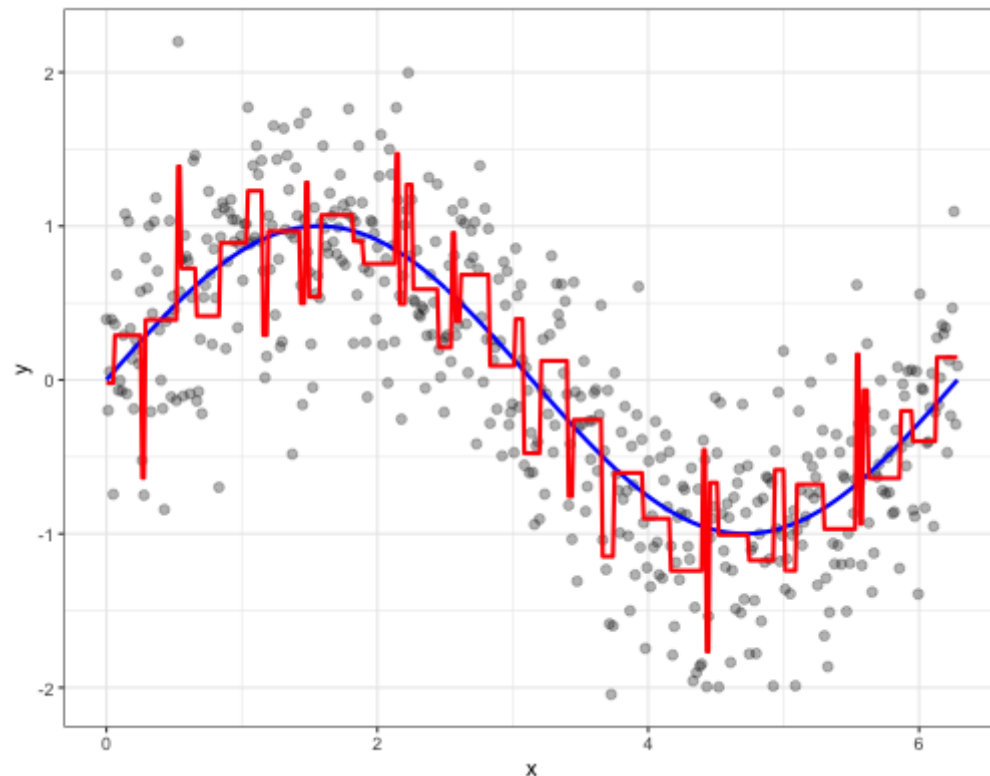
Bagging

The problem with single trees

Single pruned trees are poor predictors



Single deep trees are noisy



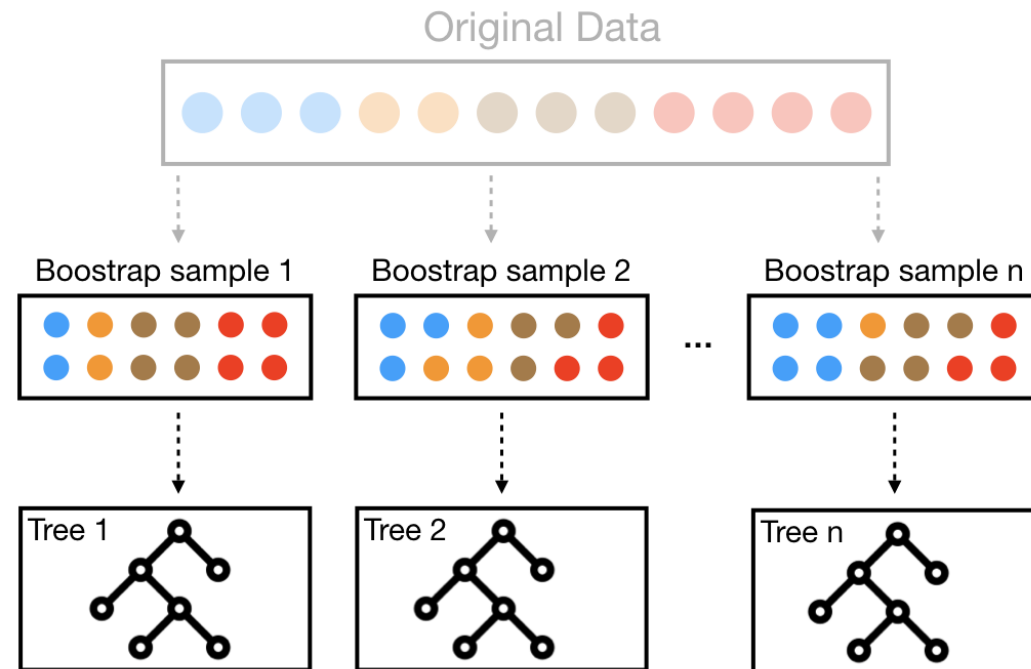
Bagging uses this high variance to our advantage ↑

Bootstrap Aggregating: wisdom of the crowd

1. Sample records with replacement (aka "bootstrap" the training data)
- 2.
- 3.

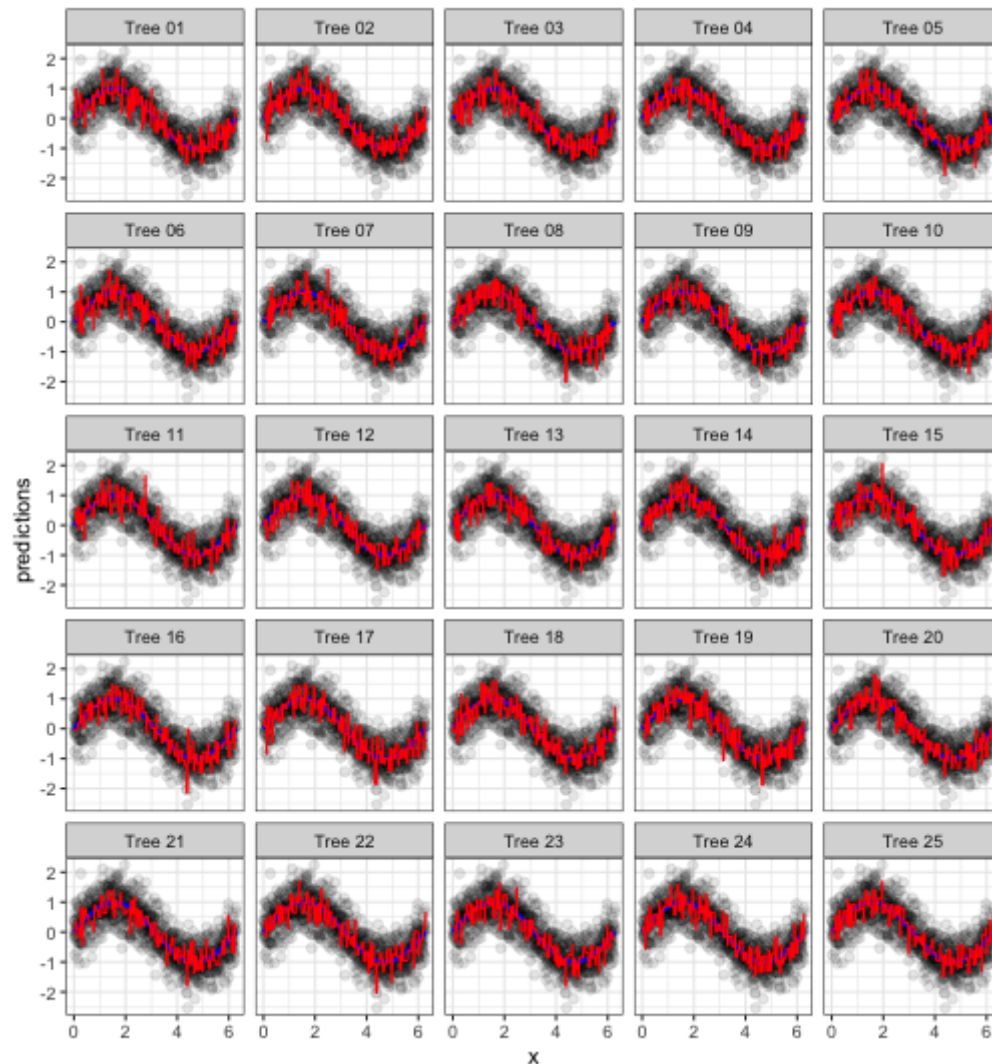
Bootstrap Aggregating: wisdom of the crowd

1. Sample records with replacement (aka "bootstrap" the training data)
2. Fit an overgrown tree to each resampled data set
- 3.



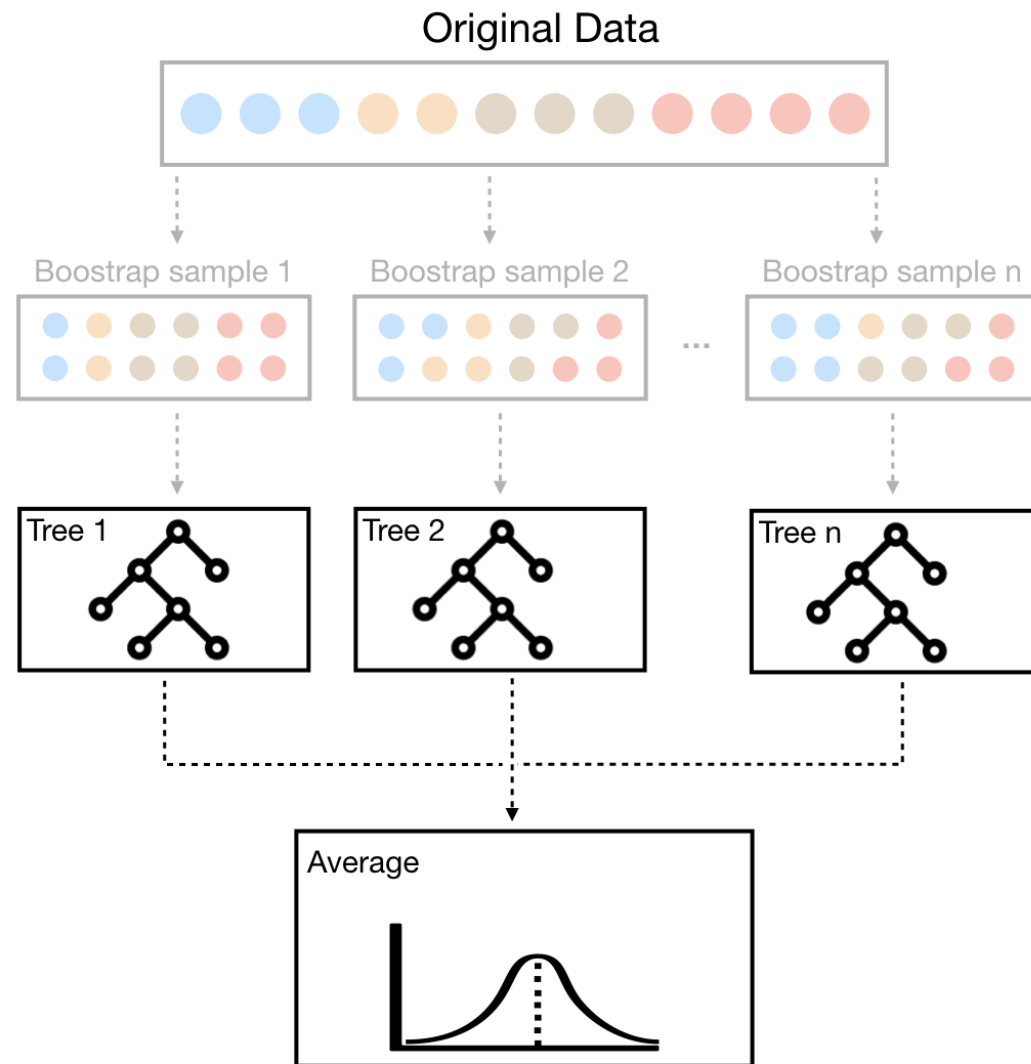
Bootstrap Aggregating: wisdom of the crowd

1. Sample records with replacement (aka "bootstrap" the training data)
2. Fit an overgrown tree to each resampled data set
- 3.



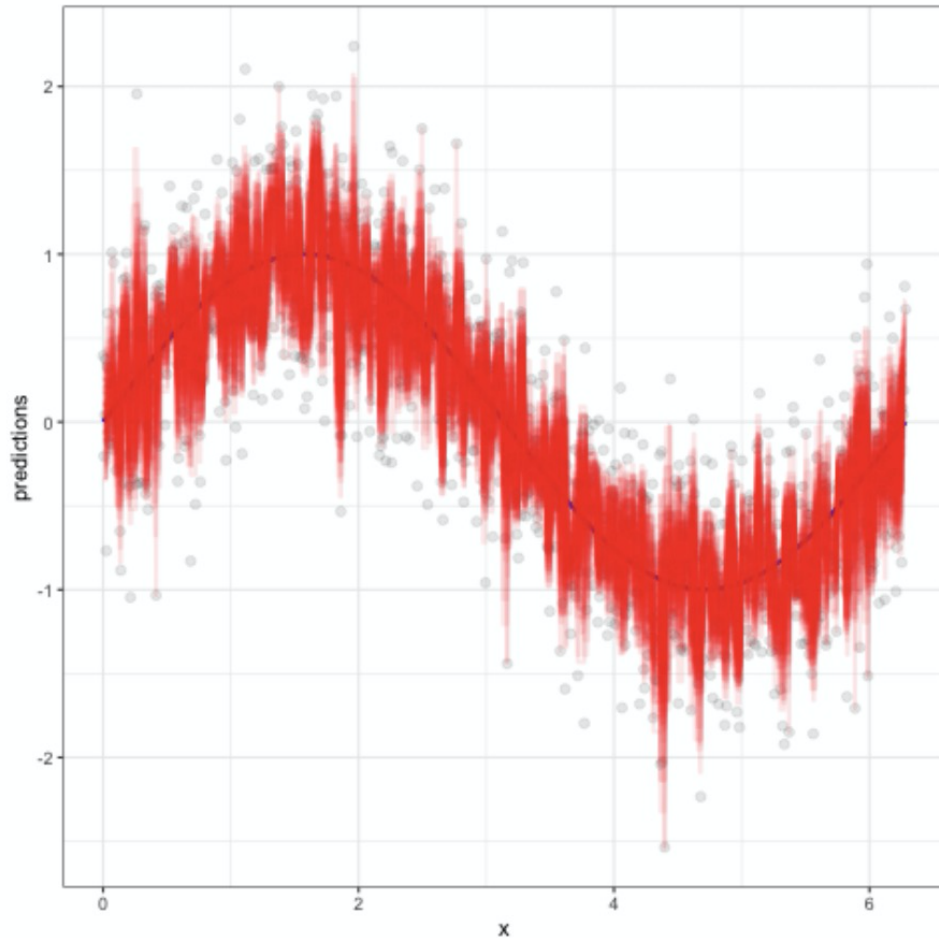
Bootstrap Aggregating: wisdom of the crowd

1. Sample records with replacement (aka "bootstrap" the training data)
2. Fit an overgrown tree to each resampled data set
3. Average predictions

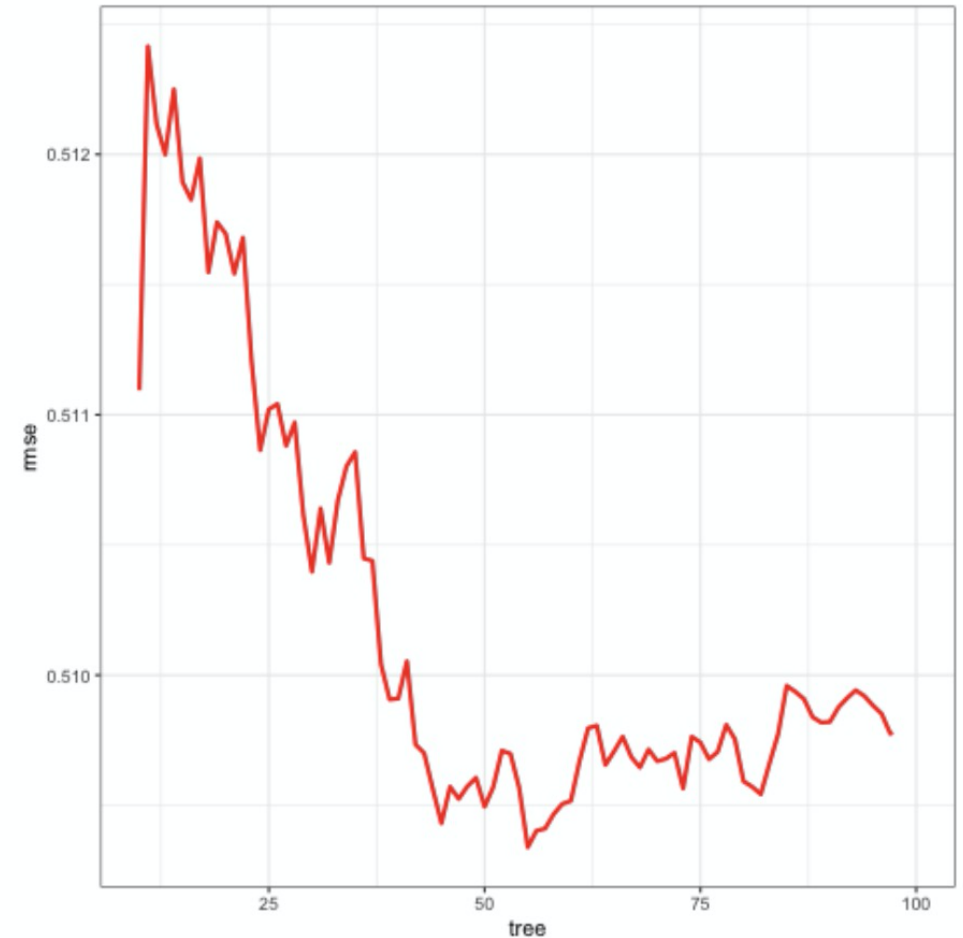


Bootstrap Aggregating: wisdom of the crowd

As we add more trees...



our average prediction error reduces



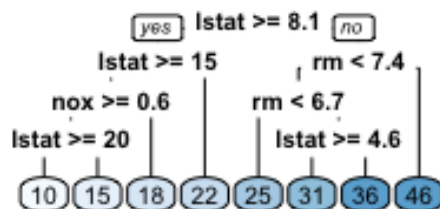
However, a **problem remains**

Bagging results in tree correlation...

Decision Tree 1



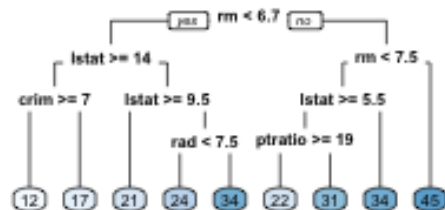
Decision Tree 2



Decision Tree 3



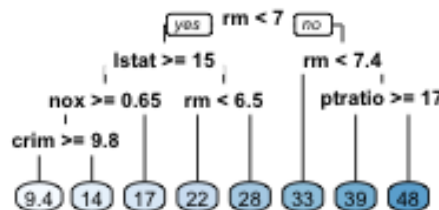
Decision Tree 4



Decision Tree 5



Decision Tree 6



which prevents bagging from optimally reducing variance of the predictive values

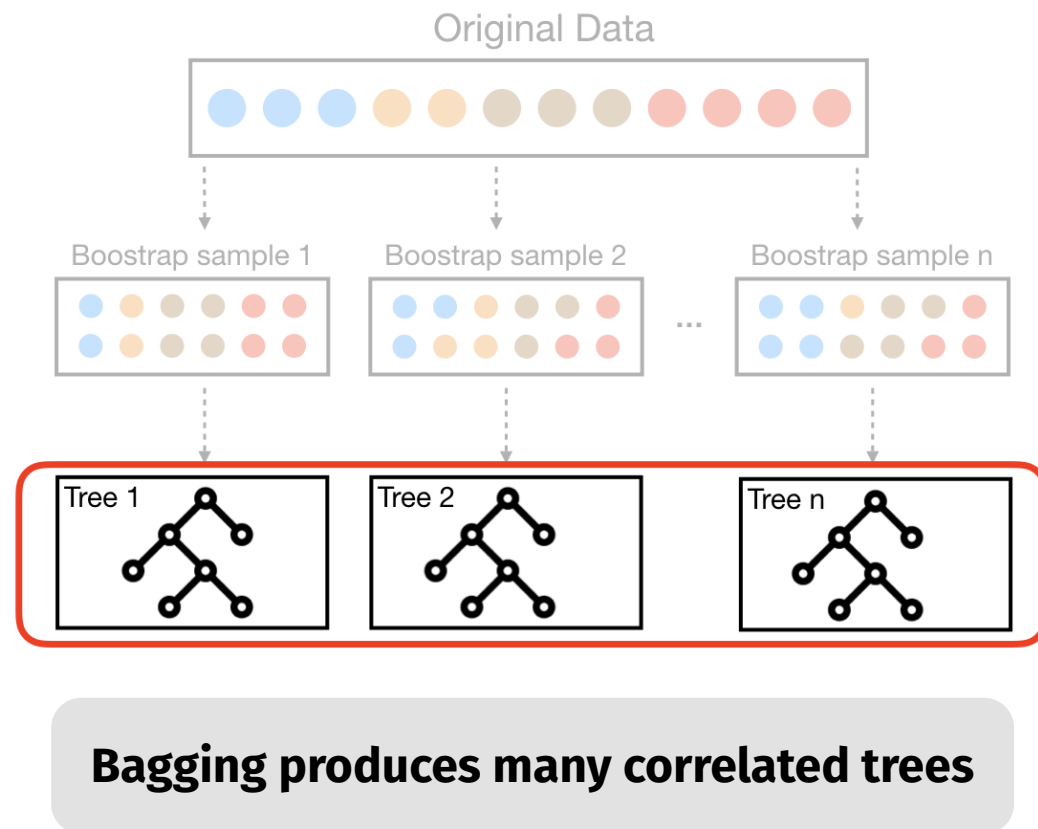


Random Forests

Idea

Split-variable randomization

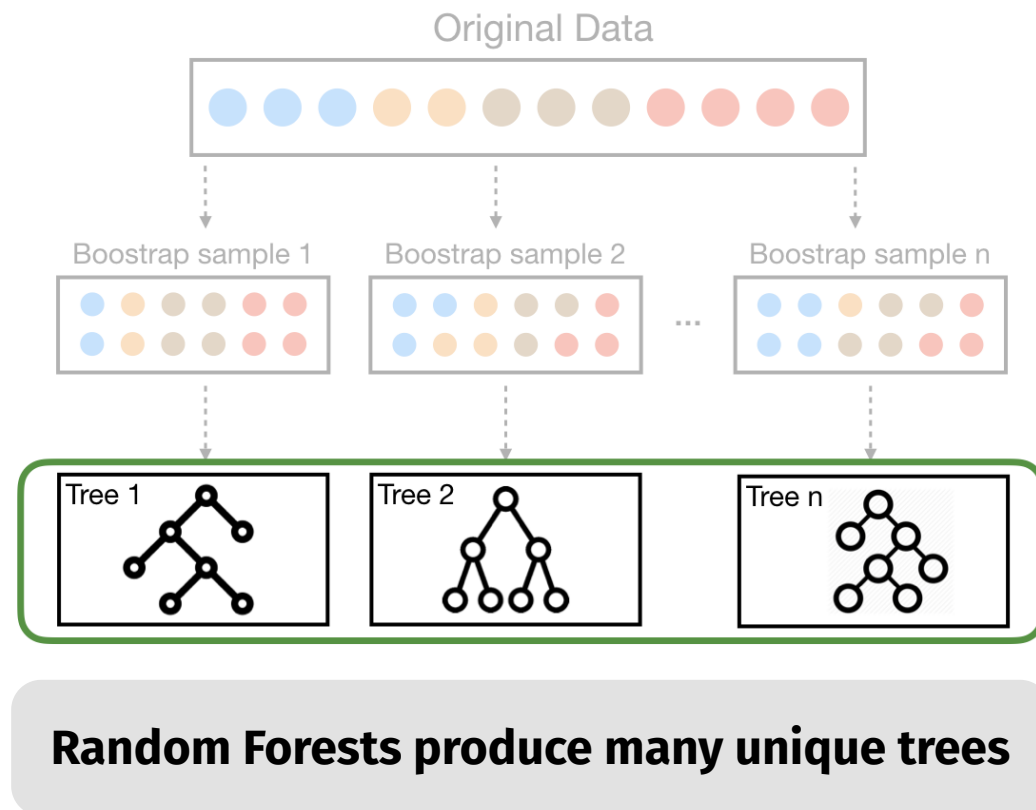
- Follow a similar bagging process but...



Idea

Split-variable randomization

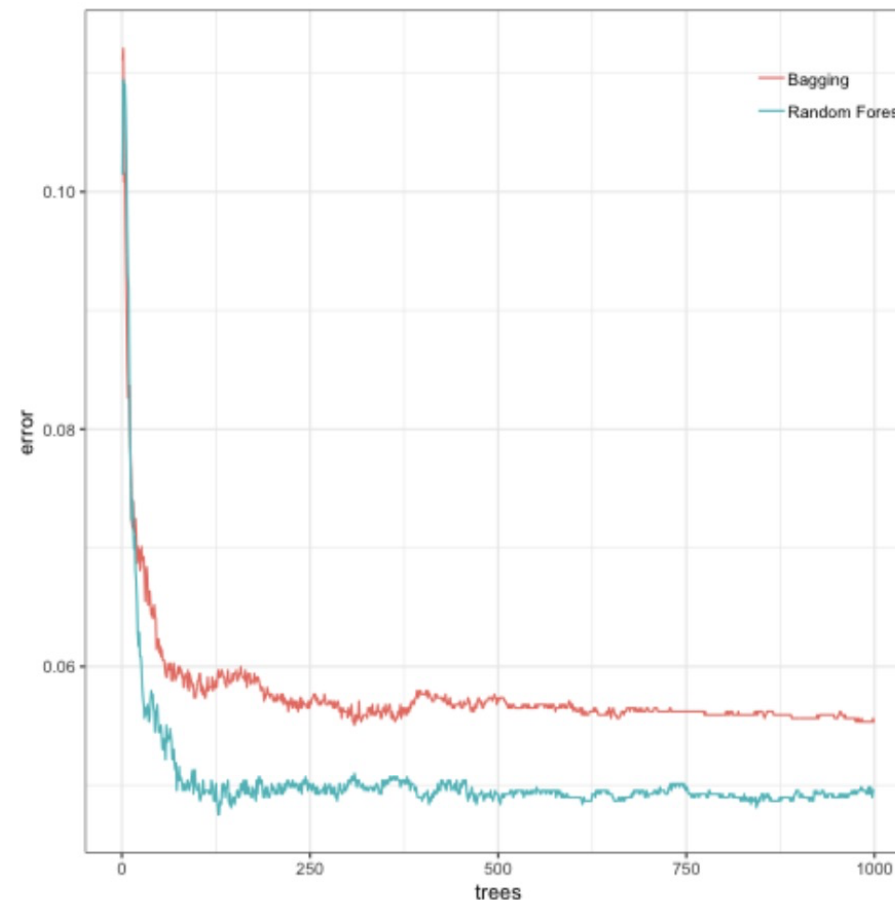
- Follow a similar bagging process but...
- each time a split is to be performed, the search for the split variable is **limited to a random subset of m of the p variables**
 - regression trees: $m = \frac{p}{3}$
 - classification trees: $m = \sqrt{p}$
 - m is commonly referred to as **mtry**



Bagging vs Random Forest

Split-variable randomization

- Follow a similar bagging process but...
- each time a split is to be performed, the search for the split variable is limited to a random subset of m of the p variables
- Bagging introduces randomness into the rows of the data
- Random forest introduces randomness into the rows and columns of the data



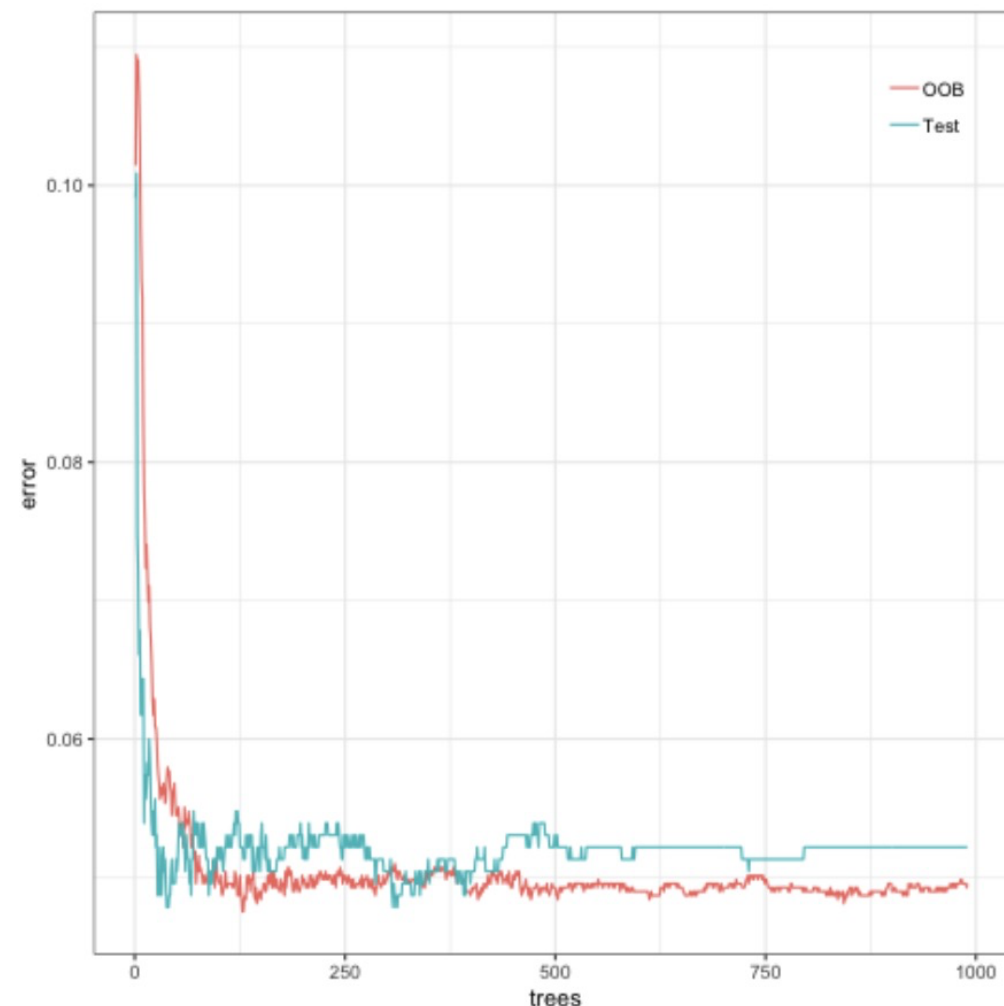
Combined, this provides a more diverse set of trees that almost always lowers our prediction error.

Out-of-bag


- For large enough N , on average, 63.21% of the original records end up in any bootstrap sample
- Roughly 36.79% of the observations are not used in the construction of a particular tree
- These observations are considered **out-of-bag (OOB)** and can be used for efficient assessment of model performance (**unstructured, but free, cross-validation**)

Pro tip:

- When N is small, OOB is less reliable than validation
- As N increases, OOB is far more efficient than k -fold CV
- When the number of trees are about 3x the number needed for the random forest to stabilize, the OOB error estimate is equivalent to leave-one-out cross-validation error.




Tuning

Random forests provide good "out-of-the-" performance but there are a few parameters we can tune to increase performance.


-

Tuning

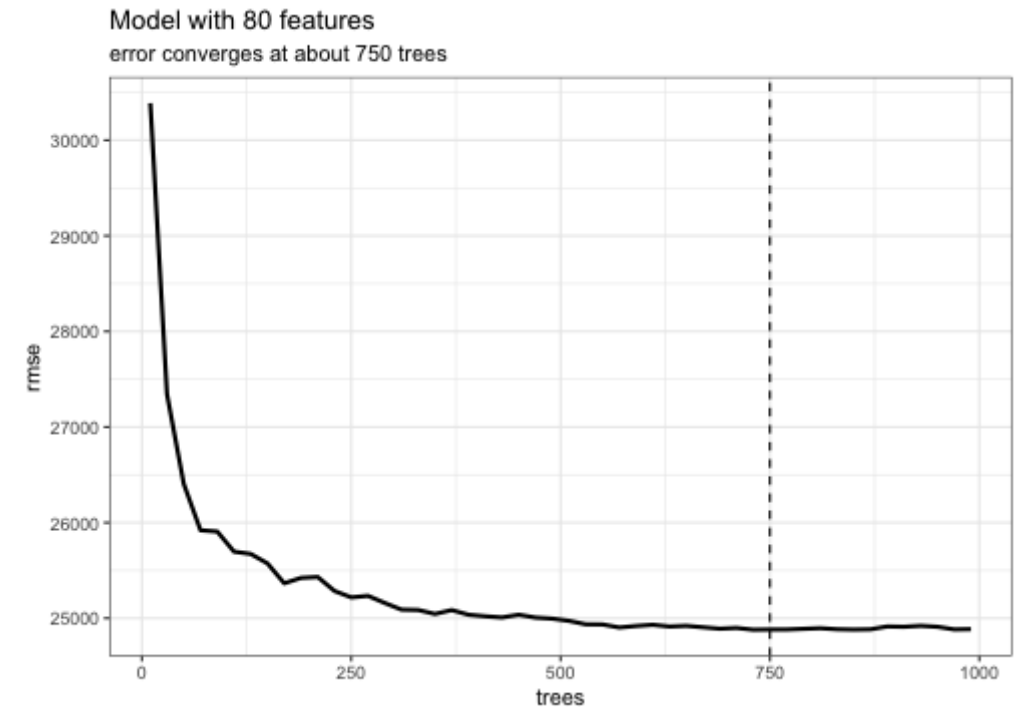
Random forests provide good "out-of-the-" performance but there are a few parameters we can tune to increase performance.

- Number of trees
- `mtry`
- Node size
- Sampling scheme
- Split rule
- Typically have the largest impact on predictive accuracy.
- Tend to have marginal impact on predictive accuracy but still worth exploring. Can also increase computational efficiency.
- Generally used to increase computational efficiency

Tuning


Random forests provide good "out-of-the-" performance but there are a few parameters we can tune to increase performance.

- **Number of trees**^a
 - **Why:** stabilize the error
 - **Rule of thumb:** start with $p \times 10$ trees and adjust as necessary
 - **Caveats:**
 - small mtry and sample size values and/or larger node size values result in less correlated trees; therefore requiring more trees to converge.
 - more trees provide more robust/stable error & variable importance measures
 - **Impact on computation time:** increases linearly with the number of trees



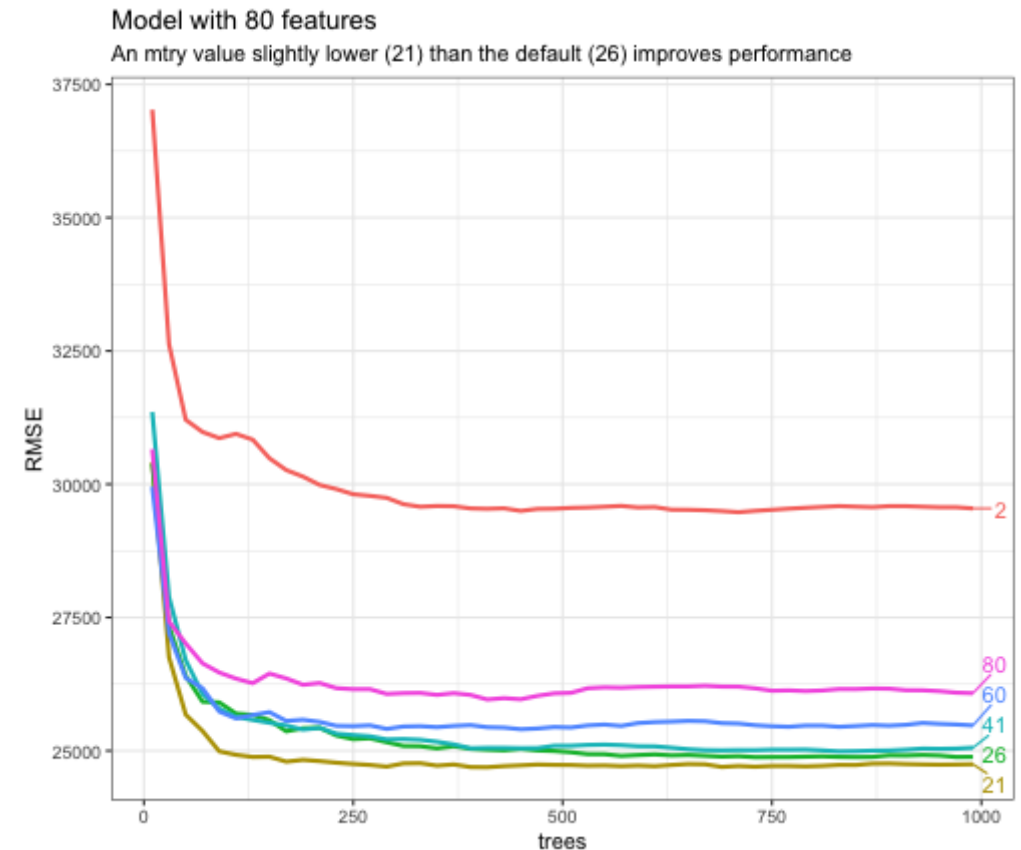
^a) Technically, the number of trees is not a real tuning parameter but it is important to have a sufficient number for the estimate to stabilize.

Tuning


Random forests provide good "out-of-the-" performance but there are a few parameters we can tune to increase performance.

- **Mtry**

- **Why:** balance low tree correlation and reasonable predictive strength
- **Rule of thumb:**
 - Regression default: $\frac{p}{3}$
 - Classification default: \sqrt{p}
 - start with 5 values evenly spaced across the range from 2 to p (include the default)
- **Caveats:**
 - few relevant predictors: \uparrow mtry
 - many relevant predictors: \downarrow mtry
- **Impact on computation time:** increases approx linearly with higher mtry values.

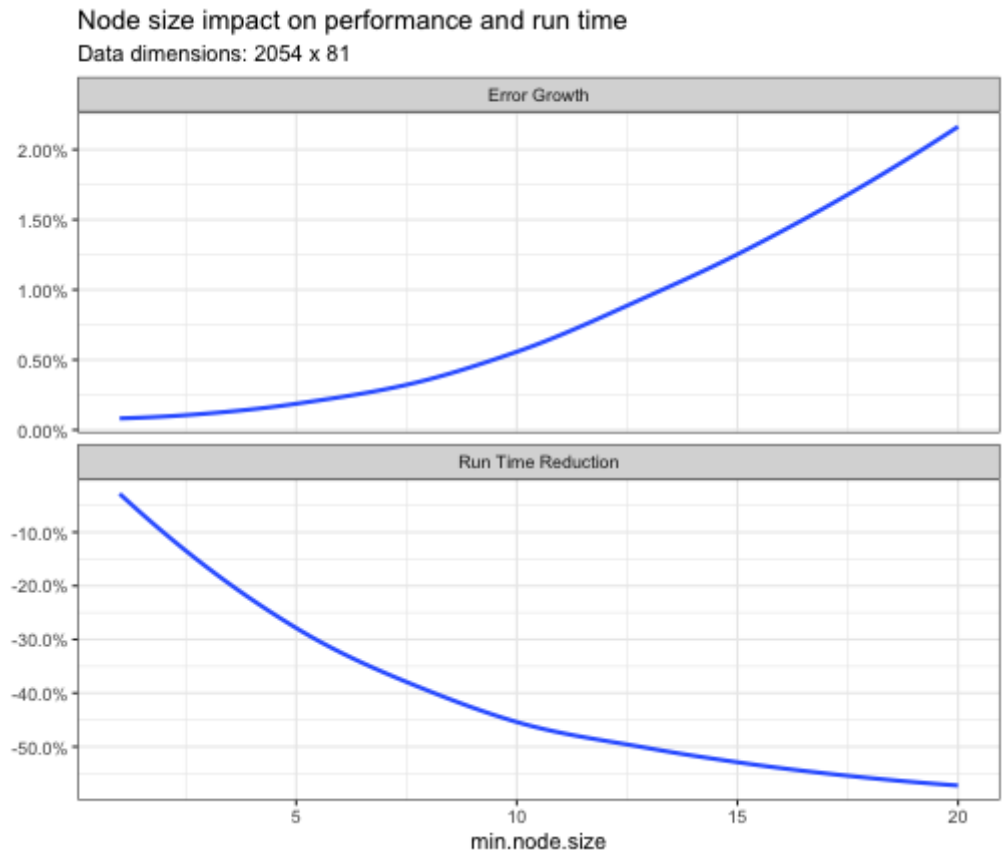


Tuning


Random forests provide good "out-of-the-" performance but there are a few parameters we can tune to increase performance.


- **Node size**

- **Why:** balance tree complexity
- **Rule of thumb:**
 - Regression default: 5
 - Classification default: 1
 - start with 3 values (1, 5, 10)
- **Caveats:**
 - many noisy predictors: **↑** node size
 - if higher mtry values are performing best, **↑** node size
- **Impact on computation time:** increases approx exponentially with small node sizes.
 - for very large data sets: **↑** node size




Tuning

Random forests provide good "out-of-the-" performance but there are a few parameters we can tune to increase performance.

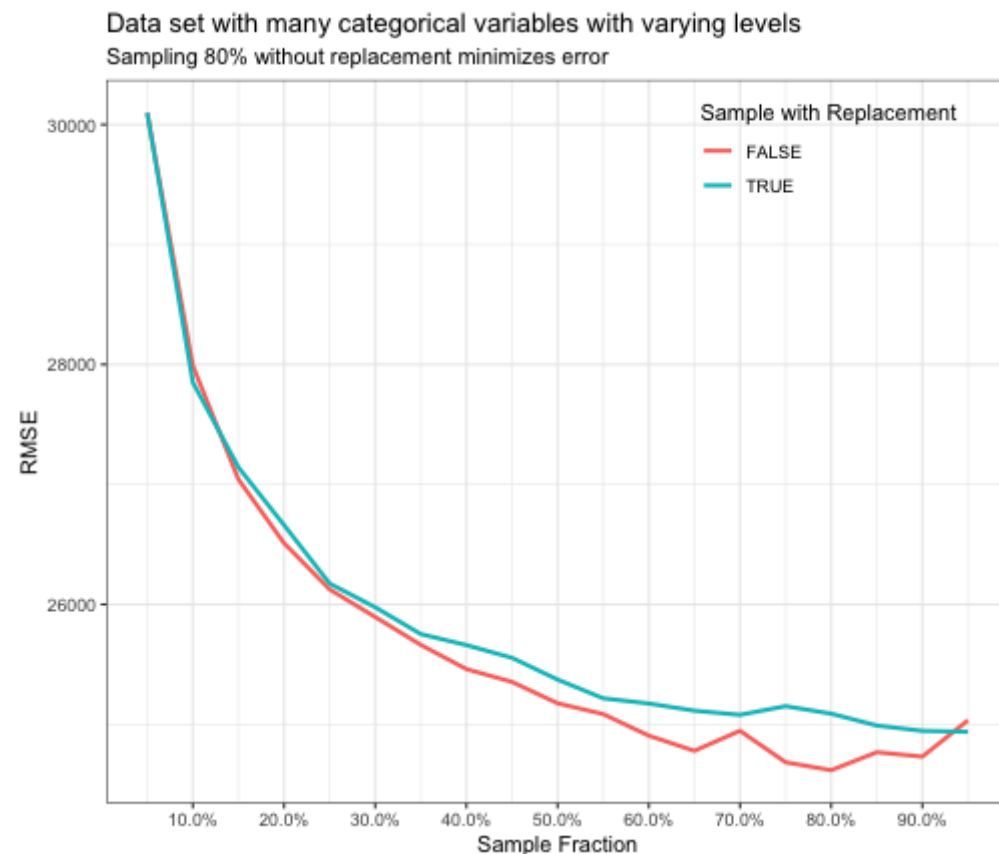
- **Node size / Required split size / Max number of nodes / Max depth**
 - Alternative parameters exist that can control tree complexity; however, most preferred random forest packages (**ranger**, **H2O**) focus on node size.
 - See  (Probst et al., 2018) for short discussion.

Tuning


Random forests provide good "out-of-the-" performance but there are a few parameters we can tune to increase performance.

- **Sampling scheme**

- **Why:** balance low tree correlation and reasonable predictive strength
- **Rule of thumb:**
 - default value is 100% with replacement
 - assess 3-4 values ranging from 25%-100%
- **Caveats:**
 - if you have dominating features - ↓ sample size to minimize tree correlation
 - if you have many categorical features with varying number of levels - try sampling without replacement
- **Impact on computation time:**
 - for very large data sets: ↓ sample size to decrease compute time



Tuning

Random forests provide good "out-of-the-" performance but there are a few parameters we can tune to increase performance.

- **Split rule**

- **Why:** Balance tree correlation and run time
- **Rule of thumb:**
 - Regression default: variance
 - Classification default: Gini / cross-entropy
- **Caveats:**
 - Default split rules favor variables with many possible splits (continuous & categorical w/many levels)
 - Try extra random tree splitting if:
 - many categorical variables with few levels
 - need to reduce run time
- **Impact on computation time:** Completely random split rule minimizes compute time since optimal split is not assessed; splits are made at random

Variable Importance

We have two approaches for [model specific variable importance](#) with random forests:

Impurity

1. At each split in each tree, compute the improvement in the split-criterion
2. Average the improvement made by each variable across all the trees that the variable is used
3. The variables with the largest average decrease in MSE are considered most important.

Notes:

- more trees lead to more stable vi estimates
- smaller mtry values lead to more equal vi estimates across all variables
- bias towards variables with many categories or numeric values

Permutation

1. For each tree, the OOB sample is passed down the tree and the prediction accuracy is recorded.
2. Then the values for each variable (one at a time) are randomly permuted and the accuracy is again computed.
3. The decrease in accuracy as a result of this randomly “shaking up” of variable values is averaged over all the trees for each variable.
4. The variables with the largest average decrease in accuracy are considered most important.

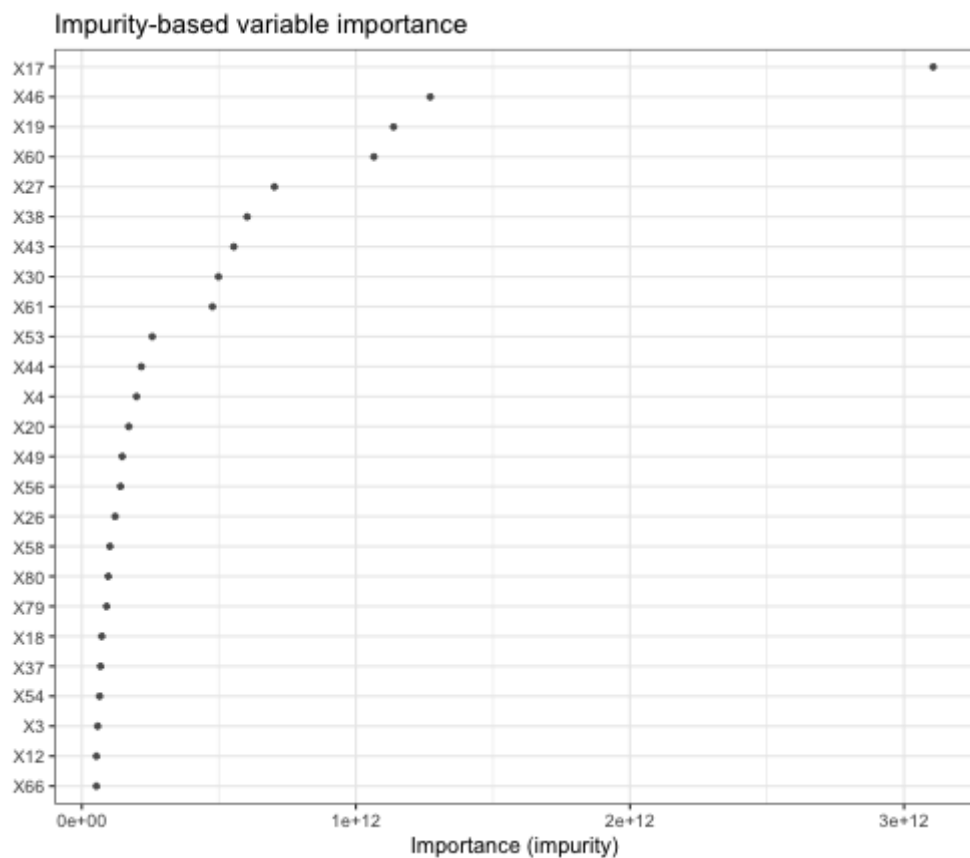
Notes:

- more trees lead to more stable vi estimates
- smaller mtry values lead to more equal vi estimates across all variables
- categorical variables with many levels can have high variance vi estimates

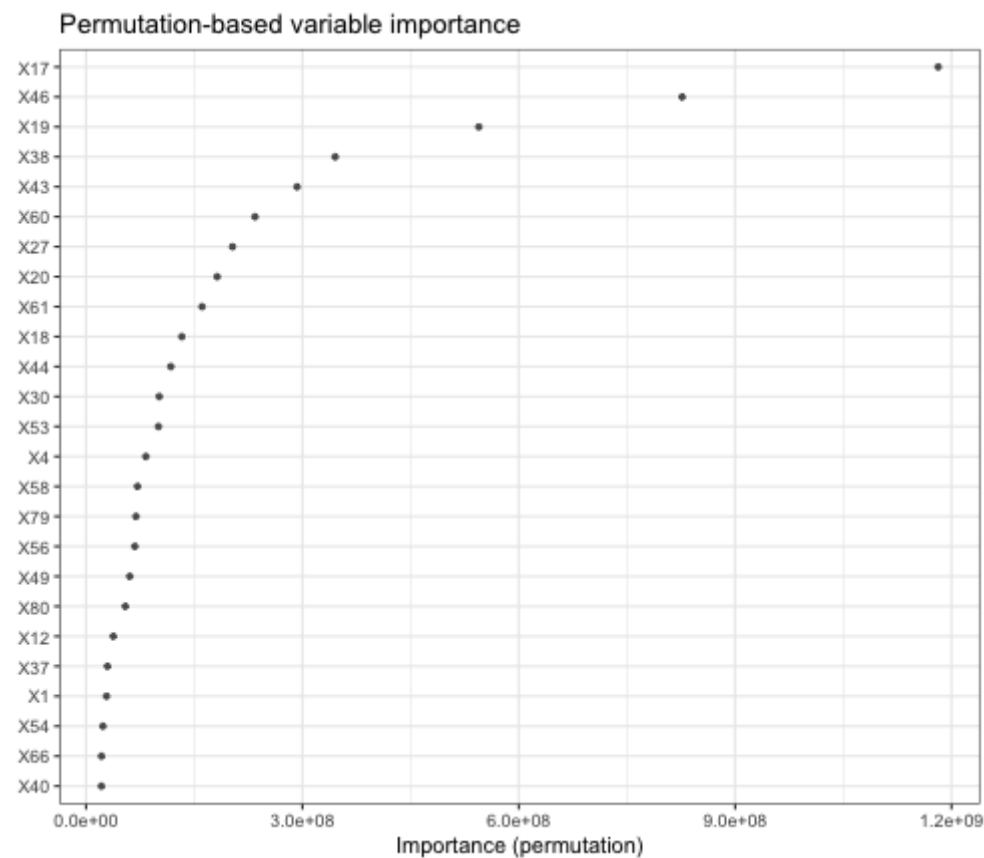
Variable Importance

The two tend to [produce similar results but with slight differences in rank order](#):

Impurity



Permutation



Random Forest Summary

Strengths 🤖

- Competitive performance.
- Remarkably good "out-of-the box" (very little tuning required).
- Built-in validation set (don't need to sacrifice data for extra validation).
- Typically does not overfit.
- Robust to outliers.
- Handles missing data (imputation not required).
- Provide automatic feature selection.
- Minimal preprocessing required.

Random Forest Summary

Strengths 🤖

- Competitive performance.
- Remarkably good "out-of-the box" (very little tuning required).
- Built-in validation set (don't need to sacrifice data for extra validation).
- Typically does not overfit.
- Robust to outliers.
- Handles missing data (imputation not required).
- Provide automatic feature selection.
- Minimal preprocessing required.

Weaknesses 🙄

- Although accurate, often cannot compete with the accuracy of advanced boosting algorithms.
- Can become slow on large data sets.
- Less interpretable (although this is easily addressed with various tools such as variable importance, partial dependence plots, LIME, etc.).