STAT 432: Basics of Statistical Learning

Tree and Random Forests

Shiwei Lan, Ph.D. <shiwei@illinois.edu>

http://shiwei.stat.illinois.edu/stat432.html

April 1, 2019

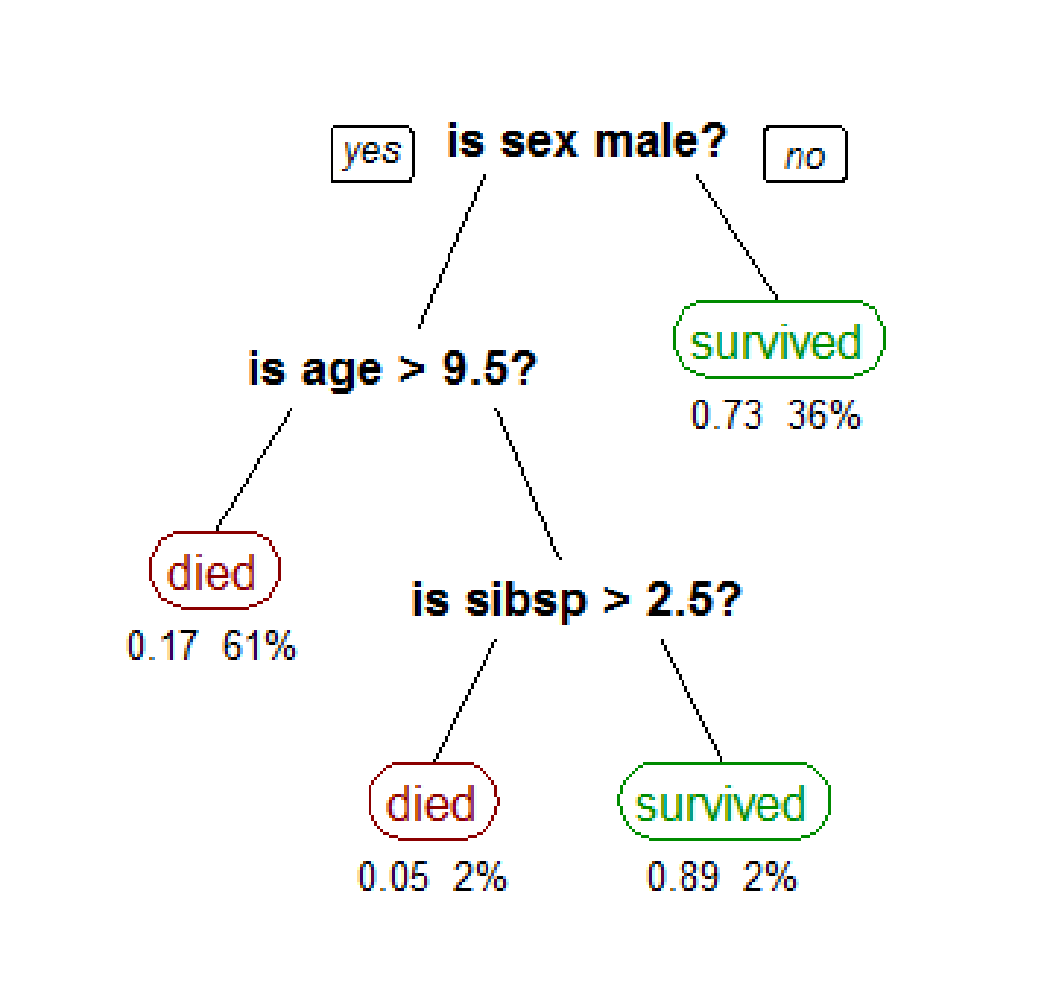
University of Illinois at Urbana-Champaign

Classification and Regression Trees (CART)

Tree-based Methods

- Tree-based methods are nonparametric methods that recursively partition the feature space into hyper-rectangular subsets, and make prediction on each subset.
- Two main streams of models:
  - Classification and regression Trees (CART): Breiman, Friedman, Olshen and Stone (1984)
  - ID3/C4.5: Quinlan (1986, 1993)
- Both are among the top algorithms in data mining (Wu et al., 2008)
- In statistics, the CART is more popular.

Titanic Survivals



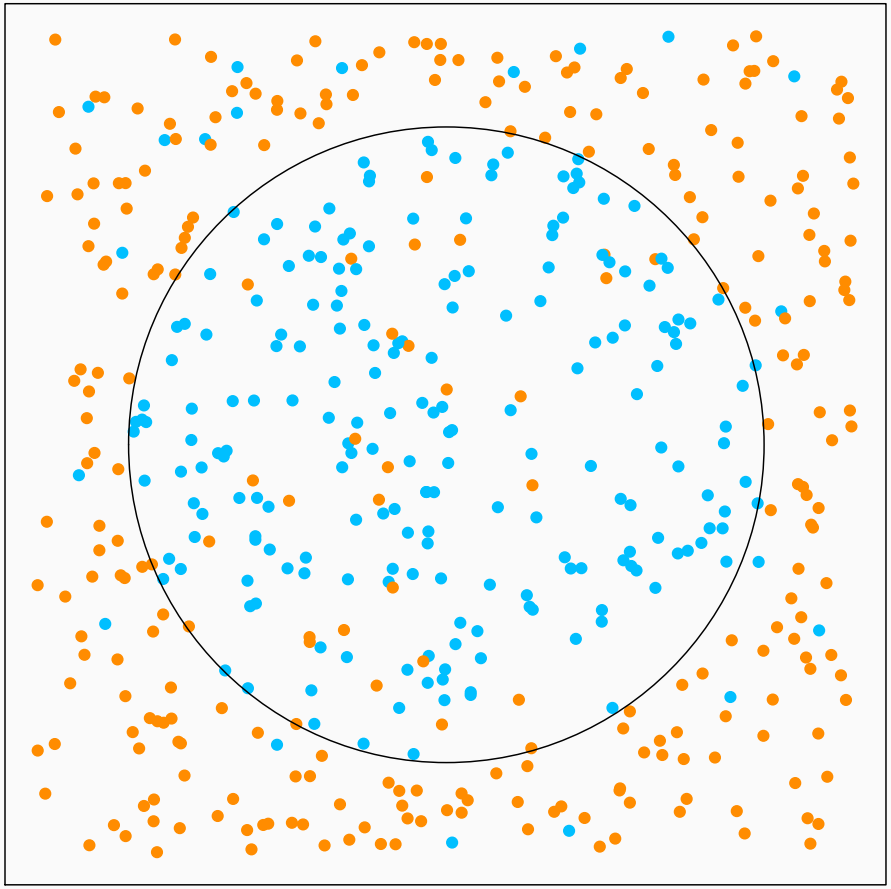
Classification and regression Trees

- Example: independent x_1 and x_2 from uniform $[-1, 1]$,

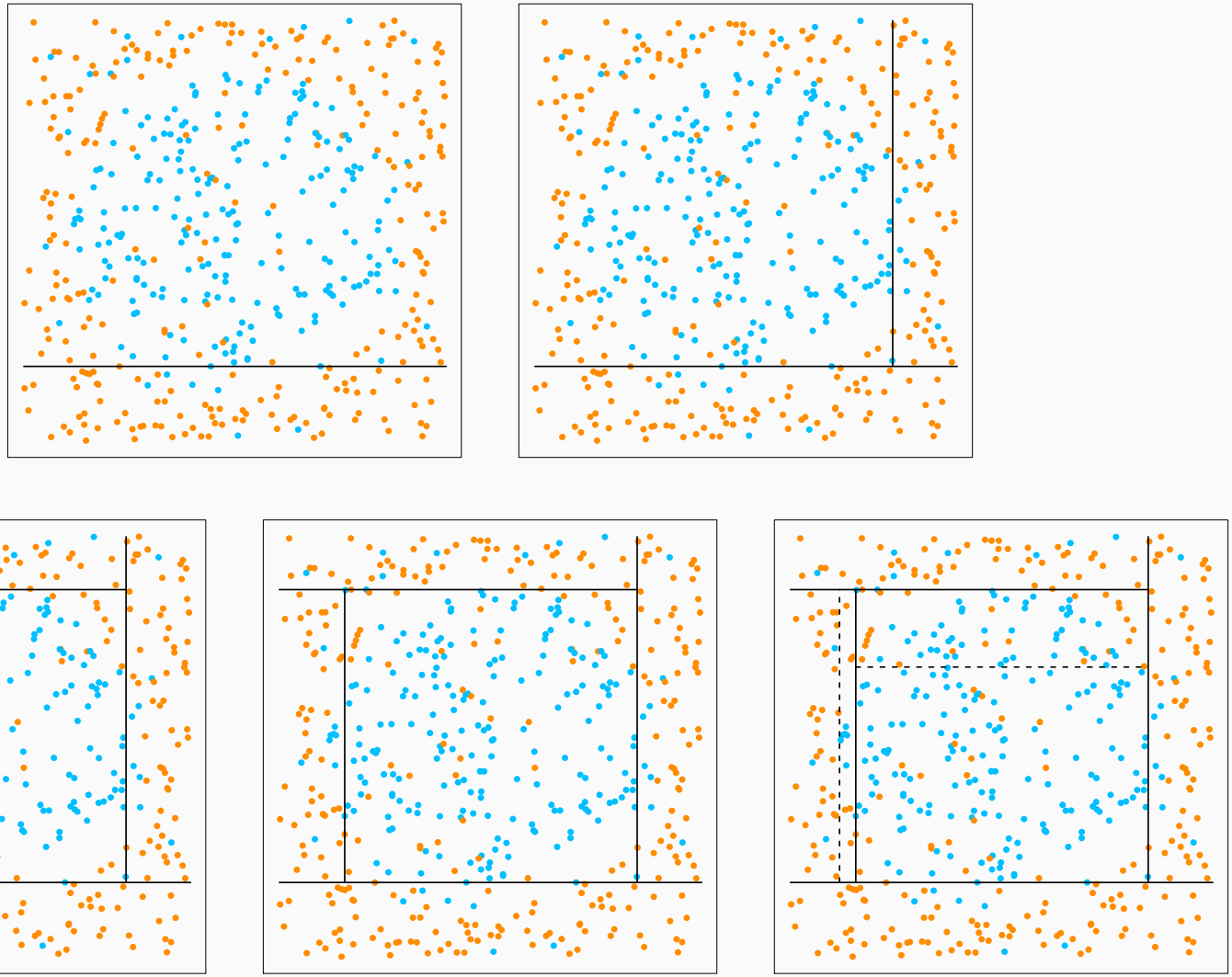
$$\mathsf{P}(Y = \text{blue} \mid x_1^2 + x_2^2 < 0.6) = 90\%$$
$$\mathsf{P}(Y = \text{orange} \mid x_1^2 + x_2^2 \geq 0.6) = 90\%$$

- Existing methods require transformation of the feature space to deal with this model. Tree and random forests do not.
- How tree works in classification?

Example



Example

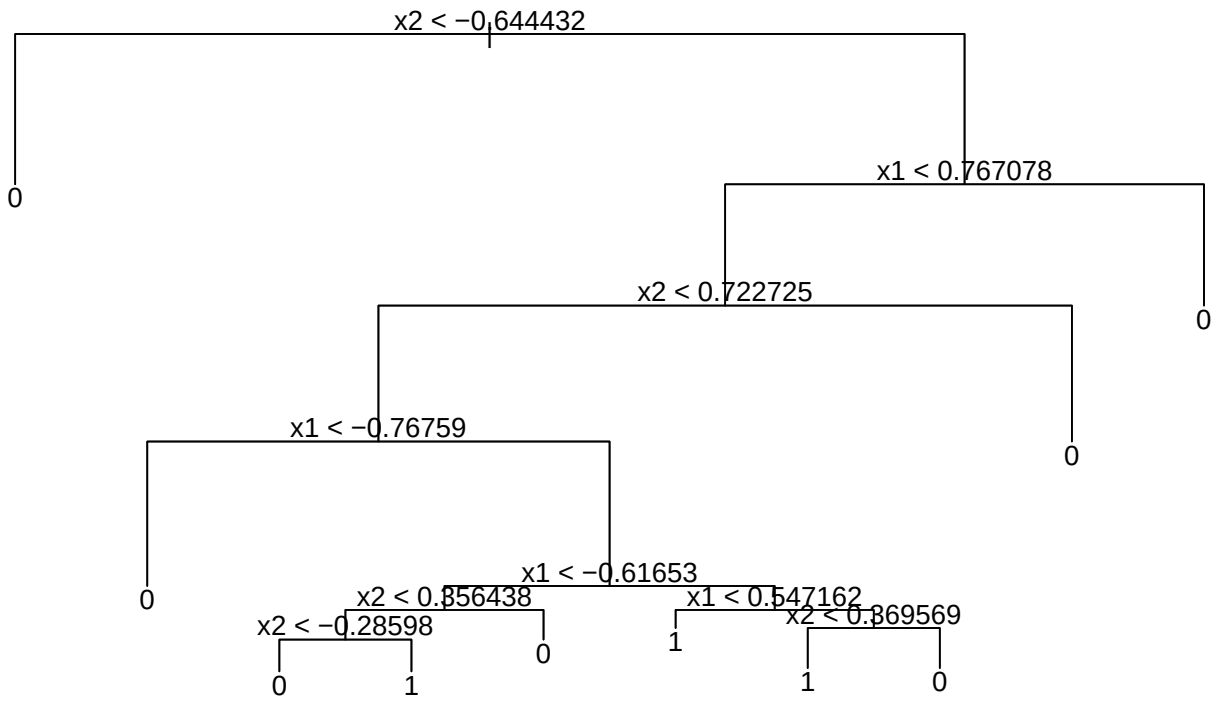


Example

- There are many popular packages that can fit a CART model: rpart, tree and party.
- Read the reference manual carefully!

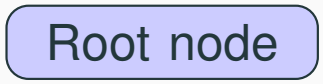
```
> library(rpart)
> x1 = runif(500, -1, 1)
> x2 = runif(500, -1, 1)
> y = rbinom(500, size = 1, prob = ifelse(x1^2 + x2^2 < 0.6, 0.9, 0.1))
> cart.fit = rpart(as.factor(y)~x1+x2, data = data.frame(x1, x2, y))
> plot(cart.fit)
> text(cart.fit)
```

Example


x2 < −0.644432
0
x1 < 0.767078
x2 < 0.722725
0
x1 < −0.76759
0
0
x1 < −0.61653
x2 < 0.356438
x1 < 0.547162
x2 < −0.28598
0
1
0
1
x2 < 0.369569
1
0

Tree Algorithm: Recursive Partitioning

- Initialized the root node: all training data



Tree Algorithm: Recursive Partitioning

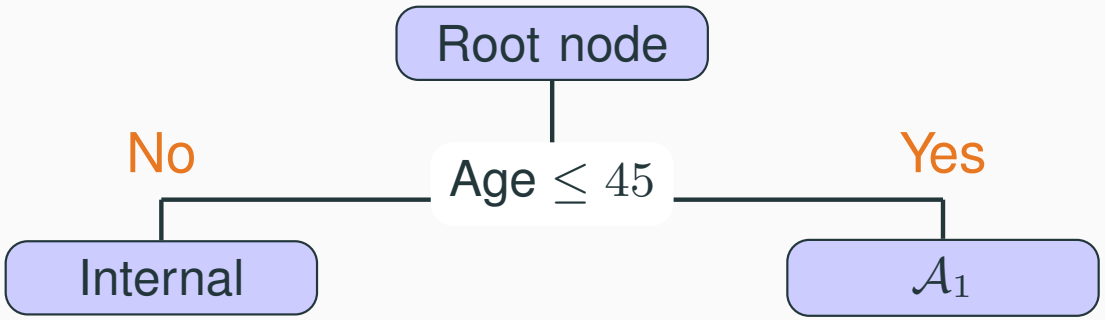
- Initialized the root node: all training data
- Find a splitting rule $\mathbf{1}\{X^{(j)} \leq c\}$ and split the node

Root node

Splitting rule $\longrightarrow$ $\mathbf{1}\{X^{(j)} \leq c\}$

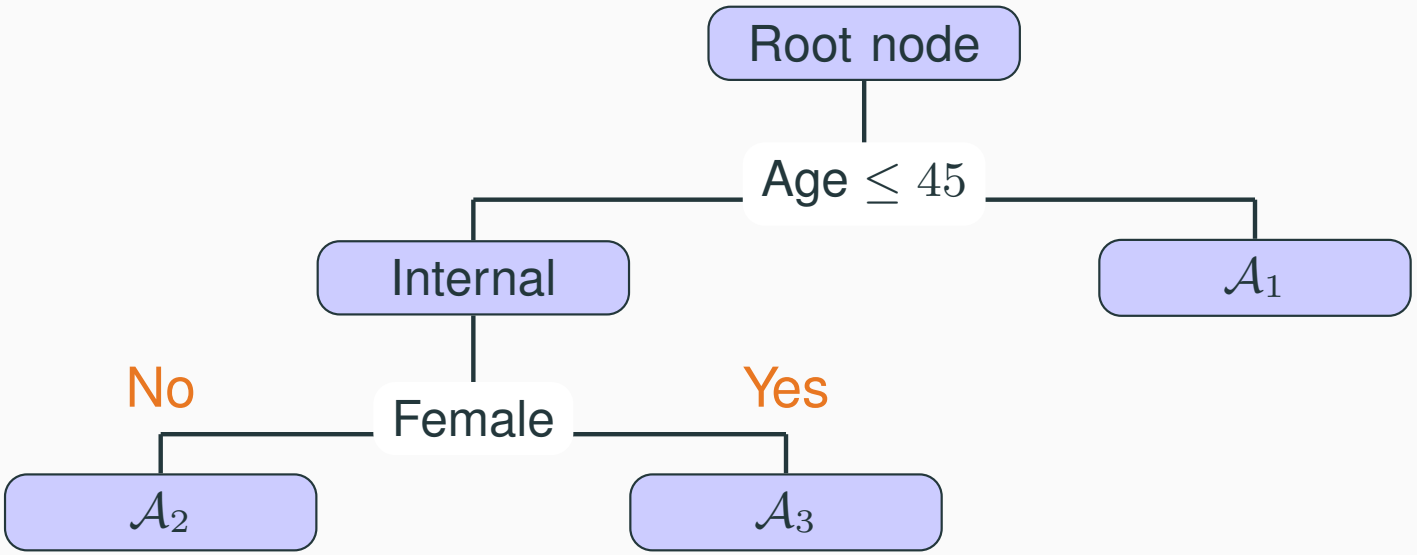
Tree Algorithm: Recursive Partitioning

- Initialized the root node: all training data
- Find a splitting rule $\mathbf{1}\{X^{(j)} \leq c\}$ and split the node
- Recursively apply the procedure on each daughter node



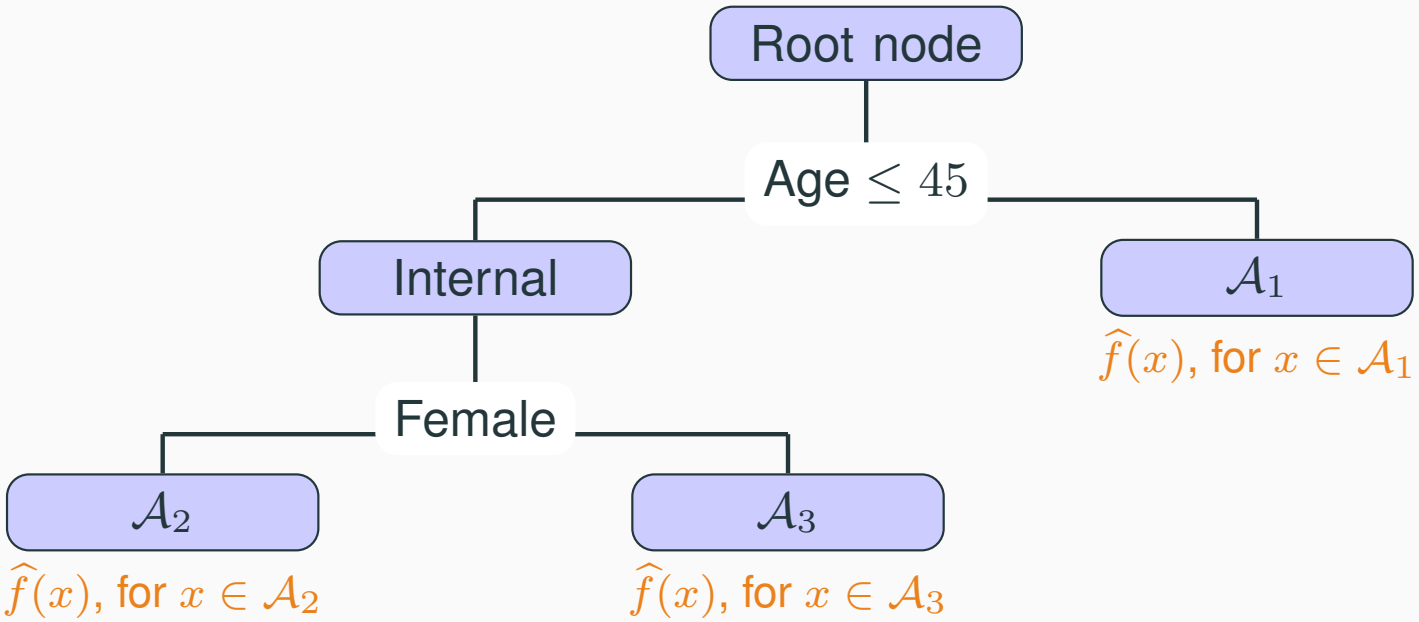
Tree Algorithm: Recursive Partitioning

- Initialized the root node: all training data
- Find a splitting rule $\mathbf{1}\{X^{(j)} \leq c\}$ and split the node
- Recursively apply the procedure on each daughter node



Tree Algorithm: Recursive Partitioning

- Initialized the root node: all training data
- Find a splitting rule $\mathbf{1}\{X^{(j)} \leq c\}$ and split the node
- Recursively apply the procedure on each daughter node
- Predict each terminal node using within-node data



Classification and Regression Trees

- How to construct the splitting rules?
  - Classification problems
  - Regression problems
- How to deal with categorical predictors?
- Tree pruning

Constructing Splitting Rules

Splitting Using Continuous Covariates

- Splitting of continuous predictors are in the form of $\mathbf{1}\{X^{(j)} \leq c\}$
- At a node $\mathcal{A}$, with $|\mathcal{A}|$ observations

$$\left\{(x_i, y_i) : x_i \in \mathcal{A}, 1 \leq i \leq n\right\}$$

- We want to split this node into two child nodes $\mathcal{A}_L$ and $\mathcal{A}_R$

$$\mathcal{A}_L = \{x \in \mathcal{A},\, x^{(j)} \leq c\}$$
$$\mathcal{A}_R = \{x \in \mathcal{A},\, x^{(j)} > c\}$$

- This is done by calculating and comparing the impurity, before and after a split.

Impurity for Classification

- We need to define the criteria for classification and regression problems separately.
- Before the split: we evaluate the impurity for the entire node $\mathcal{A}$ using the Gini index,
- Gini impurity is used as the measurement. Suppose we have K different classes,

$$\text{Gini} = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

- Interpretation: Gini = 0 means pure node (only one class), larger Gini means more diverse node.

Impurity for Classification

- After the split, we want each child node to be as pure as possible, i.e., the sum of their Gini impurities are as small as possible.
- Maximize the Gini impurity reduction after the split:

$$\text{score} = \text{Gini}(\mathcal{A}) - \frac{|\mathcal{A}_L|}{|\mathcal{A}|}\text{Gini}(\mathcal{A}_L) - \frac{|\mathcal{A}_R|}{|\mathcal{A}|}\text{Gini}(\mathcal{A}_R),$$

  where $|\cdot|$ denotes the cardinality (sample size) of a node.
- Note 1: $\text{Gini}(\mathcal{A}_L)$ and $\text{Gini}(\mathcal{A}_R)$ are calculated within their respective node.
- Note 2: An alternative (and equivalent) definition is to minimize $\frac{|\mathcal{A}_L|}{|\mathcal{A}|}\text{Gini}(\mathcal{A}_L) + \frac{|\mathcal{A}_R|}{|\mathcal{A}|}\text{Gini}(\mathcal{A}_R)$.

Impurity for Classification

- Calculating the Gini index based on the samples is very simple:
- First, for any node $\mathcal{A}$, we estimate the frequencies $\widehat{p}_k$:

$$\widehat{p}_k = \frac{\sum_i \mathbf{1}\{y_i = k\}\mathbf{1}\{x_i \in \mathcal{A}\}}{\sum_i \mathbf{1}\{x_i \in \mathcal{A}\}},$$

  which is the proportion of samples with class label k in node $\mathcal{A}$.
- Then the Gini impurity is

$$\text{Gini}(\mathcal{A}) = \sum_{k=1}^{K} \widehat{p}_k(1 - \widehat{p}_k) = 1 - \sum_{k=1}^{K} \widehat{p}_k^2$$

- Do the same for $\mathcal{A}_L$ and $\mathcal{A}_R$, then calculate the score of a split.

Choosing the Split

- To define a split $\mathbf{1}\{X^{(j)} \leq c\}$, we need to know
  - variable index j
  - cutting point c
- To find the best split at a node, we do an exhaustive search:
  - Go through each variable j, and all of its possible cutting points c
  - For each combination of j and c, calculate the score of that split
  - Compare all of such splits and choose the one with the best score
- Note: to exhaust all cutting points, we only need to examine middle points of order statistics.

Other Impurity Measures

- Gini index is not the only measurement.
  - ID3/C4.5 uses Shannon entropy from information theory

$$\text{Entropy}(\mathcal{A}) = -\sum_{k=1}^{K} \widehat{p}_k \log(\widehat{p}_k)$$

  - Misclassification error

$$\text{Error}(\mathcal{A}) = 1 - \max_{k=1,\ldots,K} \widehat{p}_k$$

- Similarly, we can use these measures to define the reduction of impurity and search for the best splitting rule

Comparing Impurity Measures

	Class 1	Class 2	$\widehat{p}_1$	$\widehat{p}_2$	Gini	Entropy	Error
$\mathcal{A}$	7	3	7/10	3/10	0.420	0.611	0.3
$\mathcal{A}_{\mathsf{L}}$	3	0	3/3	0	0	0	0
$\mathcal{A}_{\mathsf{R}}$	4	3	4/7	3/7	0.490	0.683	3/7

$$\begin{aligned}
\text{score}_{\text{Gini}} &= 0.420 - (3/10 \cdot 0 + 7/10 \cdot 0.490) = 0.077 \\
\text{score}_{\text{Entropy}} &= 0.611 - (3/10 \cdot 0 + 7/10 \cdot 0.683) = 0.133 \\
\text{score}_{\text{Error}} &= 3/10 - (3/10 \cdot 0 + 7/10 \cdot 3/7) = 0
\end{aligned}$$

Comparing Different Measures

- Gini index and Shannon entropy are more sensitive to the changes in the node probability
- They prefer to create more “pure” nodes
- Misclassification error can be used for evaluating a tree, but may not be sensitive enough for building a tree.

Regression Problems

- When the outcome Y is continuous, all we need is a corresponding impurity measure
- Use variance instead of Gini, and consider the weighted variance reduction:

$$\text{score} = \text{Var}(\mathcal{A}) - \frac{|\mathcal{A}_L|}{|\mathcal{A}|}\text{Var}(\mathcal{A}_L) - \frac{|\mathcal{A}_R|}{|\mathcal{A}|}\text{Var}(\mathcal{A}_R)$$

where for any $\mathcal{A}$, $\text{Var}(\mathcal{A})$ is just the variance of the node samples:

$$\text{Var}(\mathcal{A}) = \frac{1}{|\mathcal{A}|}\sum_{i \in \mathcal{A}}(y_i - \overline{y}_{\mathcal{A}})^2,$$

$|\mathcal{A}|$ is the cardinality of $\mathcal{A}$ and $\overline{y}_{\mathcal{A}}$ is the within-node mean.

Categorical Predictors

- If $X^{(j)}$ is a categorical variable talking values in $\{1, \ldots, C\}$, we search for a subset $\mathcal{C} \subset \{1, \ldots, C\}$, and define the child nodes

$$\mathcal{A}_L = \{x \in \mathcal{A},\, x^{(j)} \in \mathcal{C}\}$$
$$\mathcal{A}_R = \{x \in \mathcal{A},\, x^{(j)} \notin \mathcal{C}\}$$

- Maximum of $2^{C-1} - 1$ number of possible splits
- When C is too large, exhaustively searching for the best $\mathcal{C}$ can be computationally intense.
- In the R `randomForest` package C needs to be less than 53 (when C is larger than 10, this is not exhaustively searched).
- Some heuristic methods are used, such as randomly sample a subset of $\{1, \ldots, C\}$ as $\mathcal{C}$.

Overfitting and Tree Pruning

- There is a close connection with the (adaptive) histogram estimator
- A large tree (with too many splits) can easily overfit the data
  - Small terminal node $\Longleftrightarrow$ small bias, large variance
- Small tree may not capture important structures
  - Large terminal node $\Longleftrightarrow$ large bias, small variance
- Tree size is measured by the number of splits

Overfitting and Tree Pruning

- Balancing tree size and accuracy is the same as the “loss + penalty” framework
- One possible approach is to split tree nodes only if the decrease in the loss exceed certain threshold, however this can be short-sighted
- A better approach is to grow a large tree, then prune it

Cost-Complexity Pruning

- First, fit the maximum tree $\mathcal{T}_{\text{max}}$ (possibly one observation per terminal node).
- Specify a complexity penalty parameter α.
- For any sub-tree of $\mathcal{T}_{\text{max}}$, denoted as $\mathcal{T} \preceq \mathcal{T}_{\text{max}}$, calculate

$$\begin{aligned} C_\alpha(\mathcal{T}) &= \sum_{\text{all terminal nodes } \mathcal{A} \text{ of } \mathcal{T}} |\mathcal{A}| \cdot \text{Gini}(\mathcal{A}) + \alpha|\mathcal{T}| \\ &= C(\mathcal{T}) + \alpha|\mathcal{T}| \end{aligned}$$

  where $|\mathcal{A}|$ is the cardinality of node $\mathcal{A}$, $|\mathcal{T}|$ is the cardinality (number of terminal nodes) of tree T.
- Find $\mathcal{T}$ that minimizes $C_\alpha(T)$
  - Large α gives small trees
  - Choose α using CV (or plot)

Missing Values

- If each variable has 5% chance to have missing value, then if we have 50 variables, there are only 7.7% of the samples that has compete measures.
- Traditional approach is to discard observations with missing values, or impute them
- Tree-based method can handle them by either putting them as a separate category, or using surrogate variables whenever the splitting variable is missing.

Remark

- Advantages of tree-based method:
  - handles both categorical and continuous variables in a simple and natural way
  - Invariant under all monotone transformations of variables
  - Robust to outliers
  - Flexible model structure, capture iterations, easy to interpret
- Limitations
  - Small changes in the data can result in a very different series of splits
  - Non-smooth. Some other techniques such as the multivariate adaptive regression splines (MARS, Friedman 1991) can be used to generate smoothed models.

Random Forests

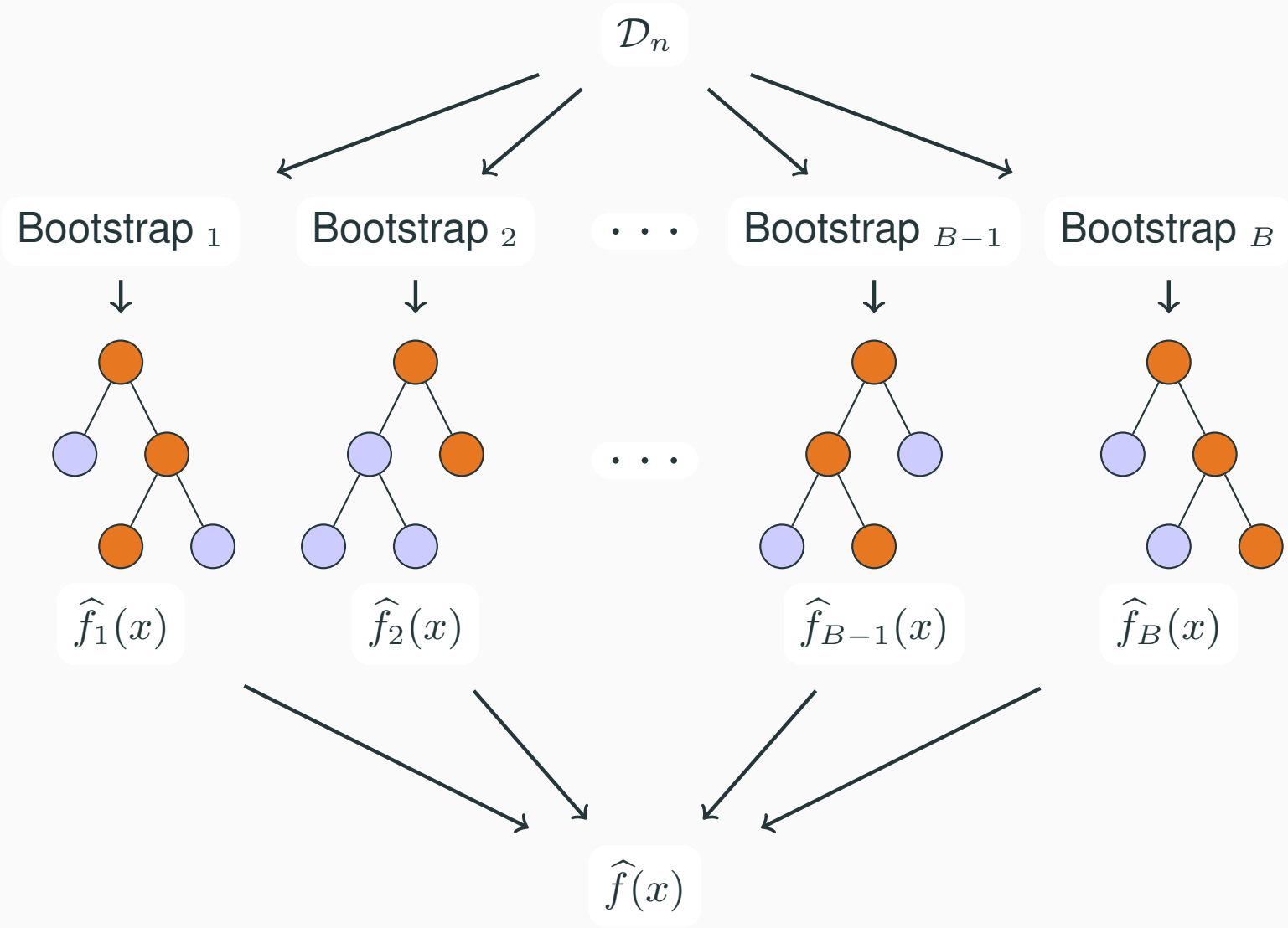
Weak and Strong Learners

- Back in the mid-late 90's, researches started to investigate whether aggregated "weak learners" (unstable, less accurate) can be a "strong learner".
- Bagging, boosting, and random forests are all methods along this line.
- Bagging and random forests learn individual trees with some random perturbations, and "average" them.
- Boosting progressively learn models with small magnitude, then "add" them
- In general, Boosting, Random Forests $\succ$ Bagging $\succ$ Single Tree.

Bagging Predictors

- Bagging stands for "Bootstrap aggregating"
- Draw B bootstrap samples from the training dataset, fit CART to each of them, then average the trees
- "Averaging" is symbolic, what we really do is to get the predictions from each tree, and average the predicted values.
- Motivation: CART is unstable, however, perturbing and averaging can improve stability and leads to better accuracy

Ensemble of Trees



Bagging Predictors

- Bootstrap sample with replacement. Fit a CART model to each bootstrap sample (may require pruning for each tree).
- To combine the bootstrap learners, for classification:

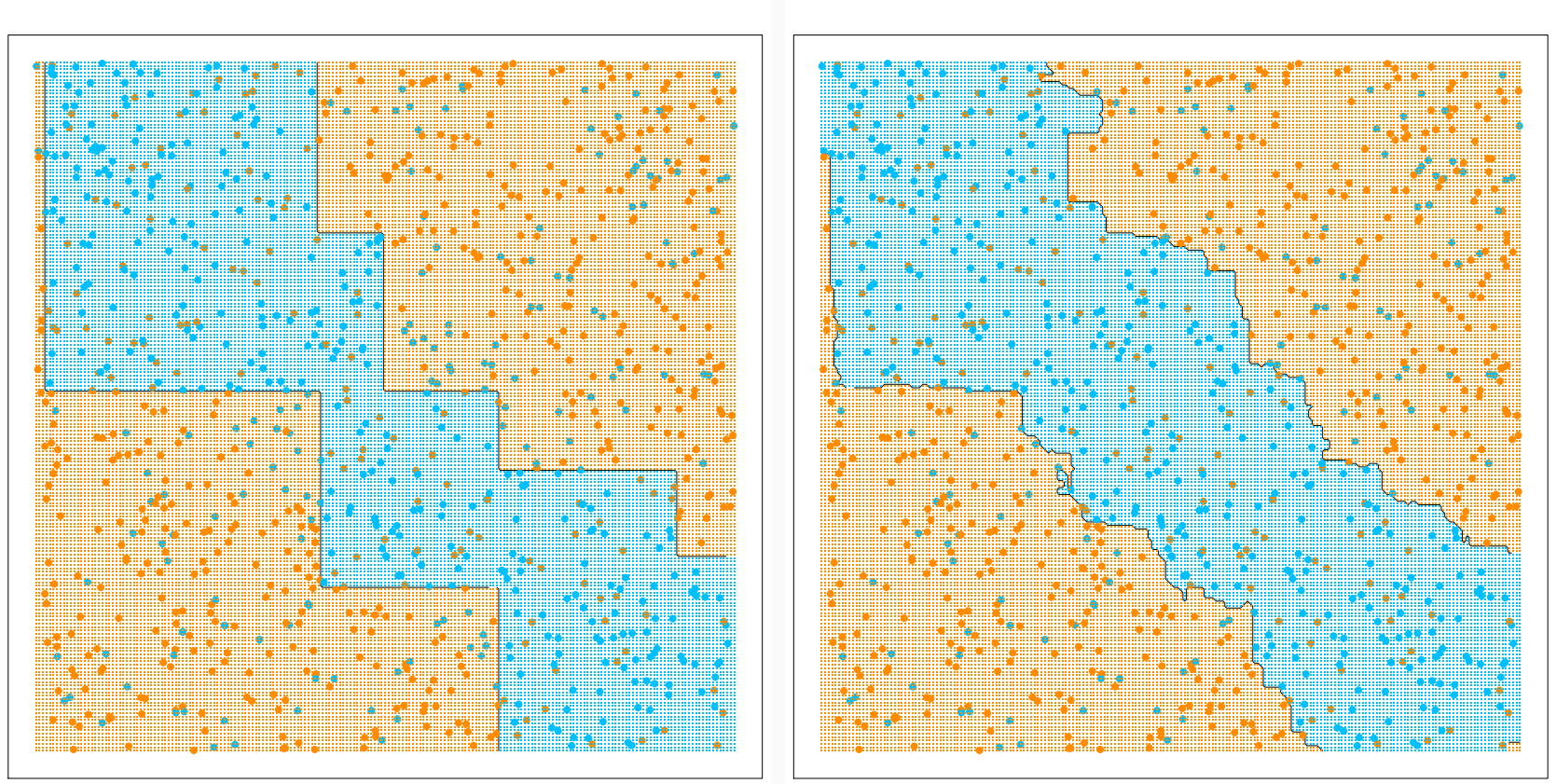
$$\widehat{f}_{\text{bagging}}(x) = \text{Majority Vote}\{\widehat{f}_b(x)\}_{b=1}^{B},$$

  and for regression:

$$\widehat{f}_{\text{bagging}}(x) = \frac{1}{B}\sum_{b=1}^{B}\widehat{f}_b(x),$$

- Dramatically reduce the variance of individual learners
- CART can be replaced by other weak learners

CART vs. Bagging



CART vs. Bagging

Remarks about Bagging

- Why Bagging works?
- Averaging (nearly) independent copies of $\widehat{f}(x)$ can lead to reduced variance
- The “independence” is introduced by bootstrapping
- However, the simple structure of trees will be lost due to averaging, hence it is difficult to interpret

Remarks about Bagging

- But, the performance of bagging in practice is oftentimes not satisfactory. Why?
- Its not really independent...
- Different trees have high correlation which makes averaging not very effective
- How to further de-correlate trees?

Random Forests

- Several articles came out in the late 90's discussing the advantages of using random features, these papers greatly influenced Breiman's idea of random forests.
- For example, in Ho (1998), each tree is constructed using a randomly selected subset of features
- Random forests take a step forward: at each splitting rule we consider a random subset of features
- Important tuning parameters: `mtry` and `nodesize`

Tuning Parameter: `mtry`

- An important tuning parameter of random forests is **`mtry`**
- At each split, randomly select **`mtry`** variables from the entire set of features $\{1, \ldots, p\}$
- Search for the best variable and the splitting point out of these **`mtry`** variables
- Split and proceed to child nodes

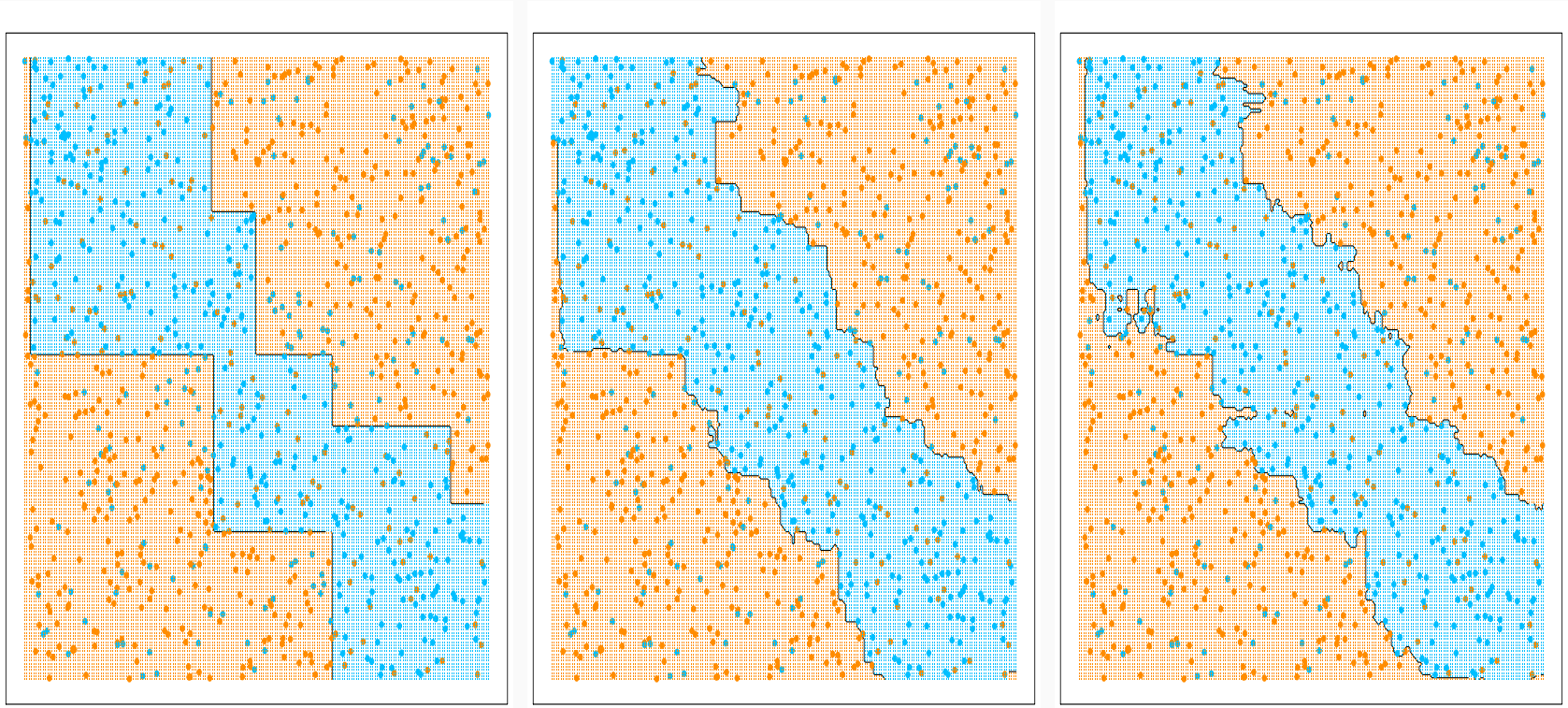
Tuning Parameter: `nodesize`

- Another important tuning parameter is (terminal) **`nodesize`**
- Random forests do not perform pruning!
- Instead, splitting does not stop until the terminal node size is less or equal to **`nodesize`**, and the entire tree is used.
- **`nodesize`** controls the trade-off between bias and variance in each tree, same as k in kNN
- In the most extreme case, $n_{min} = 1$ means exactly fit each observation, but this is not 1NN!

Tuning parameters

- A summary of important tuning parameters in Random forests (using R package randomForest)
  - ntree : number of trees, set it to be large. Default 500.
  - mtry : number of variables considered at each split. Default $p/3$ for regression, $\sqrt{p}$ for classification.
  - nodesize : terminal node size. Default 5 for regression, 1 for classification
  - sampsize : Bootstrap sample size, usually n with replacement.
- Overall, tuning is very crucial in random forests

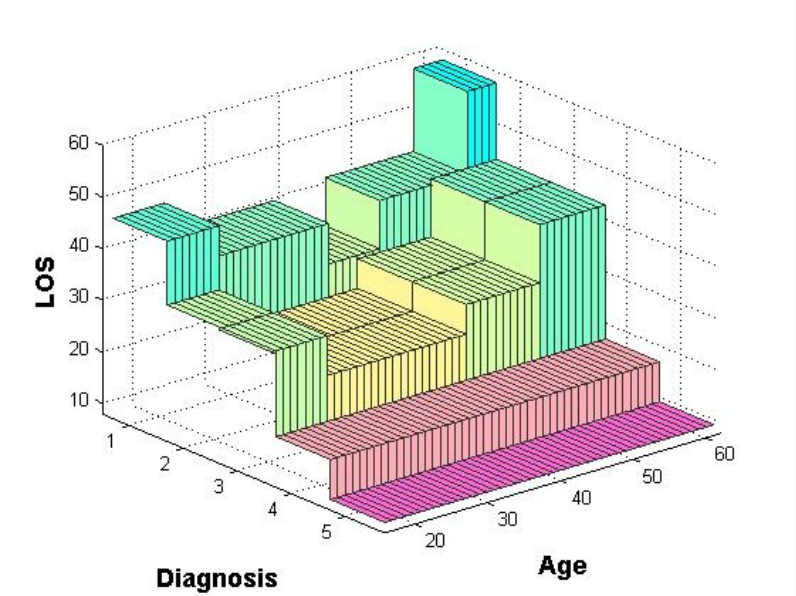
CART vs. Bagging vs. RF



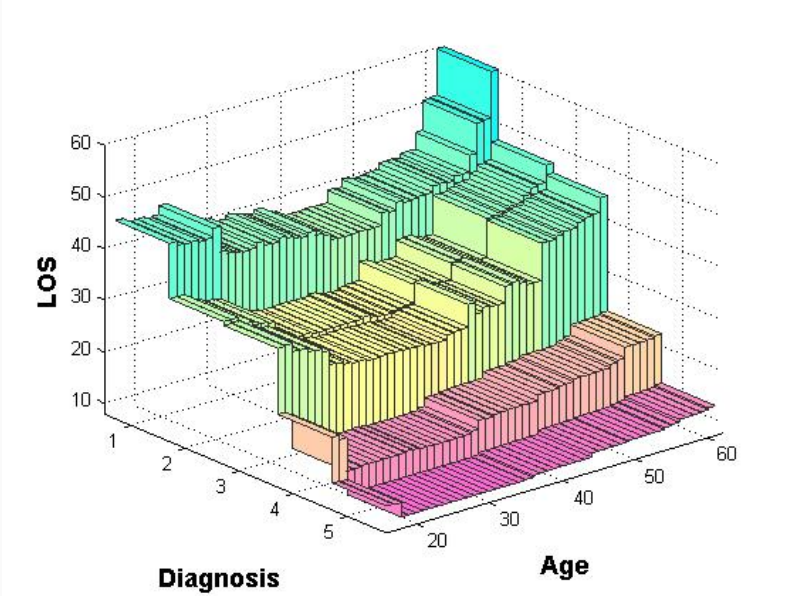
CART Bagging RF

RF: ntree = 1000, mtry = 1, nodesize = 25

Smoothness Effect of Random Forests



CART



RF

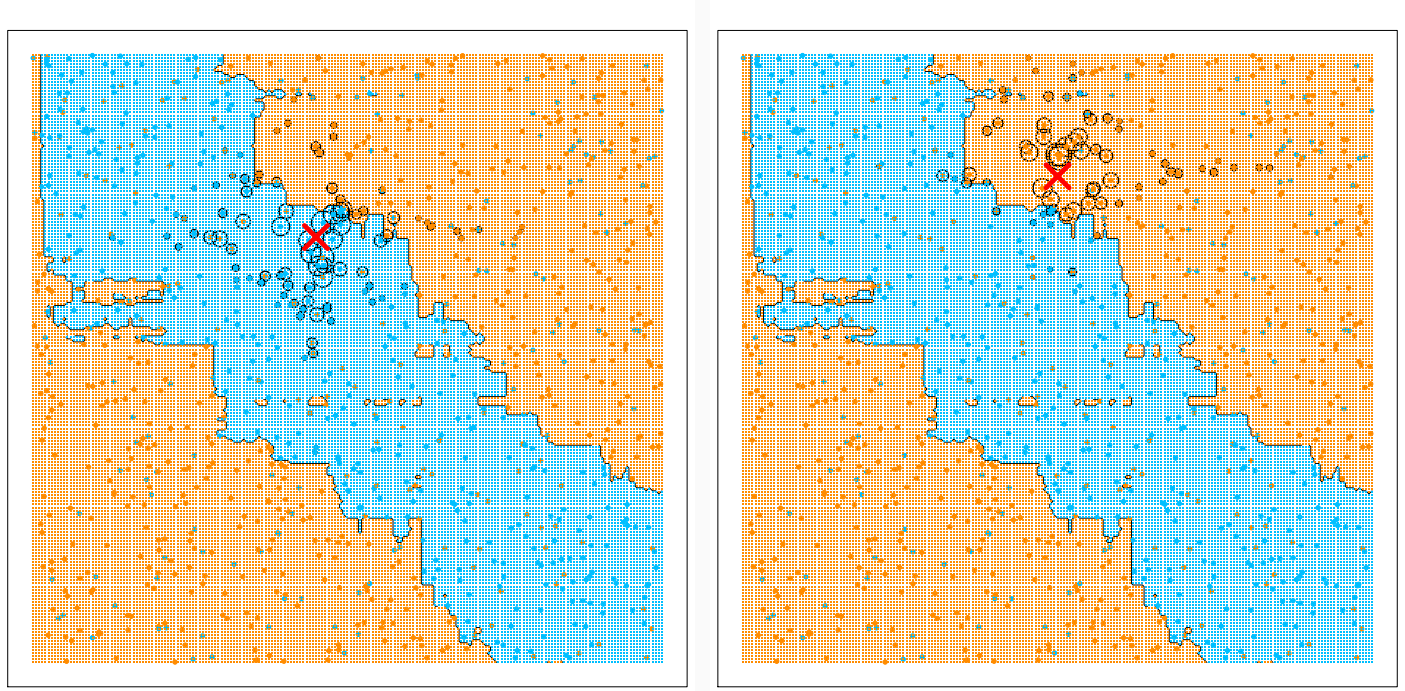
Smoothness Effect of Random Forests
(Age: continuous; Diagnosis: categorical)

Random Forests vs. Kernel

Random Forests vs. Kernel

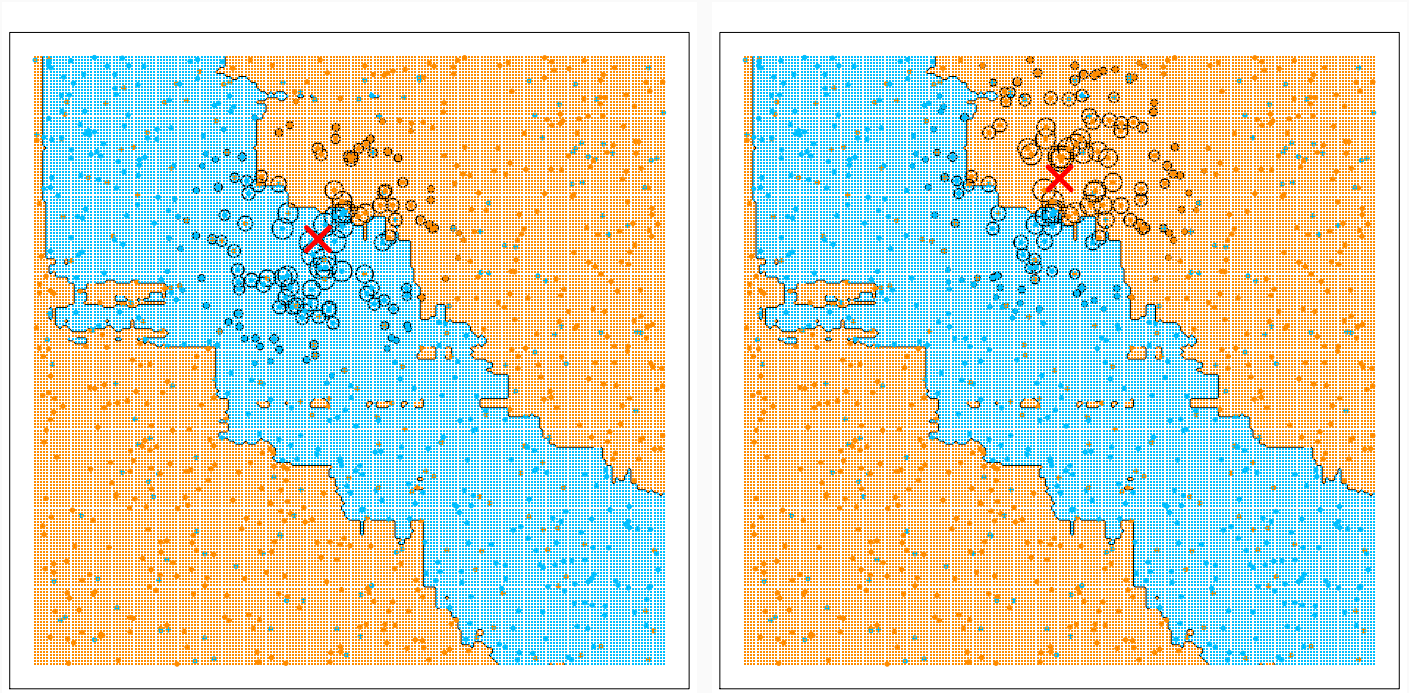
- Random forests are essentially kernel methods
- However, the distance used in random forests is adaptive to the true underlying structure
- This can be seen from the kernel weights derived from a random forests

RF vs. Kernel



Random forest kernel at two different target points

RF vs. Kernel



Gaussian Kernel at two different target points

Variable Importance

Variable Importance

- Random forests has a built-in variable selection tool: variable importance
- Variable importance utilizes samples that are not selected by bootstrapping (out-of-bag data):
  - For the b-th tree, use the corresponding out-of-bag data as the testing set to obtain the prediction error: Err_0^b
  - For each variable j, randomly permute its value among the testing samples, and recalculate the prediction error: Err_j^b
  - calculate for each j

$$\text{VI}_{bj} = \frac{\text{Err}_j^b}{\text{Err}_0^b} - 1$$

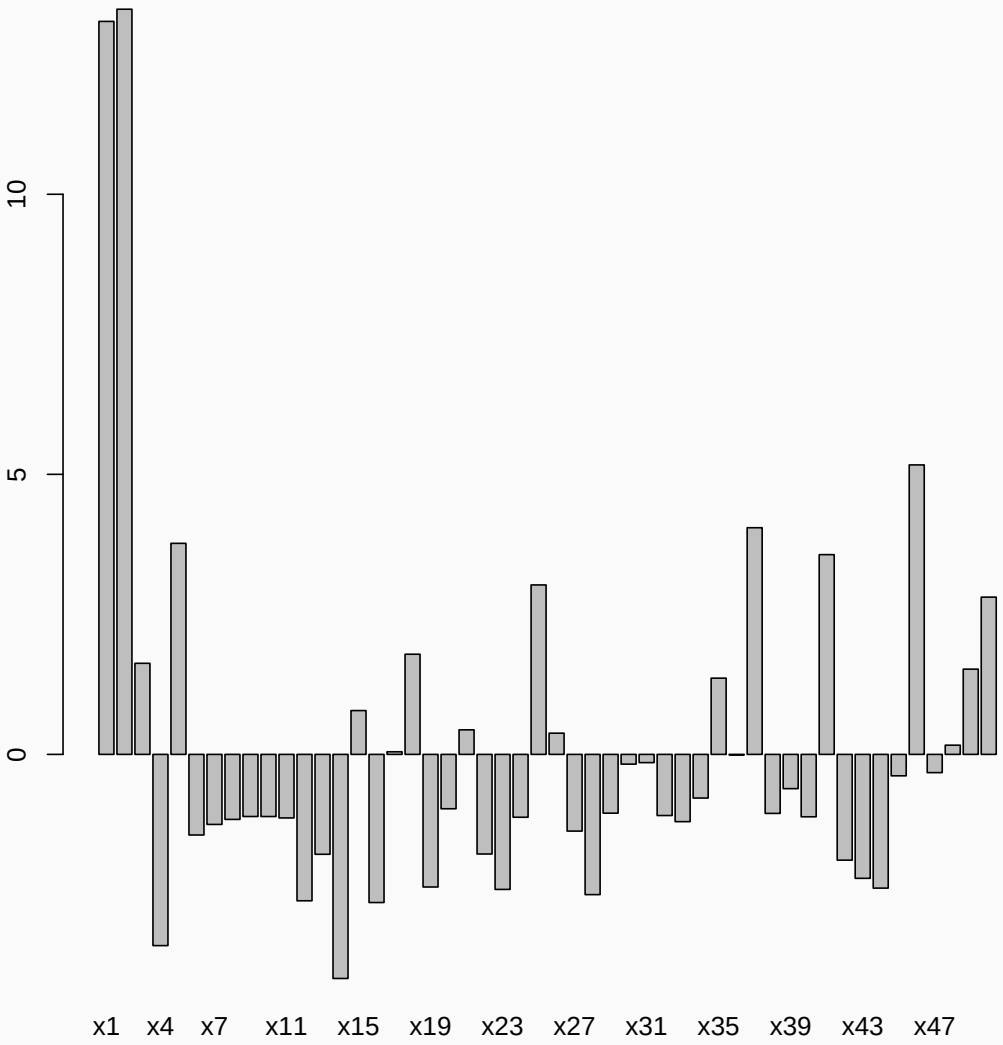
  - Average VI_{bj} across all trees

$$\text{VI}_j = \sum_{b=1}^{B} \text{VI}_{bj}$$

Variable Importance

- This essentially works like a cross-validation:
  - the in-bag samples are training samples,
  - the out-of-bag samples are testing samples
  - a bootstrapped cross-validation
- Usually the misclassification error is used instead of Gini index
- Higher VI means larger loss of accuracy due to the loss of information on $X^{(j)}$, hence more important.

Variable Importance in RF



Same simulation setting as the “circle” example, with additional 48 noise variables.

Remarks about Random Forests

- Performs well on high-dimensional data
- Tuning parameters are crucial
- Difficult to interpret
- Adaptive kernel