# STAT5003

Week 9 : Tree and Ensemble methods

Jaslene Lin
*The University of Sydney*

THE UNIVERSITY OF SYDNEY

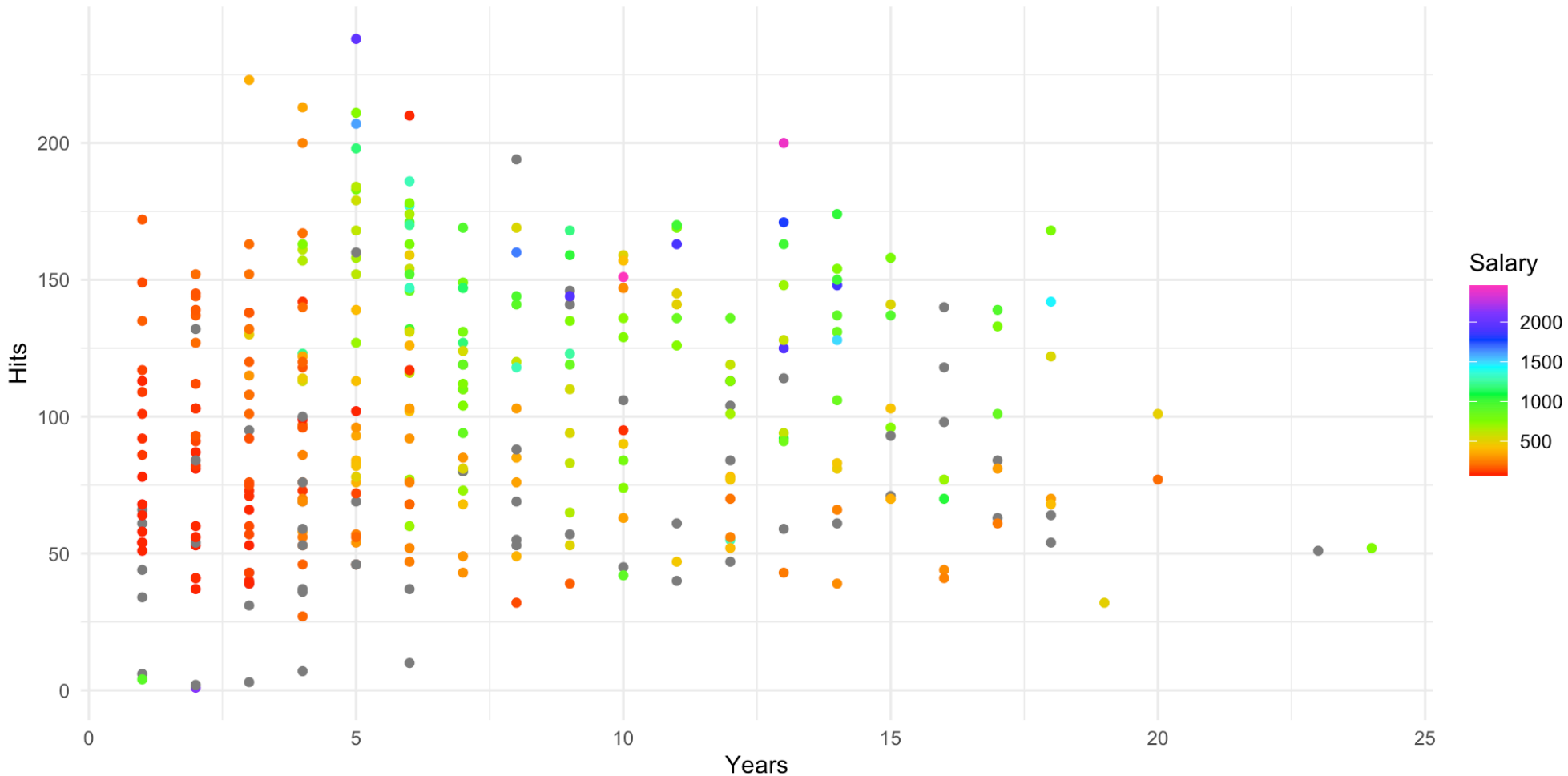
# Readings and R functions covered

**Important**

- **Introduction to Statistical Learning**
  - Tree methods covered in Chapter 8
- **R** functions
  - `tree::tree` (Create Regression and Decision trees)
  - `base::sample` (Bagging)
  - `ranger::ranger` (construct random forests)
  - `gbm::gbm` (Gradient boosting machines)

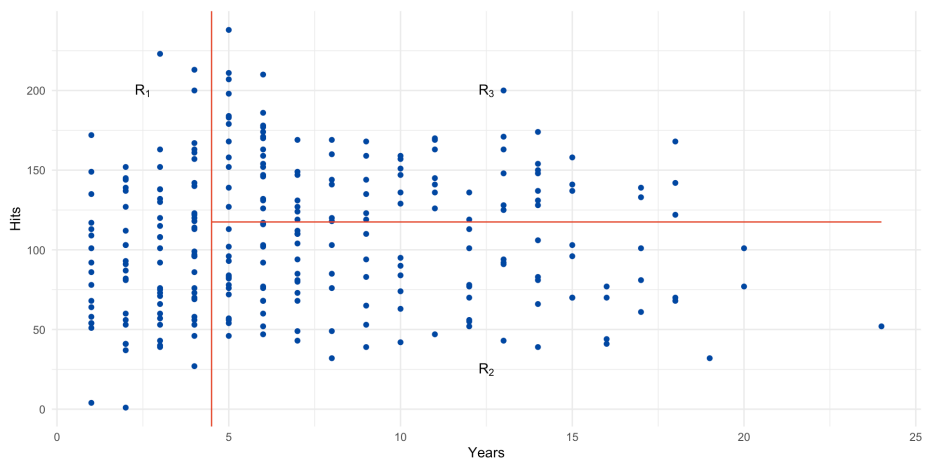
# Decision Tree for Regression

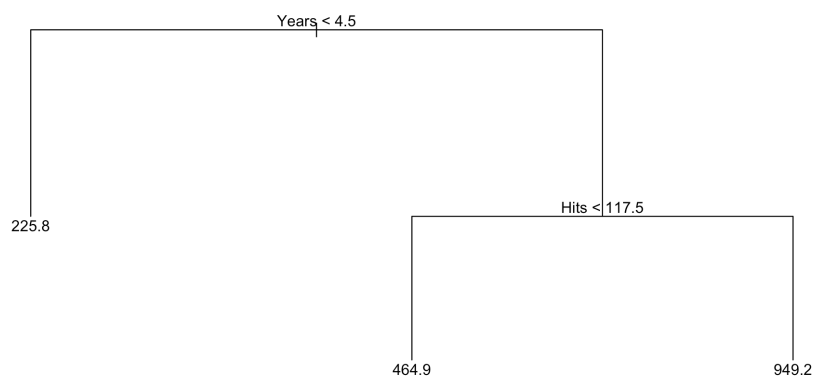
- Baseball player salary data: how would you stratify it?
  - **Salary** is colour-coded (`NA` values coded grey).



# Decision Tree for Regression

- Overall, the tree segments the players into three regions of predictor space:
  - $R_1 = \{X \mid \text{Years} < 4.5\}$
  - $R_2 = \{X \mid \text{Years} \geq 4.5,\ \text{Hits} < 117.5\}$
  - $R_3 = \{X \mid \text{Years} \geq 4.5,\ \text{Hits} \geq 117.5\}$

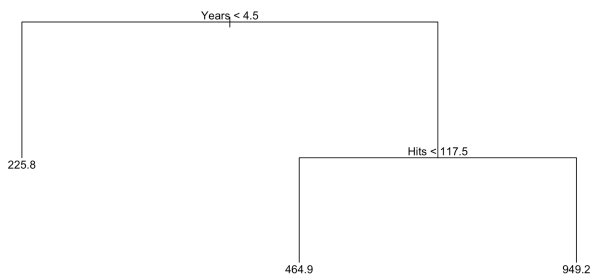


# Terminology for trees

- In keeping with the tree analogy:
  - The regions $R_1$, $R_2$, and $R_3$ are known as terminal nodes.
  - Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
  - The points along the tree where the predictor space is split are referred to as internal nodes.
- In the Baseball player salary tree, the two internal nodes are indicated by **Years** $< 4.5$ and **Hits** $< 117.5$.

# Interpretation of Results




- **Years** is the most important factor in determining **Salary**:
  - ➟ Players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced:
  - ➟ The number of **Hits** that he made in the previous year seems to play little role in his **Salary**.
- Among players who have been in the major leagues for five or more years:
  - ➟ The number of Hits made in the previous year does affect Salary.
  - ➟ Players who made more Hits last year tend to have higher salaries.
- Obviously an over-simplification:
  - ➟ Compared to some other classification models (such as a regression model), it is easy to display, interpret, and explain.
  - ➟ Even easier than linear regression; you can interpret the decision tree to your grandparents

# Details on tree building process

- In theory, the decision regions could have any shape.
- However, we choose to divide the predictor space into **high-dimensional rectangles, or boxes**, for simplicity and for ease of interpretation of the resulting predictive model
- The goal is to find boxes $R_1, \ldots, R_J$ that minimizes the residual sum of squares (RSS), given by:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  - $\hat{y}_{R_j}$ is the mean response for the training observations within the $j^{\text{th}}$ box

# Tree Building via Recursive Binary Splitting

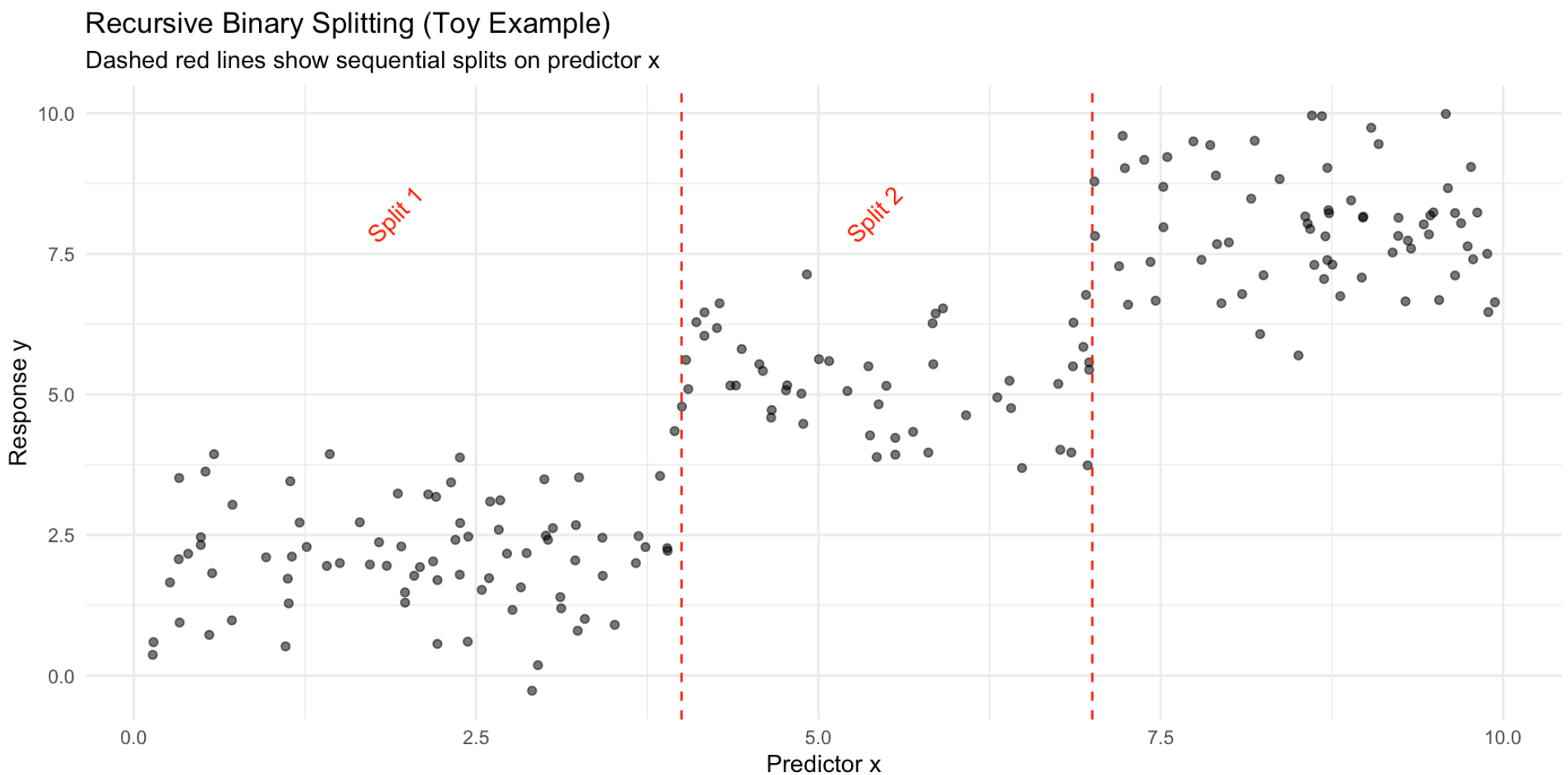
- We want to partition the feature space into $J$ distinct regions.
- **But** trying every possible partition is computationally **infeasible**.
- Instead, we use a **top-down greedy approach (recursive binary splitting)**:
  - ➟ At each step, choose the best split.
  - ➟ Recursively repeat the process on resulting subspaces.

# Recursive Binary Splitting: Key Ideas

- **Top-down**: Start at the root node (entire dataset) and split down.
- **Greedy**: At each step, choose the split that most reduces RSS or impurity *at that step*.
- **Recursive**: Once a split is made, repeat the process on each child node.

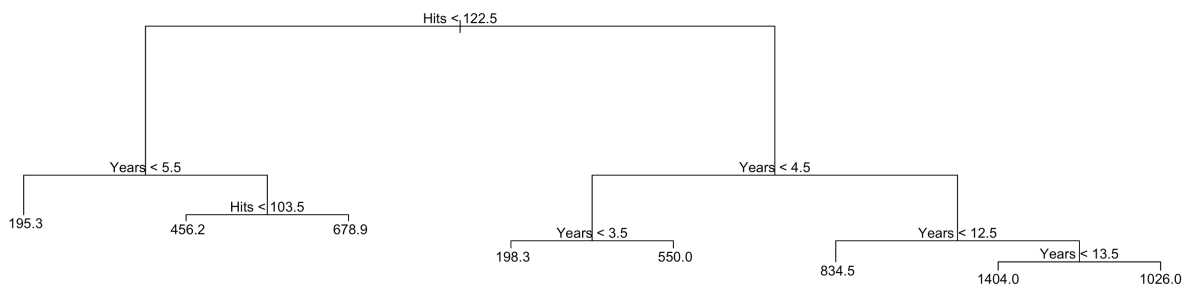
> **Note**
>
> This method does **not** look ahead — it optimises **locally**, not globally.

# Recursive Splitting Visual Example (One-Dimensional Simulated Data)



# Overfitting with Recursive Binary Splitting

- Trees are grown via **recursive binary splitting** can lead to **deep, complex trees** that fit training data perfectly.
- Such trees may **overfit** as it captures **noise** instead of signal
- This leads to **high variance** and **poor generalisation**




```r
# Step 1: Predict on the training data
predicted <- predict(tree.full, newdata = test_data)

# Step 2: Extract the actual values
actual <- test_data$Salary

# Step 3: Compute the MSE
mse <- mean((actual - predicted)^2)
mse
```

```
[1] 162111.2
```

# Tree Pruning

> **Note**
>
> Tree pruning is like regularisation — we sacrifice training accuracy to gain prediction stability (lowering the variance).

## Cost-Complexity Pruning (the weakest link pruning)

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

$T$: subtree ; $R_m$: region (leaf) $m$

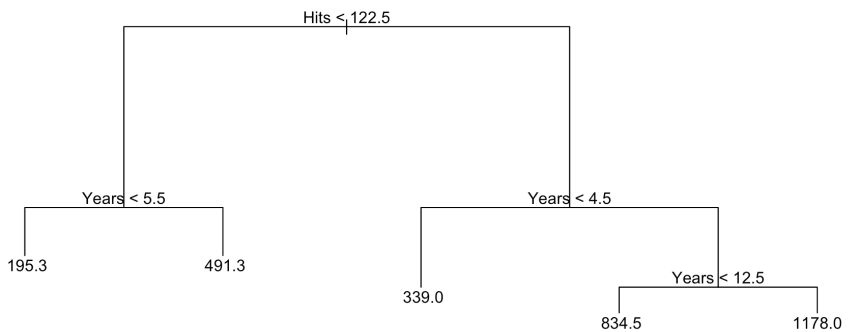
$\hat{y}_{R_m}$: mean response in region $m$

$\alpha$: tuning parameter controls a trade-off between the subtree's complexity and its fit to the training data.

# Cost-Complexity Pruning (the weakest link pruning)

▶ Code


▶ Code

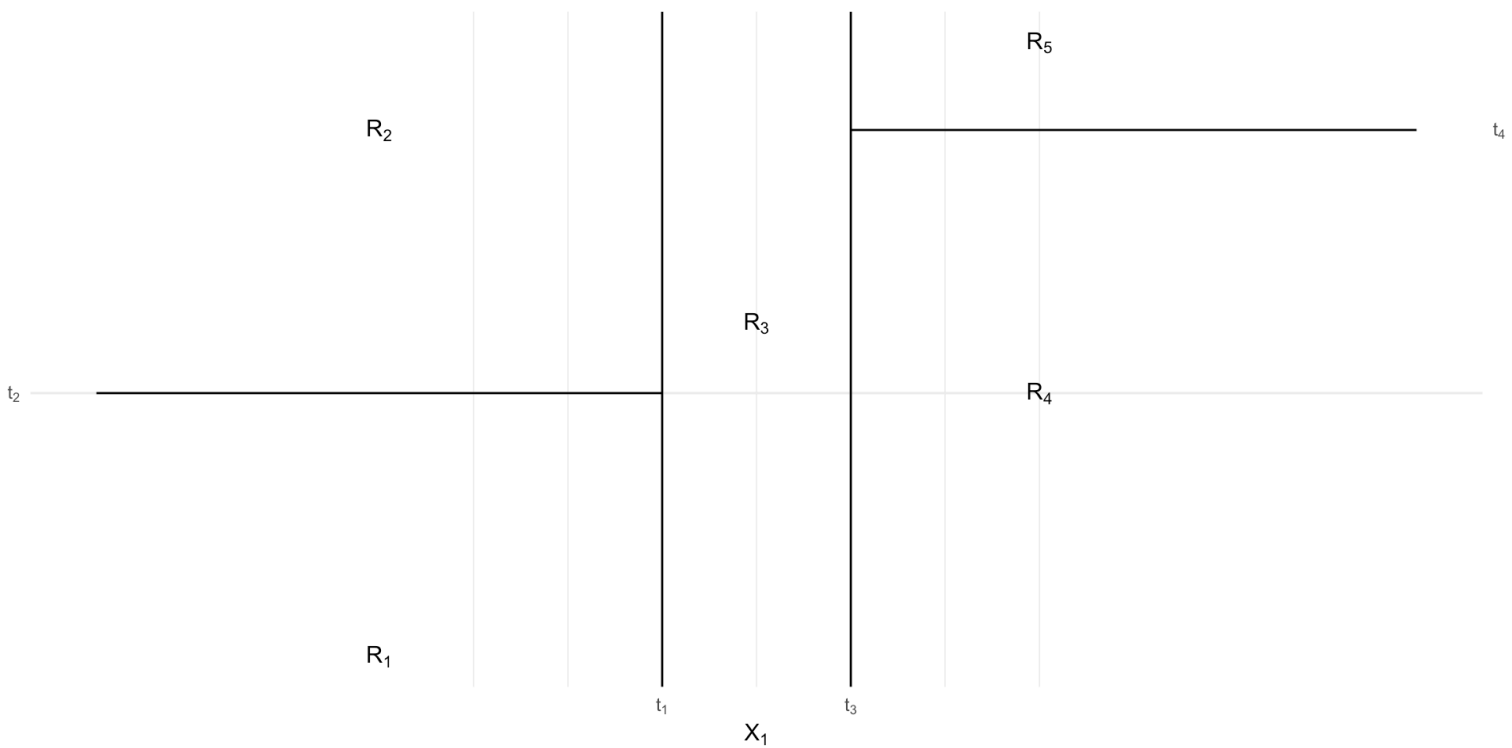



▶ Code

```
[1] 167258.6
```

# Using Decision Trees for prediction

- We divide the predictor space:
  - That is, the set of possible values for $X_1, X_2, \ldots, X_p$, into $J$ distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
- For every observation that falls into the region $R_j$, we make the same prediction, which is simply the mean of the response values for the training observations in $R_j$.



# Decision Trees for Classification

- Very similar to a regression tree
  - ➟ Exception being the prediction of a qualitative response rather than a quantitative one.
- For a classification tree
  - ➟ Inspect the region that the observation belongs and predict the most commonly occurring class in that region.

# Gini index

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits.
- Alternative measure such as Gini index is used instead.
- The Gini index is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

  - $\hat{p}_{mk}$ represents the proportion of training observations in the $m^{\text{th}}$ region that are from the $k^{\text{th}}$ class.

# Gini index

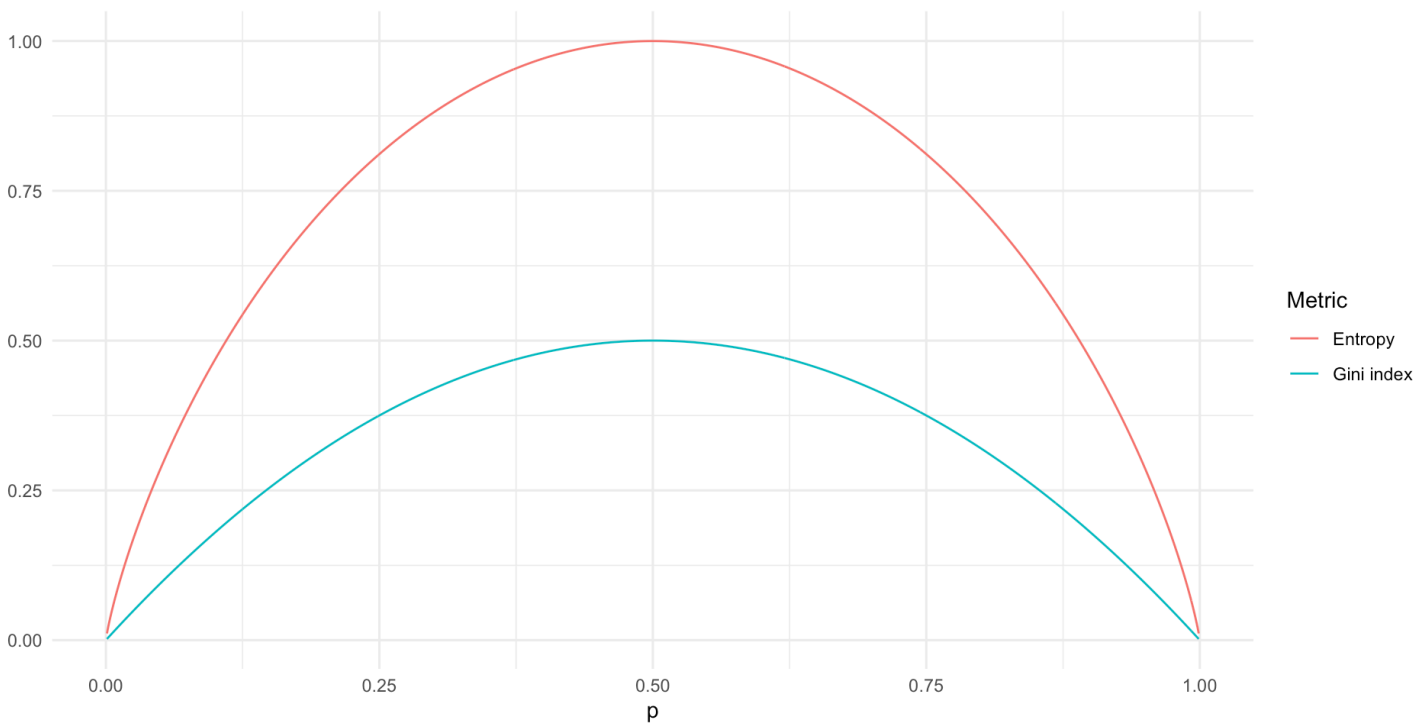
- The Gini index is a measure of total variance across the $K$ classes.
  - ➟ It takes on a small value if all of the $\hat{p}_{mk}$ values are close to zero or one.
  - ➟ This occurs when there is a clear majority class!
- For this reason the Gini index is referred to as a measure of node **purity**.
  - ➟ A small value indicates that a node contains predominantly observations from a single class.

# Entropy

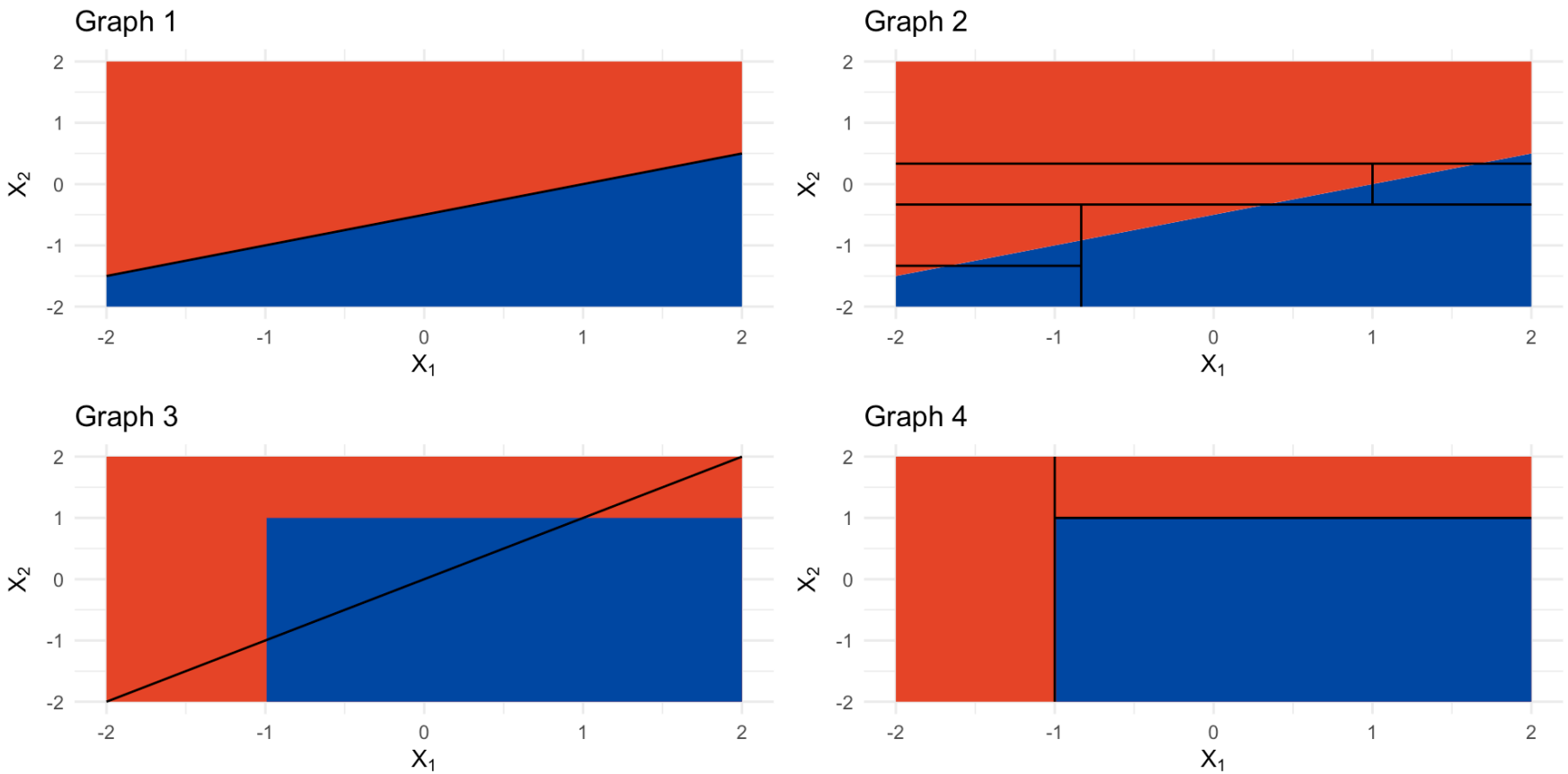
- An alternative to the Gini index is entropy, given by

$$D = -\sum_{j=1}^{J}\sum_{k=1}^{K}\hat{p}_{jk}\log\hat{p}_{jk}$$

- It turns out that the Gini index and the Entropy are very similar numerically



# Linear Model versus Tree




- If the relationship between the features and the response is well approximated by a linear model as shown in the top panel, then fitting a linear model (Graph 1) outperforms the tree-based model (Graph 2).
- If the relationship is complex and non-linear as shown on the bottom panel, then the tree-based model (Graph 4) is more appropriate.

# Advantages and disadvantages of trees

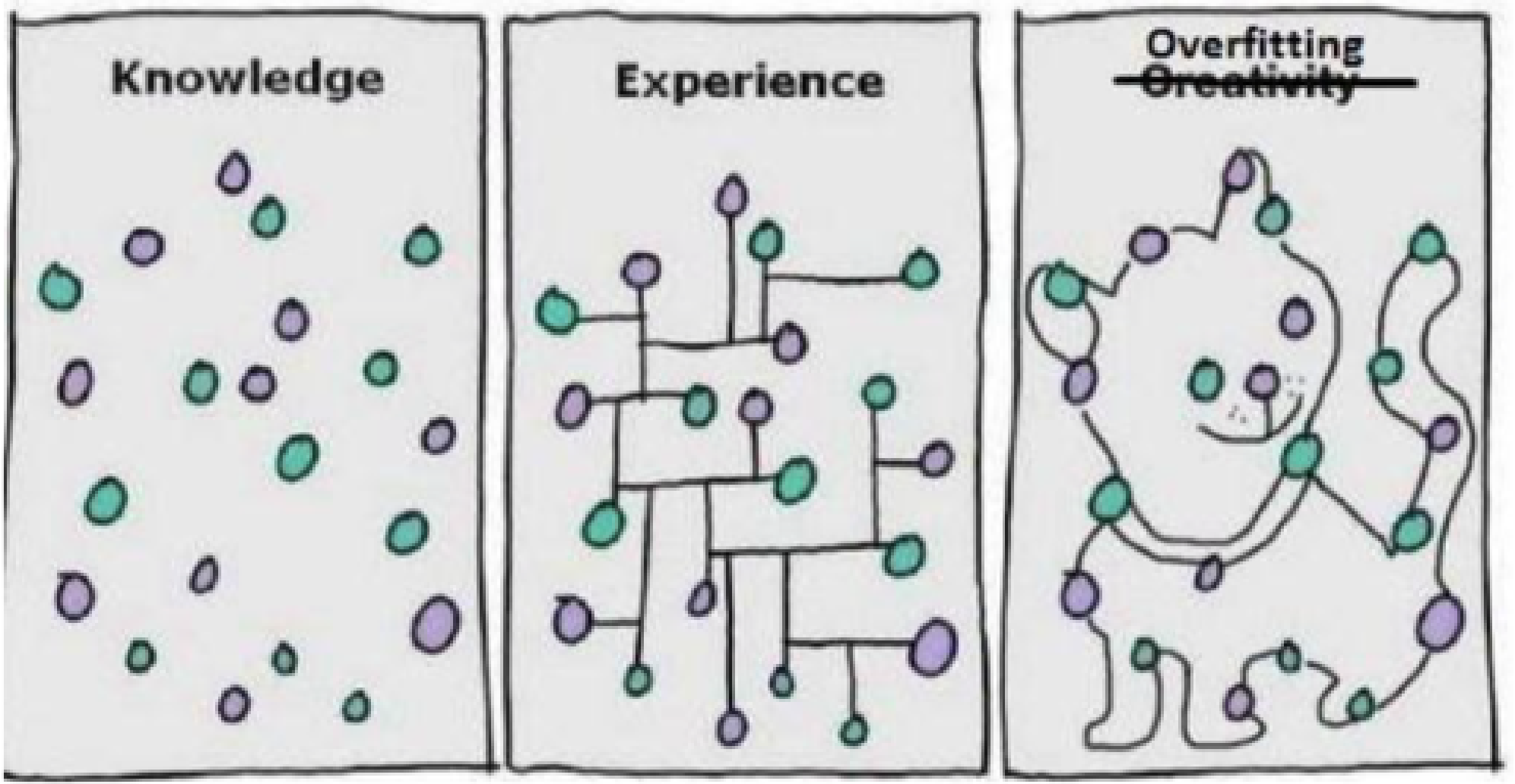
**Advantages:**

- Trees are very easy to explain to people
- Some people believe that decision trees closely relate to human decision-making
- Trees can be displayed graphically
- Can handle different data types and doesn't require scaling

**Disadvantages:**

- Trees do not have the same level of predictive accuracy as some of the other regression and classification approaches we have discussed

# A single decision tree is prone to over-fitting



# Ensemble of trees

- Build **multiple** decision trees
- Combine their results to form a more **stable and accurate** predictor

Key Ideas

- Take **samples** of trees (e.g. using bootstrapping)
- Use **random subsets of features** at each split
- **Aggregate** predictions:
  - **Majority vote** (classification)
  - **Average** (regression)

Popular Methods

- **Bagging**: Build trees in parallel and average results
- **Random Forest**: Bagging + random feature selection
- **Boosting**: Build trees sequentially to improve errors

The strength of ensemble methods lies in **reducing variance** and **improving robustness** through aggregation.

**“Many weak learners can form a strong learner.”**

# Bagging (Bootstrap Aggregating)

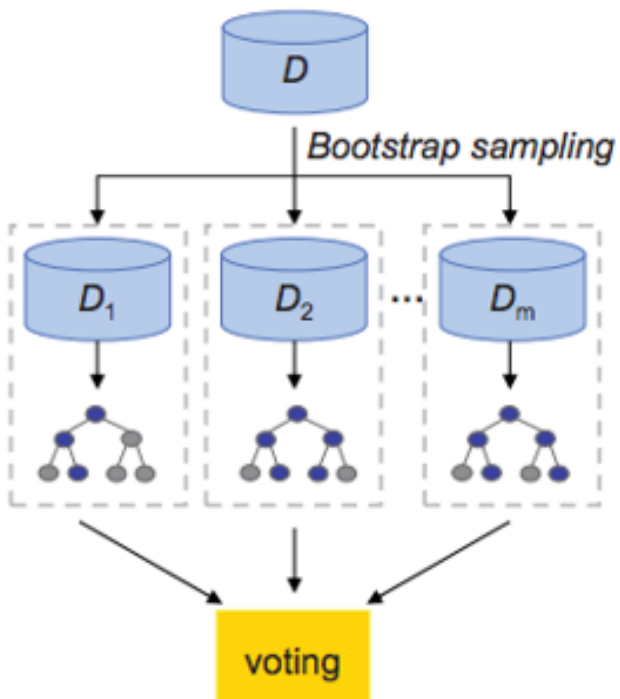
- Bootstrap aggregating, or bagging, is a general purpose procedure for reducing the variance of a statistical learning method.
  - ➟ It is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of $n$ independent observations $Z_1, \ldots, Z_n$ each with a variance of $\sigma^2$:
  - ➟ The variance of the mean $\overline{Z}$ is given by $\sigma^2/n$.
- In other words, averaging a set of observations reduces variance.
- Since it is typically not possible to have access to multiple training sets:
  - ➟ We can use **bootstrapping** to create multiple training sets.

# Bagging continued

- Use bootstrapping to take repeated samples from a (single) training data set.
- In this approach, we generate $B$ different bootstrapped training data sets.
  - ➠ Train our method of the $b^{\text{th}}$ bootstrapping set in order to obtain $\hat{f}_b^*(x)$, the prediction at a point $x$.
- Average all the observations to obtain:

$$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b^*(x).$$

- This is called bagging.



# Out of bag error estimation

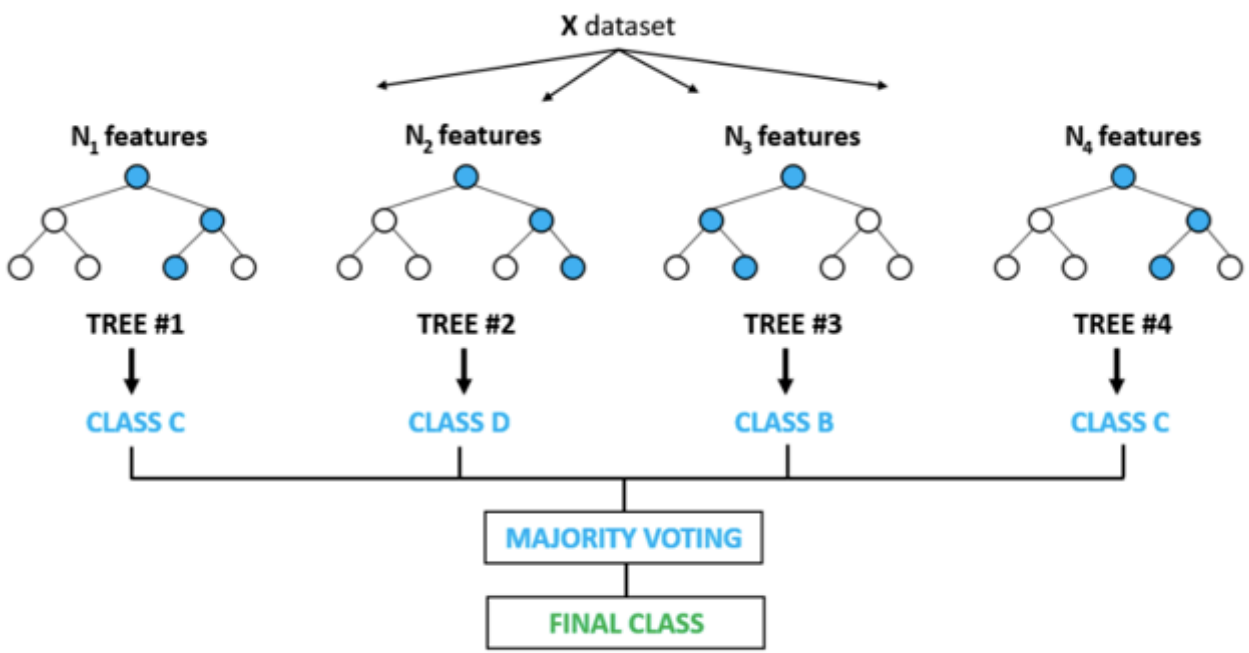
- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations.
    - One can show that on average, each bagged tree makes use of around $2/3$ of the observations when the sample size $n$ is large.
- The remaining $1/3$ of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the $i^{\text{th}}$ observation using each of the trees in which that observation was OOB.
    - This will yield around $B/3$ predictions for the $i^{\text{th}}$ observation, which we average.
    - This estimate is essentially the leave-one-out (LOO) cross-validation error for bagging, if $B$ is large.

# From bagging to Random Forest

- Random forests (sometimes) provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random selection of $m$ predictors is chosen as split candidates from the full set of $p$ predictors.
  - ➠ The split is allowed to use only one of those $m$ predictors.
- A fresh selection of $m$ predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$.
  - ➠ That is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

# Random Forest algorithm pseudocode

```
Set number of models to build, B
for i = 1 to B
    Generate a boostrap sample of the original data
    Train a tree model on this sample where
    for each split
        Randomly select m (< p) of the original predictors
        select the best predictor among the m predictors and parition the data
    endfor
endfor
```



# Ensemble methods via boosting

# Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, **independent of other trees**.
- Boosting works in a similar way, except that the trees are grown *sequentially*.
    - ➠ Each tree is grown using information from previously grown trees.

# Idea behind boosting

- Unlike fitting a single large decision tree to the data, which amounts to *fitting the data hard* and potentially overfitting, the boosting approach instead *learns slowly*.
- Each new model attempts to **correct the errors** made by the previous ones.
- The final model is a **weighted sum** of weak learners (usually decision trees).
- By fitting small trees to the residuals, we slowly improve $\hat{f}(x)$ in areas where it does not perform well.
- There is a shrinkage parameter $\lambda$ that slows the process down even further, allowing more and different shaped trees to attack the residuals.

# Boosting for regression trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set
    - $r_i$ denotes the $i^{\text{th}}$ residual, and $y_i$ is the outcome
2. For $b = 1, 2, \ldots, B$:
    - Fit a tree $\hat{f}_b$ with $d$ splits ($d + 1$ terminal nodes) to the new training data $(X, r)$
        - That is, $r$ is the new response value
    - Update $\hat{f}$ by adding in a shrunken version of the new tree
        - $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_b(x)$
    - Update the residuals
        - $r_i \leftarrow r_i - \lambda \hat{f}_b(x)$
3. Compute the final model

$$\hat{f}(x) = \sum_{i=1}^{B} \lambda \hat{f}_b(x)$$

# Parameters to tune in boosting

- Number of trees $B$
- The shrinkage parameter $\lambda$: a small positive number
  - ➟ Typical values are between 0.01 and 0.001
- Number of split $d$ in each tree
  - ➟ Controls the complexity of the boosted ensemble
  - ➟ If $d = 1$, then the tree is just a stump. This actually usually works quite well

# Gradient Boosting in Action (R Demo)

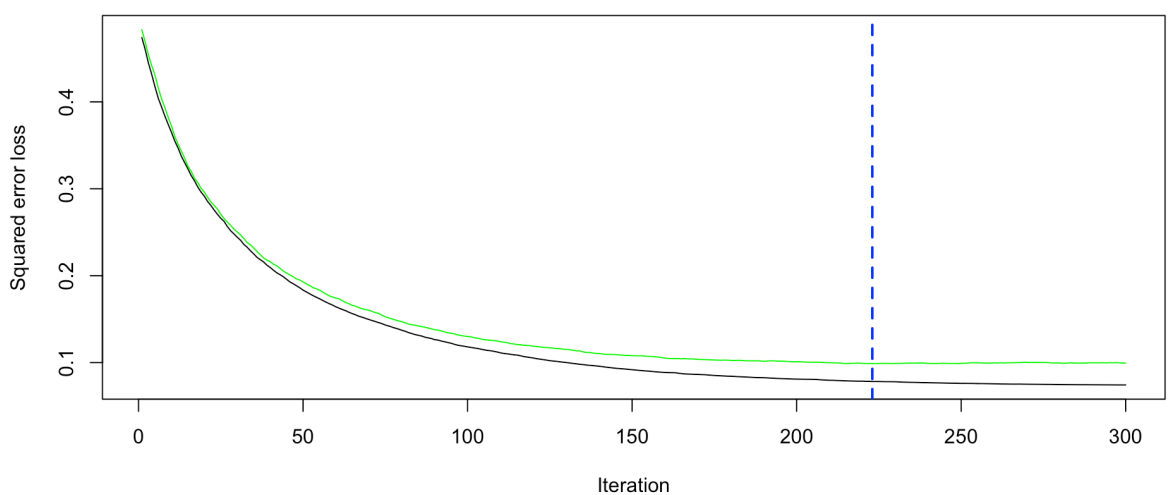
▶ Code

```
CV: 1
CV: 2
CV: 3
CV: 4
CV: 5
```

▶ Code

▶ Code

```
[1] 223
```

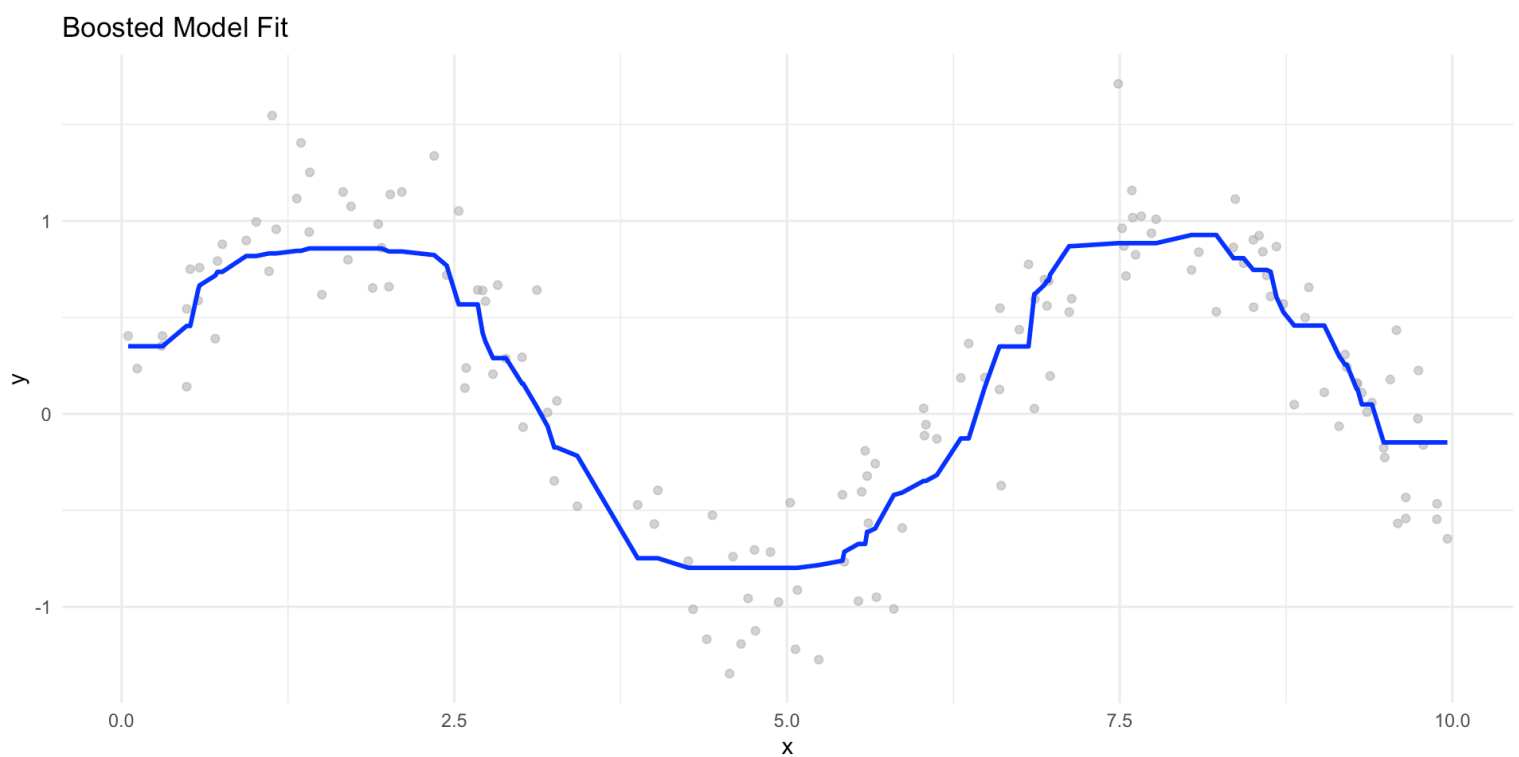


# Gradient Boosting in Action (R Demo)

▶ Code

```
CV: 1
CV: 2
CV: 3
CV: 4
CV: 5
```

▶ Code



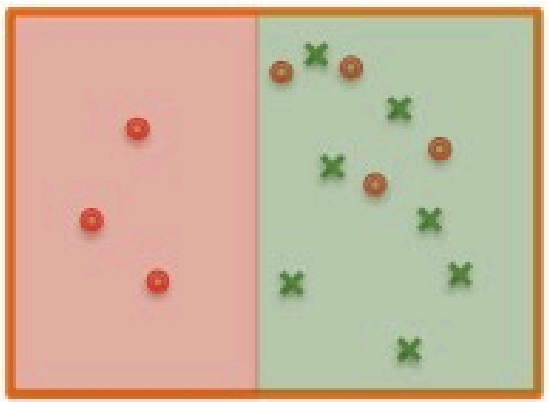
# Other boosting algorithms

- AdaBoost
- Stochastic gradient boosting
- XGBoost

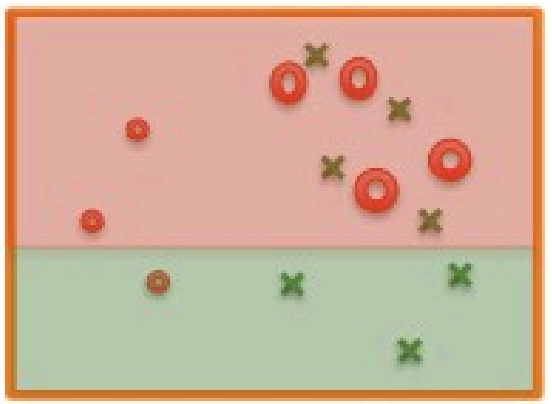
# AdaBoost

- Short for Adaptive Boosting, one of the first boosting algorithms for classification
- Convert a set of weak classifiers (*decision stumps*) into a strong one
- Basic idea
  - ➟ At each iteration, reweight the data to place more weight on data points that the classifier got wrong
- At the end, combine all the weak classifiers by taking a weighted combination
  - ➟ Put more weight on the weak classifiers with the higher accuracies

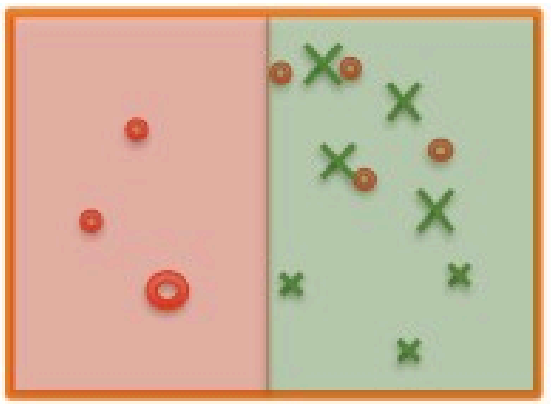
# AdaBoost plot



Iteration 1



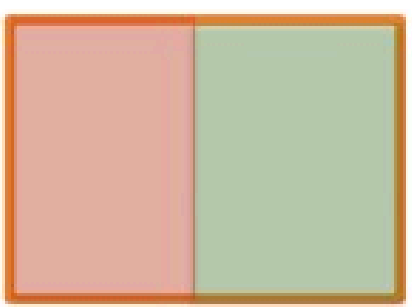
Iteration 2



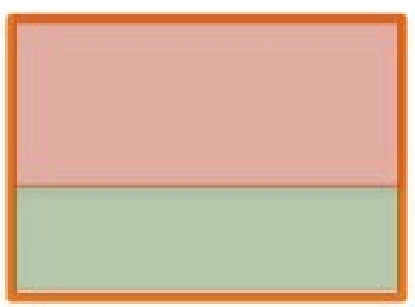
Iteration 3

Final Model

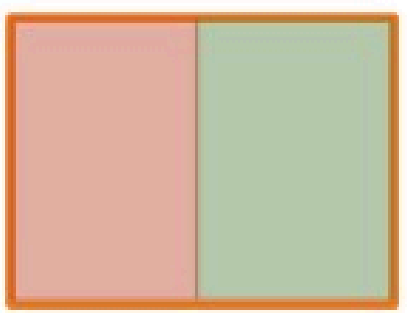
f(x) = 0.46



+ 0.29



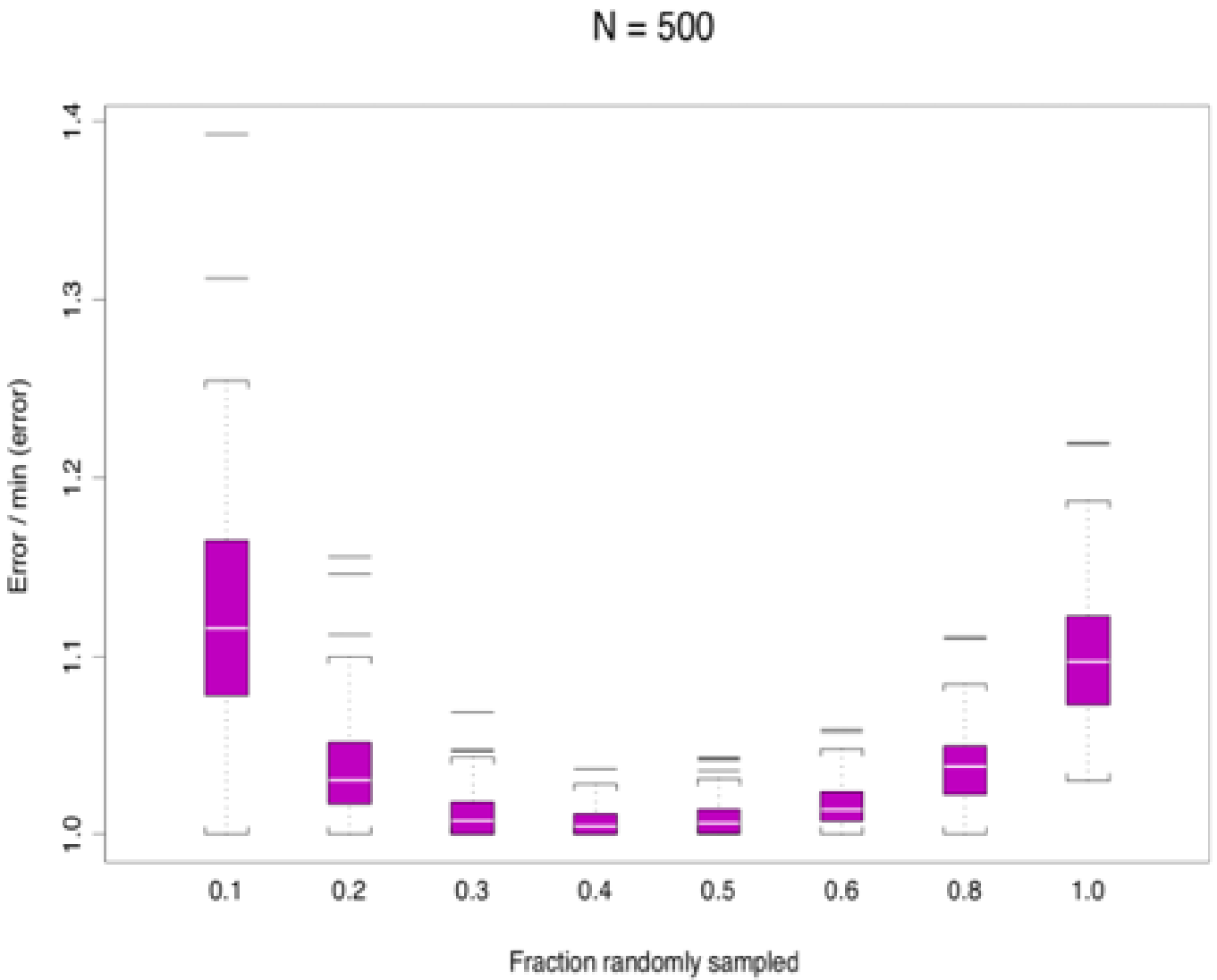
+ 0.46



# Stochastic gradient boosting

- Use the idea behind bagging
- In each iteration of stochastic boosting, a sample of the training set instead of the full training set is used
- Instead of a bootstrap sample (with replacement), the algorithms samples a fraction of the training set
- This introduced randomness can improve performance
- Paper by Stochastic gradient boosting covers this technique

# Stochastic gradient boosting plot




- Taken from Stochastic gradient boosting

# XGBoost

- XGBoost is short for eXtreme Gradient Boosting
- It is a library for high performance gradient boosting models written in C++ with implementations in Python, R, and Julia
- Designed for speed: up to 10 times faster than the `gbm` package
- Has been very popular in recent years and has won a number of machine learning competitions (especially on tabular data)
- Supports regularization
- Can handle missing data

# Bagging vs Boosting

- Both ensemble methods get $N$ learners from 1 learner
  - Built independently for bagging
  - Built sequentially for boosting
- Trees built in boosting are weak learners (sometimes just a stump) while trees in Random Forest have higher complexity
- Fewer parameters to tune in Random Forest, many more in Boosting (depending on which variation you are using)
- Both combine outputs from $N$ trees

# Summary

- Decision trees are simple and interpretable models for regression and classification.
- However, they are often not competitive with other methods in terms of prediction accuracy.
- Bagging, random forests, and boosting are good methods for improving the prediction accuracy of trees.
  - ➟ They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods, random forests and boosting, are among the state-of-the-art methods for supervised learning.
  - ➟ Their results can be difficult to interpret.
  - ➟ But we can still get **relative importance** of each predictor.
  - ➟ For classification, the relative importance is computed using the mean decrease in Gini index and expressed relative to the maximum.