CAP 5625: Computational Foundations for Artificial Intelligence

Random forests and boosting

Methods considered so far

Throughout this class we have considered a number of linear and non-linear supervised learning approaches.

These methods all have their relative merits.

For example, neural networks and kNN regression (or classification) can perform non-linear modeling, yet the kNN approach suffers from the curse of dimensionality (adding irrelevant features degrades performance).

Neural networks on the other hand are transformations of linear models, and require adjustments of inputs to be made to handle common empirical hurdles such as mixed data types, missing values, and transformations of input measurements.

Interpretability

Moreover, a more difficult task for many of the approaches we have considered is interpretation of the fitted model.

All but the simplest models can be difficult to visualize, and with so may parameters (*e.g.*, neural networks), understanding why any one parameter is important in the model can be difficult.

Here we introduce decision trees, which overcome some of the hurdles that we have discussed.

In particular, these tree approaches can naturally handle mixtures of quantitative and qualitative features without making dummy variables, can readily handle missing data, do not require input feature measurements to be scaled or transformed, and are easy to interpret.

Decision trees

Decision trees segment the feature space into a set of simple regions.

To make predictions for a given observation $X \in \mathbb{R}^p$, we will typically use the mean (regression) or the mode (classification) of the observations in that region.

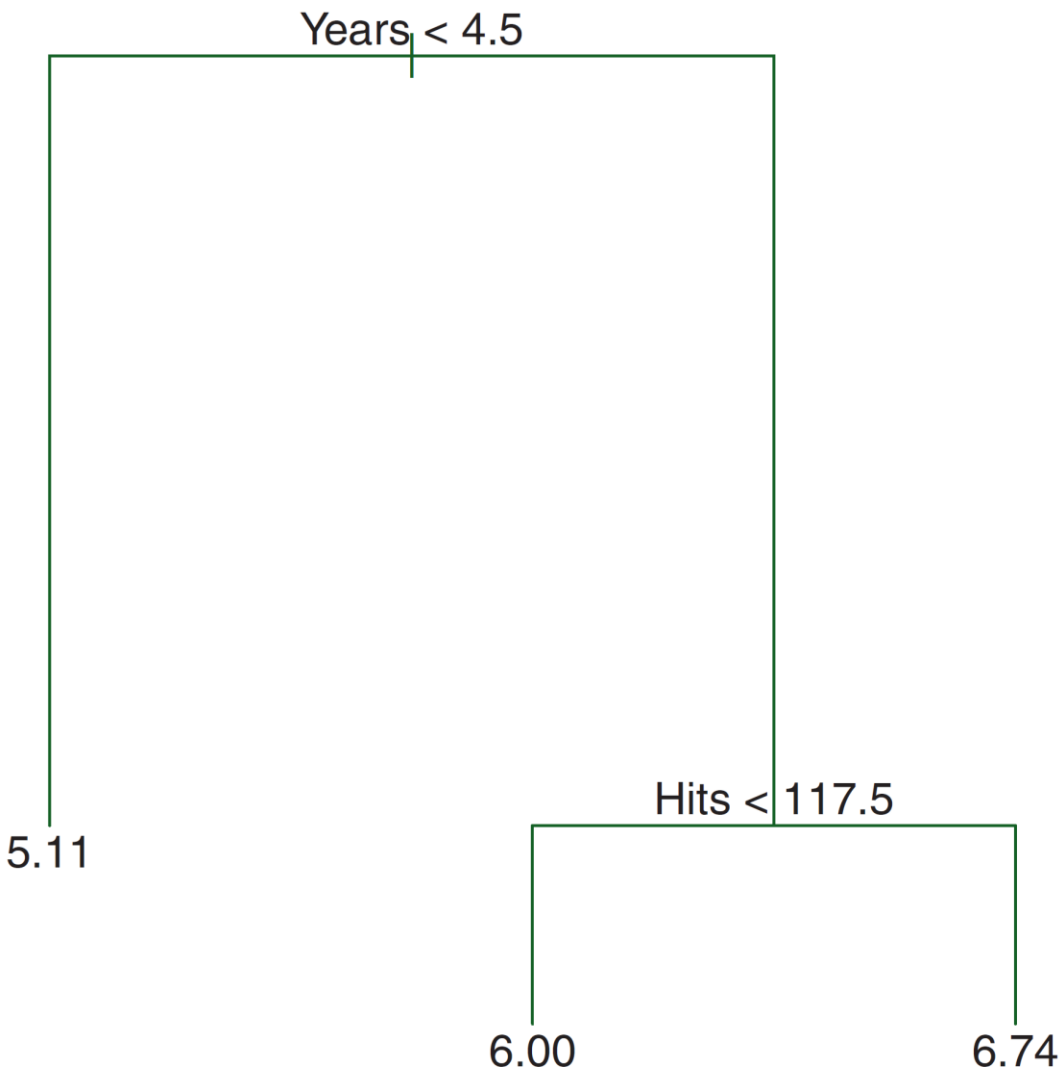
Decision trees have the advantage over many other (non-linear) methods in that they are easy to interpret and visualize.

We will begin with a description of decision trees for regression, and then move to their application in classification.

Example regression tree

Consider the problem of predicting the logarithm of a baseball player's salary based on the number of years that he has played in the major leagues and his number of hits made in the previous year.

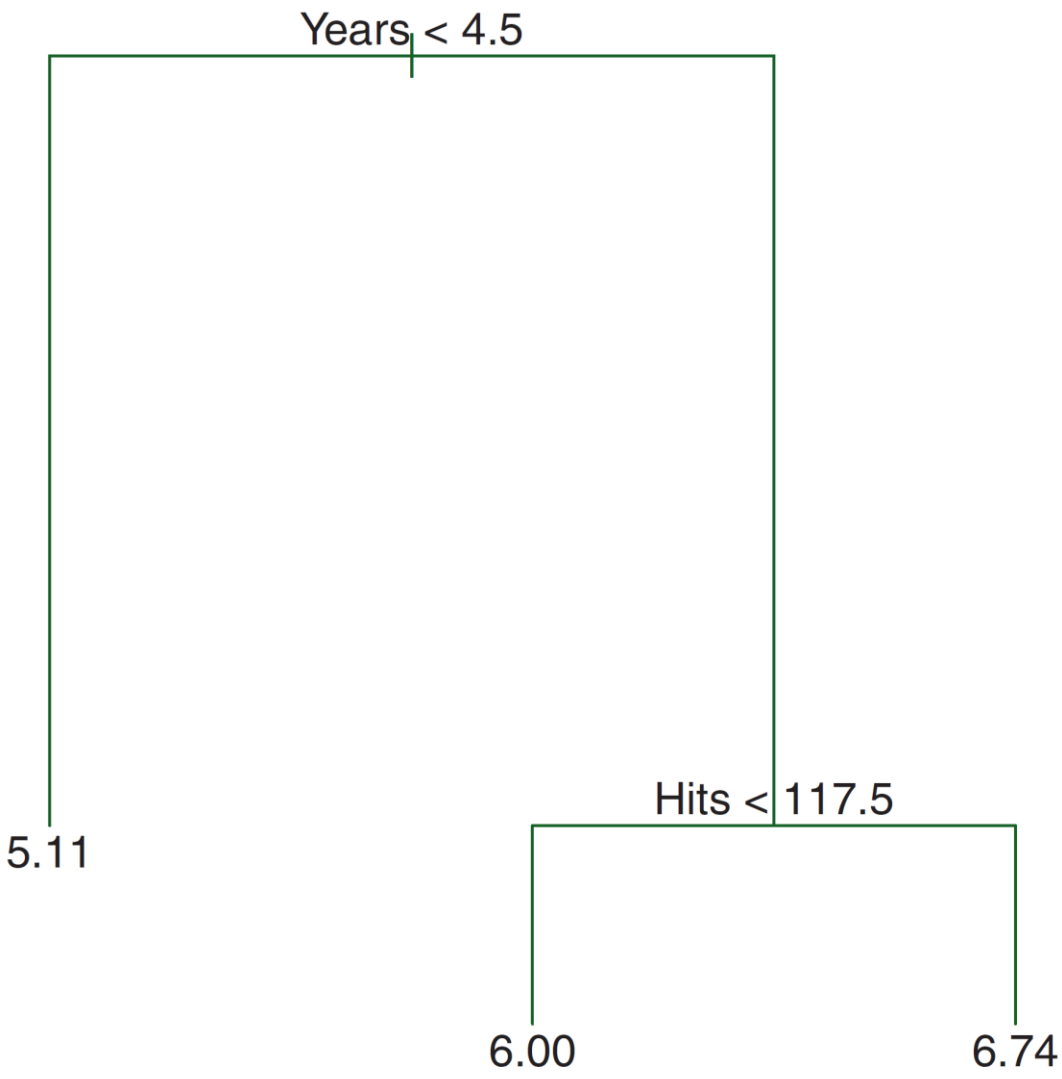
At a given internal node, the label $X_j < t$ indicates the left branch subtending that split, and the right branch corresponds to $X_j \geq t$.



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example regression tree

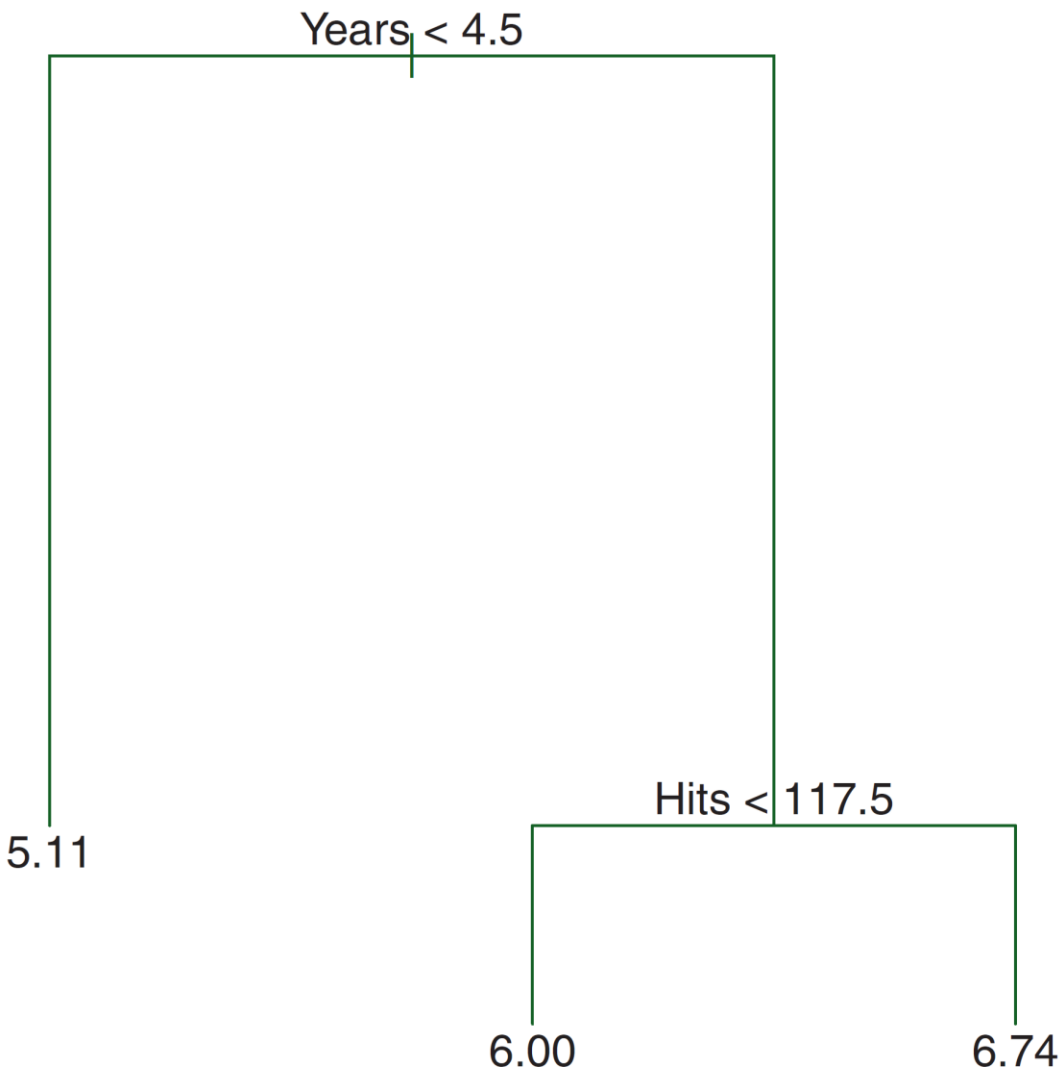
Here, the root (top split) of the tree results in two branches, where the left branch corresponds to a player with fewer than 4.5 years experience, and the right branch corresponds to one with at least 4.5 years experience.



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example regression tree

Similarly, the second split (along the right branch subtending the root) of the tree results in two branches, where the left branch corresponds to a player with fewer than 117.5 hits in the previous year, and the right branch corresponds to one with at least 117.5 hits.



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example regression tree

The tree has two internal nodes (one being the root) and three terminal or external nodes, also known as leaves or tips.

Each leaf corresponds to a particular region of input space.

The number under each leaf is the mean of the response (log salary) for training observations falling in that region.

James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

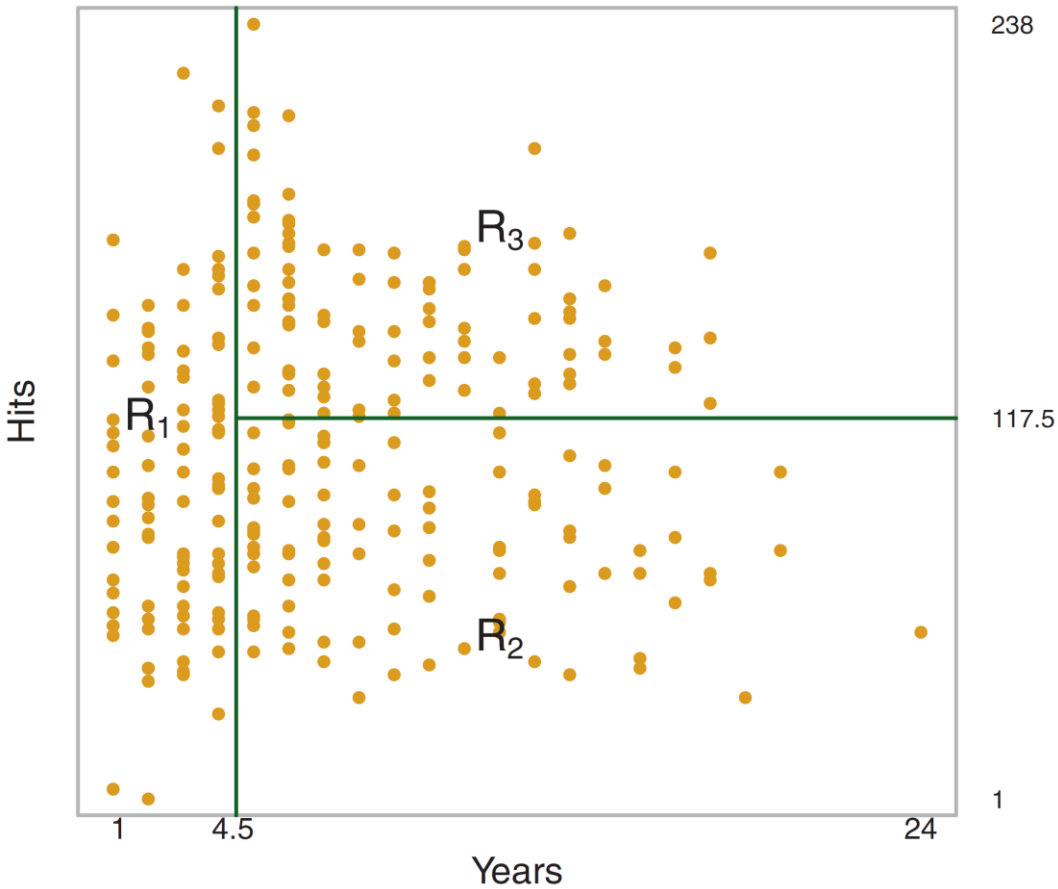
Regions defined by the example regression tree

Overall, the tree segments players in three regions of feature space:

R_1 Players with no more than 4 years experience

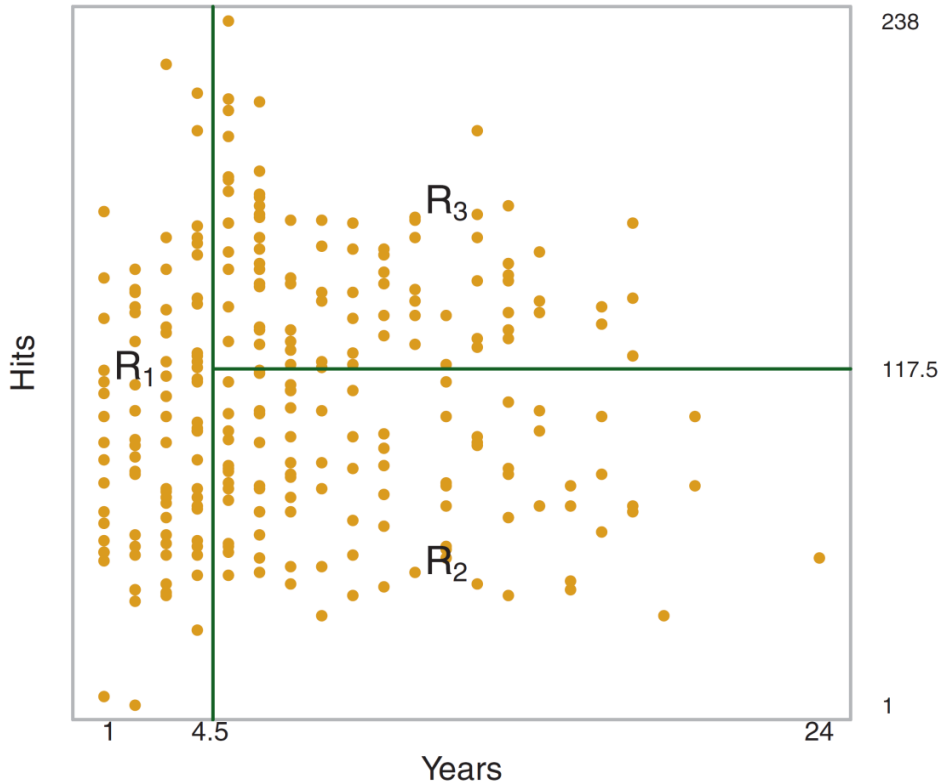
R_2 Players with at least 5 years experience with at maximum 117 hits in the previous year

R_3 Players with at least 5 years experience with at least 118 hits in the previous year



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Regions defined by the example regression tree



We write these regions as:

$$R_1 = \{X \mid \text{Years} < 4.5\}$$

$$R_2 = \{X \mid \text{Years} \geq 4.5, \text{Hits} < 117.5\}$$

$$R_3 = \{X \mid \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$$

James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Building a regression tree

To build a regression tree, we take two steps.

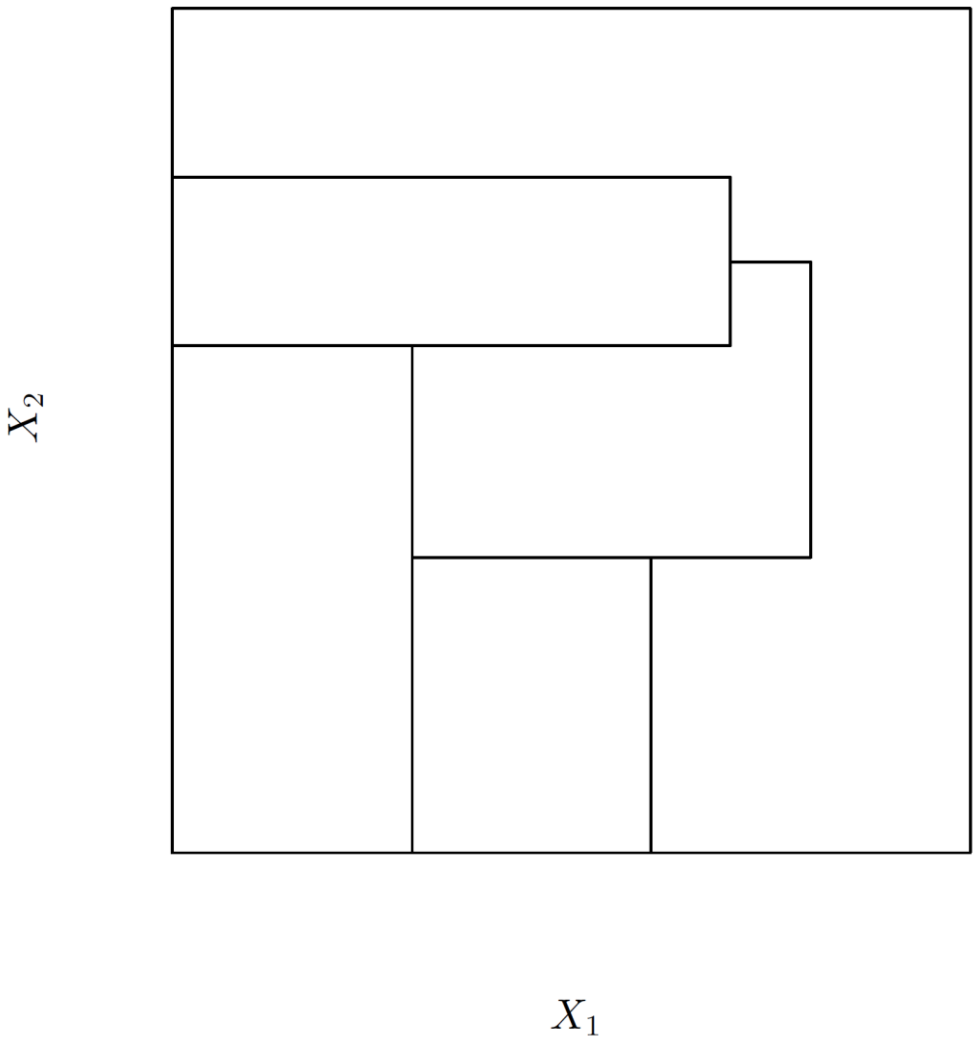
The first step is to divide the feature space $X \in \mathbb{R}^p$ into M distinct and non-overlapping regions $R_1, R_2, \ldots, R_M$.

Next, for each observation that falls into region R_m, $m = 1,2, \ldots, M$, we make the same prediction, which is the mean response Y for training observations falling in R_m.

We therefore first need to figure out how to construct the M regions that segment the input space.

Finding the M regions

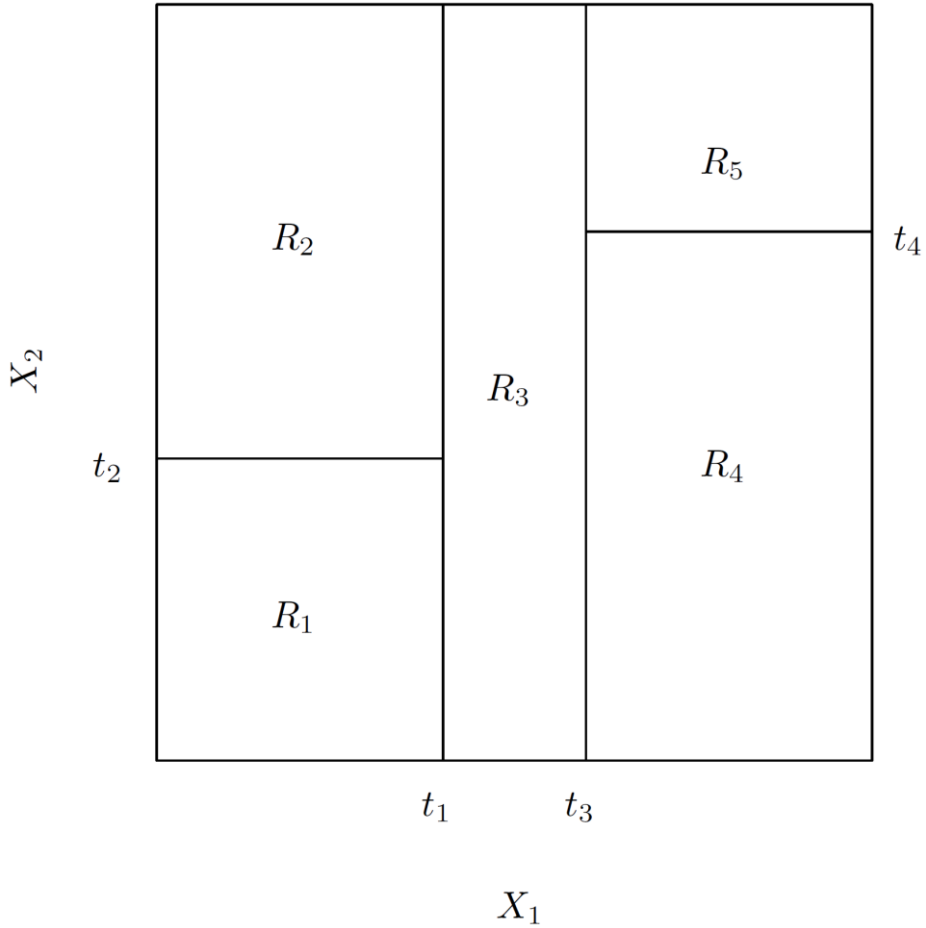
We could in theory permit the regions to have any shape.



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Finding the M regions

However, for regression trees, we divide the input (feature) space into high-dimensional rectangles, or boxes, to make interpretation of the prediction model easier.



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Finding the M regions

This represents an optimization problem in which we want to identify the M boxes $R_1, R_2, \ldots, R_M$ that minimize the RSS given by

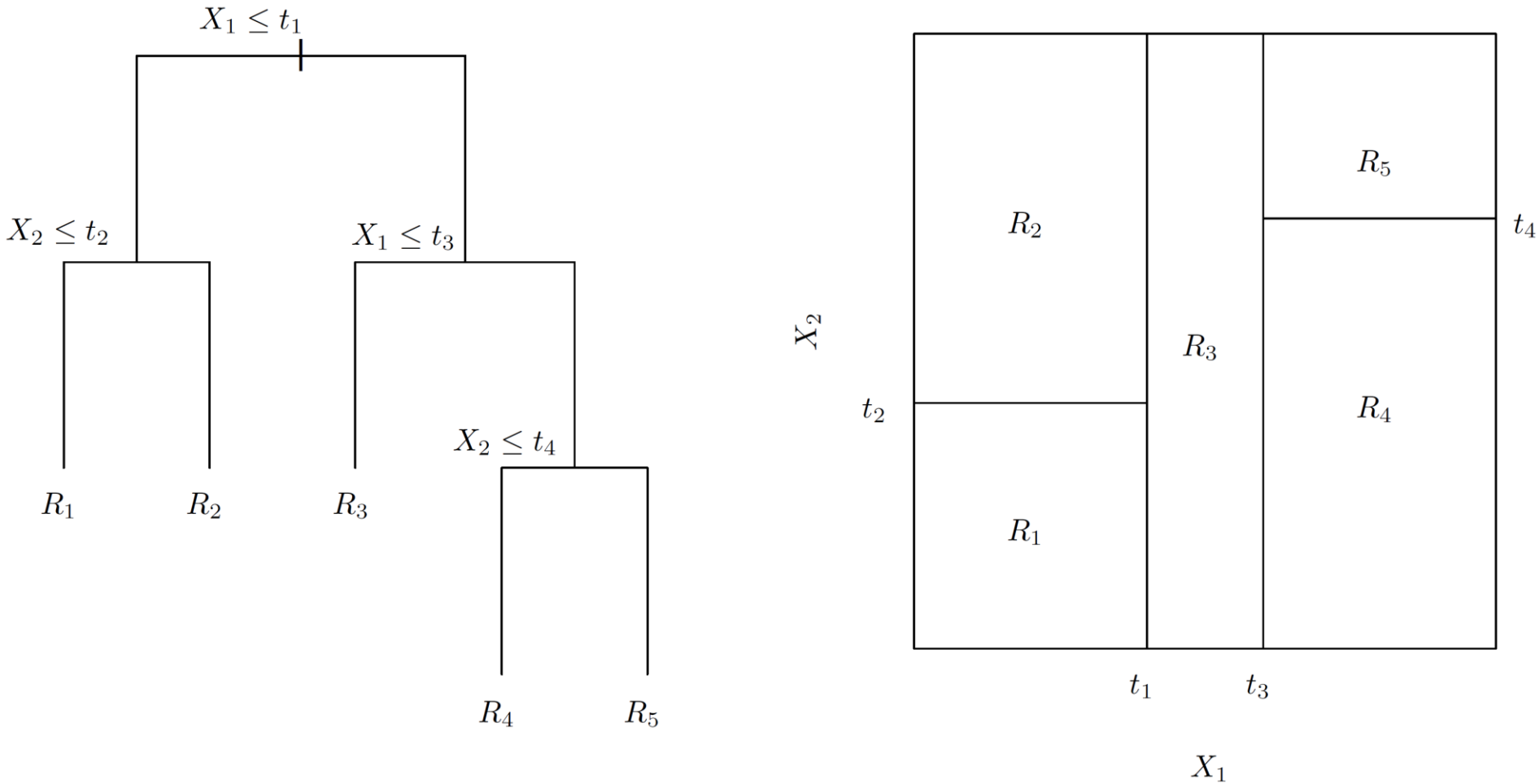
$$J = \sum_{m=1}^{M} \sum_{i:\, x_i \in R_m} \left(y_i - \hat{y}_{R_m}\right)^2$$

where $\hat{y}_{R_m}$ is the mean response for training observations falling within region R_m, $m = 1,2, \ldots, M$.

Still, the number of potential placement of boxes (regions) is large.

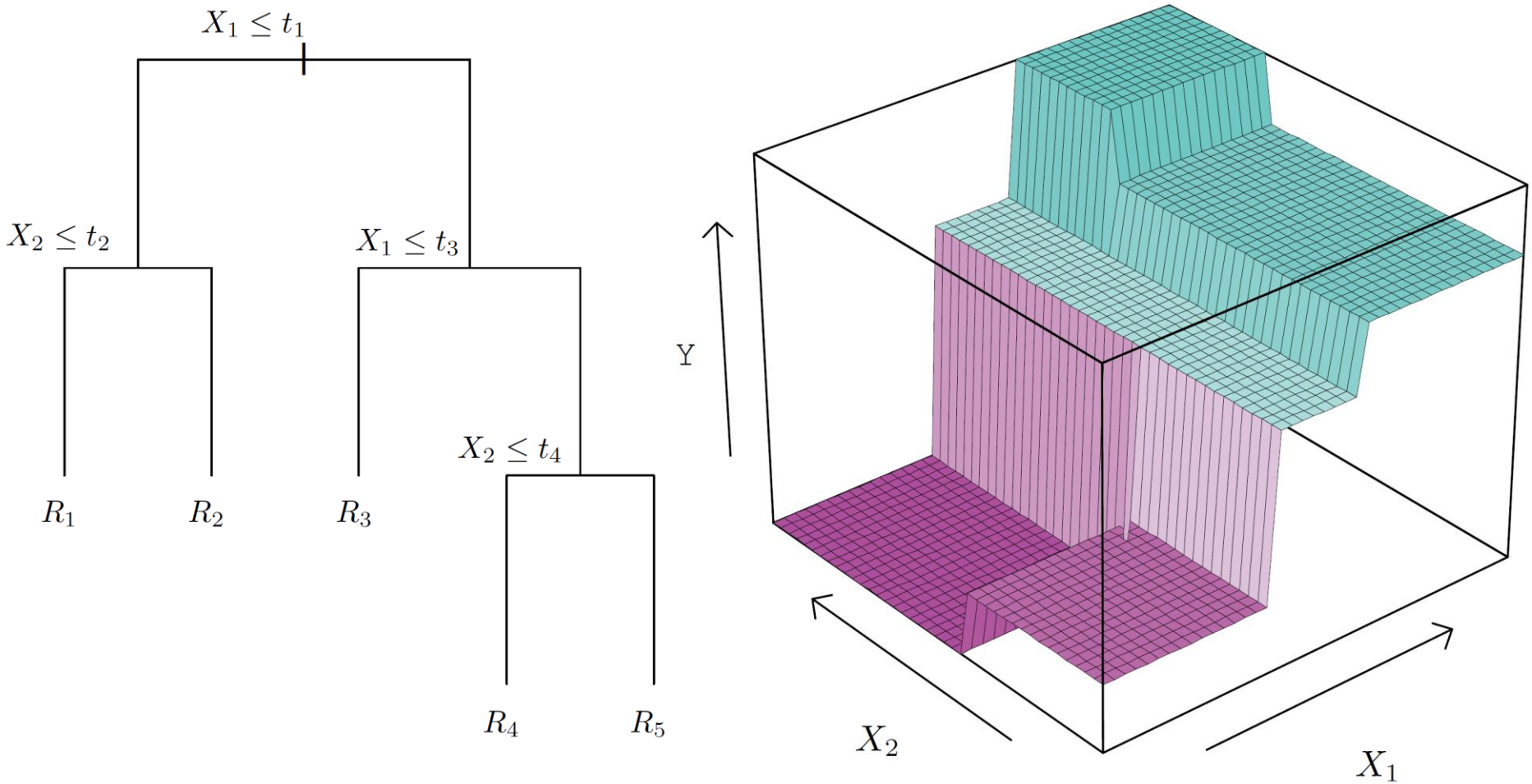
We therefore take a **greedy** (**top-down**) approach to segmenting the feature space into M regions by performing **recursive binary splitting**, in which the best split (greedy) is made at each particular step.

Top-down (greedy) construction of regions



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Prediction surface of the regression tree



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Performing recursive binary splitting

We first select feature X_j and the cutpoint t, such that splitting the feature space into regions $\{X \mid X_j < t\}$ and $\{X \mid X_j \geq t\}$ leads to the greatest possible reduction in the RSS.

That is, consider all p features $X_1, X_2, \ldots, X_p$, and all possible values of the cutpoint t for each feature.

Then choose the feature X_j and the cutpoint t such that the resulting tree has the lowest RSS.

Performing recursive binary splitting

Recursively on a particular node, for any feature j and cutpoint t, define the pair of half-planes

$$R_1(j,t) = \{X \mid X_j < t\}$$

$$R_2(j,t) = \{X \mid X_j \geq t\}$$

We seek the values of j and t that minimize the expression

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

where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean responses for training observations falling within regions $R_1(j,t)$ and $R_2(j,t)$, respectively.

There exist fast ways of minimizing the above expression to find the optimal values of $j \in \{1,2,\ldots,p\}$ and $t \in \mathbb{R}$.

Large M can overfit the training data

This process of performing recursive binary splitting can make excellent predictions on training data, but it will likely make poor predictions on test data due to overfitting.

The reason is that if M is large, then the tree will be too complex, and hence the prediction model will be too flexible, thereby having a large variance.

That is, a regression tree with M regions has $M - 1$ splits, and so one way to control the flexibility of the fit model is to control the number of splits on (or size of) the tree.

Therefore, as a strategy to prevent overfitting, we will grow a large tree T_0, and then **prune** it to obtain a **subtree**.

Cost complexity pruning

Cost complexity pruning provides a mechanism for identifying a subtree once a large complex tree T_0 is initially built, by setting up a penalized optimization function with tuning parameter $\alpha \geq 0$.

For each value of α, there exists a subtree $T \subset T_0$ that minimizes

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

where $|T| = M$ is the number of terminal nodes (regions), and has a similar effect as lasso by "selecting out" regions.

When $\alpha = 0$, we obtain $T = T_0$, and as α gets large, the subtree T gets increasingly smaller until it is a **stump** (has 1 split).

Cost complexity pruning

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

As α increases, branches get pruned from the tree in a nested and predictable manner (Breiman *et al*. 1984, Ripley 1996).

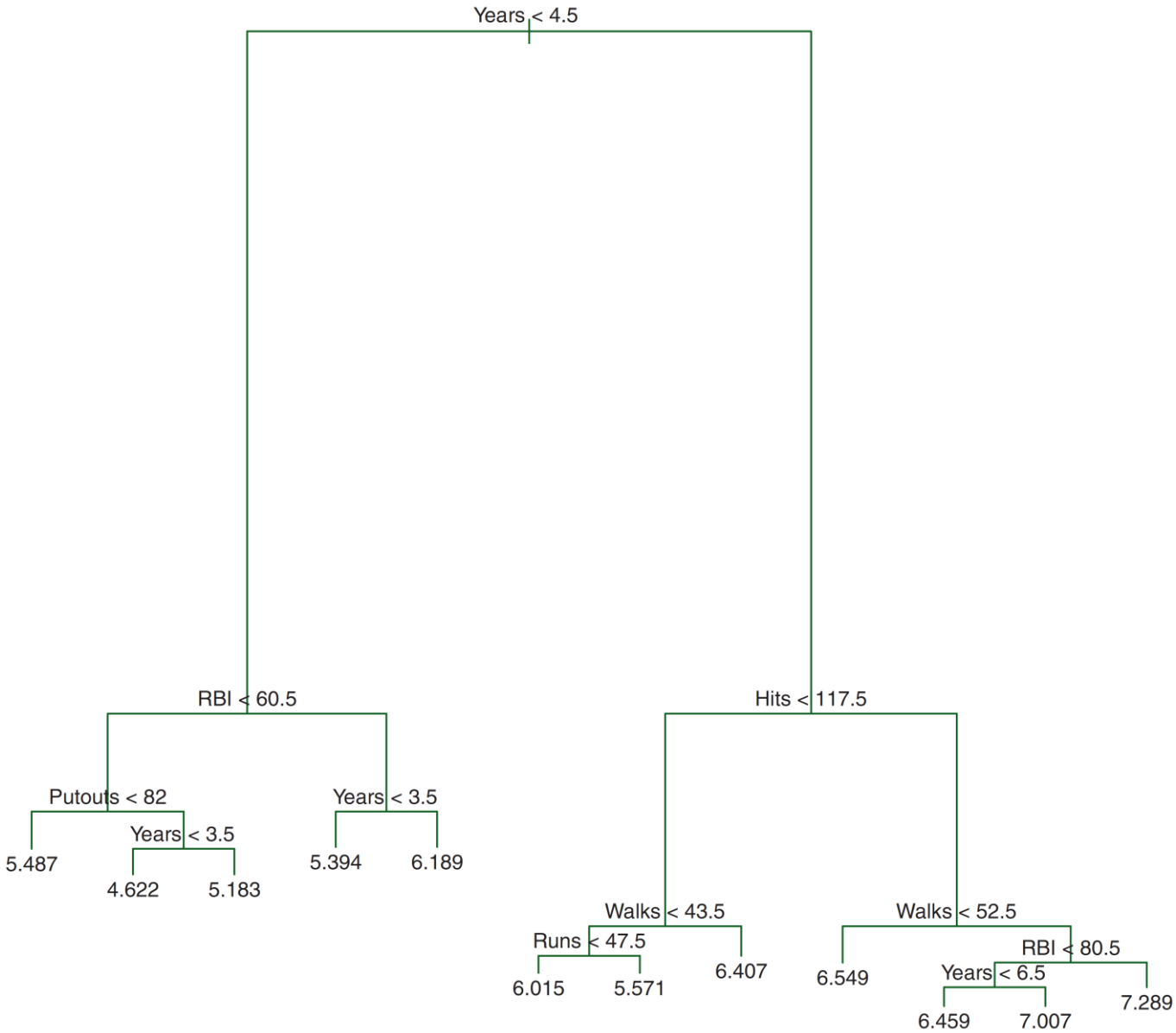
Therefore, it is easy to obtain the whole sequence of subtrees as a function of α.

The optimal α can then be chosen through cross validation.

Breiman *et al* (1984) *Classification and Regression Trees*. Wadsworth.

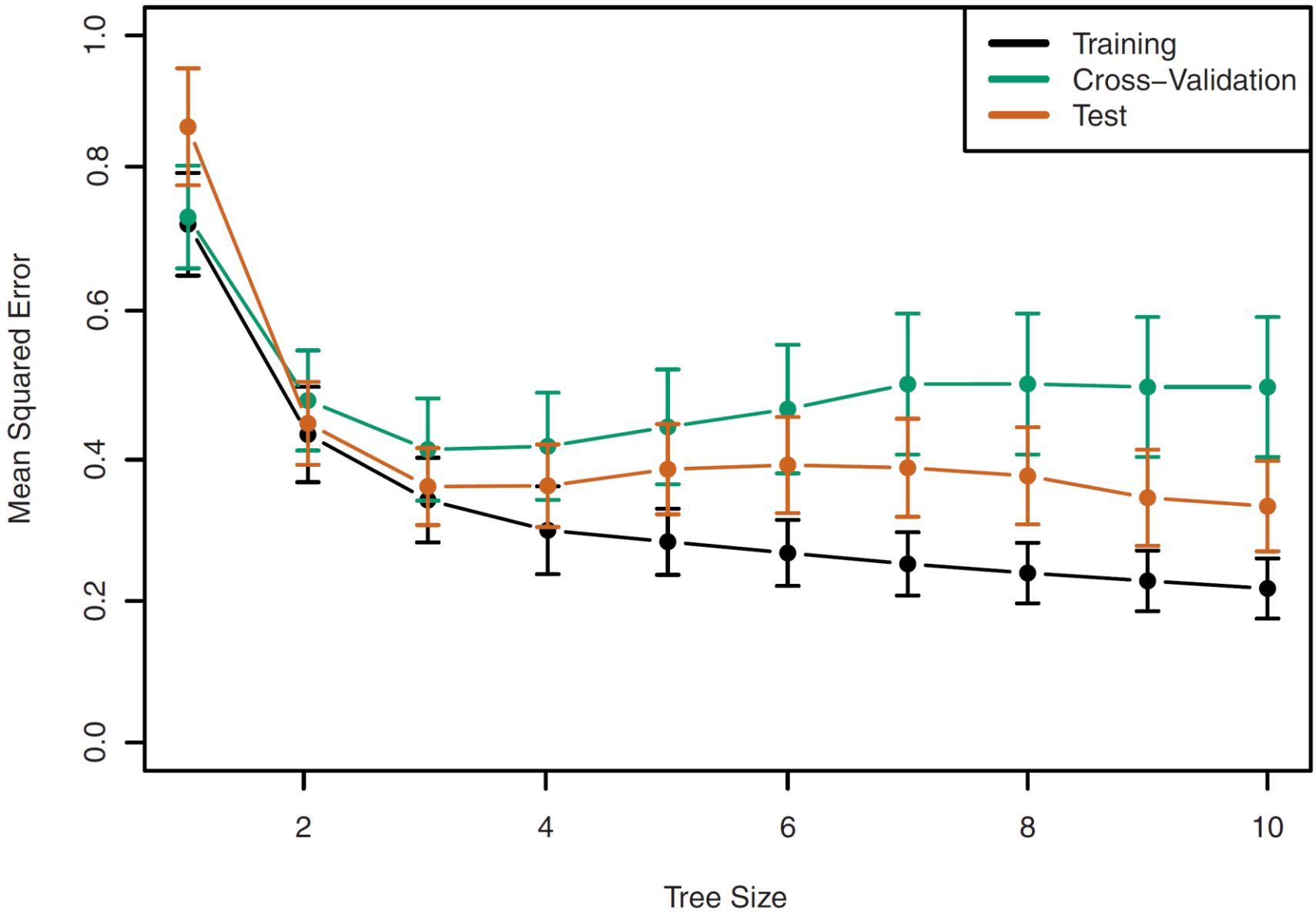
Ripley (1996) *Pattern Recognition and Neural Networks*. Cambridge University Press.

Example T_0 for predicting baseball player salary



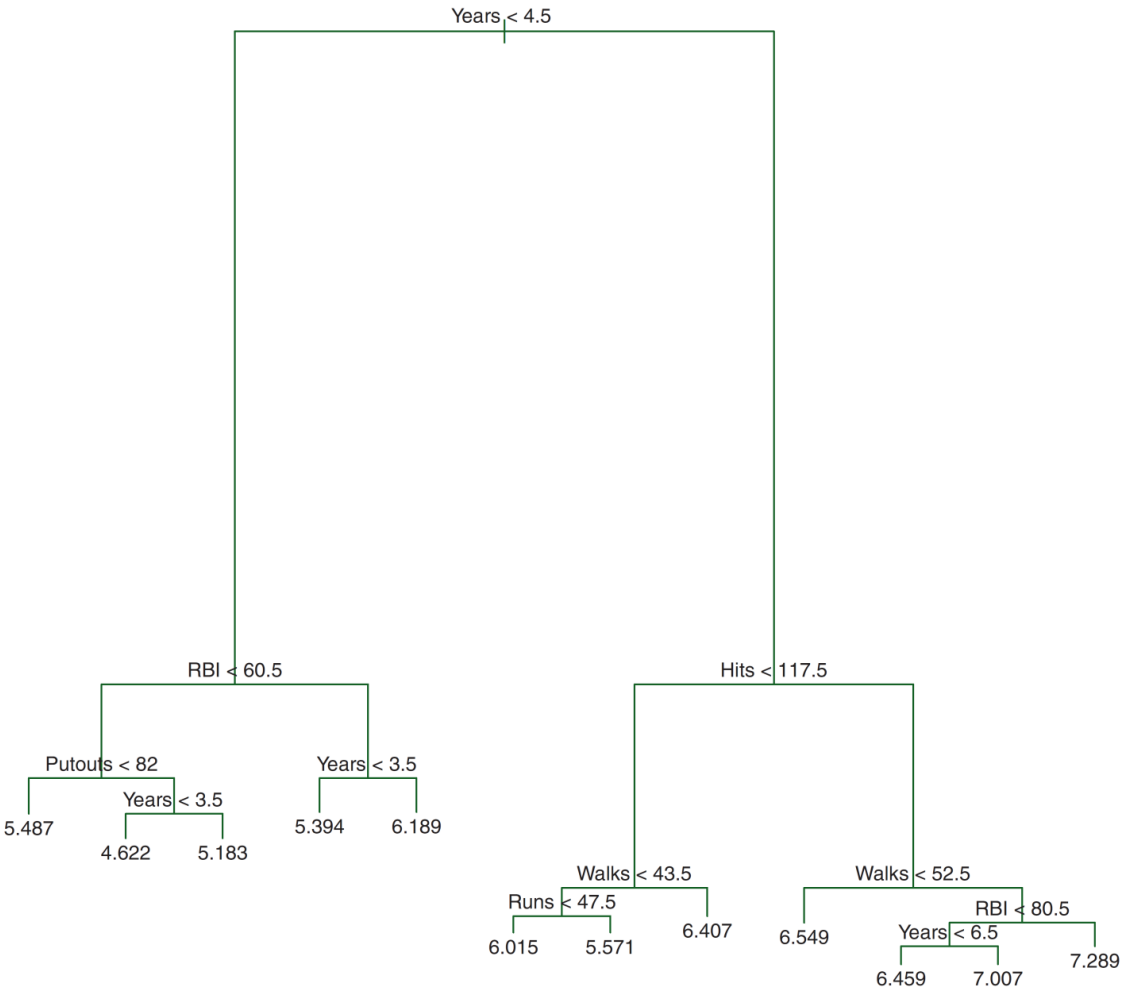
James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

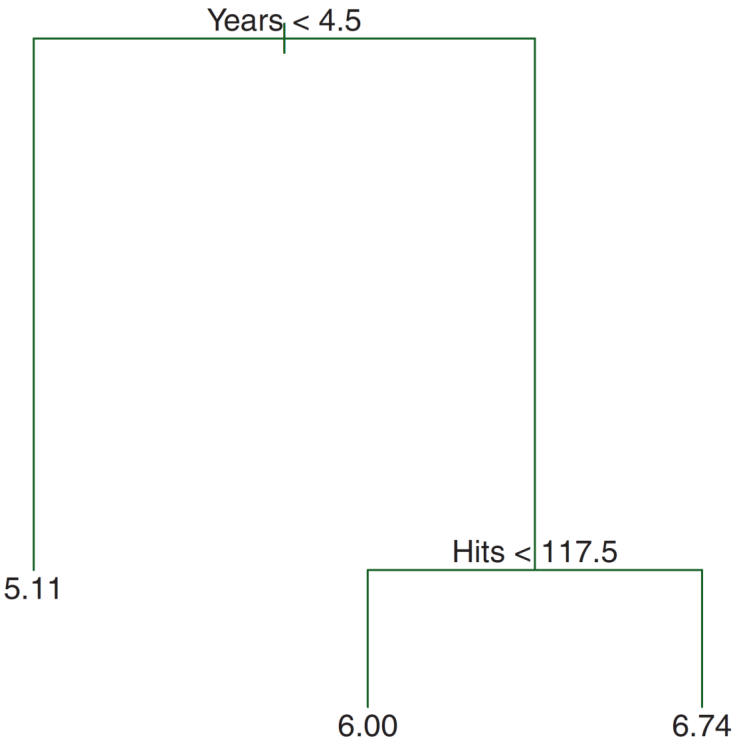
Best subtree as $|T| = 3$ regions



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Comparison of T_0 and best tree





James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Algorithm for building a regression tree

Step 1: Use recursive binary splitting to grow large tree T_0 on training data, and stop when each terminal node has fewer than some minimum number of observations.

Step 2: Obtain sequence of best trees as function of α from T_0 through cost complexity pruning.

Step 3: Use K-fold cross validation to choose α.

Step 4: Return the subtree T from Step 2 that corresponds to the value of α chosen from K-fold cross validation in Step 3.

Classification trees

Classification trees are similar to regression trees, except that they are used to predict qualitative (or categorical) responses rather than quantitative responses.

Under regression trees, the predicted response for an observation is the mean response for all training observations falling into the same region.

In contrast, for classification trees, we predict the **most commonly occurring class** (or **mode**) of the training observations falling in the same region.

When interpreting results, we may often use the estimated **class proportion** $\hat{p}_{mk}$, which is the fraction of training observations supporting class k in region m.

Building a classification tree

Building a classification tree is similar to a regression tree, with a different cost function.

That is, we cannot use the RSS as an optimization criterion for making the binary splits.

An alternative is to use the **classification error rate**

$$E = 1 - \max_k \hat{p}_{mk}$$

because we plan on assigning an input observation falling in that region to the most commonly occurring class among training observations falling within that region.

Alternate measures to grow the classification tree

In practice, the classification error rate is not sensitive enough for tree-growing (*i.e.*, it does not have good enough resolution).

Alternatives are the **Gini index**

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} \hat{p}_{mk}^2$$

and the **cross-entropy index**

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

These measures are more sensitive to node class proportions than the misclassification error E.

The Gini index G

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} \hat{p}_{mk}^2$$

The Gini index is referred to as a measure of node purity, in that small values of G indicate a node contains predominantly observations from a single class.

To solidify this, letting $\hat{p}_{m1} = 1$ and $\hat{p}_{m2} = \cdots = \hat{p}_{mK} = 0$ gives $G = 0$, with this situation representing a scenario in which a single class is represented at the node (a pure node).

Moreover, it can be shown that G is maximized with $\hat{p}_{mk} = 1/K$ for each $k = 1,2,\ldots,K$ (*i.e.*, when all class proportions are the same).

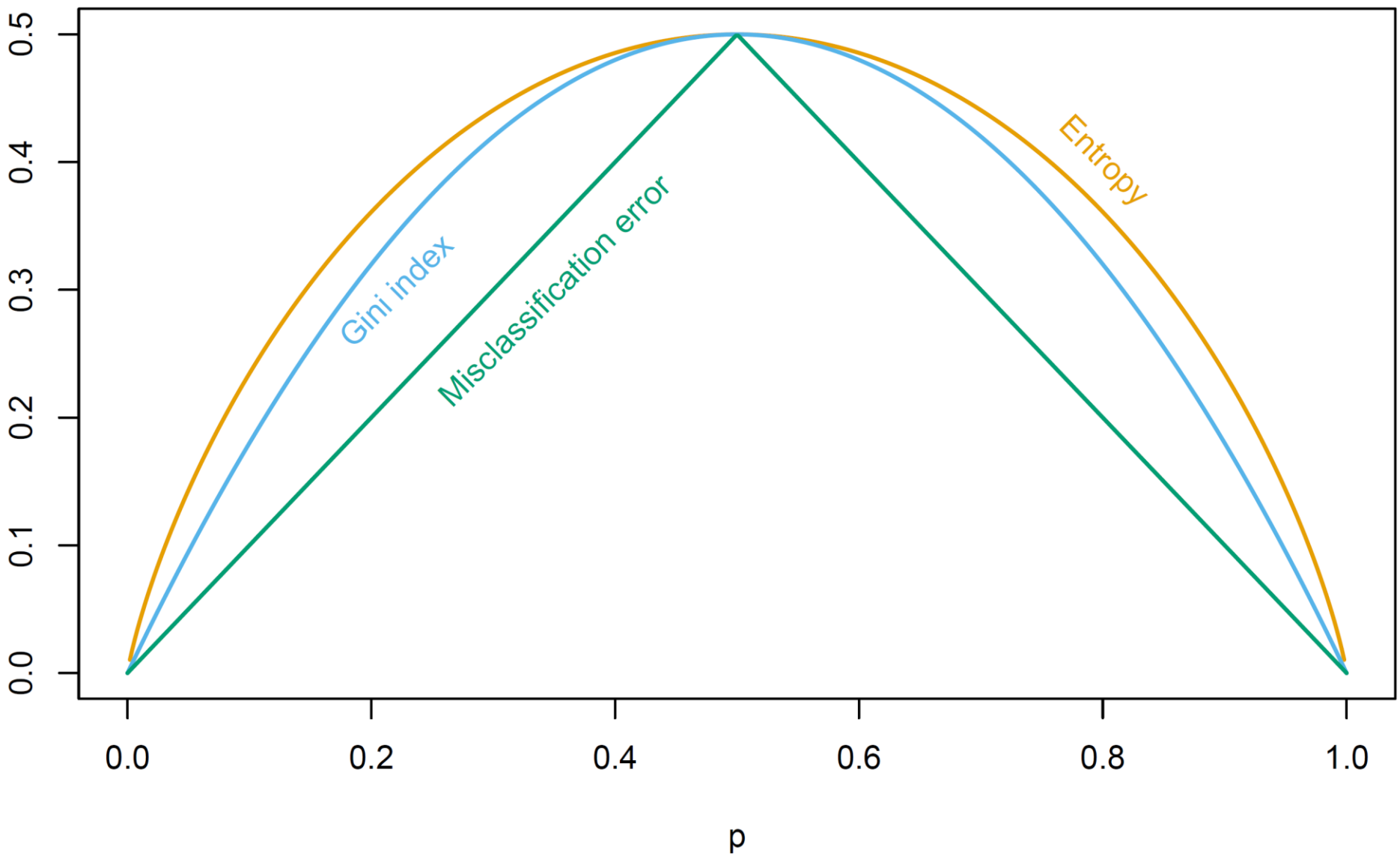
The cross-entropy index D

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

The cross-entropy index D takes values near 0 if a node is close to pure, and also achieves its maximum with $\hat{p}_{mk} = 1/K$ for each $k = 1,2,\ldots,K$ (*i.e.*, when all class proportions are the same).

Numerically, the Gini (G) and the cross-entropy (D) indices are highly correlated, and will yield similar results when growing a tree based on them.

Node (im)purity measures for $K = 2$ and $p = p_{m2}$



Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Example illustrating insensitivity of E

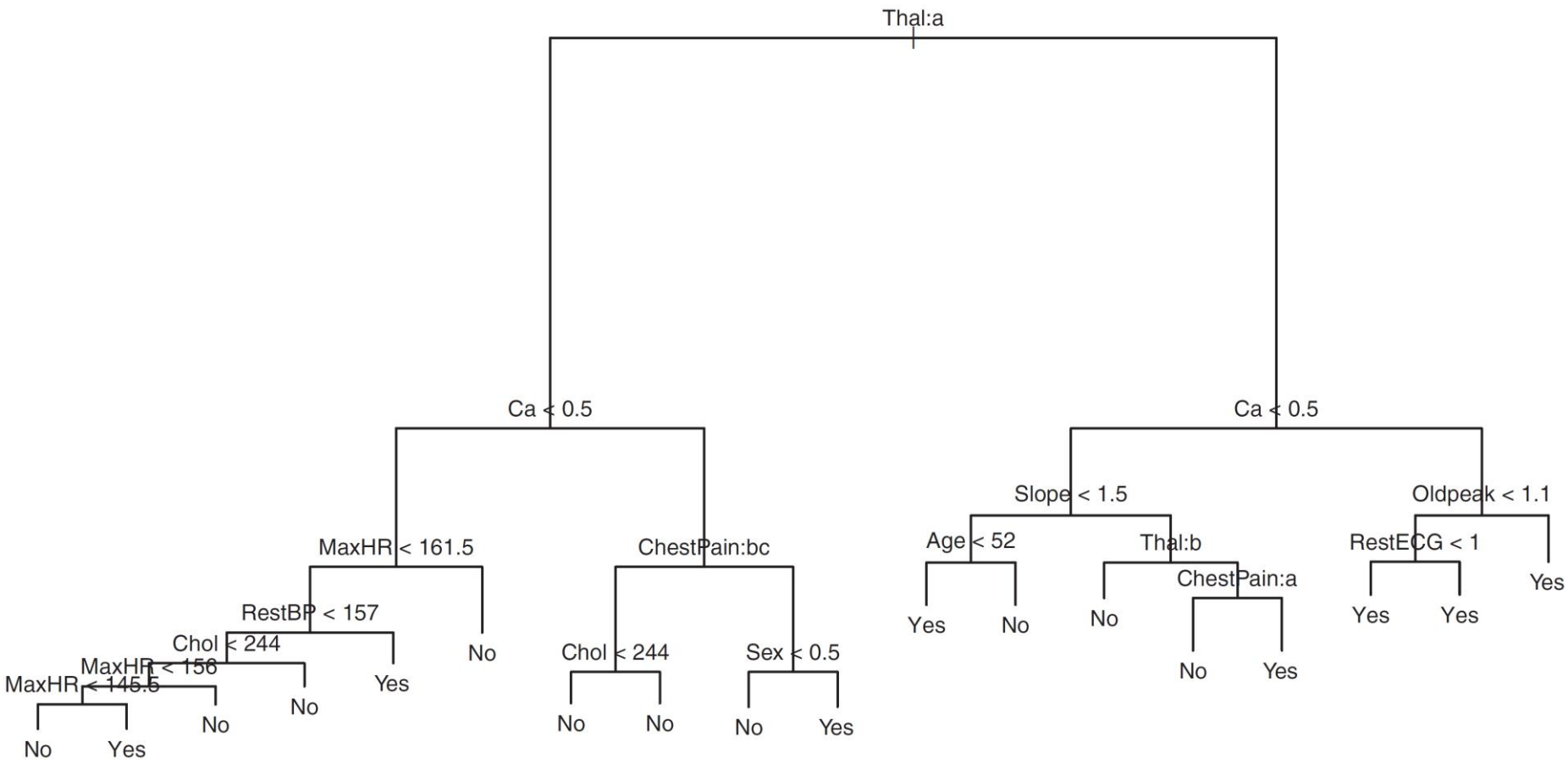
Consider a problem with $K = 2$ classes and $N/2 = 400$ training observations in each class, which we denote as $(400,400)$.

Suppose one split created nodes $(300,100)$ and $(100,300)$, while the other created nodes $(200,400)$ and $(200,0)$.

Both splits produce a misclassification error $E = 1/4$, but the second split produces a pure node (node 2) that is preferable.

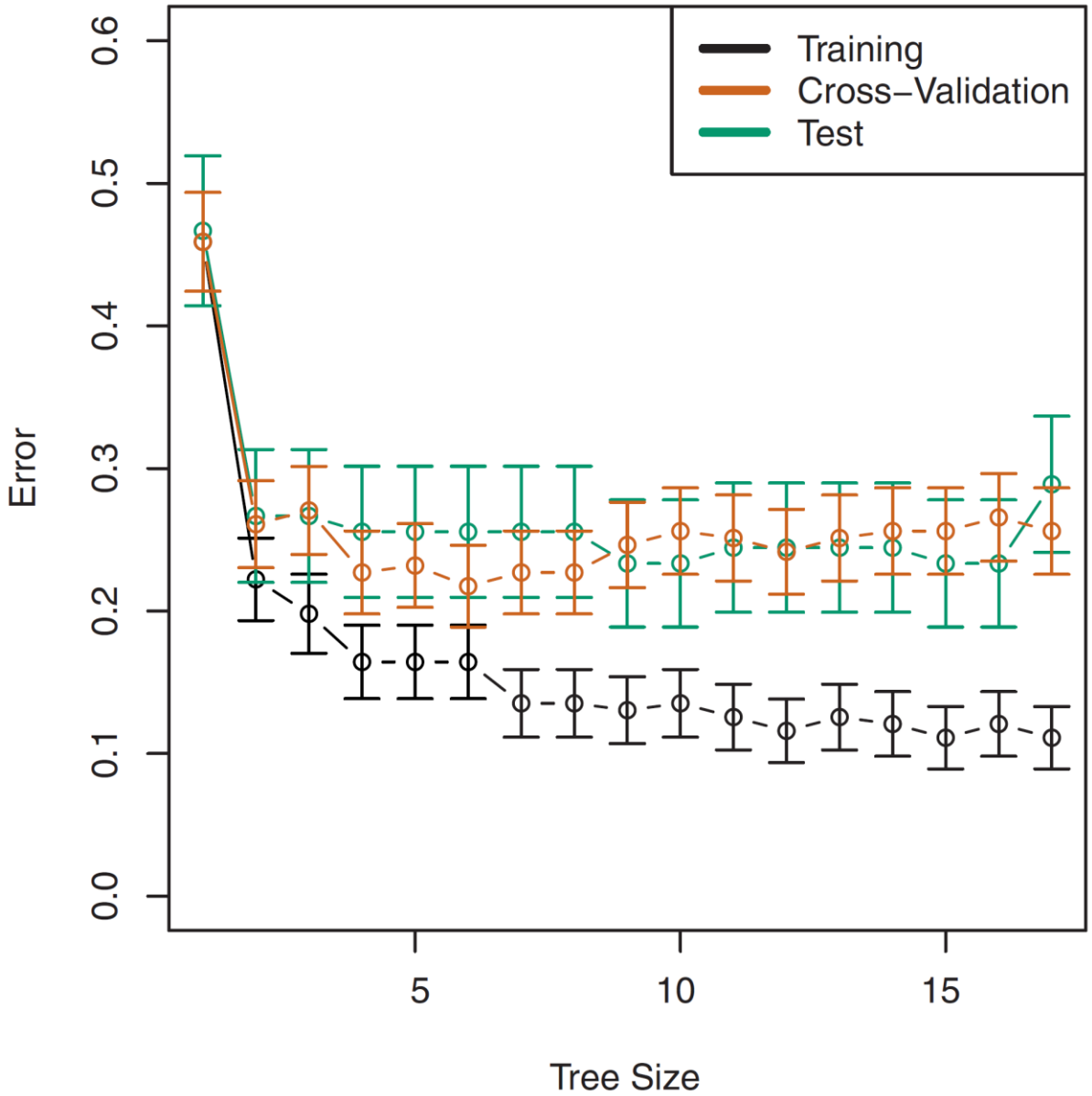
The Gini (G) and cross-entropy (D) indices would be lower for the second split, and for this reason, these indices are preferable for growing trees.

Example T_0 for predicting heart disease



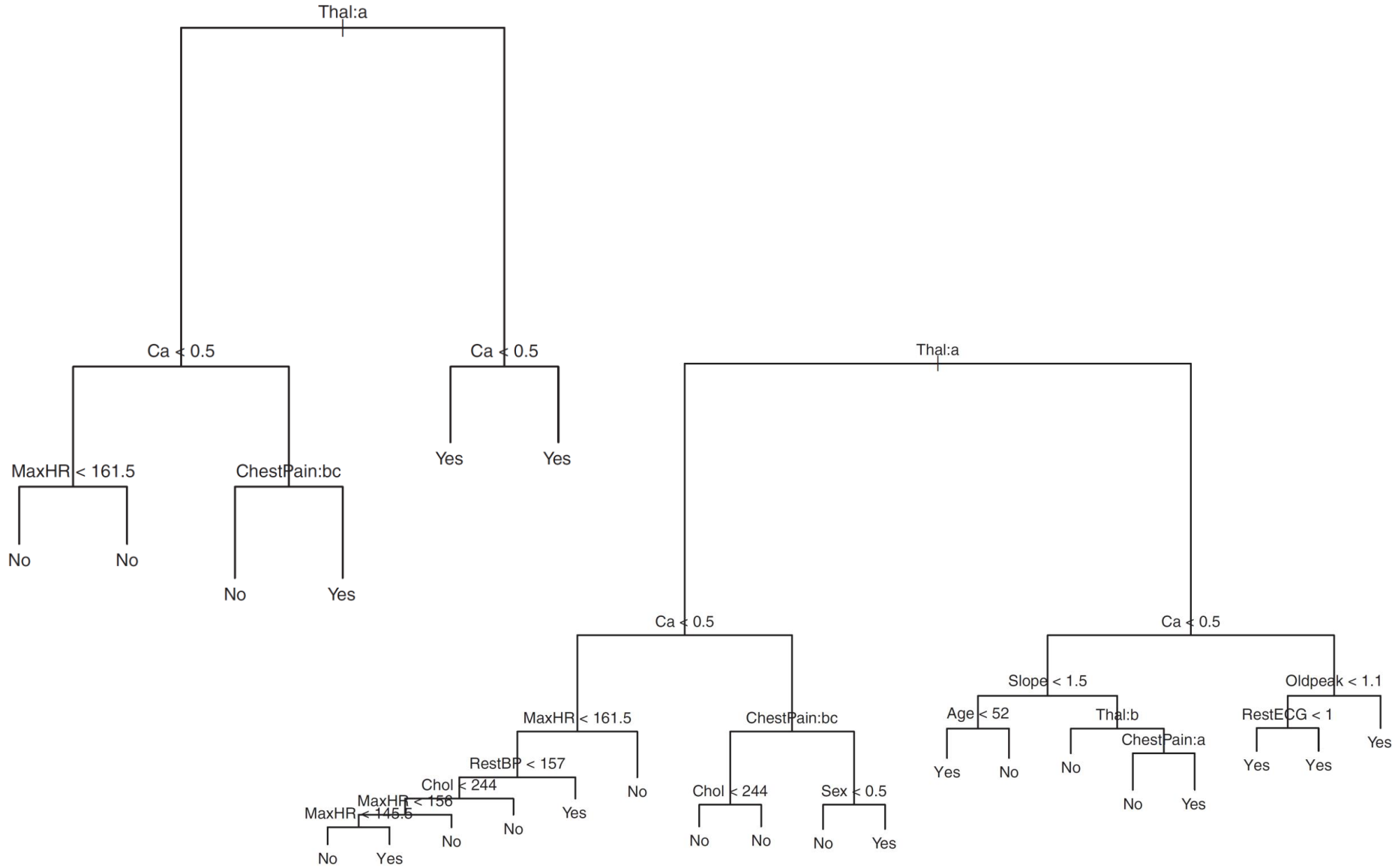
Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Best subtree as $|T| = 6$ regions



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Comparison of T_0 and best tree



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Trees versus linear models

Regression and classification trees are quite different from the classical regression and classification approaches we have considered in this class.

Specifically, the linear regression model assumes a form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

whereas a regression tree assumes a model of the form

$$f(X) = \sum_{m=1}^{M} \hat{y}_{R_m} I(X \in R_m)$$

where $\hat{y}_{R_m}$ is the mean response for training observations falling within region R_m, $m = 1,2, \dots, M$.

Trees versus linear models

$$\text{Linear regression: } f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

$$\text{Regression tree: } f(X) = \sum_{m=1}^{M} \hat{y}_{R_m} I(X \in R_m)$$

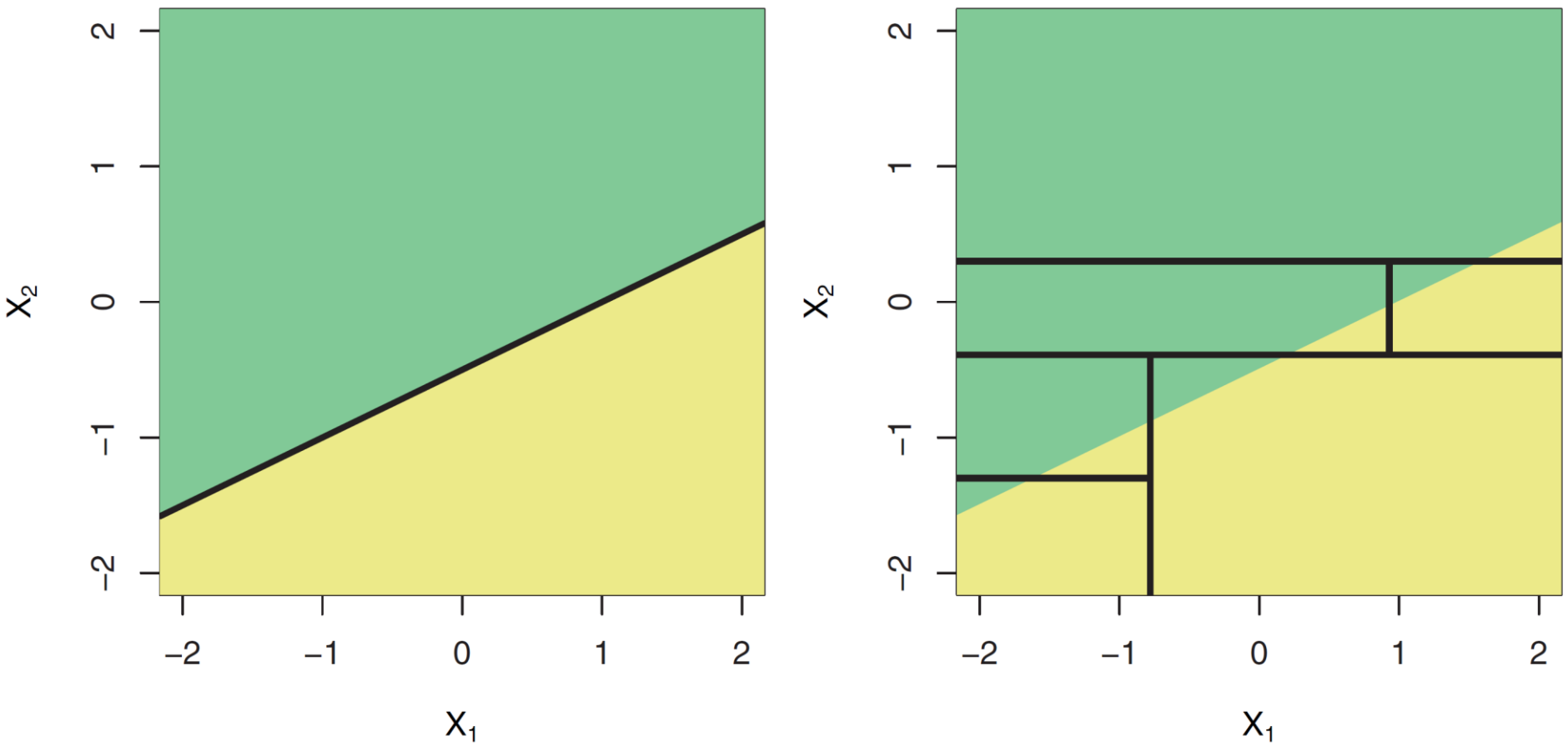
How do we know which model is better? As with all other methods considered in this class, it will depend on the problem at hand.

If the response to feature relationship is well-approximated by a linear model, then linear regression will work well and outperform other approaches like regression trees.

If there is a highly non-linear and complex relationship, then a regression tree may outperform linear regression.

Example comparing classification trees to linear model

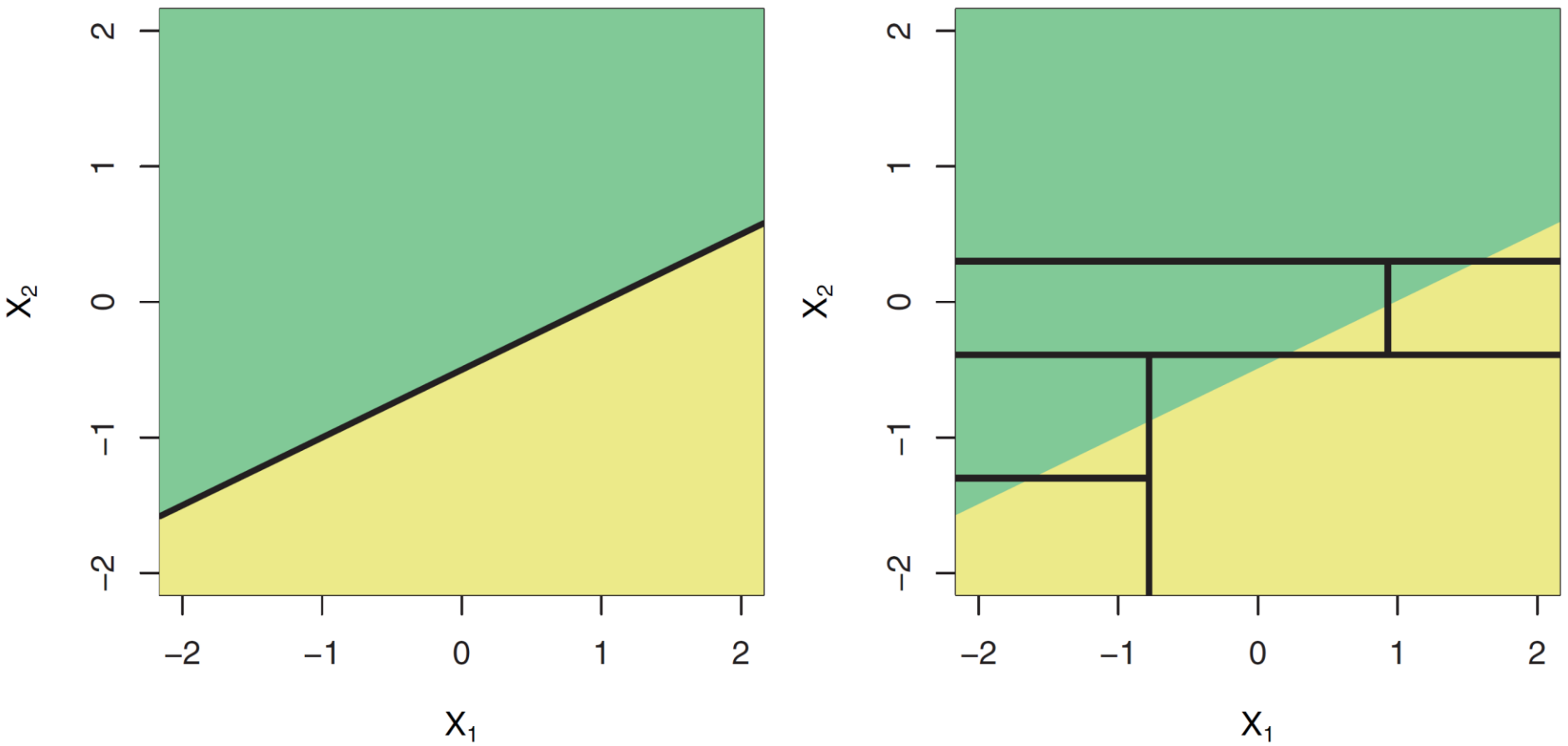
Classification problem with $K = 2$ classes and $p = 2$ features, with a linear decision boundary, separating the yellow and green classes.



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example comparing classification trees to linear model

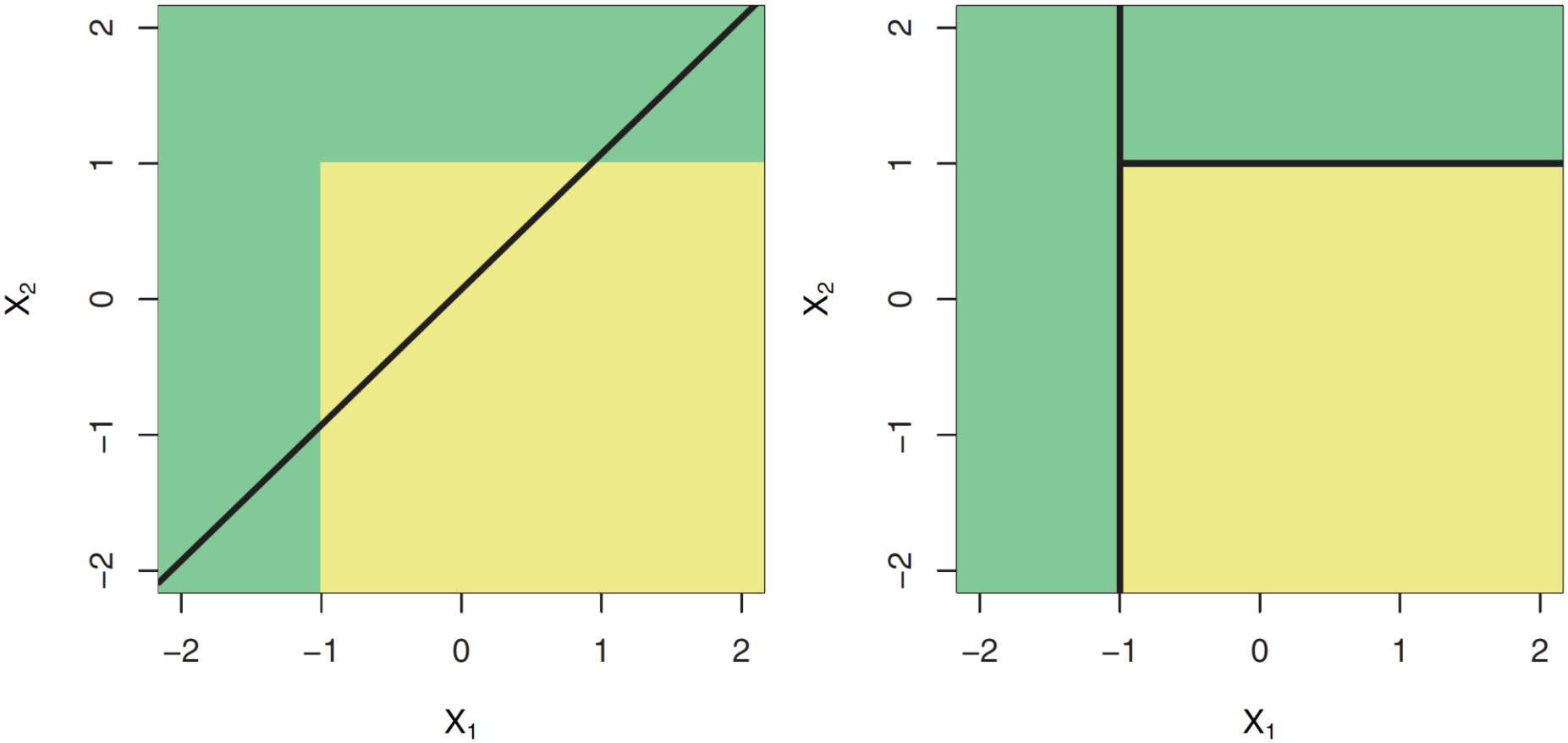
A linear model (left) is able to easily capture the linear decision boundary (estimated hyperplane in black), whereas the classification tree (right) was unable to accurately capture this linear relationship.



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example comparing classification trees to linear model

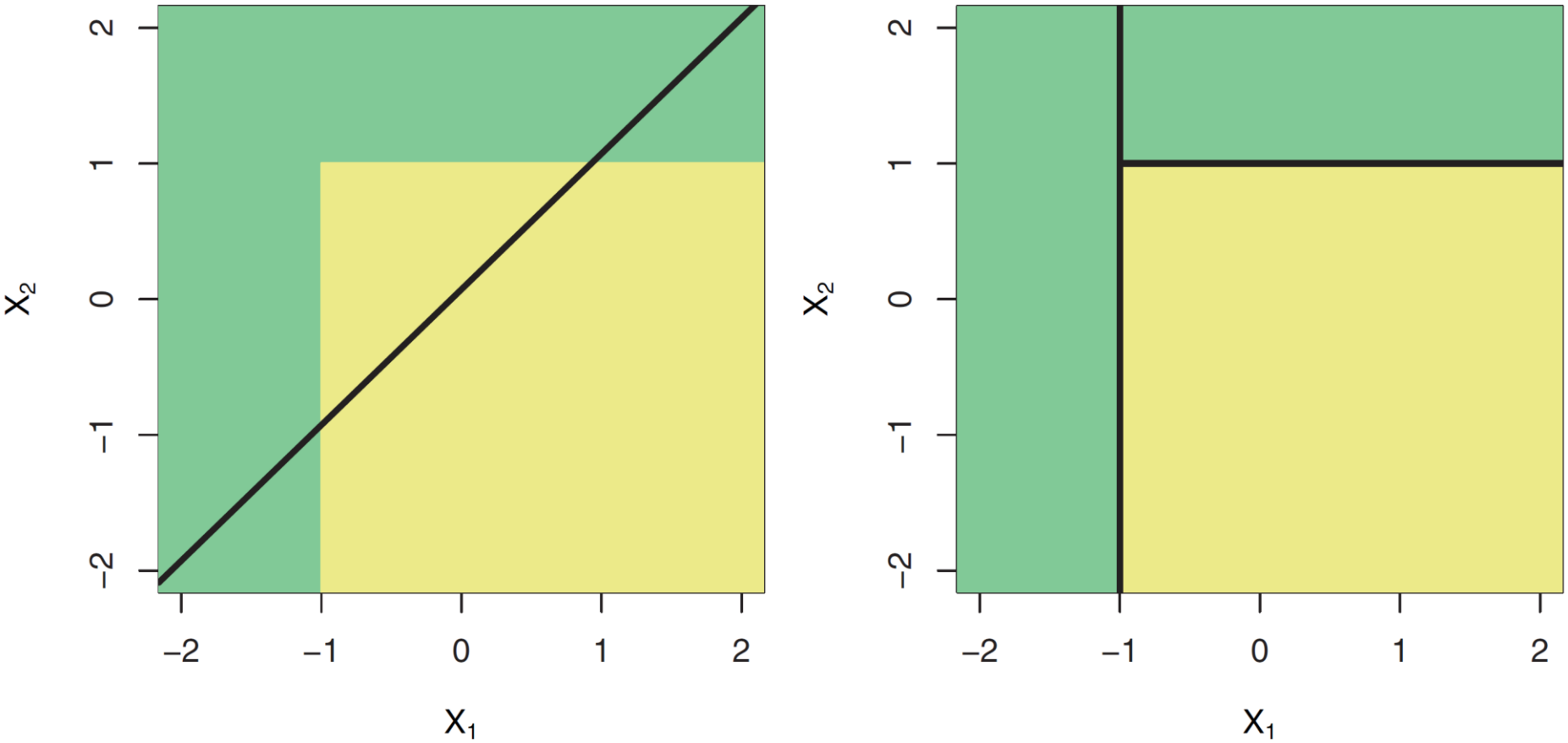
Classification problem with $K = 2$ classes and $p = 2$ features, with a complex non-linear decision boundary, separating the yellow and green classes.



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example comparing classification trees to linear model

A linear model (left) is unable to capture the non-linear decision boundary (estimated hyperplane in black), whereas the classification tree (right) was able to accurately capture this non-linear relationship.



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Advantages and disadvantages of trees

Advantages:

Easy to explain model and to interpret.

Trees can be displayed graphically.

Can easily handle qualitative features without creating dummy variables.

Disadvantages:

Generally do not have same predictive accuracy as a number of the regression and classification approaches we have considered.

Trees can be highly non-robust, in that small changes in the input data can yield large changes in the predicted model (*i.e.*, the model has high variance).

Improving the variance of decision trees

It would helpful if we could improve the variance of decision trees, while also maintaining some of their nice properties.

One way of doing this is a technique called bagging.

But before we discuss bagging, we need to introduce a general resampling technique, termed the bootstrap.

Bootstrap

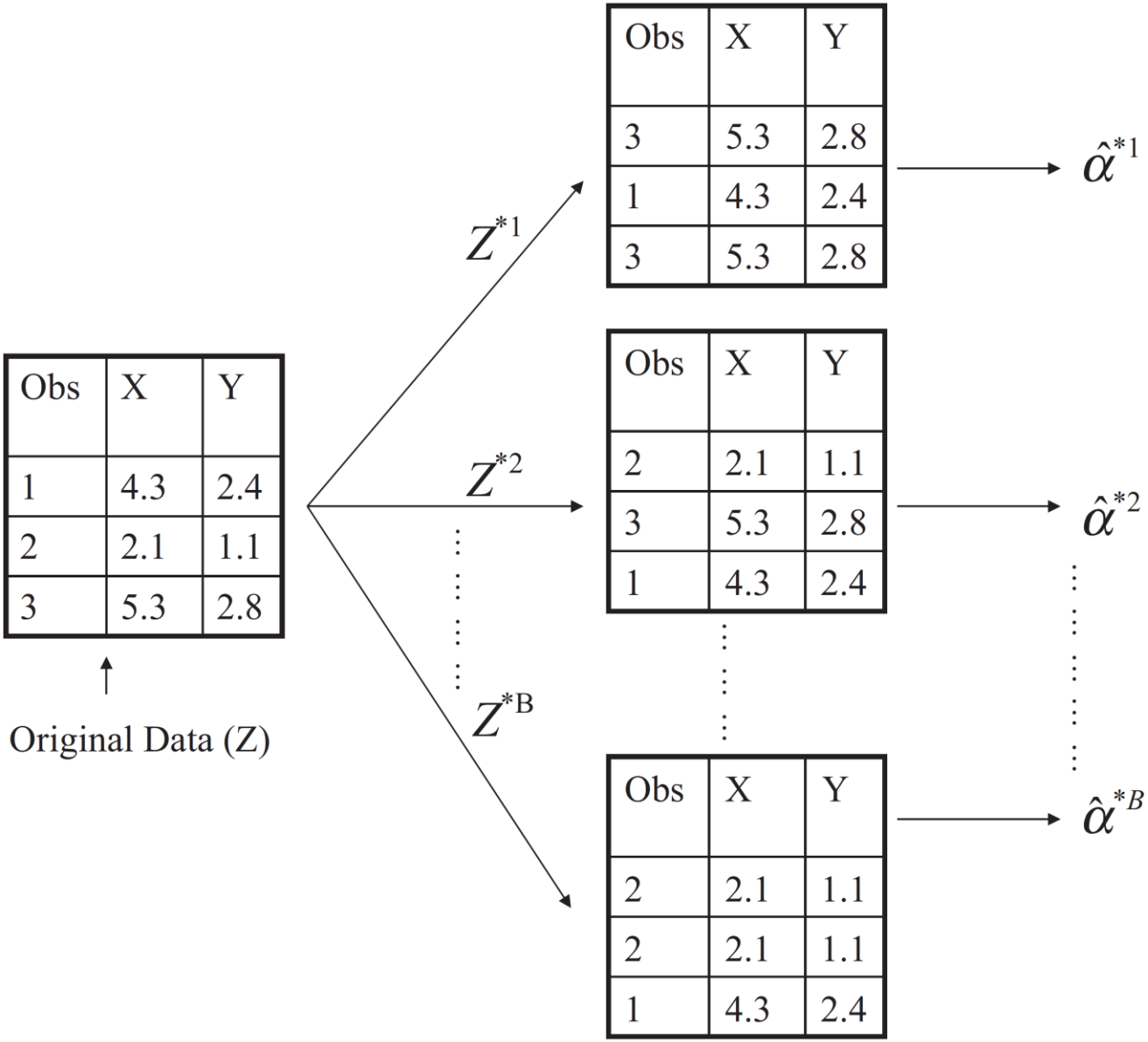
Bootstrap is a statistical technique that can quantify uncertainty associated with a statistical learning approach or some other statistical estimator.

For a sample of N observations, bootstrap proceeds by sampling N observations **with replacement** to create a new replicate dataset.

This procedure is repeated B times, and the distribution of estimates across the B replicates can provide measures of uncertainty.

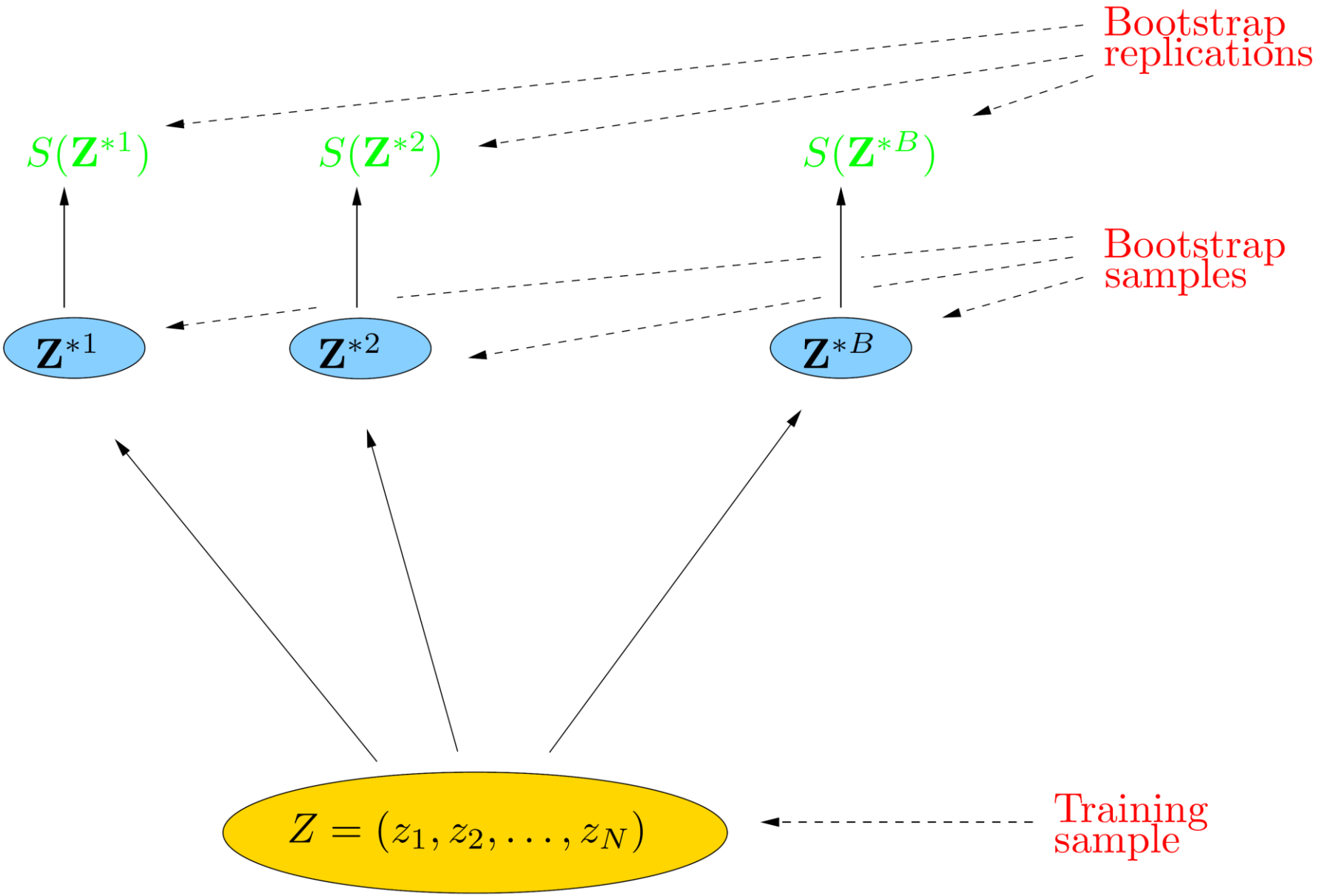
These replicates can also be averaged, thereby creating potentially more robust estimators.

Example with $N = 3$ observations



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Schematic illustrating the bootstrap procedure

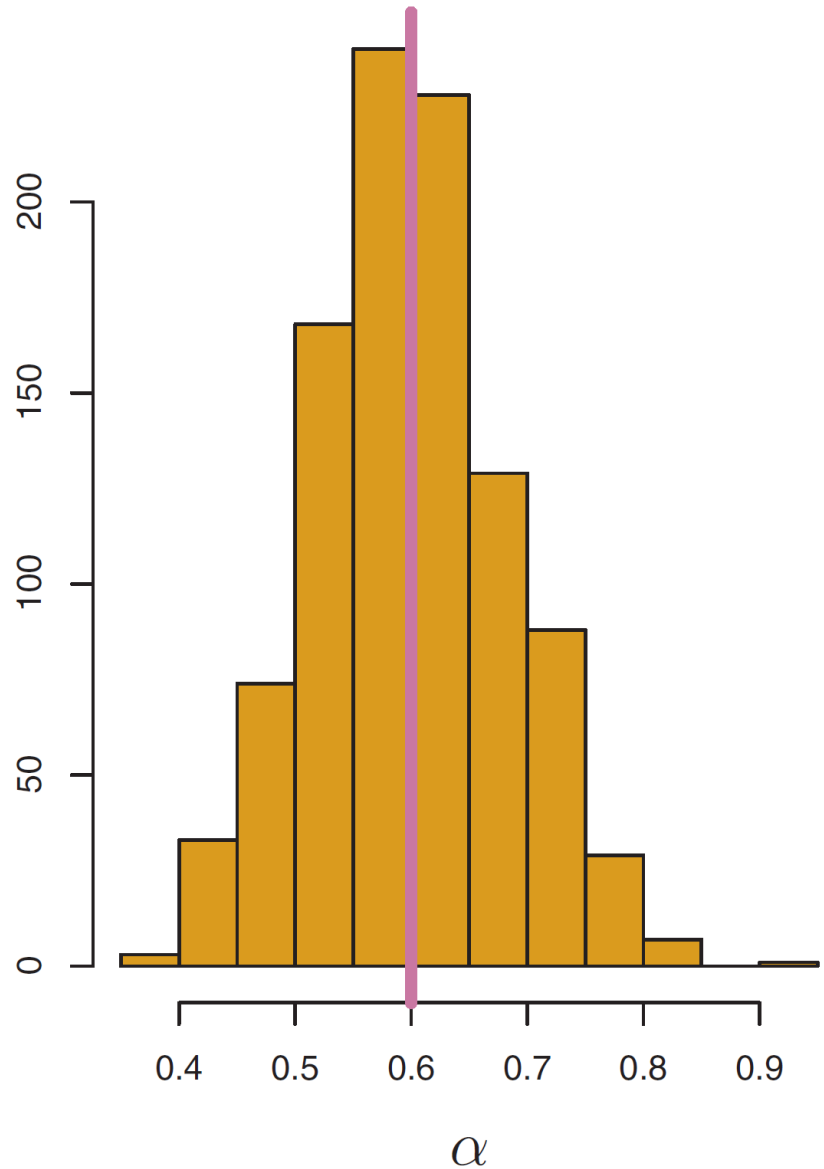


Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Consider estimating some parameter α

Histogram of the estimates of a parameter α from $1{,}000$ independent simulated datasets from the true population.

Vertical pink line indicates the true value of α, highlighting that with enough independent samples, the mean of the samples is a good estimate of the parameter here.

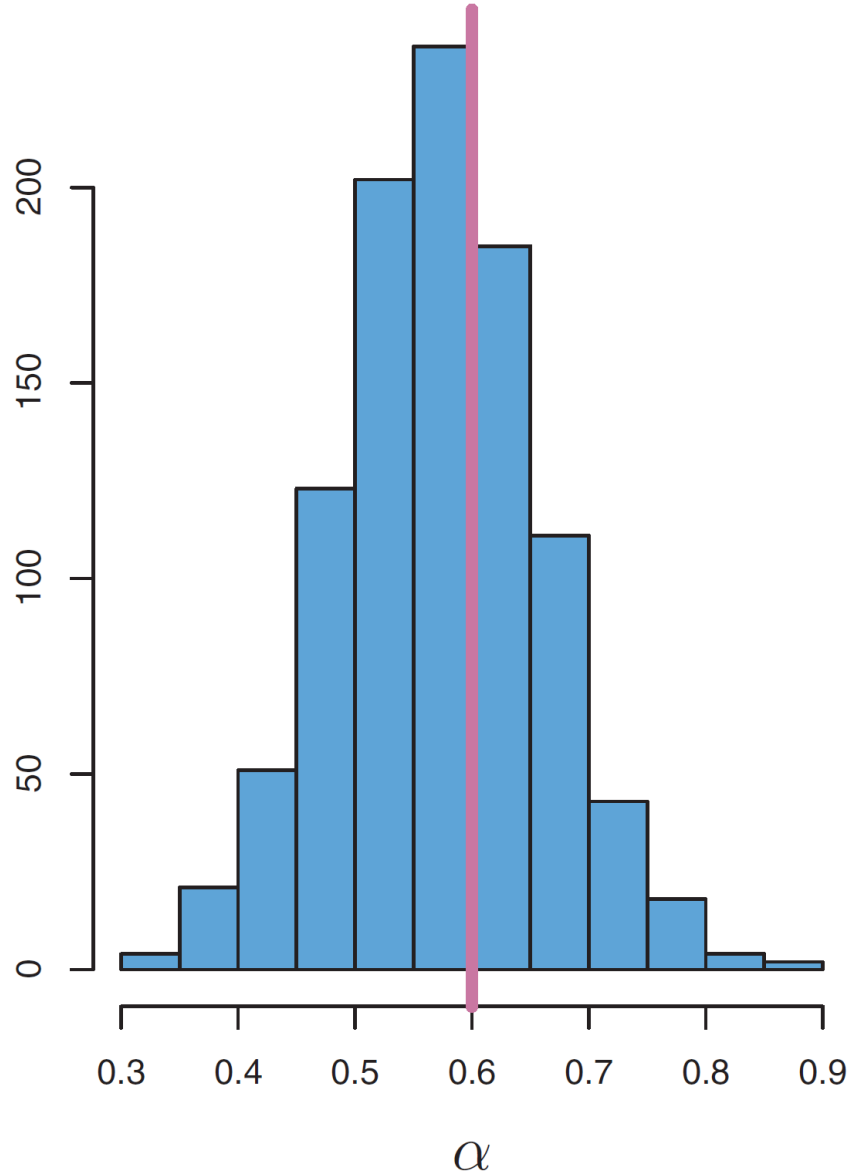


Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Consider estimating some parameter α

Histogram of the estimates of a parameter α from $1{,}000$ bootstrap replicates sampled from a single dataset.

Vertical pink line indicates the true value of α, highlighting that with enough bootstrap samples, the mean of the samples here slightly underestimates the parameter, yet this distribution looks pretty similar to the $1{,}000$ independent datasets, even though it was generated from a single dataset.



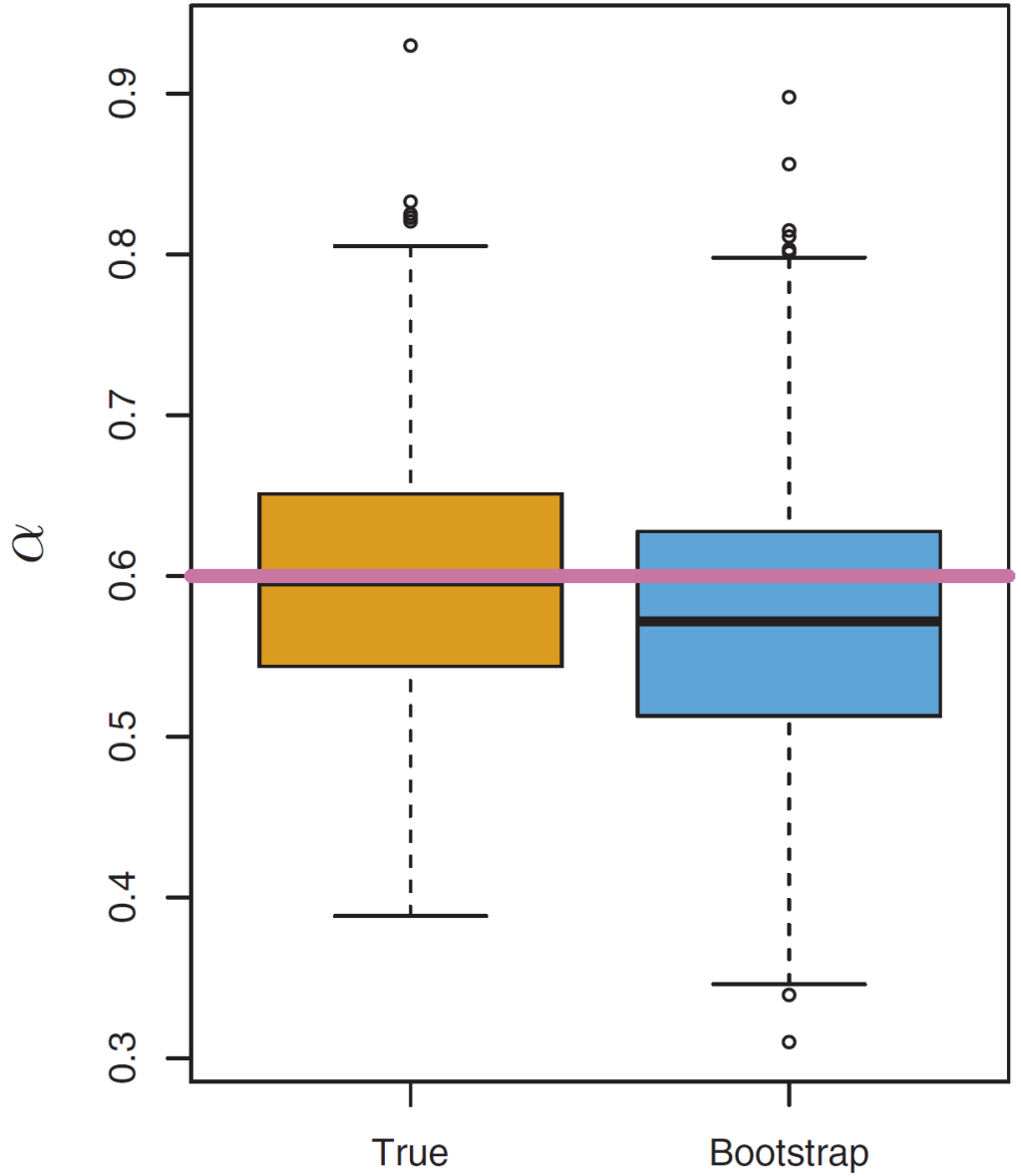
Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Consider estimating some parameter α

Box plots summarizing the distributions of α estimates across $1{,}000$ independent simulated replicates (orange) and $1{,}000$ bootstrap replicates from a single dataset (blue).

Horizontal pink line indicates the true value of α.

Bootstrap replicates do a fairly good job at capturing the distribution of parameter estimates.



Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Consider estimating some parameter α

We can obtain estimates of the mean of a parameter α across B bootstrap replicates by using the estimates $\hat{\alpha}_b$, $b = 1,2,\ldots,B$, for each bootstrap replicate b.

For example, we can estimate the mean as

$$\bar{\alpha} = \frac{1}{B}\sum_{b=1}^{B}\hat{\alpha}_b$$

and the variance as

$$\hat{\sigma}_\alpha^2 = \frac{1}{B-1}\sum_{b=1}^{B}(\hat{\alpha}_b - \bar{\alpha})^2$$

Adapted from James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Application to statistical learning algorithms

So how can this help with statistical learning algorithms?

Suppose we are trying to train a model to estimate a response Y from input $X \in \mathbb{R}^p$ from N training observations (x_i, y_i), $i = 1,2, \dots, N$.

If we are using a flexible method to train a model, then the estimate will have high variance but low bias, and hence the estimator will have on average the value equal to the true parameter.

Because we cannot improve the bias, we can only improve the overall performance of the method (*i.e.*, the MSE) by reducing the variance.

Bagging

We can help reduce variance through **bagging** (**bootstrap aggregation**), by developing an estimator from B estimators, each trained on a separate bootstrap replicate.

Suppose we train a model on each of B bootstrap replicates, where the estimate of the trained model is $\hat{f}_b(x)$, $b = 1,2,\ldots,B$, for bootstrap replicate b.

The bagged estimator is then simply the average estimate

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)$$

Bagging improves variance

Because each estimate generated by bagging is identically distributed, the expectation of the average of B estimators is the same as the average of any of them.

Therefore the bias of a bagged estimator is the same as that of an individual estimator, which is excellent for highly-flexible models.

However, variance of B correlated identically-distributed random variables (*i.e.*, bootstrap replicates), with positive pairwise correlation coefficient ρ, is

$$\mathrm{Var}\left[\hat{f}_{\mathrm{bag}}(x)\right] = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

where $\mathrm{Var}\left[\hat{f}_b(x)\right] = \sigma^2$ is the variance of bootstrap replicate b.

Bagging improves variance

$$\mathrm{Var}\left[\hat{f}_{\mathrm{bag}}(x)\right] = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Note that with bootstrap, on average roughly $2/3$ (0.632) of the original N observations will appear in any bootstrap replicate (Efron 1979).

Therefore bootstrap replicates will on average only be partially correlated (*i.e.*, $\rho < 1$).

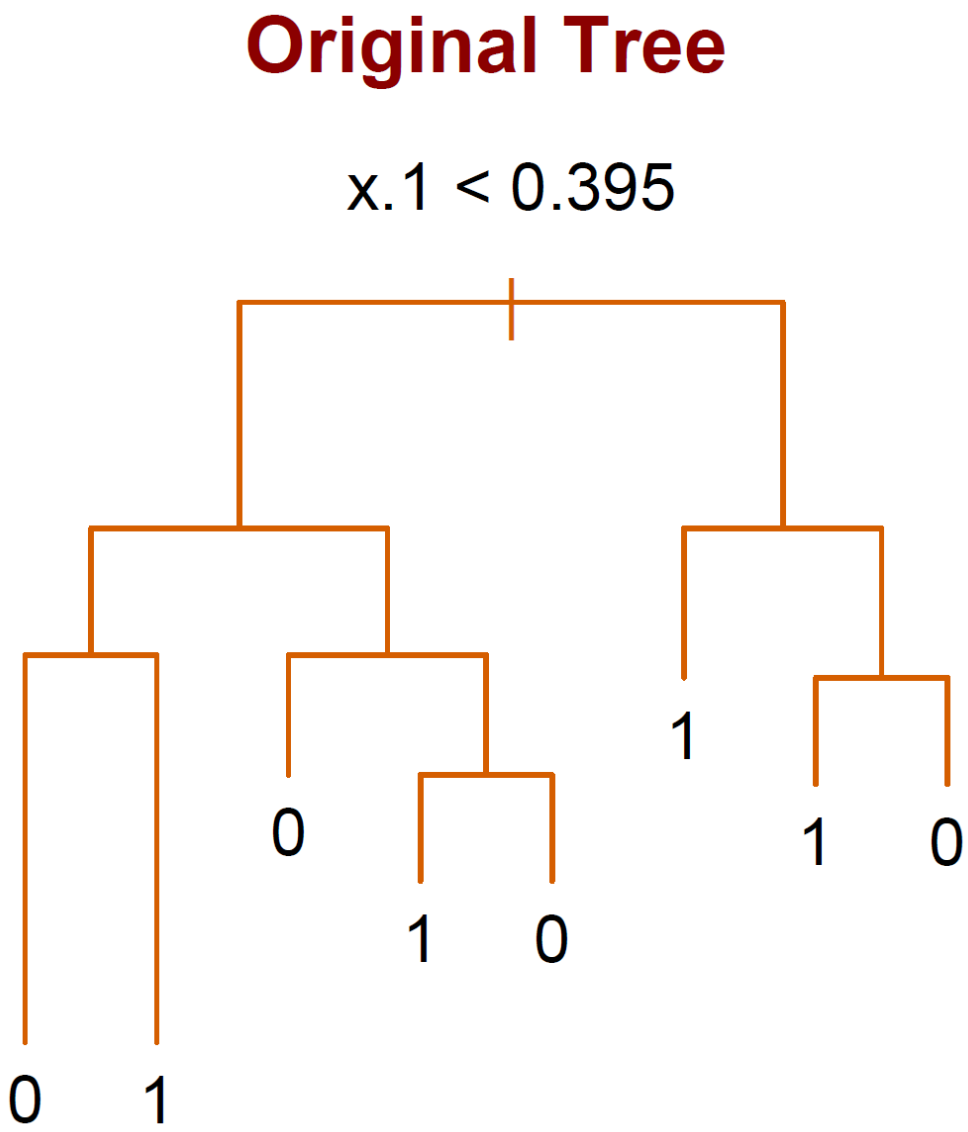
Note as B increases, variance of the bagged estimator approaches

$$\mathrm{Var}\left[\hat{f}_{\mathrm{bag}}(x)\right] \approx \rho\sigma^2 < \sigma^2 = \mathrm{Var}\left[\hat{f}_b(x)\right]$$

which is less than σ^2 as $\rho < 1$, thereby reducing the overall variance of the estimator while maintaining a small bias.

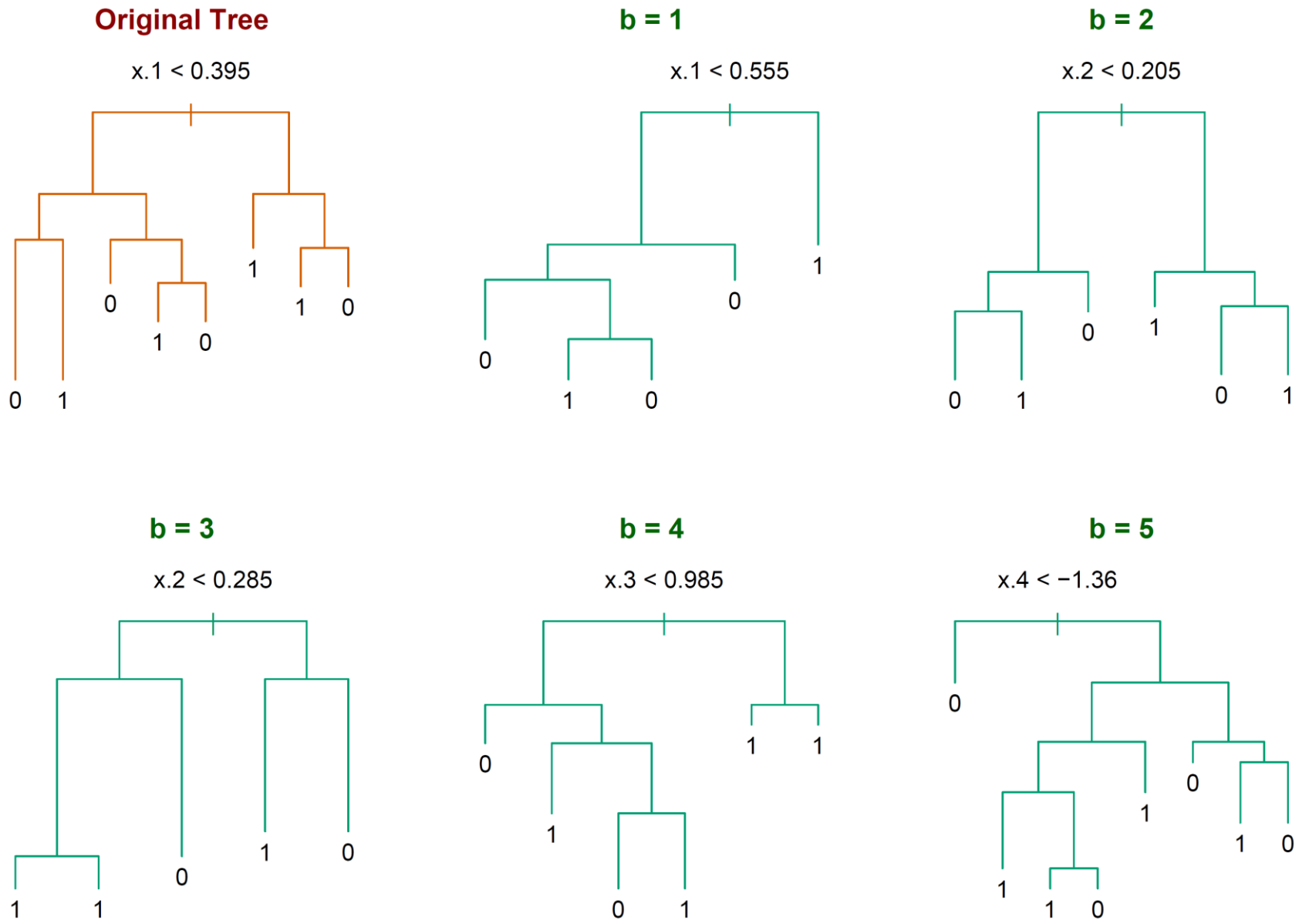
Efron (1979) *Ann Stat* 7:1-26.

Variability in root split across bagged trees



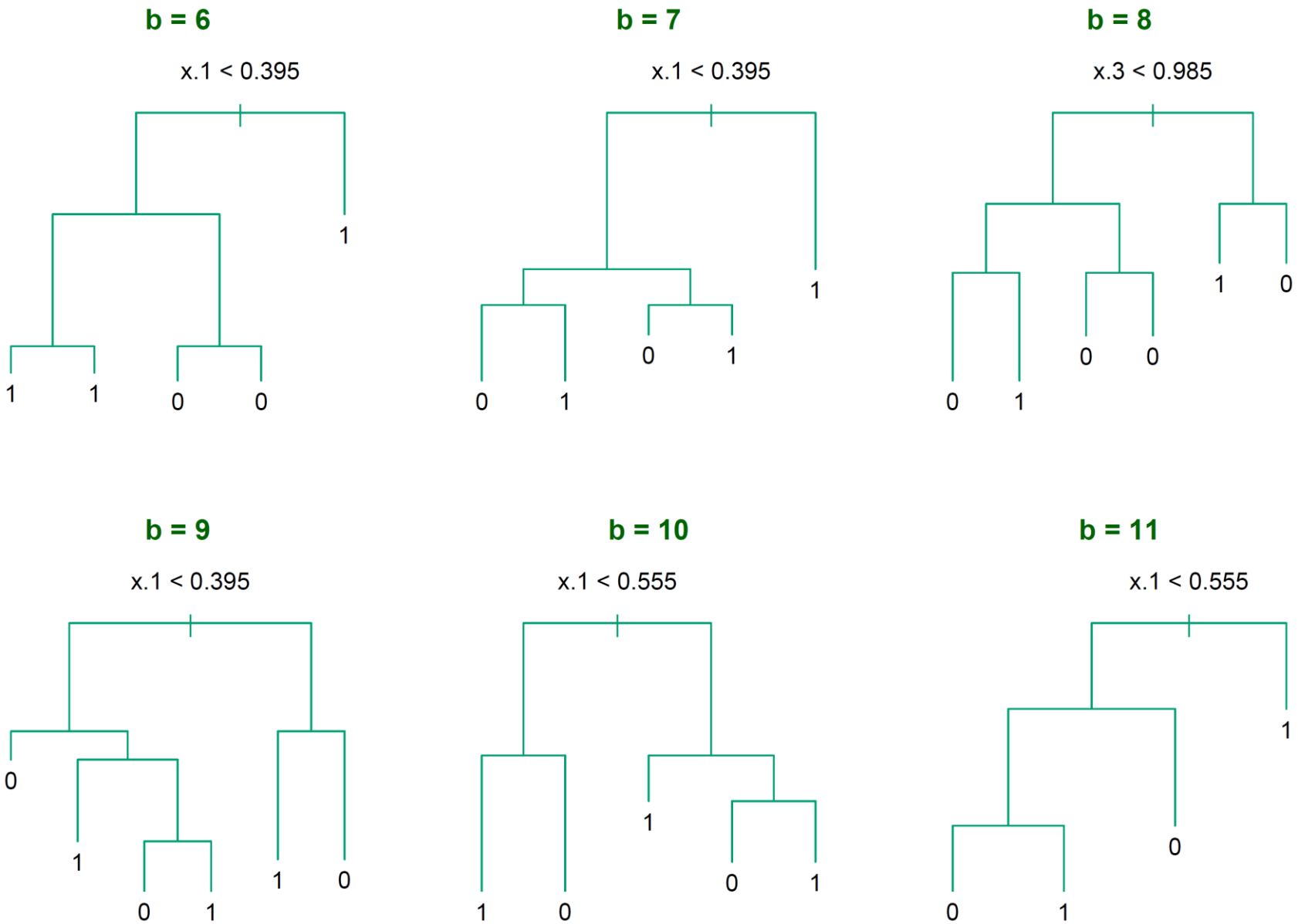
Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Variability in root split across bagged trees



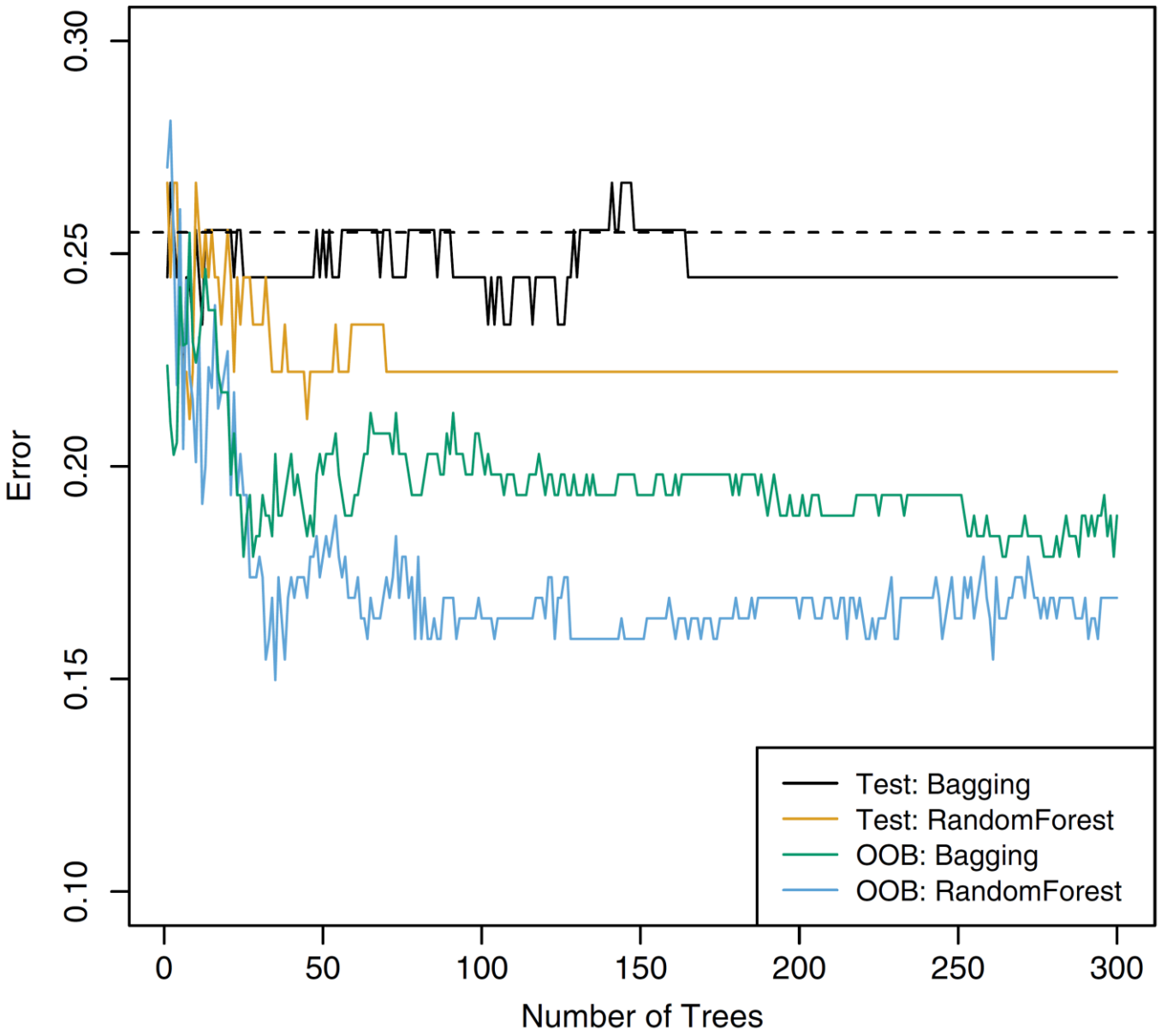
Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Variability in root cutoff across bagged trees



Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Bagging outperforms single tree (dashed line)



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Out-of-bag (OOB) error estimates

Computing cross-validation to choose the optimal number of bagged datasets B can be costly.

Out-of-bag (**OOB**) error estimation is an efficient way of assessing validation error without performing cross validation.

Because on average each bagged tree uses only approximately $2/3$ of the observations, about $1/3$ of the observations on average are not used to fit a given bagged tree

These observations are known as OOB observations.

Efron (1979) *Ann Stat* 7:1-26.

Out-of-bag (OOB) error estimates

We can then predict the response of observation i, $i = 1,2,\ldots,N$, using each of the trees in which that observation was OOB.

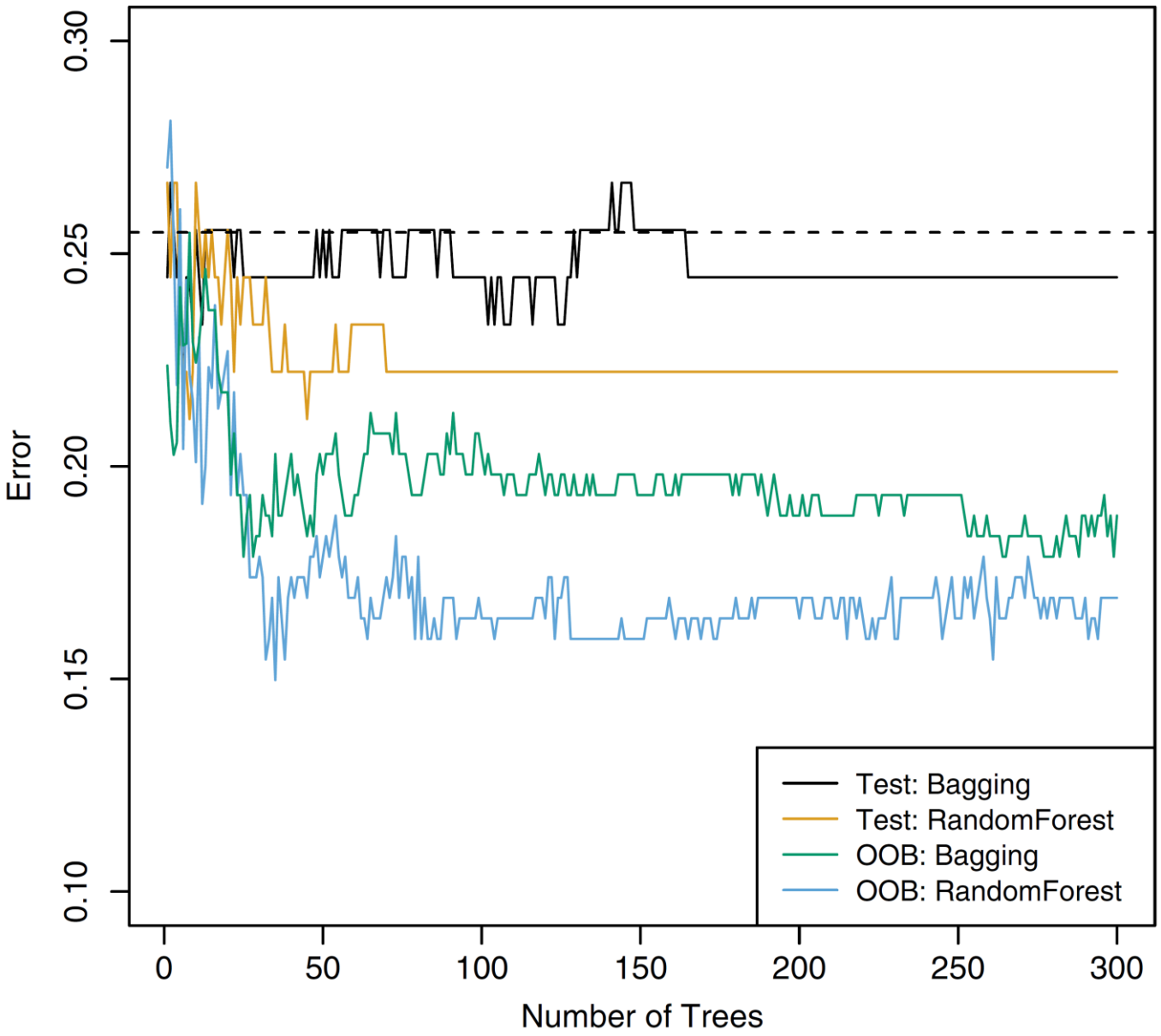
This will yield on average $B/3$ predictions for observation i, and the prediction for this observation will be the mean of the predicted responses across the roughly $B/3$ bagged trees for which it is included.

This approach provides a single OOB prediction for observation i, and the overall OOB MSE (for regression) or OOB classification error (for classification) is computed.

It turns out that this OOB error estimate is akin to leave-one-out cross validation.

James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Trend in OOB error estimate tracks test error estimate



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Bagging nullifies easy interpretability of decision trees

One advantage of decision trees is that they were easily interpreted, and also be viewed.

However, a bagged regression tree is not a tree at all, and represents some average of an **ensemble** of trees.

It is therefore not as straightforward to understand the importance of some features relative to others.

Though the collection of bagged trees can be more difficult to interpret, we can capture an overall summary of the importance of each feature by using the RSS for bagged regression trees or the Gini index for bagged classification trees.

Measuring importance of features

For bagged regression trees, we can record the amount that the RSS is decreased due to splits over that feature for each bagged regression tree.

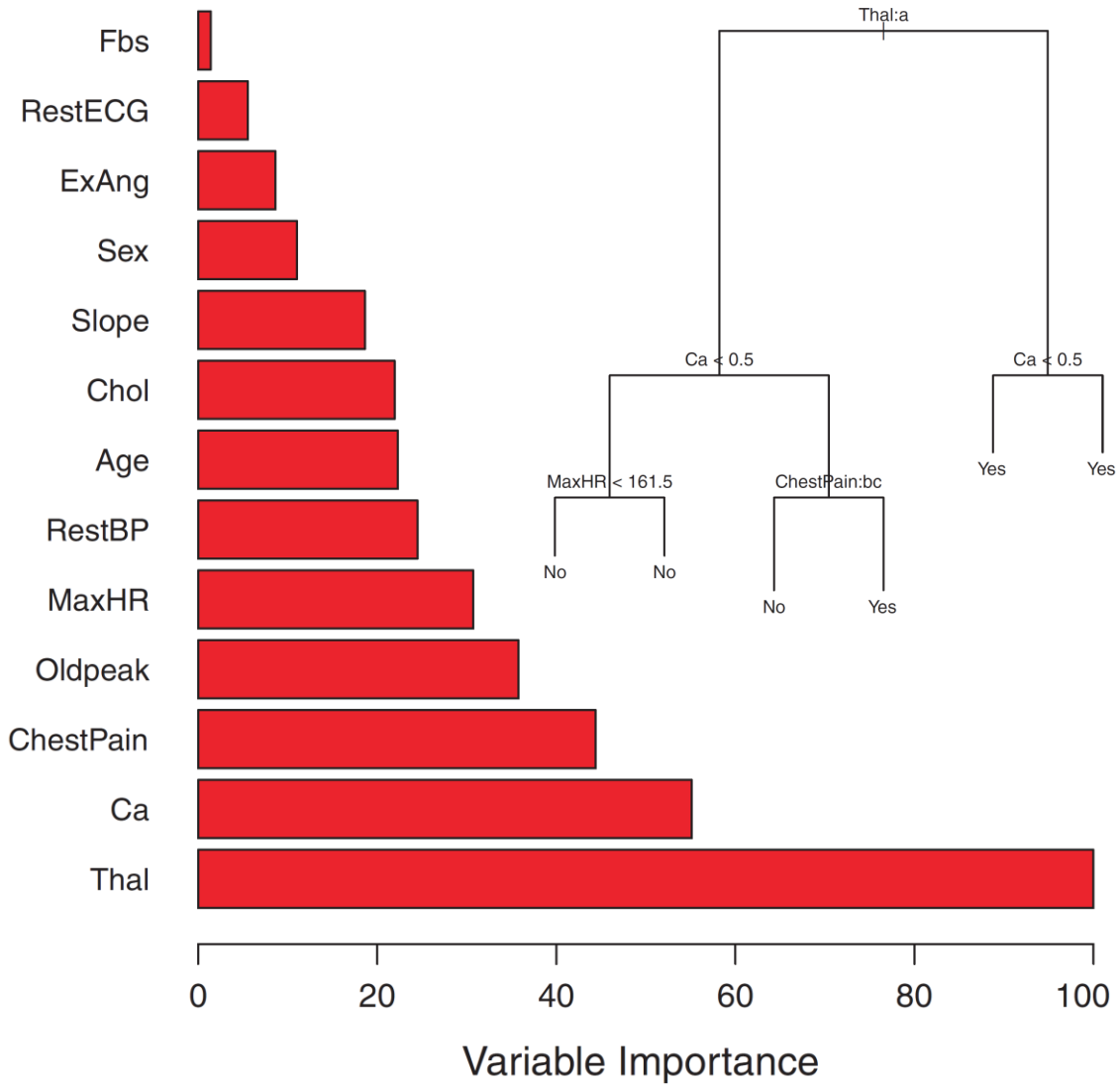
We can then average this value across all B bagged trees, with larger values indicating more important predictors.

Similarly, for a bagged classification tree, we can record the amount that the Gini index (G) is decreased due to splits over that feature for each bagged classification tree.

We can then average this value across all B bagged trees.

Feature importance example from heard disease data

The features with the largest mean decrease in Gini index for building a classification tree to predict heart disease are the qualitative features Thal and ChestPain and the quantitative feature Ca.



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

De-correlating bootstrap replicates improves variance

While discussing decision trees, we found that bagging, which aggregates bootstrap replicates can improve tree-based estimates by maintaining the low bias of decision trees while also reducing their variance.

However, there exists further methods to reduce the variance of tree-based methods.

Recall that with bagging, as the number B of bootstrap replicates increase, the variance of the bagged estimate approaches

$$\mathrm{Var}\left[\hat{f}_{\mathrm{bag}}(x)\right] \approx \rho\sigma^2$$

Variance can be reduced by de-correlating the bootstrap replicates.

Random forests

Random forests, like bagging, take means of tree predictions estimated from bootstrap replicates, while de-correlating these trees further.

Denote parameter m as the number of p features to use during splits when growing a decision tree.

When building a decision tree from a bootstrap replicate, each time a split is considered, a set of m features are selected uniformly at random from the p features as split candidates.

When $m = p$, random forests reduce to bagging, but when m is small, the trees built from different replicates are based on only partially overlapping feature sets.

Typical values of m and reason for small m

Typical values of the number of features to consider at each split are $m \approx \sqrt{p}$, which is much smaller than the full set of features.

That is, at each split in the tree, the algorithm considers only a minority of the available features.

Aside from de-correlating trees due to the set of features included in random forests, there is another motivation for permitting a small number of features m at each split.

If there is a strong (influential) feature in the dataset among a set of moderately strong features, then most trees under bagging will use this strong feature in the top (root) split.

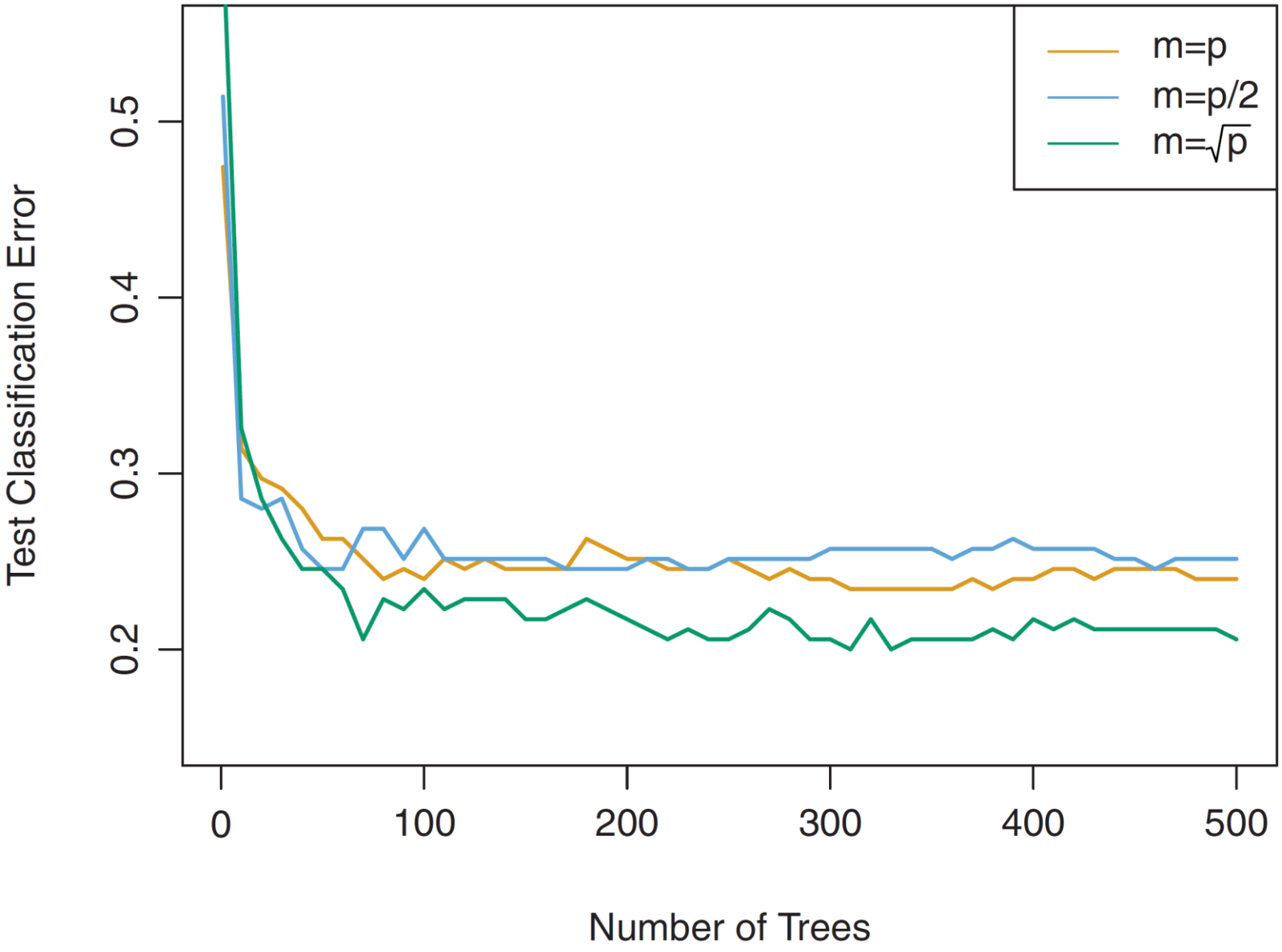
Typical values of m and reason for small m

Because of this, the bagged trees will all have a similar structure, and will therefore be highly correlated.

With random forests, there is a chance that this strong feature will not even be considered in a split.

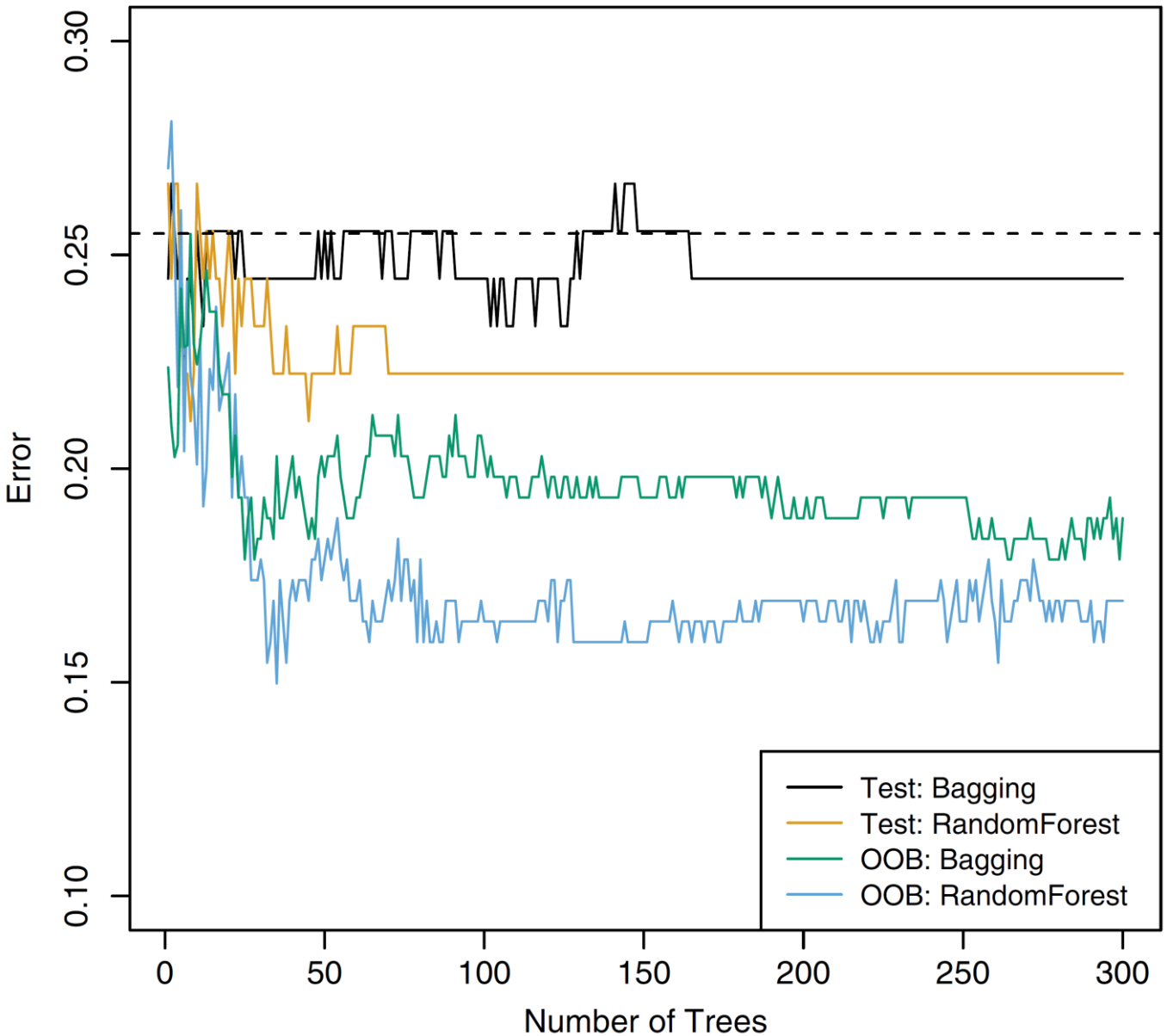
By not permitting this strong feature to not be considered in a split, it will give other, more moderately strong, features to be considered at the root, thereby leading to different overall trees structures, and further decoupling the bootstrap trees.

Impact of m on test classification error



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Random forests ($m = \sqrt{p}$) outperforms bagging

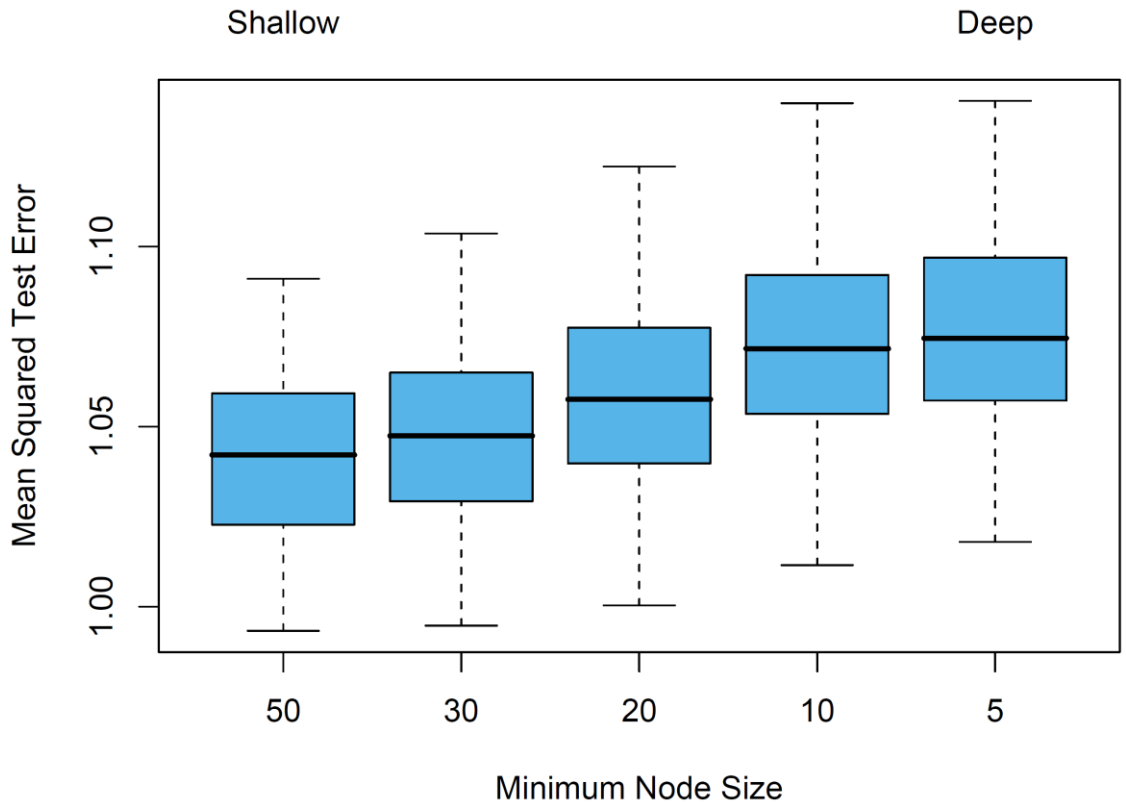


James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Effect of tree size on random forest test error

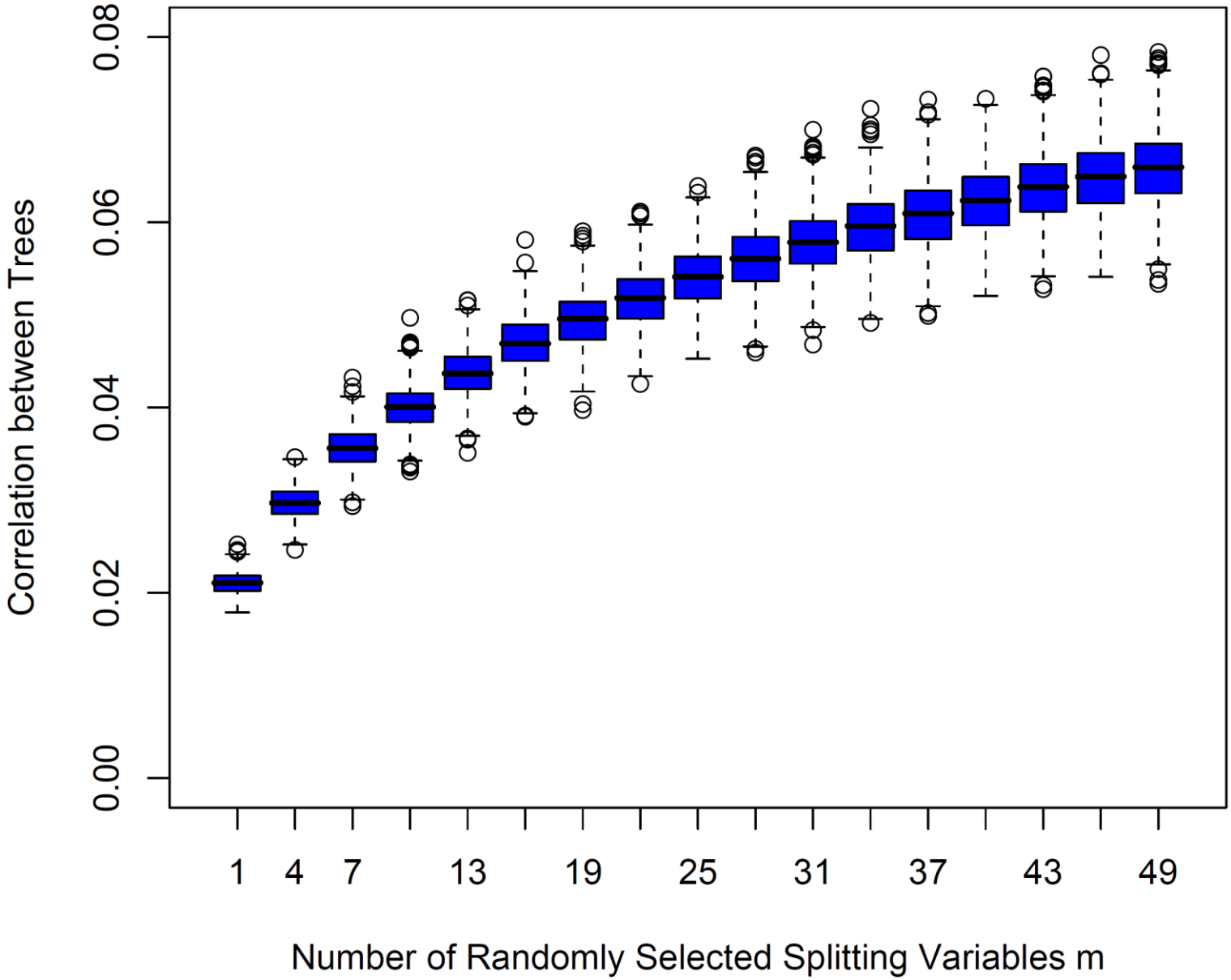
There were 12 additive features related to the response with additive unit variance Gaussian noise added to the features.

Tree depth is controlled by minimum node size (number of observations), with smaller sizes yielding deeper trees.



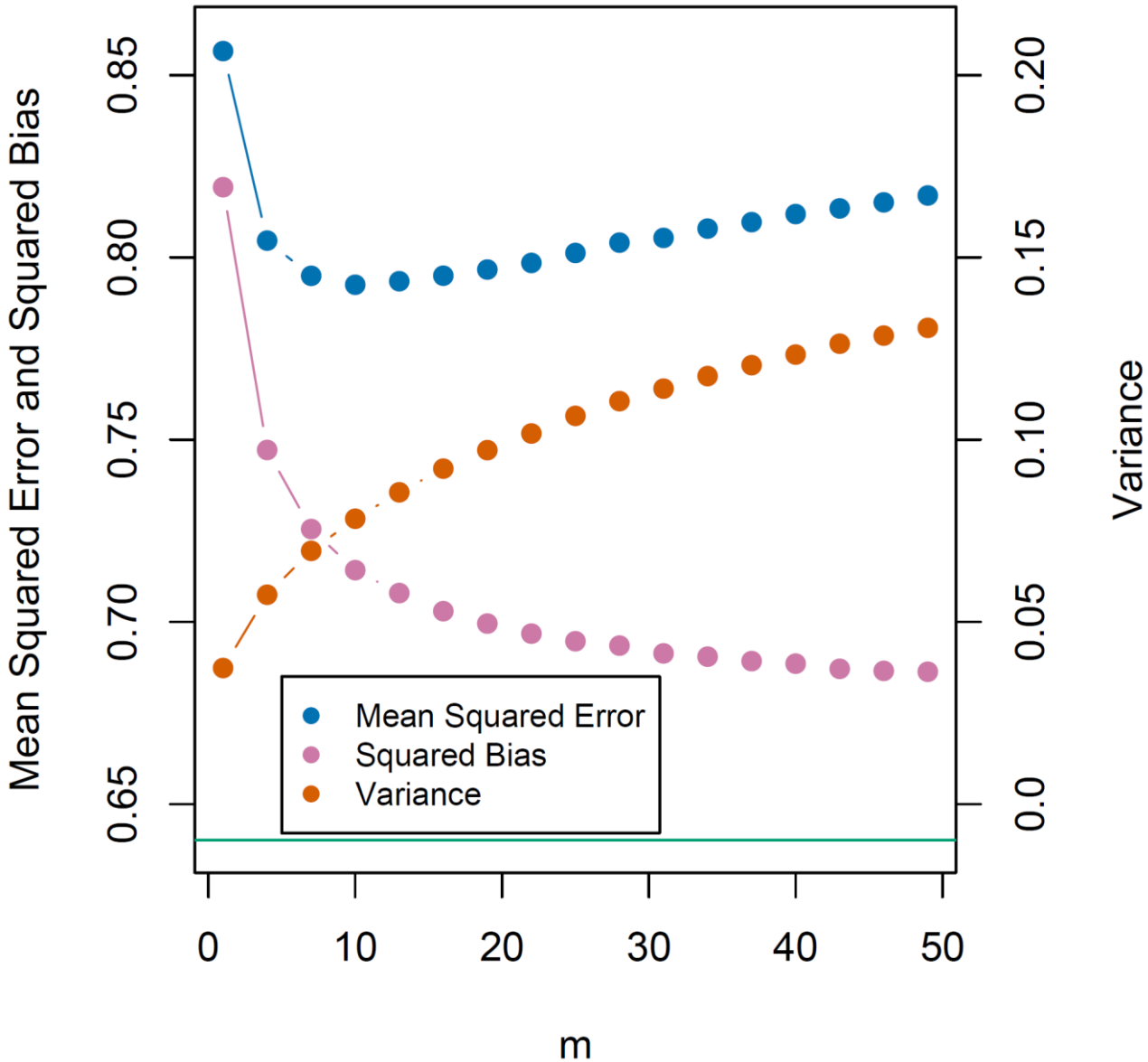
Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Pairwise tree correlation as function of m



Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

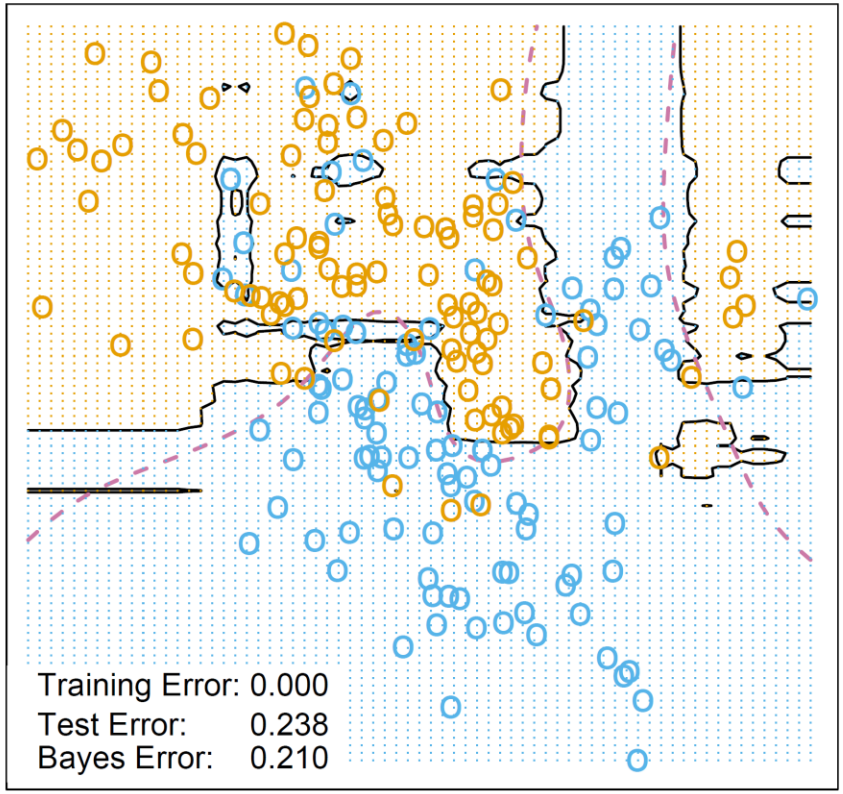
Parameter m makes typical bias-variance tradeoff



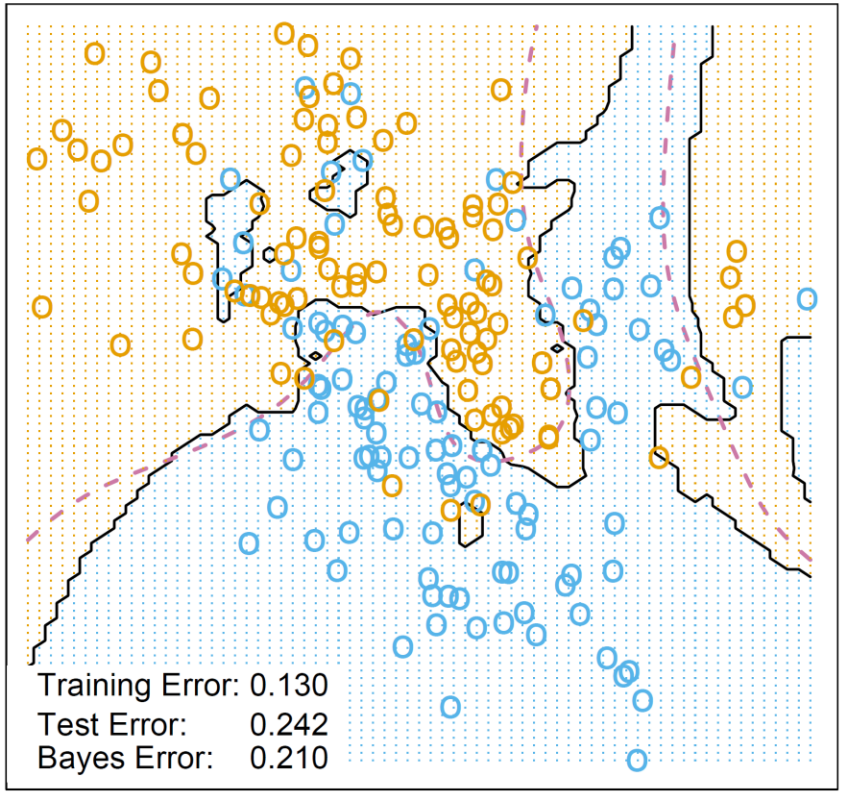
Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Random forests can model complex surfaces

Random Forest Classifier



3−Nearest Neighbors



Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Boosting

Boosting is another general approach for improving predictions for many statistical learning methods for regression or classification, and here we particularly highlight its application to decision trees.

Recall that bagging involves creating multiple replicates of the original training dataset using bootstrap, and fitting separate decision trees to each replicate, and then combining all the estimated trees to create a single predictive model.

Specifically, each tree is grown on a bootstrap dataset, independent of the other bootstrap replicates.

Boosting works in a similar fashion, except that the trees are grown sequentially, with subsequent trees grown using information from previously-grown trees.

Algorithm for boosting a regression tree

Step 1: Initialize the boosted prediction model as $\hat{f}(x) = 0$ and set the residual $r_i = y_i$ for $i = 1,2,\dots,N$.

Step 2: For $b = 1,2,\dots,B$, repeat:

2a: Fit a tree $\hat{f}_b(x)$ with d splits ($d + 1$ terminal nodes) to the training data (x_i, r_i), $i = 1,2,\dots,N$.

2b: Update $\hat{f}(x)$ by adding a shrunken version of new tree

$$\hat{f}(x) \coloneqq \hat{f}(x) + \lambda \hat{f}_b(x)$$

2c: Update the residuals as

$$r_i \coloneqq r_i - \lambda \hat{f}_b(x_i)$$

Step 3: Output the boosted model

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

Overview of boosting for regression trees

Similar to bagging, boosting will combine predictions from a set of B decision trees, with $\hat{f}_b(x)$, $b = 1,2,\ldots,B$, the prediction of tree b.

Unlike with a single decision tree, which fits the data in one go, boosting learns a model slowly.

Given the current model, we fit a decision tree to the residuals of the model.

That is, we fit a tree using the current residuals as the response rather than the response Y.

We then add this new decision tree into the fitted function, which updates the current residuals.

Overview of boosting for regression trees

Each of the trees learned at each step can be fairly small, with only a few terminal nodes, which we control with a parameter d.

That is, the parameter d controls how many regions a regression tree is able to model at any one time.

By setting d to be small, we are only fitting small trees, and hence slowly improve on the overall estimated function $\hat{f}(x)$ in areas in which it does not perform well due to fitting the current residuals.

Tuning parameters

Boosting therefore has 3 tuning parameters:

1. The number of trees B to examine
2. The number of splits d permitted in each tree.
3. The shrinkage parameter λ.

Tuning parameters

Boosting therefore has 3 tuning parameters:

1. The number of trees B to examine
2. The number of splits d permitted in each tree.
3. The shrinkage parameter λ.

The B parameter:

Unlike bagging and random forests, boosting can overfit the training data if B is too large, though overfitting tends to occur slowly.

Can select B through cross validation, which can be computationally expensive.

Alternatively, can use early stopping (similar to in neural networks).

Tuning parameters

Boosting therefore has 3 tuning parameters:

1. The number of trees B to examine
2. The number of splits d permitted in each tree.
3. The shrinkage parameter λ.

The d parameter:

Controls the complexity of the boosted ensemble.

The use of stumps ($d = 1$) often works well.

With stumps, the boosted ensemble is fitting an additive model because each term involves only a single variable.

Tuning parameters

Boosting therefore has 3 tuning parameters:

1. The number of trees B to examine
2. The number of splits d permitted in each tree.
3. The shrinkage parameter λ.

The λ parameter:

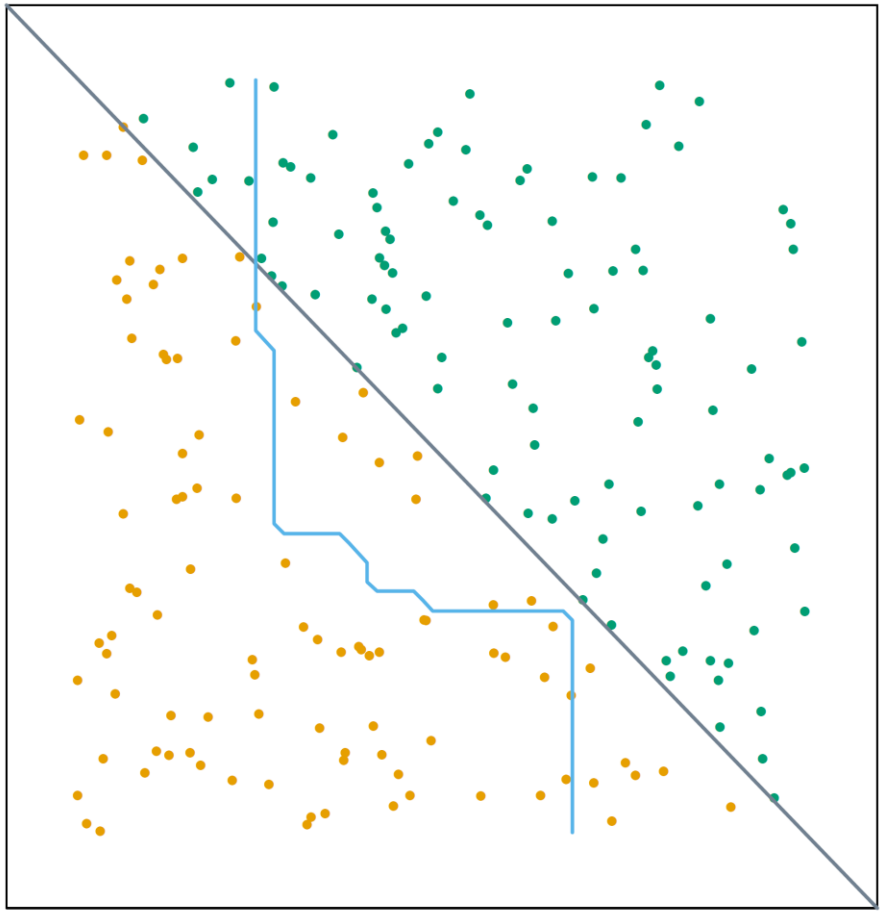
A small positive number ($\lambda > 0$) that controls the rate at which boosting learns.

Typical values are 0.01 or 0.001.

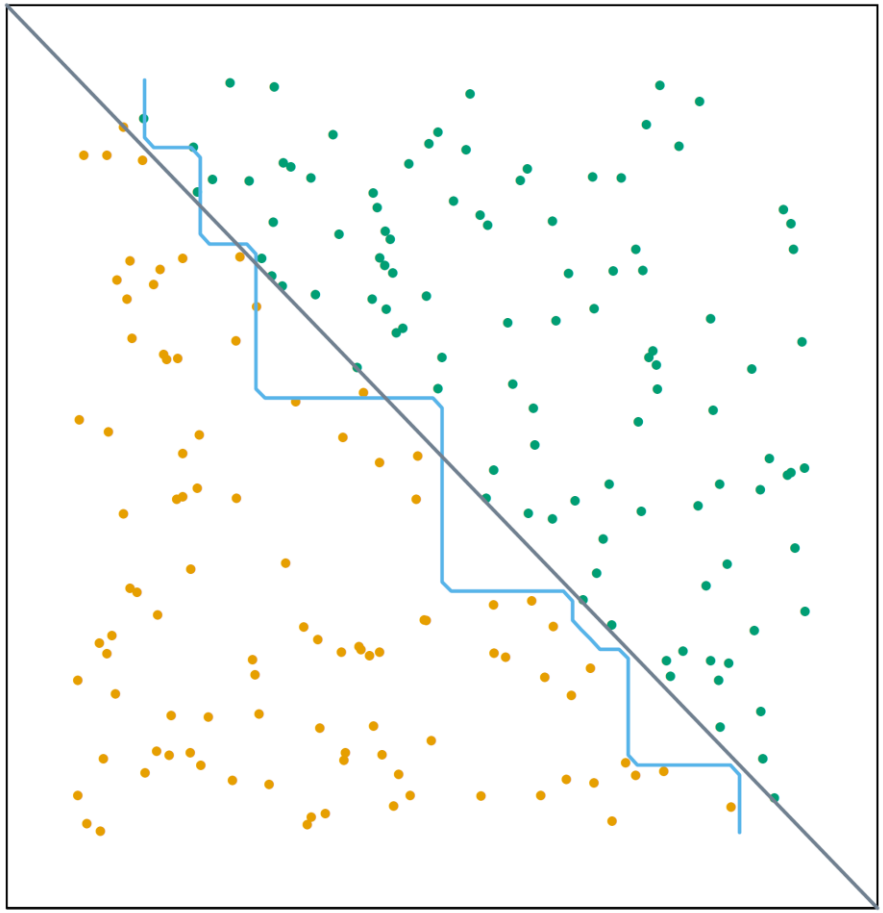
Very small λ can require large values of B to achieve good prediction accuracy.

Boosting better models linear decision boundary

Bagged Decision Rule

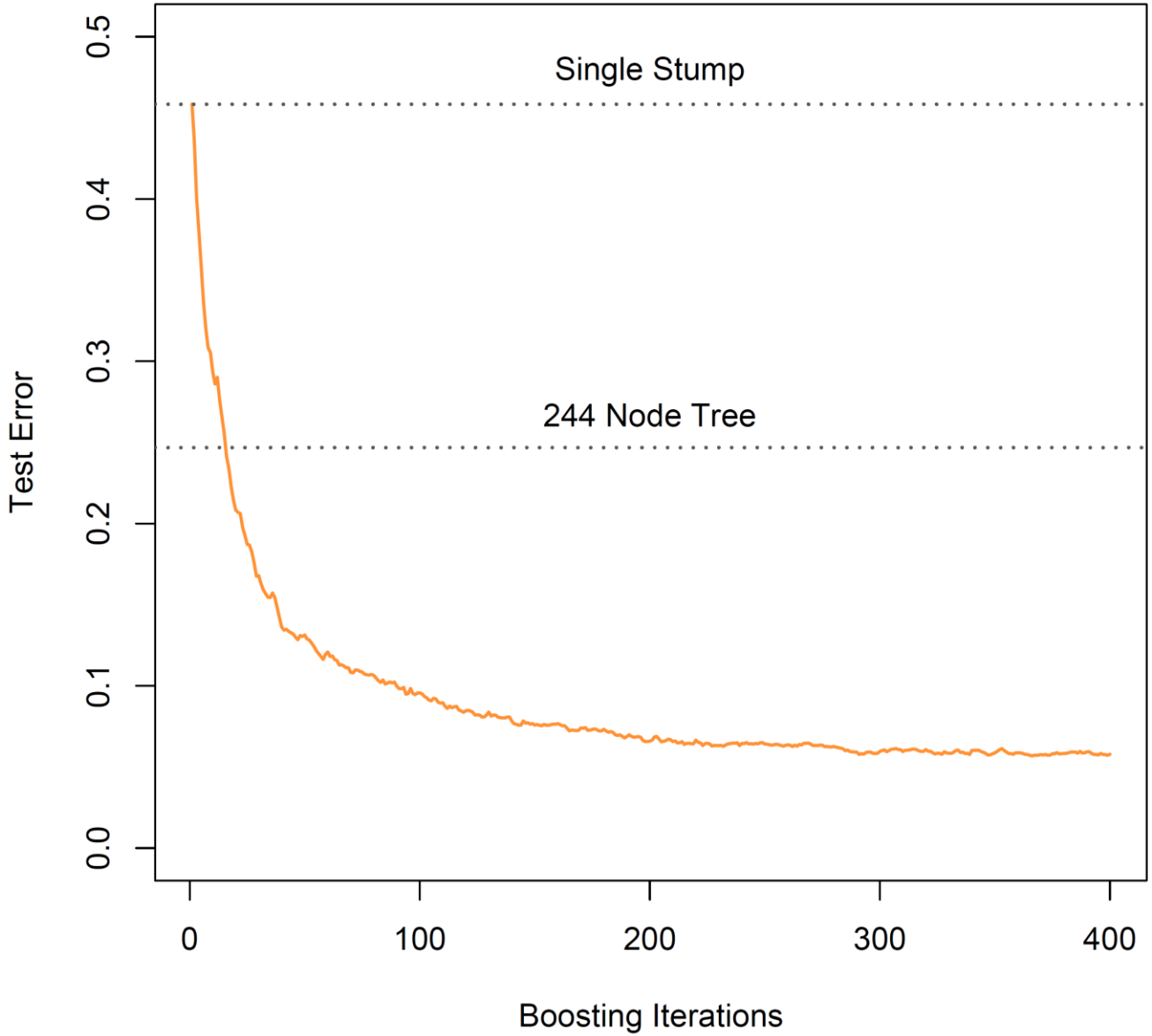


Boosted Decision Rule



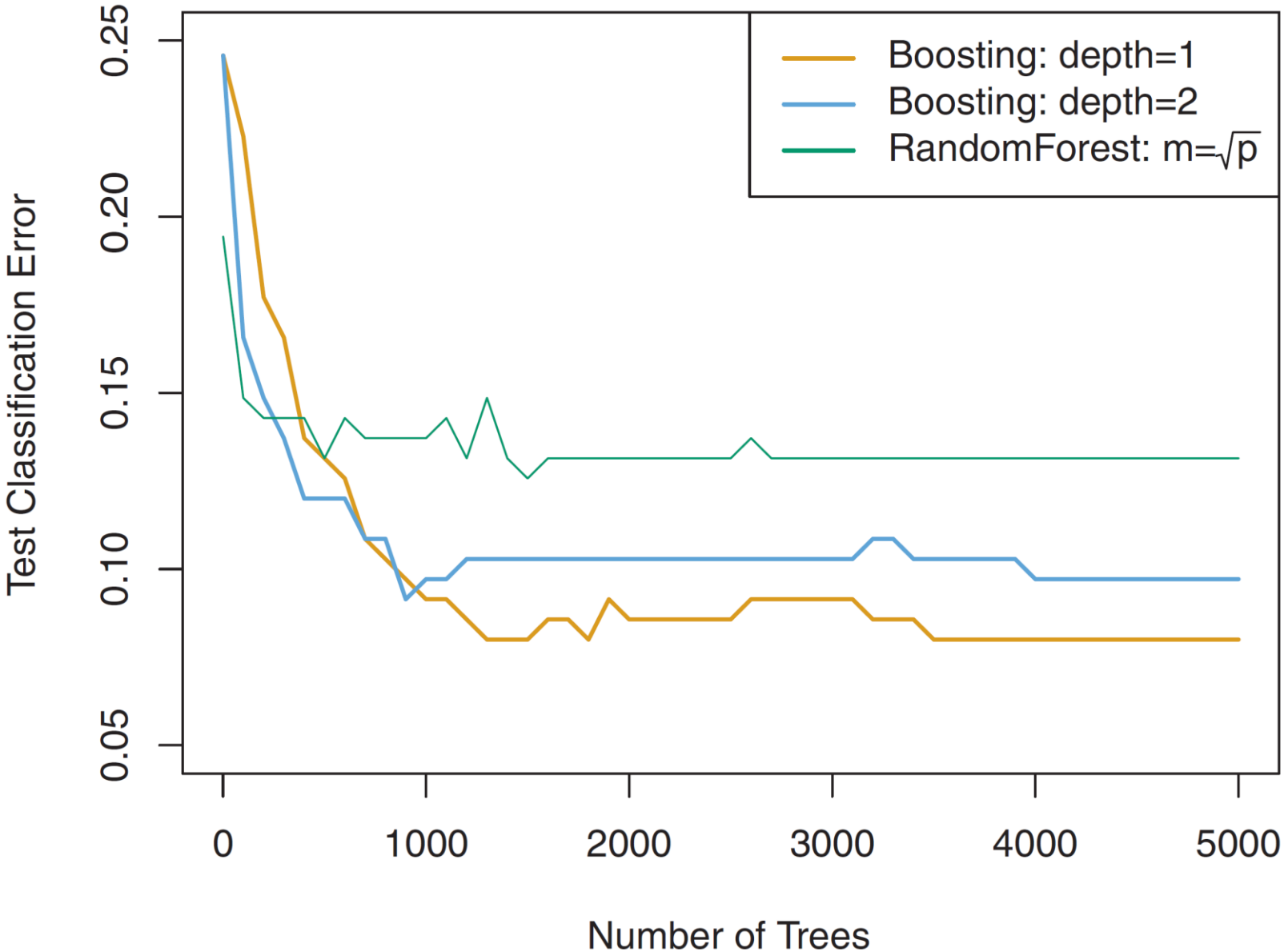
Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Boosting with stumps outperforms single trees



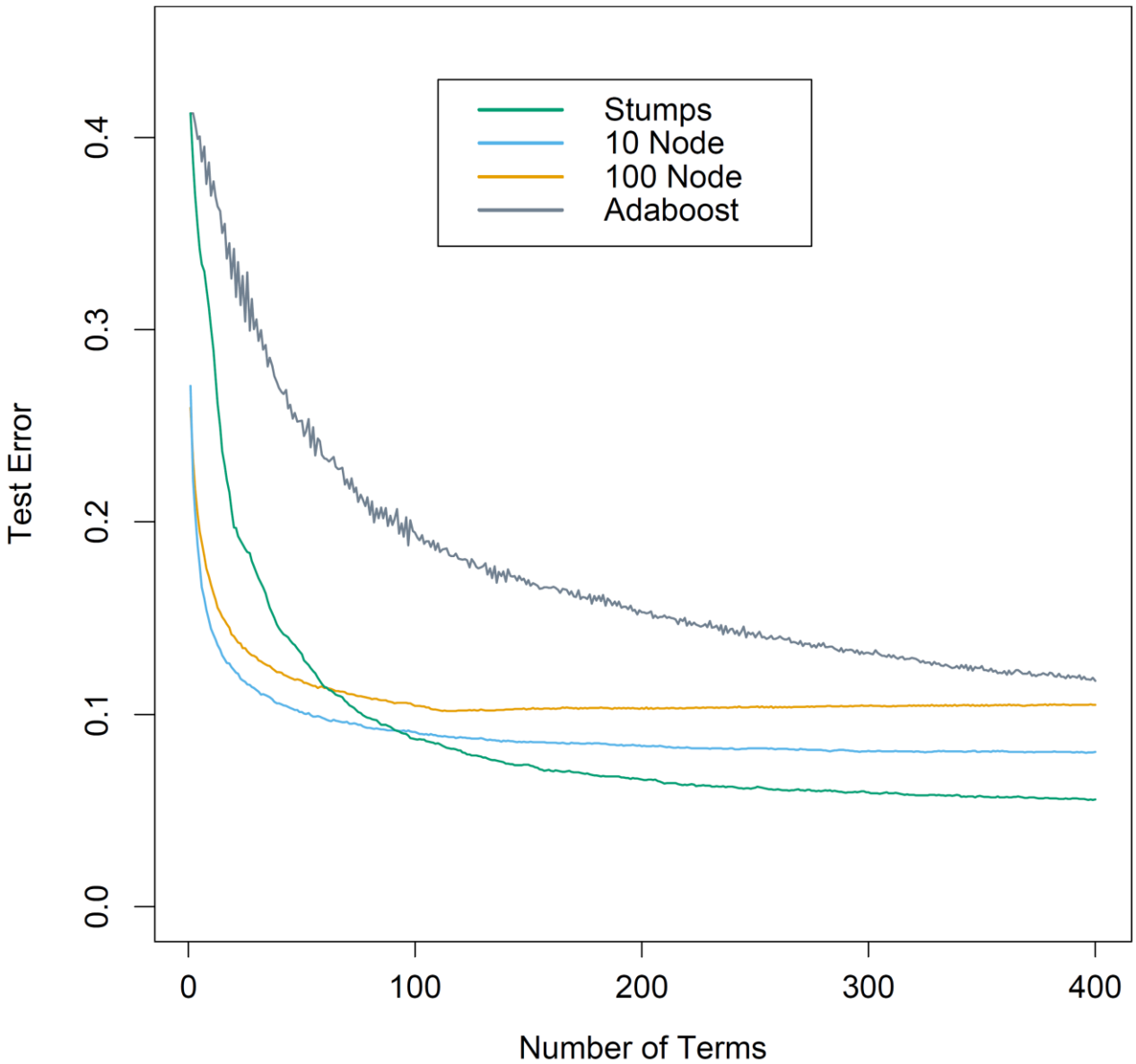
Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Boosting with stumps outperforms random forests



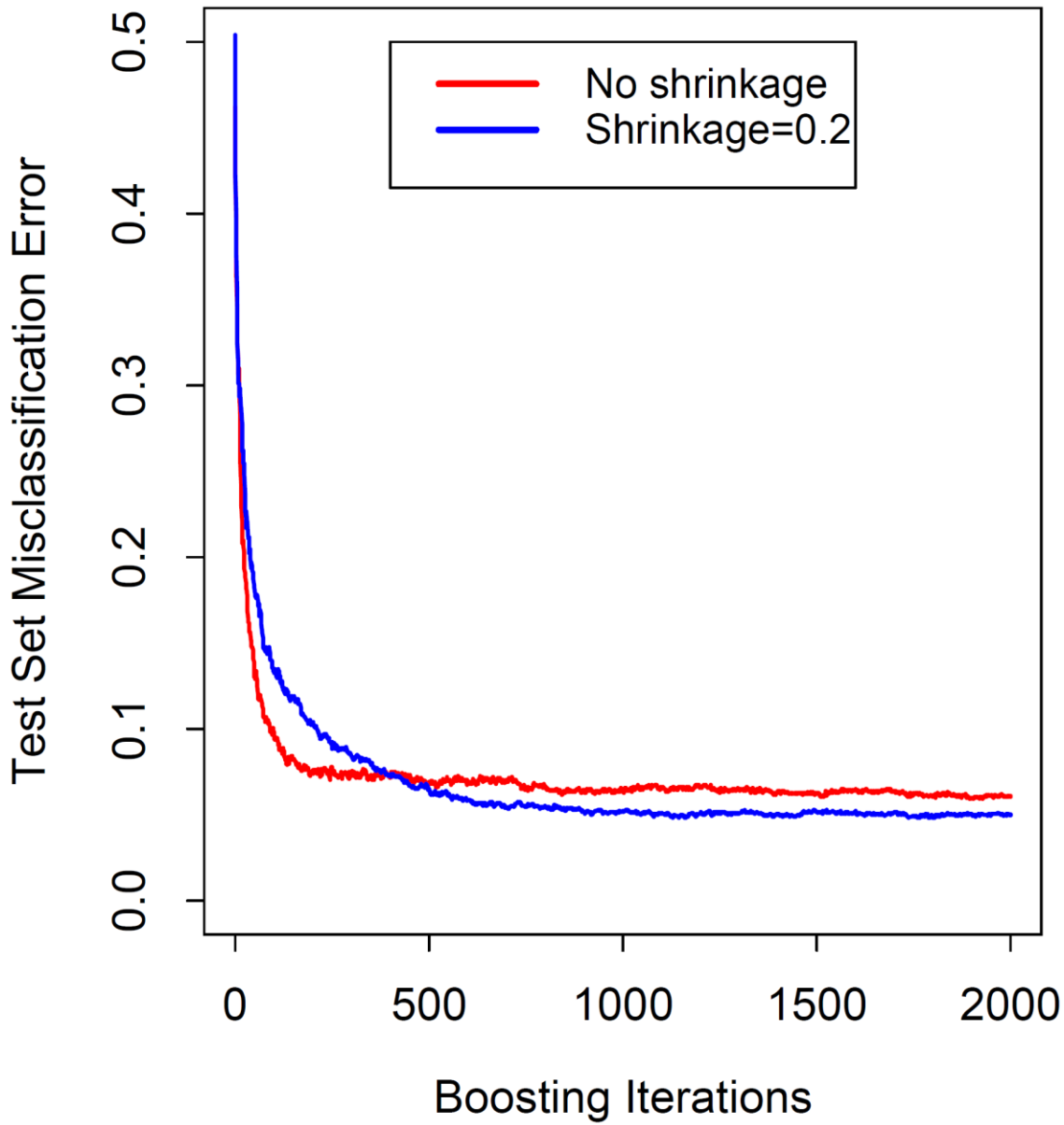
James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Using $d = 1$ (stumps) outperforms moderate and large d



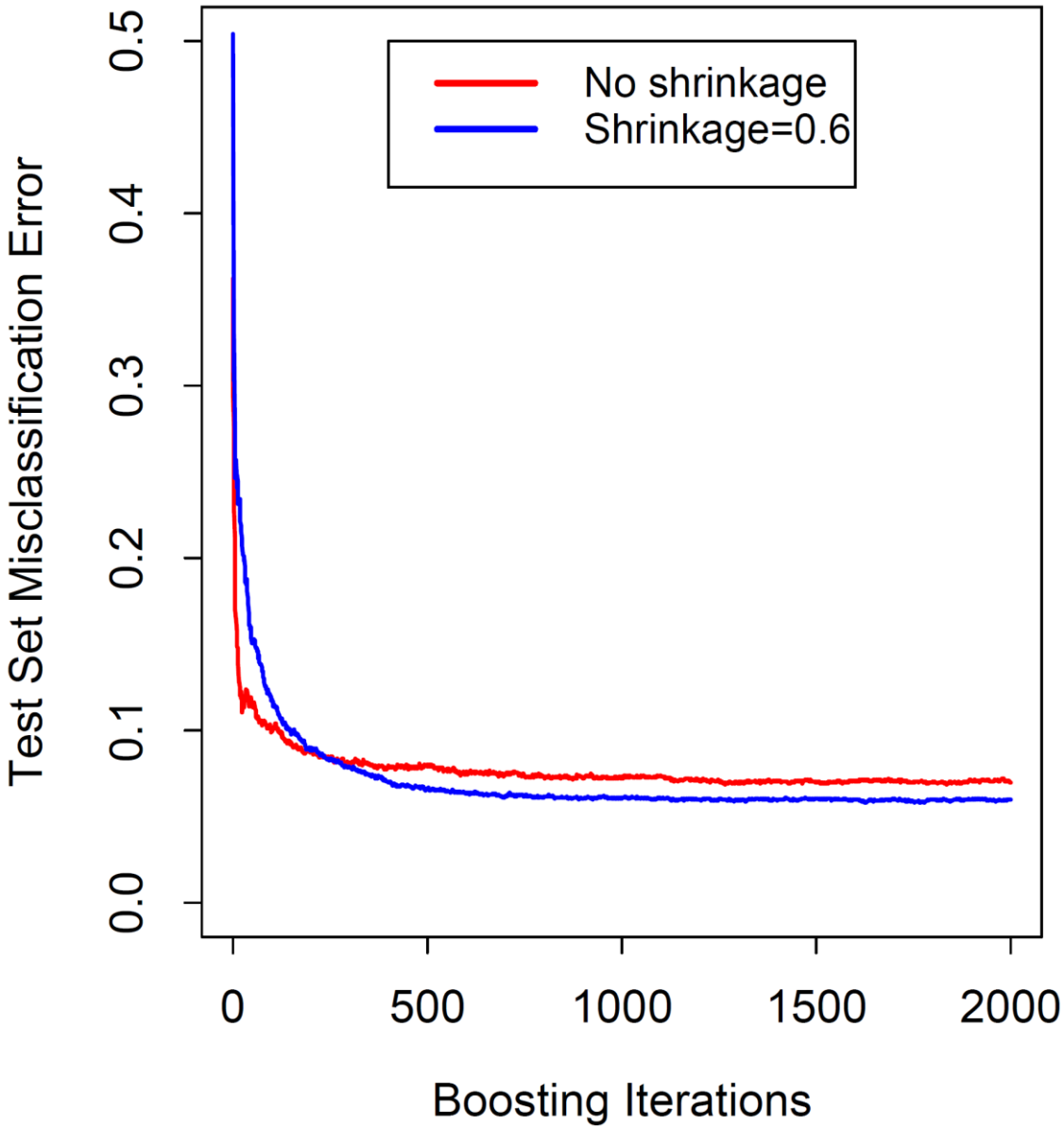
Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Effect of shrinkage parameter λ with stumps ($d = 1$)



Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Effect of shrinkage parameter λ with $d = 5$



Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

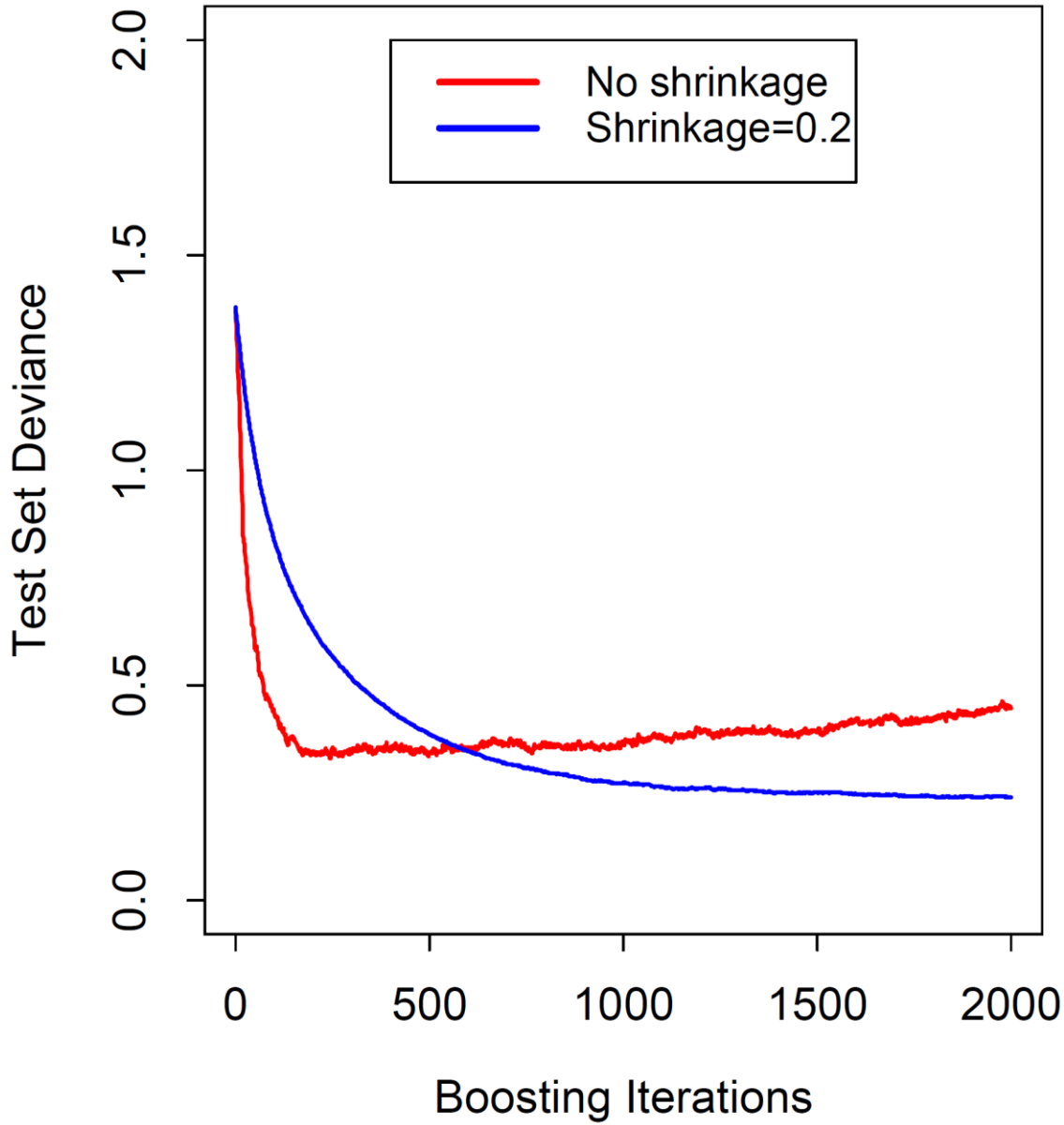
Using a more sensitive measure of test error

The previous two plots highlighted how shrinkage can help in obtaining lower test misclassification error.

However, misclassification error is not a sensitive measure, as it does not take the probability of the class into account.

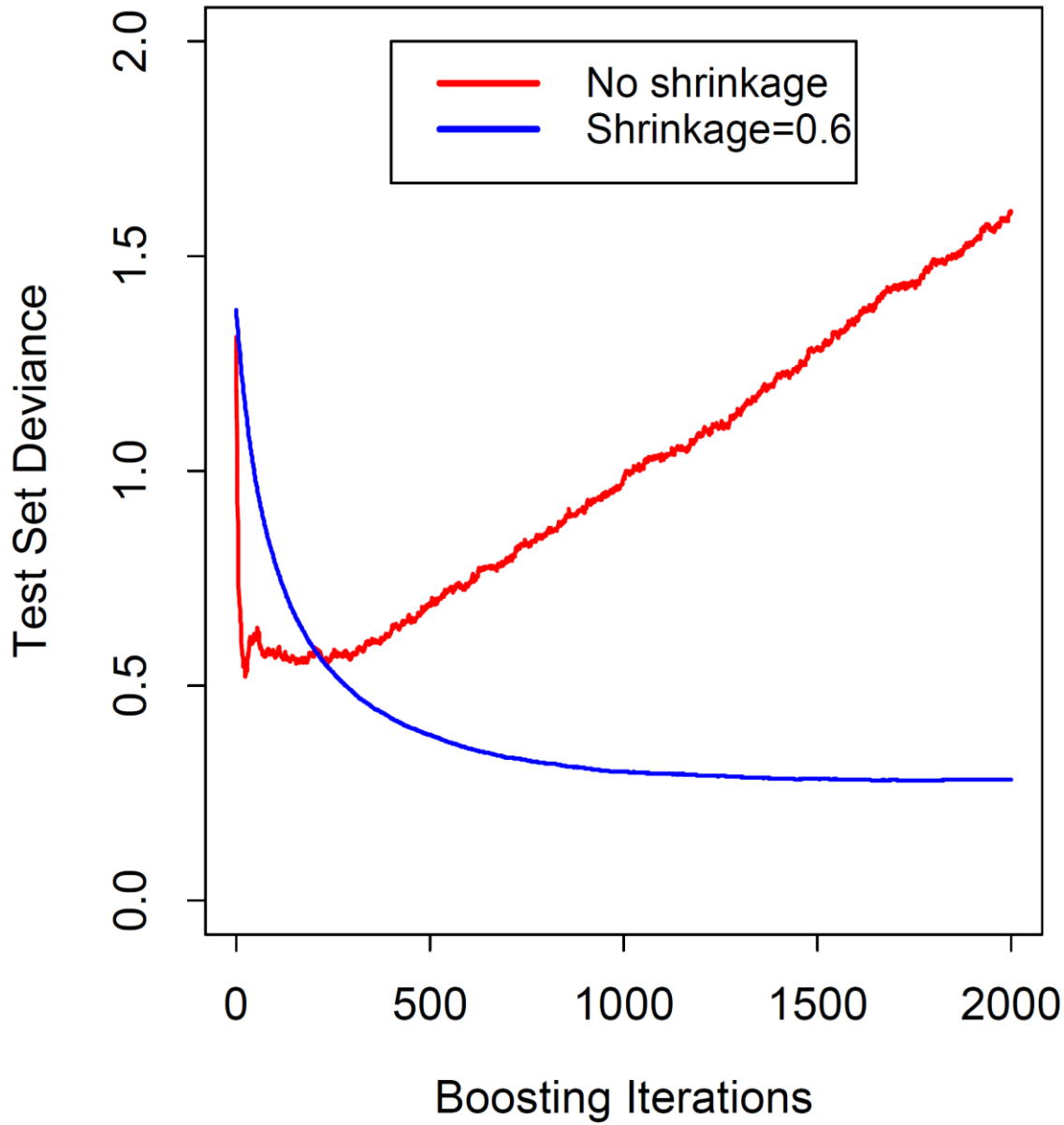
Instead, we consider a more sensitive measure of test error, termed **deviance**, which is the same as the cross-entropy**.**

Effect of λ with stumps ($d = 1$) on deviance



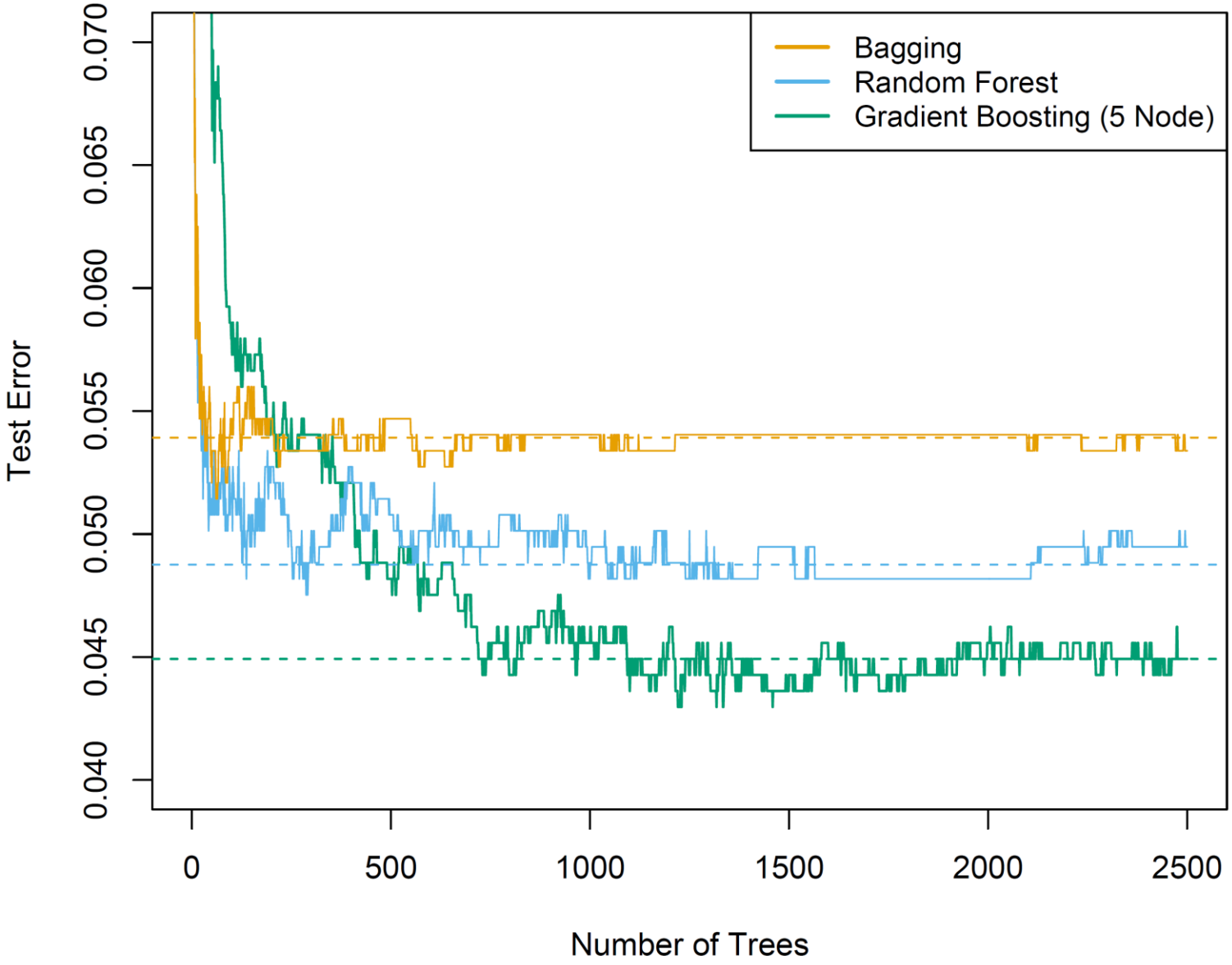
Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Effect of λ with $d = 5$ on deviance



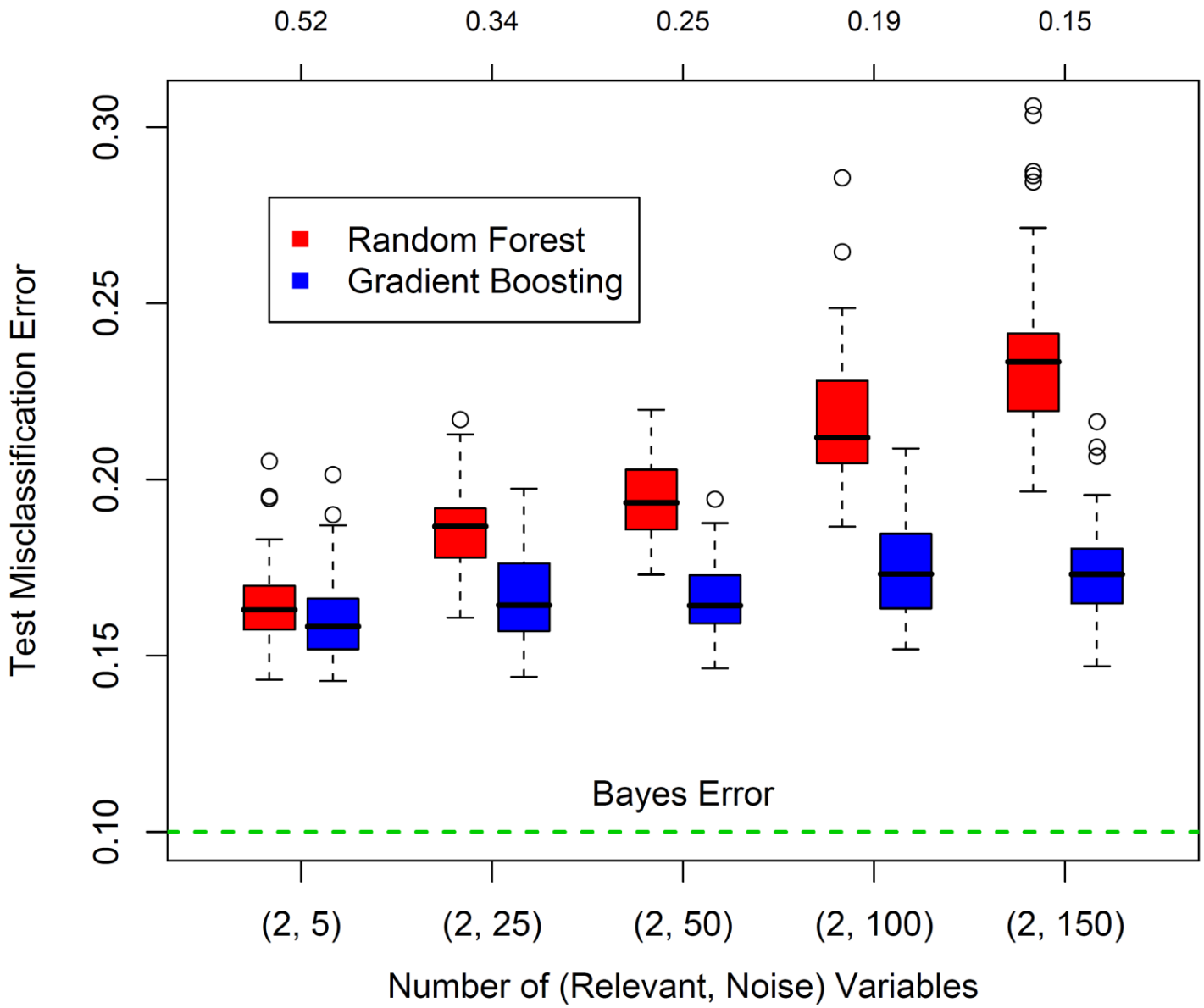
Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Bagging vs. random forests vs. boosting



Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Boosting robust to noise variables, RF ($m = \sqrt{p}$) is not



Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.