Random Forest

Natalie Nelson, PhD | BAE Environmental and Agricultural Analytics and Modeling

Learning Objectives

- **Explain** and **apply** the Random Forest model algorithm
- **Describe** what “bagging” is and how “out of bag error” is calculated
- **Explain** the bias-variance tradeoff
- **Calculate** the predictive skill of a classification model using the Kappa coefficient
- **Explain** an approach for estimating feature importance in a RF model
- **Apply** the RF algorithm in a k-fold cross-validation workflow

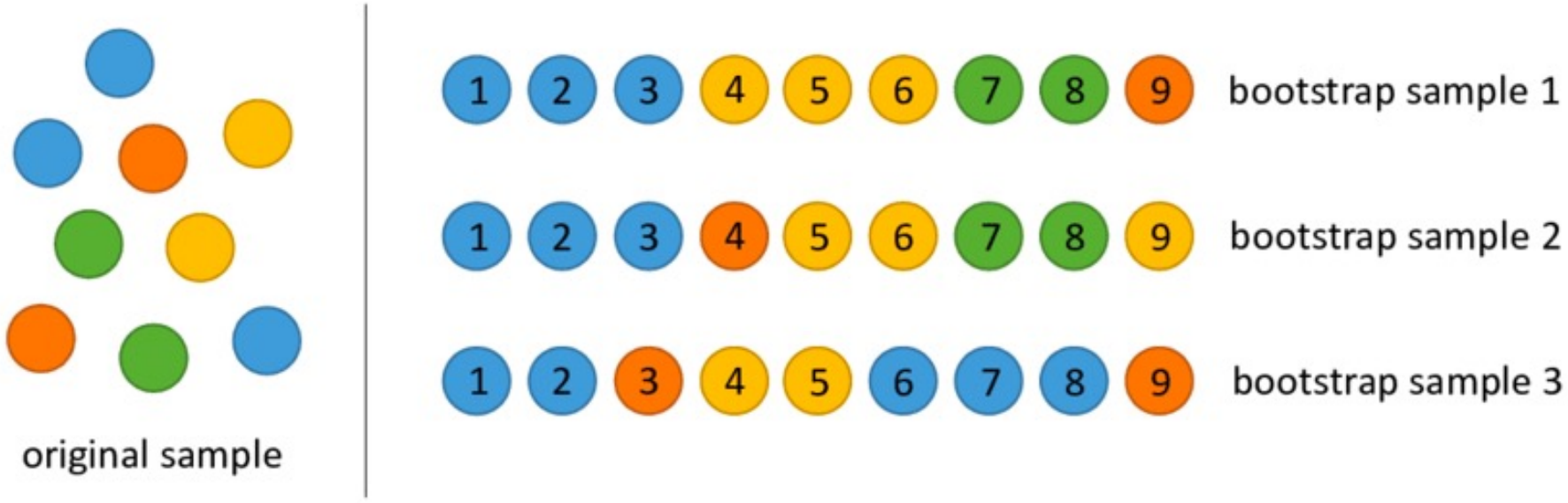
Random Forest

A bootstrap aggregation (“bagging”) and split-variable randomization approach to developing an ensemble of classification or regression trees

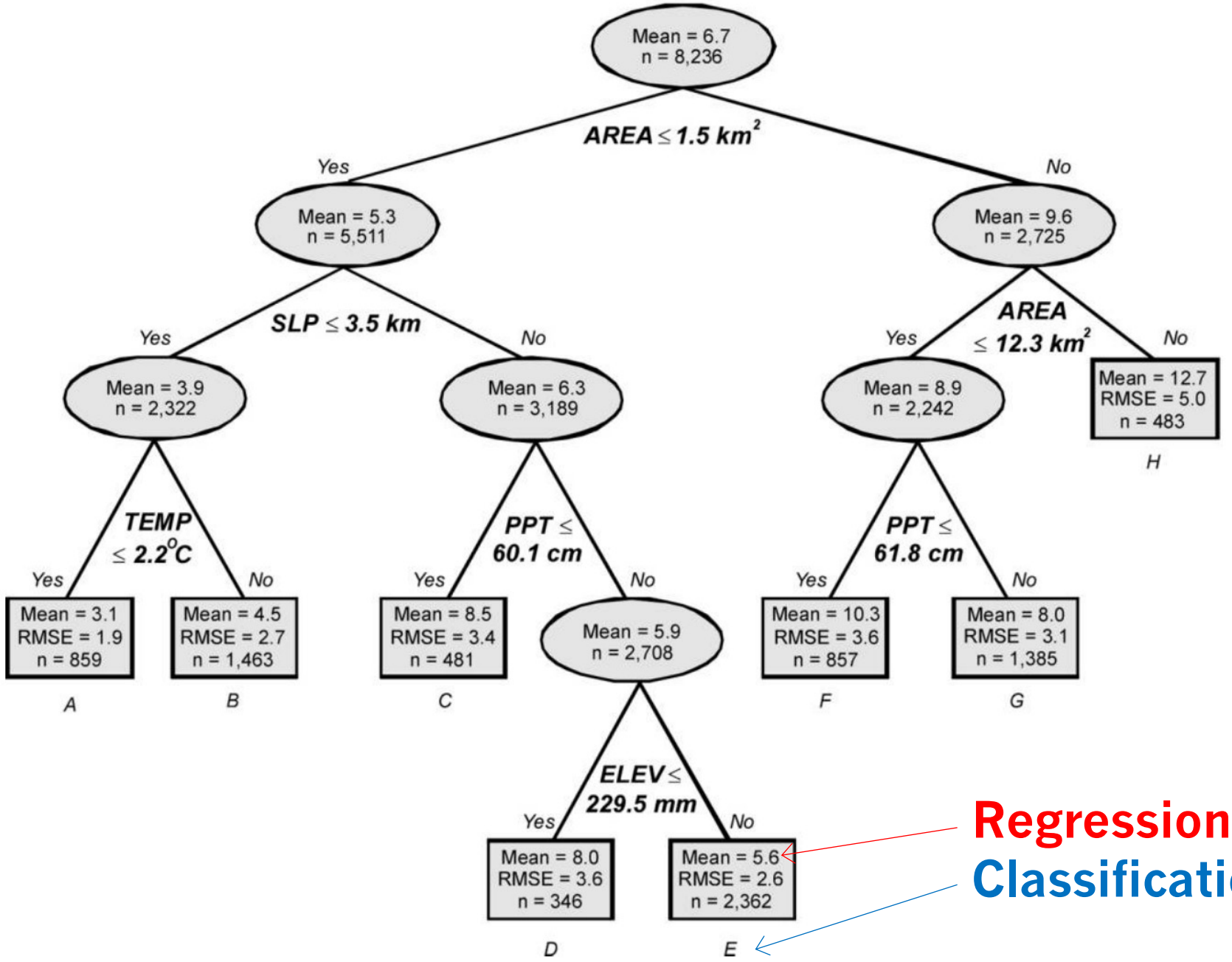
Breiman (2001), Random forests, Machine Learning 45(1): 5-32

Bootstrap sampling or “bootstrapping”

Random sampling with replacement, where many samples are taken from a large dataset.

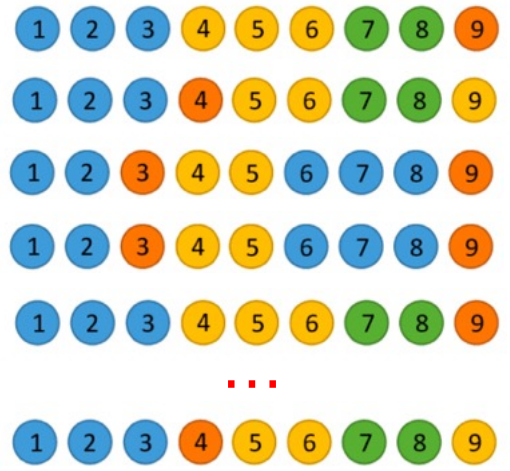


Breiman (2001), Random forests, Machine Learning 45(1): 5-32; image credit: Galdi and Tagliaferri (Dec 2017)

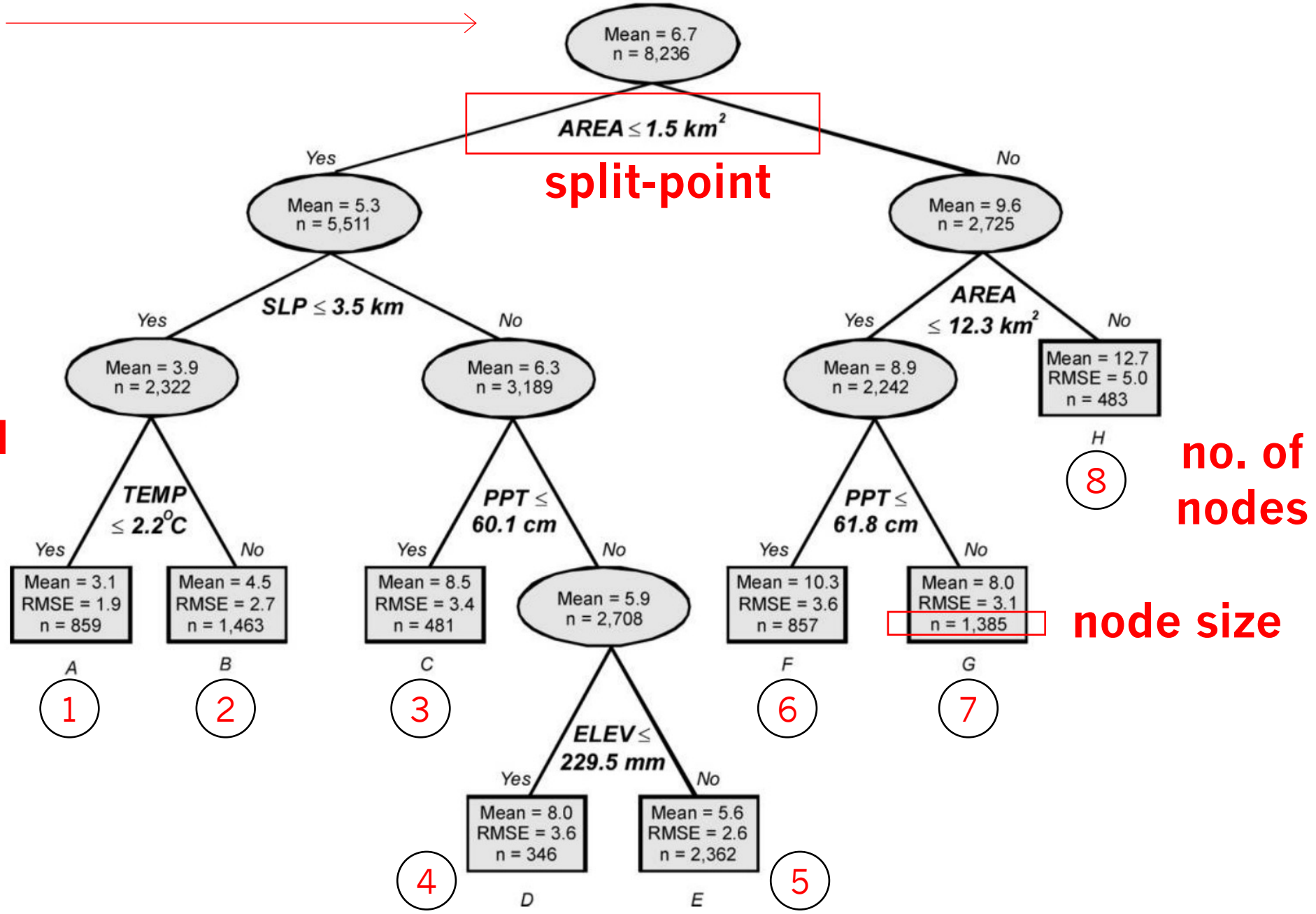


Regression (quantitative) or
Classification (qualitative)

Olden et al. (2008), Machine Learning Methods Without Tears: A Primer for Ecologists



of **trees** =
of **bootstrapped samples**



Olden et al. (2008), Machine Learning Methods Without Tears: A Primer for Ecologists

Algorithm 15.1 *Random Forest for Regression or Classification.*

Grow trees

1. For $b = 1$ to B: B = number of trees

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

Predict from forest

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Hastie, Tibshirani, and Friedman (2009), The Elements of Statistical Learning: Data Mining, Inference, and Prediction

1a. Draw a bootstrap sample of size N from the training data

N typically ranges from 60-100%; 100% → overfitting more likely. Default in R is 63.25%

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Bootstrap sample: iris[c(1, 30, 17, 4, ..., 121, 30),]

1b. Grow a random forest tree to the bootstrapped data by recursively repeating the following steps until the minimum node size is reached:

i. Select **m** variables at random from the **p** variables

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Predictors: p = 4

Response

1b. Grow a random forest tree to the bootstrapped data by recursively repeating the following steps until the minimum node size is reached:

i. Select **m** variables at random from the **p** variables

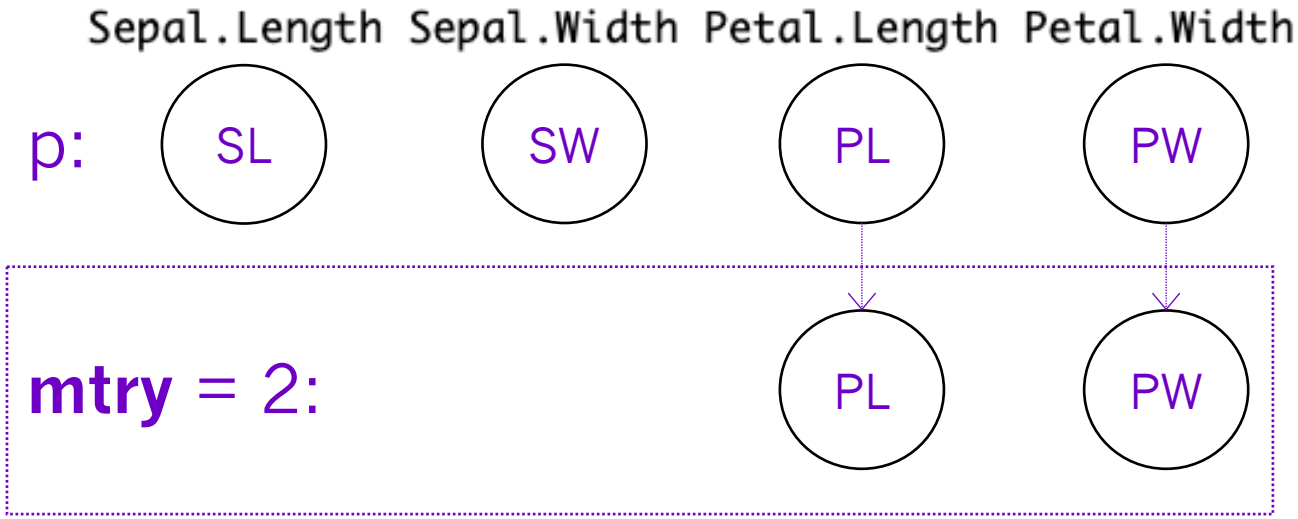
m, or "**mtry**" in R, is the main parameter we tune in RF models. The default mtry value is p/3 for regression models, and sqrt(p) for classification models.
When you use train( ) to train a RF model, a few mtry values will be screened (i.e., tuned).

Iris example:
Classification: mtry = sqrt(p) = sqrt(4) = 2

1b. Grow a random forest tree to the bootstrapped data by recursively repeating the following steps until the minimum node size is reached:

i. Select **m** variables at random from the **p** variables

Classification: mtry = sqrt(p) = sqrt(4) = 2

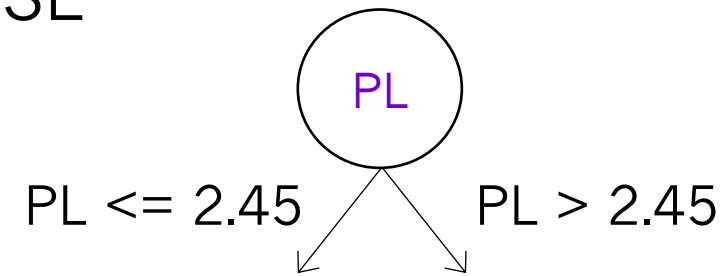


1b. Grow a random forest tree to the bootstrapped data by recursively repeating the following steps until the minimum node size is reached:

ii. Pick the best variable/split-point among the m variables

mtry = 2: PL PW → Bootstrap sample: iris[c(1, 30, 17, 4, ..., 121, 30),]

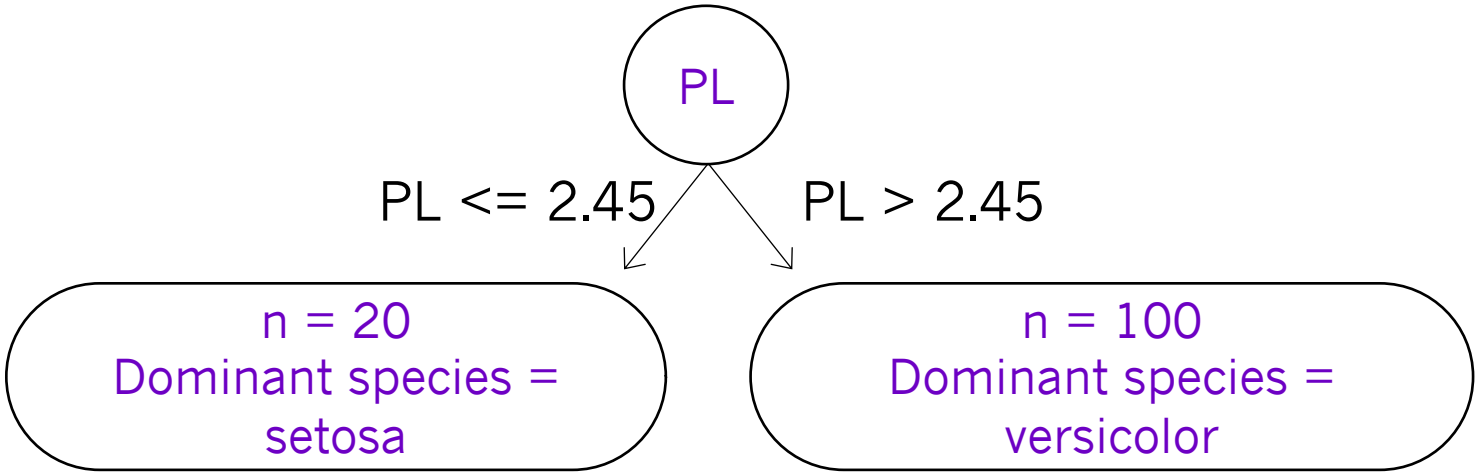
From these two variables, determine how to start the tree → minimize SSE



1b. Grow a random forest tree to the bootstrapped data by recursively repeating the following steps until the minimum node size is reached:

iii. Split the node into two daughter nodes

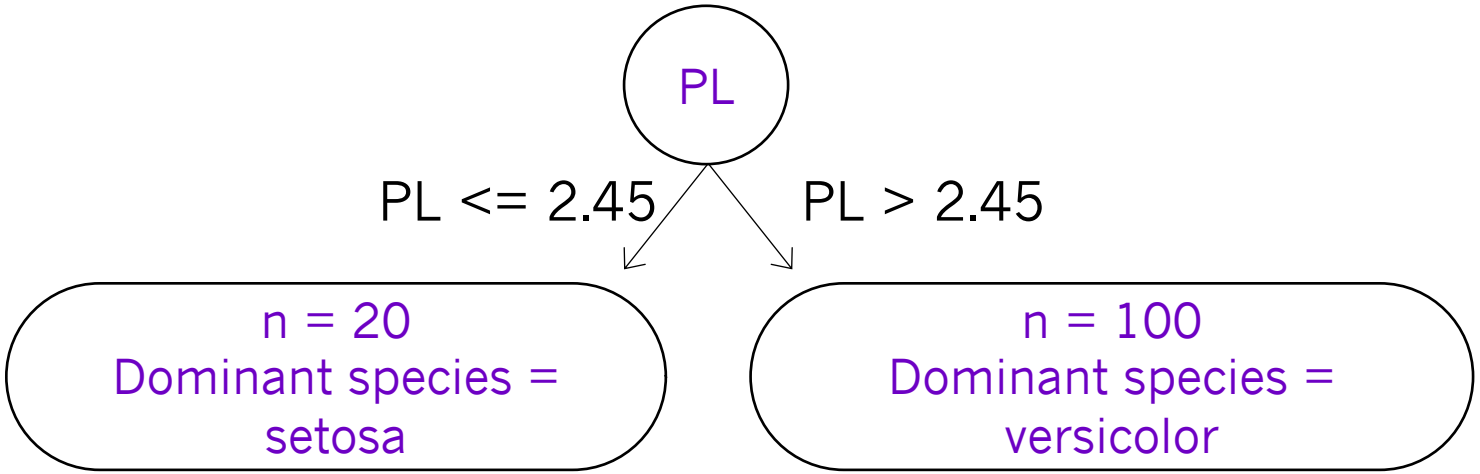
Bootstrap sample: iris[c(1, 30, 17, 4, ..., 121, 30),]



1b. Grow a random forest tree to the bootstrapped data by recursively repeating the following steps until the minimum node size is reached:

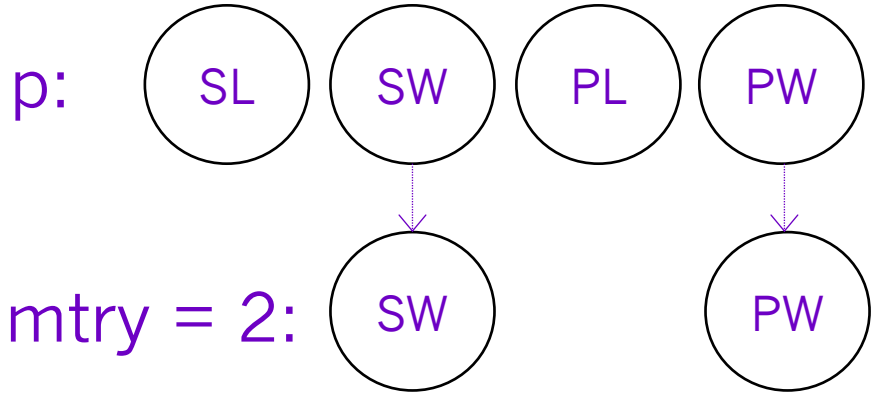
iii. Split the node into two daughter nodes

Bootstrap sample: iris[c(1, 30, 17, 4, ..., 121, 30),]

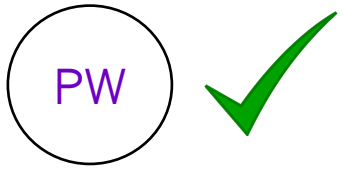


Default minimum node size = 1 for classification, 5 for regression

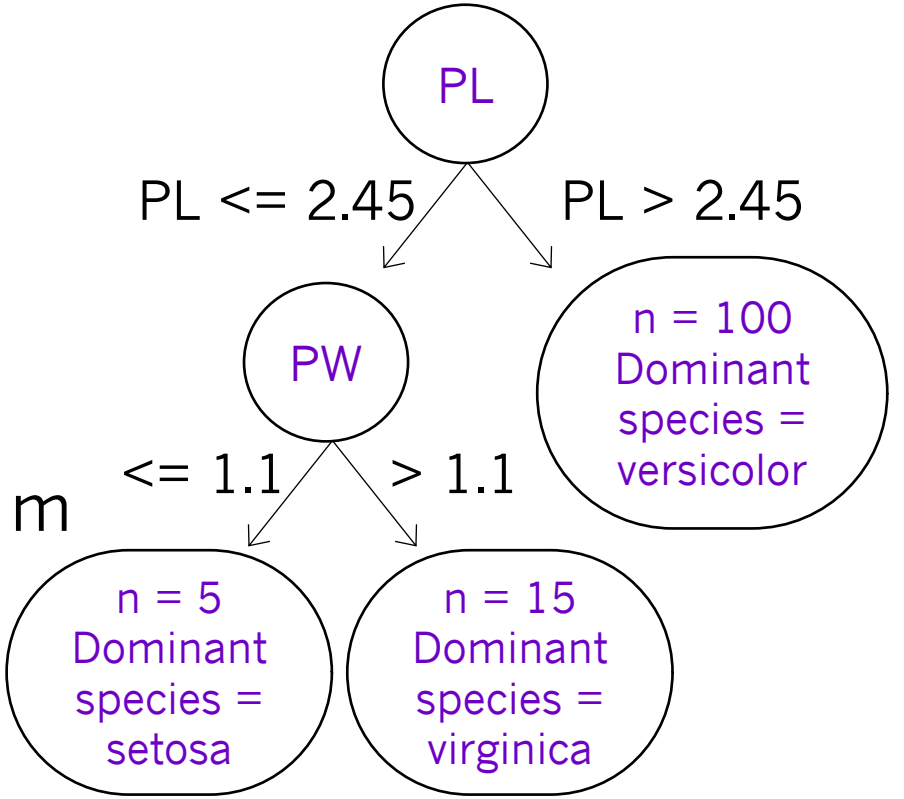
i. Select m variables at random from the p variables



ii. Pick the best variable/split-point among the m



iii. Split the node into two daughter nodes



- Continue with i-iii until the minimum node size is reached for all terminal nodes
- Then pull another bootstrap sample to create the next tree

Algorithm 15.1 *Random Forest for Regression or Classification.*

Grow trees

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

Predict from forest

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

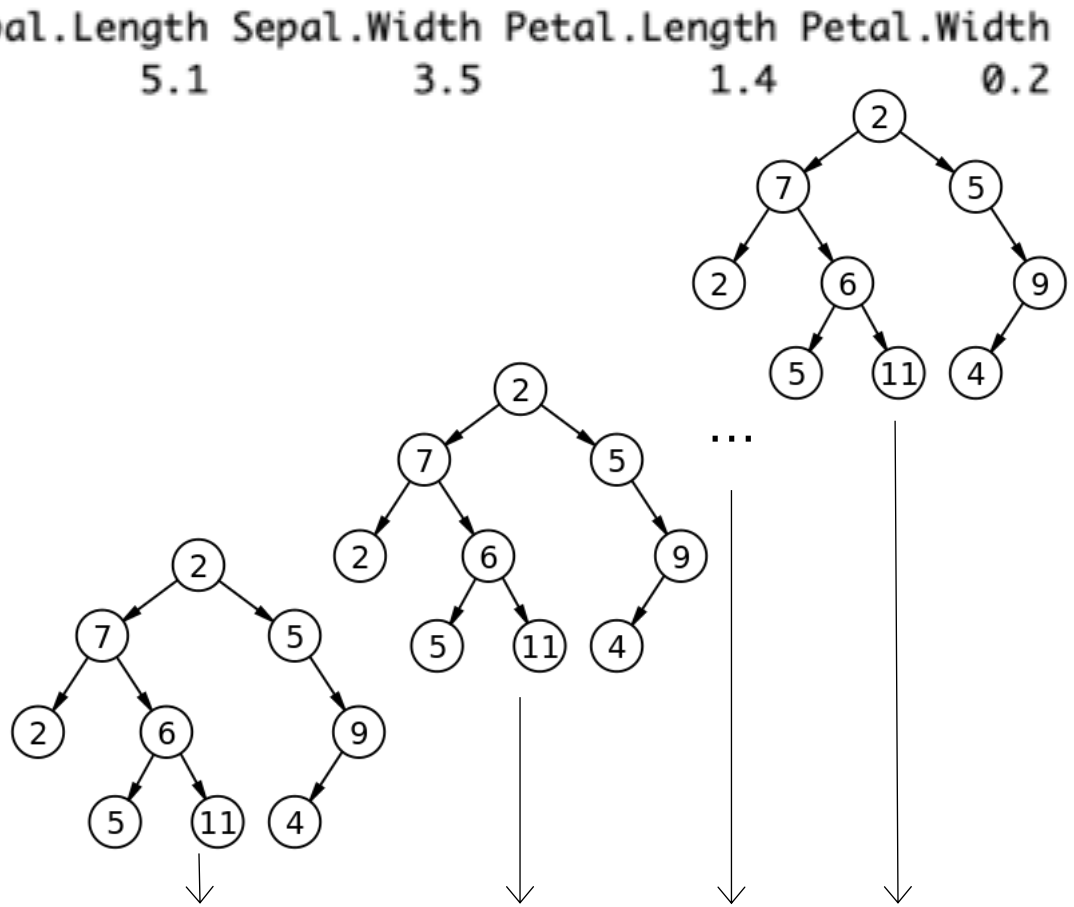
Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

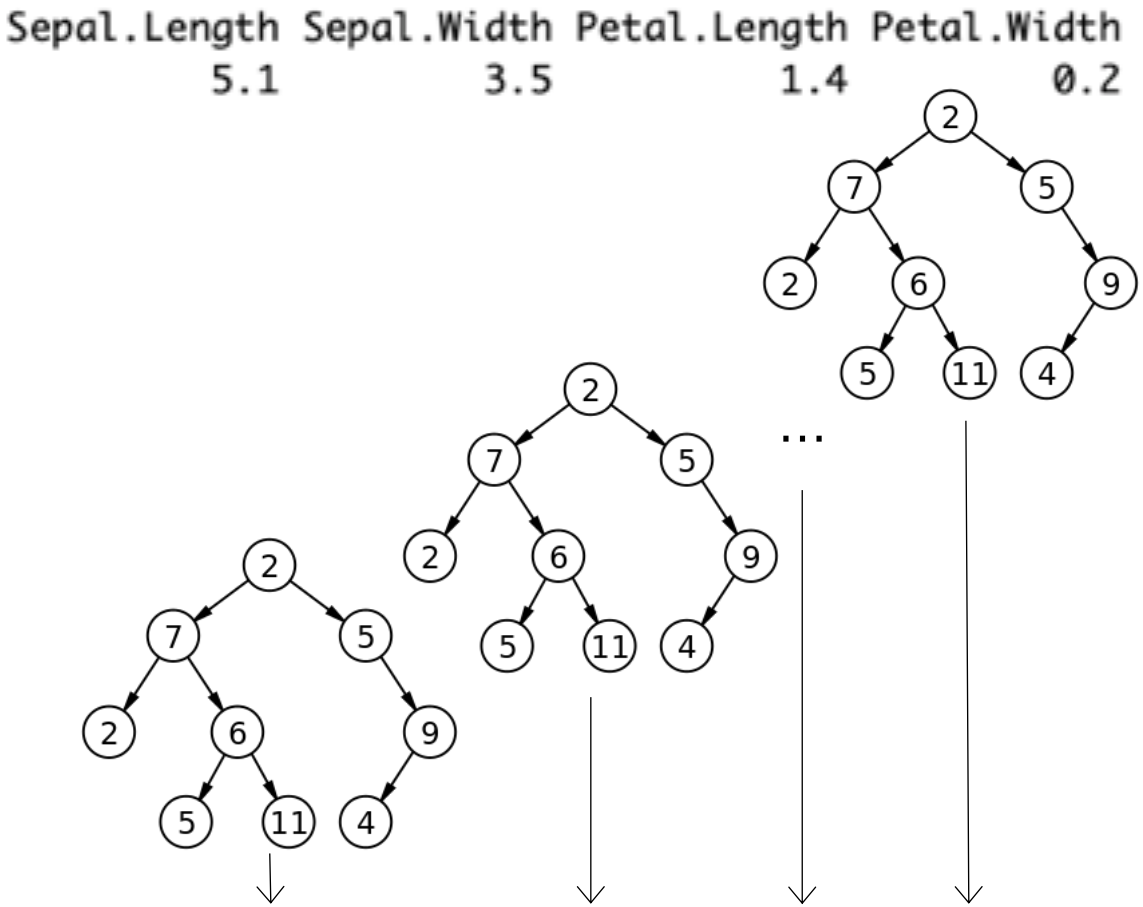
Hastie, Tibshirani, and Friedman (2009), The Elements of Statistical Learning: Data Mining, Inference, and Prediction

Regression

Run a set of observations (= 1 row) through all trees



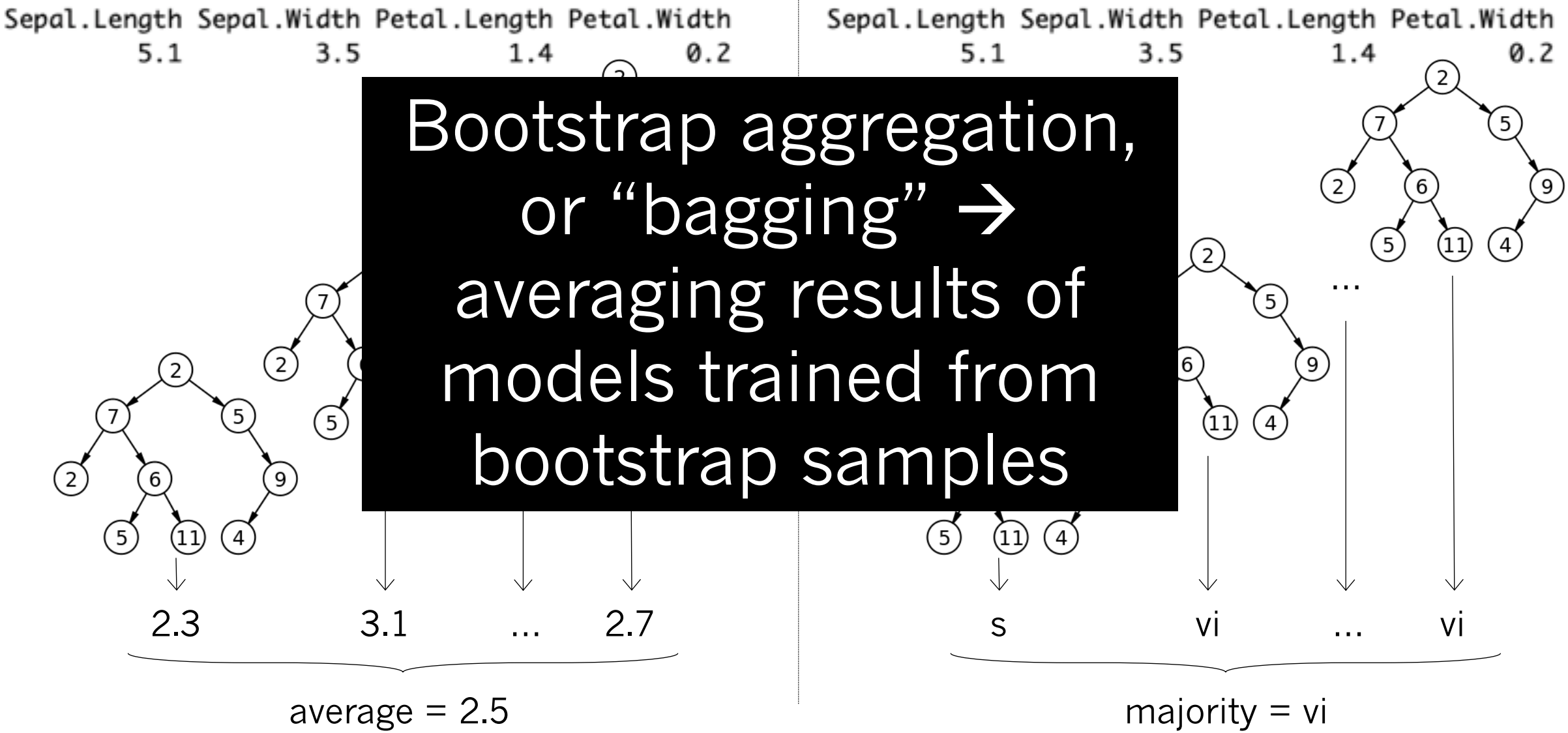
Classification



Regression

Run a set of observations (= 1 row) through all trees

Classification



Revisiting the definition: Random Forest

A bootstrap aggregation (“bagging”) and split-variable randomization approach to developing an ensemble of classification or regression trees

Breiman (2001), Random forests, Machine Learning 45(1): 5-32

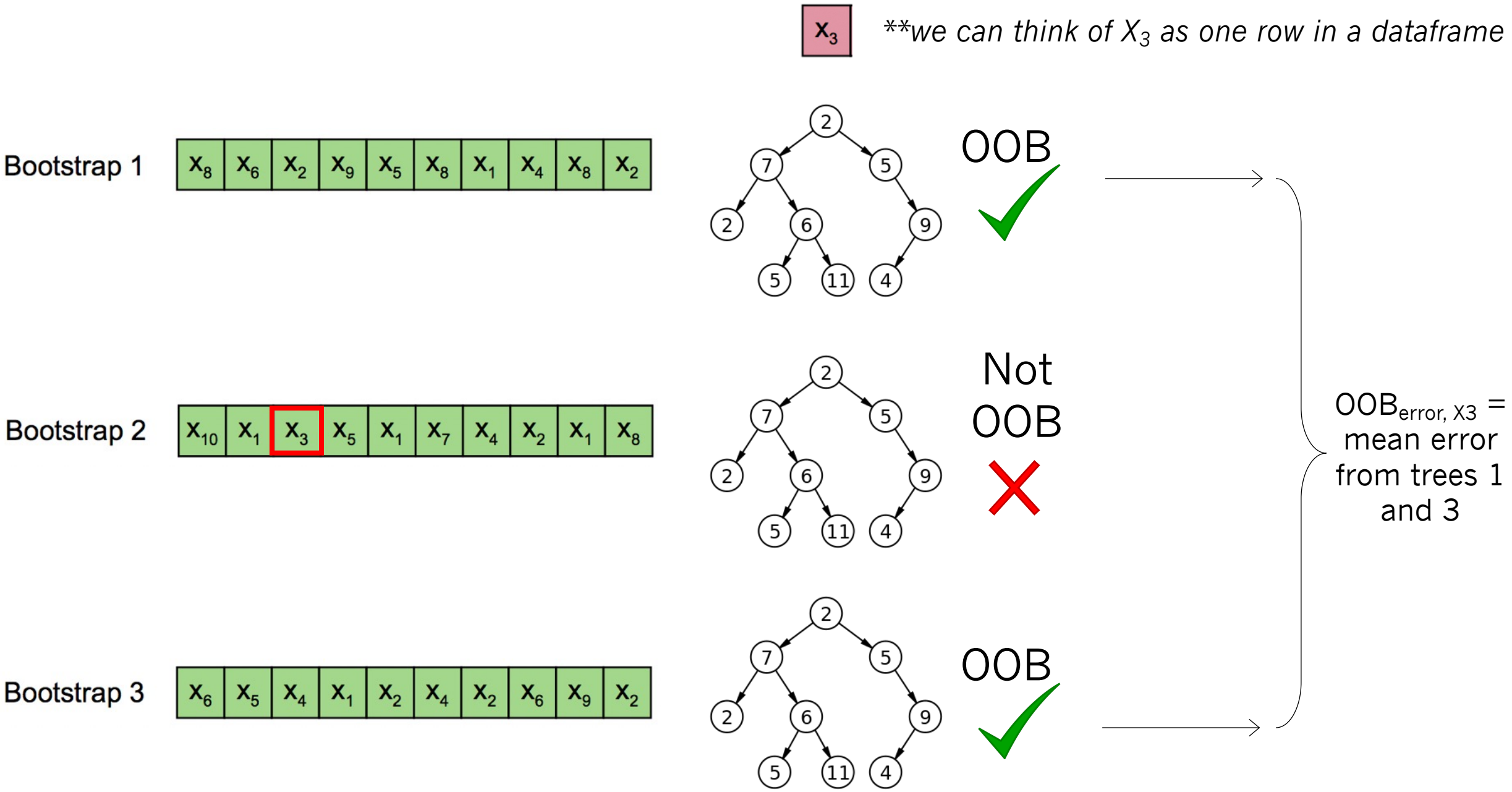
Error statistics for Random Forest

We can use the same approaches we've discussed previously to estimate Random Forest model error, but we have one additional option: **out of bag (OOB) error**. OOB error can be calculated for any approach that uses bagging.

OOB_{error} = mean error for each value of X_i estimated from trees that were <u>not</u> trained from a bootstrap sample with X_i

OOB error is regularly reported for Random Forest models

What is an "**out of bag**" sample?


x_3
**we can think of X_3 as one row in a dataframe
Bootstrap 1
x_8 x_6 x_2 x_9 x_5 x_8 x_1 x_4 x_8 x_2
OOB
Bootstrap 2
x_{10} x_1 x_3 x_5 x_1 x_7 x_4 x_2 x_1 x_8
Not OOB
Bootstrap 3
x_6 x_5 x_4 x_1 x_2 x_4 x_2 x_6 x_9 x_2
OOB
$OOB_{error, X3}$ = mean error from trees 1 and 3

Important!

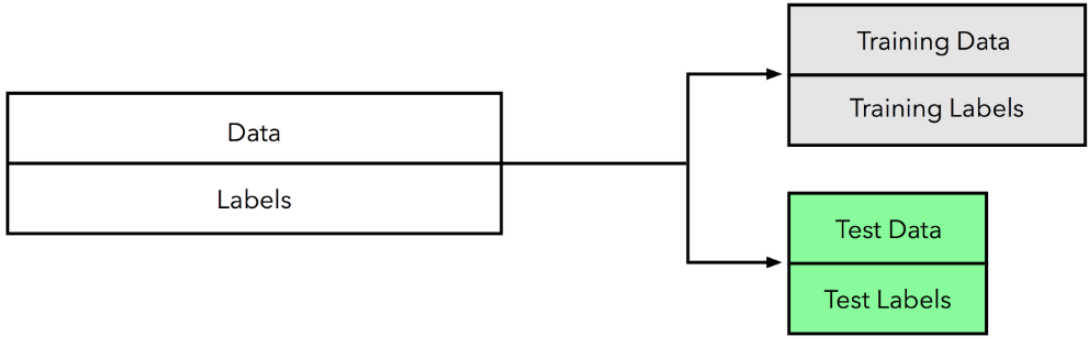
- The Random Forest algorithm is applied to the training data
  - The bootstrap samples are pulled from the training data
  - The OOB error estimates are calculated from “out of bag” samples in the training data
- Some modelers argue that the ability to use OOB error to assess error means that k-fold cross-validation is not necessary

Random Forest workflow (data splitting + algorithm)

1. Data are split into training/testing (partition, k-folds)
2. The test data subset is put aside
3. The training data are fed into the Random Forest algorithm with the `train()` function in the caret package. The `train()` algorithm:
   - Trains multiple versions of the model with different values of `mtry`
     - The approach for estimating mtry is specified with `trainControl()`. With Random Forest, we can use "oob", cross validation ("cv"), and more. See help documentation for all options. **I recommend oob!**
   - The model with the lowest error is selected, and this model becomes the output of the `train()` function; the corresponding values of mtry, model error are included in the output
     - The model error essentially describes the "goodness-of-fit" of the model
     - Remember, a Random Forest model is an ensemble of trees
4. The test data subset is run through the model with `predict()`, and error statistics are calculated to determine the model's predictive skill

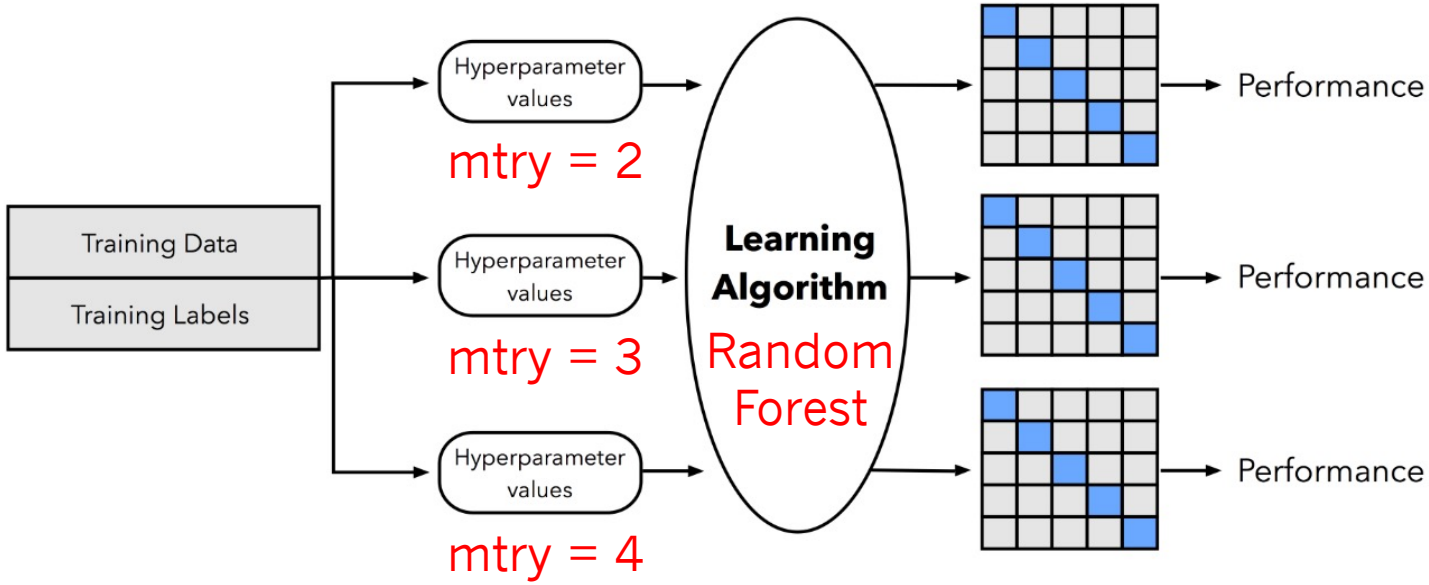
Split the data into training and testing subsets

1



Train the model using different parameter values

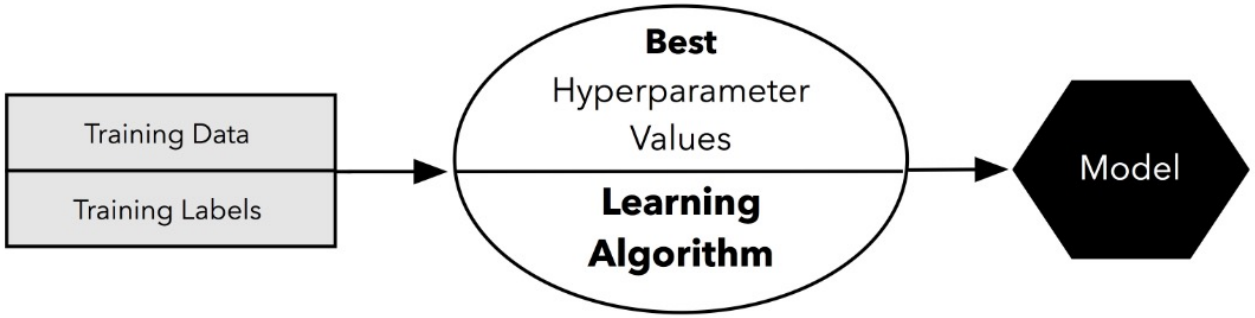
2



Here, the gray cells are the bootstrap samples, and the blue cells are the "out of bag" samples. The OOB samples are used to determine error ("performance")

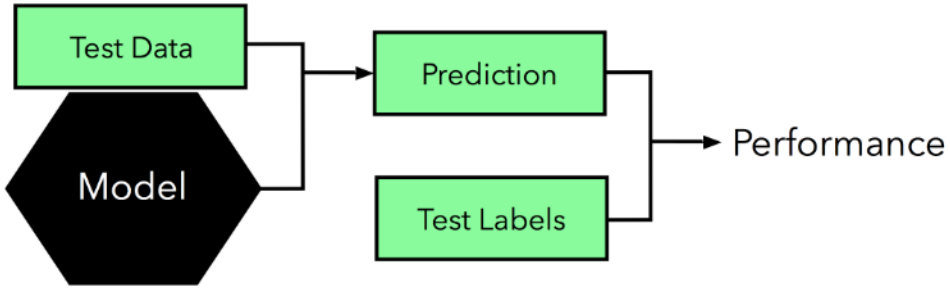
The final model; associated with one value of mtry (e.g. mtry = 2)

3



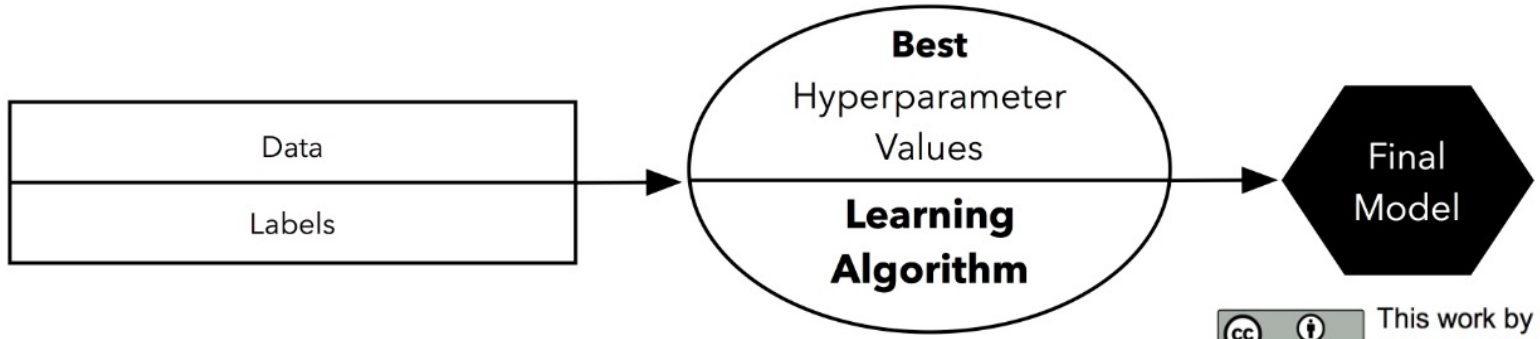
The test data are run through the model

4

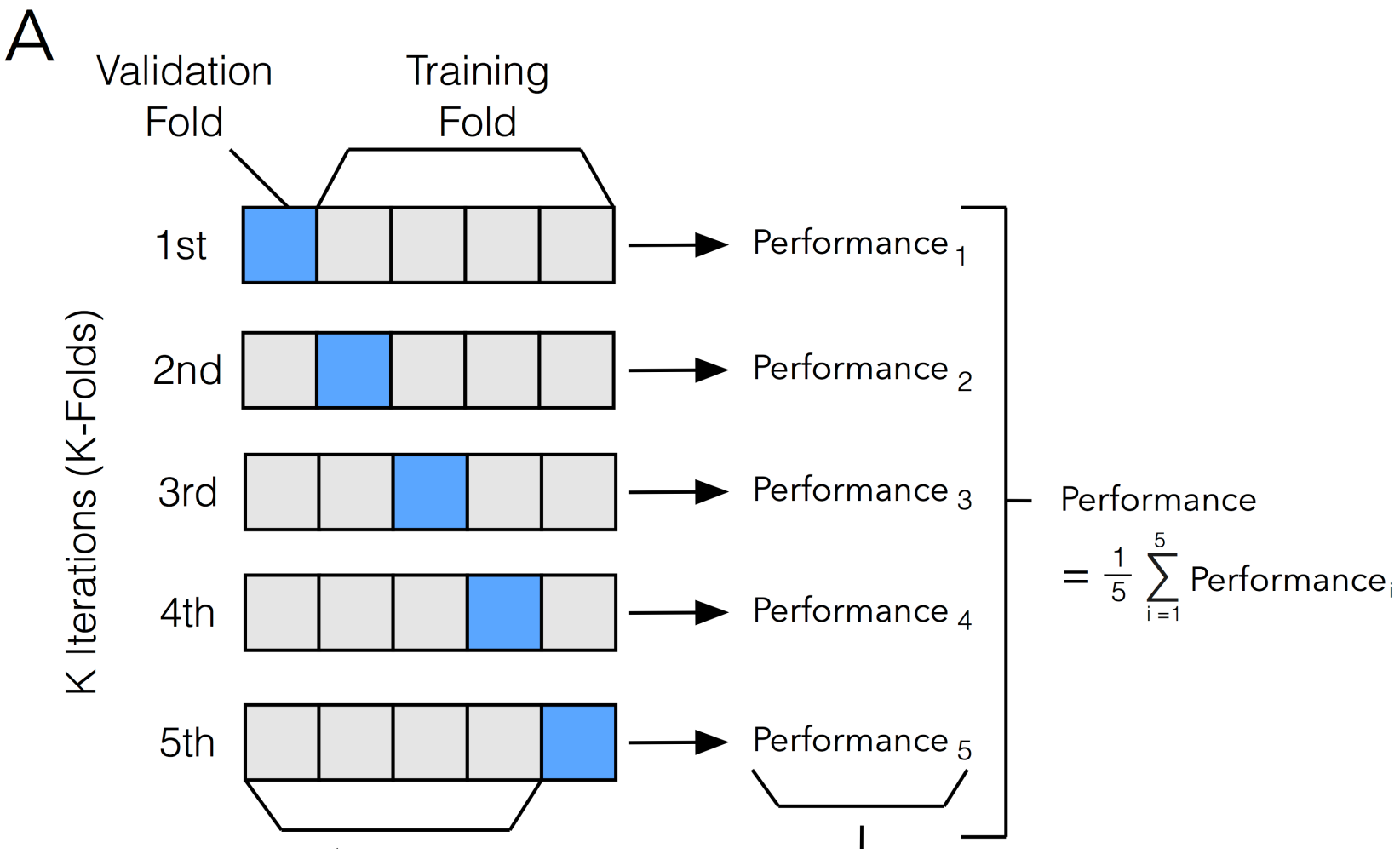


The final model is created

5



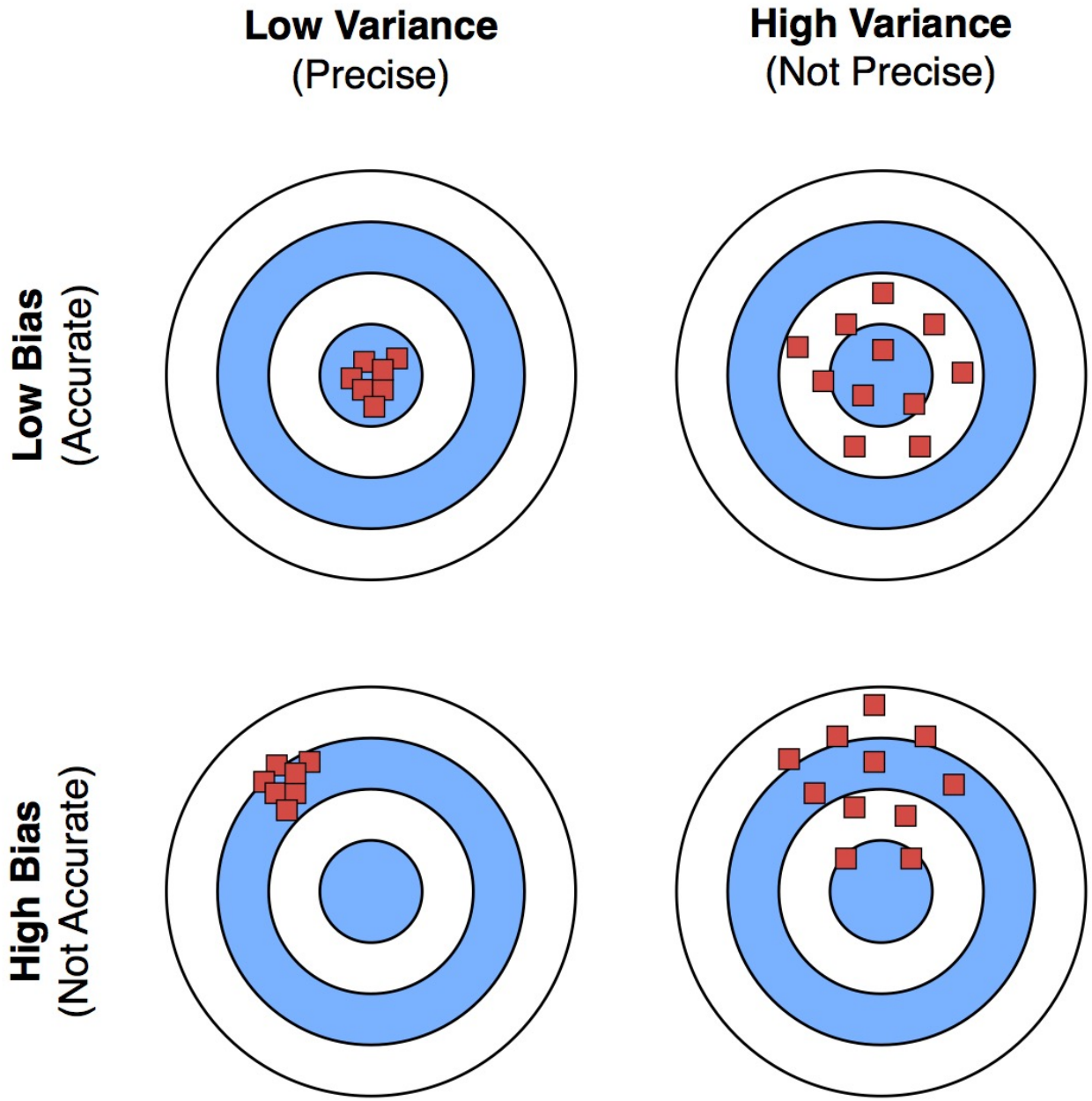
The previous example involved a data split (one training set, one testing set). When we run k-fold cross-validation, we repeat **k** times:



When thinking about error, it's important to also think about the **bias-variance tradeoff**

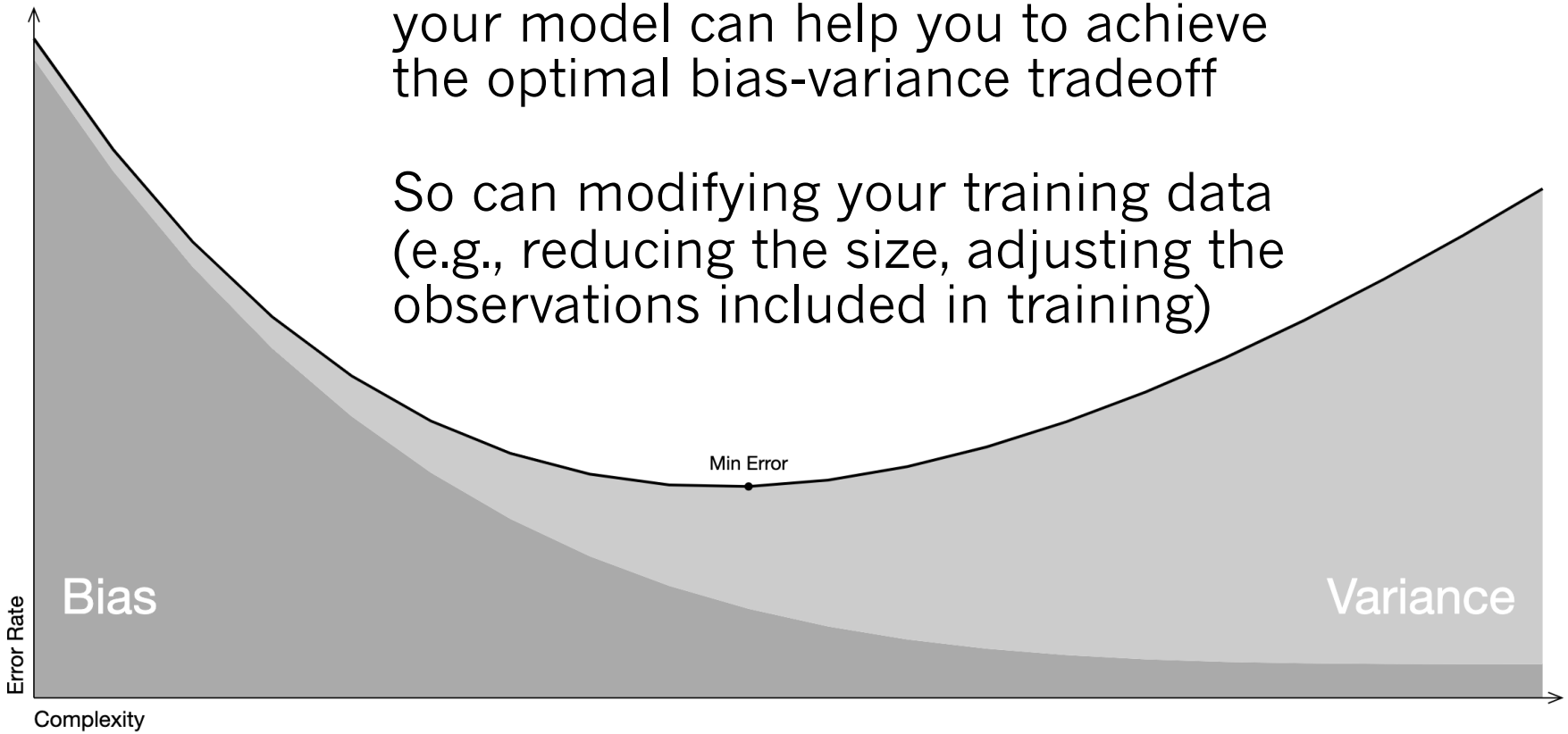
Bias: Wrong in consistent ways (the model does not capture enough detail, "underfit")

Variance: Wrong in inconsistent ways (the model captures too much detail, "overfit")

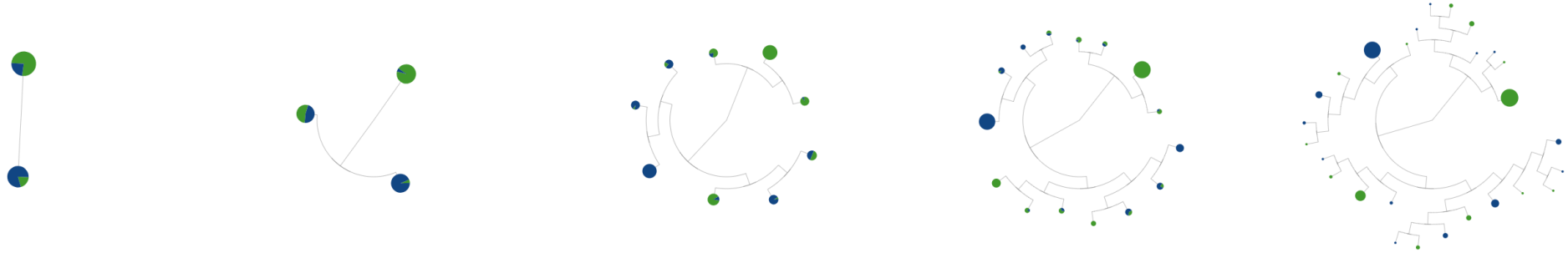


Adjusting the minimum node size in your model can help you to achieve the optimal bias-variance tradeoff

So can modifying your training data (e.g., reducing the size, adjusting the observations included in training)



Simple decision tree ("stump")



Detailed decision tree ("fully grown")

Source: R2D3, http://www.r2d3.us/visual-intro-to-machine-learning-part-2/

Feature importance

We know machine learning models are essentially black boxes, but it would be valuable to know which predictor variables have the greatest influence on model predictions.

How do you think we would go about identifying important predictors?

Feature importance

There are a few ways of evaluating variable or “feature” importance, with one of the most common metrics being permutation importance.

Permutation importance: a predictor is “important” if model prediction error increases after **permuting** the predictor and running it through the model. Also sometimes referred to as a **mean decrease in accuracy** method.

Permutation importance

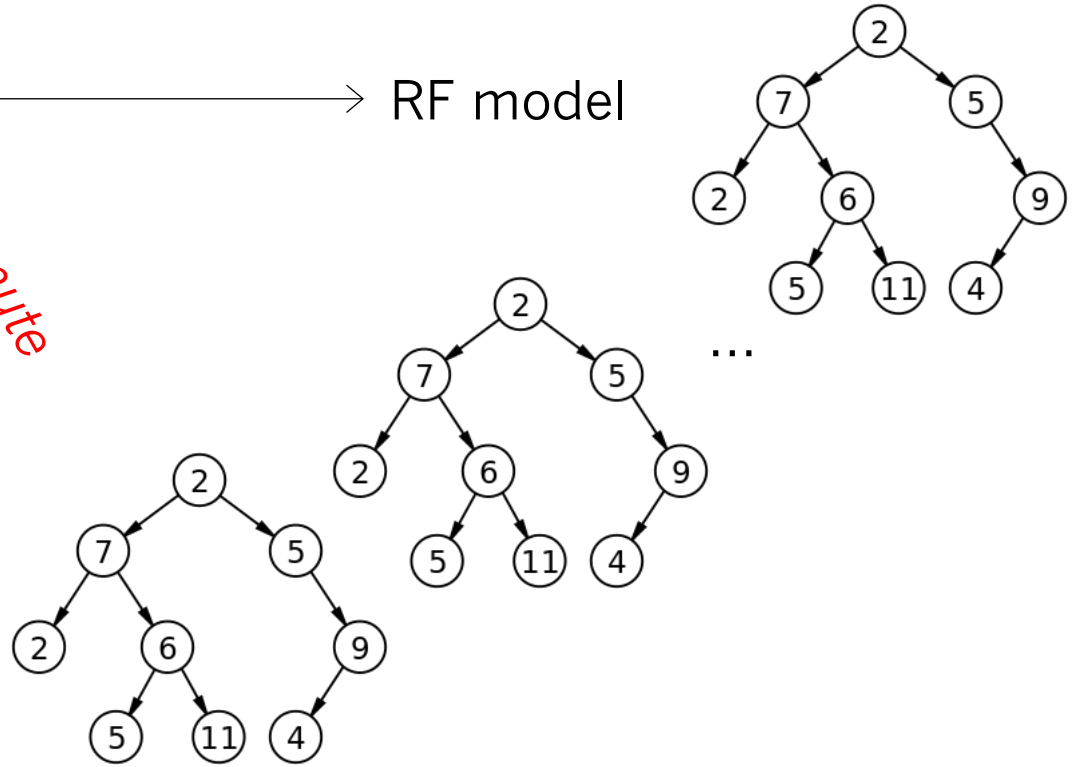
Permute a predictor → RF model

Sepal.Length	Sepal.Width
5.1	3.0
4.9	3.6
4.7	3.5
4.6	3.2
5.0	3.9
5.4	3.1

permute

Original

Sepal.Length	Sepal.Width
5.1	3.5
4.9	3.0
4.7	3.2
4.6	3.1
5.0	3.6
5.4	3.9

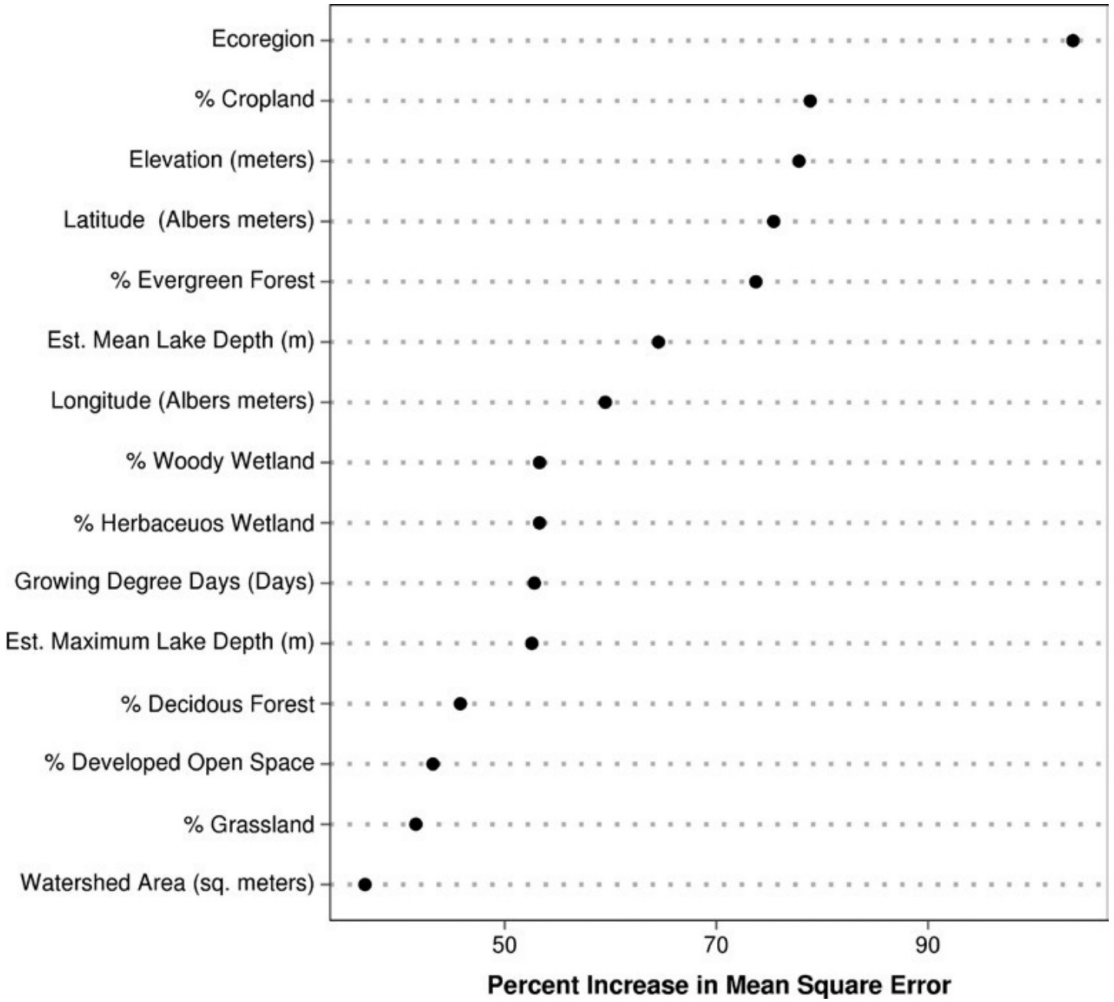


$MSE_{permuted}$

% change in MSE $\dfrac{MSE_{permuted} - MSE_{baseline}}{MSE_{baseline}} \times 100$

Larger value = more important

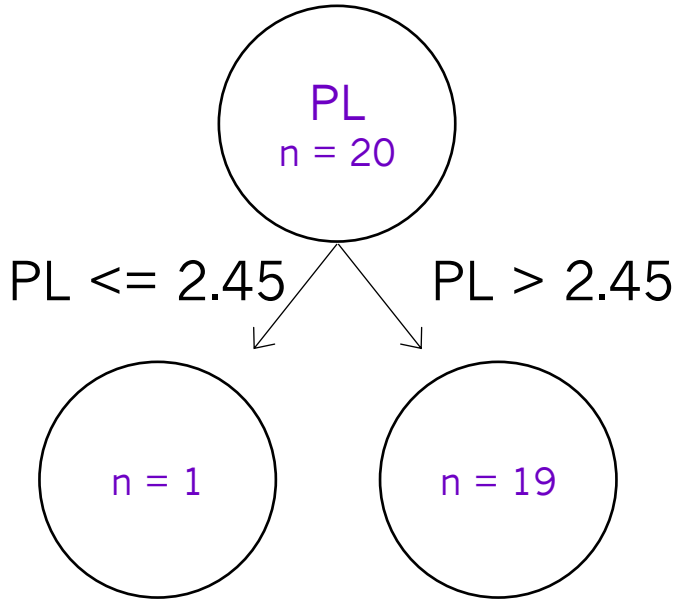
Permutation importance



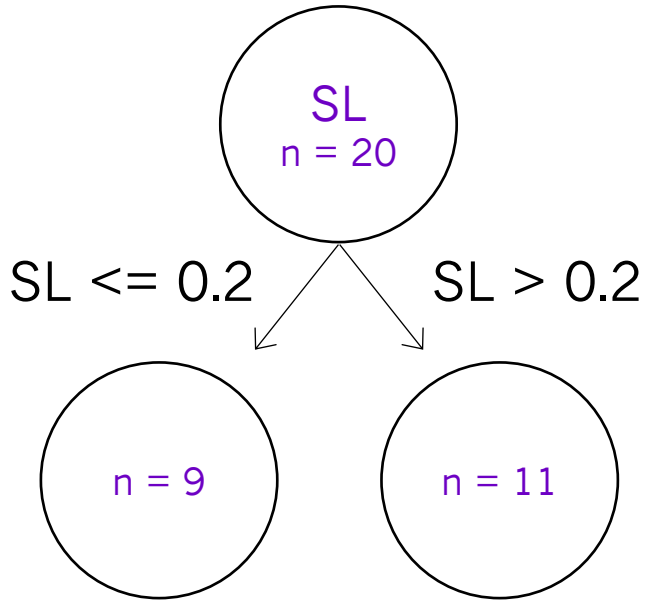
Hollister et al. (2016), Modeling lake trophic state: A random forest approach, *Ecosphere* 7(3):1-14

Feature importance

Node purity: a predictor is "important" if it consistently produces nodes of greater purity upon being split. Also sometimes referred to as a **mean decrease in impurity** method. Measured with the Gini index for classification models. *Note: I'm not a fan of this approach because the metric is difficult to interpret.*



more pure



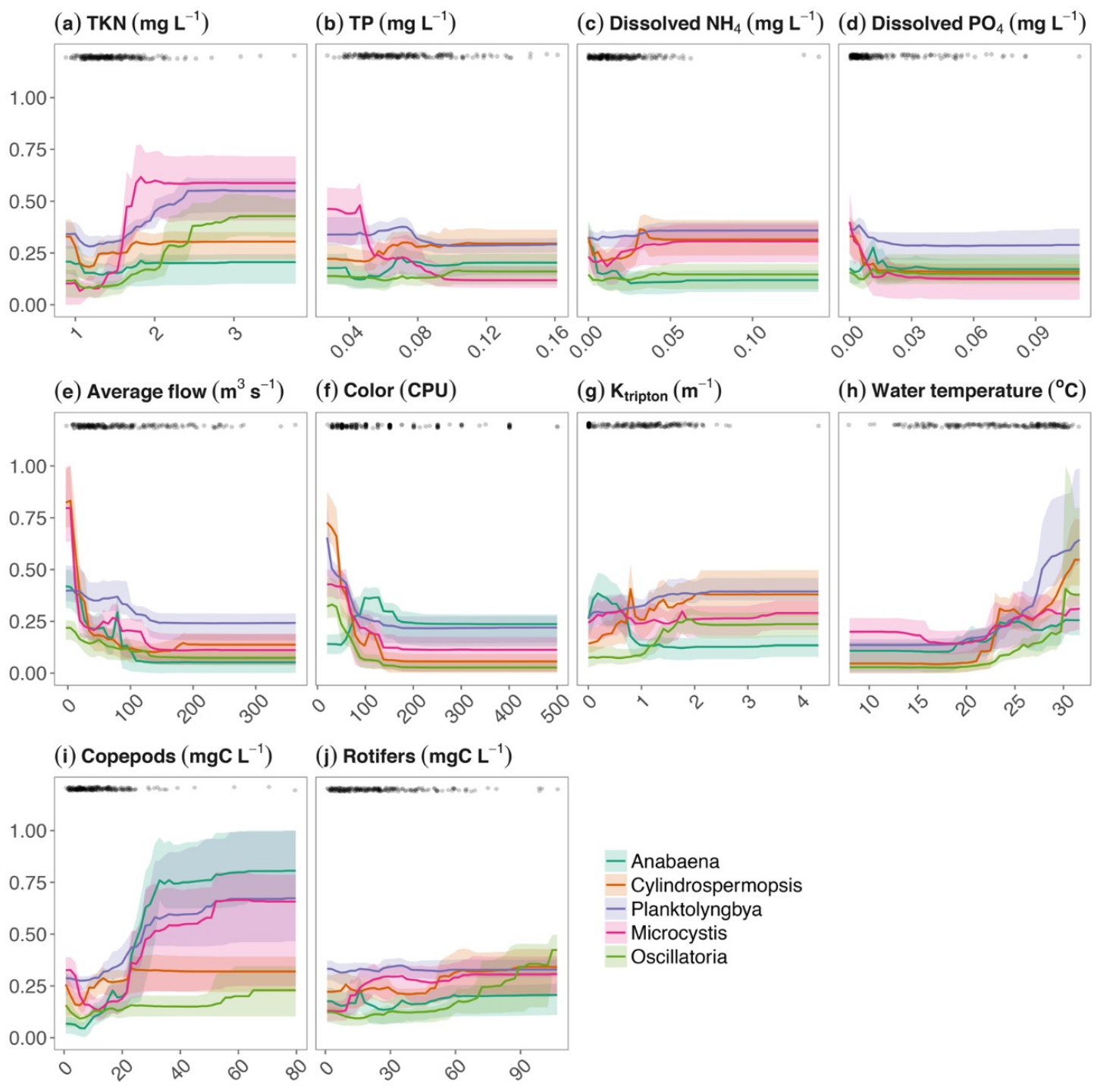
less pure

Partial dependence

The marginal effect of a variable on the class probability (classification) or response (regression).

Said another way, partial dependence shows you how your response/target changes in the model relative to one of your predictors.

Helps you “see in the black box”



Feature selection

The process of determining which features to include in the model. Ideally, you will have a model that is **parsimonious**, i.e., it achieves satisfactory model performance while minimizing the number of predictor variables. Feature selection helps us achieve parsimony.

How might you go about feature selection?

Feature selection

First, **use your expertise!**

- Justify each candidate predictor variable, ideally with some sort of causal mechanism
- Remove redundant variables
  - Not actually necessary, but helps with model interpretability
  - Check for collinearity: Variance inflation factor. Can also look at correlations.
  - When determining which collinear variables to remove, consider what makes the most sense

Feature selection

Next, use a **feature selection algorithm** to identify optimal predictors. There are many algorithms to choose from.

Simplest: Recursive feature elimination. Train/test a model with all predictors. Remove the least important variable and re-train/test. Repeat until a change in error is not noted.

More sophisticated: Boruta algorithm. Builds upon the permutation importance concept. A RF model is trained with all predictors AND permuted versions of all predictors. If a predictor's importance is greater than the importance of the most important randomly permuted variable (i.e., a nonsense variable), it is kept.

How many trees should be in a Random Forest?

11.4.1 Number of trees

The first consideration is the number of trees within your random forest. Although not technically a hyperparameter, the number of trees needs to be sufficiently large to stabilize the error rate. A good rule of thumb is to start with 10 times the number of features as illustrated in Figure 11.1; however, as you adjust other hyperparameters such as m_{try} and node size, more or fewer trees may be required. More trees provide more robust and stable error estimates and variable importance measures; however, the impact on computation time increases linearly with the number of trees.

Start with $p \times 10$ trees and adjust as necessary

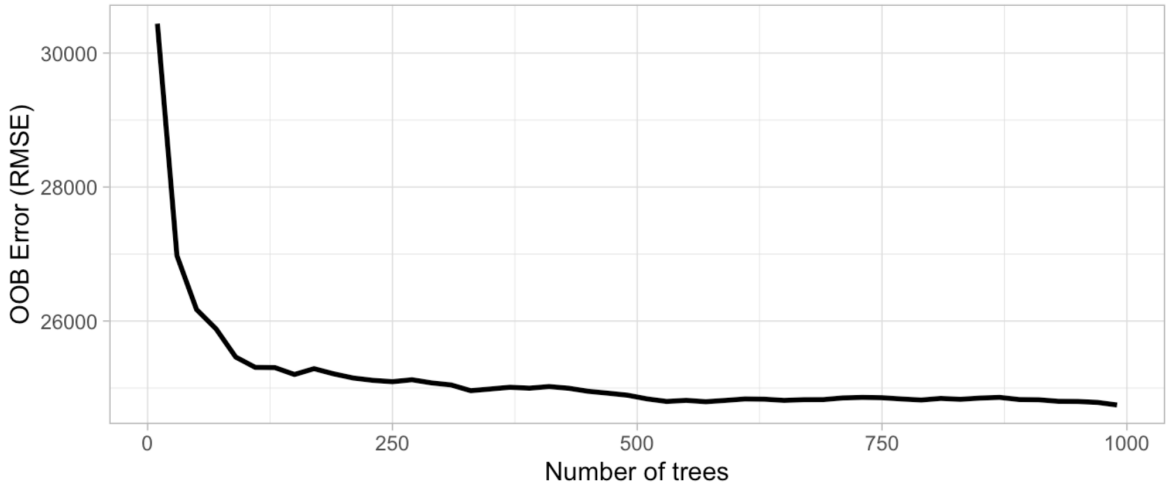


Figure 11.1: The Ames data has 80 features and starting with 10 times the number of features typically ensures the error estimate converges.

Boehmke and Greenwell (2020), eBook: Hands-On Machine Learning with R. https://bradleyboehmke.github.io/HOML/random-forest.html

How many trees should be in a Random Forest?

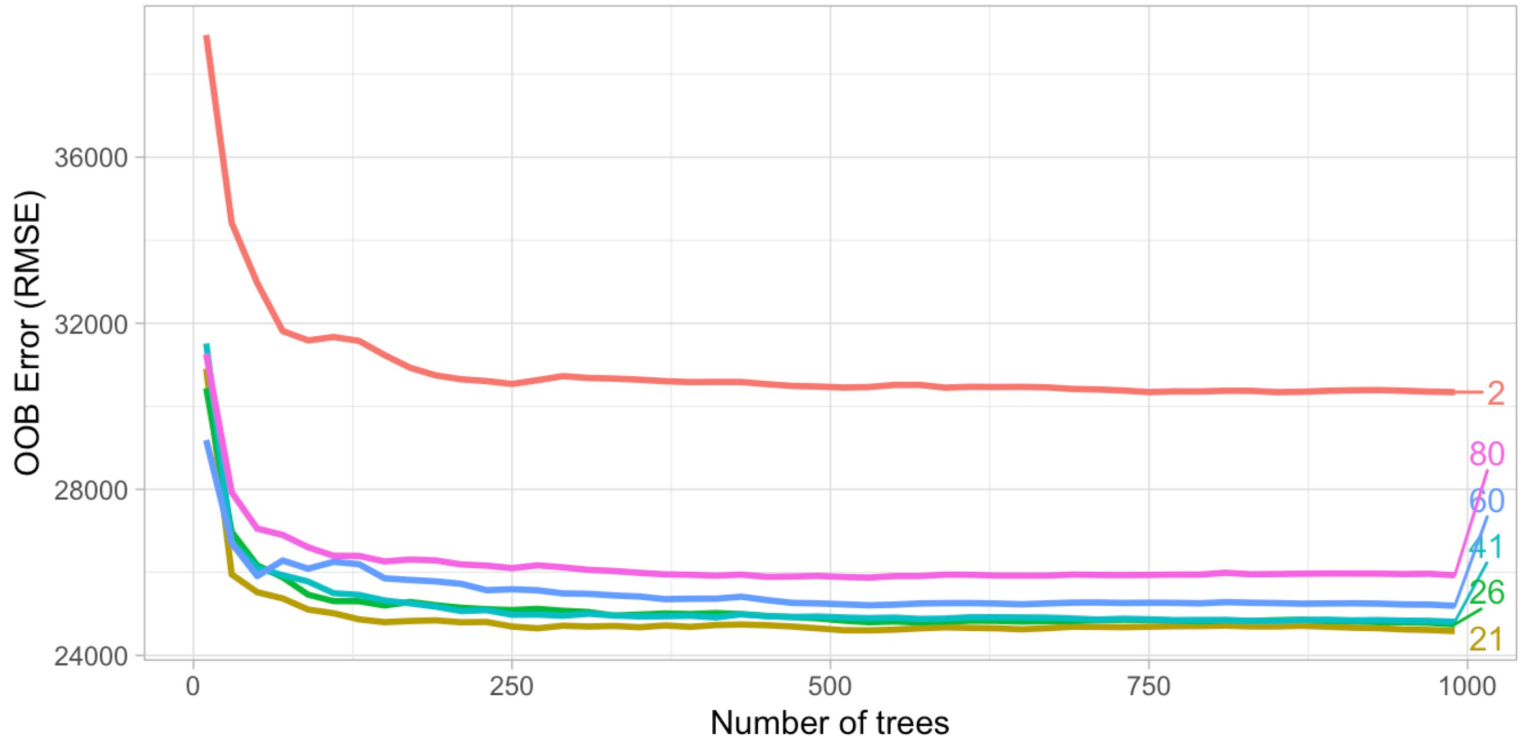


Figure 11.2: For the Ames data, an mtry value slightly lower (21) than the default (26) improves performance.

Usually, you can use the default number of trees, which is usually around 500

You can also choose to grow a large number of trees (e.g. 1000), but your model will take longer to run

Boehmke and Greenwell (2020), eBook: Hands-On Machine Learning with R. https://bradleyboehmke.github.io/HOML/random-forest.html

Random Forest with **caret**: `train( )`

```
ML_model <-
  train(
    Species ~ .,
    data = training,
    method = "rf",
    importance = TRUE,
    trControl = trainControl(method = "none")
  )
```

Max Kuhn, The caret Package, eBook: http://topepo.github.io/caret/index.html

6 Available Models

The models below are available in `train`. The code behind these protocols can be obtained using the function `getModelInfo` or by going to the github repository.

Show 238 entries

Search: random forest

Model	*method* Value	Type	Libraries	Tuning Parameters
Conditional Inference Random Forest	cforest	Classification, Regression	party	mtry
Oblique Random Forest	ORFlog	Classification	obliqueRF	mtry
Oblique Random Forest	ORFpls	Classification	obliqueRF	mtry
Oblique Random Forest	ORFridge	Classification	obliqueRF	mtry
Oblique Random Forest	ORFsvm	Classification	obliqueRF	mtry
Parallel Random Forest	parRF	Classification, Regression	e1071, randomForest, foreach, import	mtry
Quantile Random Forest	qrf	Regression	quantregForest	mtry
Random Forest	ordinalRF	Classification	e1071, ranger, dplyr, ordinalForest	nsets, ntreeperdiv, ntreefinal
Random Forest	ranger	Classification, Regression	e1071, ranger, dplyr	mtry, splitrule, min.node.size
Random Forest	Rborist	Classification, Regression	Rborist	predFixed, minNode
Random Forest	rf	Classification, Regression	randomForest	mtry
Random Forest by Randomization	extraTrees	Classification, Regression	extraTrees	mtry, numRandomCuts
Random Forest Rule-Based Model	rfRules	Classification, Regression	randomForest, inTrees, plyr	mtry, maxdepth
Regularized Random Forest	RRF	Classification, Regression	randomForest, RRF	mtry, coefReg, coefImp
Regularized Random Forest	RRFglobal	Classification, Regression	RRF	mtry, coefReg
Weighted Subspace Random Forest	wsrf	Classification	wsrf	mtry

Showing 1 to 16 of 16 entries (filtered from 238 total entries)

Random Forest with **caret**: `trainControl( )`

- When training a Random Forest model with `method = "rf"`, `trainControl( )` is used to tune the parameter `mtry`

```
1  Define sets of model parameter values to evaluate
2  for each parameter set do
3      for each resampling iteration do
4          Hold–out specific samples
5          [Optional] Pre–process the data
6          Fit the model on the remainder
7          Predict the hold–out samples
8      end
9      Calculate the average performance across hold–out predictions
10 end
11 Determine the optimal parameter set
12 Fit the final model to all the training data using the optimal parameter set
```

With `trainControl( )`, you specify the resampling approach, which you can also think of as the hold-out approach.

Method options for `trainControl( )` include:
"boot", "cv", "LOOCV", "LGOCV", "repeatedcv", "timeslice", "none", "**oob**"

On your own: Read through these webpages (~20 minutes)

Visualize how a regression or classification tree (also referred to as a “decision” tree) works

- http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

And then learn about the Bias-Variance Tradeoff

- http://www.r2d3.us/visual-intro-to-machine-learning-part-2/

Random Forest: **Summary**

- Random Forest is among the most popular and robust ML algorithms
- A RF model is an **ensemble** of classification or regression trees
- RF models use **bagging** (bootstrap aggregating) and **split-variable randomization** (m of p variables) to create trees
- To calculate “out of bag” error, we run individual samples through only the trees that were not fit (or “grown”) to them
- The Kappa coefficient is used to quantify the predictive quality of a classification model; it accounts for the expected rate of correctly classifying a response by chance
- One approach for estimating feature importance is **permutation importance**, in which you quantify what happens to model predictions when you shuffle or “permute” a predictor variable