SIRCSS – Machine Learning for Social Science

Lecture 1 (cont.)

Martin Arvidsson
Institute of Analytical Sociology, Linköping University

2025-12-01

First, quick recap of what we've covered so-far

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y.

First, quick recap of what we've covered so-far

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y.
- Two types of objectives: prediction vs. inference
  - $\hat{y}$ focus—predict as good as possible
  - $\hat{\beta}$ focus—understand the relation between X and y.

First, quick recap of what we've covered so-far

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y.
- Two types of objectives: prediction vs. inference
  - $\hat{y}$ focus—predict as good as possible
  - $\hat{\beta}$ focus—understand the relation between X and y.
- Limitations of standard linear paradigm
  - *Non-linearity*: standard linear models *underfit* data
  - *High-dimensionality*: standard linear models *overfit* data

First, quick recap of what we've covered so-far

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y.
- Two types of objectives: prediction vs. inference
  - $\hat{y}$ focus—predict as good as possible
  - $\hat{\beta}$ focus—understand the relation between X and y.
- Limitations of standard linear paradigm
  - *Non-linearity*: standard linear models *underfit* data
  - *High-dimensionality*: standard linear models *overfit* data
- This can be understood in terms of *Bias-Variance Trade-off*
  - *High bias*: $\hat{f}$ not capturing the underlying patterns in the data
  - *High variance*: $\hat{f}$ capturing noise in data and therefore being sensitive to the data it is estimated on.

First, quick recap of what we've covered so-far

- Supervised learning
  - Learn a function $\hat{f}$ that maps input X to output Y.
- Two types of objectives: prediction vs. inference
  - $\hat{y}$ focus—predict as good as possible
  - $\hat{\beta}$ focus—understand the relation between X and y.
- Limitations of standard linear paradigm
  - *Non-linearity*: standard linear models *underfit* data
  - *High-dimensionality*: standard linear models *overfit* data
- This can be understood in terms of *Bias-Variance Trade-off*
  - *High bias*: $\hat{f}$ not capturing the underlying patterns in the data
  - *High variance*: $\hat{f}$ capturing noise in data and therefore being sensitive to the data it is estimated on.
- Beyond the standard linear model
  - *Polynomials*—enable $\hat{f}$ to pick up more patterns in data ($\downarrow$bias).
  - *Ridge*—penalizes $\hat{f}$ that picks up too much patterns ($\downarrow$variance)
  - *Cross-validation*—helps find a good level of complexity; balancing bias- and variance trade-off.

Also—a brief note on classification

In today's lecture, I have (for reasons of brevity) focused on regression —where Y is *continuous*.

But all *concepts* & *methods* extend—with little change—to classification —where Y is *categorical*.

Also—a brief note on classification

In today's lecture, I have (for reasons of brevity) focused on regression —where Y is *continuous*.

But all *concepts* & *methods* extend—with little change—to classification —where Y is *categorical*.

One thing that does differ though: **how performance is measured**.

Also—a brief note on classification

In today's lecture, I have (for reasons of brevity) focused on regression —where Y is *continuous*.

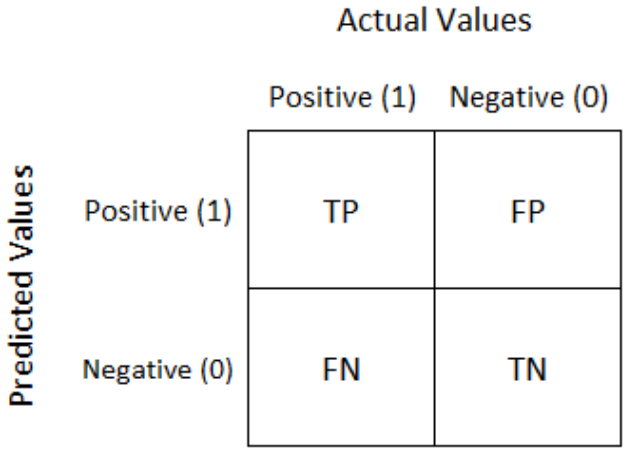
But all *concepts* & *methods* extend—with little change—to classification —where Y is *categorical*.

One thing that does differ though: **how performance is measured**.

- Most common measure for classification: **accuracy** (% correct classifications).

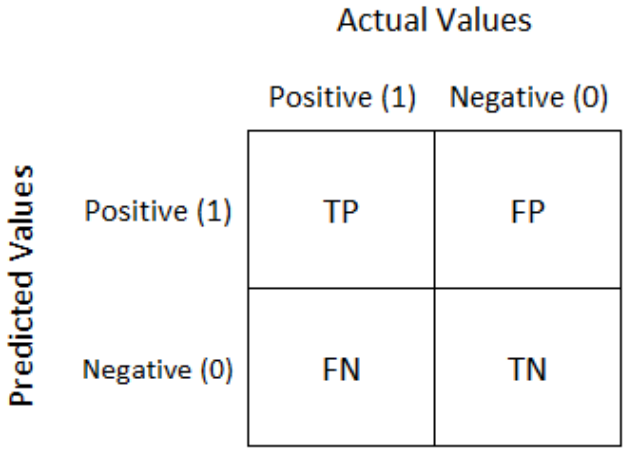
Also—a brief note on classification

Depending on the setting — it can also be worth distinguishing between *false positives* and *false negatives* (*true positives* and *true negatives*).



Also—a brief note on classification

Depending on the setting — it can also be worth distinguishing between *false positives* and *false negatives* (*true positives* and *true negatives*).



It may for example be that *certain errors are more important*. E.g.,

- Incorrectly classifying an individual with cancer as not having cancer (*false negative*) vs. incorrectly classifying an individual with no cancer as having cancer (*false positive*).

Remainder of lecture

Class of methods that 'learn' from data in more automatized ways.

- **Non-parametric** supervised learning
  - KNN
  - Decision trees
  - Random forest

Non-parametric SL

Parametric vs. non-parameteric models

Parametric models

- Methods covered so-far.
- Makes strong assumptions about the properties of $\hat{f}$: e.g., *linear*, *additive*, *Gaussian errors*.

Parametric vs. non-parameteric models

Parametric models

- Methods covered so-far.
- Makes strong assumptions about the properties of $\hat{f}$: e.g., *linear*, *additive*, *Gaussian errors*.
- **Strengths** (because of strong assumptions...)
  - Requires *relatively little data* required for estimation.
  - Easy to *interpret.*
  - Facilitates *inference*.

Parametric vs. non-parameteric models

Parametric models

- Methods covered so-far.
- Makes strong assumptions about the properties of $\hat{f}$: e.g., *linear*, *additive*, *Gaussian errors*.
- **Strengths** (because of strong assumptions...)
  - Requires *relatively little data* required for estimation.
  - Easy to *interpret.*
  - Facilitates *inference*.
- **Weaknesses**
  - If strong assumptions are not met, performance suffers.
  - While extensions relax *some* assumptions, they only do so *partly*.

Parametric vs. non-parameteric models

Parametric models

- Methods covered so-far.
- Makes strong assumptions about the properties of $\hat{f}$: e.g., *linear*, *additive*, *Gaussian errors*.
- **Strengths** (because of strong assumptions...)
  - Requires *relatively little data* required for estimation.
  - Easy to *interpret.*
  - Facilitates *inference*.
- **Weaknesses**
  - If strong assumptions are not met, performance suffers.
  - While extensions relax *some* assumptions, they only do so *partly*.

Non-parametric models

- Does not make explicit assumptions about the parametric form of $\hat{f}$.
- Instead let's "data speak".

Which to use?

Depends on the *nature of your data* and the *objective of your analysis*.

Which to use?

Depends on the *nature of your data* and the *objective of your analysis*.

Inference

- *Parametric models* approaches are generally favorable.

Which to use?

Depends on the *nature of your data* and the *objective of your analysis*.

Inference

- *Parametric models* approaches are generally favorable.

Prediction

- If you know the functional form of $\hat{f}$ AND the mapping between X and Y is relatively simple $\rightarrow$ *parametric models*.

Which to use?

Depends on the *nature of your data* and the *objective of your analysis*.

Inference

- *Parametric models* approaches are generally favorable.

Prediction

- If you know the functional form of $\hat{f}$ AND the mapping between X and Y is relatively simple $\rightarrow$ *parametric models*.
- If not: *non-parametric models* — "let data speak"

Which to use?

Depends on the *nature of your data* and the *objective of your analysis*.

Inference

- *Parametric models* approaches are generally favorable.

Prediction

- If you know the functional form of $\hat{f}$ AND the mapping between X and Y is relatively simple $\rightarrow$ *parametric models*.
- If not: *non-parametric models* — "let data speak"

Non-parametric methods is what we'll consider now!

KNN

K-nearest neighbours (KNN)

One of the *simplest* and *best-known* non-parametric methods.

K-nearest neighbours (KNN)

One of the *simplest* and *best-known* non-parametric methods.

Basic idea:
Given a value for K (e.g., 10) and a *test* observation to predict:
$x_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$ *KNN*...

K-nearest neighbours (KNN)

One of the *simplest* and *best-known* non-parametric methods.

Basic idea:
Given a value for K (e.g., 10) and a *test* observation to predict:
$x_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$ *KNN*...

1. Identifies the K training observations "closest" to x_i—indexed by N_i.

K-nearest neighbours (KNN)

One of the *simplest* and *best-known* non-parametric methods.

Basic idea:
Given a value for K (e.g., 10) and a *test* observation to predict:
$x_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$ *KNN*...

1. Identifies the K training observations "closest" to x_i—indexed by N_i.
2. Estimates $\hat{f}(x_i)$ using:
   - Regression: *average* of all responses in N_i.

K-nearest neighbours (KNN)

One of the *simplest* and *best-known* non-parametric methods.

Basic idea:
Given a value for K (e.g., 10) and a *test* observation to predict:
$x_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$ *KNN*...

1. Identifies the K training observations "closest" to x_i—indexed by N_i.
2. Estimates $\hat{f}(x_i)$ using:
   - Regression: *average* of all responses in N_i.
   - Classification: *majority class* of all responses in N_i.

K-nearest neighbours (KNN)

One of the *simplest* and *best-known* non-parametric methods.

Basic idea:
Given a value for K (e.g., 10) and a *test* observation to predict:
$x_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$ *KNN*...

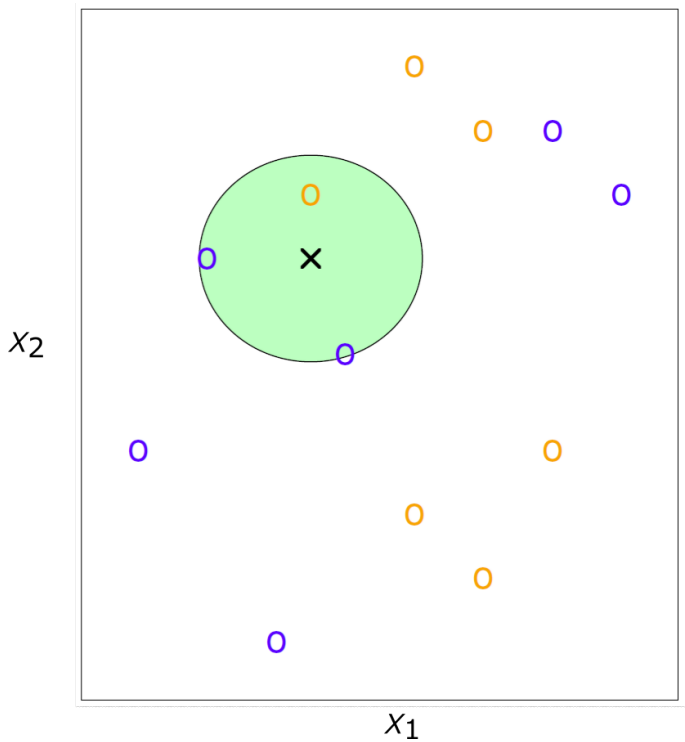
1. Identifies the K training observations "closest" to x_i—indexed by N_i.
2. Estimates $\hat{f}(x_i)$ using:
   - Regression: *average* of all responses in N_i.
   - Classification: *majority class* of all responses in N_i.

Let's consider a simple example to get some intuition!

Toy example

Data: suppose we have 12 observations, each described by:

- Two continuous input variables: X_1 and X_2.
- An outcome $Y \in \{\text{blue}, \text{orange}\}$.



KNN prediction

- Step 1: Identify the K (here: 3) closest *training observations*
- Step 2: Perform "majority-vote": 2/3 blue $\rightarrow \hat{y}_x =$ blue

How to think about the parameter K?

- K is set by the researcher.
- How should we reason when setting it?

How to think about the parameter K?

- K is set by the researcher.
- How should we reason when setting it?
- **Small** K
  - Local prediction
  - Implies *more* flexibility / wiggly prediction line.

How to think about the parameter K?

- K is set by the researcher.
- How should we reason when setting it?
- **Small** K
  - Local prediction
  - Implies *more* flexibility / wiggly prediction line.
- **Large** K
  - Predictions based on observations **farther away**
  - This "smoothens" prediction line (*less* flexibility)

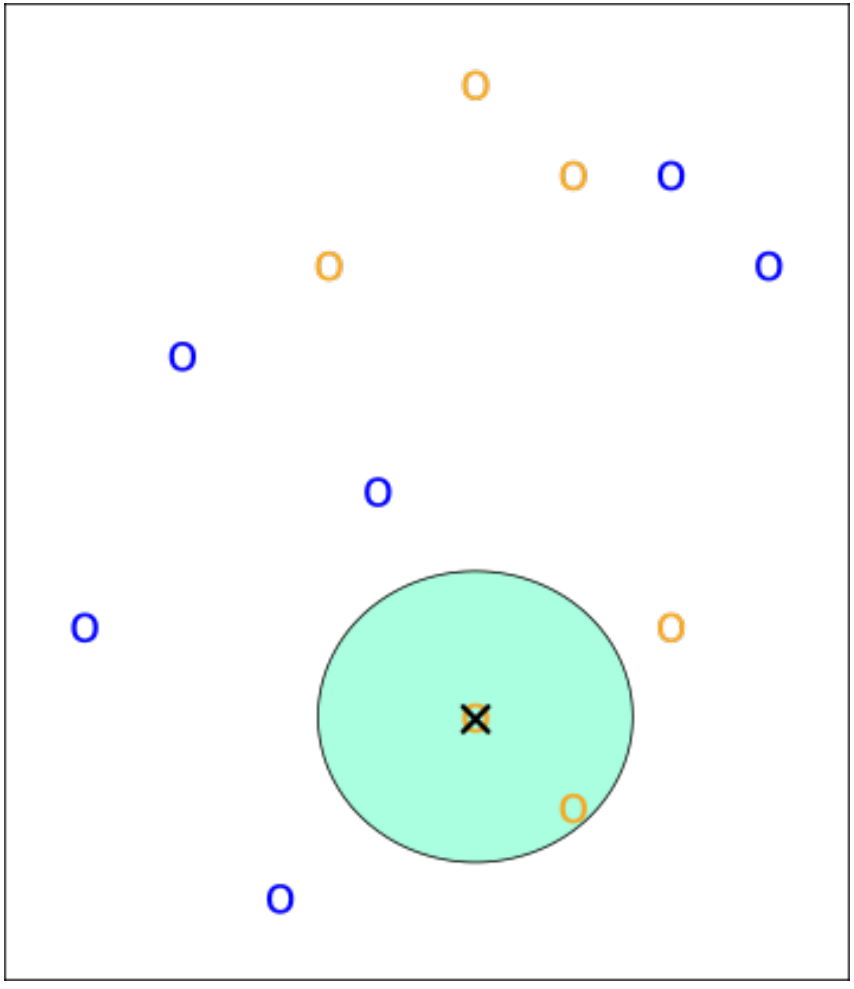
How to think about the parameter K?

- K is set by the researcher.
- How should we reason when setting it?
- **Small** K
  - Local prediction
  - Implies *more* flexibility / wiggly prediction line.
- **Large** K
  - Predictions based on observations **farther away**
  - This "smoothens" prediction line (*less* flexibility)

For demonstration, let's consider the toy example again!

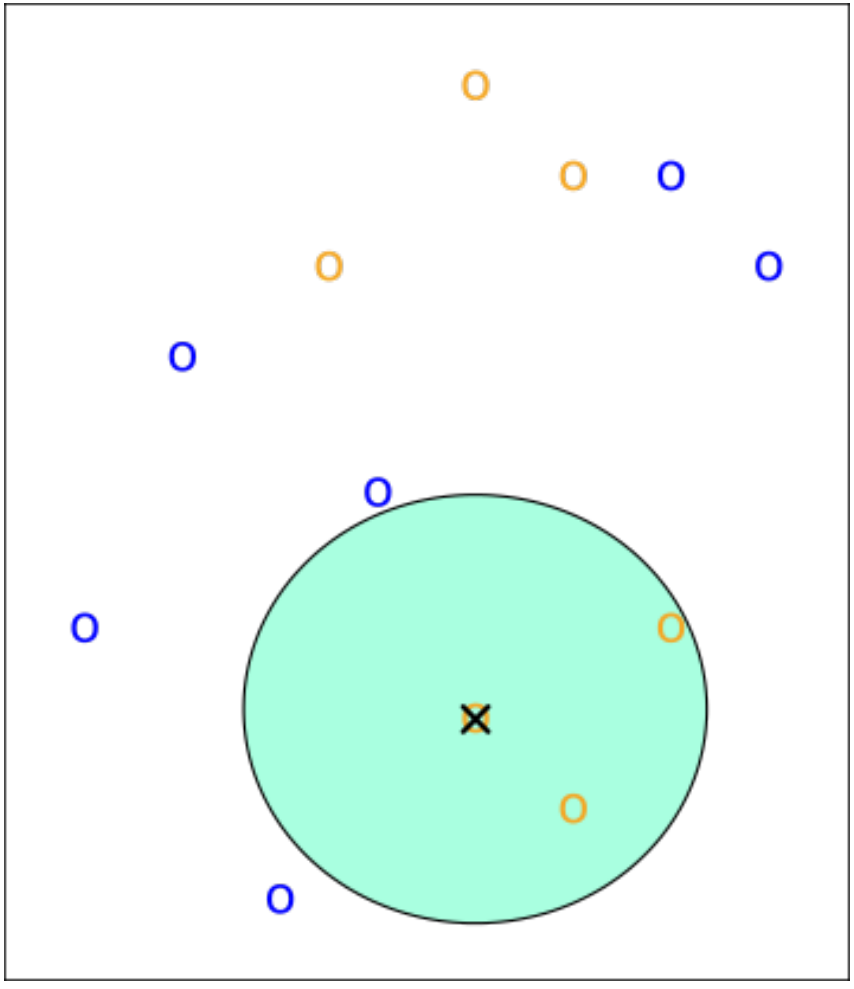
$K = 1$

Prediction: 1/1 orange $\rightarrow \hat{y}_x =$ orange (correct)



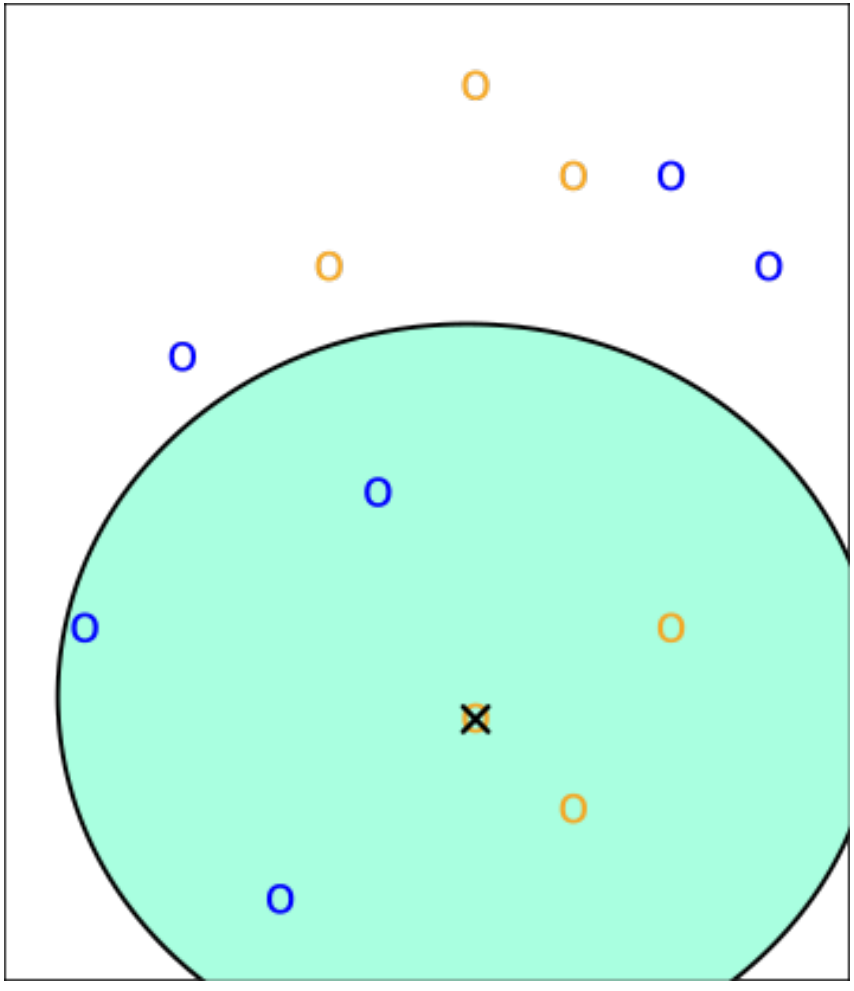
$K = 2$

Prediction: 2/2 orange $\rightarrow \hat{y}_x =$ orange (correct)



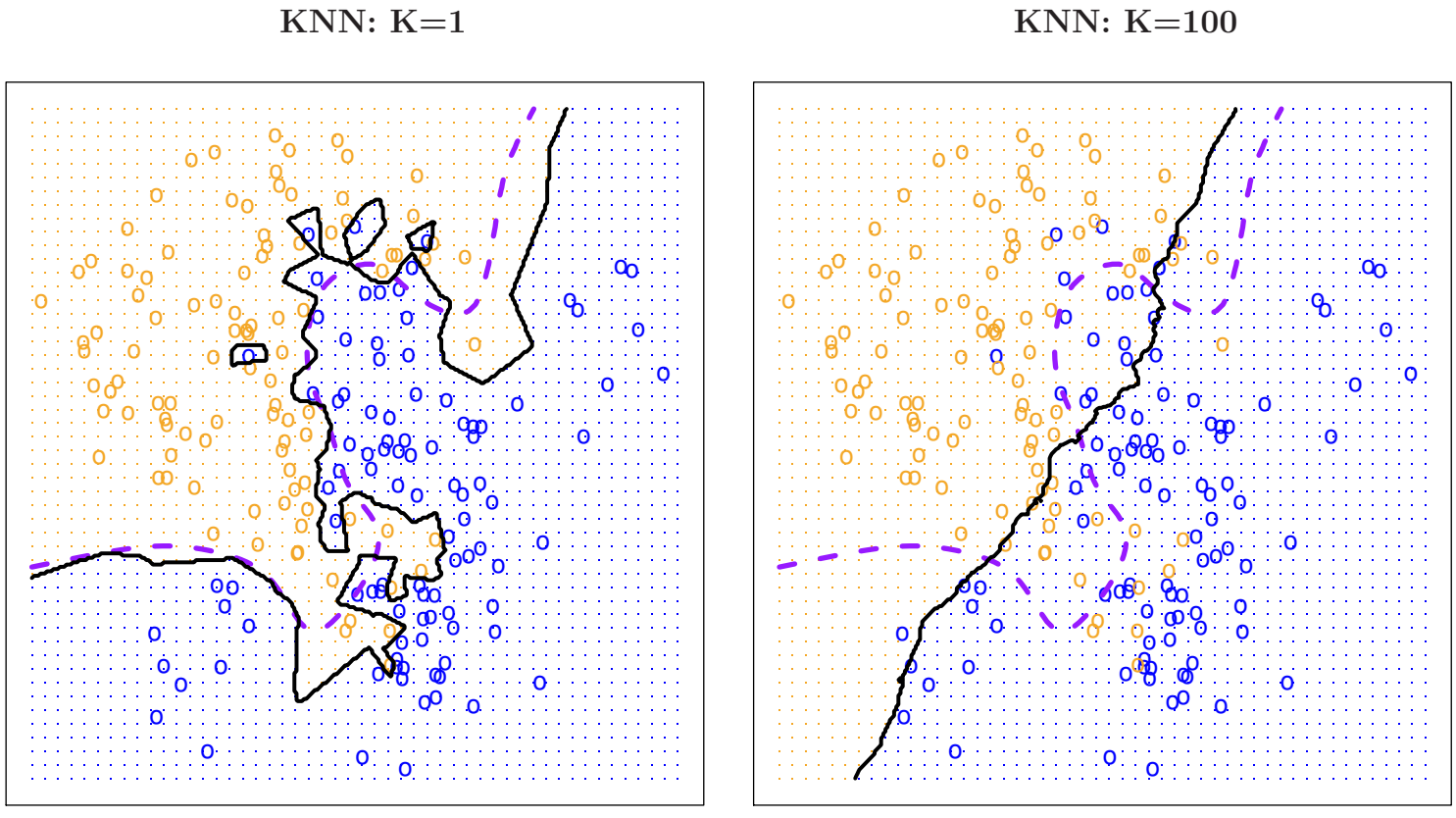
$K = 5$

Prediction: 3/5 blue $\rightarrow \hat{y}_x =$ blue (incorrect)



Applying this to all obs → *prediction boundary*

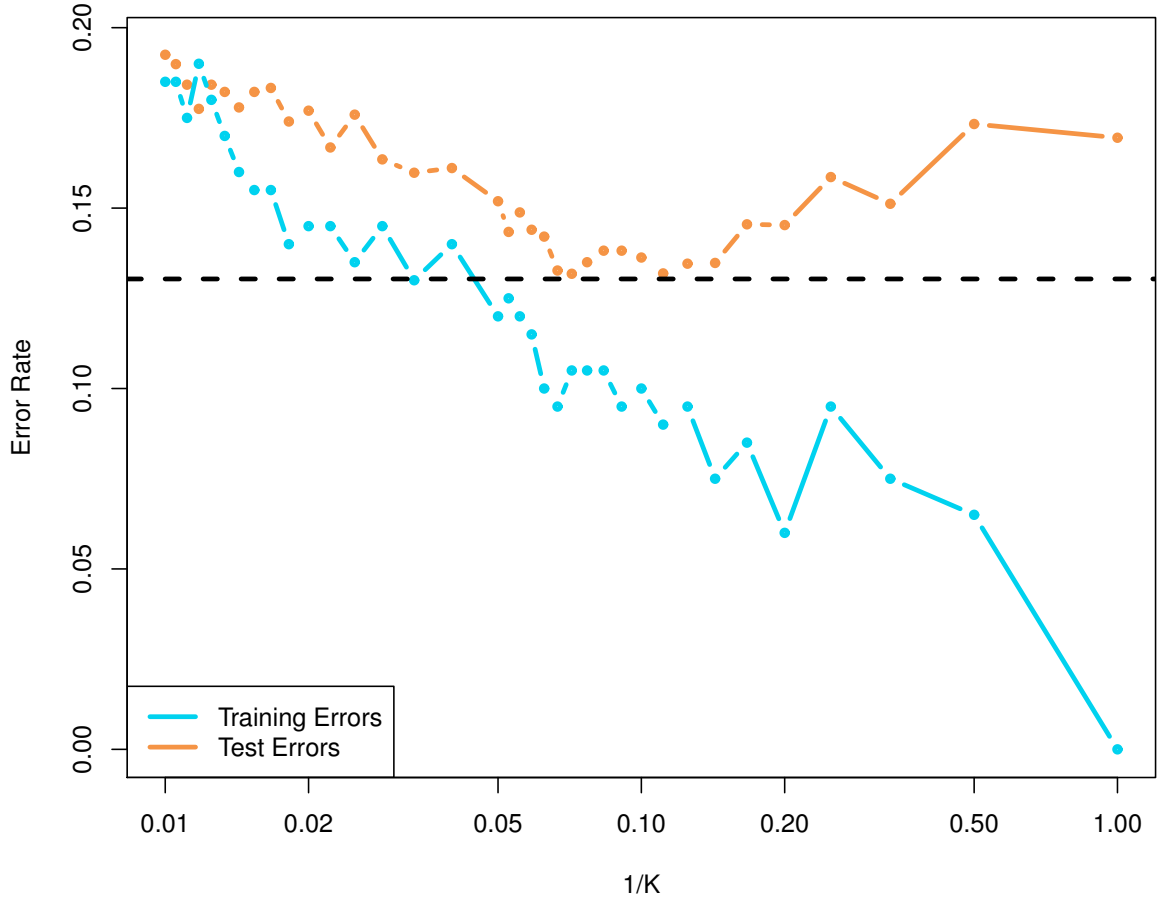
If we apply KNN-prediction to *all training observations*, one by one—for different K—we get the following *prediction boundaries* (black lines):



- Line for $K = 100$ is **not flexible enough**:
- Line for $K = 1$ is **overly flexible**.

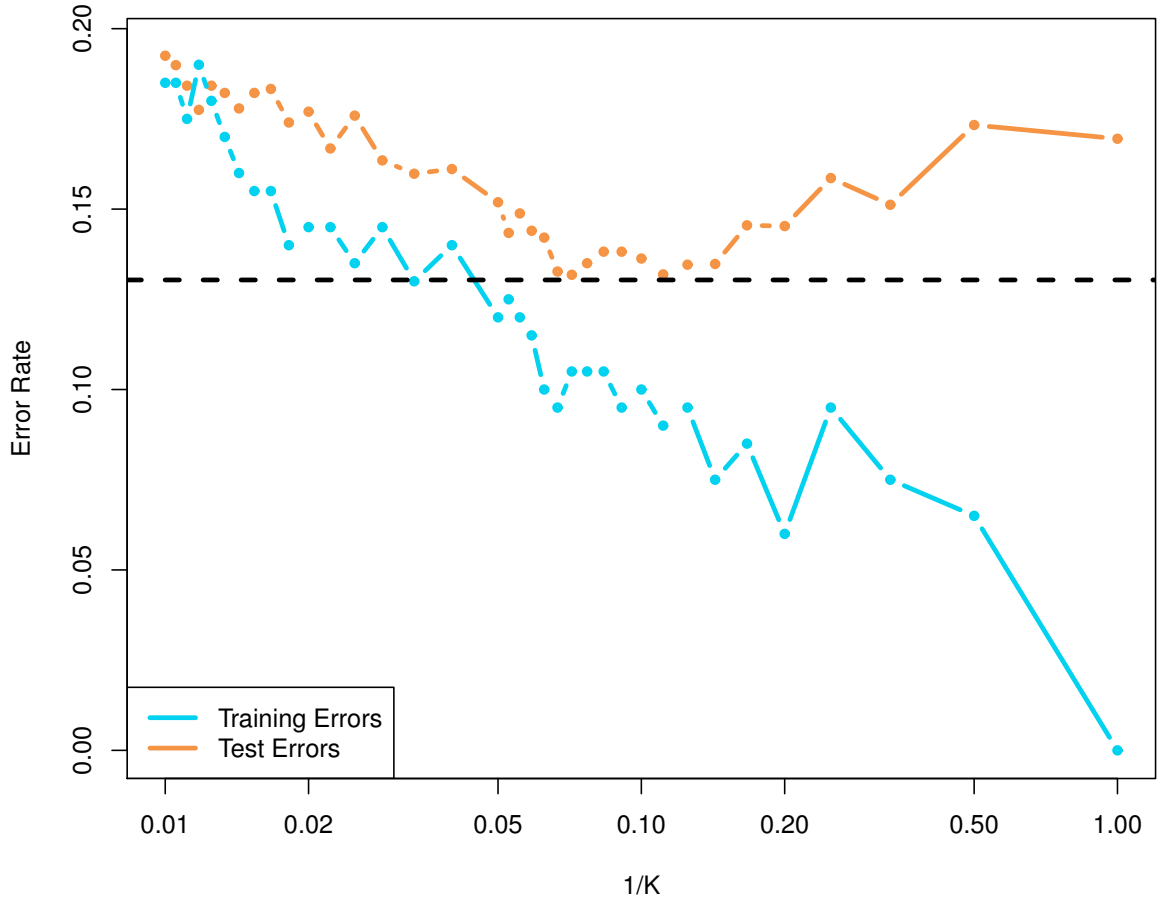
Turns out: another case of *bias-variance trade-off*

- Too *small* $K \rightarrow$ low bias, high variance (overfitting)
- Too *large* $K \rightarrow$ high bias, low variance (underfitting)



Turns out: another case of *bias-variance trade-off*

- Too *small* $K \rightarrow$ low bias, high variance (overfitting)
- Too *large* $K \rightarrow$ high bias, low variance (underfitting)



How to choose K? Surprise, surprise: **cross-validation**!

Pros and cons of KNN

Pros

- Highly flexible — can capture considerable *non-linearity*.
- One single parameter (K) that regulates smoothness — relatively easy to calibrate via cross-validation.
- Simple.

[1] As is the case more generally for non-parametric methods; weak for *inference*.

Pros and cons of KNN

Pros

- Highly flexible — can capture considerable *non-linearity*.
- One single parameter (K) that regulates smoothness — relatively easy to calibrate via cross-validation.
- Simple.

Cons

- Lack of interpretability — no parameters to say why a given observation was classified the way it was.[1]
- Does not *weight* X_i's based on how predictive they are $\rightarrow$ even more sensitive to inclusion of unimportant features.

[1] As is the case more generally for non-parametric methods; weak for *inference*.

Decision trees

Decision trees

Decision trees address some of the limitations of the simple KNN-model.

Decision trees

Decision trees address some of the limitations of the simple KNN-model.

- Also makes predictions based on *nearby observations*—but defines "nearby" in a *sophisticated way* that also facilitates interpretability.

Decision trees

Decision trees address some of the limitations of the simple KNN-model.

- Also makes predictions based on *nearby observations*—but defines "nearby" in a *sophisticated way* that also facilitates interpretability.

Basic idea:

1. Split data space (X) into regions $R_1, R_2, \ldots, R_J$.

Decision trees

Decision trees address some of the limitations of the simple KNN-model.

- Also makes predictions based on *nearby observations*—but defines "nearby" in a *sophisticated way* that also facilitates interpretability.

Basic idea:

1. Split data space (X) into regions $R_1, R_2, \ldots, R_J$.
2. For a *test* observation x_0, we predict its outcome based on the *mean/mode* of the region R_j it belongs.

Decision trees

Decision trees address some of the limitations of the simple KNN-model.

- Also makes predictions based on *nearby observations*—but defines "nearby" in a *sophisticated way* that also facilitates interpretability.

Basic idea:

1. Split data space (X) into regions $R_1, R_2, \ldots, R_J$.
2. For a *test* observation x_0, we predict its outcome based on the *mean/mode* of the region R_j it belongs.

So, how do we split X into regions R_j?

Decision trees

Decision trees address some of the limitations of the simple KNN-model.

- Also makes predictions based on *nearby observations*—but defines "nearby" in a *sophisticated way* that also facilitates interpretability.

Basic idea:

1. Split data space (X) into regions $R_1, R_2, \ldots, R_J$.
2. For a *test* observation x_0, we predict its outcome based on the *mean/mode* of the region R_j it belongs.

So, how do we split X into regions R_j?

- We want to split X s.t. obs. in the same R_j are similar w.r.t. Y.

Decision trees

Decision trees address some of the limitations of the simple KNN-model.

- Also makes predictions based on *nearby observations*—but defines "nearby" in a *sophisticated way* that also facilitates interpretability.

Basic idea:

1. Split data space (X) into regions $R_1, R_2, \ldots, R_J$.
2. For a *test* observation x_0, we predict its outcome based on the *mean/mode* of the region R_j it belongs.

So, how do we split X into regions R_j?

- We want to split X s.t. obs. in the same R_j are similar w.r.t. Y.
- For continuous Y, decision trees seek to *minimize*:

Decision trees

Decision trees address some of the limitations of the simple KNN-model.

- Also makes predictions based on *nearby observations*—but defines "nearby" in a *sophisticated way* that also facilitates interpretability.

Basic idea:

1. Split data space (X) into regions $R_1, R_2, \ldots, R_J$.
2. For a *test* observation x_0, we predict its outcome based on the *mean/mode* of the region R_j it belongs.

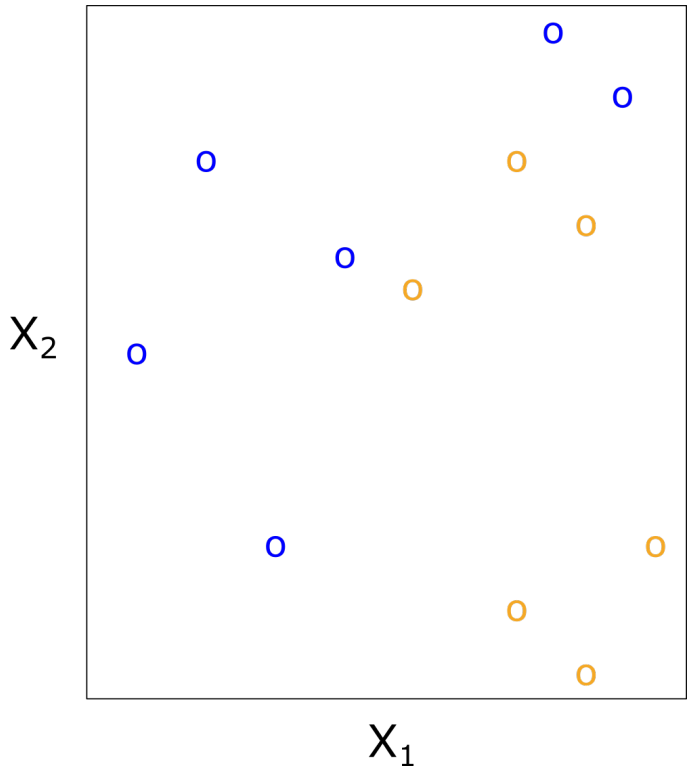
So, how do we split X into regions R_j?

- We want to split X s.t. obs. in the same R_j are similar w.r.t. Y.
- For continuous Y, decision trees seek to *minimize*:

$$\underbrace{\sum_{j}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2}_{\text{Summed error across regions}}$$

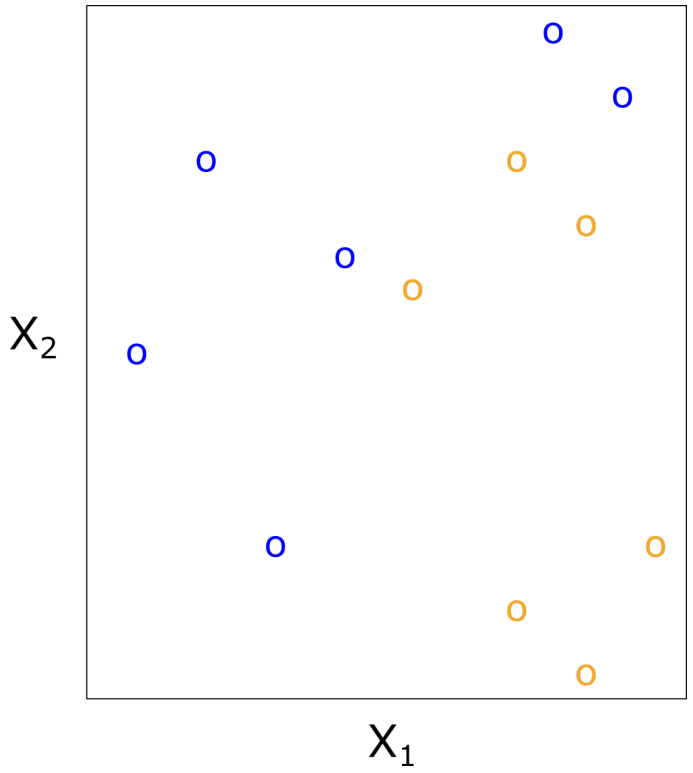
Toy example

Let's reconsider the toy example we used for KNN.



Toy example

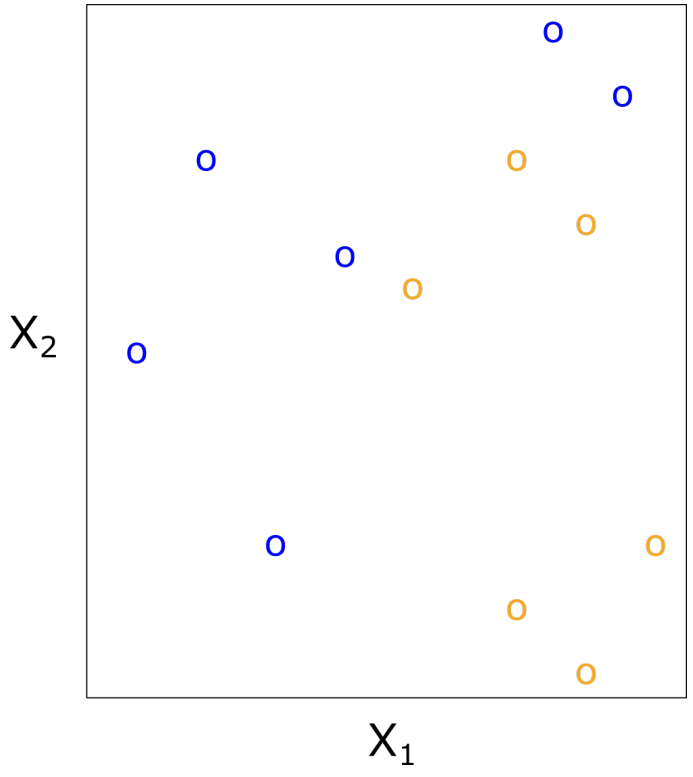
Let's reconsider the toy example we used for KNN.



How can we partition this X_1,X_2 space so that:

Toy example

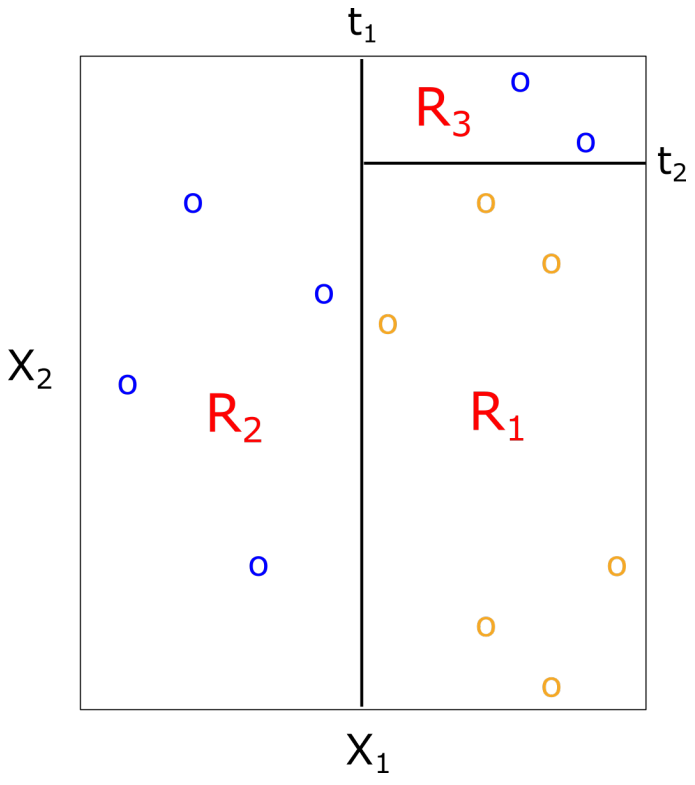
Let's reconsider the toy example we used for KNN.



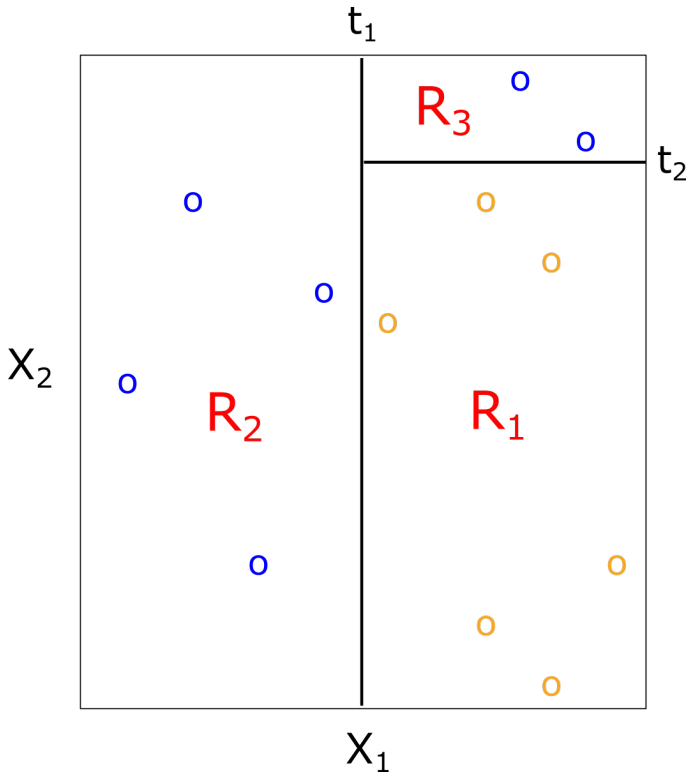
How can we partition this X_1,X_2 space so that:

- We get *homogeneous regions*—i.e., "blue" and "yellow" regions?

Here is a decision tree solution



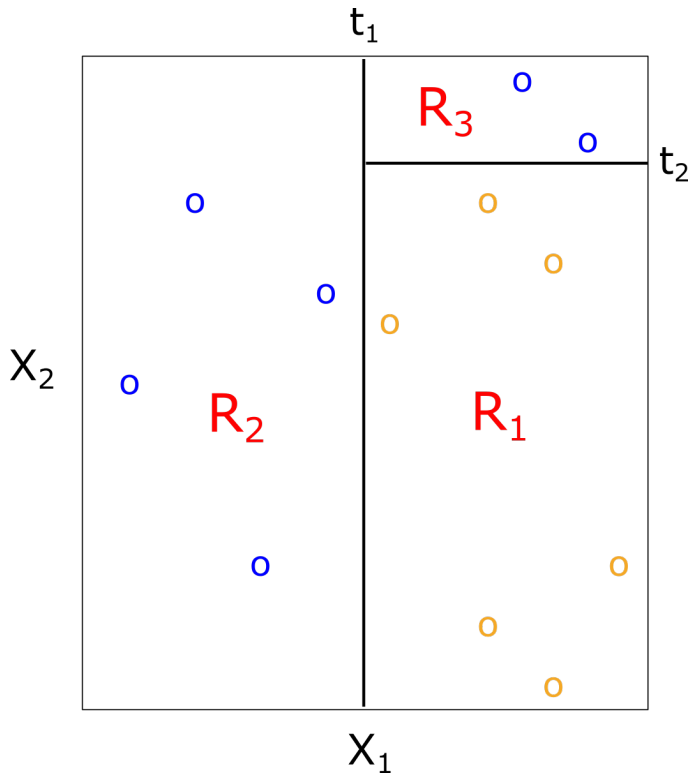
Here is a decision tree solution



The decision tree partitioned X into 3 regions; each highly *homogenous*

- R_1: all orange
- R_2: all blue
- R_3: all blue

Here is a decision tree solution



The decision tree partitioned X into 3 regions; each highly *homogenous*

- R_1: all orange
- R_2: all blue
- R_3: all blue

$\rightarrow$ How did it arrive at this solution?

How did the decision tree arrive at this solution?

Perhaps it evaluated *error* for **all possible splits** and picked the *best one*?

- But no—this is generally *not* computationally feasible.

How did the decision tree arrive at this solution?

Perhaps it evaluated *error* for **all possible splits** and picked the *best one*?

- But no—this is generally *not* computationally feasible.

Instead: **recursive binary splitting**

How did the decision tree arrive at this solution?

Perhaps it evaluated *error* for **all possible splits** and picked the *best one*?

- But no—this is generally *not* computationally feasible.

Instead: **recursive binary splitting**

- Start with all observations in a *single region*.

How did the decision tree arrive at this solution?

Perhaps it evaluated *error* for **all possible splits** and picked the *best one*?

- But no—this is generally *not* computationally feasible.

Instead: **recursive binary splitting**

- Start with all observations in a *single region*.
- Then *successively* split X; each time dividing a region R_j into *two*.

How did the decision tree arrive at this solution?

Perhaps it evaluated *error* for **all possible splits** and picked the *best one*?

- But no—this is generally *not* computationally feasible.

Instead: **recursive binary splitting**

- Start with all observations in a *single region*.
- Then *successively* split X; each time dividing a region R_j into *two*.
- At every step, seeking to find the 'split' that reduces error the most.

How did the decision tree arrive at this solution?

Perhaps it evaluated *error* for **all possible splits** and picked the *best one*?

- But no—this is generally *not* computationally feasible.

Instead: recursive binary splitting

- Start with all observations in a *single region*.
- Then *successively* split X; each time dividing a region R_j into *two*.
- At every step, seeking to find the 'split' that reduces error the most.
  - Split: a *variable* j and a *value* s.

How did the decision tree arrive at this solution?

Perhaps it evaluated *error* for **all possible splits** and picked the *best one*?

- But no—this is generally *not* computationally feasible.

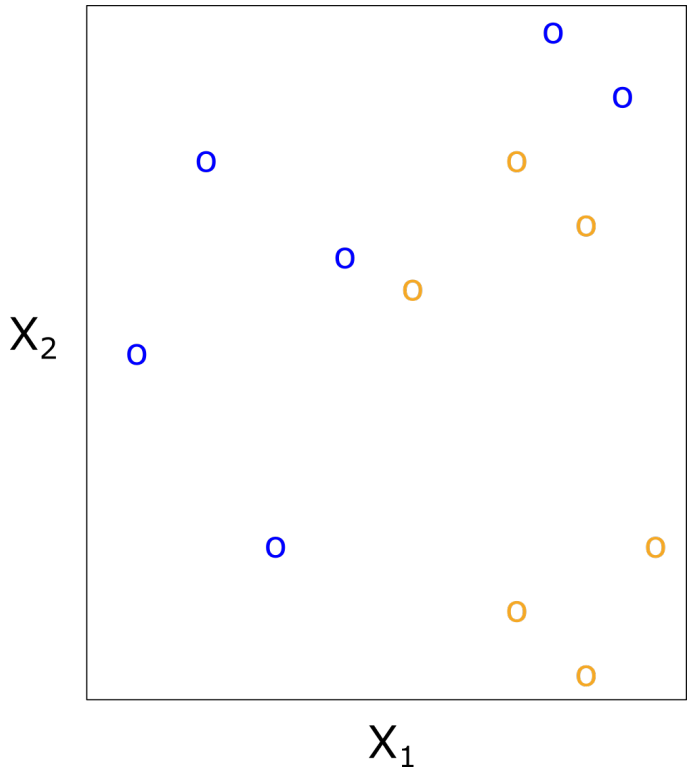
Instead: recursive binary splitting

- Start with all observations in a *single region*.
- Then *successively* split X; each time dividing a region R_j into *two*.
- At every step, seeking to find the 'split' that reduces error the most.
  - Split: a *variable* j and a *value* s.

Let's return to our toy example!

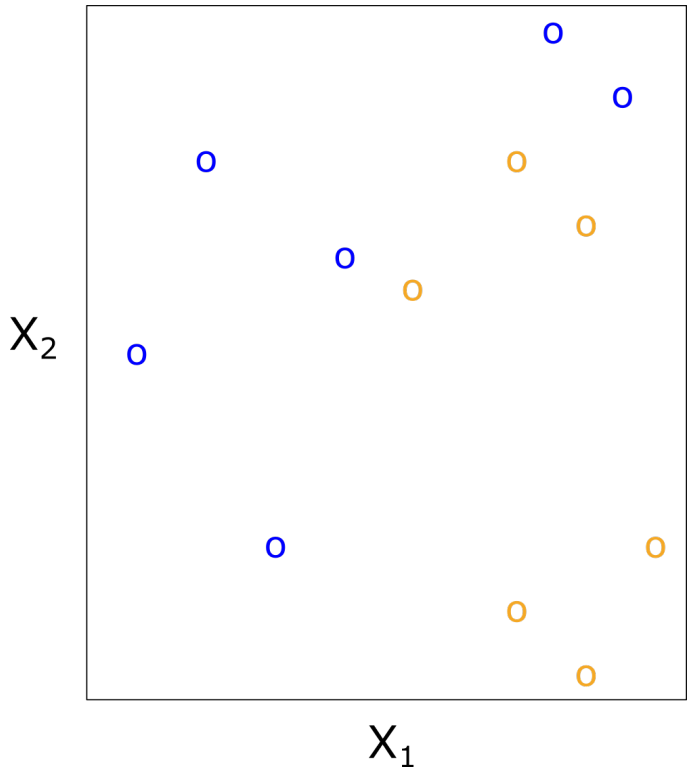
Recursive binary splitting of toy example

So, we begin with all observations belonging to a single region.



Recursive binary splitting of toy example

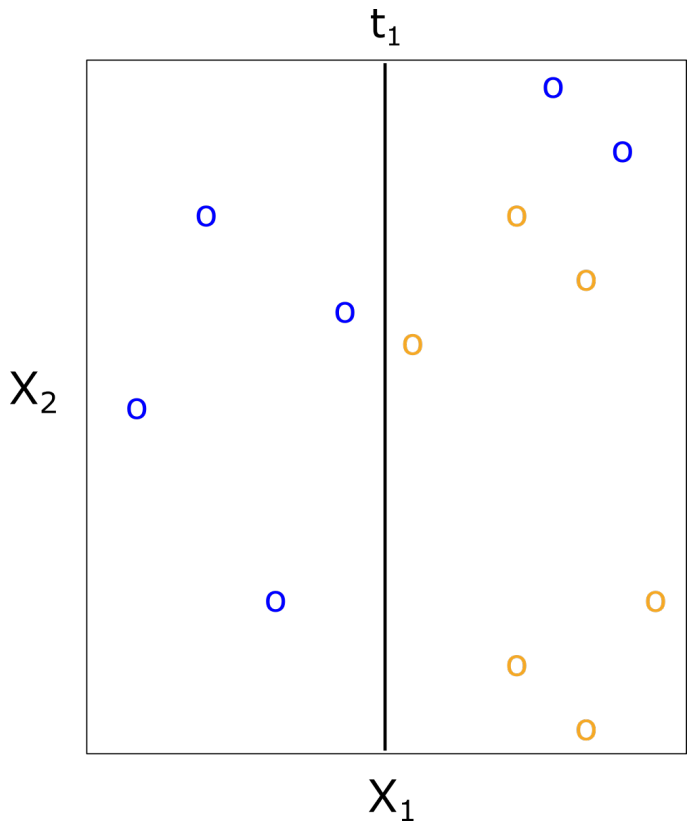
So, we begin with all observations belonging to a single region.



Then we ask—how can we insert a **vertical** (splitting on X_1) or **horizontal** (splitting on X_2) line such that the resulting two regions becomes *maximally more homogeneous*?

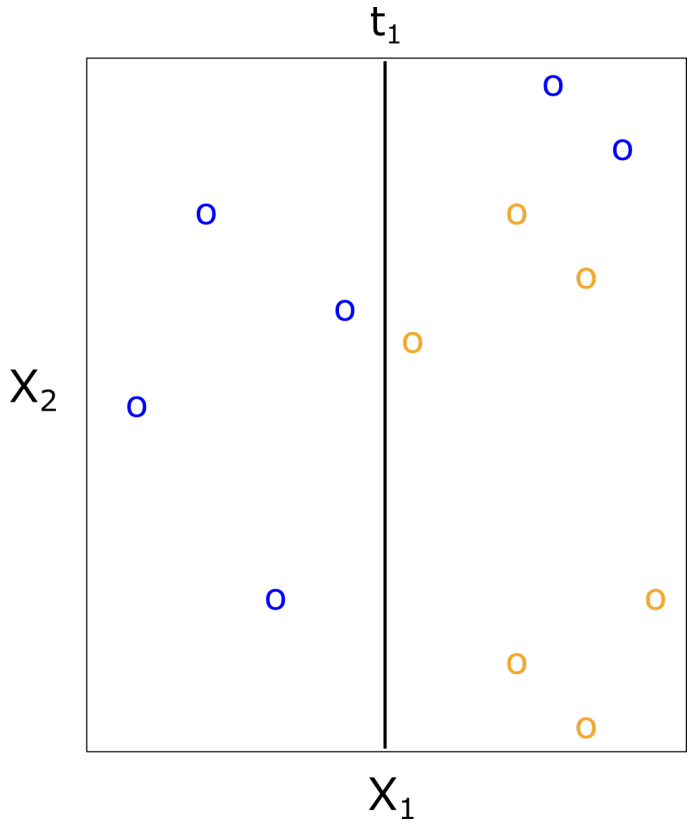
Split no. 1

In this example, it turns out that creating a split at *value* t_1 on *variable* X_1 is the split which results in the *biggest error reduction*.



Split no. 1

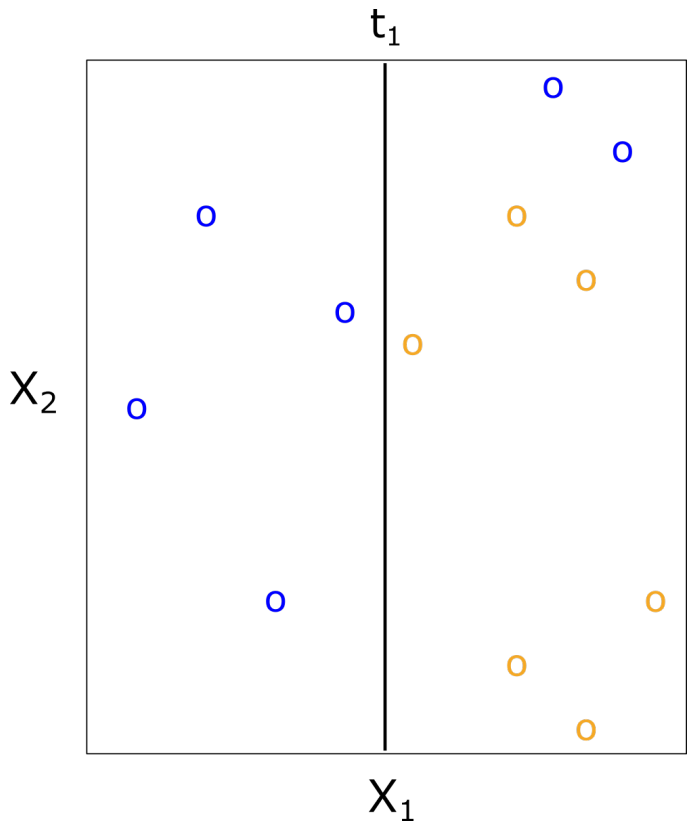
In this example, it turns out that creating a split at *value* t_1 on *variable* X_1 is the split which results in the *biggest error reduction*.



We see that the region to the left now is fully homogeneous (all blue).

Split no. 1

In this example, it turns out that creating a split at *value* t_1 on *variable* X_1 is the split which results in the *biggest error reduction*.

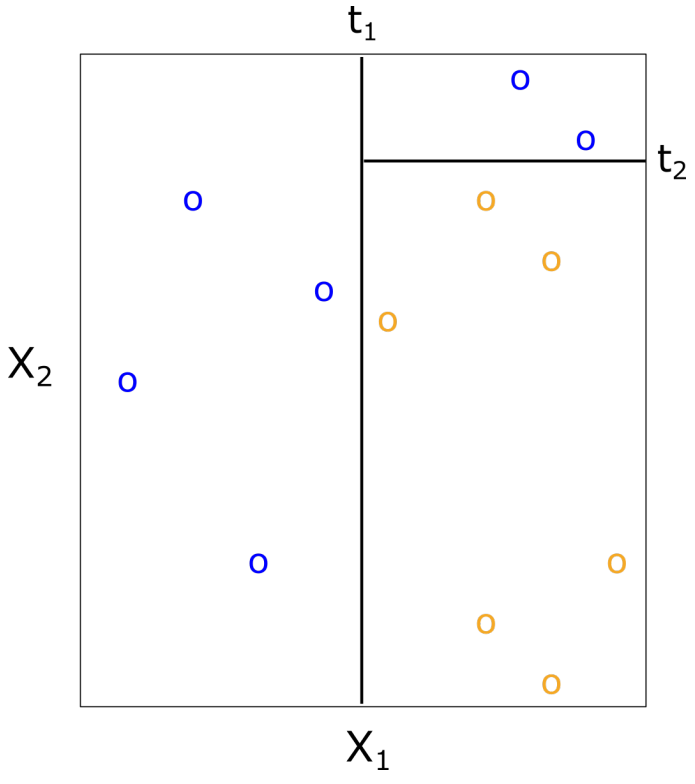


We see that the region to the left now is fully homogeneous (all blue).

Q: Can we add additional splits that further reduces the error?

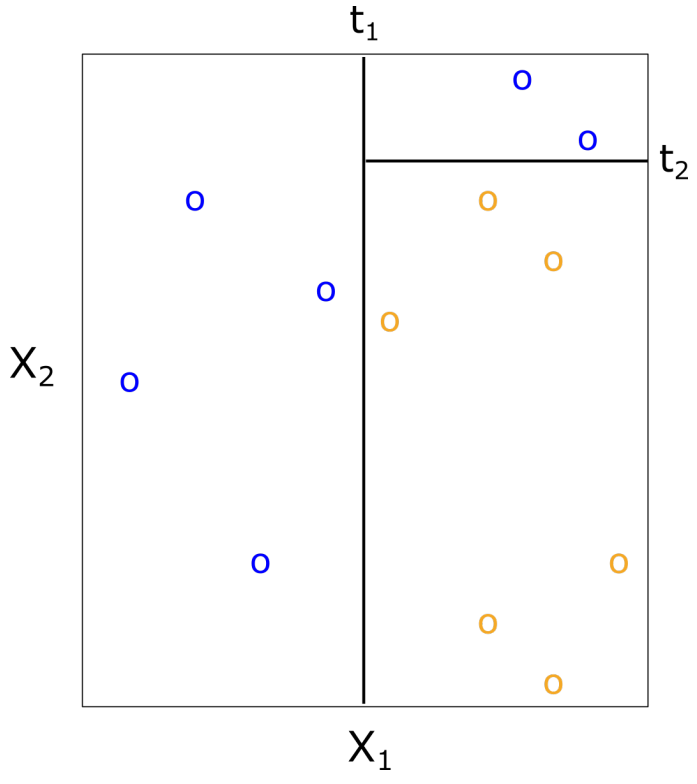
Split no. 2

Yes! Splitting the "right-region" at value t_2 on variable X_2 reduces error further.



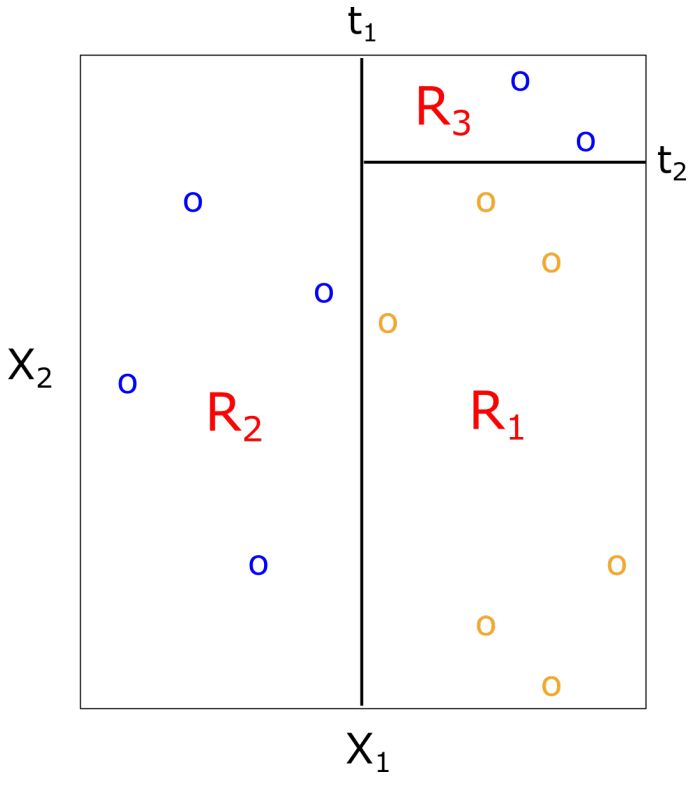
Split no. 2

Yes! Splitting the "right-region" at value t_2 on variable X_2 reduces error further.

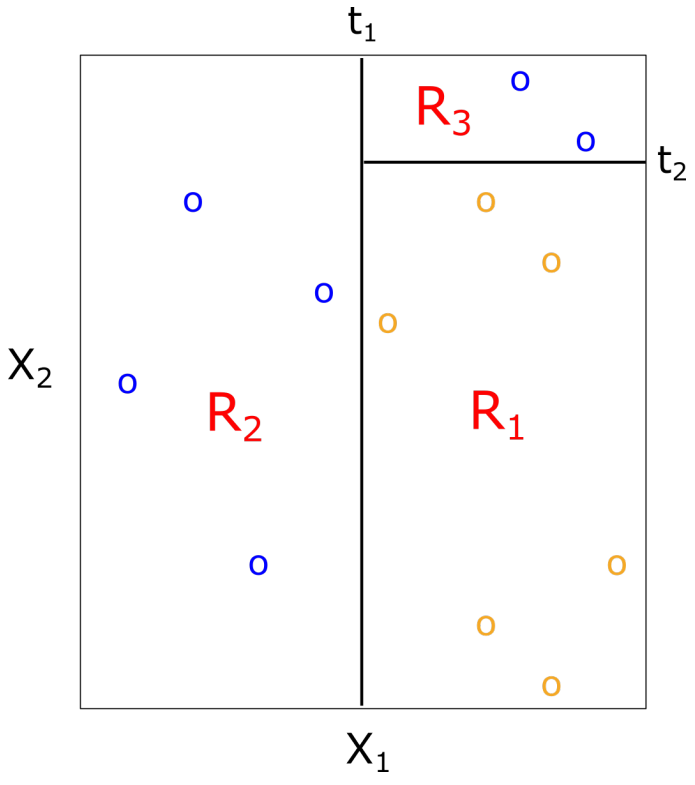


After this, no further improvements are possible, and we retain three regions.

Three-region solution

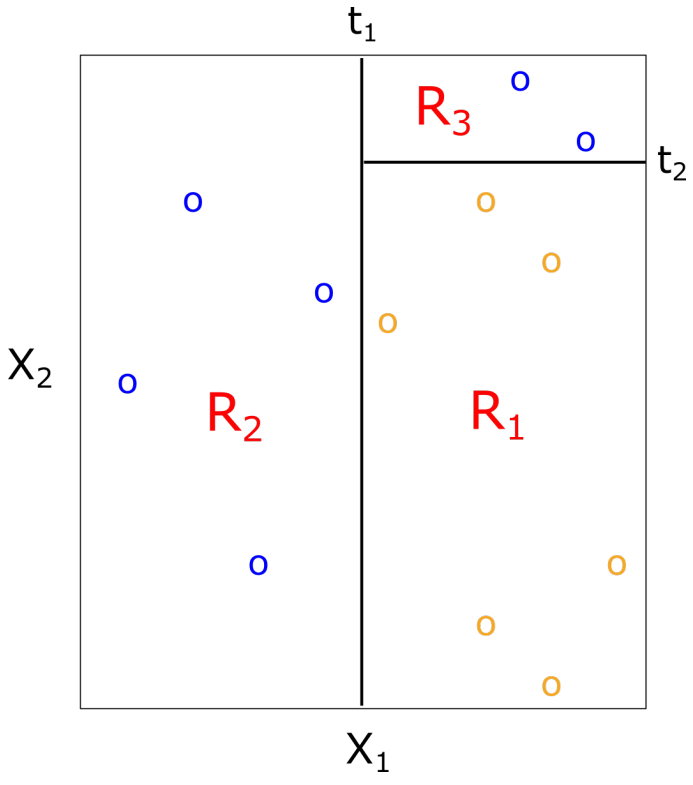


Three-region solution



With these regions identified—how do we use them to make **predictions**?

Three-region solution

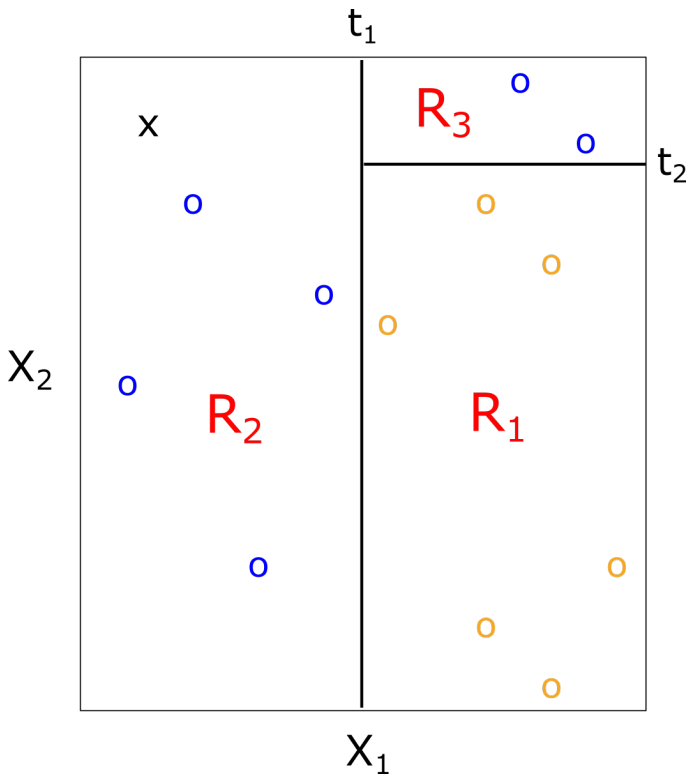


With these regions identified—how do we use them to make **predictions**? Again:

- *Categorical Y*: *majority vote* with each region.
- *Continuous Y*: *mean* within each region.

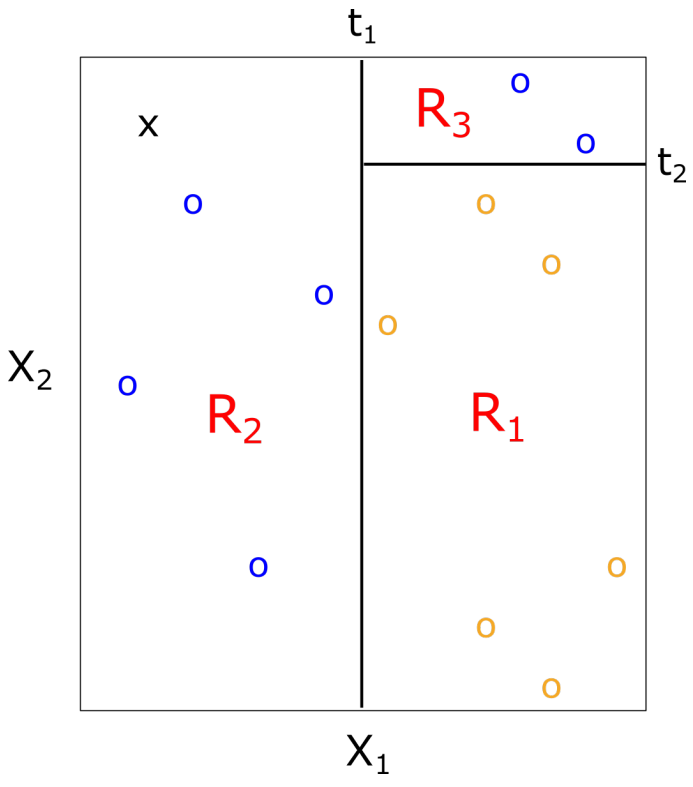
Example prediction

What would we predict for this *test* observation "x"?



Example prediction

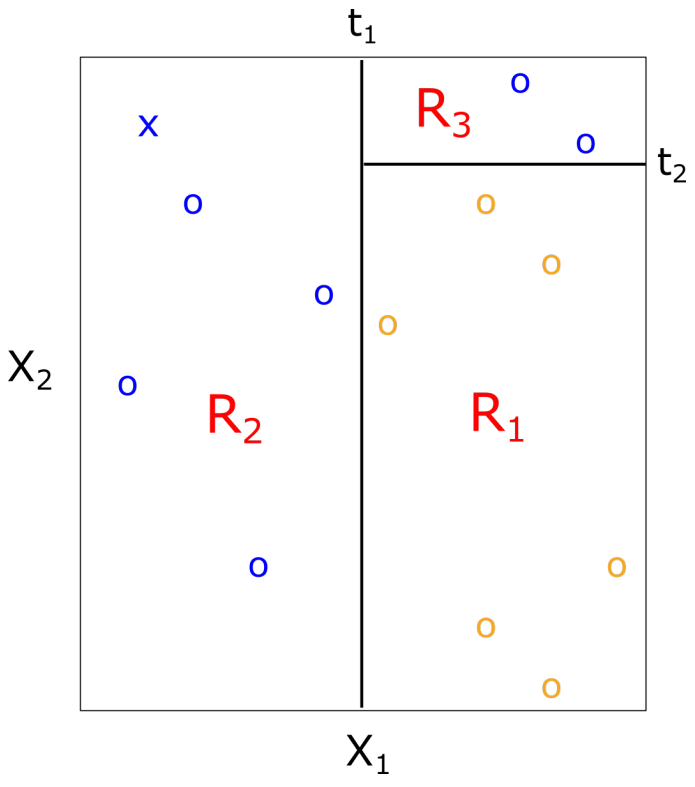
What would we predict for this *test* observation "x"?



Because a *majority* (4/4) of *training obs.* in R_2 are blue $\rightarrow$ predict $\hat{y}_x =$ blue.

Example prediction

What would we predict for this *test* observation "x"?



Because a *majority* (4/4) of *training obs.* in R_2 are blue $\rightarrow$ predict $\hat{y}_x =$ blue.

Why do we call this a "decision tree"?

Because the **splitting rules**—that partitions X into regions—can be effectively **represented in a "tree form"**.

Why do we call this a "decision tree"?

Because the **splitting rules**—that partitions X into regions—can be effectively **represented in a "tree form"**.

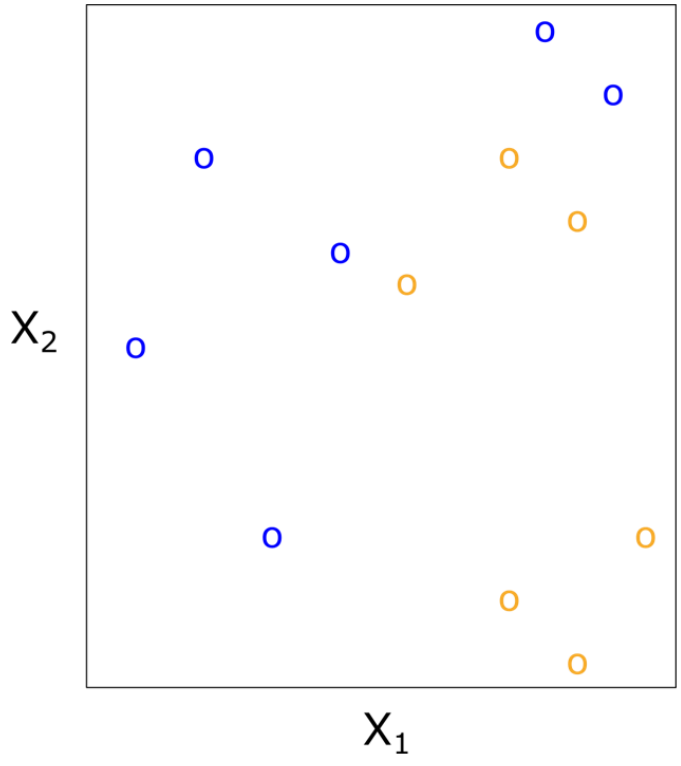
To see this, let's go step-by-step.

Why do we call this a "decision tree"?

Because the **splitting rules**—that partitions X into regions—can be effectively **represented in a "tree form"**.

To see this, let's go step-by-step.

All observations

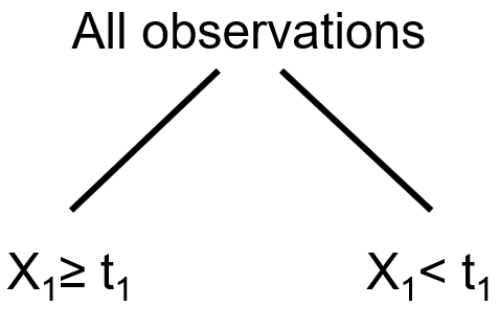


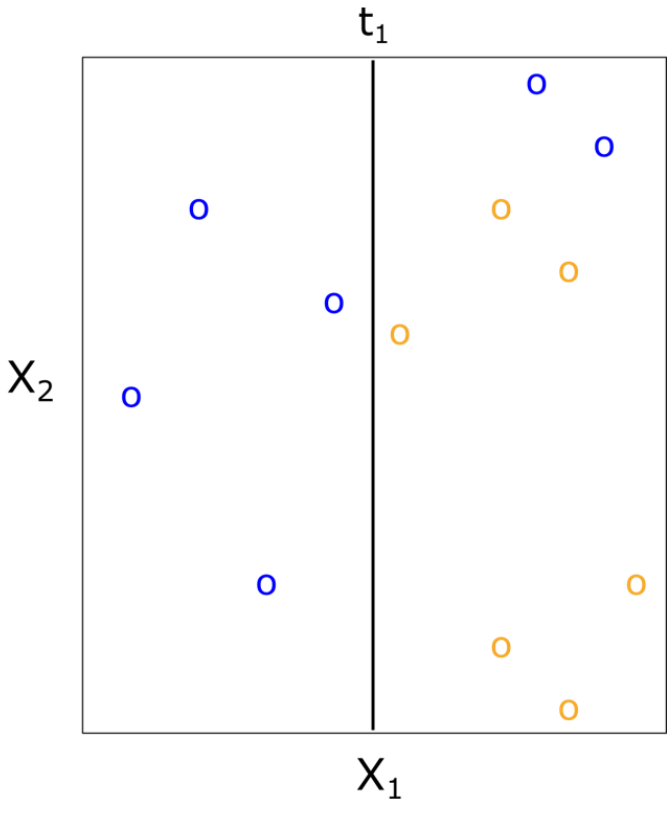
Why do we call this a "decision tree"?

Because the **splitting rules**—that partitions X into regions—can be effectively **represented in a "tree form"**.

To see this, let's go step-by-step.

Saved to this PC

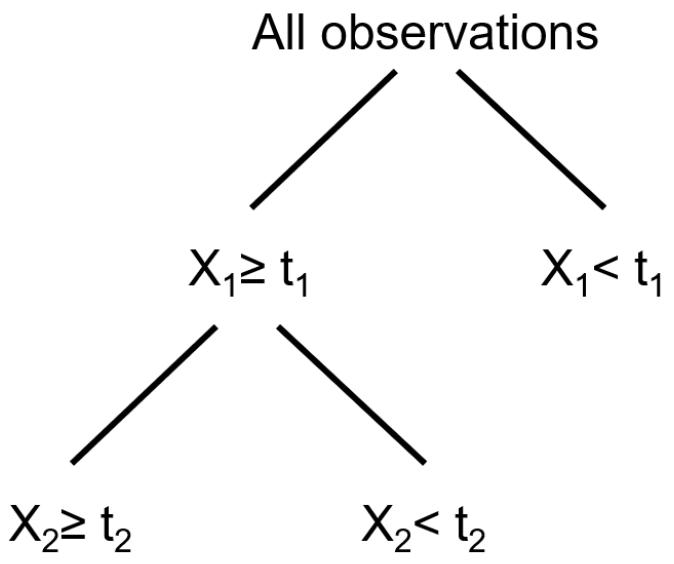


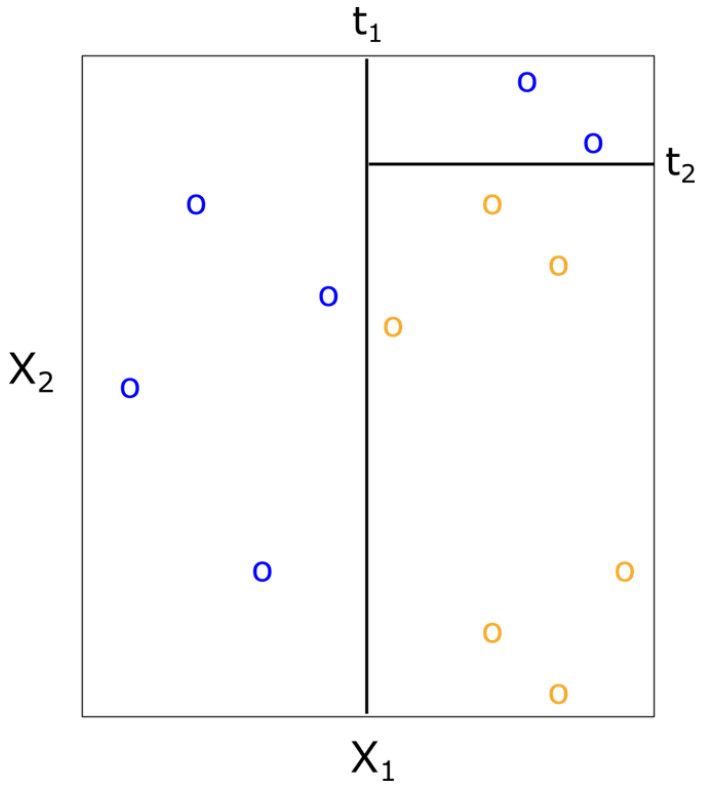


Why do we call this a "decision tree"?

Because the **splitting rules**—that partitions X into regions—can be effectively **represented in a "tree form"**.

To see this, let's go step-by-step.

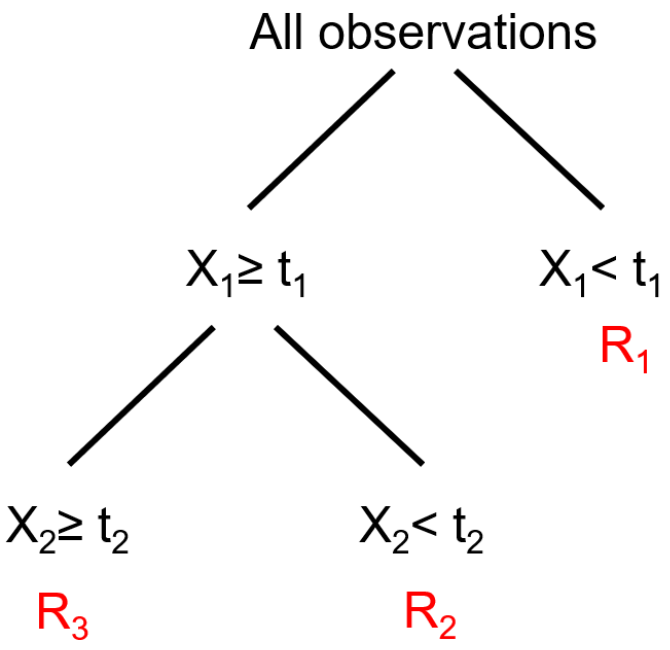


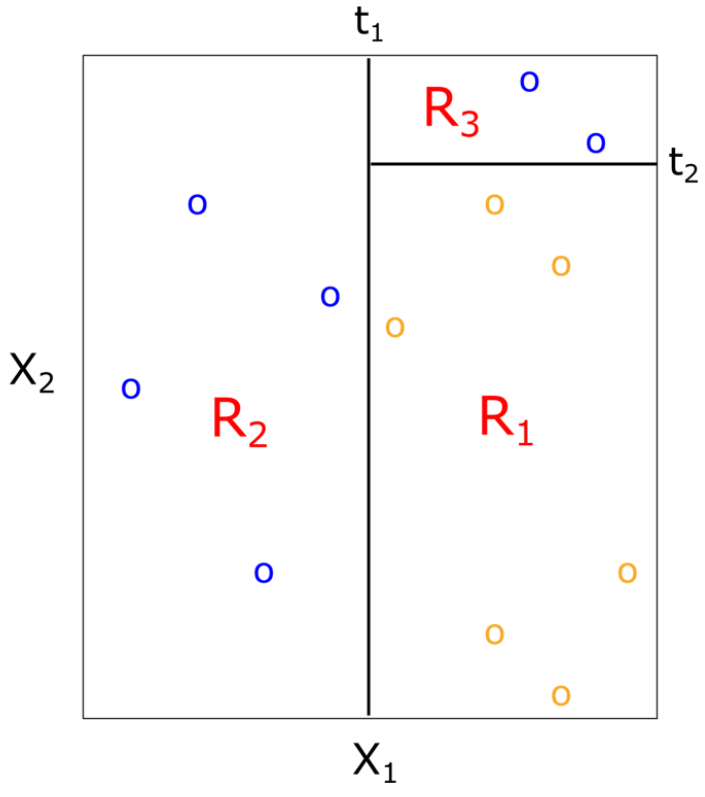


Why do we call this a "decision tree"?

Because the **splitting rules**—that partitions X into regions—can be effectively **represented in a "tree form"**.

To see this, let's go step-by-step.

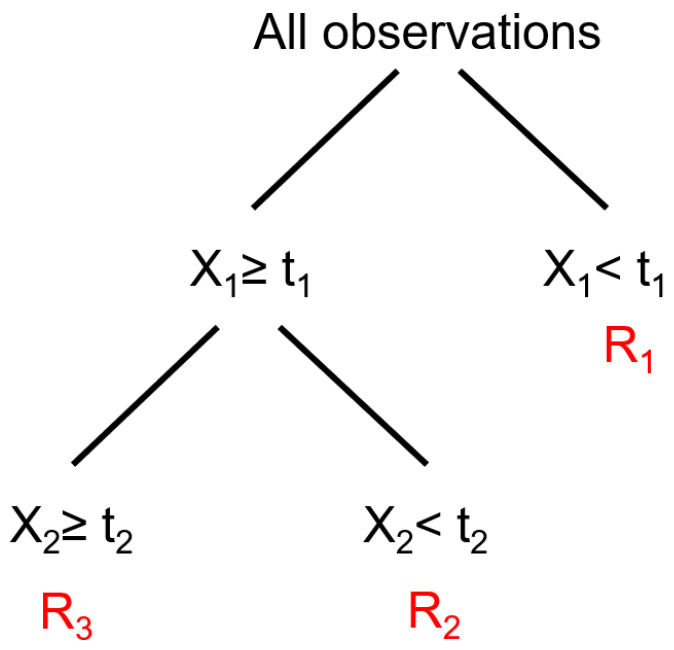


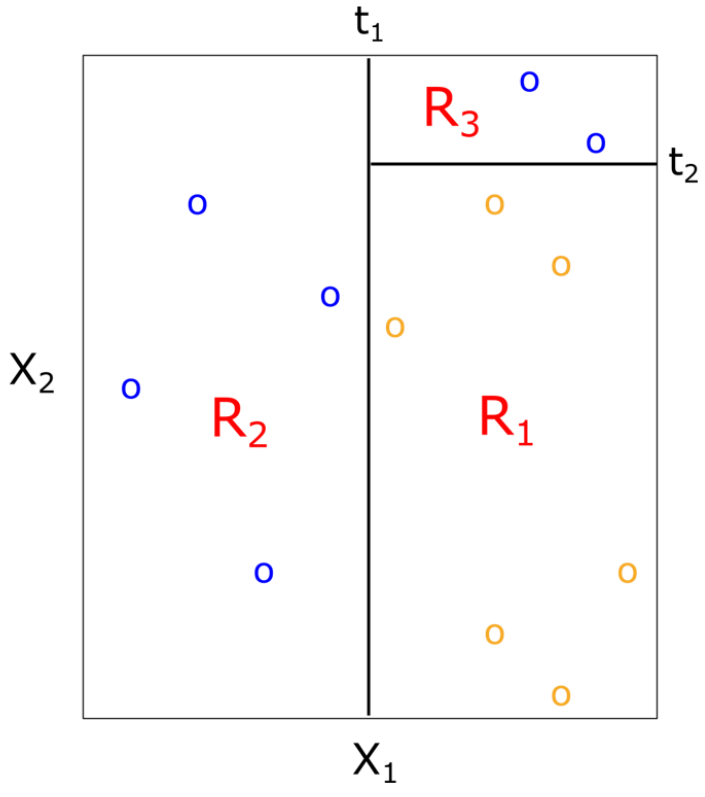


Why do we call this a "decision tree"?

Because the **splitting rules**—that partitions X into regions—can be effectively **represented in a "tree form"**.

To see this, let's go step-by-step.





This is neat—our predictions become very interpretable!

Interpretation of toy example

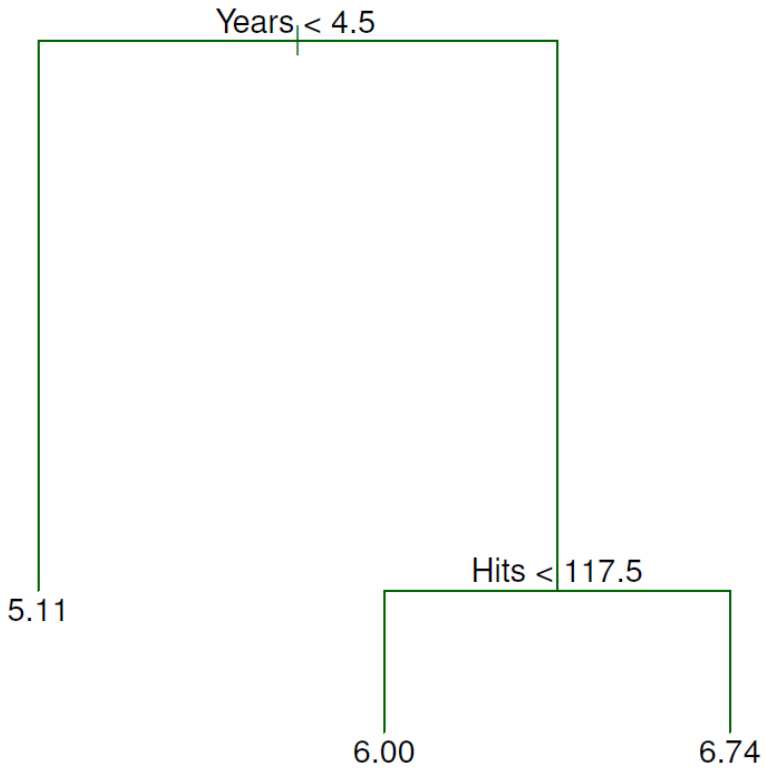
- Small $X_1 \rightarrow$ blue
- Large X_1 & large $X_2 \rightarrow$ blue
- Large X_1 & small $X_2 \rightarrow$ orange

Interpretation of toy example

- Small $X_1 \rightarrow$ blue
- Large X_1 & large $X_2 \rightarrow$ blue
- Large X_1 & small $X_2 \rightarrow$ orange

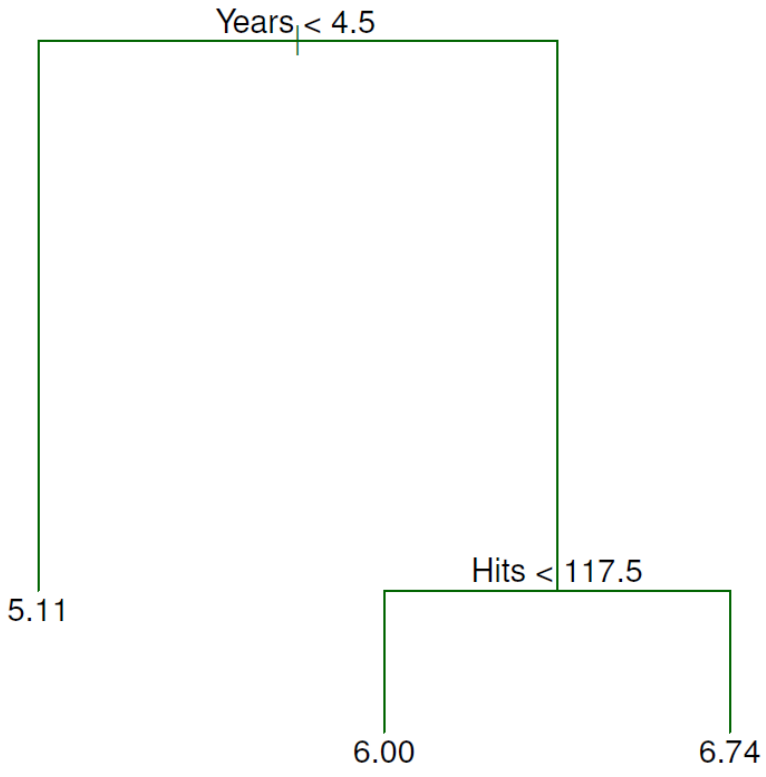
Here we see something else which is neat — the *automatic discovery* of important *non-linearities* & *interactions*!

ISL example



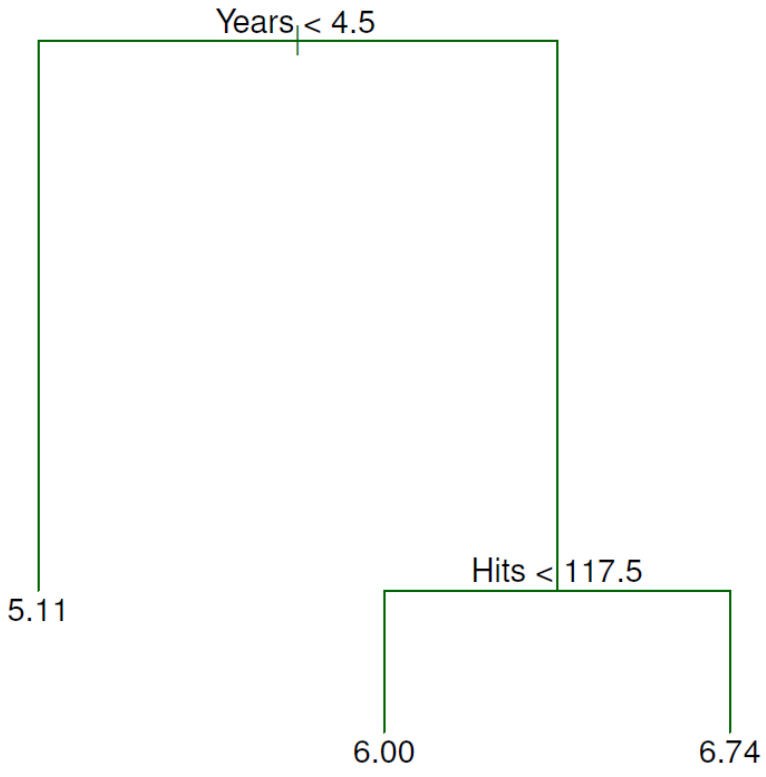
- A simple *regression tree* to predict baseball players' (log) salary, using two variables, *Years* and *Hits*.

ISL example



- A simple *regression tree* to predict baseball players' (log) salary, using two variables, *Years* and *Hits*.
- For a player with less than 4.5 (Years) experience, the predicted log(Salary) is 5.11, i.e., $1000 * e^{5.11} = \$165,174$.

ISL example



- A simple *regression tree* to predict baseball players' (log) salary, using two variables, *Years* and *Hits*.
- For a player with less than 4.5 (Years) experience, the predicted log(Salary) is 5.11, i.e., $1000 * e^{5.11} = \$165,174$.
- For a player with Years ≥ 4.5 and Hits < 117.5, the prediction is $1000 * e^{6} = \$402,834$.

Natural Q—how *deep* to grow the tree?

As we grow the **tree** *larger & larger* $\Rightarrow$ **error** becomes *smaller & smaller*.

- Does this suggest that we should grow the tree *very large*?

Natural Q—how *deep* to grow the tree?

As we grow the **tree** *larger & larger* $\Rightarrow$ **error** becomes *smaller & smaller*.

- Does this suggest that we should grow the tree *very large*?

No, not as simple as that:

- *Linear models* with too many parameters becomes too flexible & captures noise in training data $\rightarrow$ *overfitting*

Natural Q—how *deep* to grow the tree?

As we grow the **tree** *larger & larger* ⇒ **error** becomes *smaller & smaller*.

- Does this suggest that we should grow the tree *very large*?

No, not as simple as that:

- *Linear models* with too many parameters becomes too flexible & captures noise in training data → *overfitting*
- *Decision trees* with too many splits/leaves also becomes too flexible; it then splits based on noise in the training data → *overfitting*

Natural Q—how *deep* to grow the tree?

As we grow the **tree** *larger & larger* $\Rightarrow$ **error** becomes *smaller & smaller*.

- Does this suggest that we should grow the tree *very large*?

No, not as simple as that:

- *Linear models* with too many parameters becomes too flexible & captures noise in training data $\rightarrow$ *overfitting*
- *Decision trees* with too many splits/leaves also becomes too flexible; it then splits based on noise in the training data $\rightarrow$ *overfitting*

In the *linear model* case, we addressed this with a combination of:

- *Penalization*
- *Cross-validation*.

Natural Q—how *deep* to grow the tree?

As we grow the **tree** *larger & larger* ⇒ **error** becomes *smaller & smaller*.

- Does this suggest that we should grow the tree *very large*?

No, not as simple as that:

- *Linear models* with too many parameters becomes too flexible & captures noise in training data → *overfitting*
- *Decision trees* with too many splits/leaves also becomes too flexible; it then splits based on noise in the training data → *overfitting*

In the *linear model* case, we addressed this with a combination of:

- *Penalization*
- *Cross-validation*.

These ideas/concepts reemerge here as well, as we'll see.

For trees, we use something called "tree pruning"

Basic idea:

1. Use *recursive binary splitting* to grow a **large tree** T_0—i.e., as on previous slides.

For trees, we use something called "tree pruning"

Basic idea:

1. Use *recursive binary splitting* to grow a **large tree** T_0—i.e., as on previous slides.
2. Re-evaluate splits *furthest down* in the tree, accounting also for a **penalty** proportional to the *size of the tree*: $|T_0|$.

For trees, we use something called "tree pruning"

Basic idea:

1. Use *recursive binary splitting* to grow a **large tree** T_0—i.e., as on previous slides.
2. Re-evaluate splits *furthest down* in the tree, accounting also for a **penalty** proportional to the *size of the tree*: $|T_0|$.
   - If the penalty $>$ error reduction $\rightarrow$ undo split.

More detail: the penalty

When **constructing** the *initial* (large) tree, we want to minimize:

$$\underbrace{\sum_{j}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2}_{\text{Summed error across leaves}}$$

More detail: the penalty

When **constructing** the *initial* (large) tree, we want to minimize:

$$\underbrace{\sum_{j}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2}_{\text{Summed error across leaves}}$$

When **pruning** this tree, we want to minimize:

$$\underbrace{\sum_{j}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2}_{\text{Summed error across leaves}} + \underbrace{\alpha * |T_0|}_{\textit{Penalty}}$$

More detail: the penalty

When **constructing** the *initial* (large) tree, we want to minimize:

$$\underbrace{\sum_{j}^{J}\sum_{i\in R_j}(y_i - \bar{y}_{R_j})^2}_{\text{Summed error across leaves}}$$

When **pruning** this tree, we want to minimize:

$$\underbrace{\sum_{j}^{J}\sum_{i\in R_j}(y_i - \bar{y}_{R_j})^2}_{\text{Summed error across leaves}} + \underbrace{\alpha * |T_0|}_{\textit{Penalty}}$$

Two things:

- Note—this is very similar to **lasso/ridge**!

More detail: the penalty

When **constructing** the *initial* (large) tree, we want to minimize:

$$\underbrace{\sum_{j}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2}_{\text{Summed error across leaves}}$$

When **pruning** this tree, we want to minimize:

$$\underbrace{\sum_{j}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2}_{\text{Summed error across leaves}} + \underbrace{\alpha * |T_0|}_{\textit{Penalty}}$$

Two things:

- Note—this is very similar to **lasso/ridge**!
- Q—How do we set α?? Surprise, surprise: **cross-validation**!

Illustration of pruning (toy example)

Suppose this is our *fully grown tree* T_0:

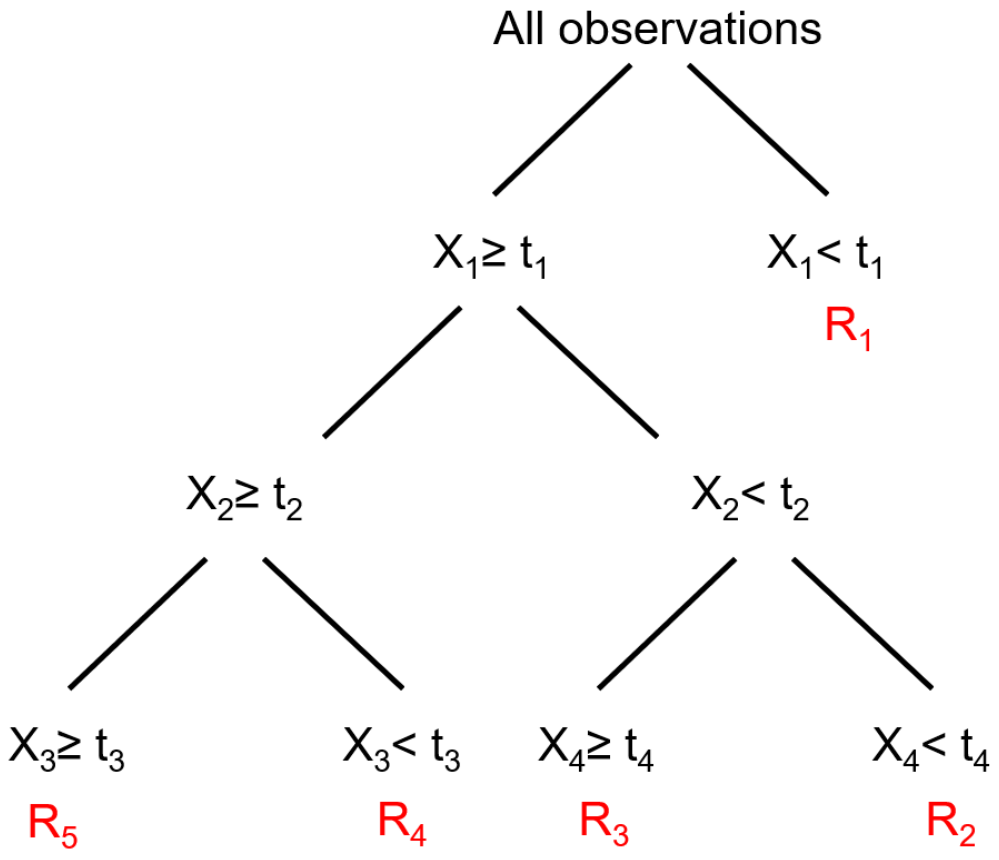
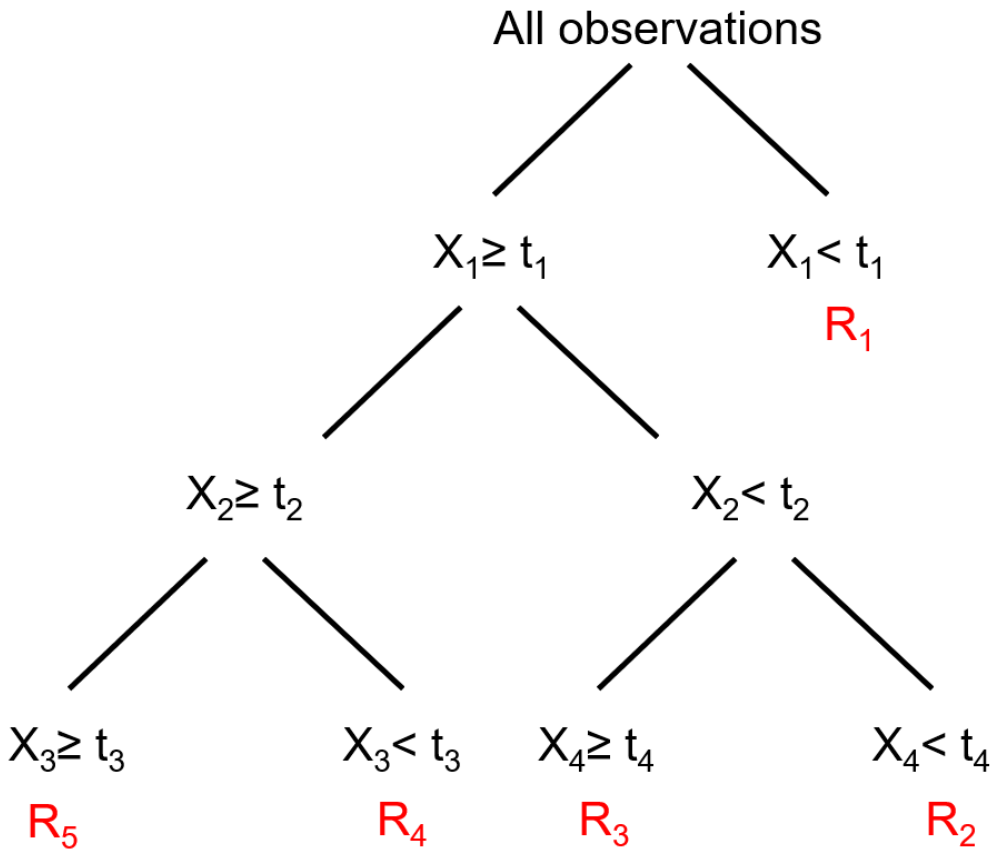


Illustration of pruning (toy example)

Suppose this is our *fully grown tree* T_0:



Then—to prune—we *reconsider* the last split.

Illustration of pruning (toy example)

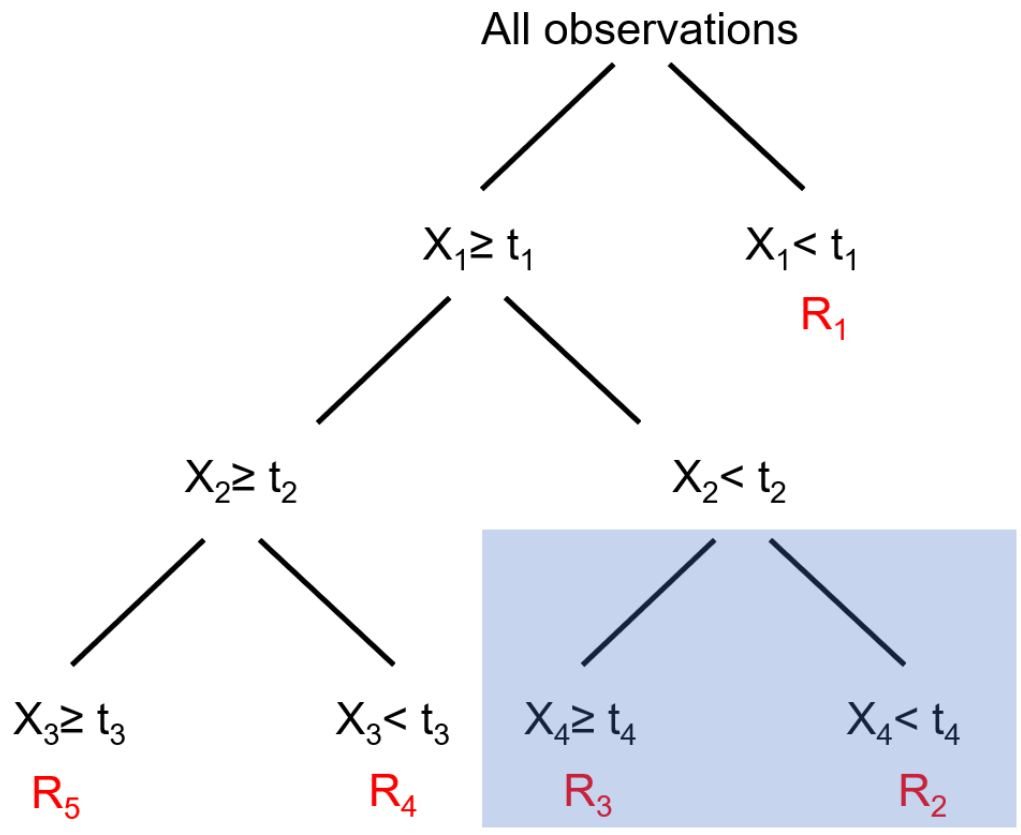
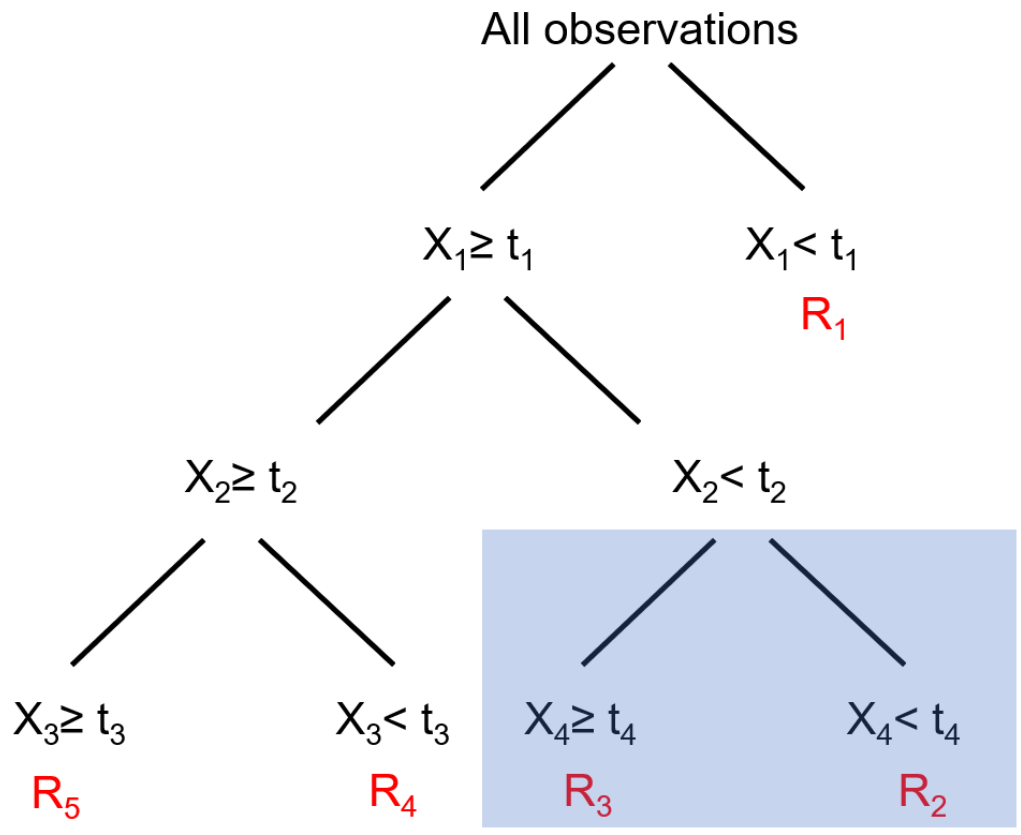
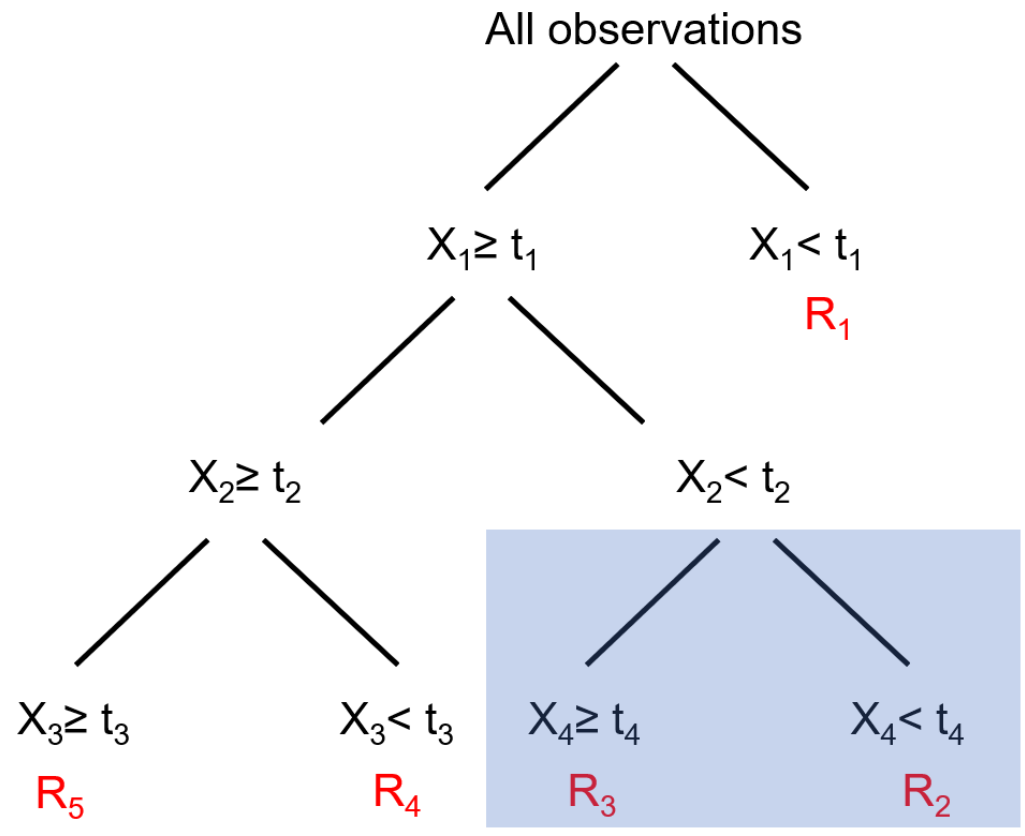


Illustration of pruning (toy example)



If the *error reduction* from splitting on X_4 is $< 2 \times \alpha$, then prune!

Illustration of pruning (toy example)



If the *error reduction* from splitting on X_4 is $< 2 \times \alpha$, then prune!

- It is ($\times 2$) because we add *two extra nodes* in the split.

Illustration of pruning (toy example)

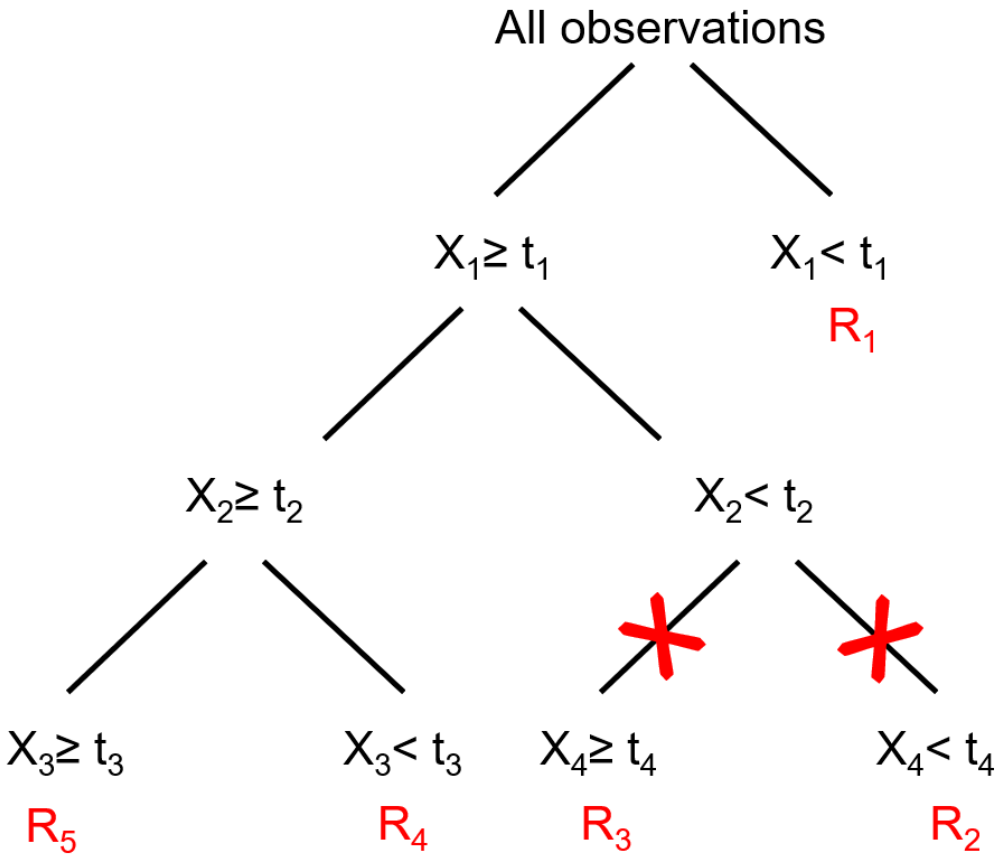


Illustration of pruning (toy example)

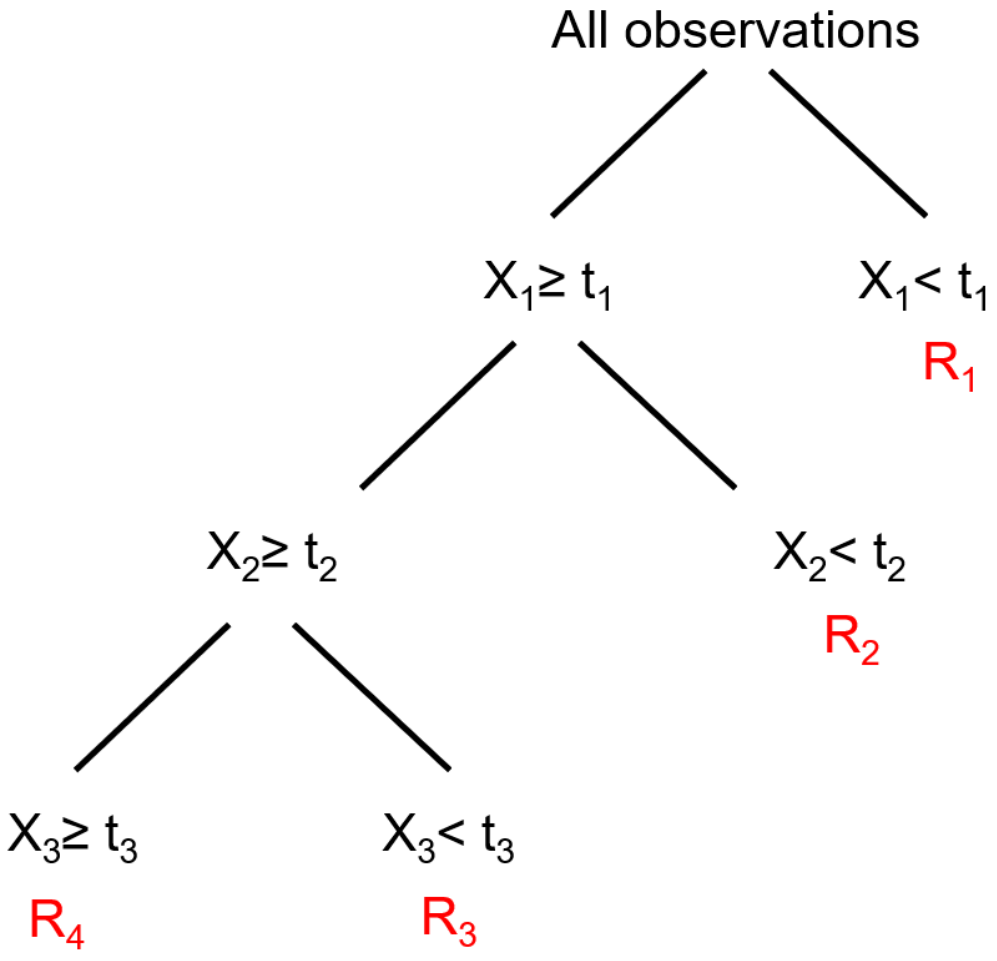
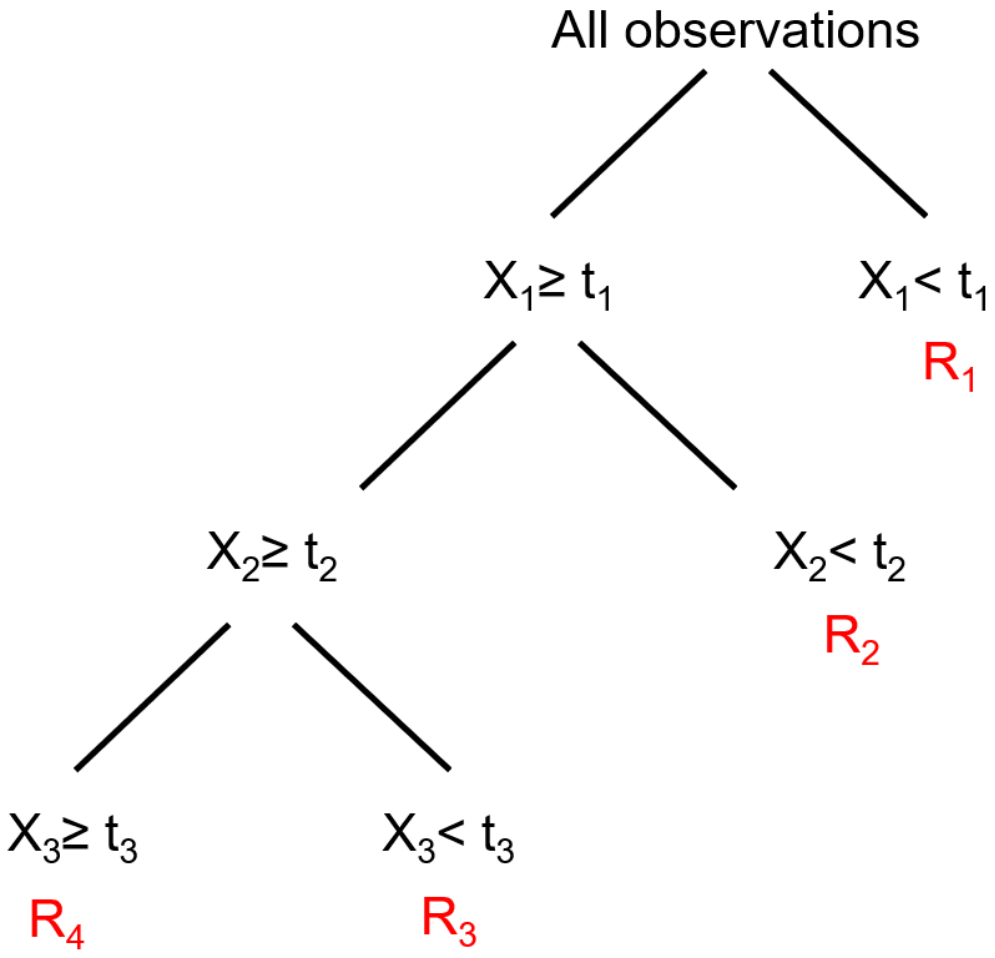


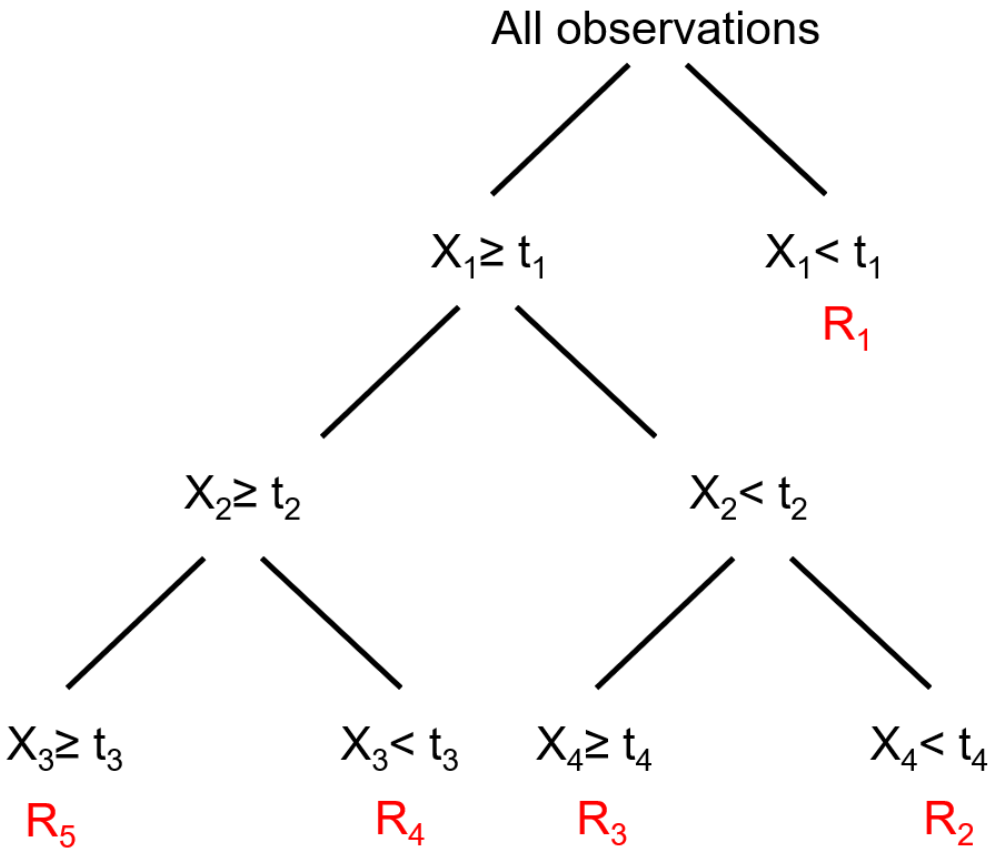
Illustration of pruning (toy example)



Now that we have some intuition for how *pruning* works—what is the effect of α?

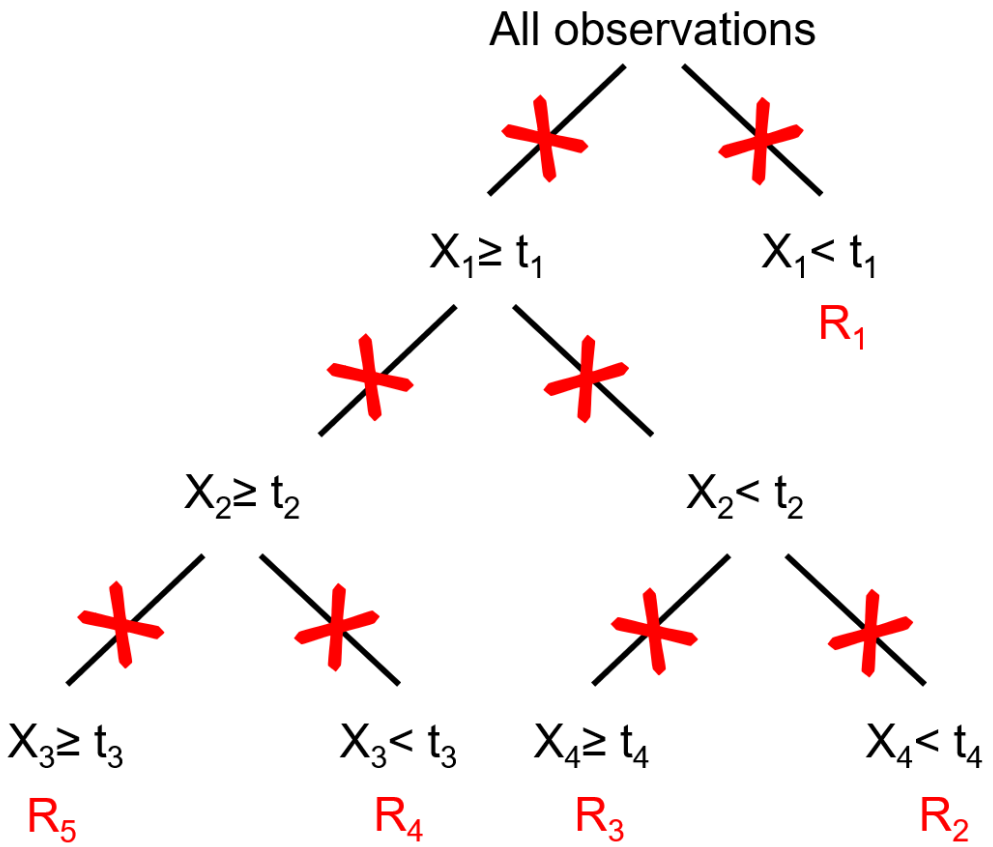
Effect of α? Extreme #1: $\alpha = 0$

Similar to lasso/ridge, when $\alpha = 0$, we retain the *full tree* (T_0).



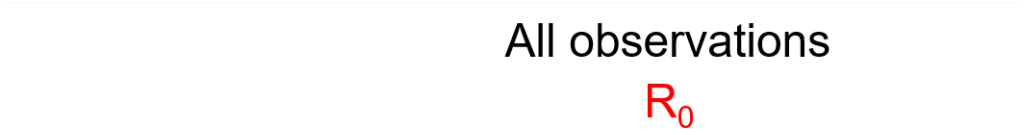
Effect of α? Extreme #2: $\alpha = \infty$

Also similar to lasso/ridge, when $\alpha = \infty$, we retain the *NULL* model.

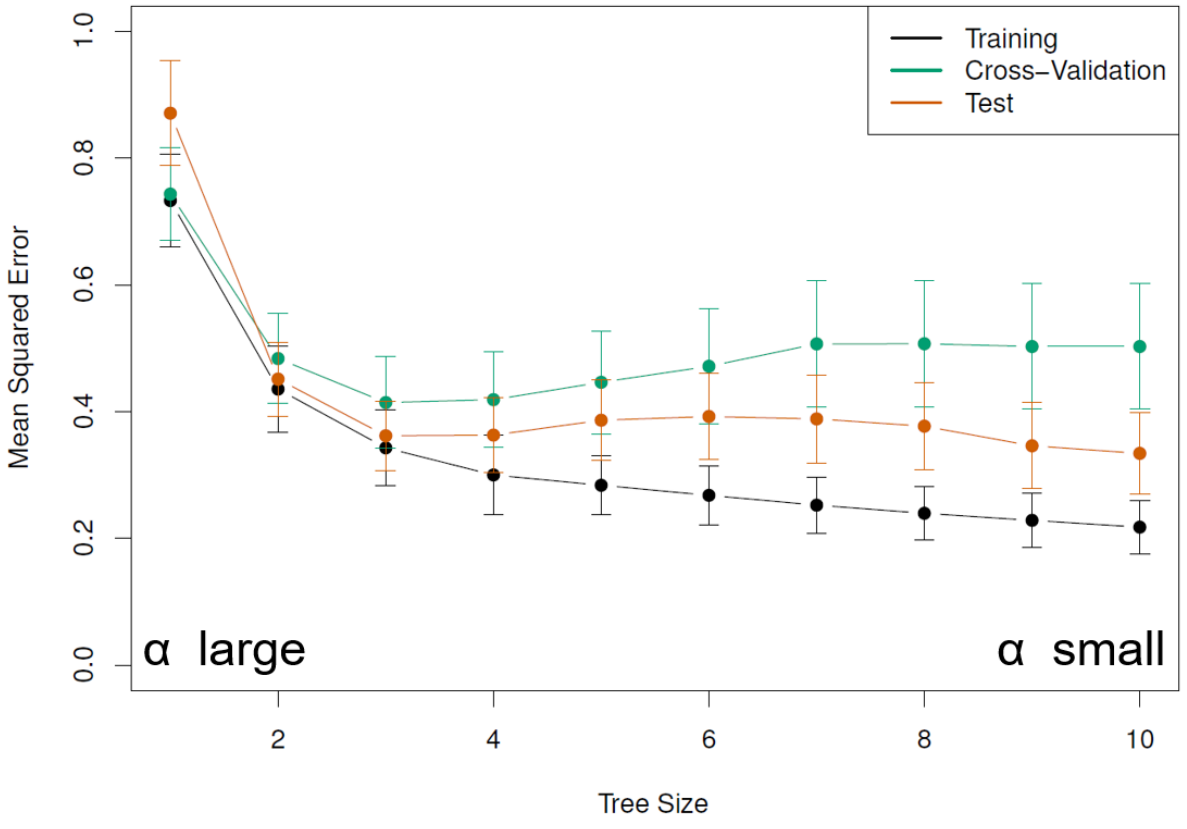


Extreme #2: $\alpha = \infty$

Also similar to lasso/ridge, when $\alpha = \infty$, we retain the *NULL* model.

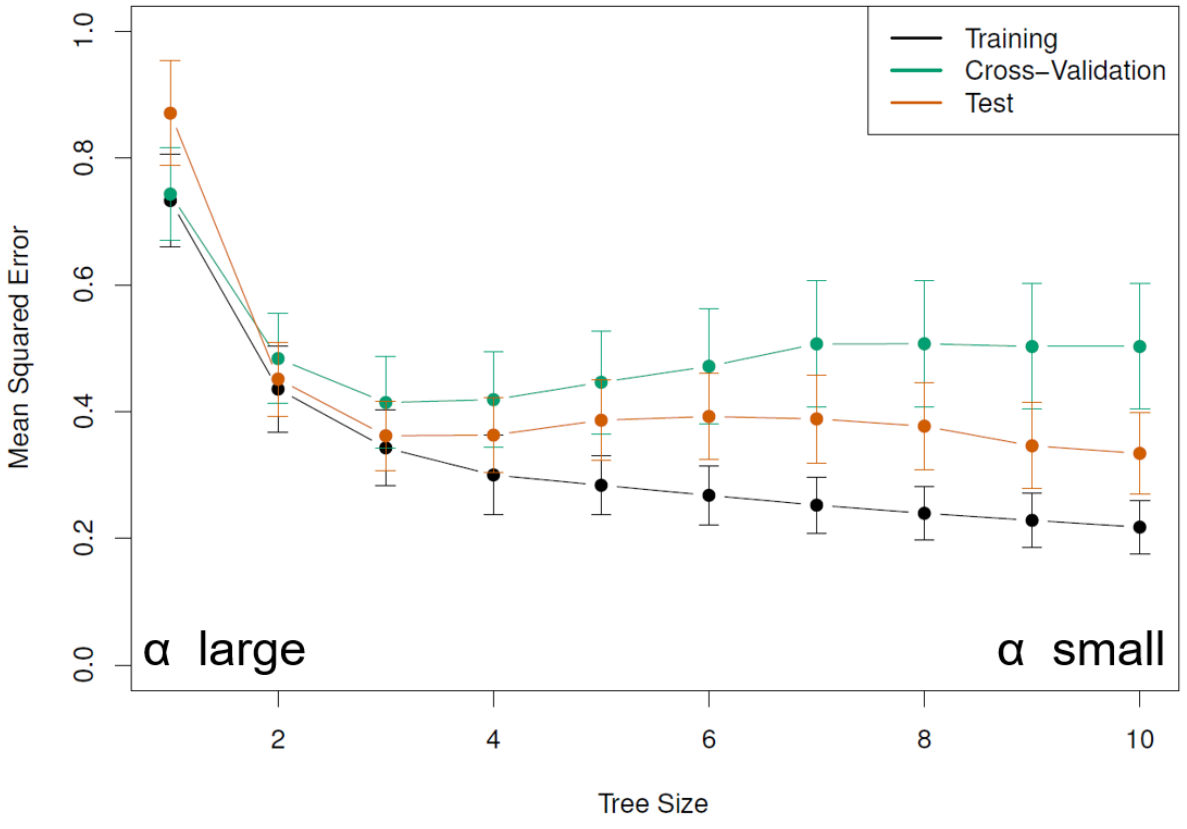


More generally: *tree-size* vs. *test error* (ISL, Fig. 8.5)



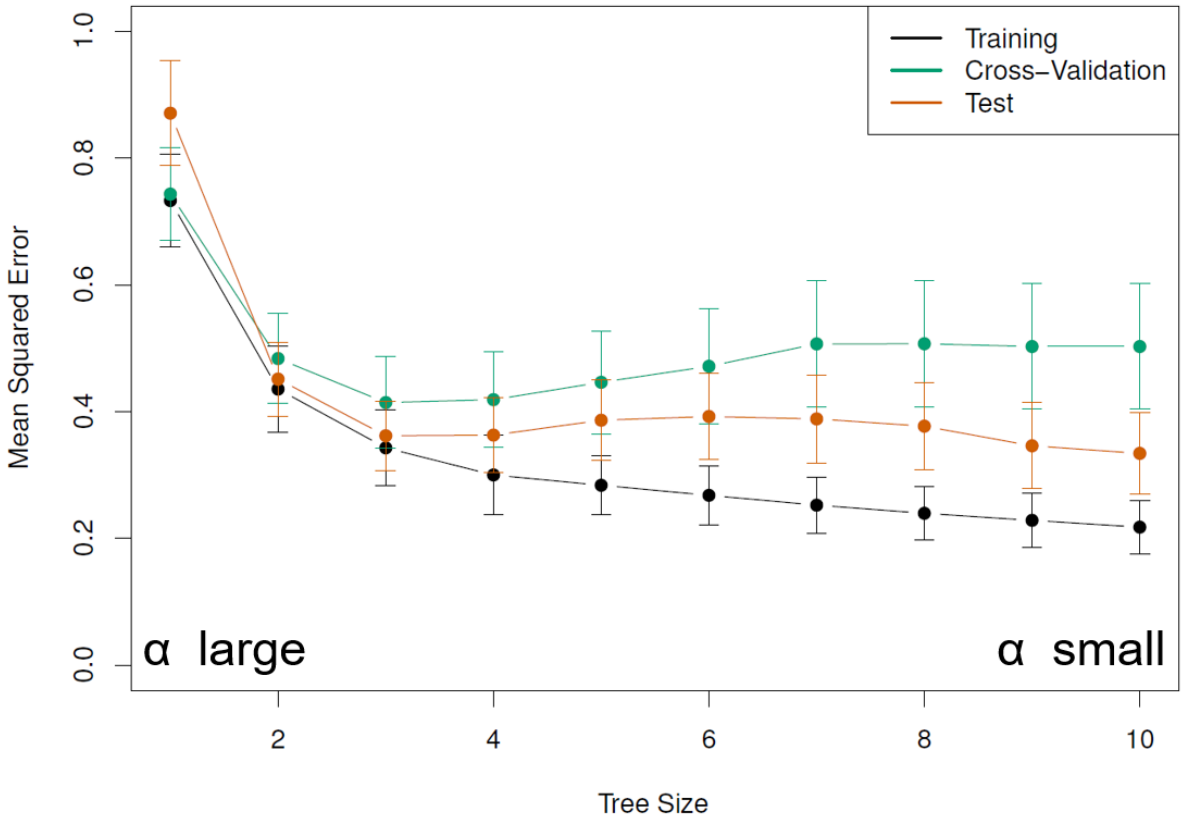
- As we grow *tree size* (complexity) training error always improves.

More generally: *tree-size* vs. *test error* (ISL, Fig. 8.5)



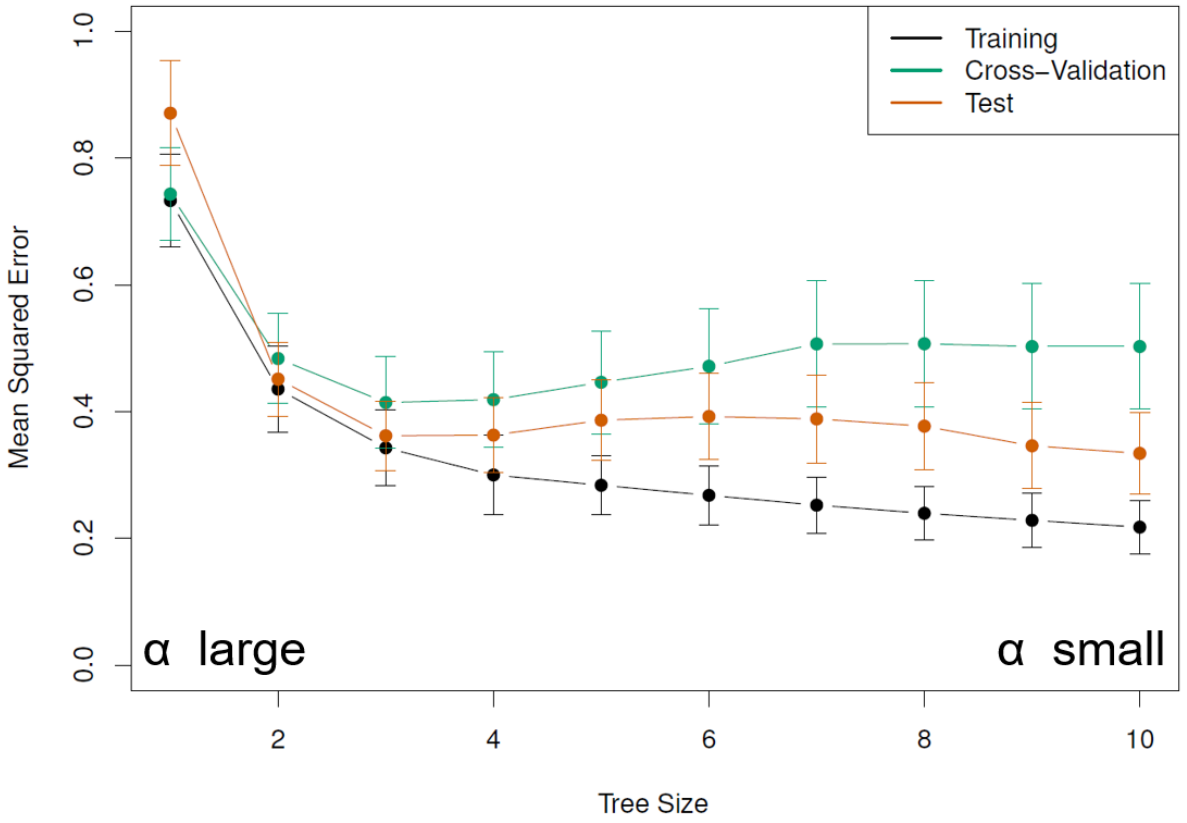
- As we grow *tree size* (complexity) training error always improves.
- But test error only improves initially, then it starts to worsen.

More generally: *tree-size* vs. *test error* (ISL, Fig. 8.5)



- As we grow *tree size* (complexity) training error always improves.
- But test error only improves initially, then it starts to worsen.
  - It improves *first* by *reducing bias*.

More generally: *tree-size* vs. *test error* (ISL, Fig. 8.5)



- As we grow *tree size* (complexity) training error always improves.
- But test error only improves initially, then it starts to worsen.
  - It improves *first* by *reducing bias*.
  - It starts to worsen because of *increased variance*.

Note: minimum 'leaf size'

- Typically, we do **not** grow the (large) tree T_0 until each leaf contains a single observation (perfect fit).

Note: minimum 'leaf size'

- Typically, we do **not** grow the (large) tree T_0 until each leaf contains a single observation (perfect fit).
- Instead — until a minimum leaf size (in R, default: 10).

Note: minimum 'leaf size'

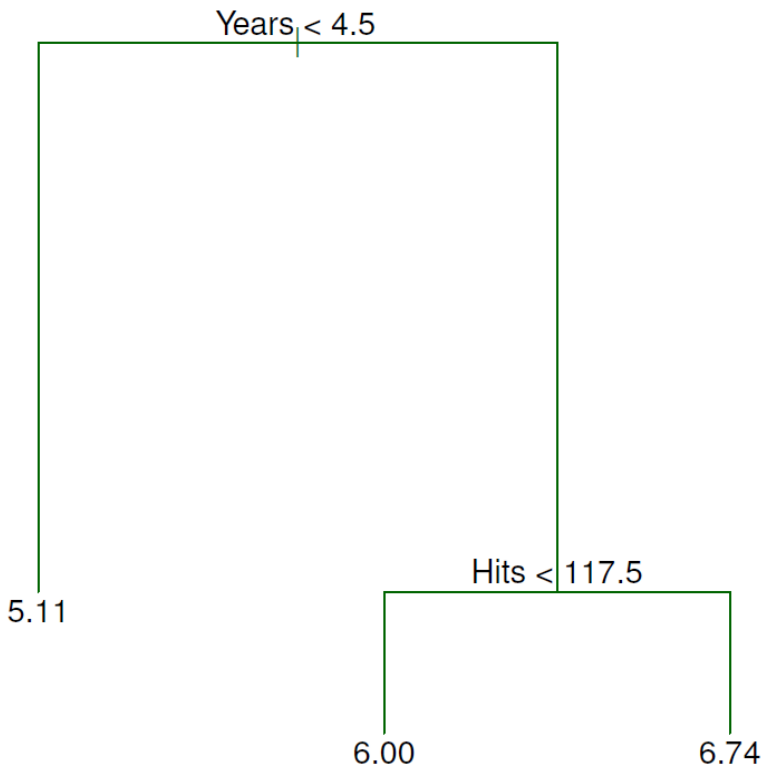
- Typically, we do **not** grow the (large) tree T_0 until each leaf contains a single observation (perfect fit).
- Instead — until a minimum leaf size (in R, default: 10).
- Implies a baseline (weak) protection against *overfitting*.

Some additional applied concerns

Now that we have a pretty good understanding of the "inner-workings" of decision trees, we'll consider some *additional applied concerns*:

- Interpretability
- General strengths, weaknesses.

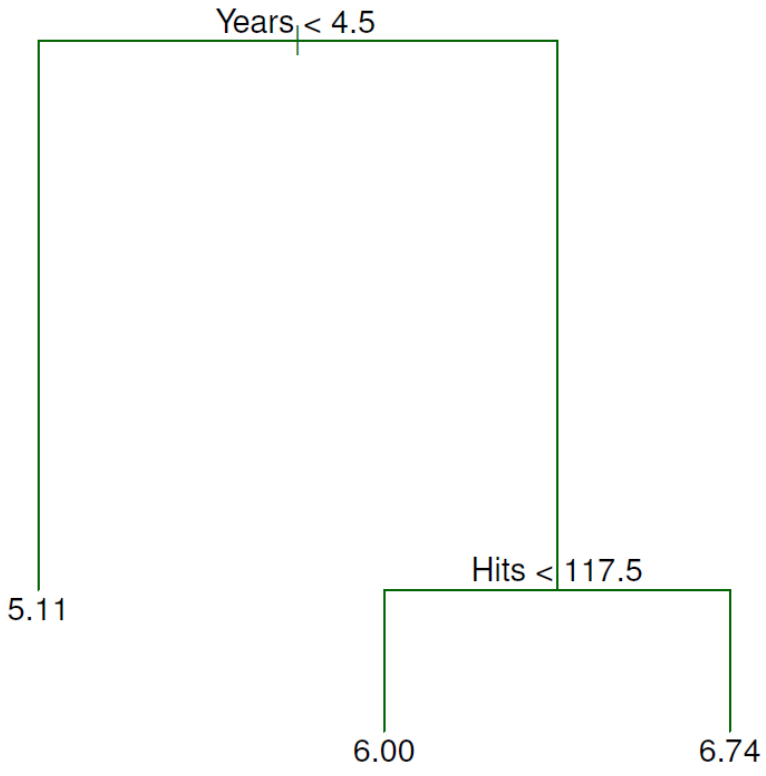
Re. interpretability



As noted earlier:

- Very interpretable because it provides simple depiction of how combinations of variables predict outcome.

Re. interpretability



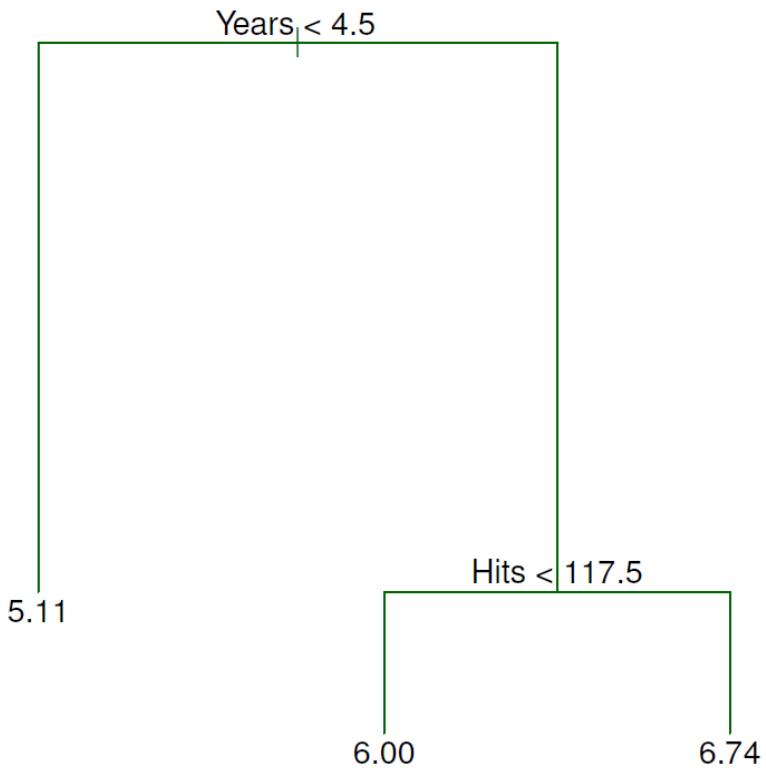
As noted earlier:

- Very interpretable because it provides simple depiction of how combinations of variables predict outcome.

Additionally:

- Measure 'variable importance' by *sum of error reduction* across splits.

Re. interpretability



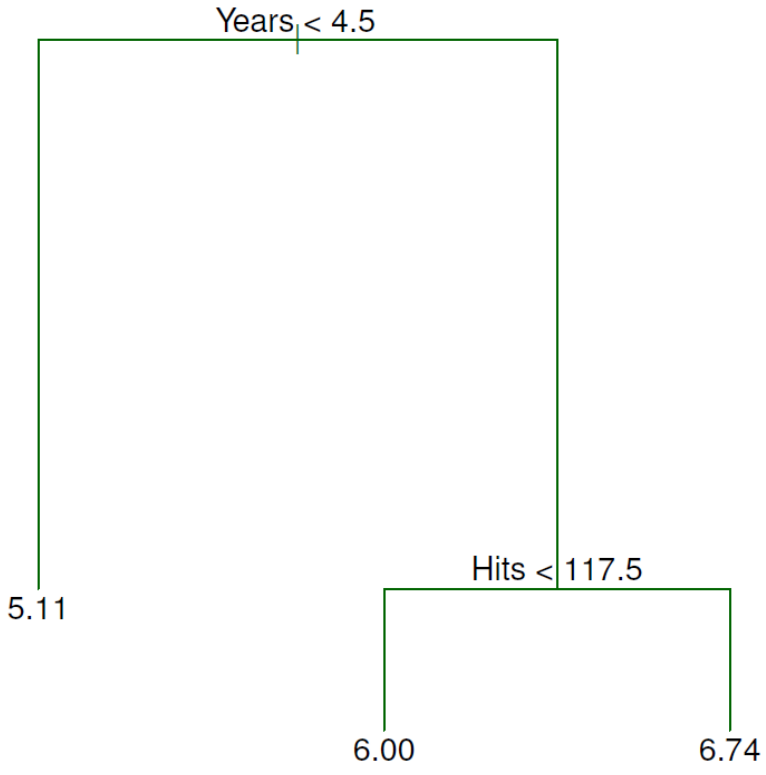
As noted earlier:

- Very interpretable because it provides simple depiction of how combinations of variables predict outcome.

Additionally:

- Measure 'variable importance' by *sum of error reduction* across splits.
- Including splits evaluated but not realized.

Re. interpretability



However, some caveats:

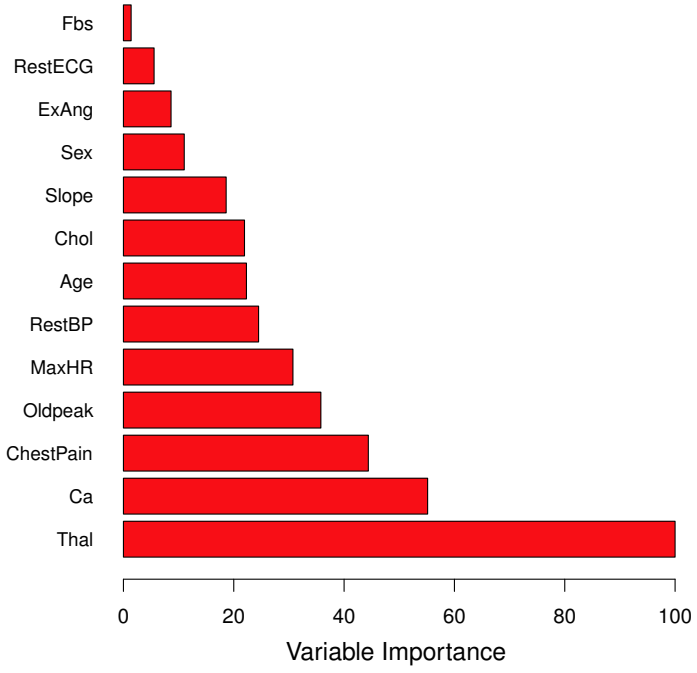
1. The structure does not reveal how X is associated with Y controlling for Z.
   - It simply finds splits that are predictive.
   - Thus—careful with interpretation.

Re. interpretability

However, some caveats:

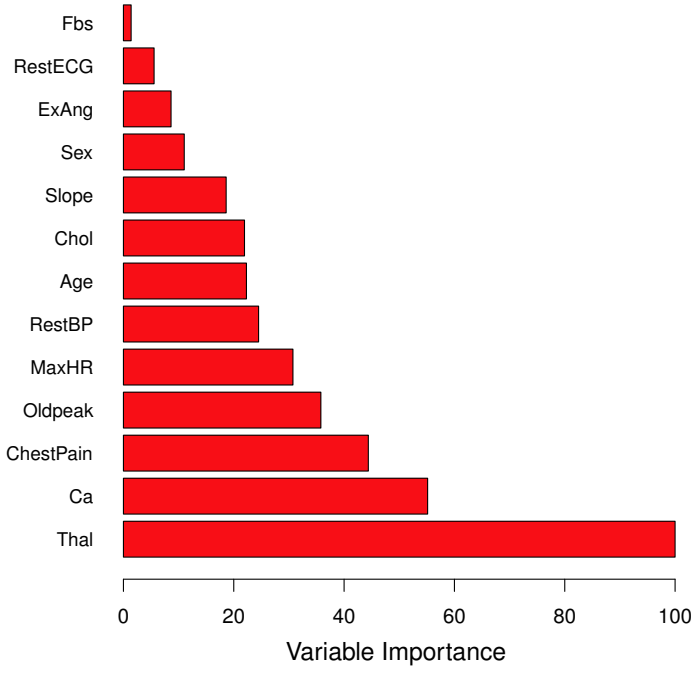
2. The structure is sometimes sensitive to observations used for training.
   - Does the branch you find interesting re-appear often or rarely?
   - Sensitivty analysis—bootstrap to examine this Q.

Plotting variable importance (ISL, Fig 8.9)



We usually present 'variable importance' in *relative terms*.

Plotting variable importance (ISL, Fig 8.9)

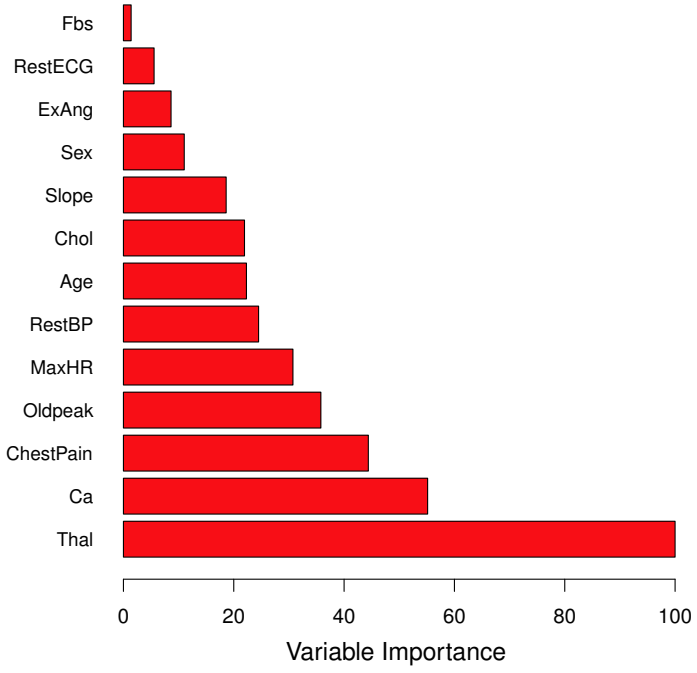


We usually present 'variable importance' in *relative terms*.

In this example:

- The variables with the largest decrease in error are `Thal` and `Ca`.

Plotting variable importance (ISL, Fig 8.9)



We usually present 'variable importance' in *relative terms*.

In this example:

- The variables with the largest decrease in error are `Thal` and `Ca`.
- `Ca` reducing the error about half as much as `Thal`.

Pros and cons of decision trees

Pros

a. Can be displayed graphically, and are easy to interpret.

Pros and cons of decision trees

Pros

a. Can be displayed graphically, and are easy to interpret.

b. They *inductively* create interactions & non-linear functions of X.

Pros and cons of decision trees

Pros

a. Can be displayed graphically, and are easy to interpret.

b. They *inductively* create interactions & non-linear functions of X.

c. a+b $\rightarrow$ can discover novel interactions predictive of the outcome.

Pros and cons of decision trees

Pros

a. Can be displayed graphically, and are easy to interpret.

b. They *inductively* create interactions & non-linear functions of X.

c. a+b $\rightarrow$ can discover novel interactions predictive of the outcome.

d. Key building block for more advanced methods.

Pros and cons of decision trees

Pros

a. Can be displayed graphically, and are easy to interpret.

b. They *inductively* create interactions & non-linear functions of X.

c. a+b $\rightarrow$ can discover novel interactions predictive of the outcome.

d. Key building block for more advanced methods.

Cons

- Usually *not competitive* in terms of prediction performance.

Pros and cons of decision trees

Pros

a. Can be displayed graphically, and are easy to interpret.

b. They *inductively* create interactions & non-linear functions of X.

c. a+b $\rightarrow$ can discover novel interactions predictive of the outcome.

d. Key building block for more advanced methods.

Cons

- Usually *not competitive* in terms of prediction performance.
  - Often requires many splits to capture patterns in data $\rightarrow$ overfitting.

Pros and cons of decision trees

Pros

a. Can be displayed graphically, and are easy to interpret.

b. They *inductively* create interactions & non-linear functions of X.

c. a+b $\rightarrow$ can discover novel interactions predictive of the outcome.

d. Key building block for more advanced methods.

Cons

- Usually *not competitive* in terms of prediction performance.
  - Often requires many splits to capture patterns in data $\rightarrow$ overfitting.
- Relatedly—can be very *non-robust*: small change in data $\rightarrow$ large change in tree structure.

Ensembles

Extensions to decision trees

We will now consider two methods—Bagging, Random Forest—that *address the limitations* of decision trees and that *increase performance* considerably.

Extensions to decision trees

We will now consider two methods—Bagging, Random Forest—that *address the limitations* of decision trees and that *increase performance* considerably.

They fall under the family of methods called **"ensemble methods"**

Bagging

As we just discussed, an important limitation of *decision trees* is that it suffers from **high variance**

Bagging

As we just discussed, an important limitation of *decision trees* is that it suffers from **high variance**

- E.g., fitting a decision tree to two different halves of training data may yield quite different results/structure.

Bagging

As we just discussed, an important limitation of *decision trees* is that it suffers from **high variance**

- E.g., fitting a decision tree to two different halves of training data may yield quite different results/structure.

Bagging is a general-purpose technique for reducing variance of machine learning methods.

Bagging

As we just discussed, an important limitation of *decision trees* is that it suffers from **high variance**

- E.g., fitting a decision tree to two different halves of training data may yield quite different results/structure.

Bagging is a general-purpose technique for reducing variance of machine learning methods.

- At the heart of bagging is *bootstrapping*.

Bagging

As we just discussed, an important limitation of *decision trees* is that it suffers from **high variance**

- E.g., fitting a decision tree to two different halves of training data may yield quite different results/structure.

Bagging is a general-purpose technique for reducing variance of machine learning methods.

- At the heart of bagging is *bootstrapping*.
- Recall the *basic procedure* of bootstrapping:

Bagging

As we just discussed, an important limitation of *decision trees* is that it suffers from **high variance**

- E.g., fitting a decision tree to two different halves of training data may yield quite different results/structure.

Bagging is a general-purpose technique for reducing variance of machine learning methods.

- At the heart of bagging is *bootstrapping*.
- Recall the *basic procedure* of bootstrapping:
  1. Resample original data (with replacement).

Bagging

As we just discussed, an important limitation of *decision trees* is that it suffers from **high variance**

- E.g., fitting a decision tree to two different halves of training data may yield quite different results/structure.

Bagging is a general-purpose technique for reducing variance of machine learning methods.

- At the heart of bagging is *bootstrapping*.
- Recall the *basic procedure* of bootstrapping:
  1. Resample original data (with replacement).
  2. Compute something on resampled data.

Bagging

As we just discussed, an important limitation of *decision trees* is that it suffers from **high variance**

- E.g., fitting a decision tree to two different halves of training data may yield quite different results/structure.

Bagging is a general-purpose technique for reducing variance of machine learning methods.

- At the heart of bagging is *bootstrapping*.
- Recall the *basic procedure* of bootstrapping:
  1. Resample original data (with replacement).
  2. Compute something on resampled data.
  3. Repeat 1–2 many times.

Why should this help to *reduce variance* of a model?

Classic statistics: given a set of n independent variables $Z_1, \ldots, Z_n$, each with a *variance* σ^2, the *variance* of the *mean* $\bar{Z}$ is given by $\frac{\sigma^2}{n}$.

Why should this help to *reduce variance* of a model?

Classic statistics: given a set of n independent variables $Z_1, \ldots, Z_n$, each with a *variance* σ^2, the *variance* of the *mean* $\bar{Z}$ is given by $\frac{\sigma^2}{n}$.

- That is: averaging *independent* observations reduces variance.

Why should this help to *reduce variance* of a model?

Classic statistics: given a set of n independent variables $Z_1, \ldots, Z_n$, each with a *variance* σ^2, the *variance* of the *mean* $\bar{Z}$ is given by $\frac{\sigma^2}{n}$.

- That is: averaging *independent* observations reduces variance.

This suggests that we can *reduce variance*—and hence increase the prediction accuracy—of a *ML method* by:

Why should this help to *reduce variance* of a model?

Classic statistics: given a set of n independent variables $Z_1, \ldots, Z_n$, each with a *variance* σ^2, the *variance* of the *mean* $\bar{Z}$ is given by $\frac{\sigma^2}{n}$.

- That is: averaging *independent* observations reduces variance.

This suggests that we can *reduce variance*—and hence increase the prediction accuracy—of a *ML method* by:

1. Collecting *many* independent *training sets* from the population,

Why should this help to *reduce variance* of a model?

Classic statistics: given a set of n independent variables $Z_1, \ldots, Z_n$, each with a *variance* σ^2, the *variance* of the *mean* $\bar{Z}$ is given by $\frac{\sigma^2}{n}$.

- That is: averaging *independent* observations reduces variance.

This suggests that we can *reduce variance*—and hence increase the prediction accuracy—of a *ML method* by:

1. Collecting *many* independent *training sets* from the population,
2. Build a separate prediction model $\hat{f}^b$ for *each training set* b, and

Why should this help to *reduce variance* of a model?

Classic statistics: given a set of n independent variables $Z_1, \ldots, Z_n$, each with a *variance* σ^2, the *variance* of the *mean* $\bar{Z}$ is given by $\frac{\sigma^2}{n}$.

- That is: averaging *independent* observations reduces variance.

This suggests that we can *reduce variance*—and hence increase the prediction accuracy—of a *ML method* by:

1. Collecting *many* independent *training sets* from the population,
2. Build a separate prediction model $\hat{f}^b$ for *each training set* b, and
3. Average resulting predictions to obtain a *single low-variance model*.

Why should this help to *reduce variance* of a model?

Classic statistics: given a set of n independent variables $Z_1, \ldots, Z_n$, each with a *variance* σ^2, the *variance* of the *mean* $\bar{Z}$ is given by $\frac{\sigma^2}{n}$.

- That is: averaging *independent* observations reduces variance.

This suggests that we can *reduce variance*—and hence increase the prediction accuracy—of a *ML method* by:

1. Collecting *many* independent *training sets* from the population,
2. Build a separate prediction model $\hat{f}^b$ for *each training set* b, and
3. Average resulting predictions to obtain a *single low-variance model*.

Can we get some more *intuition* for why this should work? *Toy example*!

Why averaging can help

Suppose we have 3 different *binary classifiers* ($\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$), each independently correct with probability 0.8.

Why averaging can help

Suppose we have 3 different *binary classifiers* ($\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$), each independently correct with probability 0.8.

The **binomial distribution** tells us the following:

- $P(\text{all 3 right}) = 0.8^3 = 0.512$

Why averaging can help

Suppose we have 3 different *binary classifiers* ($\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$), each independently correct with probability 0.8.

The **binomial distribution** tells us the following:

- $P(\text{all 3 right}) = 0.8^3 = 0.512$
- $P(\text{2 right, 1 wrong}) = 3 * 0.8^2(1 - 0.8) = 0.384$

Why averaging can help

Suppose we have 3 different *binary classifiers* ($\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$), each independently correct with probability 0.8.

The **binomial distribution** tells us the following:

- $P(\text{all 3 right}) = 0.8^3 = 0.512$
- $P(\text{2 right, 1 wrong}) = 3 * 0.8^2(1 - 0.8) = 0.384$
- $P(\text{1 right, 2 wrong}) = 3 * 0.8^1(1 - 0.8)^2 = 0.096$

Why averaging can help

Suppose we have 3 different *binary classifiers* ($\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$), each independently correct with probability 0.8.

The **binomial distribution** tells us the following:

- $P(\text{all 3 right}) = 0.8^3 = 0.512$
- $P(\text{2 right, 1 wrong}) = 3 * 0.8^2(1 - 0.8) = 0.384$
- $P(\text{1 right, 2 wrong}) = 3 * 0.8^1(1 - 0.8)^2 = 0.096$
- $P(\text{all 3 wrong}) = (1 - 0.8)^3 = 0.008$

Why averaging can help

Suppose we have 3 different *binary classifiers* ($\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$), each independently correct with probability 0.8.

The **binomial distribution** tells us the following:

- $P(\text{all 3 right}) = 0.8^3 = 0.512$
- $P(\text{2 right, 1 wrong}) = 3 * 0.8^2(1 - 0.8) = 0.384$
- $P(\text{1 right, 2 wrong}) = 3 * 0.8^1(1 - 0.8)^2 = 0.096$
- $P(\text{all 3 wrong}) = (1 - 0.8)^3 = 0.008$

Thus: averaging the three, we get an *expected accuracy* of:
$0.512 + 0.384 = \mathbf{0.896} > 0.8$!

OK great, but in practice—don't have multiple data sets?

OK great, but in practice—don't have multiple data sets?

Solution: Emulate this "ideal process" by—instead of collecting new training sets—taking repeated samples from the (single) training data set we do have:

OK great, but in practice—don't have multiple data sets?

Solution: Emulate this "ideal process" by—instead of collecting new training sets—taking repeated samples from the (single) training data set we do have:

1. Generate B different *bootstrapped* training data sets,

OK great, but in practice—don't have multiple data sets?

Solution: Emulate this "ideal process" by—instead of collecting new training sets—taking repeated samples from the (single) training data set we do have:

1. Generate B different *bootstrapped* training data sets,
2. Train our method on the b'th bootstrapped training set to get $\hat{f}^b(x)$, and

OK great, but in practice—don't have multiple data sets?

Solution: Emulate this "ideal process" by—instead of collecting new training sets—taking repeated samples from the (single) training data set we do have:

1. Generate B different *bootstrapped* training data sets,
2. Train our method on the b'th bootstrapped training set to get $\hat{f}^b(x)$, and
3. Finally, we average all the predictions to obtain:

OK great, but in practice—don't have multiple data sets?

Solution: Emulate this "ideal process" by—instead of collecting new training sets—taking repeated samples from the (single) training data set we do have:

1. Generate B different *bootstrapped* training data sets,
2. Train our method on the b'th bootstrapped training set to get $\hat{f}^b(x)$, and
3. Finally, we average all the predictions to obtain:

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

OK great, but in practice—don't have multiple data sets?

Solution: Emulate this "ideal process" by—instead of collecting new training sets—taking repeated samples from the (single) training data set we do have:

1. Generate B different *bootstrapped* training data sets,
2. Train our method on the b'th bootstrapped training set to get $\hat{f}^b(x)$, and
3. Finally, we average all the predictions to obtain:

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

This is called **bagging**.

Bagged trees

Bagging is a *general approach* but has been shown to work *especially well* for *decision trees*.

Bagged trees

Bagging is a *general approach* but has been shown to work *especially well* for *decision trees*.

Basic idea

- Train B separate regression trees using B *bootstrapped training sets*, and average the resulting predictions.

Bagged trees

Bagging is a *general approach* but has been shown to work *especially well* for *decision trees*.

Basic idea

- Train B separate regression trees using B *bootstrapped training sets*, and average the resulting predictions.
- Each tree is trained **deep** and *not pruned* $\rightarrow$ each tree has *low bias* and *high variance*.

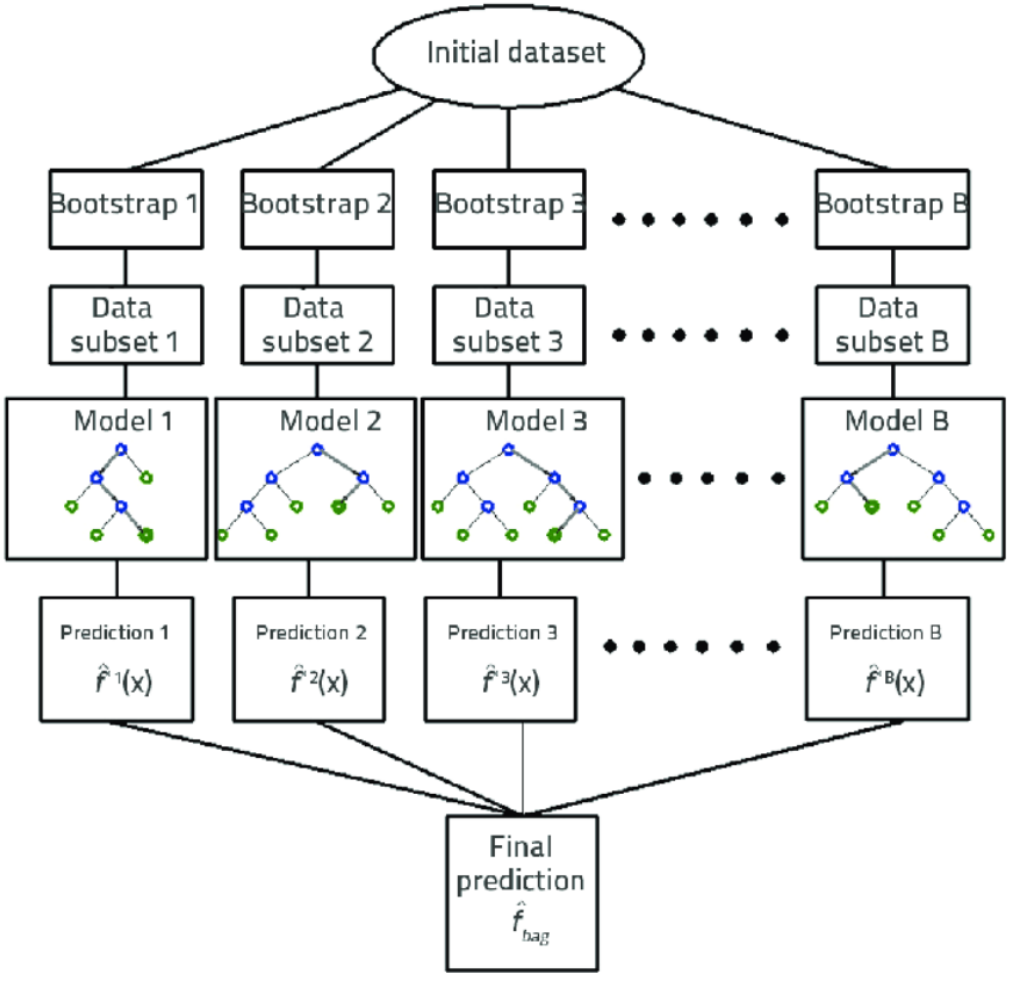
Bagged trees

Bagging is a *general approach* but has been shown to work *especially well* for *decision trees*.

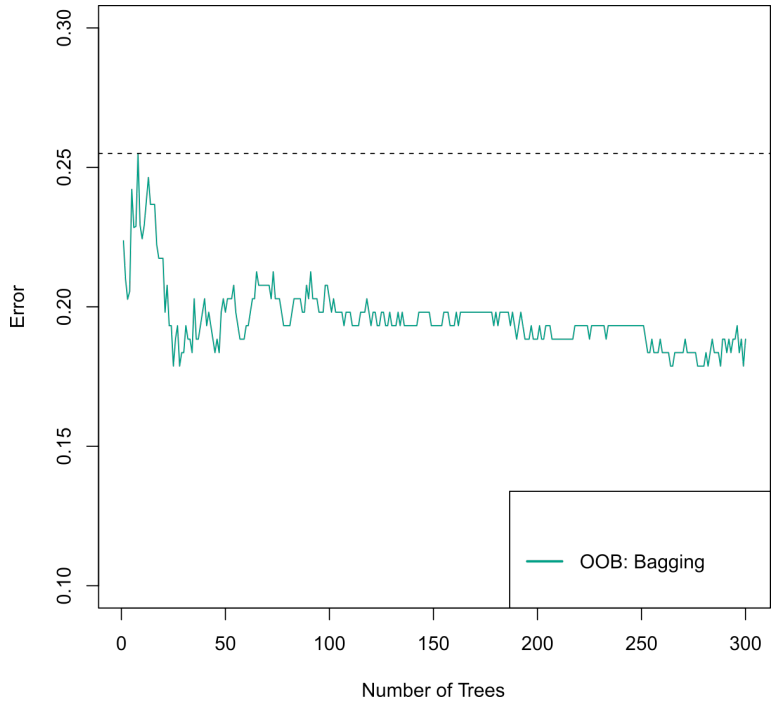
Basic idea

- Train B separate regression trees using B *bootstrapped training sets*, and average the resulting predictions.
- Each tree is trained **deep** and *not pruned* $\rightarrow$ each tree has *low bias* and *high variance*.
- Averaging these B trees reduces the variance—in the hope to get the "best of both worlds" (low bias, low variance).

Illustration



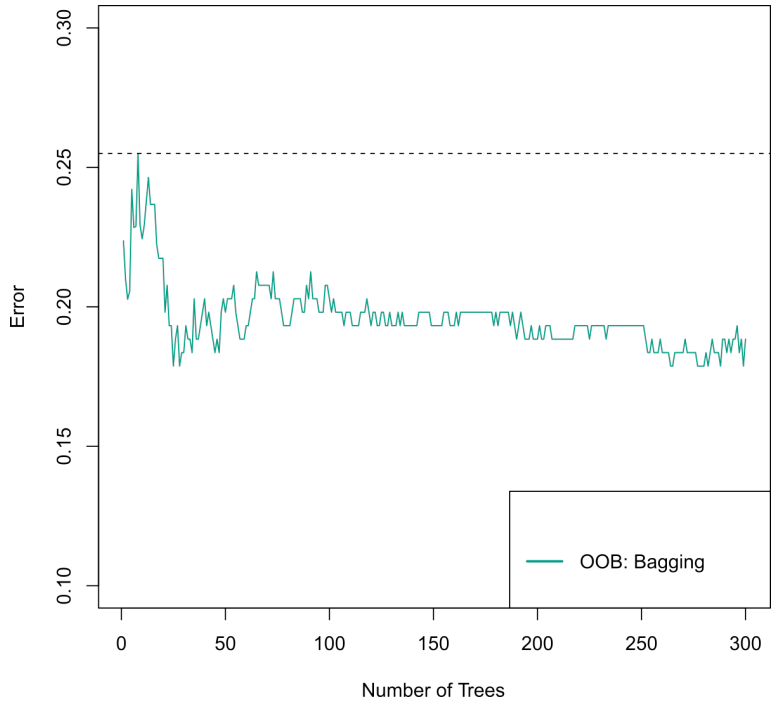
One tree vs. many (ISL, Fig. 8.8)



Remarks:

- *Averaging many trees* improves *test error* compared to a single tree.

One tree vs. many (ISL, Fig. 8.8)



Remarks:

- *Averaging many trees* improves *test error* compared to a single tree.
- Results are robust to the *choice of no. of trees*. Typical choice: 100-1000.

Interpretability

- While bagging tends to improve accuracy compared to single tree, it comes at the expense of reduction in interpretability.

Interpretability

- While bagging tends to improve accuracy compared to single tree, it comes at the expense of reduction in interpretability.
  - Why? We can *no longer* inspect *tree structure* and see rules for predictions.

Interpretability

- While bagging tends to improve accuracy compared to single tree, it comes at the expense of reduction in interpretability.
  - Why? We can *no longer* inspect *tree structure* and see rules for predictions.
- However—we can still obtain a *variable importance* measure.

Interpretability

- While bagging tends to improve accuracy compared to single tree, it comes at the expense of reduction in interpretability.
  - Why? We can *no longer* inspect *tree structure* and see rules for predictions.
- However—we can still obtain a *variable importance* measure.
  - Same measure as for a single tree, just *averaged across all trees*.

Pros and cons of *bagging*

Pros

- Can *improve* prediction *performance* considerably

Pros and cons of *bagging*

Pros

- Can *improve* prediction *performance* considerably
- Increases stability (also on *variable importance* measures)

Pros and cons of *bagging*

Pros

- Can *improve* prediction *performance* considerably
- Increases stability (also on *variable importance* measures)

Cons

- Reduces interpretability

Can we improve prediction performance further?

- **Recall**: the statistical theory suggested that the reason *why bagging can help*—reducing variance—relied on the *assumption* of independent samples.

Can we improve prediction performance further?

- **Recall**: the statistical theory suggested that the reason *why bagging can help*—reducing variance—relied on the *assumption* of independent samples.
- Because we resample from the *same original data*, and because grow the *tree to be large*, it is likely that the *different bagged trees* actually are rather correlated.

Can we improve prediction performance further?

- **Recall**: the statistical theory suggested that the reason *why bagging can help*—reducing variance—relied on the *assumption* of independent samples.
- Because we resample from the *same original data*, and because grow the *tree to be large*, it is likely that the *different bagged trees* actually are rather correlated.
- Averaging correlated predictions $\rightarrow$ *less of reduction* in variance.

Can we improve prediction performance further?

- **Recall**: the statistical theory suggested that the reason *why bagging can help*—reducing variance—relied on the *assumption* of independent samples.
- Because we resample from the *same original data*, and because grow the *tree to be large*, it is likely that the *different bagged trees* actually are rather correlated.
- Averaging correlated predictions $\rightarrow$ *less of reduction* in variance.

Can we make trees *less correlated*?

Yes! *Random forest*

A method called *random forest* addresses this problem by slightly tweaking the *bagging algorithm* to decorrelate the trees.

Yes! *Random forest*

A method called *random forest* addresses this problem by slightly tweaking the *bagging algorithm* to decorrelate the trees. **How?**

Yes! *Random forest*

A method called *random forest* addresses this problem by slightly tweaking the *bagging algorithm* to decorrelate the trees. **How?**

As in bagging:

- Build a number of trees on bootstrapped samples.

Yes! *Random forest*

A method called *random forest* addresses this problem by slightly tweaking the *bagging algorithm* to decorrelate the trees. **How?**

As in bagging:

- Build a number of trees on bootstrapped samples.

But different from bagging:

- Each time a split is *considered*, only a random sample of $m < p$ predictors are chosen as "split candidates" instead of full set (p).

Yes! *Random forest*

A method called *random forest* addresses this problem by slightly tweaking the *bagging algorithm* to decorrelate the trees. **How?**

As in bagging:

- Build a number of trees on bootstrapped samples.

But different from bagging:

- Each time a split is *considered*, only a random sample of $m < p$ predictors are chosen as "split candidates" instead of full set (p).
- Heruistic: $m = \sqrt{p}$.

Yes! *Random forest*

A method called *random forest* addresses this problem by slightly tweaking the *bagging algorithm* to decorrelate the trees. **How?**

As in bagging:

- Build a number of trees on bootstrapped samples.

But different from bagging:

- Each time a split is *considered*, only a random sample of $m < p$ predictors are chosen as "split candidates" instead of full set (p).
- Heruistic: $m = \sqrt{p}$.
  - E.g., for 36 predictors, only 6 will be considered at a given split.

Yes! *Random forest*

A method called *random forest* addresses this problem by slightly tweaking the *bagging algorithm* to decorrelate the trees. **How?**

As in bagging:

- Build a number of trees on bootstrapped samples.

But different from bagging:

- Each time a split is *considered*, only a random sample of $m < p$ predictors are chosen as "split candidates" instead of full set (p).
- Heruistic: $m = \sqrt{p}$.
  - E.g., for 36 predictors, only 6 will be considered at a given split.
  - Thus—each split does not even consider a majority of predictors.

What is the rationale for such strong restrictions?

Counterintuitive: why exclude most important variables from some trees?

What is the rationale for such strong restrictions?

Counterintuitive: why exclude most important variables from some trees?

Basic answer: If we don't;

- They will dominate most trees.

What is the rationale for such strong restrictions?

Counterintuitive: why exclude most important variables from some trees?

Basic answer: If we don't;

- They will dominate most trees.
- Thus create similarity (correlation) across trees.

What is the rationale for such strong restrictions?

Counterintuitive: why exclude most important variables from some trees?

Basic answer: If we don't;

- They will dominate most trees.
- Thus create similarity (correlation) across trees.
- Averaging will not reduce variance as much.

What is the rationale for such strong restrictions?

Counterintuitive: why exclude most important variables from some trees?

Basic answer: If we don't;

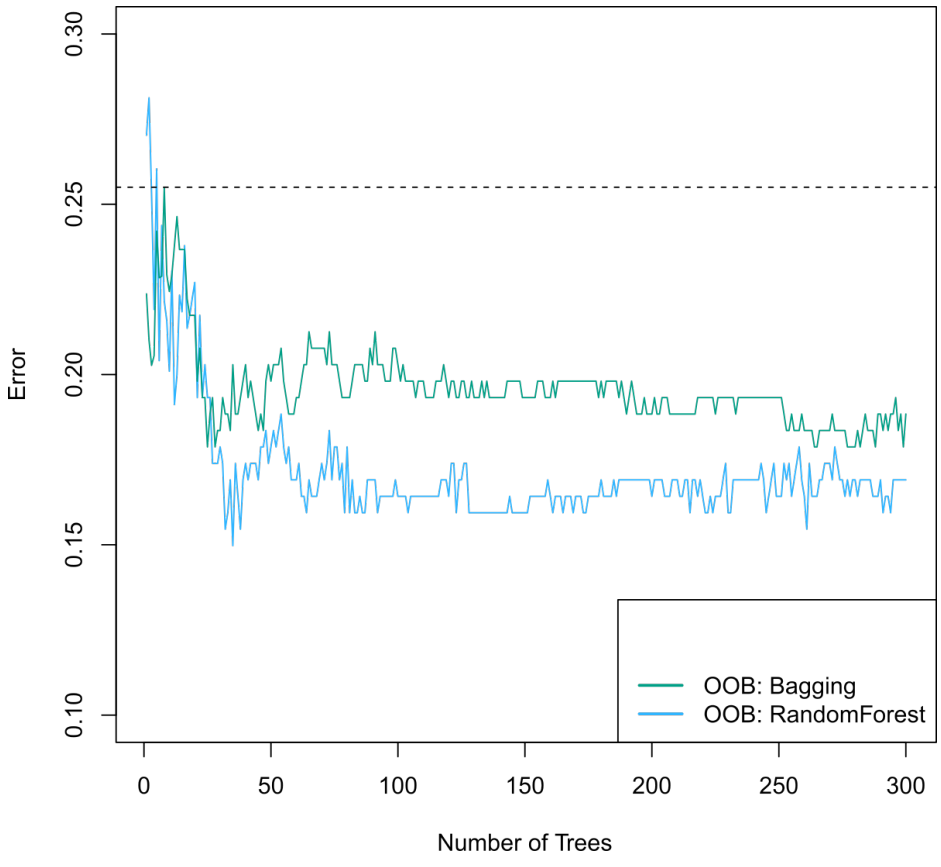
- They will dominate most trees.
- Thus create similarity (correlation) across trees.
- Averaging will not reduce variance as much.

In other words:

Imposing restrictions reduce the power of any given tree, but pooling them, they together become stronger.

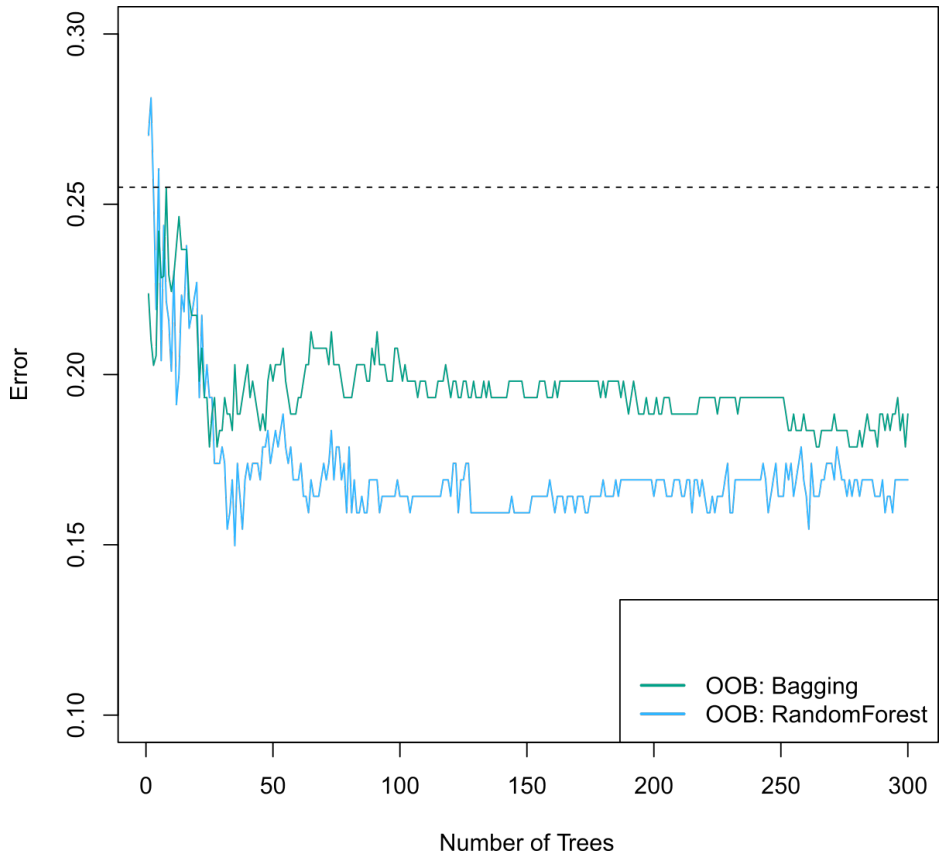
ISL example (Fig. 8.8)

Comparing *random forest* to *bagging*, we see that *random forest* here provides a substantial improvement!



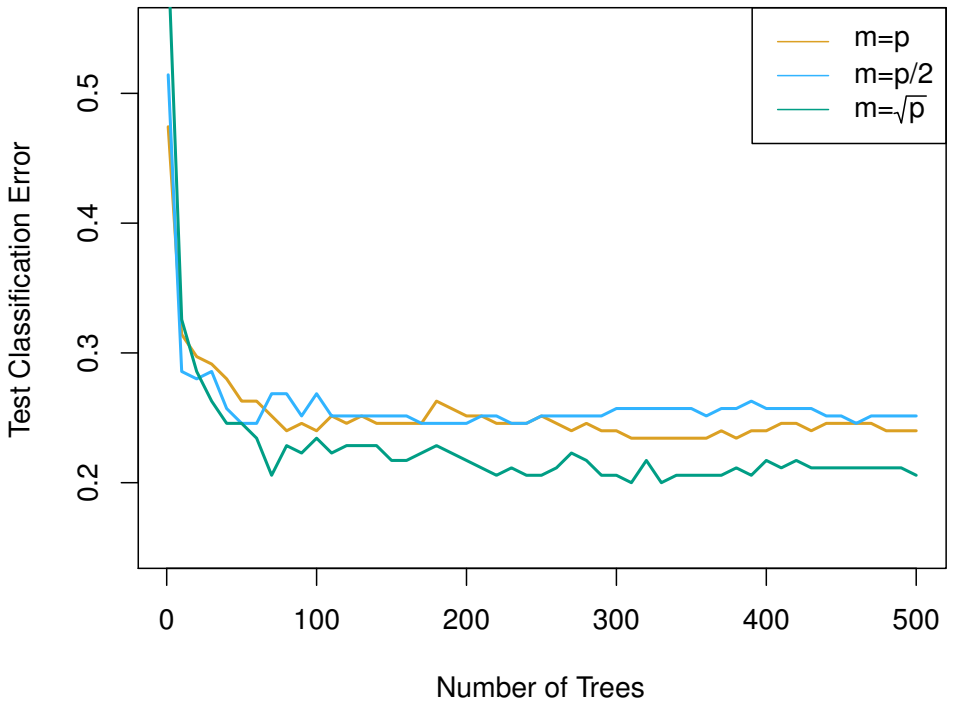
ISL example (Fig. 8.8)

Comparing *random forest* to *bagging*, we see that *random forest* here provides a substantial improvement!



Note: as with bagging, the results are usually quite stable under different *number of trees used* (beyond a few hundred).

Effect of m (ISL, Fig. 8.10)



This example shows how *restricting the number of variables considered in every split* can improve performance.

Pros and cons of *random forest*

Pros

- Even *stronger performance* compared to *bagging*.

Cons

- In comparison to *standard decision tree*—less interpretable.

Pros and cons of *random forest*

Pros

- Even *stronger performance* compared to *bagging*.

Cons

- In comparison to *standard decision tree*—less interpretable.

For both bagging and RFs:

- Better predictions — but less interpretability

Pros and cons of *random forest*

Pros

- Even *stronger performance* compared to *bagging*.

Cons

- In comparison to *standard decision tree*—less interpretable.

For both bagging and RFs:

- Better predictions — but less interpretability
- Would be very useful to understand why predictions improve!

Pros and cons of *random forest*

Pros

- Even *stronger performance* compared to *bagging*.

Cons

- In comparison to *standard decision tree*—less interpretable.

For both bagging and RFs:

- Better predictions — but less interpretability
- Would be very useful to understand why predictions improve!
- Maybe picked up novel pattern that is scientifically interesting?

Interpretable ML

Interpretable Machine Learning

Recent focus in machine learning literature (Molnar 2020)

- Develops **post-hoc techniques** to *extract* interpretability from *black-box methods*.

Interpretable Machine Learning

Recent focus in machine learning literature (Molnar 2020)

- Develops **post-hoc techniques** to *extract* interpretability from *black-box methods*.

We will briefly consider two popular techniques:

Interpretable Machine Learning

Recent focus in machine learning literature (Molnar 2020)

- Develops **post-hoc techniques** to *extract* interpretability from *black-box methods*.

We will briefly consider two popular techniques:

- *Permutation variable importance* measure

Interpretable Machine Learning

Recent focus in machine learning literature (Molnar 2020)

- Develops **post-hoc techniques** to *extract* interpretability from *black-box methods*.

We will briefly consider two popular techniques:

- *Permutation variable importance* measure
- *Partial dependence* plots

Permutation variable importance

Very informative quote from Boelaert & Ollion (2018):

- Read when you get time.

BOX 3. – ***Permutation importance: which variables matter for prediction?***

As its name suggests, the *permutation importance* measure aims at assessing the relative (predictive) importance of the variables at hand, using permutation. It exploits the fact that once the model has been trained, it works as a prediction machine that can be fed any predictor data. Its principle is simple: using a subset of the population under study (preferably the test set), the values of a given variable are randomly permuted (rearranged in a random order), leaving all other variables intact. When these perturbed data are passed through the model, the prediction quality is degraded (when compared to prediction on the unperturbed data). This degradation in prediction quality is interpreted as an importance measure for the current variable: a variable that makes the model much worse when it is permuted is important, while a variable that can be permuted without affecting prediction quality is not important. The procedure is repeated for each variable in sequence, and may be repeated several times for each variable to improve robustness. The whole process thus offers a standardized, interpretable measure of variable importance that can be compared across predictors (both continuous or categorical), does not depend on the type of learning model used, and demands only a single trained model.[26]

Permutation variable importance

Basic procedure:

1. Estimate your $\hat{f}$.

Permutation variable importance

Basic procedure:

1. Estimate your $\hat{f}$.
2. Generate predictions from $\hat{f}$ to assess *baseline accuracy*.

Permutation variable importance

Basic procedure:

1. Estimate your $\hat{f}$.
2. Generate predictions from $\hat{f}$ to assess *baseline accuracy*.
3. Permutate variable X_1.

Permutation variable importance

Basic procedure:

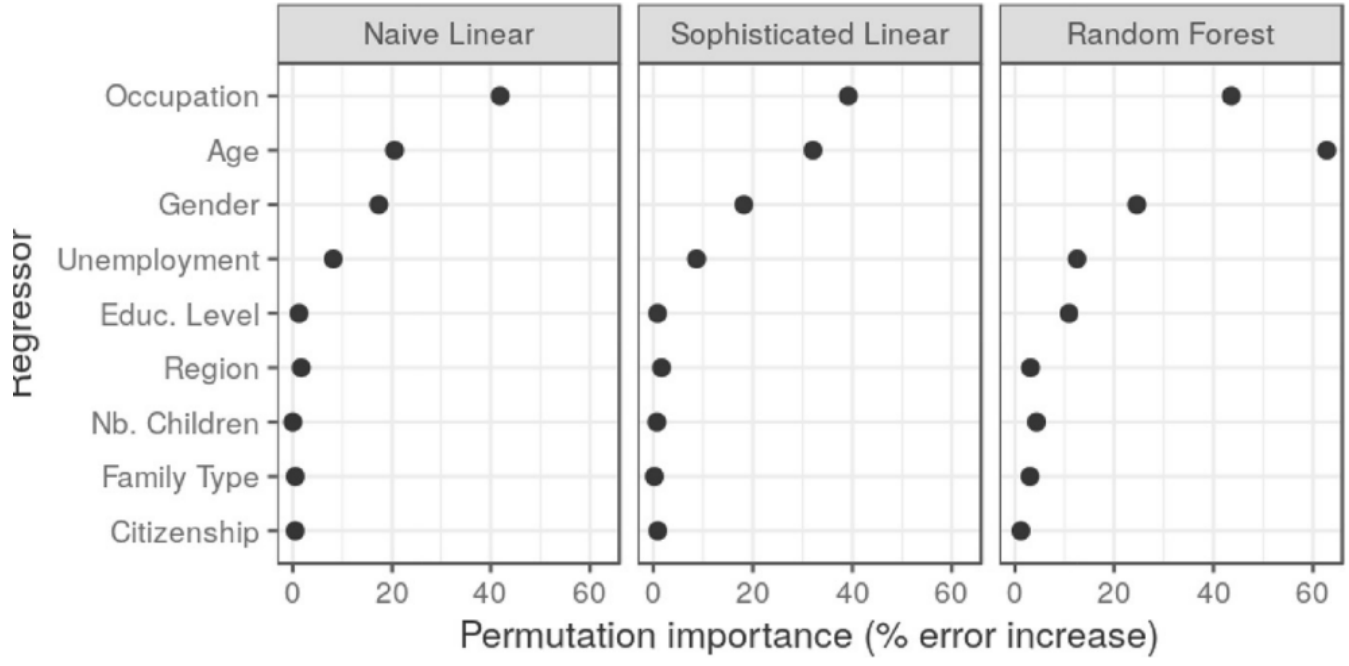
1. Estimate your $\hat{f}$.
2. Generate predictions from $\hat{f}$ to assess *baseline accuracy*.
3. Permutate variable X_1.
4. Generate predictions from $\hat{f}$ on *modified data* and assess *accuracy*.

Permutation variable importance

Basic procedure:

1. Estimate your $\hat{f}$.
2. Generate predictions from $\hat{f}$ to assess *baseline accuracy*.
3. Permutate variable X_1.
4. Generate predictions from $\hat{f}$ on *modified data* and assess *accuracy*.
5. Difference in accuracy between step 2 and 4 = importance of variable X_1.

Example (Boelaert & Ollion 2018)



Partial dependence plot

Very informative quote from Boelaert & Ollion (2018):

- Read when you get time.

BOX 4. – ***Partial dependence: isolating the effect of a single variable in a complex model***

The intuition for partial dependence is that, if one is to study the association of a single variable with the outcome, the effects of all other predictors can be neutralized by *averaging them out*. Just as permutation importance, partial dependence combines predictive power and perturbations of the original dataset to produce interpretable measures. The partial dependence of the outcome variable on a given predictor X_1 is computed in the following way: for each value x that X_1 takes in the dataset (and preferably in its test-subsample), a new dataset is created in which all values of X_1 are replaced by this single value x. If X_1 is gender, for instance, two synthetic datasets are created, each one with as many observations (N) as the base dataset: in the first table all observations are assumed to be women (while all other variables are left untouched), in the other all are set to be men. Both these synthetic datasets are fed, in turn, to the predictive model, yielding N predicted wages for synthetic women, and another N for synthetic men. Finally, we compute the average prediction for the synthetic men and for the synthetic women, and these two average predictions make up the partial dependence plot.[28] In the case of age, 47 different synthetic datasets of size N are created: one where all individuals are set to be 18 years old, one for 19, and so on until 65 years old.

Partial dependence plot

Basic procedure
For some variable $X_1 \ldots$

Partial dependence plot

Basic procedure
For some variable $X_1 \ldots$

1. For a given value x that X_1 takes in the data, create a new synthethic data set in which all values of X_1 is replaced by this value.

Partial dependence plot

Basic procedure
For some variable $X_1 \ldots$

1. For a given value x that X_1 takes in the data, create a new synthethic data set in which all values of X_1 is replaced by this value.
2. Generate predictions for data set from step 1.

Partial dependence plot

Basic procedure
For some variable $X_1 \ldots$

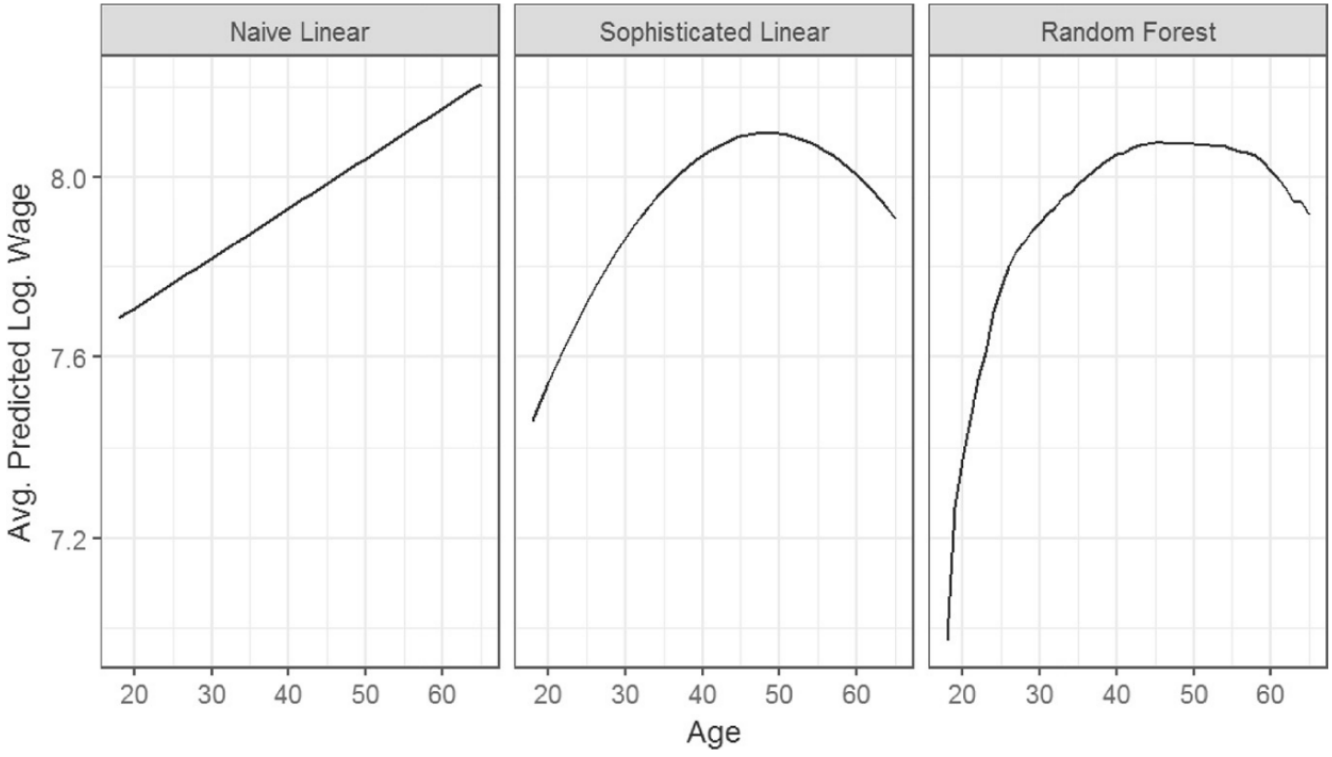
1. For a given value x that X_1 takes in the data, create a new synthethic data set in which all values of X_1 is replaced by this value.
2. Generate predictions for data set from step 1.
3. Average predictions from step 2 into a single prediction.

Partial dependence plot

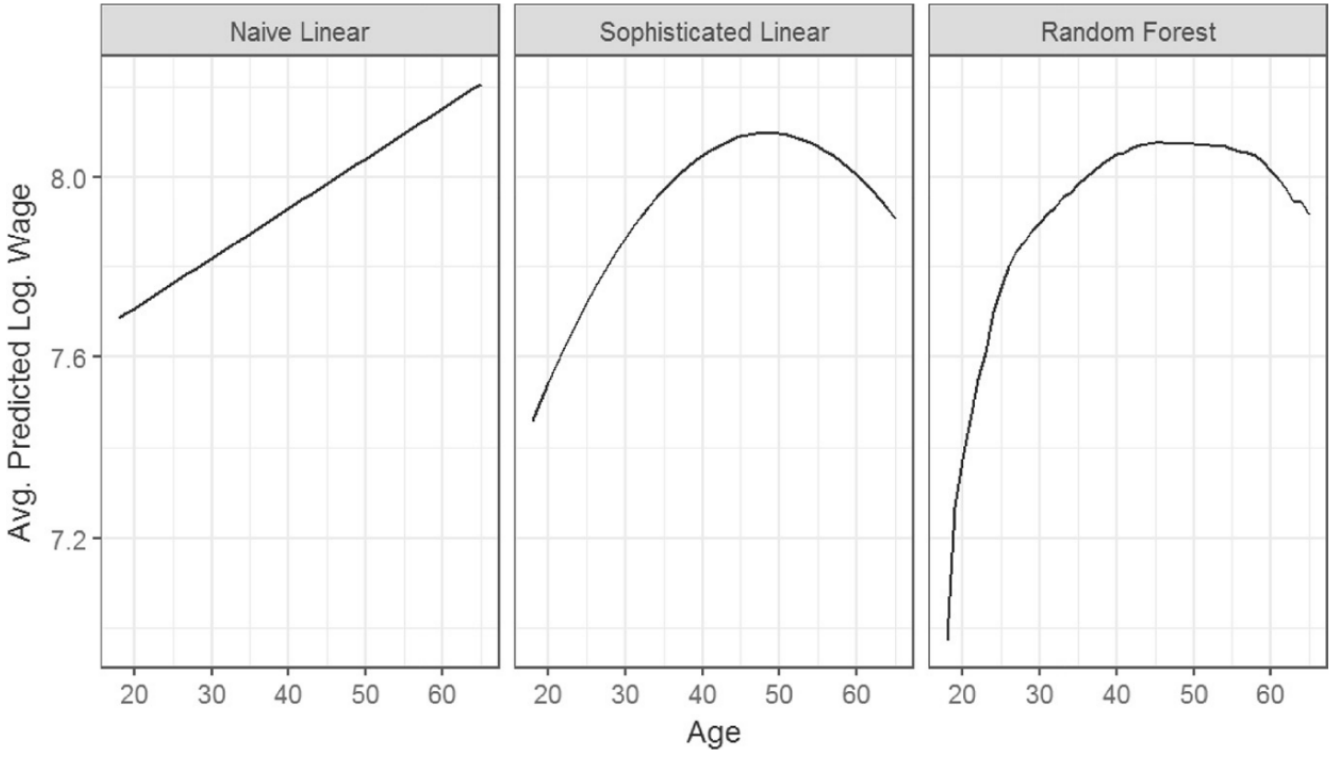
Basic procedure
For some variable $X_1 \ldots$

1. For a given value x that X_1 takes in the data, create a new synthethic data set in which all values of X_1 is replaced by this value.
2. Generate predictions for data set from step 1.
3. Average predictions from step 2 into a single prediction.
4. Repeat step 1–3 for all unique values for X_1.

Example (Boelaert & Ollion 2018)



Example (Boelaert & Ollion 2018)



Here, variable/pattern not so-so interesting perhaps. But you could imagine a model revealing some non-linearity not previously considered in the literature.

Recapping this last part

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$

Recapping this last part

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$
- KNN — Predict $\hat{y}_i$ based on closest k training obs to i.

Recapping this last part

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$
- KNN — Predict $\hat{y}_i$ based on closest k training obs to i.
- Decision trees
  - Partition X space to create homogeneous y regions.
  - *Recursive binary splitting*: nested if-else statements using variables that, at each step, increase homogeneity in y the most.
  - Predict $\hat{y}_i$ based on which region/leaf i belongs.

Recapping this last part

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$
- KNN — Predict $\hat{y}_i$ based on closest k training obs to i.
- Decision trees
  - Partition X space to create homogeneous y regions.
  - *Recursive binary splitting*: nested if-else statements using variables that, at each step, increase homogeneity in y the most.
  - Predict $\hat{y}_i$ based on which region/leaf i belongs.
- Bagging, random forest
  - Bootstrap original data, estimate decision trees in each bootstrap sample, and average predictions across trees.
  - Random forest extends bagging by decorrelating the trees.

Recapping this last part

- Non-parameteric supervised learning
  - Parametric: makes strong assumptions about $\hat{f}$ (e.g., additive)
  - Non-parametric: does not make explicit assumptions about parametric form or $\hat{f}$
- KNN — Predict $\hat{y}_i$ based on closest k training obs to i.
- Decision trees
  - Partition X space to create homogeneous y regions.
  - *Recursive binary splitting*: nested if-else statements using variables that, at each step, increase homogeneity in y the most.
  - Predict $\hat{y}_i$ based on which region/leaf i belongs.
- Bagging, random forest
  - Bootstrap original data, estimate decision trees in each bootstrap sample, and average predictions across trees.
  - Random forest extends bagging by decorrelating the trees.
- Interpretable machine learning

That is all — now break, then lab!