Decision Trees (II)

Dr. Tun-Wen Pai

1) Classification and Regression Trees (CART)
2) Gini Index/ Information Gain
3) Pruning Tree

Random Forest from Predrag Radenković, Electrical Engineering Dept. University of Belgrade

March 29, 2023

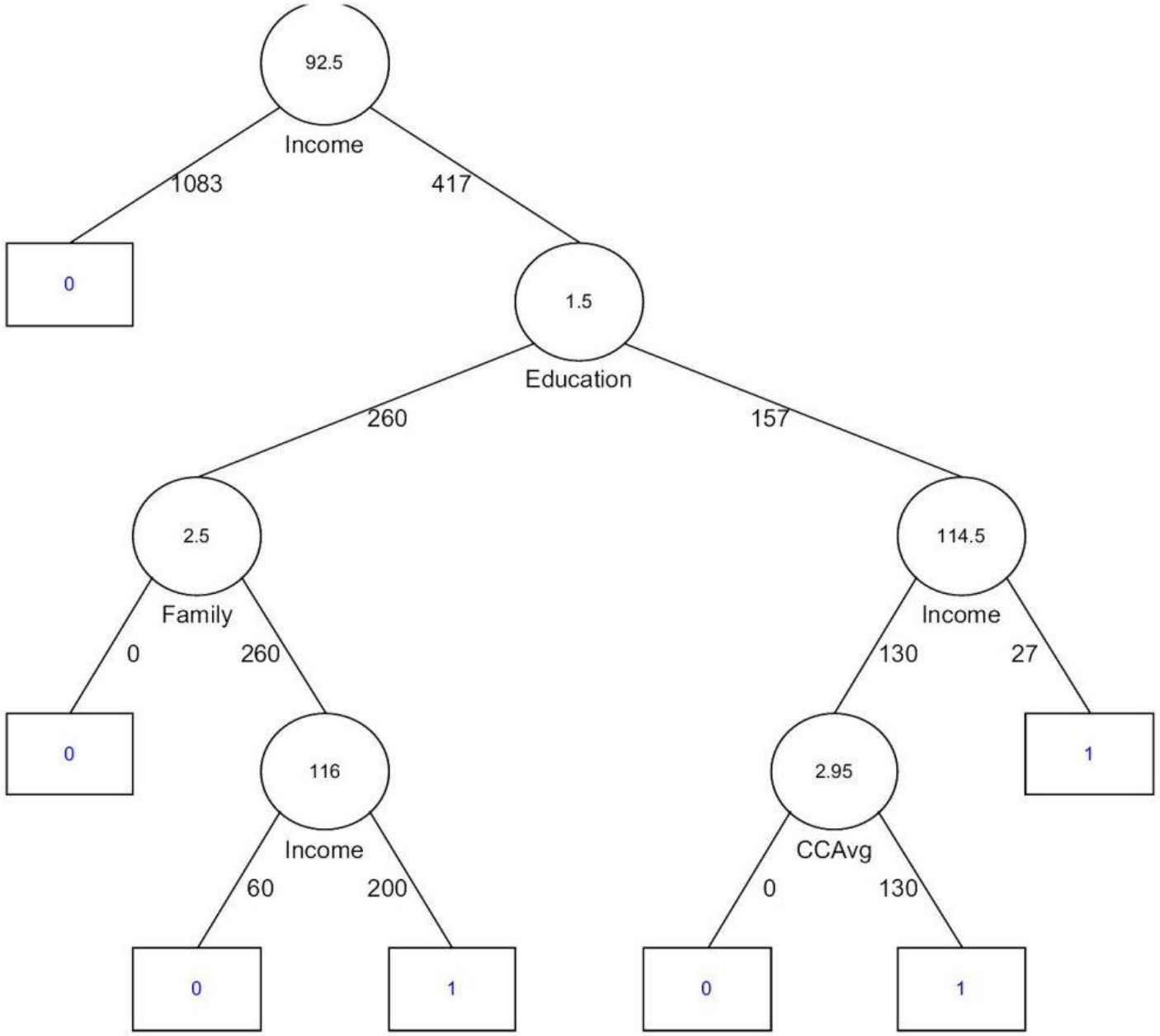
Trees and Rules

Goal: Classify or predict an outcome based on a set of predictors

The output is a set of **rules**

Example:

- Goal: classify a record as “will accept credit card offer” or “will not accept credit card offer”
- Rule might be “IF (Income > 92.5K) AND (Education < 1.5) AND (Family <= 2.5) THEN Class = 0 (non-acceptor)
- Also called CART, Decision Trees, or just Trees
- Rules are represented by tree diagrams


92.5
Income
1083
417
0
1.5
Education
260
157
2.5
Family
0
260
0
116
Income
60
200
0
1
114.5
Income
130
27
2.95
CCAvg
0
130
0
1
1

Key Ideas

Recursive partitioning: Repeatedly split the records into two parts so as to achieve **maximum homogeneity** within the new parts

Pruning the tree: Simplify the tree by pruning peripheral branches to avoid overfitting

Recursive Partitioning Steps

- Pick one of the predictor variables, x_i
- Pick a value of x_i, say s_i, that divides the training data into two (not necessarily equal) portions
- Measure how "pure" or homogeneous each of the resulting portions are

  "Pure" = containing records of mostly one class
- Algorithm tries different values of x_i, and s_i to maximize purity in initial split
- After you get a "maximum purity" split, repeat the process for a second split, and so on

Example: Riding Mowers

- Goal: Classify 24 households as owning or not owning riding mowers
- Predictors = Income, Lot Size

Income	Lot_Size	Ownership
60.0	18.4	owner
85.5	16.8	owner
64.8	21.6	owner
61.5	20.8	owner
87.0	23.6	owner
110.1	19.2	owner
108.0	17.6	owner
82.8	22.4	owner
69.0	20.0	owner
93.0	20.8	owner
51.0	22.0	owner
81.0	20.0	owner
75.0	19.6	non-owner
52.8	20.8	non-owner
64.8	17.2	non-owner
43.2	20.4	non-owner
84.0	17.6	non-owner
49.2	17.6	non-owner
59.4	16.0	non-owner
66.0	18.4	non-owner
47.4	16.4	non-owner
33.0	18.8	non-owner
51.0	14.0	non-owner
63.0	14.8	non-owner

How to split

- Order records according to one variable, say lot size
- Find **midpoints** between successive values
  E.g. first midpoint is 14.4 (halfway between 14.0 and 14.8)
- Divide records into those with lotsize > 14.4 and those < 14.4
- After evaluating that split, try the next one, which is 15.4 (halfway between 14.8 and 16.0)

Note: Categorical Variables

- Examine all possible ways in which the categories can be split.

- E.g., categories A, B, C can be split 3 ways

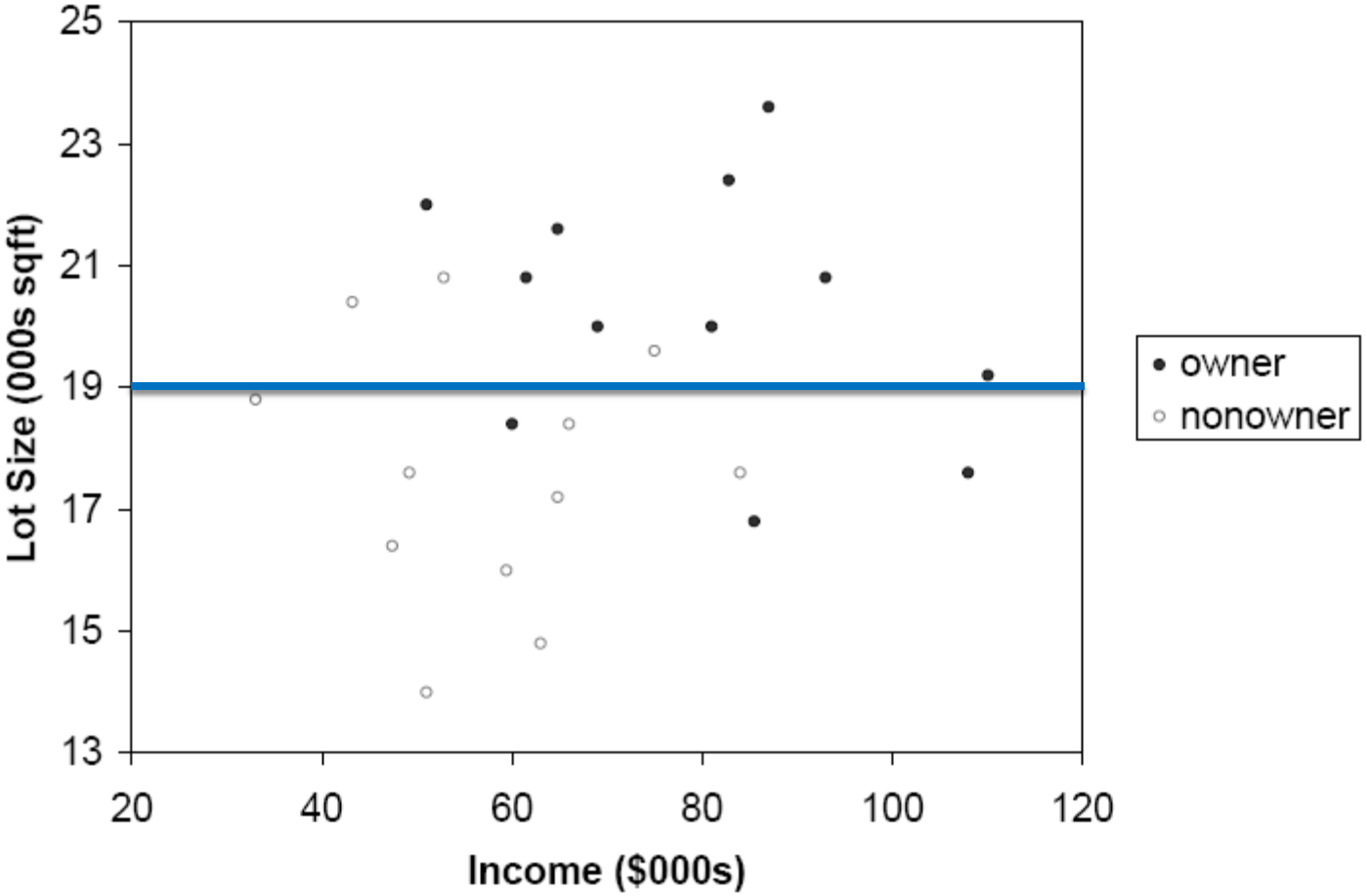
  {A} and {B, C}

  {B} and {A, C}

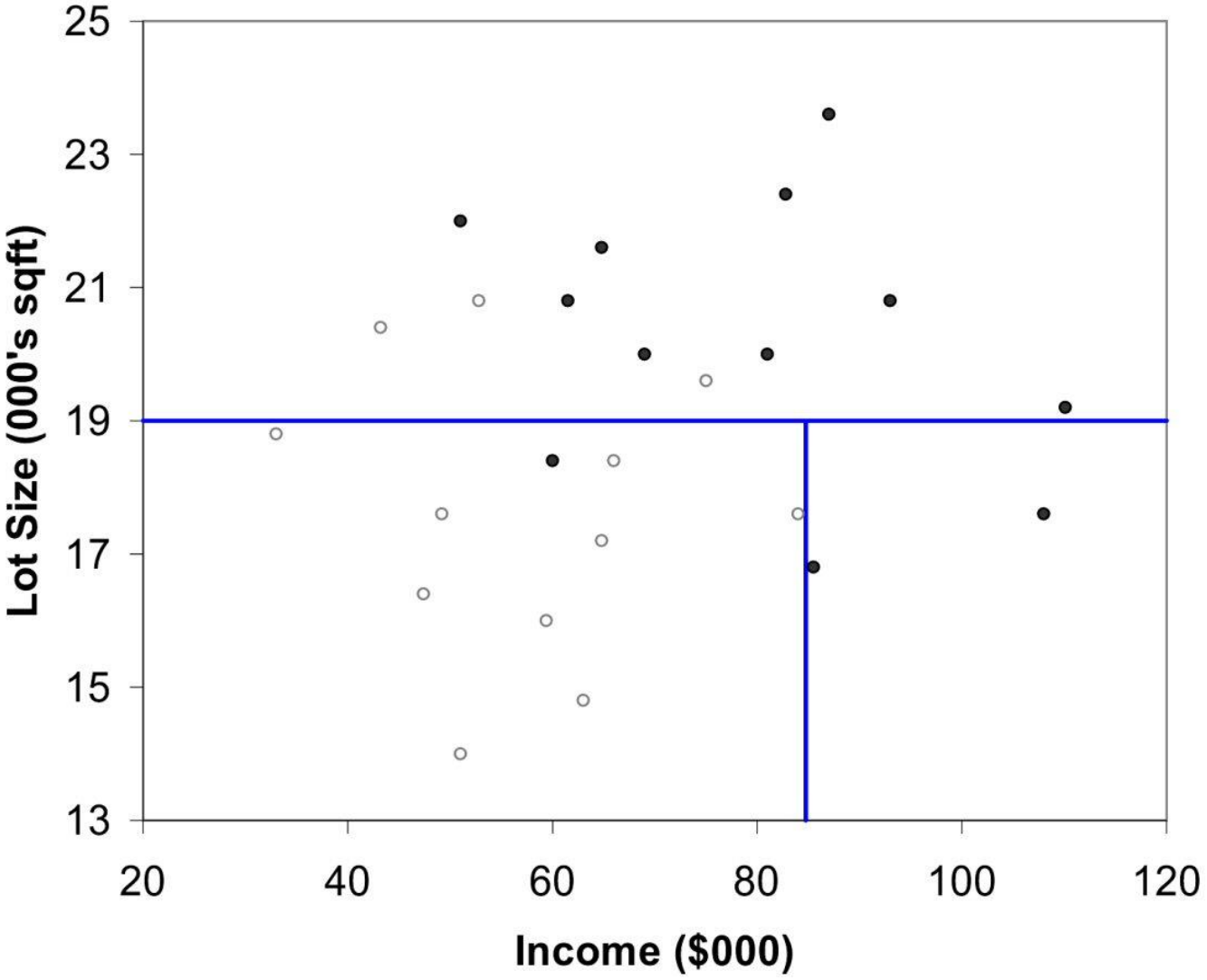
  {C} and {A, B}

- With many categories, # of splits becomes huge

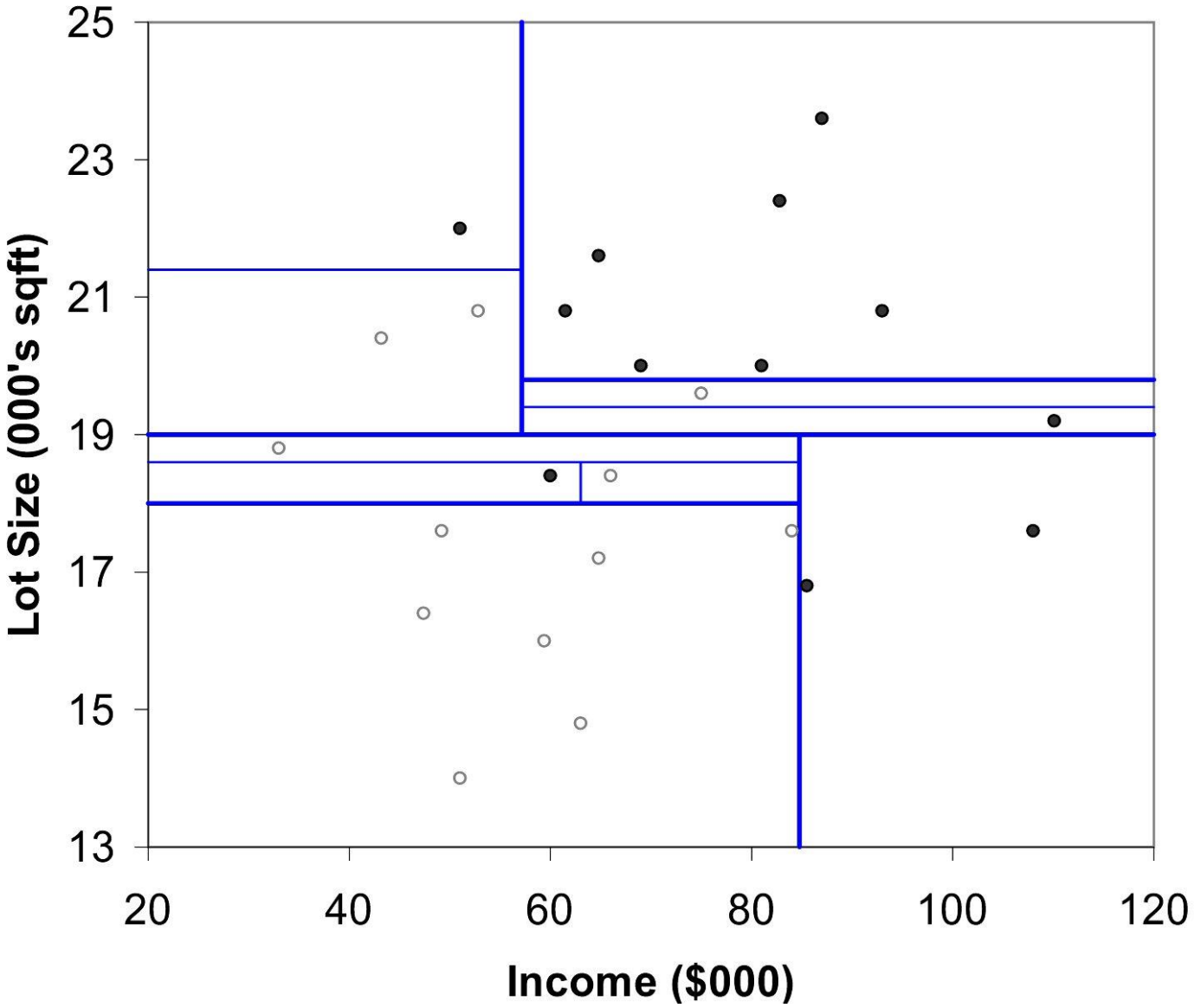
The first split: Lot Size = 19



Second Split: Income = $84,000



After All Splits



Measuring Reduction in Entropy (ID3/C4.5)

$$entropy\,(A) = -\sum_{k=1}^{m} p_k \log_2(p_k)$$

p_k = proportion of cases (out of m) in rectangle A that belong to class k

- Entropy ranges between 0 (most pure) and 1 (equal representation of classes)

Measuring Impurity - Gini Index (CART)

Gini Index for rectangle *A* containing *m classes*

$$GI(A) = 1 - \sum_{k=1}^{m} p_k^2$$

p_k = proportion of cases in rectangle *A* that belong to class *k*

- GI(A) = 0 when all cases belong to same class
- Max value when all classes are equally represented (= 0.50 in binary case)

Impurity and Recursive Partitioning

- Obtain overall impurity measure (**weighted avg. of individual rectangles**)
- At each successive stage, compare this measure across **all possible splits in all variables**
- Choose the split that reduces impurity the most
- Chosen split points become nodes on the tree

Tree Structure

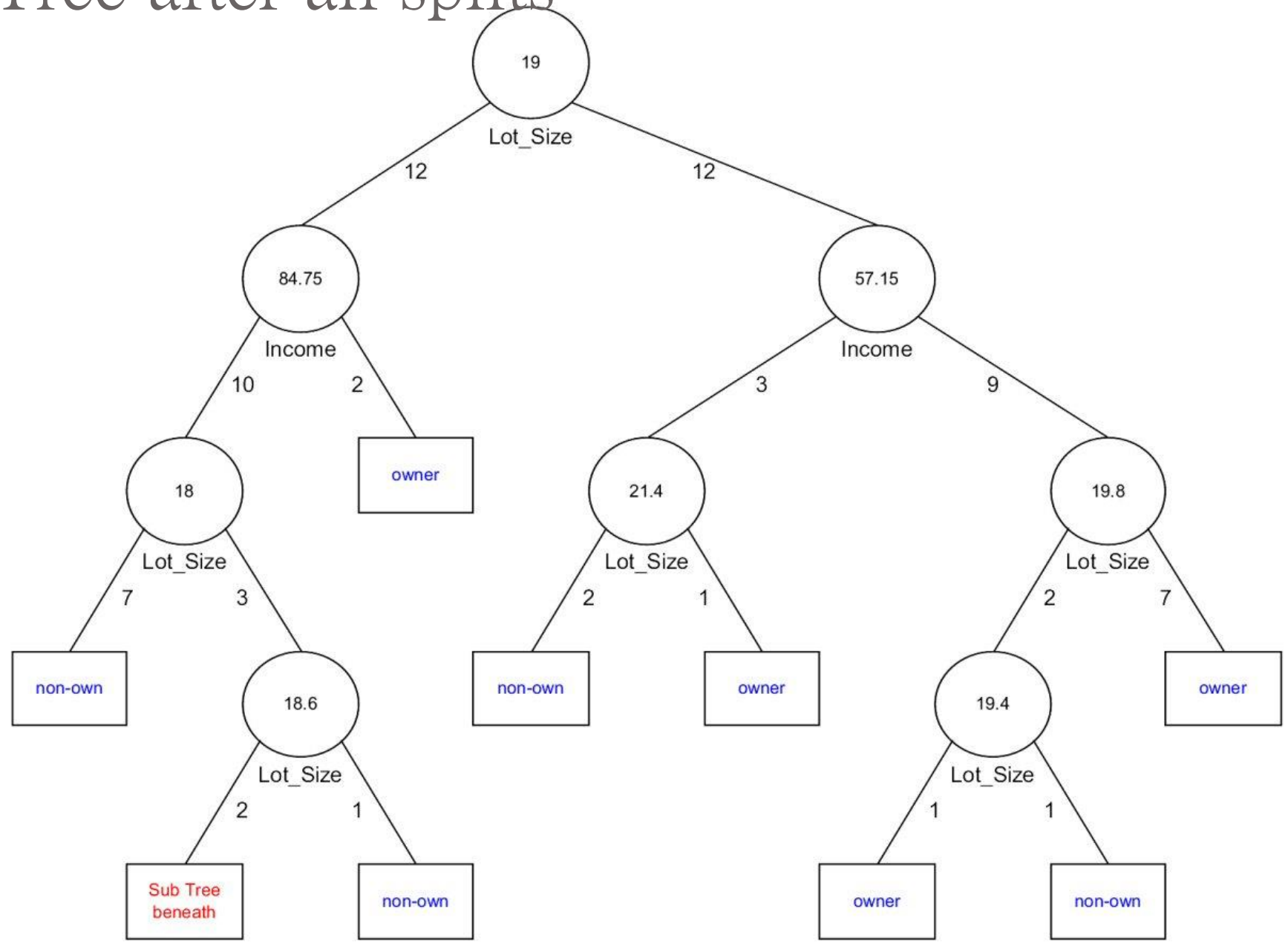
- Split points become nodes on tree (circles with split value in center)
- Rectangles represent “leaves” (terminal points, no further splits, classification value noted)
- Numbers on lines between nodes indicate # cases
- Read down tree to derive rule

  E.g., If lot size < 19, and if income > 84.75, then class = “owner”

Determining Leaf Node Label

- Each leaf node label is determined by “voting” of the records within it, and by the cutoff value
- Records within each leaf node are from the training data
- Default cutoff=0.5 means that the leaf node’s label is the majority class.

Tree after all splits

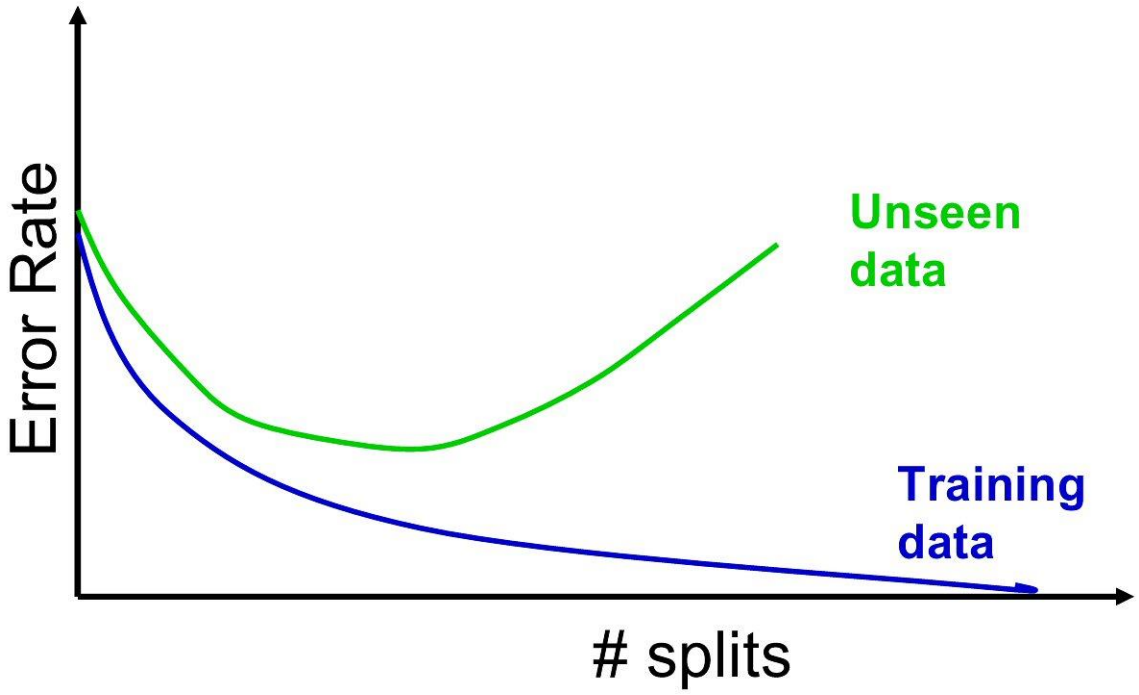


The Overfitting Problem

Stopping Tree Growth

- Natural end of process is 100% purity in each leaf
- This **overfits** the data, which end up fitting noise in the data
- Overfitting leads to low predictive accuracy of new data
- Past a certain point, the error rate for the validation data starts to increase

Full Tree Error Rate



Pruning

- CART lets tree grow to full extent, then prunes it back
- Idea is to find that point at which the **validation error begins to rise**
- Generate successively smaller trees by **pruning leaves**
- At each pruning stage, multiple trees are possible
- Use ***cost complexity*** to choose the best tree at that stage

Regression Trees

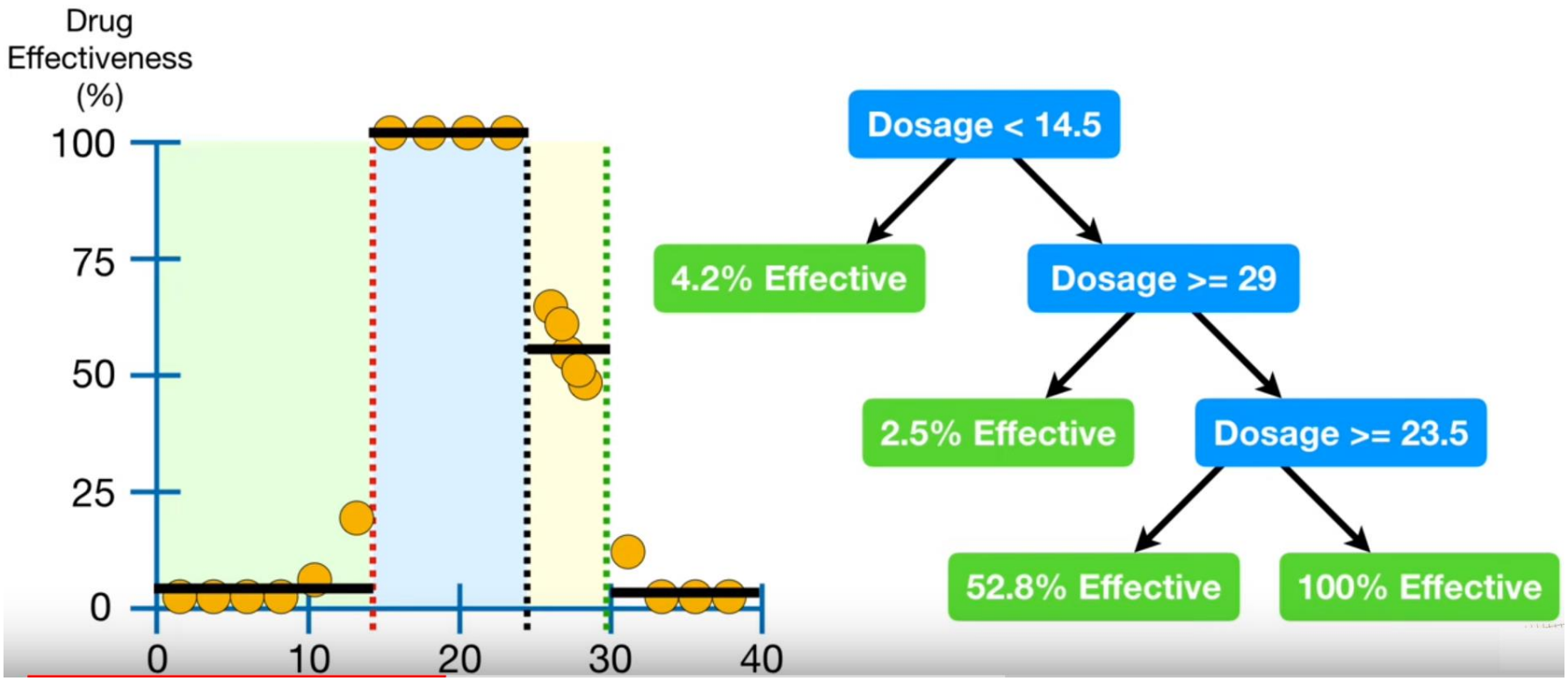
Regression Trees for Prediction

- Used with continuous outcome variable
- Procedure similar to classification tree(CT)
- Many splits attempted, choose the one that minimizes impurity

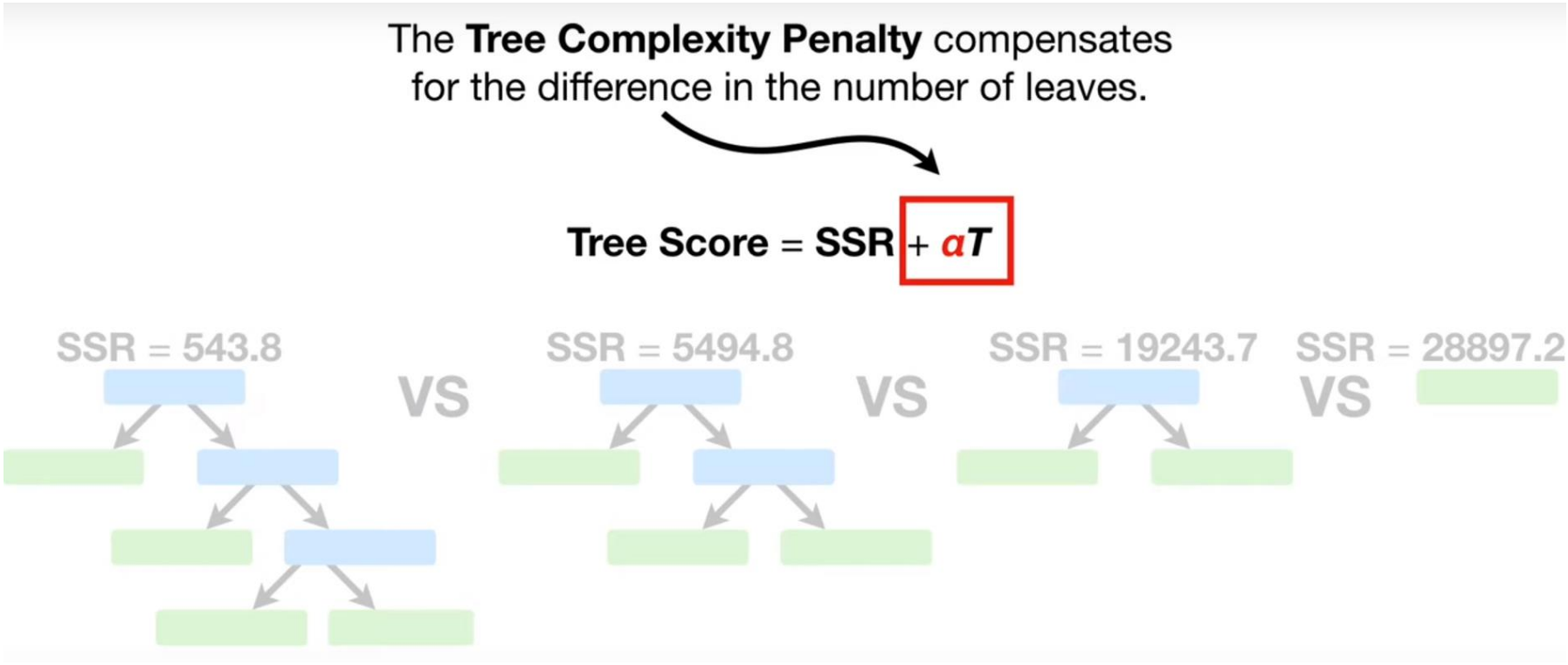
Differences from CT

- Prediction is computed as the **average** of numerical target variable in the rectangle (in CT it is majority vote)
- Impurity measured by **sum of squared residuals (deviations)** from leaf mean
- Performance measured by RMSE (root mean squared error)

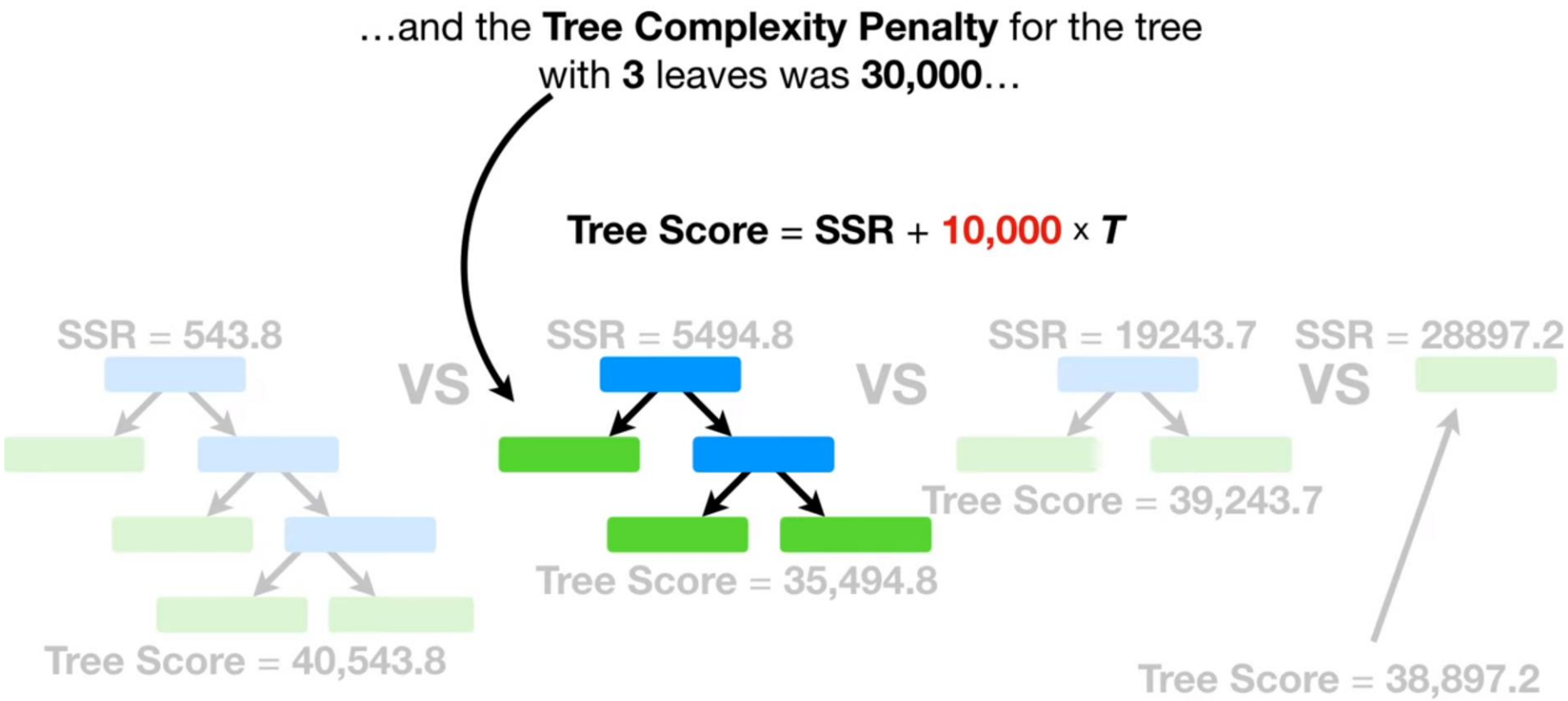
Cost complexity pruning (Weakest link pruning)



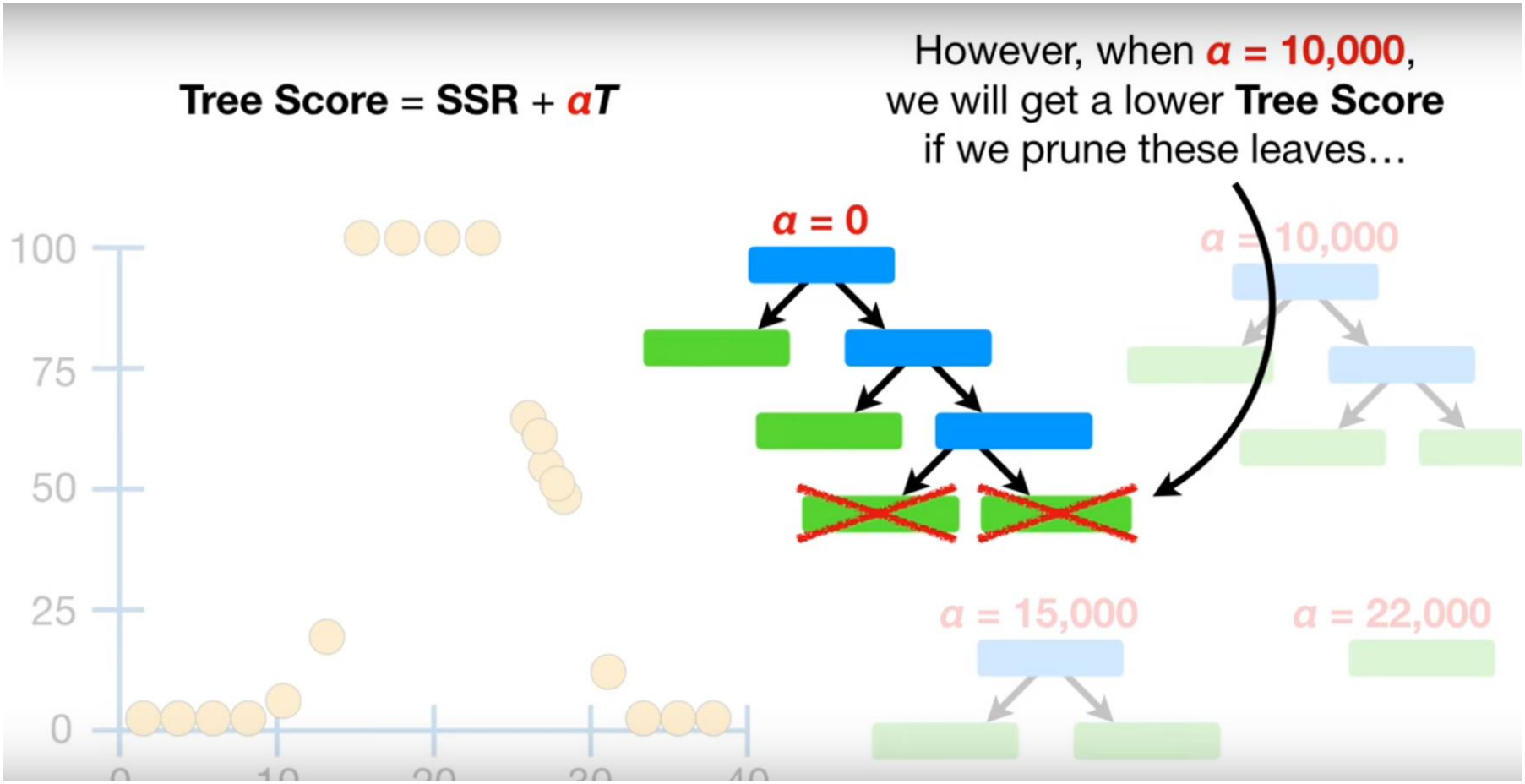
https://www.youtube.com/watch?v=D0efHEJsfHo



https://www.youtube.com/watch?v=D0efHEJsfHo

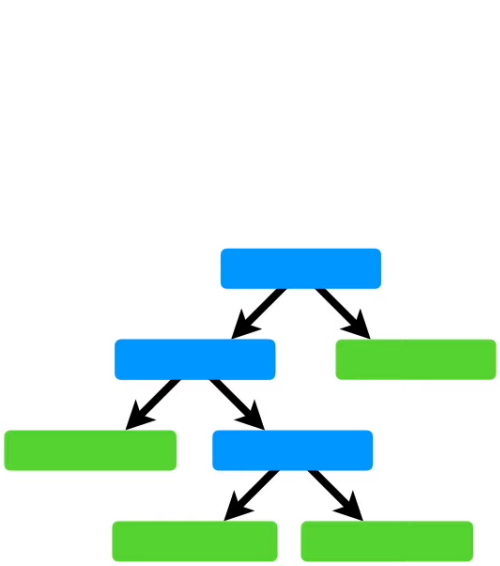


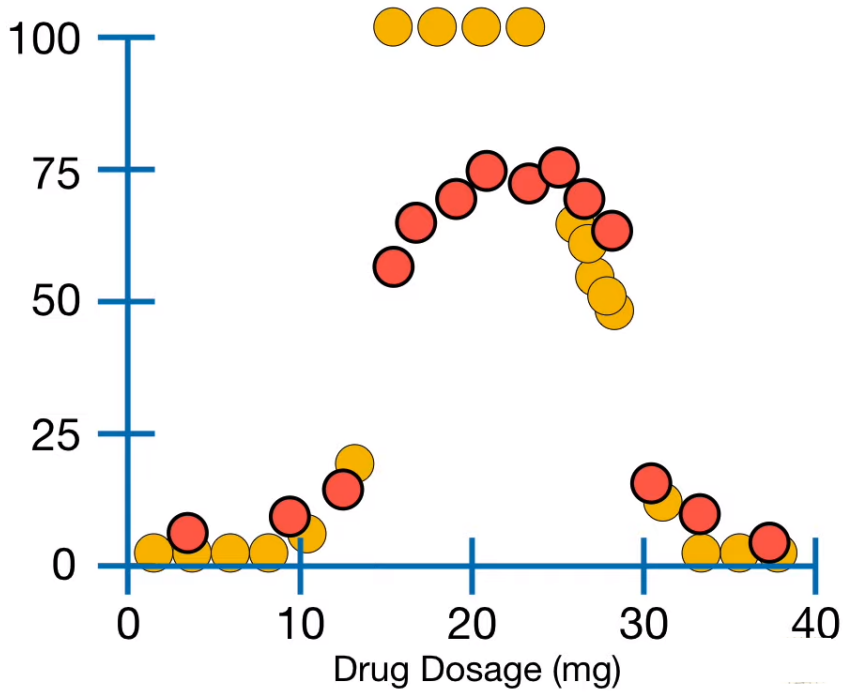
https://www.youtube.com/watch?v=D0efHEJsfHo


Tree Score = SSR + αT
However, when α = 10,000,
we will get a lower Tree Score
if we prune these leaves…
α = 0
α = 10,000
α = 15,000
α = 22,000
100
75
50
25
0
0
10
20
30
40

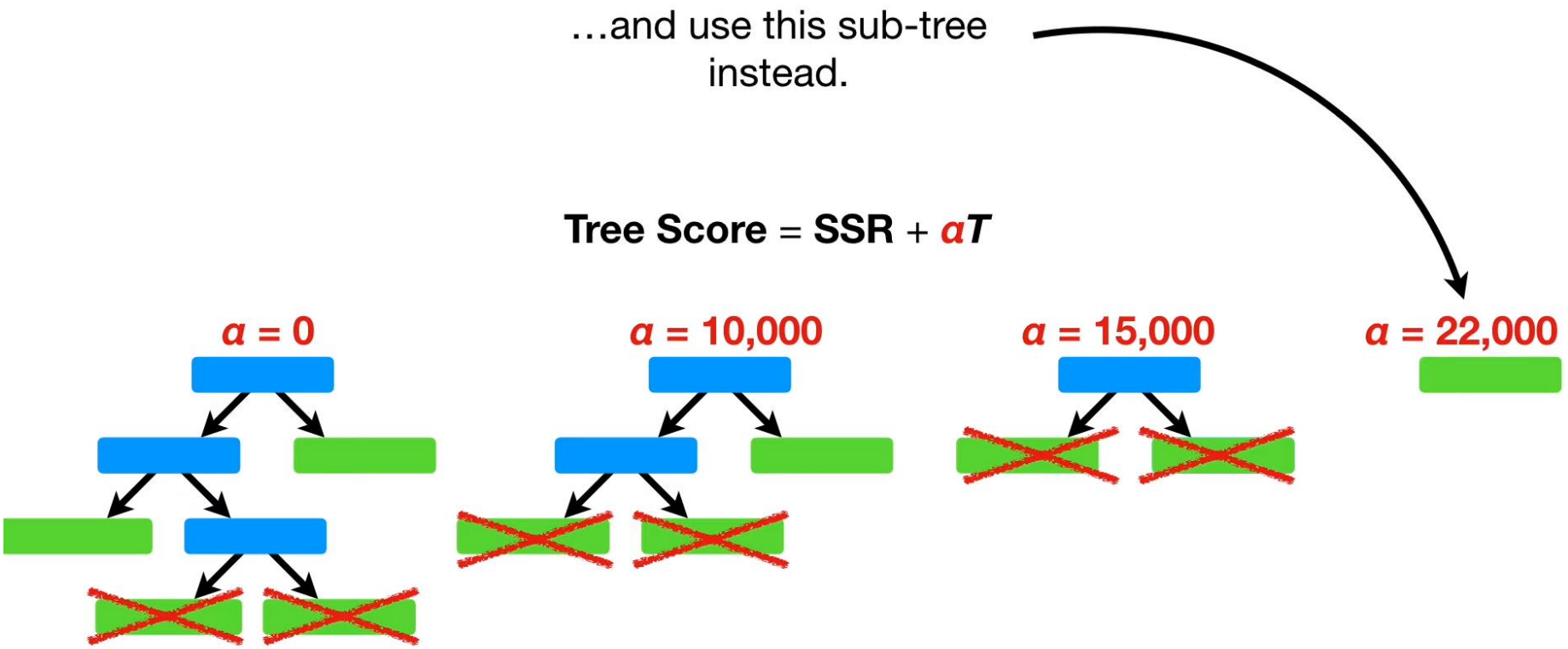
How to find the best alpha-value?

NOTE: This full sized tree is different than before because it was fit to ***all*** of the data, not just the **Training Data**.

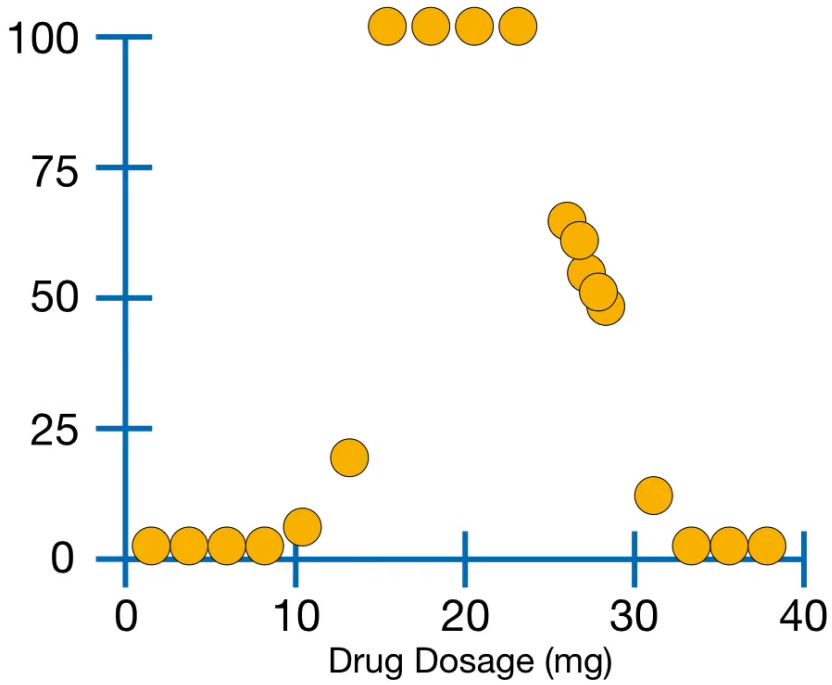




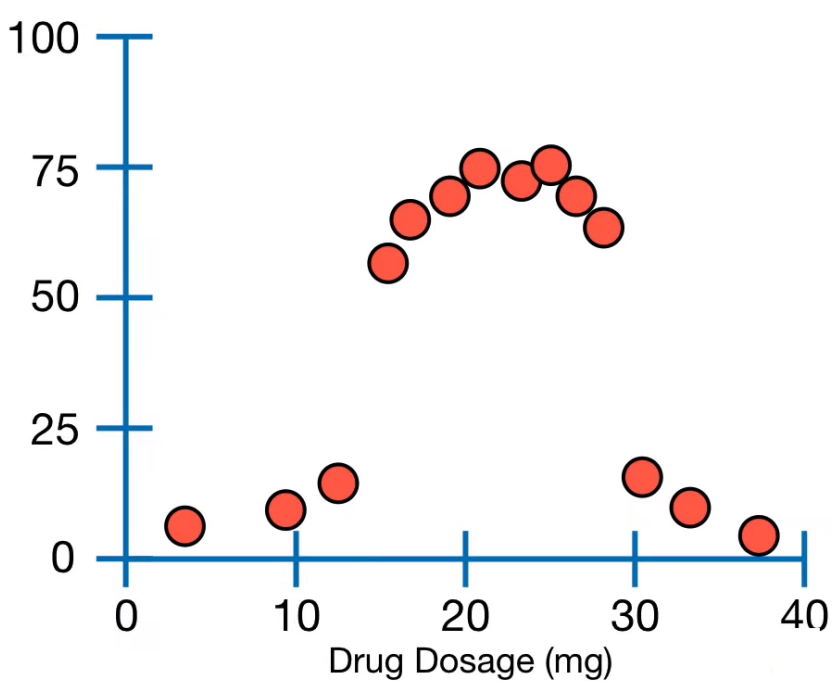
Keep increasing alpha-value until pruning leaves would give us a lower Tree Score



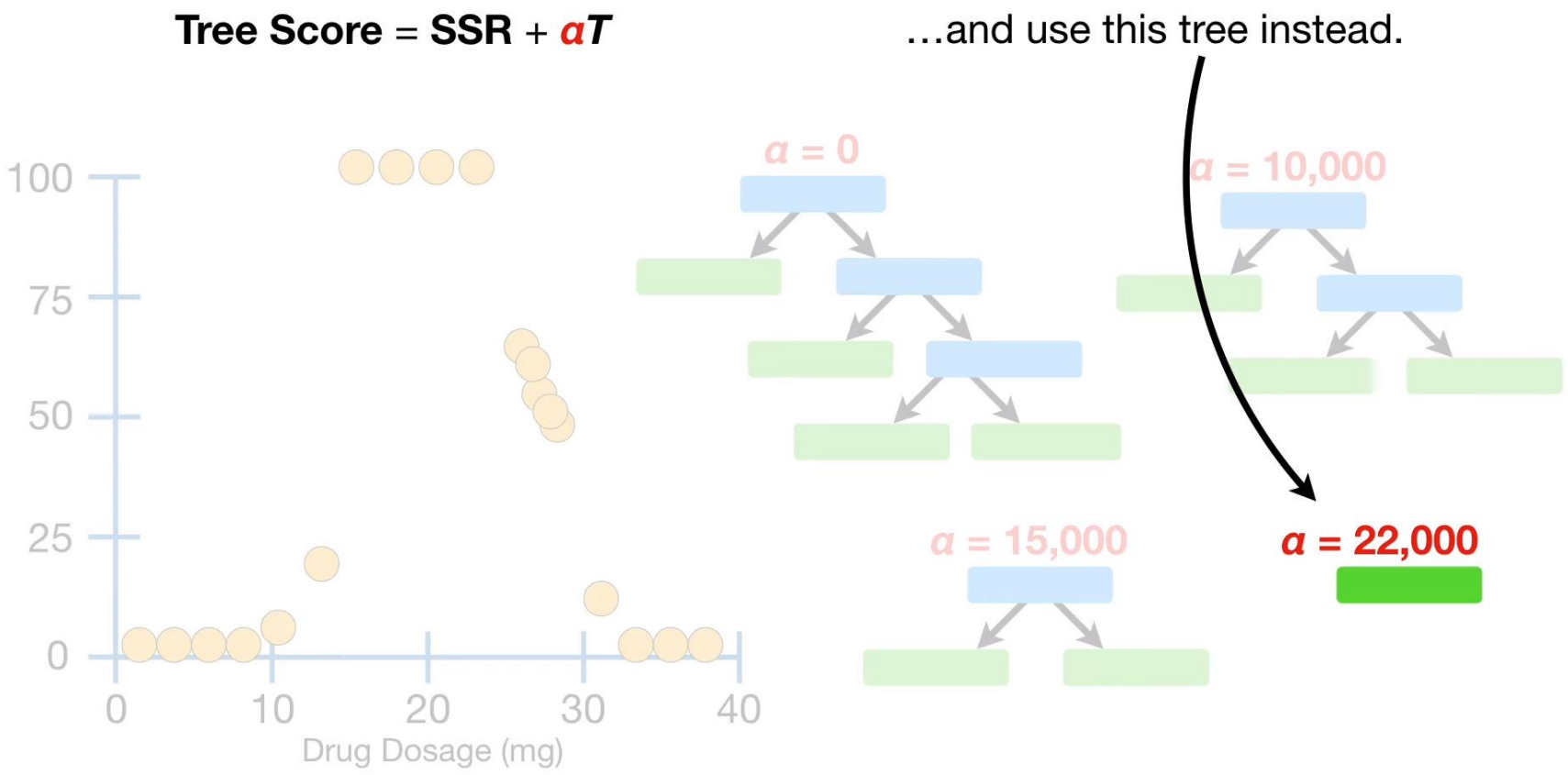
…and divide it into **Training**…



…and **Testing Datasets**.

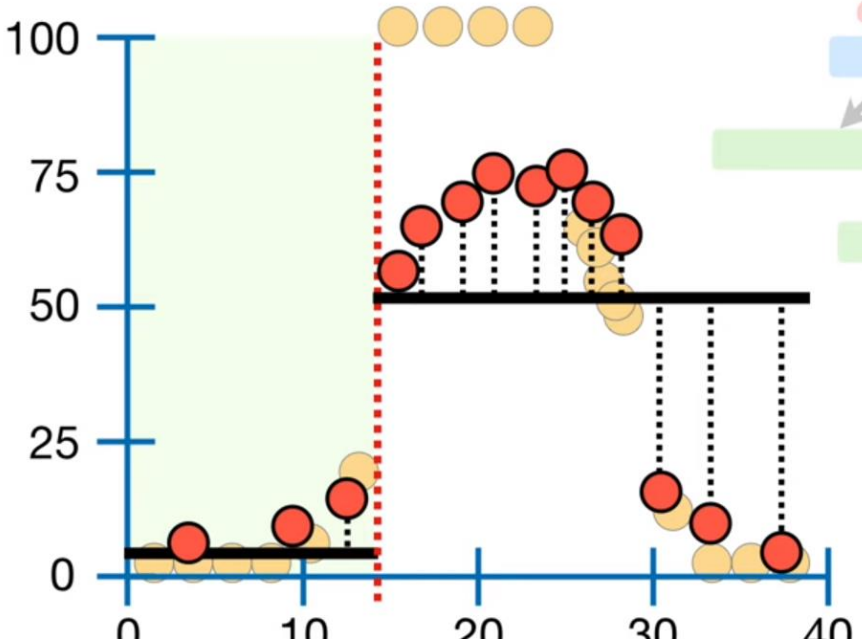


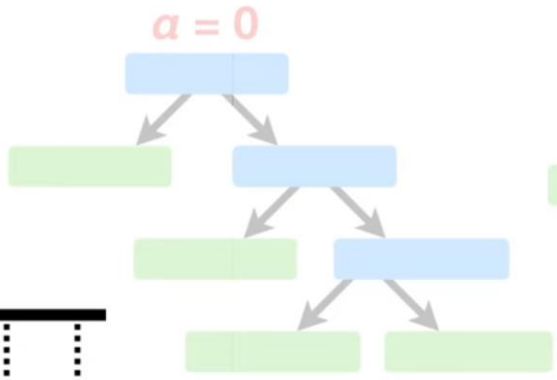
Using found alpha-values to create Trees

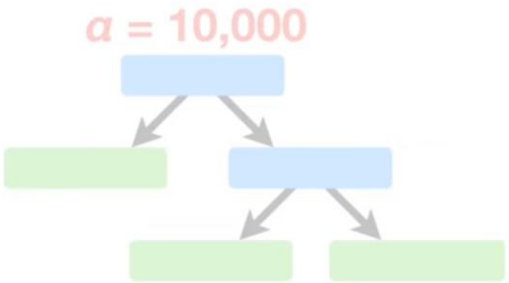


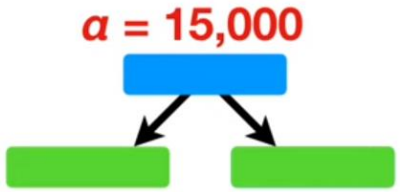
Using testing to calculate SSRs

Now calculate the **Sum of Squared Residuals** for each new tree using only the **Testing Data**.



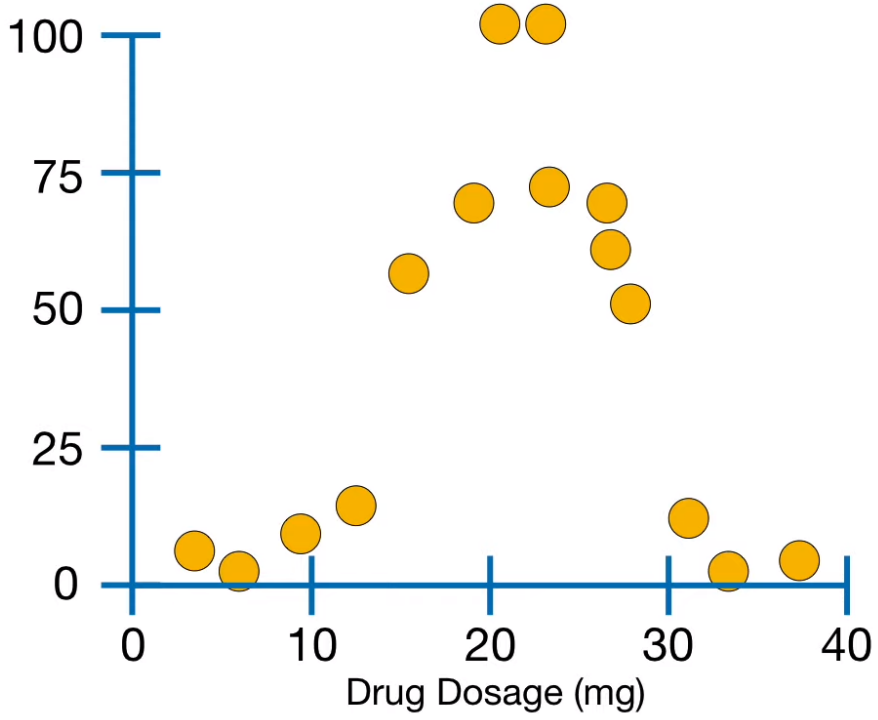




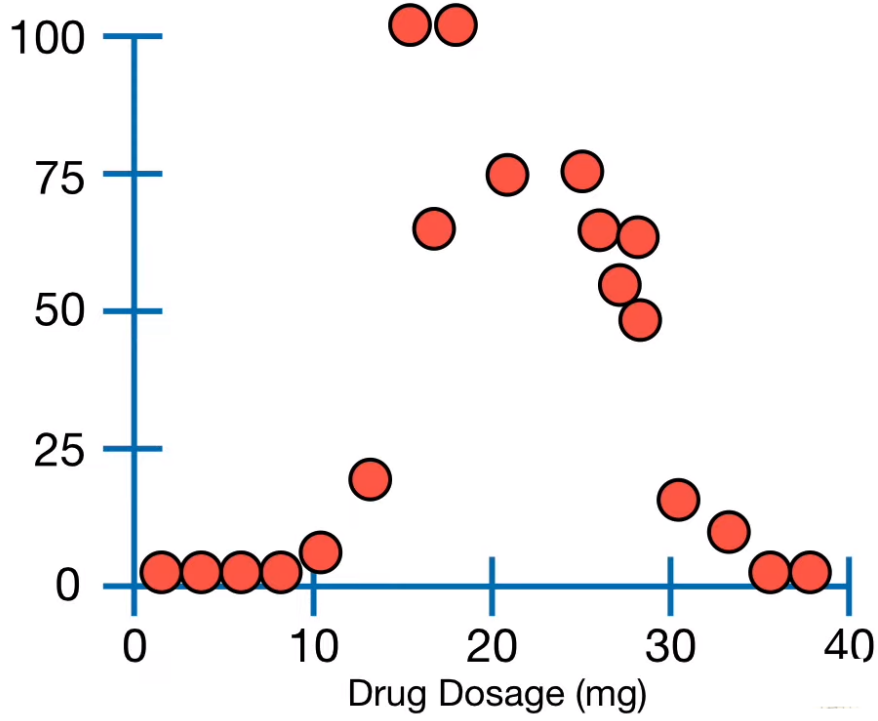


Perform k-fold simulations

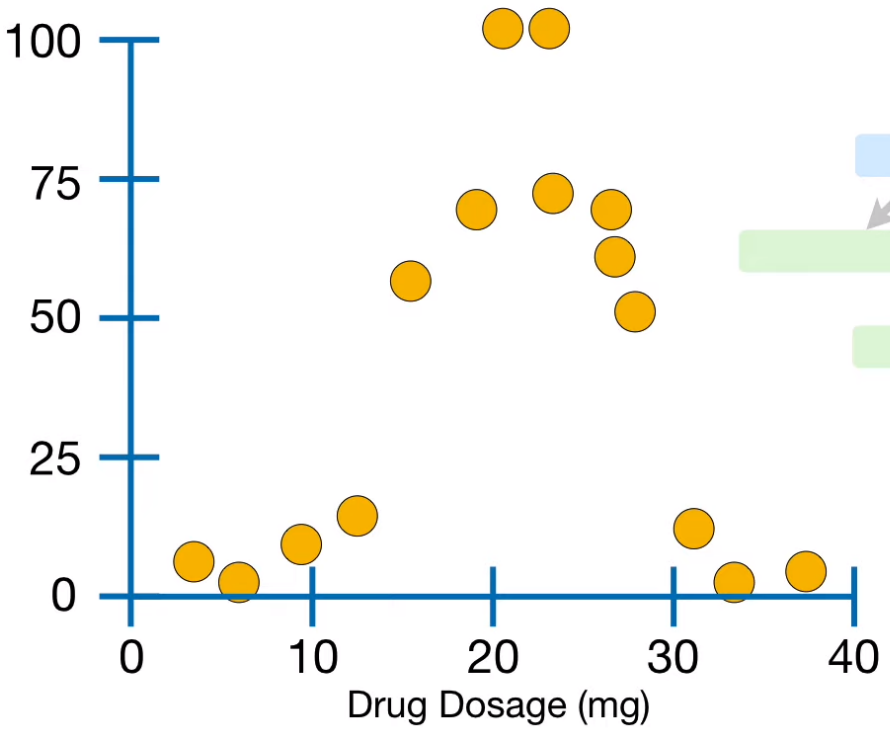
Now we go back and create new **Training Data**…



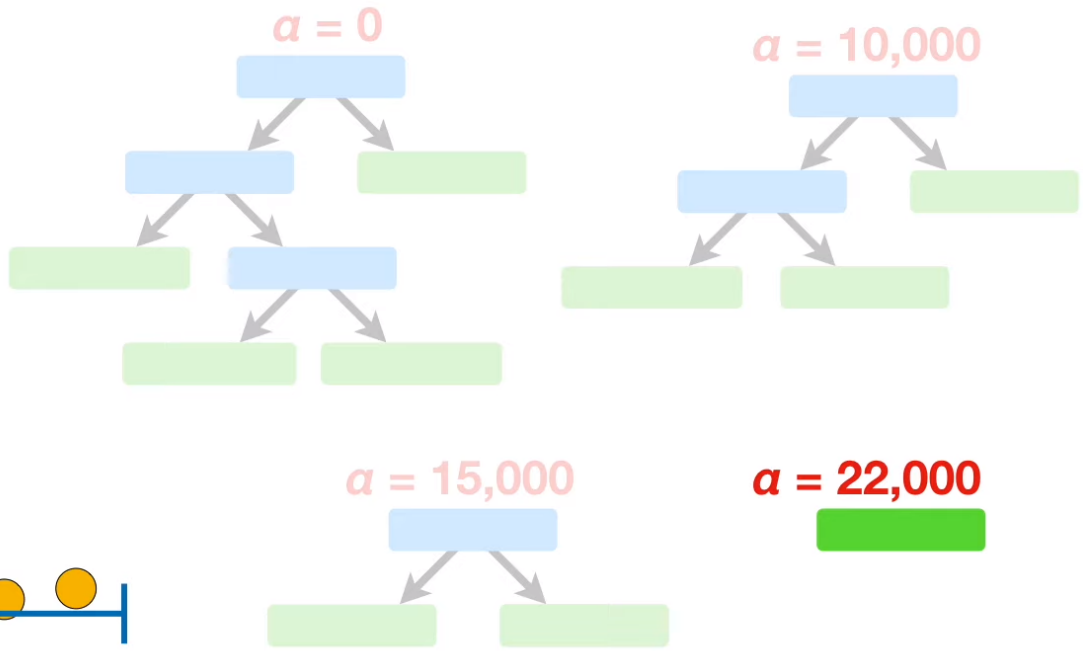
…and new **Testing Data**.

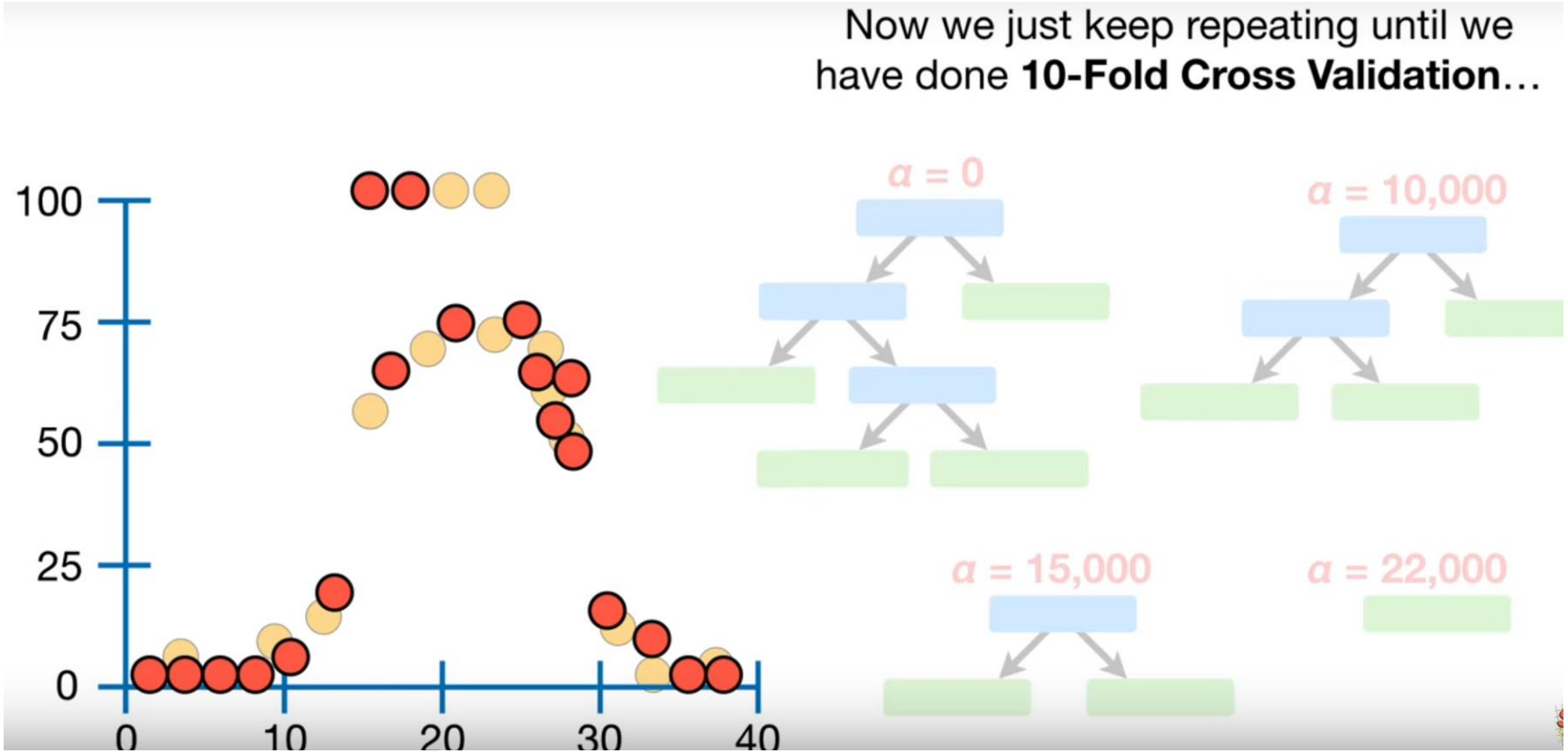


And just using the new **Training Data**…

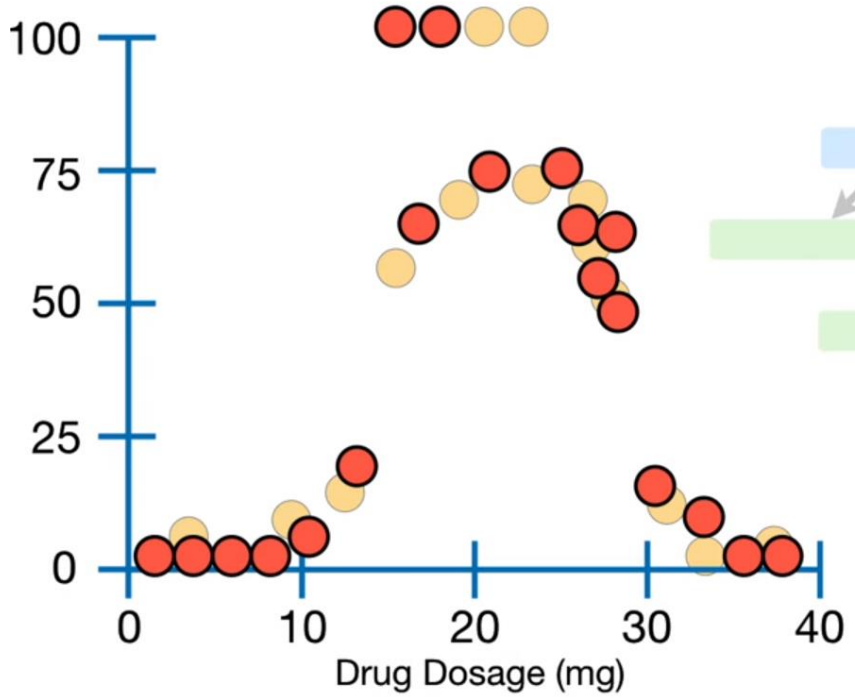


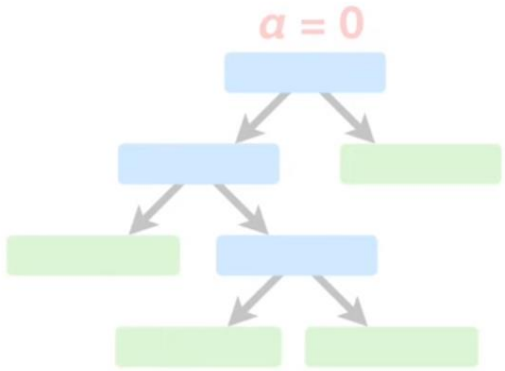
…build a new sequence of trees, from full sized to a leaf, using the α values we found before.

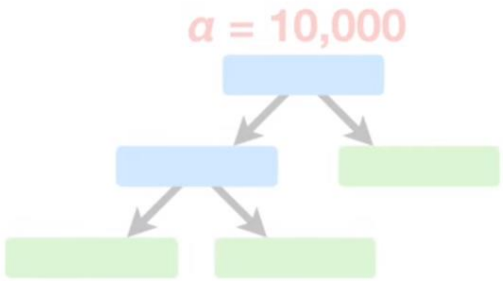


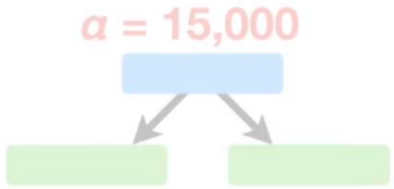

Now we just keep repeating until we have done **10-Fold Cross Validation**…
100
75
50
25
0
0
10
20
30
40
α = 0
α = 10,000
α = 15,000
α = 22,000

…and the value for α that, on average, gave us the lowest **Sum of Squared Residuals** with the **Testing Data**, is the final value for α.

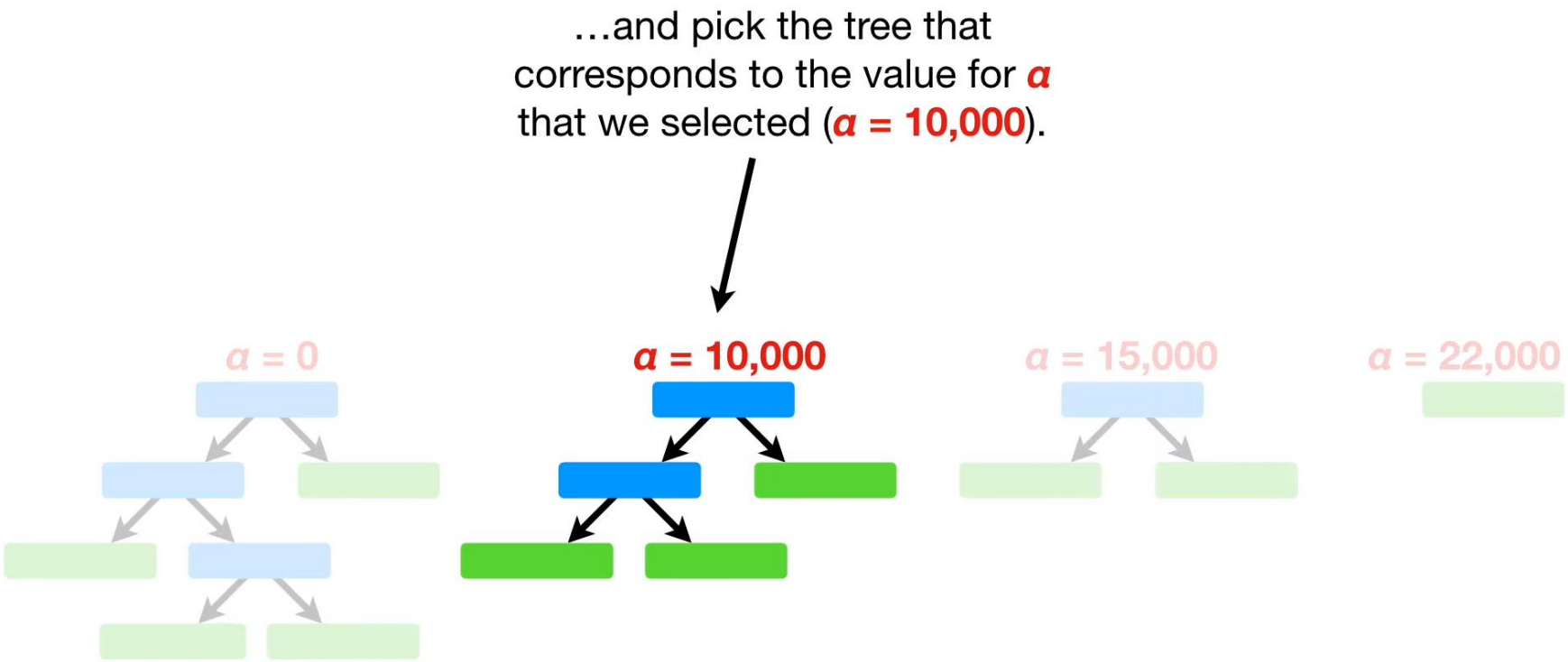








Use the selected alpha-value to complete the cost complexity pruning



Random forest > Bagging

- Bagging: Bootstrap aggregation
- Technique of ensemble learning...
  - ... to avoid over-fitting
    - Important since trees are unpruned
  - ... to improve stability and accuracy
- Two steps
  - Bootstrap sample set
  - Aggregation

Random forest > Bagging > Bootstrap

- L: original learning set composed of p samples
- Generate K learning sets L_k...
  - ... composed of q samples, $q \leq p$,...
  - ... obtained by uniform sampling with replacement from L
  - In consequences, L_k may contain repeated samples
- Random forest: q = p
  - Asymptotic proportion of unique samples in L_k = 100 (1 – 1/e) ~ 63%
  - → The remaining samples can be used for testing

Example of Random Forest

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease

To create a bootstrapped dataset that is the same size as the original, we just randomly select samples from the original dataset.

The important detail is that we're allowed to pick the same sample more than once.

https://www.youtube.com/watch?v=J4Wdy0Wc_xQ

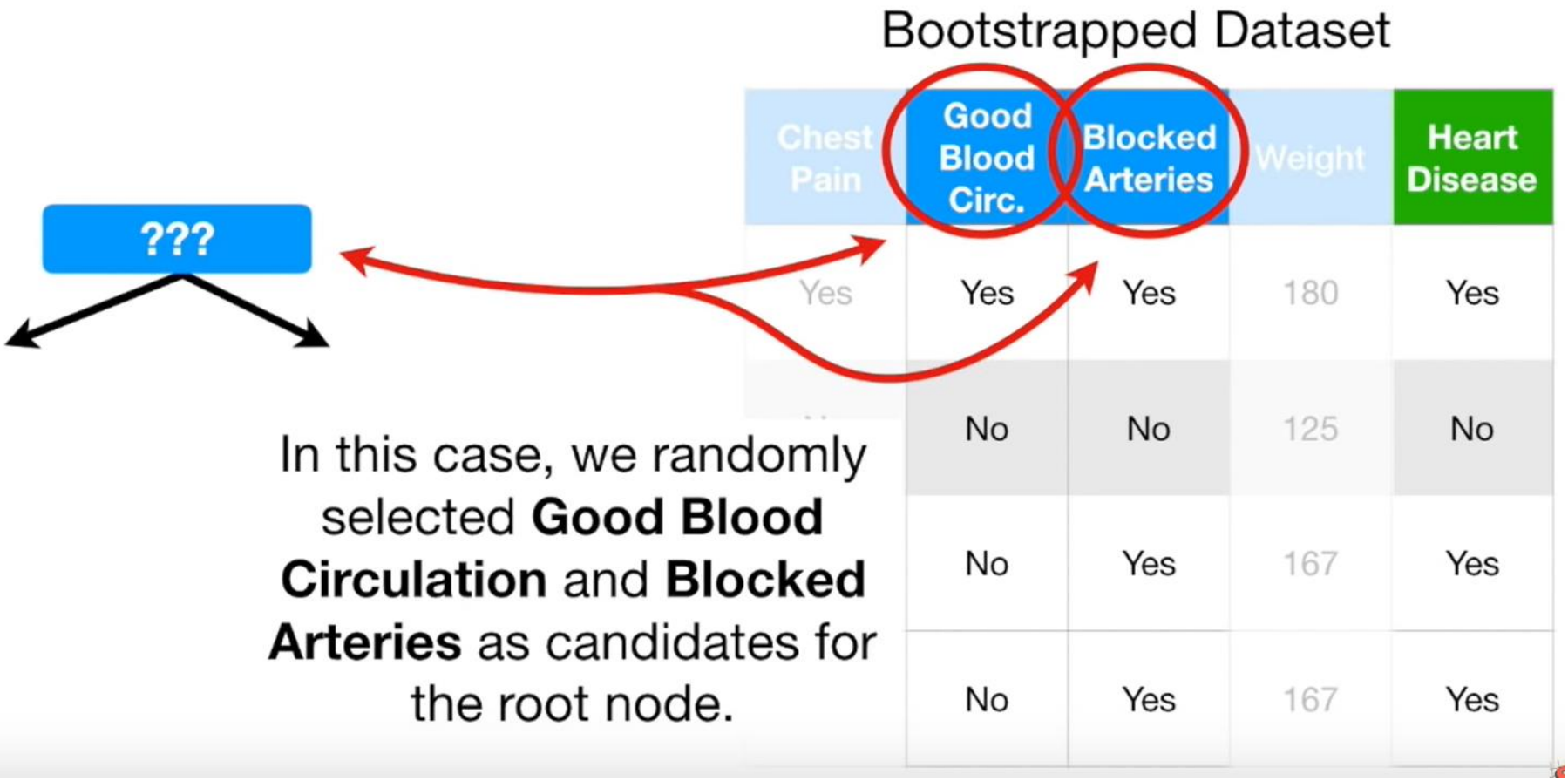
Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

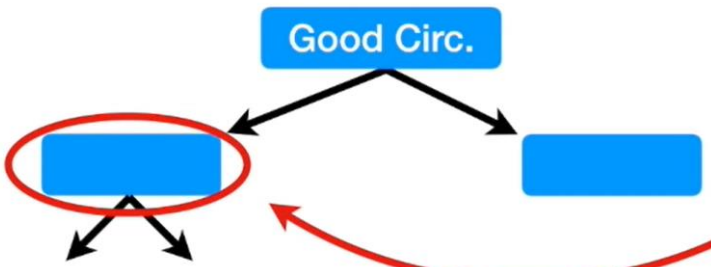
Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

https://www.youtube.com/watch?v=J4Wdy0Wc_xQ



https://www.youtube.com/watch?v=J4Wdy0Wc_xQ



Bootstrapped Dataset

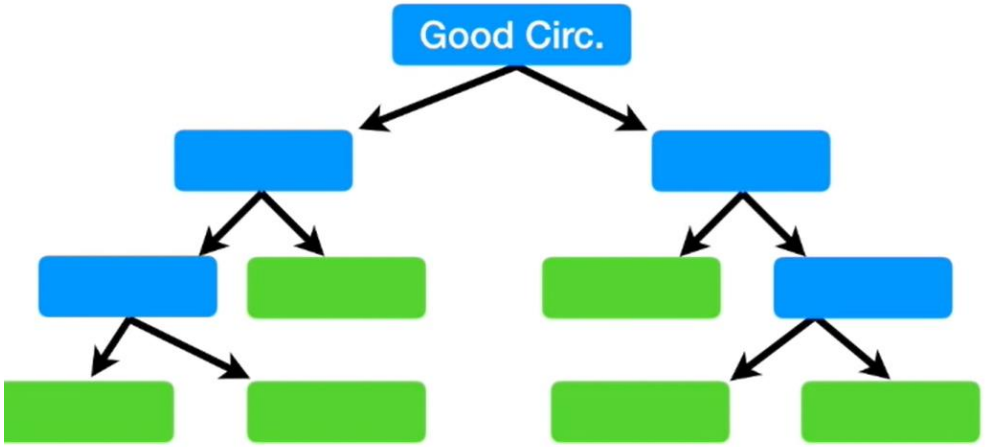
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Just like for the root, we randomly select 2 variables as candidates, instead of all 3 remaining columns.

We built a tree…

1) Using a bootstrapped dataset

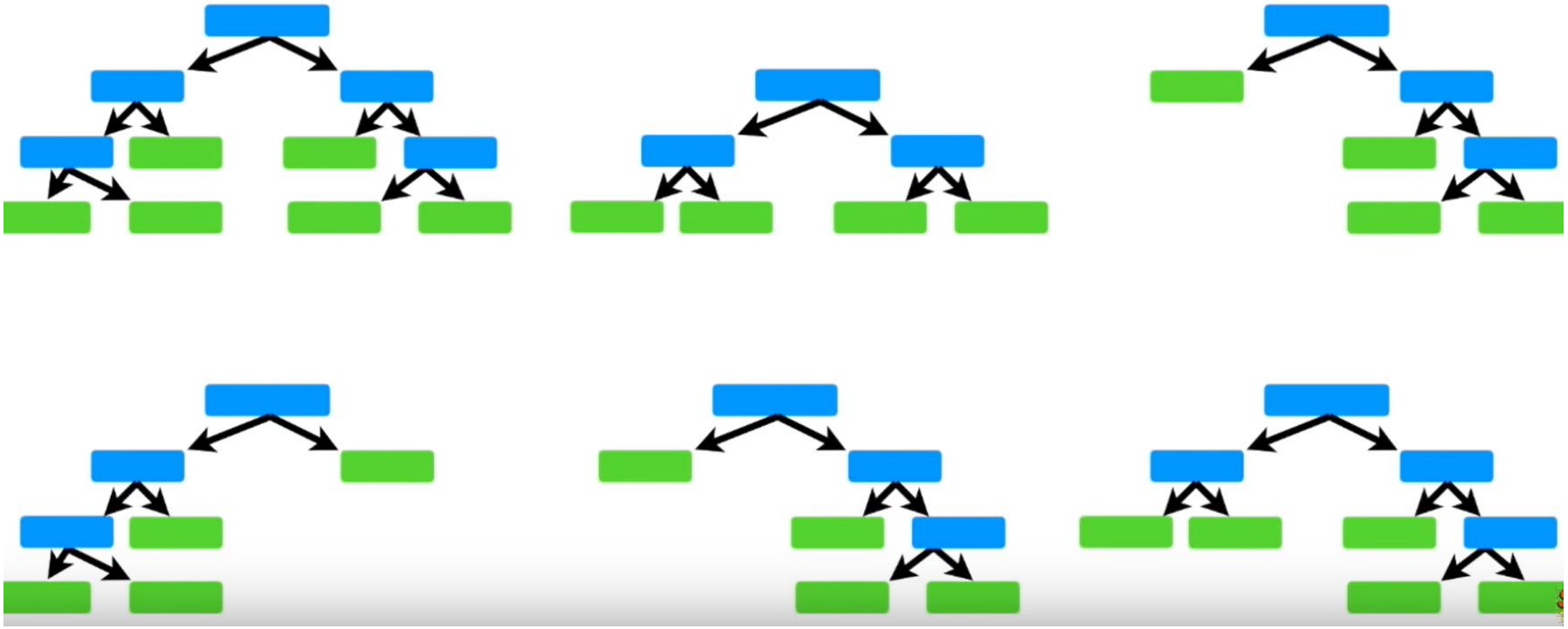
2) Only considering a random a subset of variables at each step.



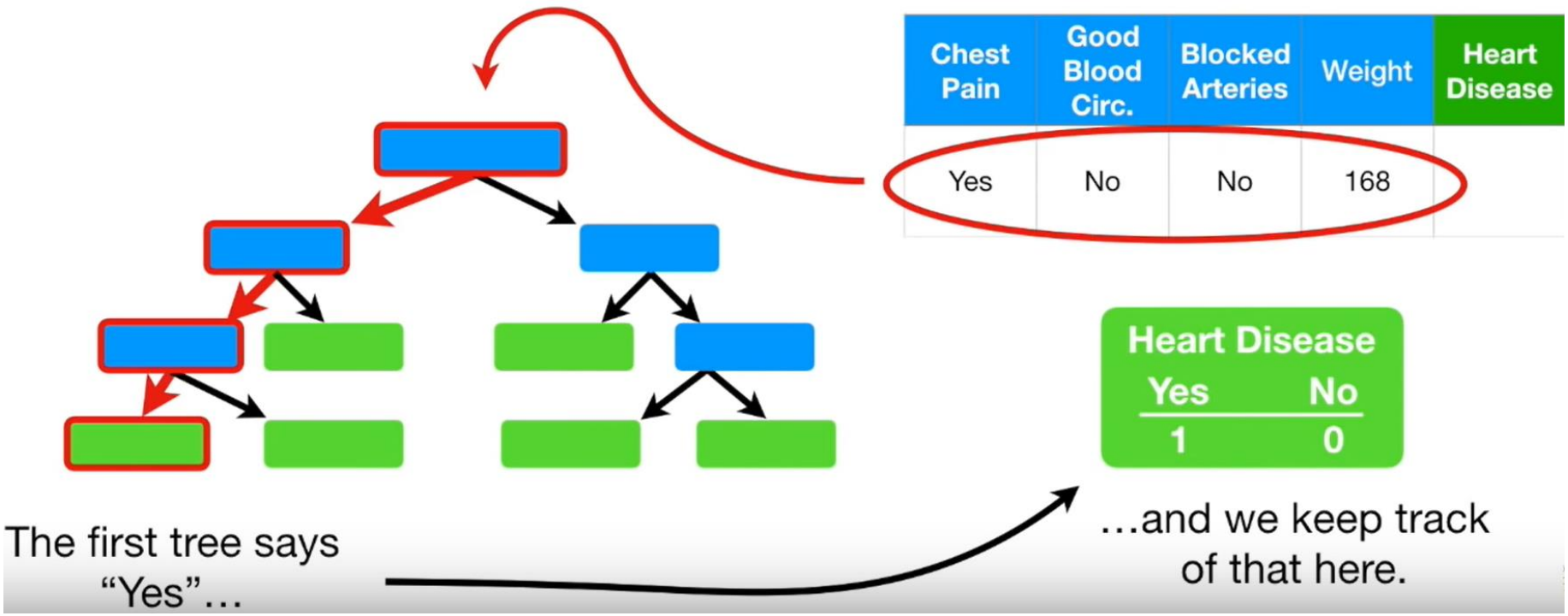
Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

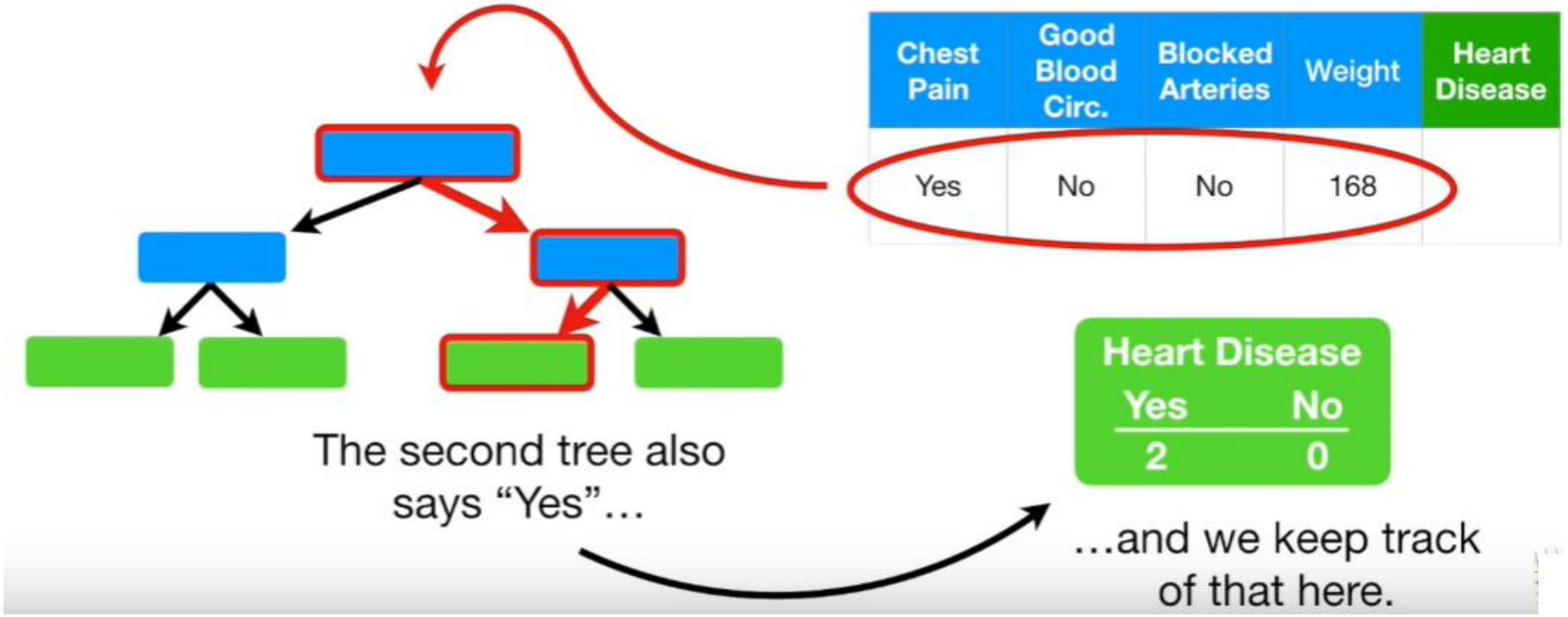
Now go back to Step 1 and repeat: Make a new bootstrapped dataset and build a tree considering a subset of variables at each step.



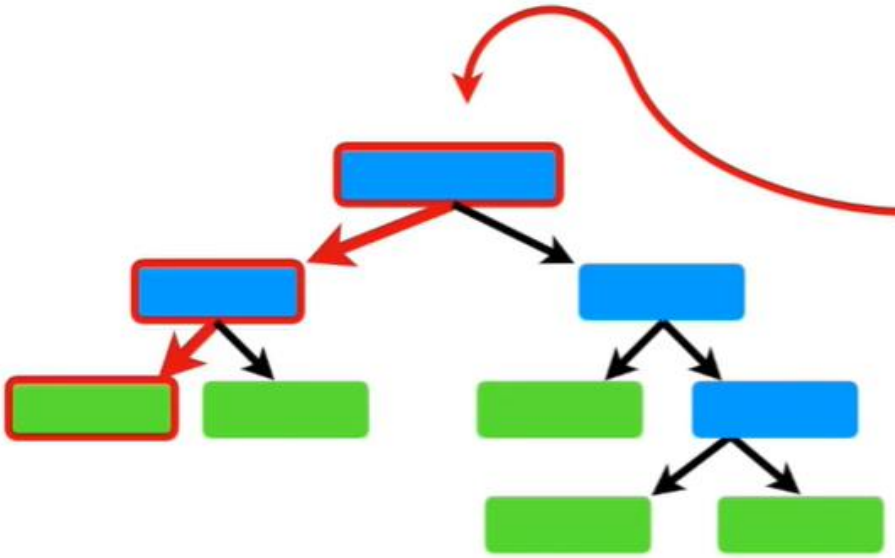
A new testing data sample for the first subtree



The new testing data sample for the second subtree



Then we repeat for all the trees that we made…



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

Heart Disease	
Yes	No
5	1

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	**YES**

In this case, "**Yes**" received the most votes, so we will conclude that this patient has heart disease.

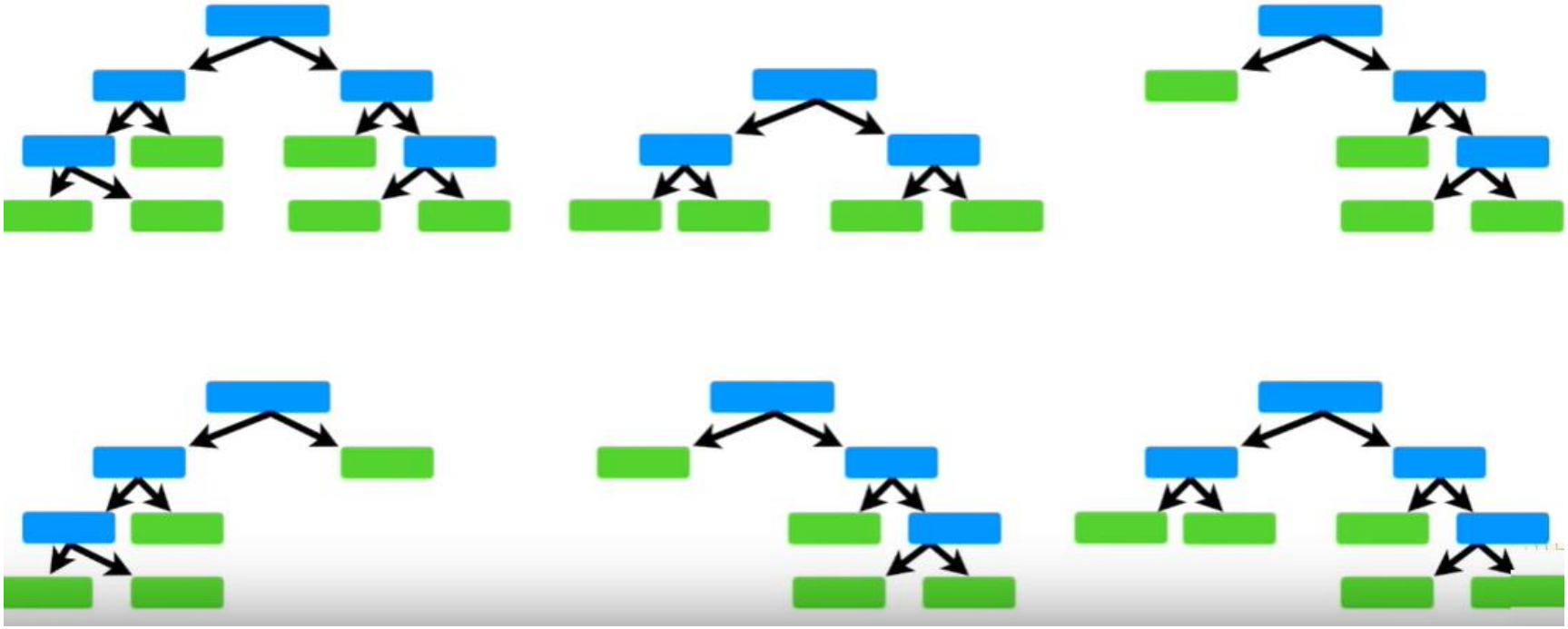
Terminology Alert!!!

Bootstrapping the data plus using the **agg**regate to make a decision is called "**Bagging**"

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	**YES**

Heart Disease	
Yes	No
5	1

How do we know if it's any good?



As a result, this entry was not included in the bootstrapped dataset.

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Typically, about 1/3 of the original data does not end up in the bootstrapped dataset.

Original Dataset

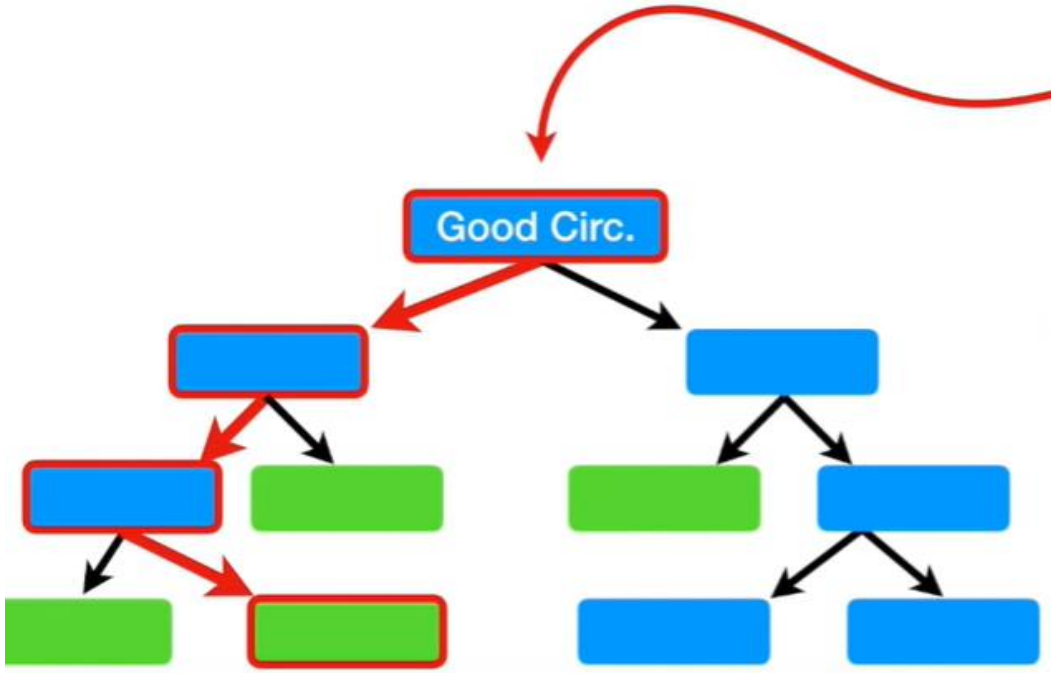
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

(psst… if the original dataset were larger, we'd have more than just 1 entry over here…)

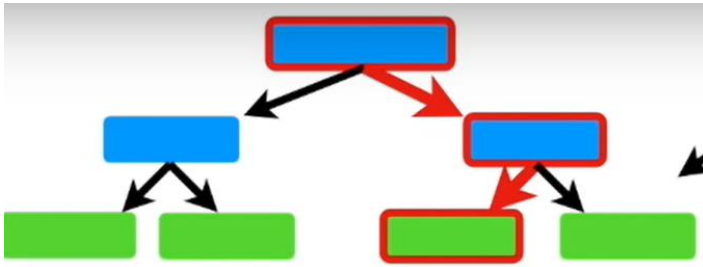
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

This is called the **"Out-Of-Bag Dataset"**

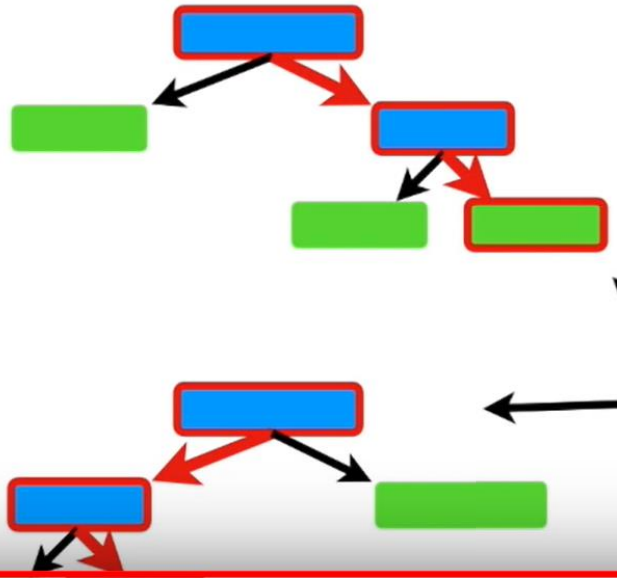
…we can run it through and see if it correctly classifies the sample as "No Heart Disease"



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No



This tree incorrectly labeled the Out-of-Bag sample "**Yes**".



These trees correctly labeled the Out-of-Bag sample "**No**".

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Since the label with the most votes wins, it is the label that we assign this Out-of-Bag sample.

In this case, the Out-of-Bag sample is correctly labeled by the Random Forest.

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Classification of the Out-Of-Bag sample	
Yes	No
4	0

Classification of the Out-Of-Bag sample	
Yes	No
3	1

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No

This Out-of-Bag sample was *incorrectly* labeled…

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Classification of the Out-Of-Bag sample	
Yes	No
4	0

Classification of the Out-Of-Bag sample	
Yes	No
3	1

etc… etc… etc…

Ultimately, we can measure how accurate our random forest is by the proportion of Out-Of-Bag samples that were correctly classified by the Random Forest.

The proportion of Out-Of-Bag samples that were *incorrectly* classified is the "**Out-Of-Bag Error**"

1) Build a Random Forest

2) Use a Random Forest

3) Estimate the accuracy of a Random Forest.

…to random forest built using 3 variables per step…

…and we test a bunch of different settings and choose the most accurate random forest.

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

…change the number of variables used per step…

1) Build a Random Forest

2) Estimate the accuracy of a Random Forest.

Do this for a bunch of times and then choose the one that is most accurate.

Typically, we start by using the square of the number of variables and then try a a few settings above and below that value.

Missing Data

Random Forests consider 2 types of missing data…

1) Missing data in the original dataset used to create the random forest.

2) Missing data in a new sample that you want to categorize.

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	???	???	No

New Sample

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	???	

https://www.youtube.com/watch?v=nyxTdL_4Q-Q

Missing data in the original dataset for RF

→ make the first guess based on the target value (“No”)

Filled-in Missing Values

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**167.5**	No

Now we want to refine these guesses.

We do this by first determining which samples are similar to the one with missing data.

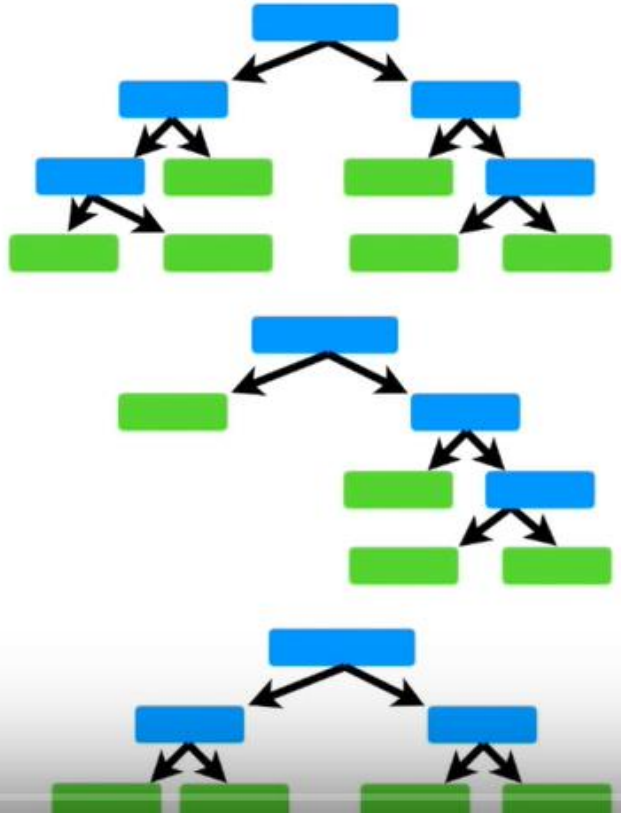
So let’s talk about how to determine similarity…

https://www.youtube.com/watch?v=nyxTdL_4Q-Q

Step 1: Build a random forest…

Filled-in Missing Values

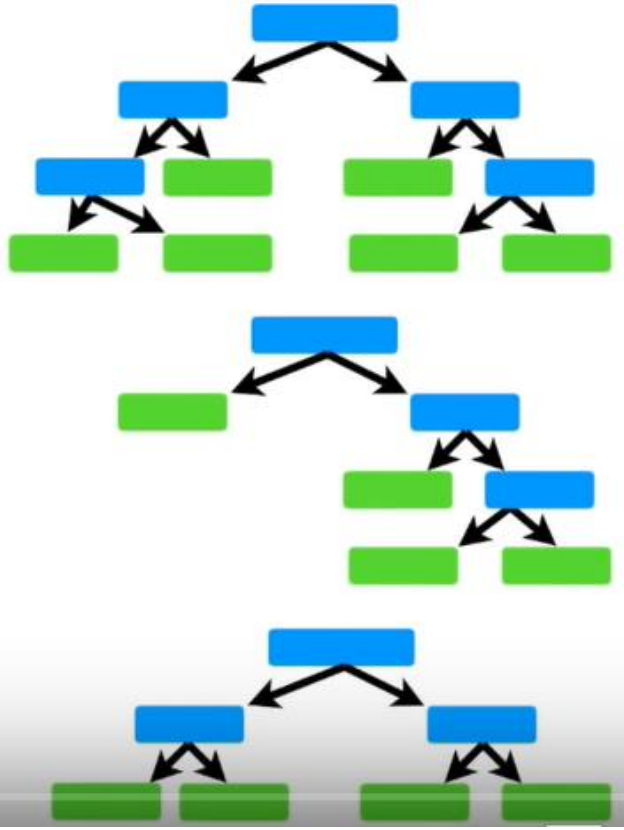
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**167.5**	No



Filled-in Missing Values

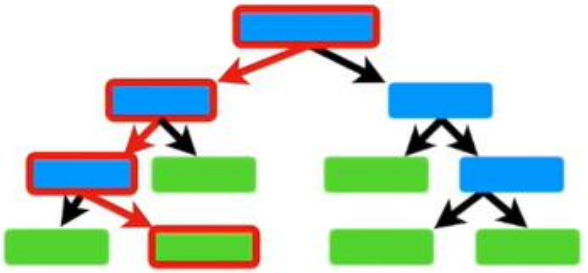
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**167.5**	No

Step 2: Run all of the data down all of the trees.



Filled-in Missing Values

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**167.5**	No



Notice that Sample 3 and Sample 4 both ended up at the same leaf node.

Filled-in Missing Values

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**167.5**	No

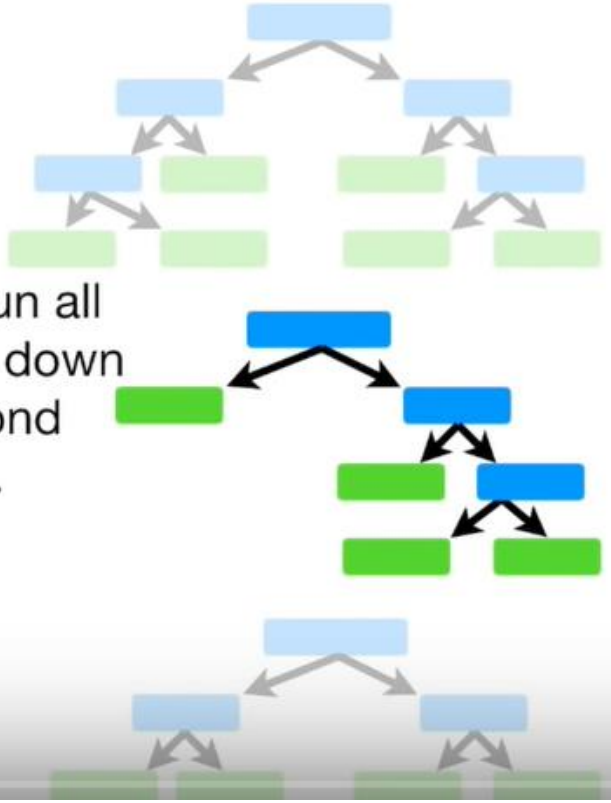
Because no other pair of samples ended in the same leaf node, our proximity matrix looks like this after running the samples down the first tree.

	1	2	3	4
1				
2				
3				1
4			1	

Filled-in Missing Values

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**167.5**	No

Now we run all of the data down the second tree…



Filled-in Missing Values

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**No**	**167.5**	No

Ultimately, we run the data down all the trees and the proximity matrix fills in.

	1	2	3	4
1		2	1	1
2	2		1	1
3	1	1		8
4	1	1	8	

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

Then we divide each proximity value by the total number of trees. In this example, assume we had 10 trees.

Filled-in Missing Values

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**NO**	???	No

The weighted frequency for **"No"** is…

Yes = 1/3

No = 2/3

$$\text{Yes} = \frac{1}{3} \times 0.1 = 0.03$$

$$\text{No} = \frac{2}{3} \times 0.9 = 0.6$$

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

Weighted average = (125 x 0.1) + (180 x 0.1) + (210 x 0.8)

= 198.5

Filled-in Missing Values

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	NO	**198.5**	No

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

The weighted average weight!

Filled-in Missing Values

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	Yes	**NO**	**198.5**	No

Now that we've revised our guesses a little bit, we do the whole thing over again…

We build a random forest, run the data through the trees, recalculate the proximities and recalculate the missing values.

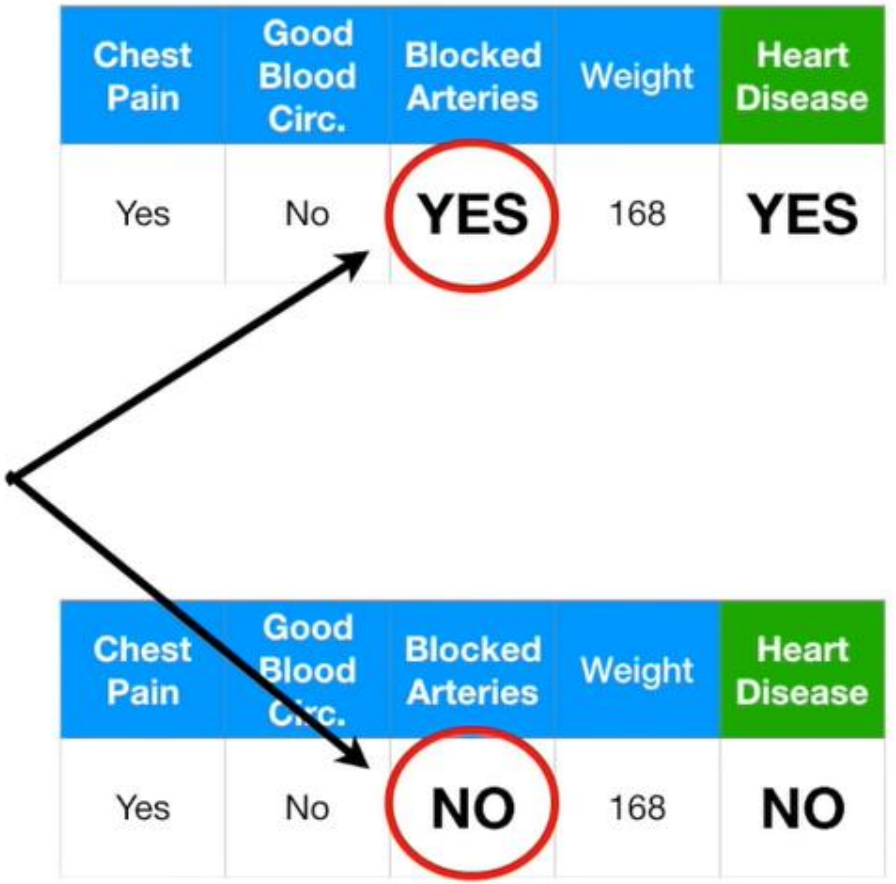
We do this 6 or 7 times until the missing values converge (i.e. no longer change each time we recalculate).

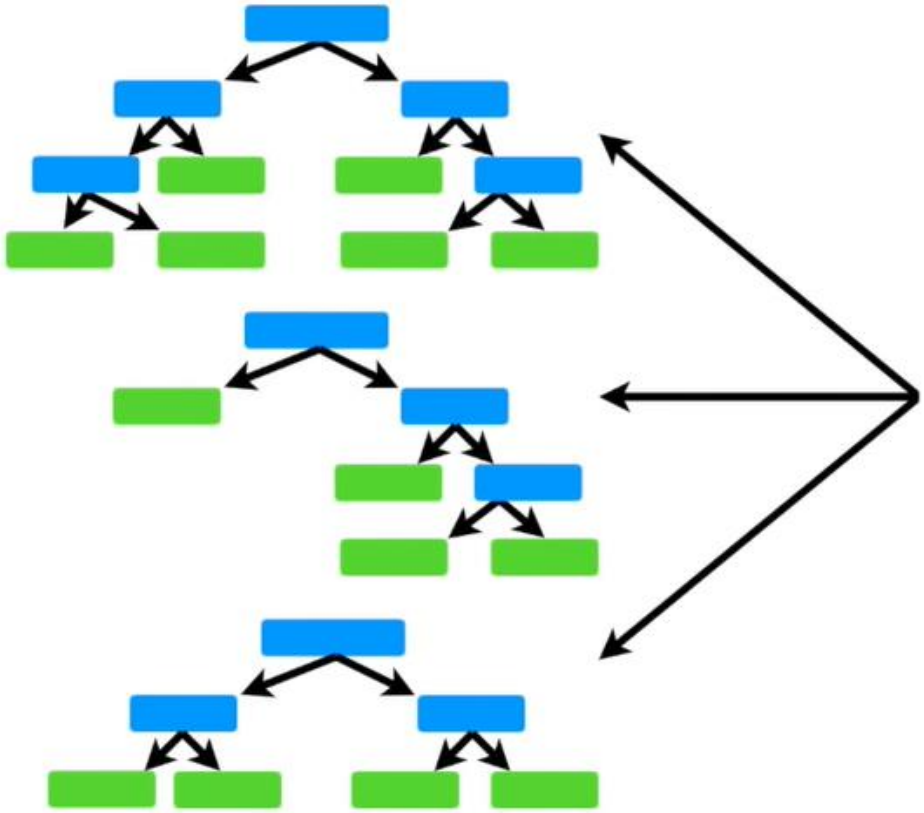
Missing data in a new testing sample?

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	???	168	

…but we don't know if they have blocked arteries…

Then we use the iterative method we just talked about to make a good guess about the missing values.

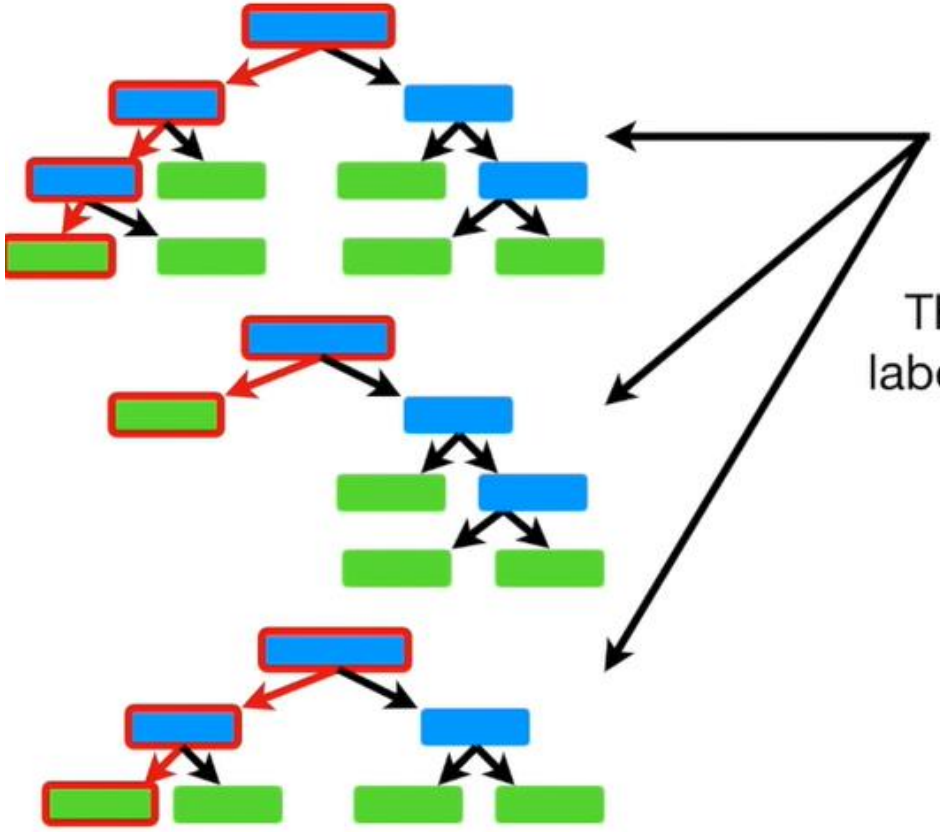




Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	YES	168	YES

…and we see which of the two is correctly labeled by the random forest the most times.

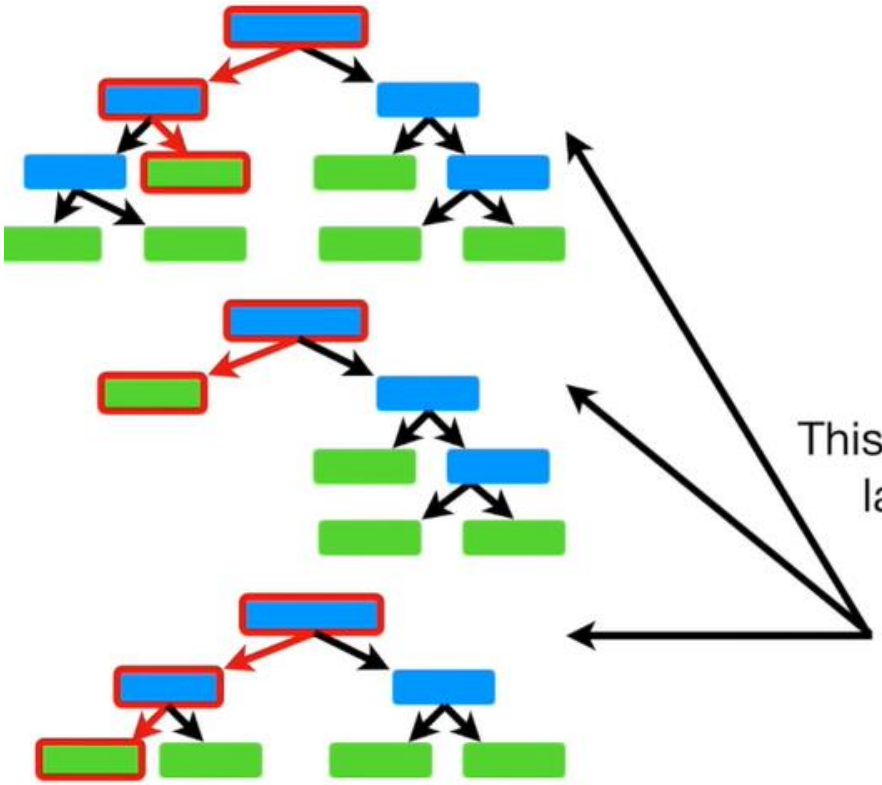
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	NO	168	NO



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	**YES**	168	**YES**

This option was correctly labeled "**Yes**" in all 3 trees…

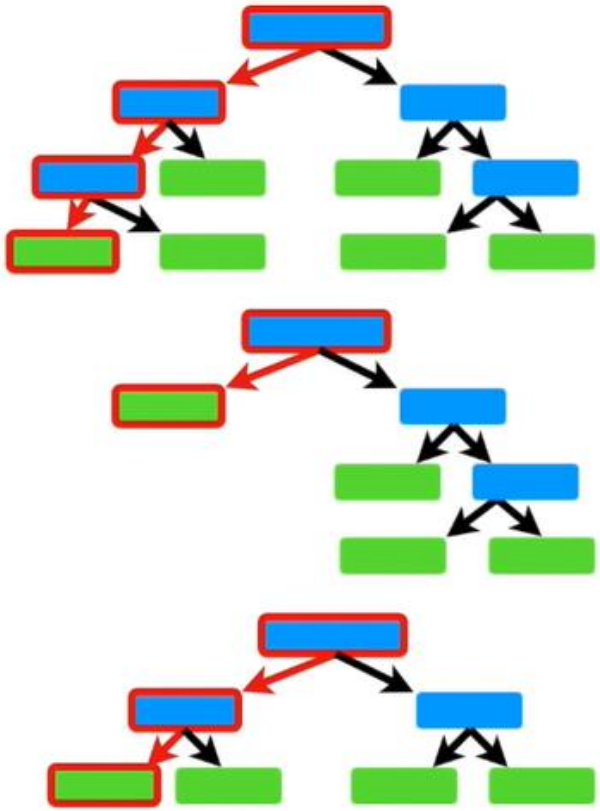
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	**NO**	168	**NO**



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	YES	168	YES

This option was only correctly labeled "**No**" in 1 tree…

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	NO	168	NO



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	**YES**	168	**YES**

This option wins because it was correctly labeled more than the other option.

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	**NO**	168	**NO**

作業三：試以年齡、收入、學生身分及信用卡紀錄情況四個特徵建構一棵預測是否會購買電腦的決策數 (Due 4/12)

HW3 Construct a decision tree using ID3 algorithm for the database given in the following table.

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Example of Random Forest

- Please link to https://www.youtube.com/watch?v=J4Wdy0Wc_xQ