UNIVERSITÀ DEGLI STUDI DI TRIESTE

Statistics

Decision Tree and Random Forest

Luca Pennella
September 19th, 2024

Decision and Classification Tree

Decision forests are most effective when you have a tabular dataset (data you might represent in a spreadsheet, csv file, or database table). Tabular data is one of the most common data formats, and decision forests should be your “go-to” solution for modeling it.

Unlike neural networks, decision forests natively consume model tabular data. When developing decision forests, you don't have to do tasks like the following:

+ Perform preprocessing like feature normalization or one-hot encoding
+ Perform imputation (for example, replacing a missing value with -1)

However, decision forests are not well suited to directly consume non-tabular data (also called unstructured data), such as images or text. Yes, workarounds for this limitation do exist, but neural networks generally handle unstructured data better.

Decision and Classification Tree

Decision forests are most effective when you have a tabular dataset (data you might represent in a spreadsheet, csv file, or database table). Tabular data is one of the most common data formats, and decision forests should be your "go-to" solution for modeling it.

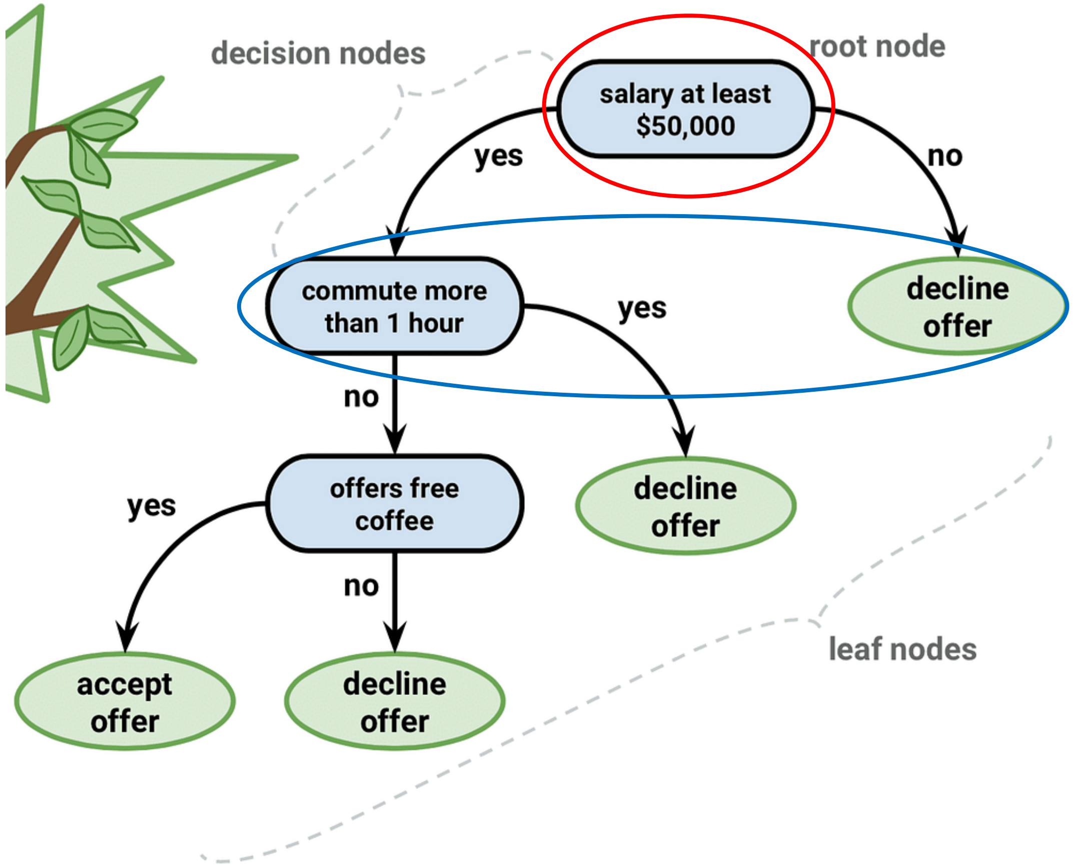
Unlike neural networks, decision forests natively consume model tabular data. When developing decision forests, you don't have to do tasks like the following:

+ Perform preprocessing like feature normalization or one-hot encoding
+ Perform imputation (for example, replacing a missing value with -1)

However, decision forests are not well suited to directly consume non-tabular data (also called unstructured data), such as images or text. Yes, workarounds for this limitation do exist, but neural networks generally handle unstructured data better.

Decision and Classification Tree

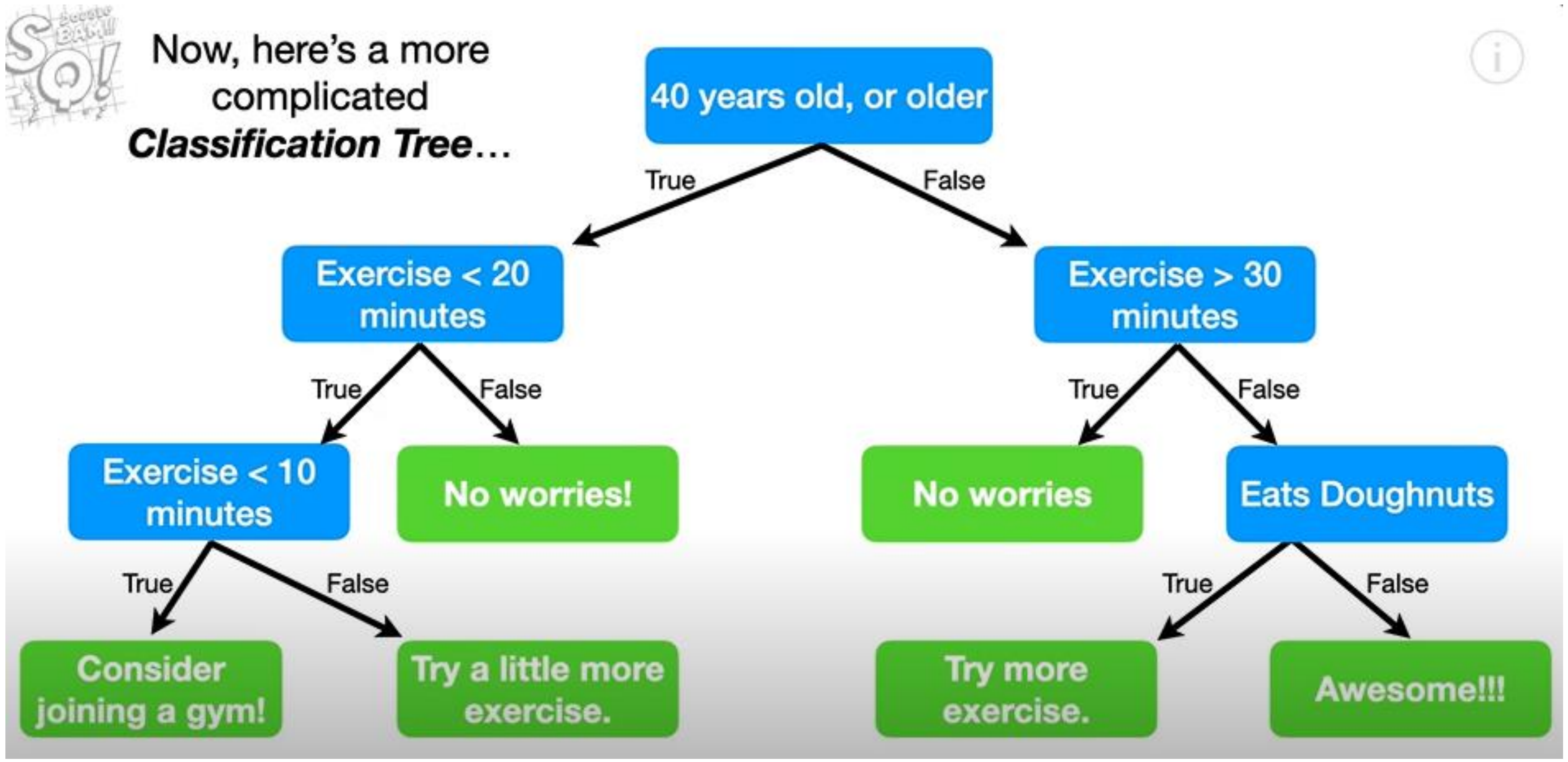
- Should I accept a new job offer?



1. In general Decision Tree makes a statement
2. Then makes a decision based on whether or not the statement is **True** or **False**

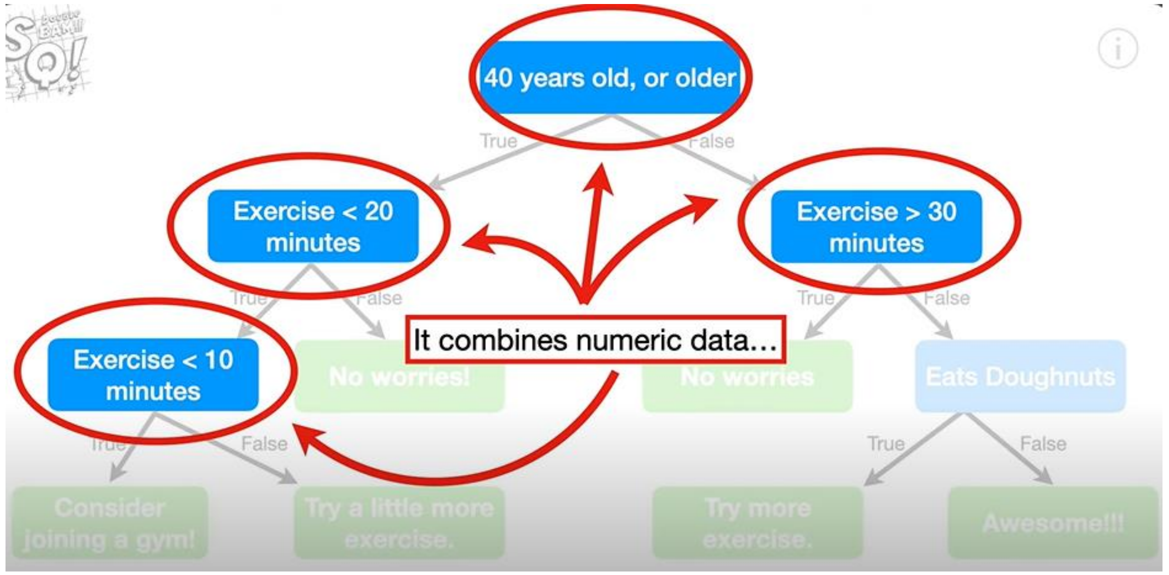
+ When a Decision Tree **classifies things into categories** it's called a **Classification Tree**
+ When a Decision Tree **predicts numeric values**, it's called **Regression Tree**

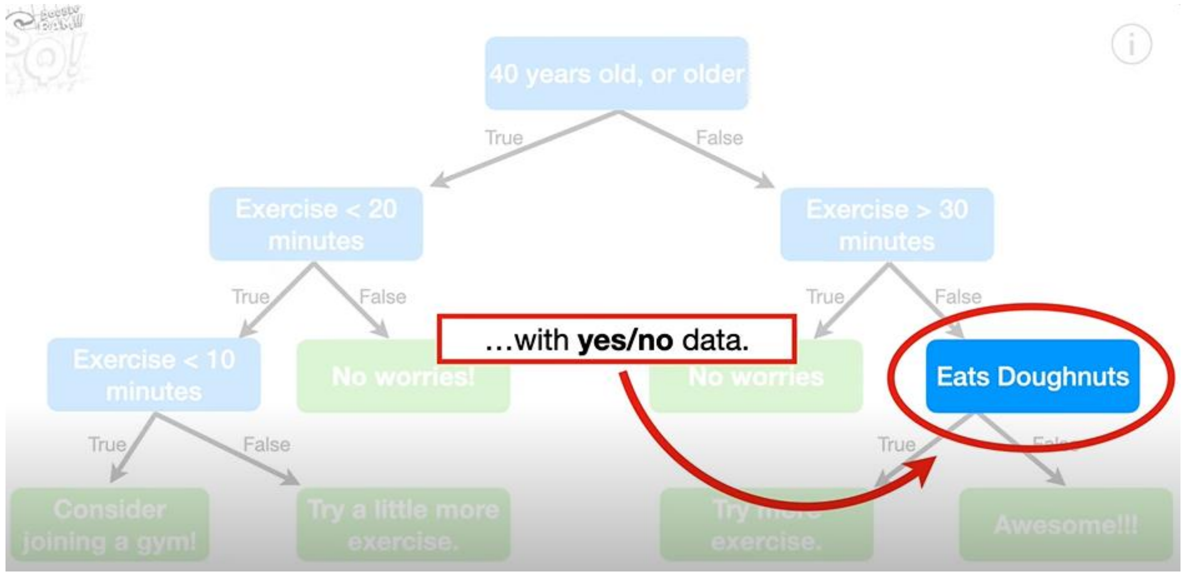
A first Decision Tree



*from StatQuest youtube channel

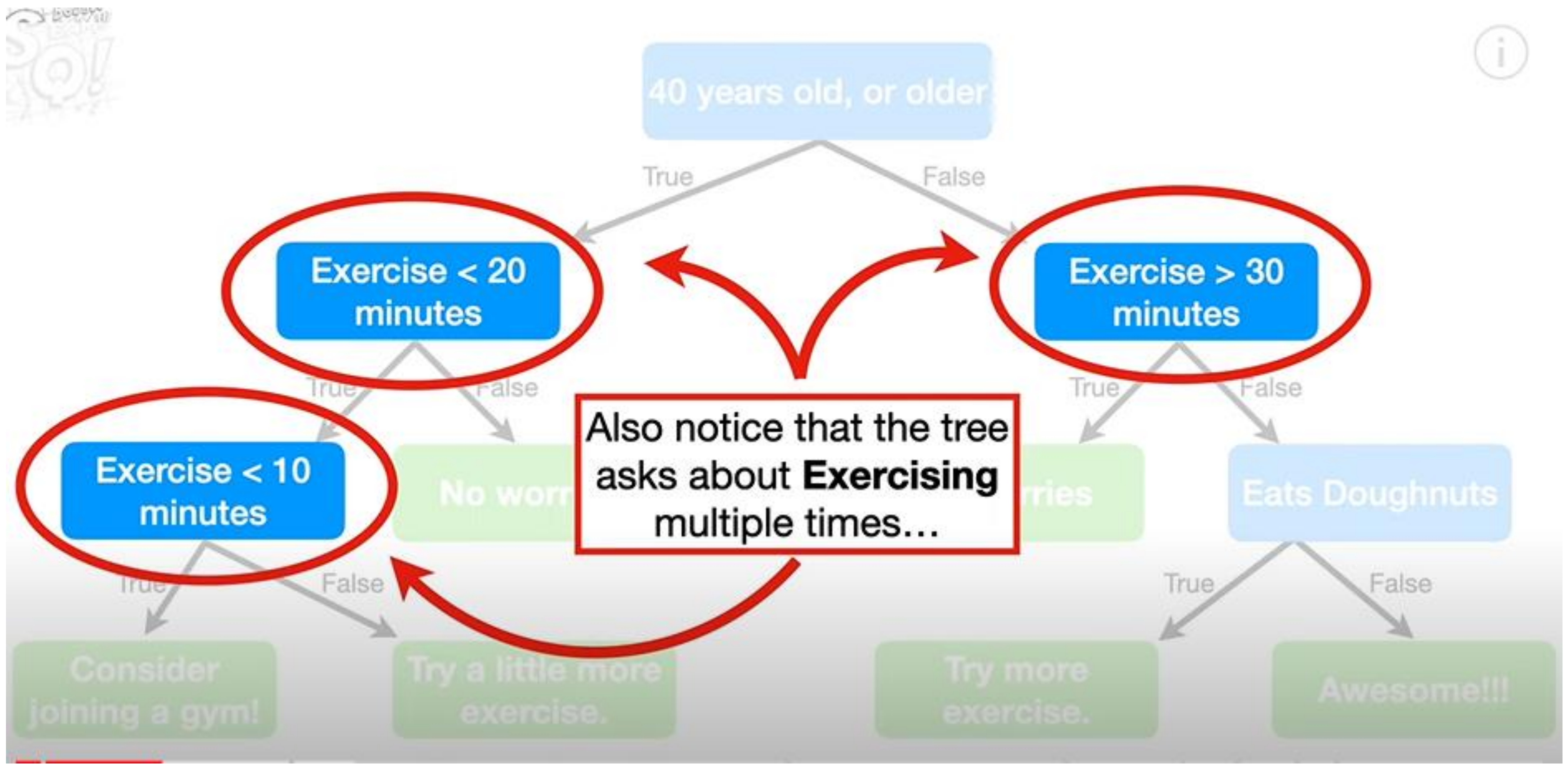
What types of data?





*from StatQuest youtube channel

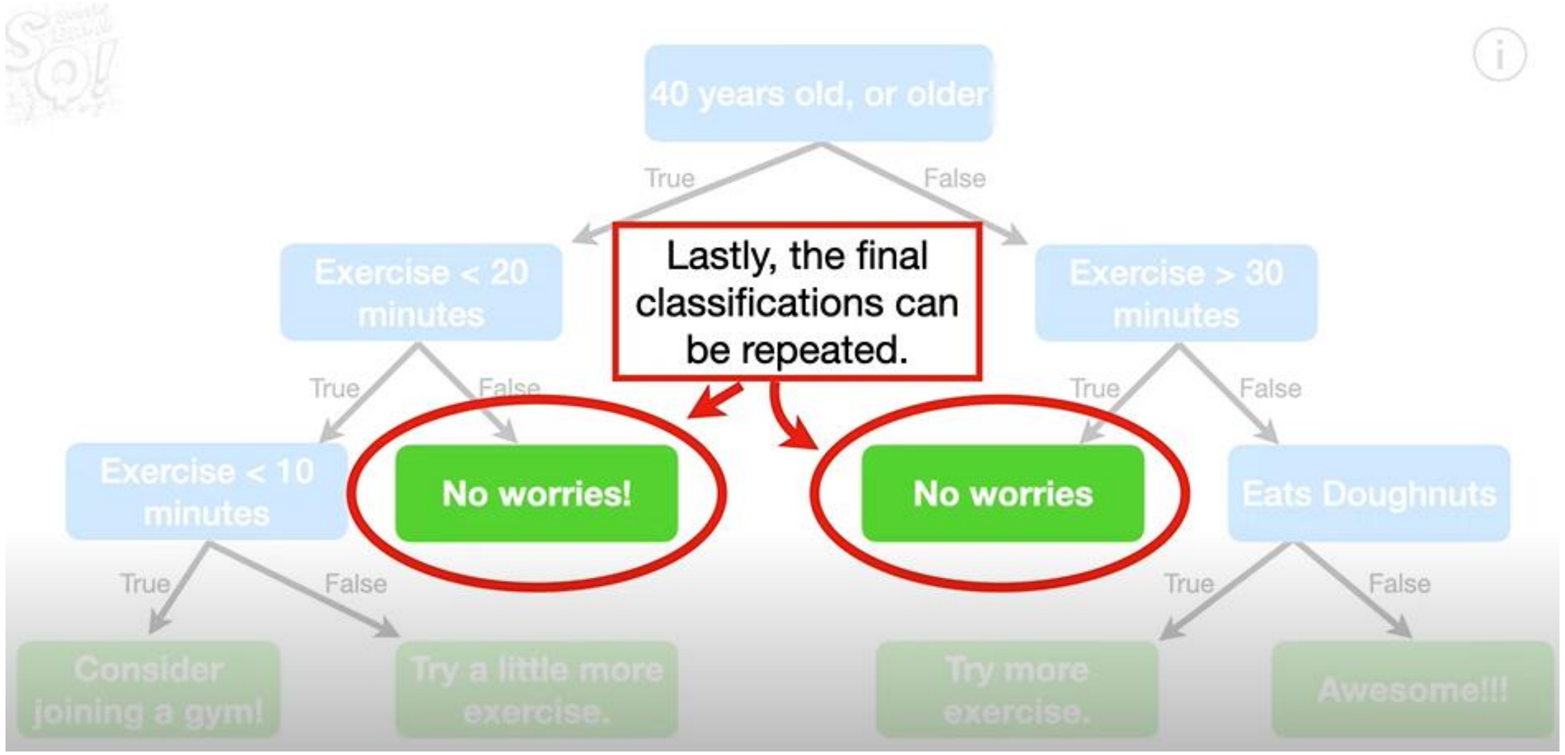
What types of data?



+ But the amount of time **Exercising** isn't always the same.
+ The numeric thresholds can be different for the same data.

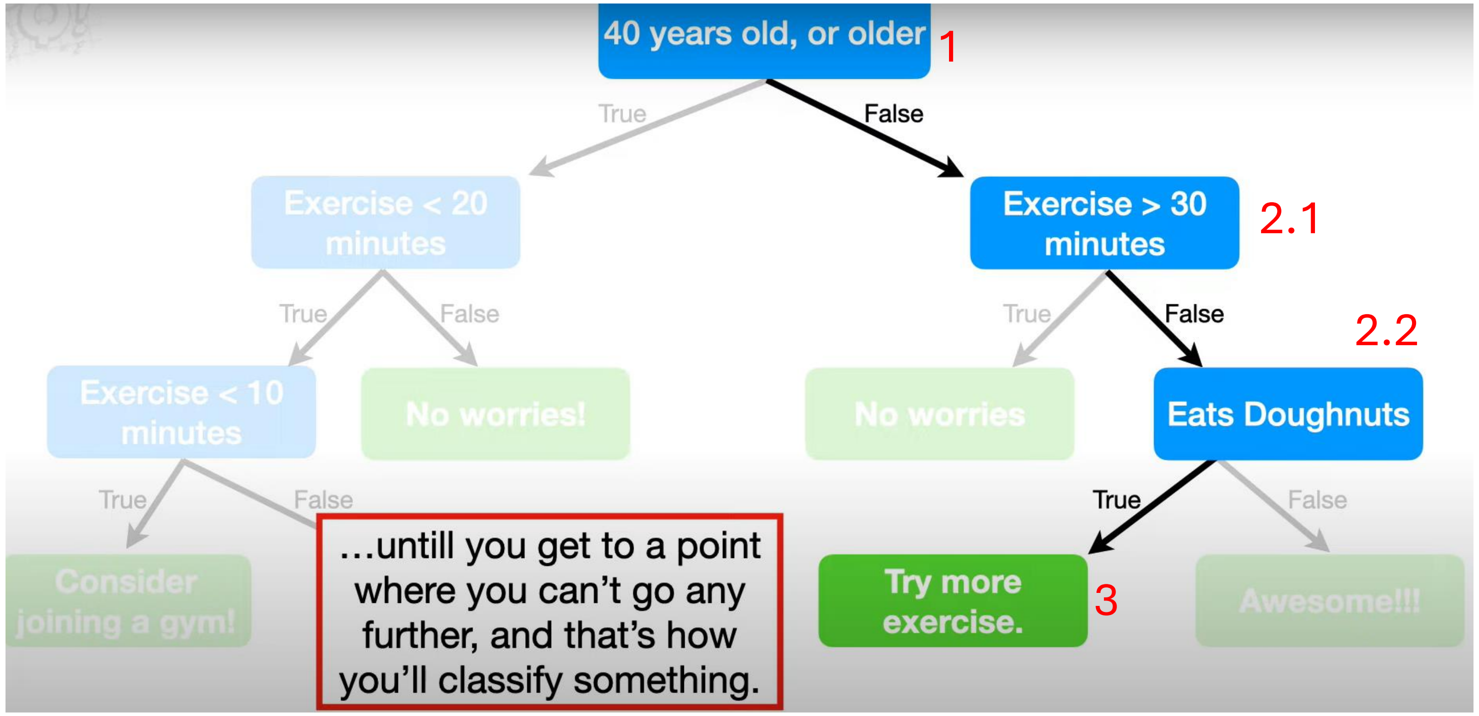
*from StatQuest youtube channel

A first Decision Tree



*from StatQuest youtube channel

A first Decision Tree

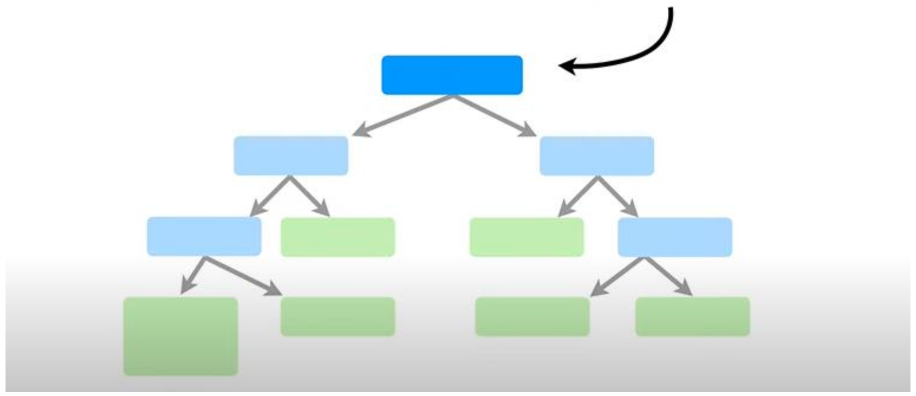


1. We start from the root and make the initial split
2. Then work our way down
3. until we achieve a classification.

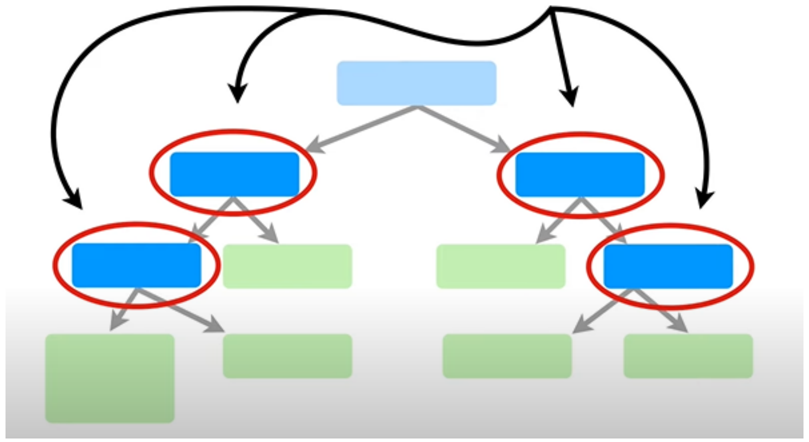
*from StatQuest youtube channel

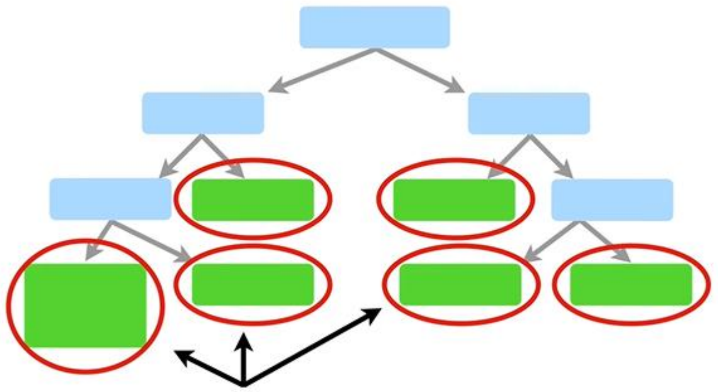
A bit of terminology

The very top of the tree is called the **Root Node** or just **The Root**.



These are called **Internal Nodes**, or **Branches**.

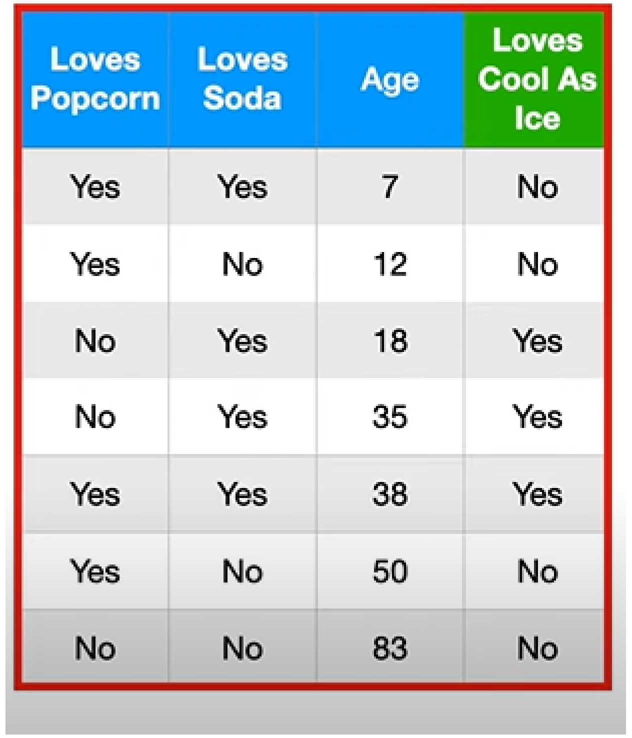




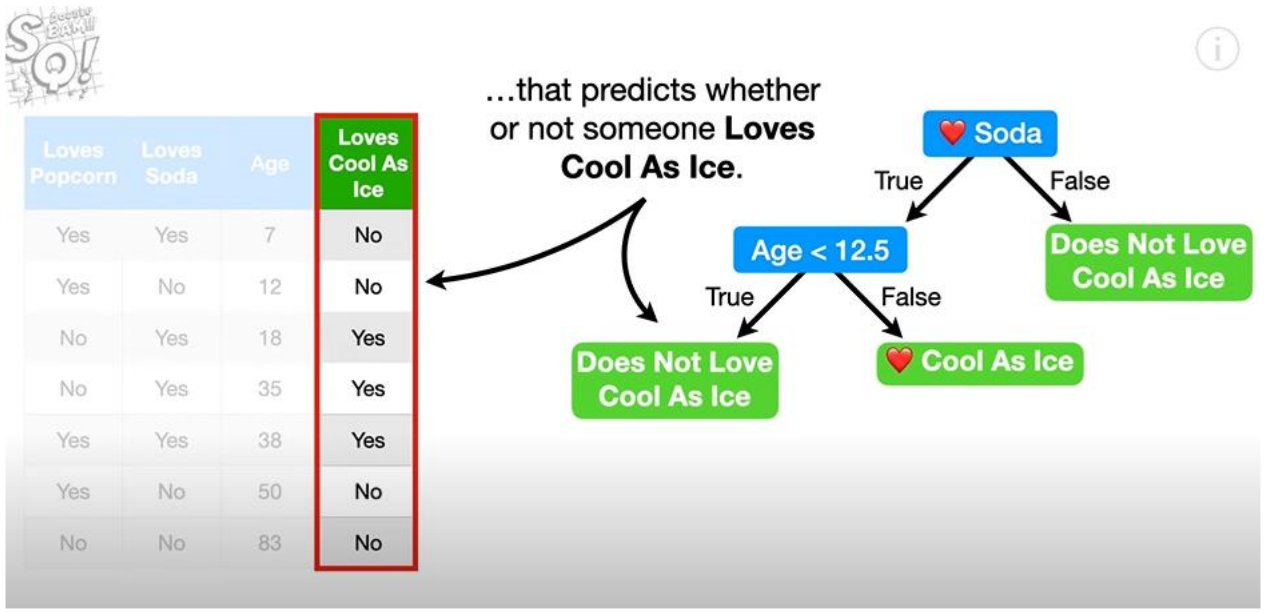
Lastly, these are called **Leaf Nodes**, or just **Leaves**.

*from StatQuest youtube channel

Building a Tree – From table to Tree



Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

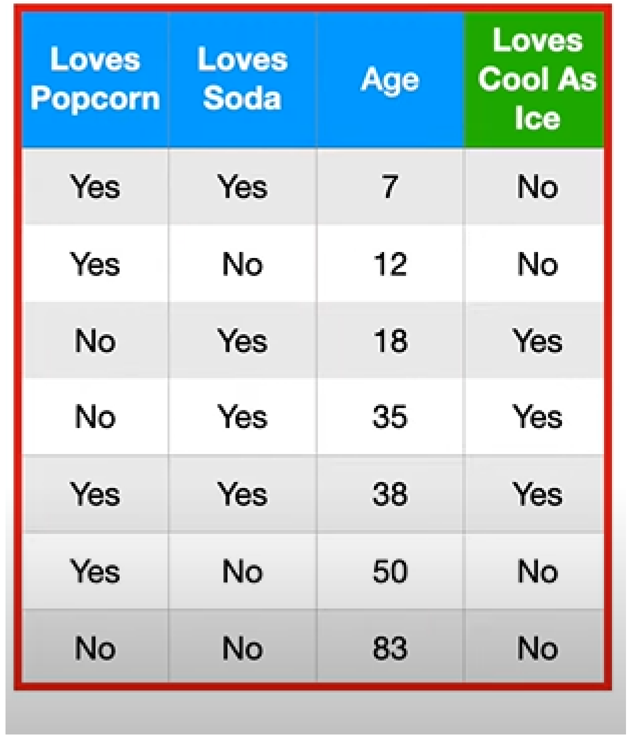


Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

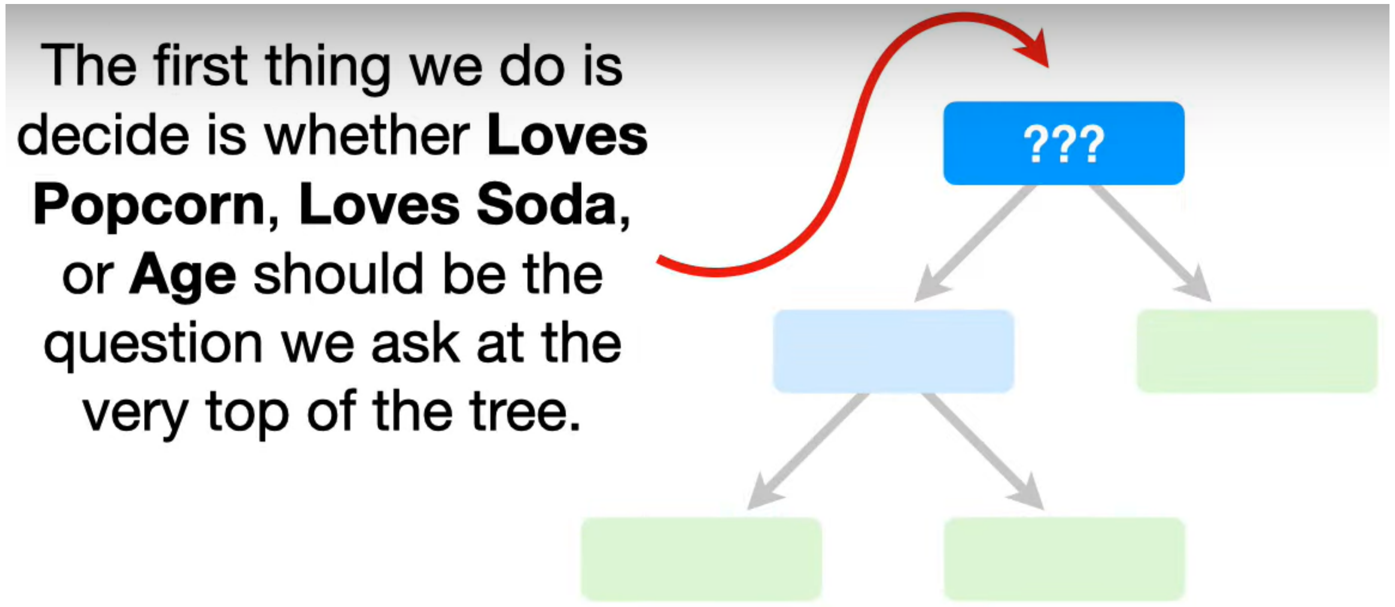
...that predicts whether or not someone **Loves Cool As Ice.**

*from StatQuest youtube channel

Building a Tree – First split



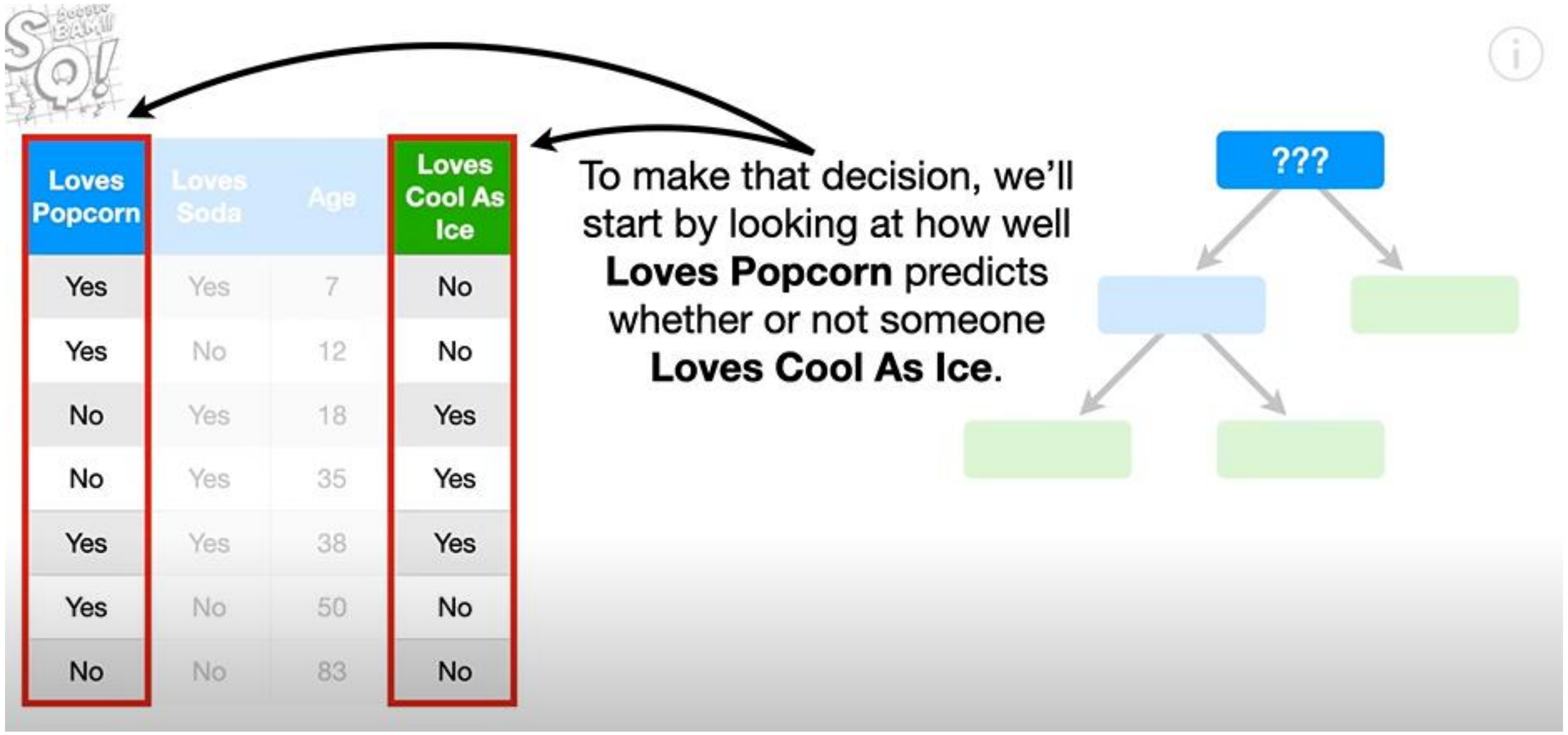
Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No



The first thing we do is decide is whether **Loves Popcorn**, **Loves Soda**, or **Age** should be the question we ask at the very top of the tree.

*from StatQuest youtube channel

Building a Tree – First split

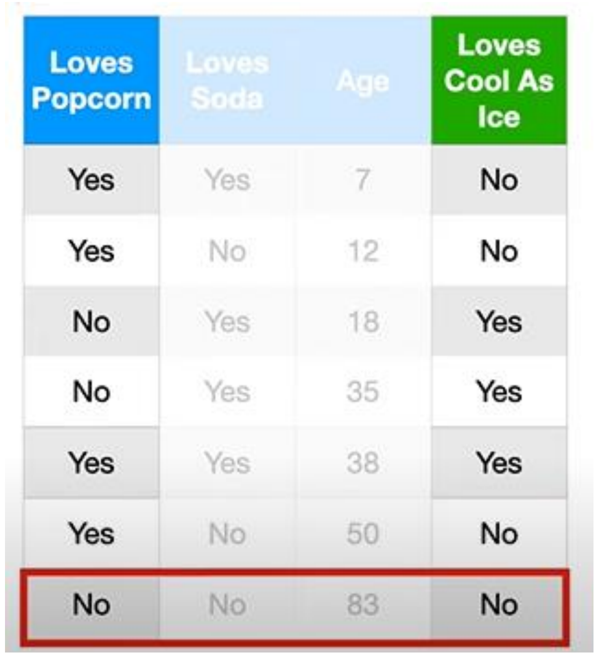


Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

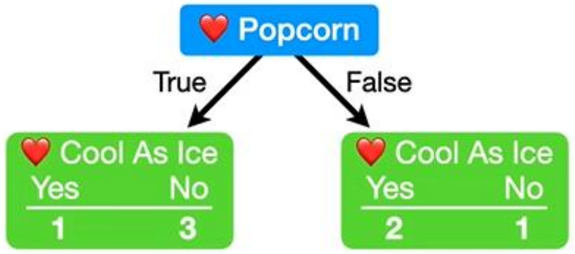
To make that decision, we'll start by looking at how well **Loves Popcorn** predicts whether or not someone **Loves Cool As Ice.**

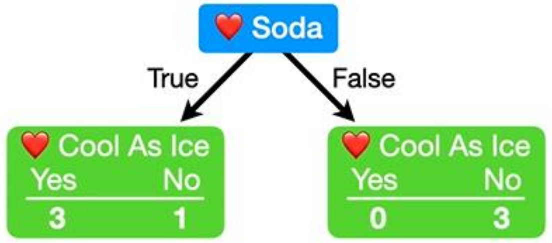
*from StatQuest youtube channel

Building a Tree – First split



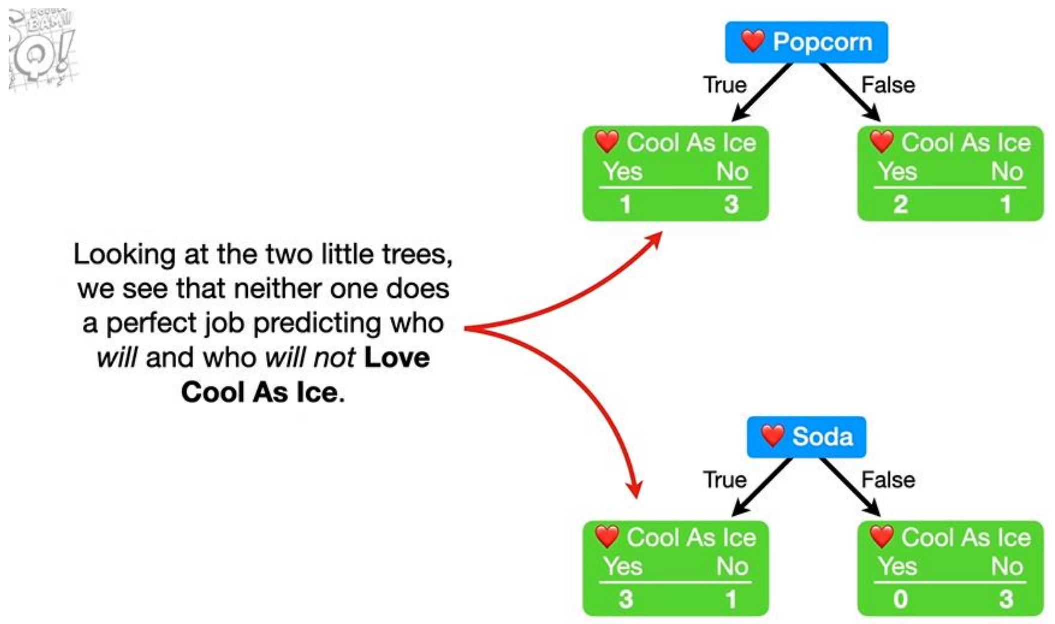
Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No



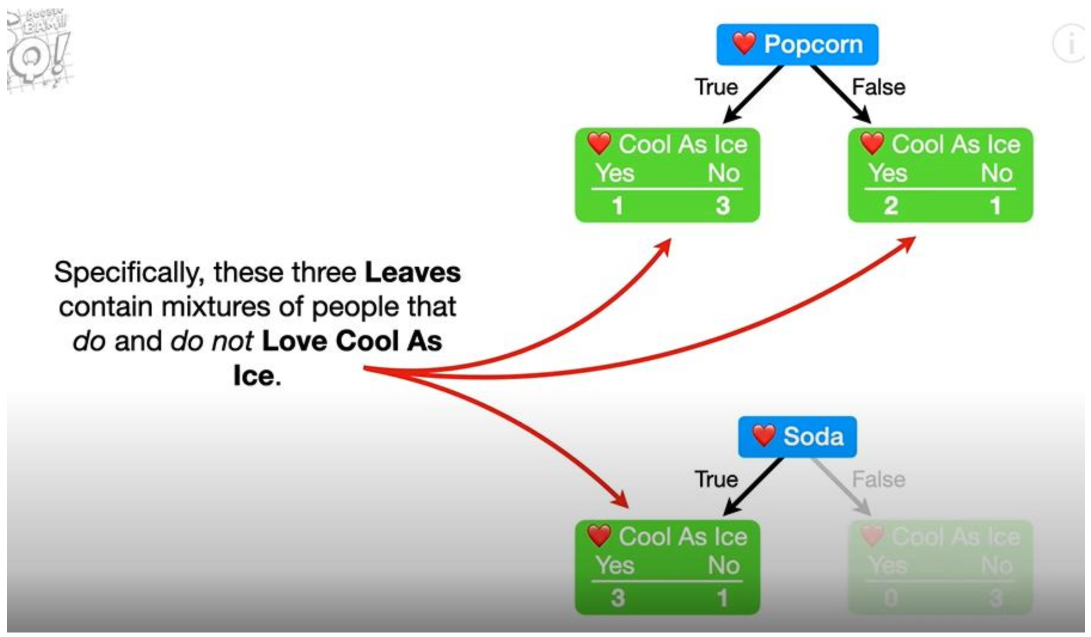


*from StatQuest youtube channel

Building a Tree – First split



Looking at the two little trees, we see that neither one does a perfect job predicting who *will* and who *will not* **Love Cool As Ice.**



Specifically, these three **Leaves** contain mixtures of people that *do* and *do not* **Love Cool As Ice.**

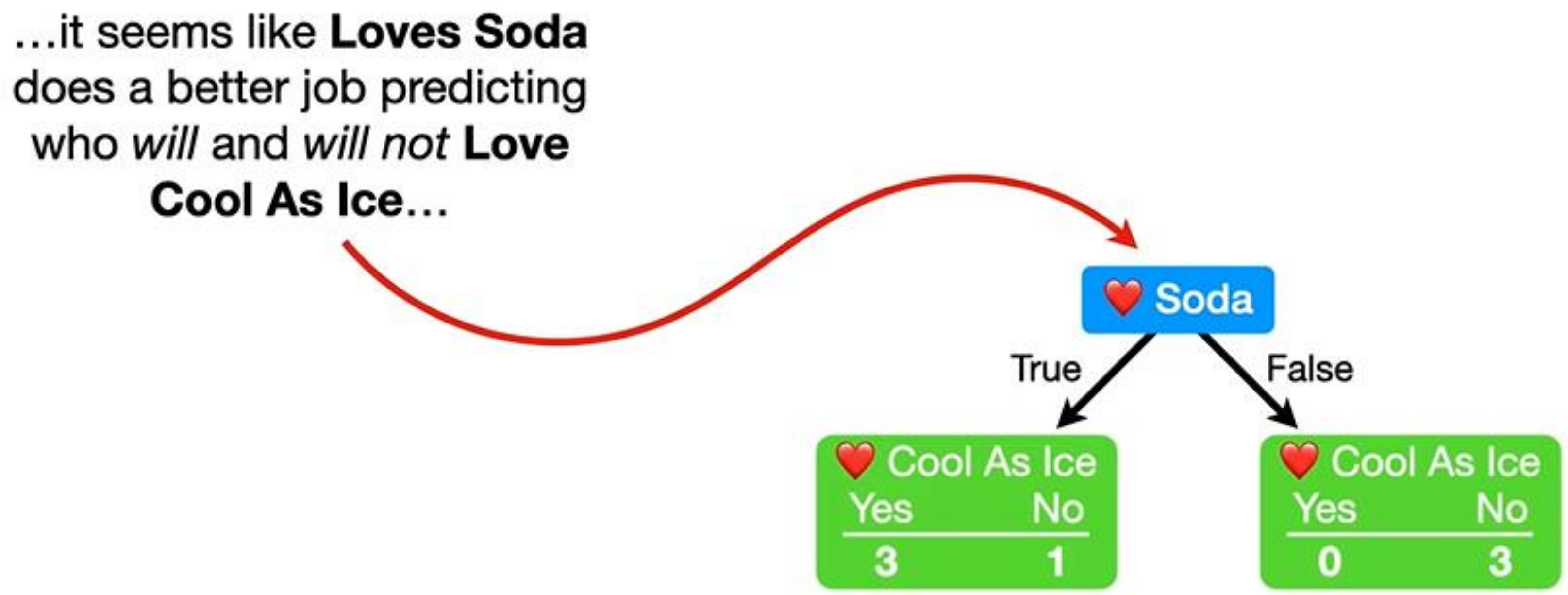
If the leaves contain a mixture, they are called **impure.**

Who do you think does the best job?

*from StatQuest youtube channel

Building a Tree

Who do you think does the best job?


...it seems like **Loves Soda** does a better job predicting who *will* and *will not* **Love Cool As Ice**...
Soda
True
False
Cool As Ice
Yes
No
3
1
Cool As Ice
Yes
No
0
3

*from StatQuest youtube channel

Building a Tree – Impurity

Ok, but we need to find a way to quantify this purity or impurity

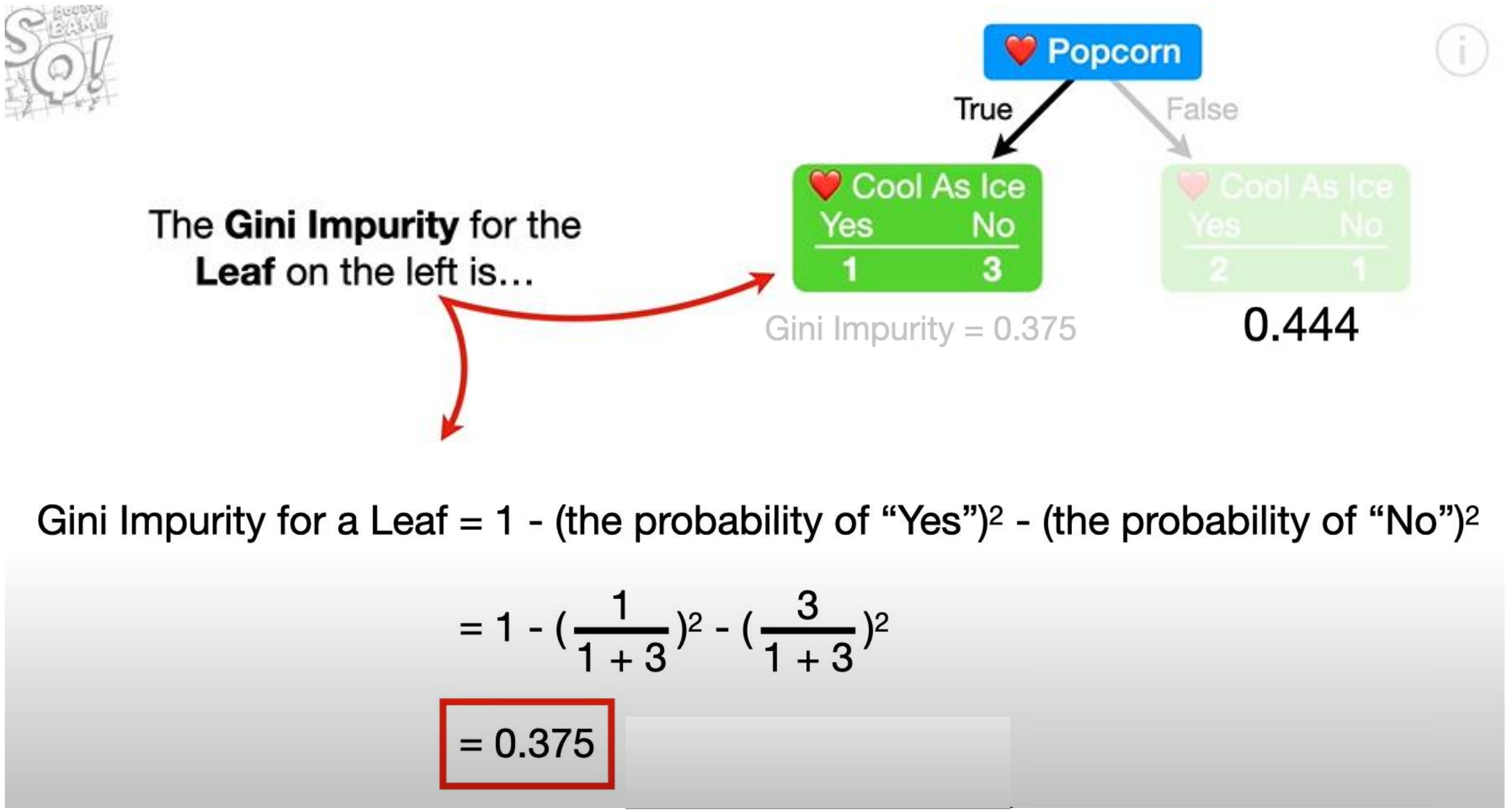
One of the most popular methods is called **Gini Impurity**.
There also other methods like **Entropy** and **Information Gain**.
However, numerically, the methods are all quite similar.
We focus on **Gini Impurity.**

Building a Tree - Gini Impurity

$$\mathrm{I}_G(p) = \sum_{i=1}^{J} \left(p_i \sum_{k \neq i} p_k \right) = \sum_{i=1}^{J} p_i (1 - p_i) = \sum_{i=1}^{J} (p_i - p_i^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} p_i^2 = 1 - \sum_{i=1}^{J} p_i^2.$$

Impurity of a classification tree node is computed using the count of each target category across all records corresponding to the given node. Gini impurity total is computed as a sum of squares of count proportions across all target categories per node subtracted from one, and the result is multiplied by the number of records.

Building a Tree - Gini Impurity



The **Gini Impurity** for the **Leaf** on the left is…

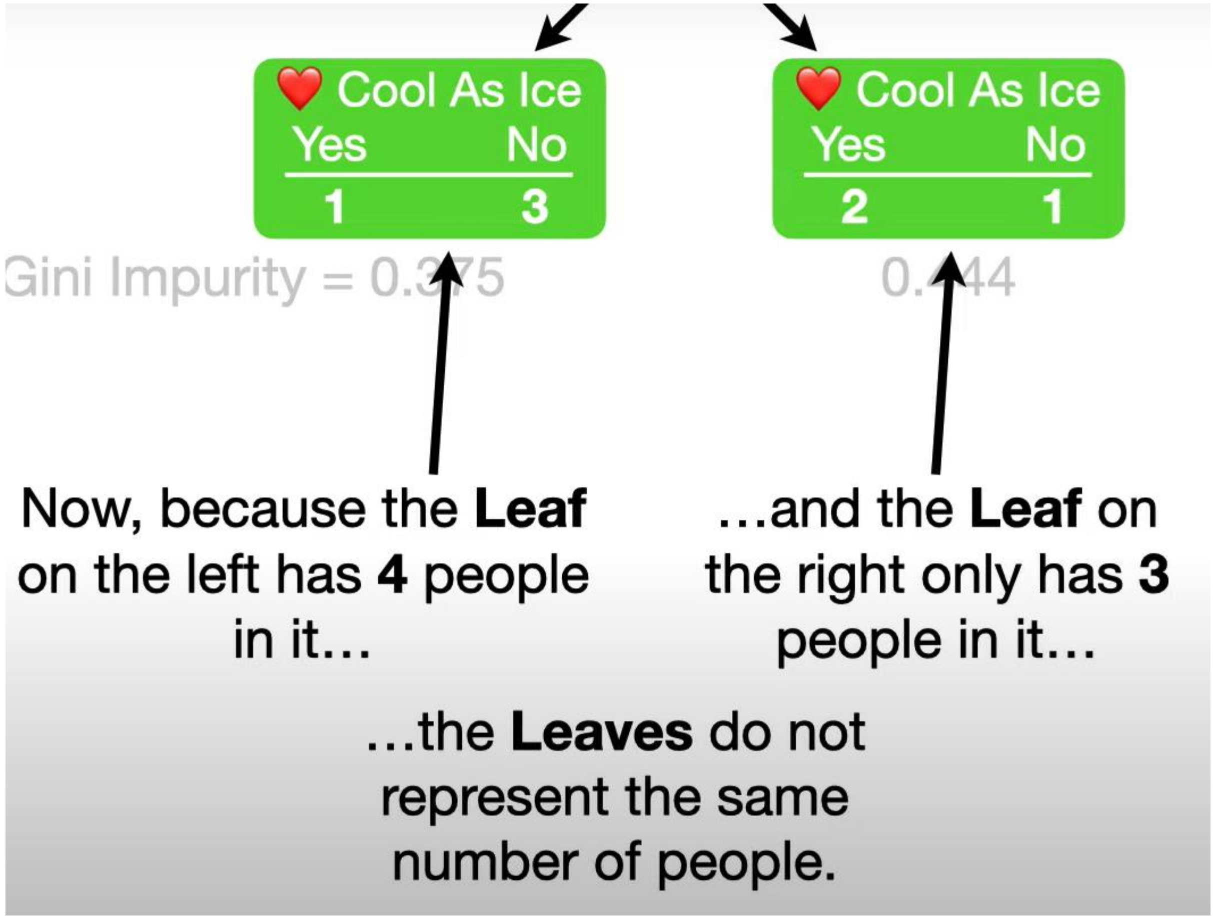
Gini Impurity for a Leaf = 1 - (the probability of “Yes”)^2 - (the probability of “No”)^2

$$= 1 - \left(\frac{1}{1+3}\right)^2 - \left(\frac{3}{1+3}\right)^2$$

$$\boxed{= 0.375}$$

*from StatQuest youtube channel

Building a Tree - Gini Impurity



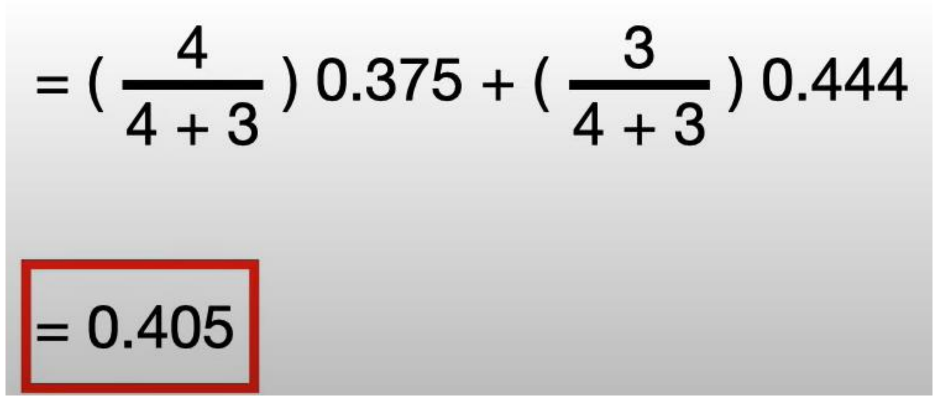
Now, because the **Leaf** on the left has **4** people in it…

…and the **Leaf** on the right only has **3** people in it…

…the **Leaves** do not represent the same number of people.

For this reason, the Total Gini Impurity is equal to:

Total **Gini Impurity** = weighted average of **Gini Impurities** for the **Leaves**



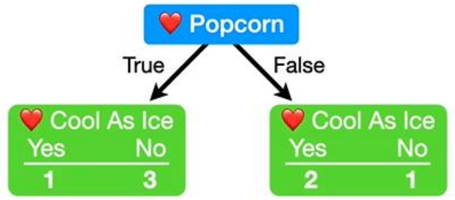
$$= \left(\frac{4}{4+3}\right) 0.375 + \left(\frac{3}{4+3}\right) 0.444$$

$$\boxed{= 0.405}$$

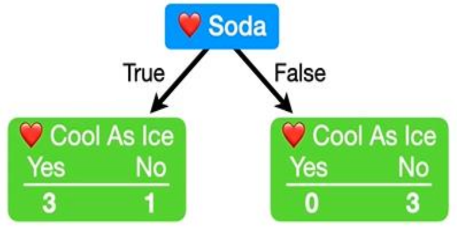
*from StatQuest youtube channel

Building a Tree – First split

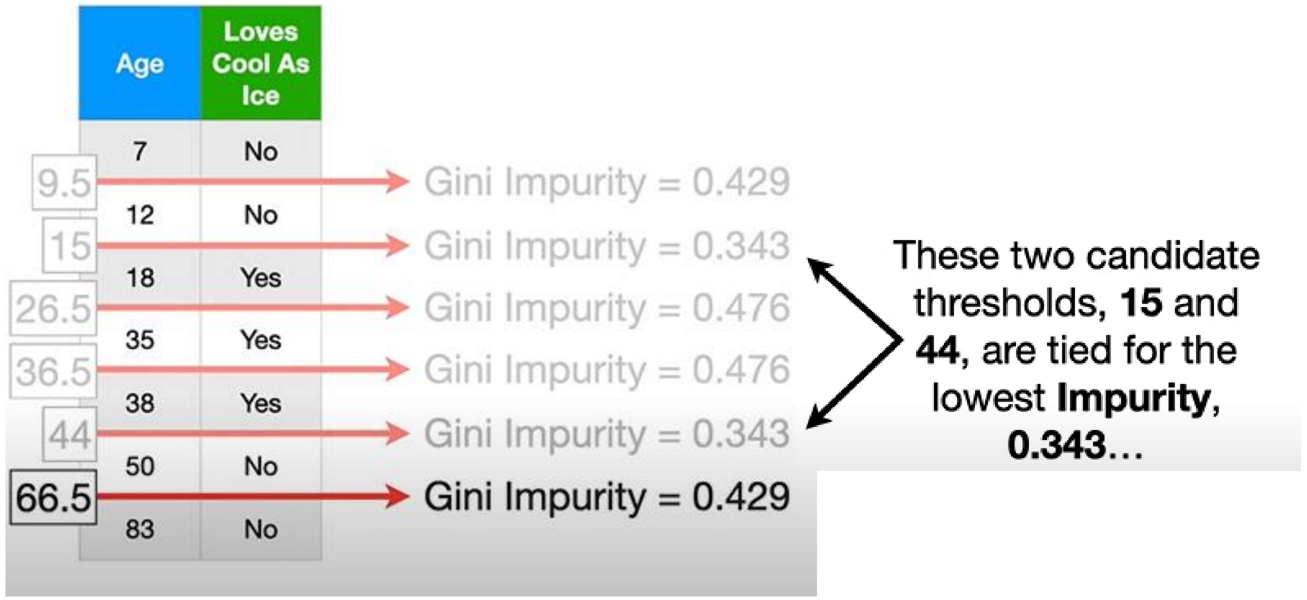
Gini Impurity for Loves Popcorn = 0.405 ar



Gini Impurity for Loves Soda = 0.214



Calculating the Gini Impurity for variables containing numeric data, such as **Age**, can be a bit more intricate



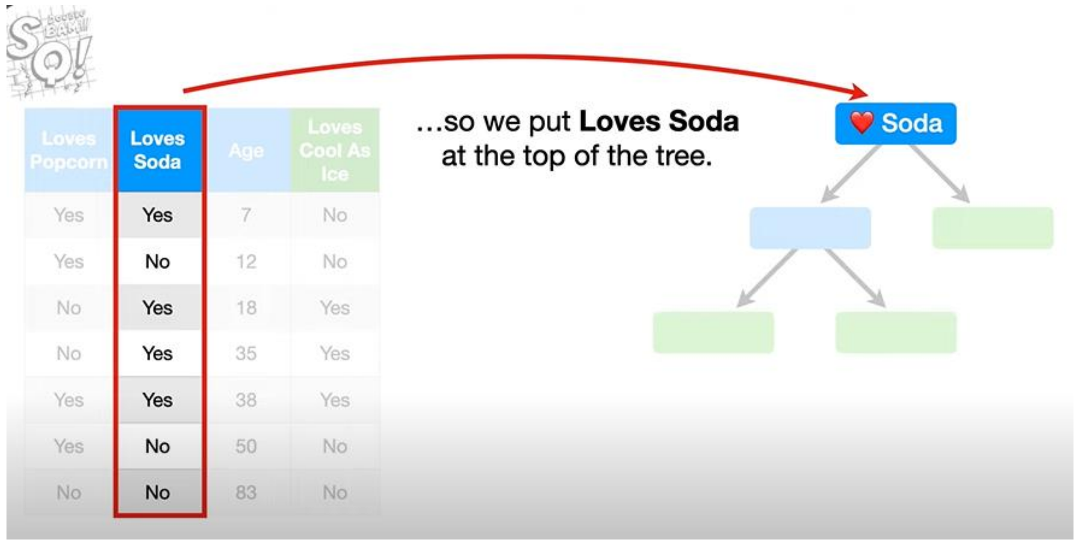
Age	Loves Cool As Ice
7	No
12	No
18	Yes
35	Yes
38	Yes
50	No
83	No

9.5 → Gini Impurity = 0.429
15 → Gini Impurity = 0.343
26.5 → Gini Impurity = 0.476
36.5 → Gini Impurity = 0.476
44 → Gini Impurity = 0.343
66.5 → Gini Impurity = 0.429

These two candidate thresholds, **15** and **44**, are tied for the lowest **Impurity**, **0.343**...

Building a Tree – First split

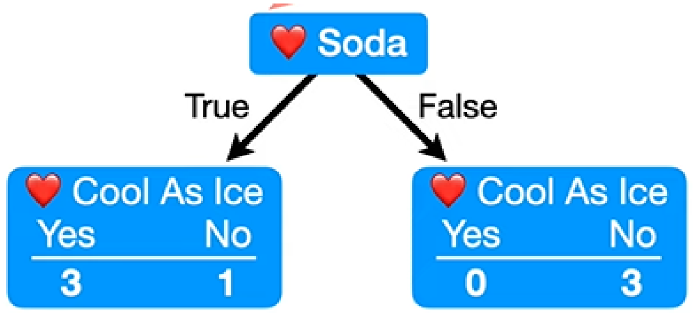
Because **Loves Soda** has the lowest Giny Impurity overall:



Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

...so we put **Loves Soda** at the top of the tree.

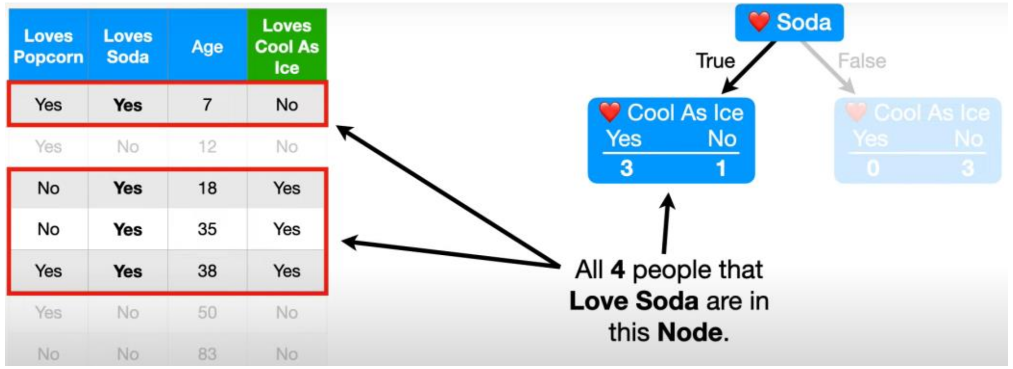
Now the 4 people tha **Loves Soda** go to a **Node** on the left and others on the right:



*from StatQuest youtube channel

Building a Tree – Other splits

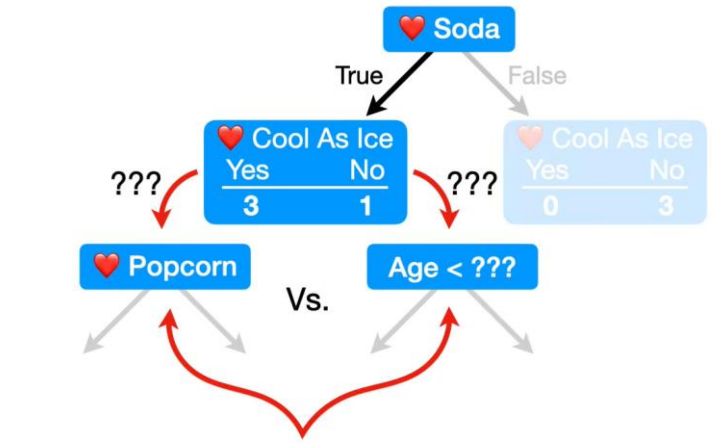
Now, we focus on the left Node:



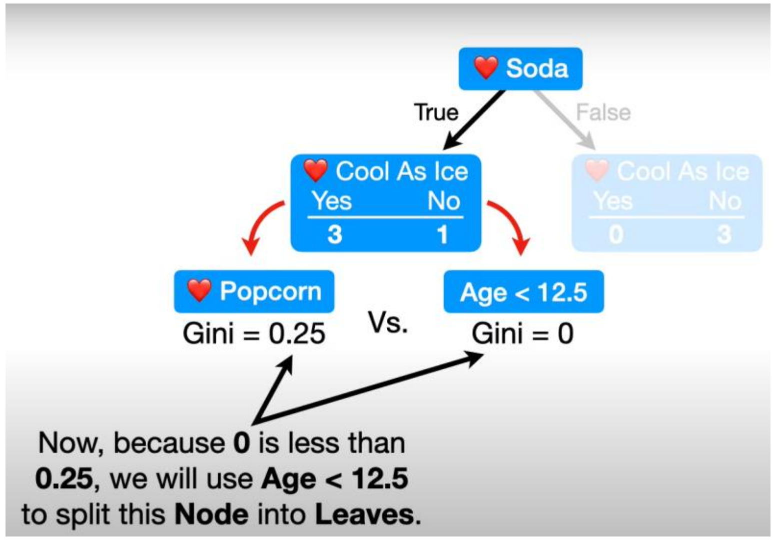
Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

All **4** people that **Love Soda** are in this **Node**.

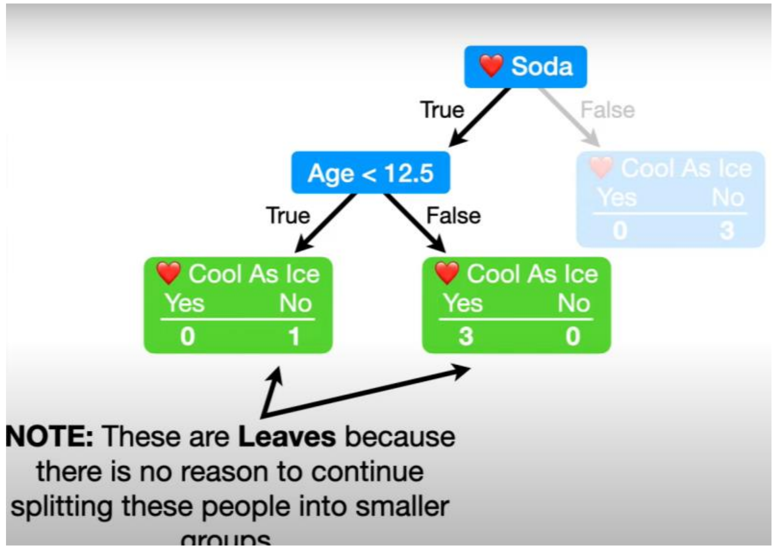
The node is Impure



So let's see if we can reduce the **Impurity** by splitting the people that **Love Soda** based on **Loves Popcorn** *or* **Age.**



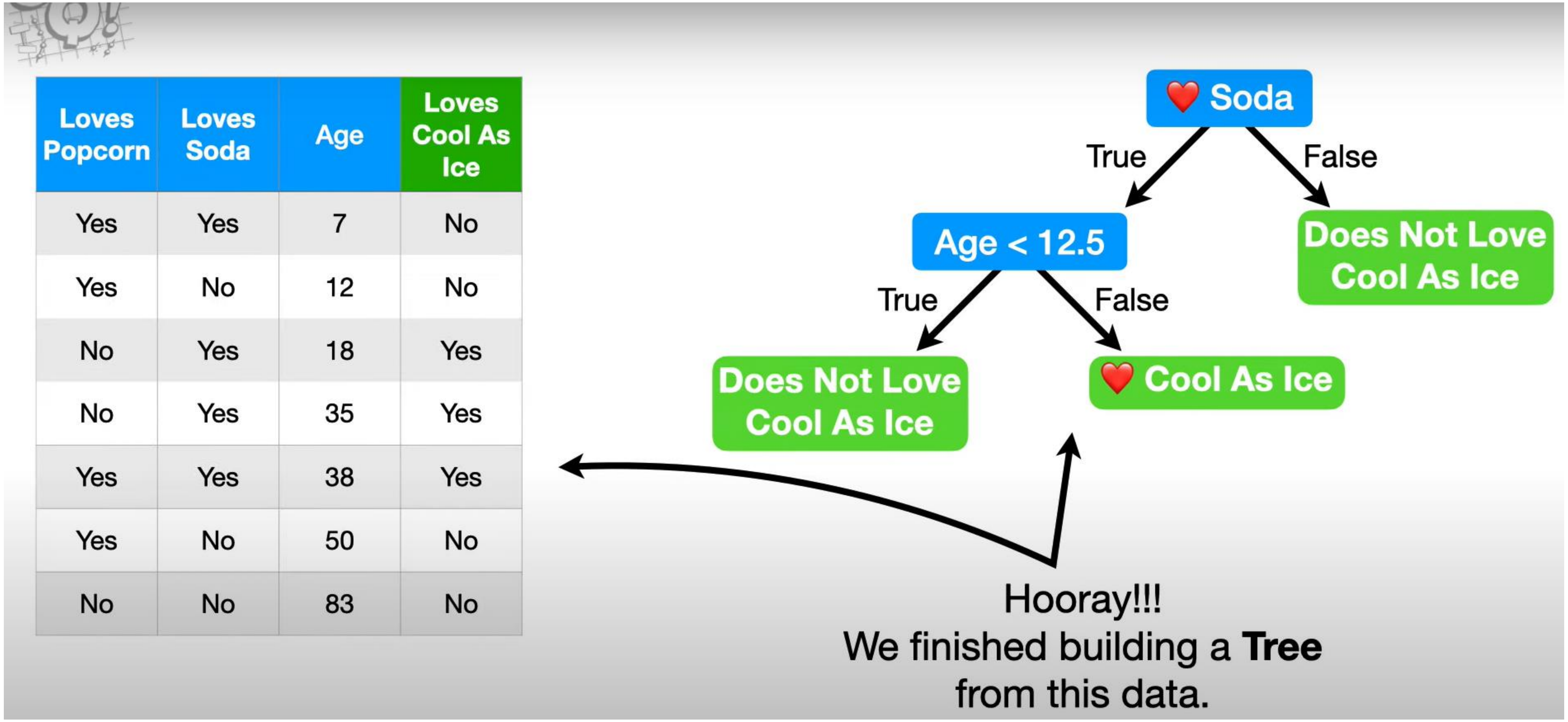
Now, because **0** is less than **0.25**, we will use **Age < 12.5** to split this **Node** into **Leaves**.



NOTE: These are **Leaves** because there is no reason to continue splitting these people into smaller groups

*from StatQuest youtube channel

Building a Tree – End!

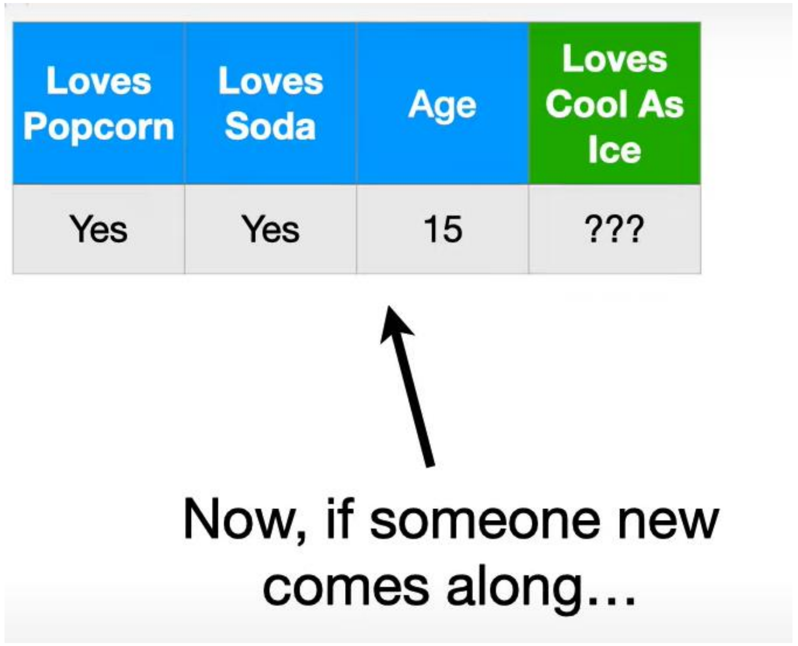


Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Hooray!!!
We finished building a **Tree** from this data.

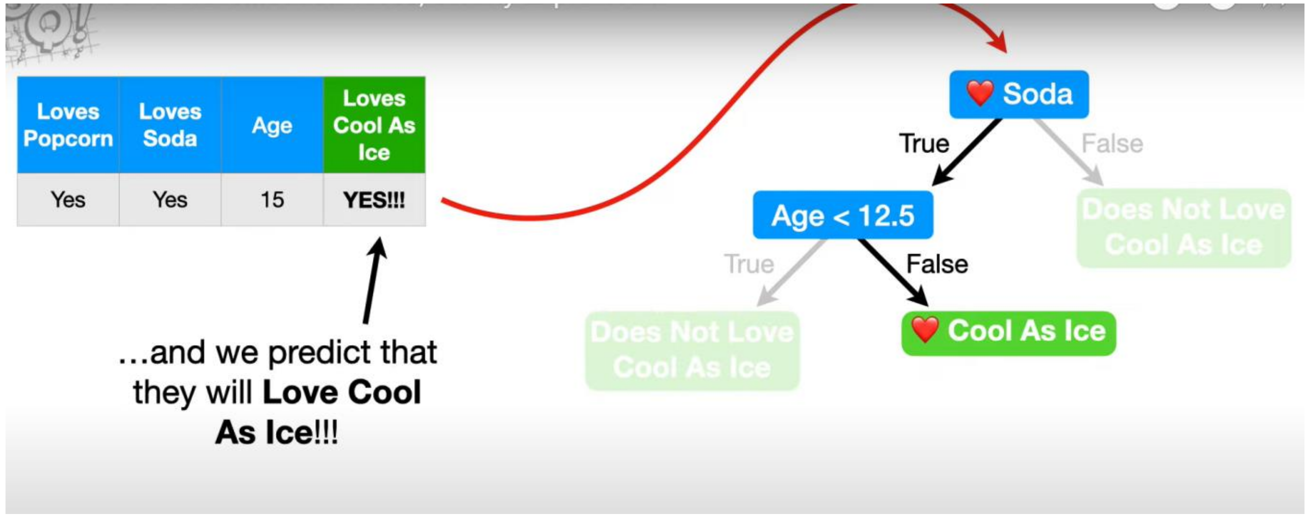
*from StatQuest youtube channel

Building a Tree – Using the Tree



Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	15	???

Now, if someone new comes along…

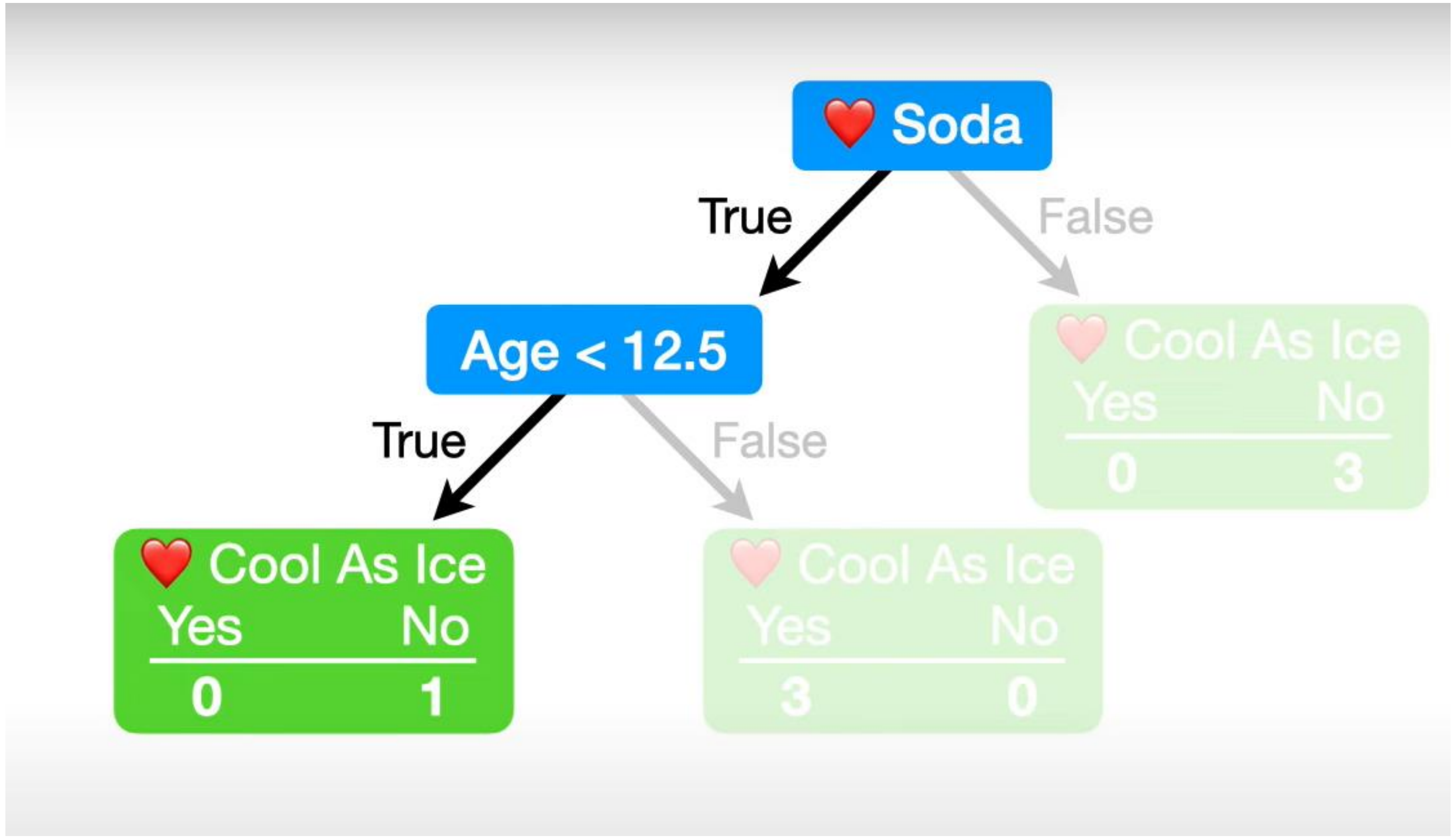


Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	15	YES!!!

…and we predict that they will **Love Cool As Ice**!!!

*from StatQuest youtube channel

Building a Tree – Overfitting?

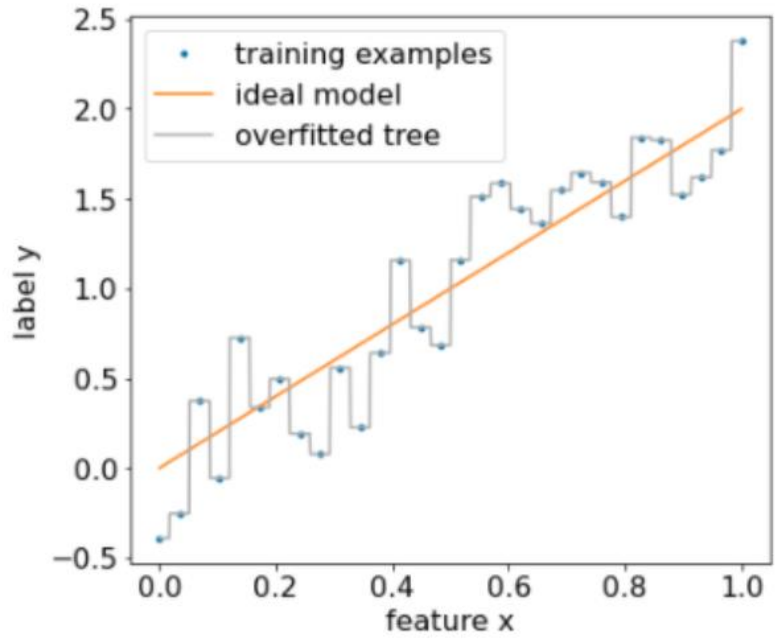


*from StatQuest youtube channel

Overfitting and pruning

Using the algorithm described above, we can train a decision tree that will perfectly classify training examples, assuming the examples are separable. However, if the dataset contains noise, this tree willo verfit to the data and show poor test accuracy.

The following figure shows a noisy dataset with a linear relation between a feature x and the label y. The figure also shows a decision tree trained on this dataset without any type of regularization. This model correctly predicts all the training examples (the model's prediction match the training examples). However, on a new dataset containing the same linear pattern and a different noise instance, the model would perform poorly.



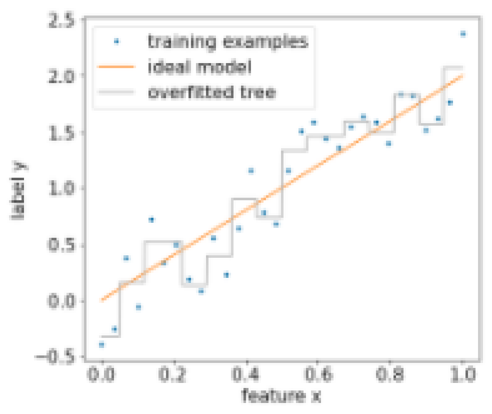
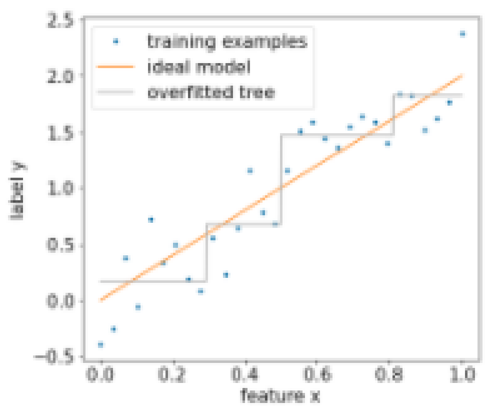
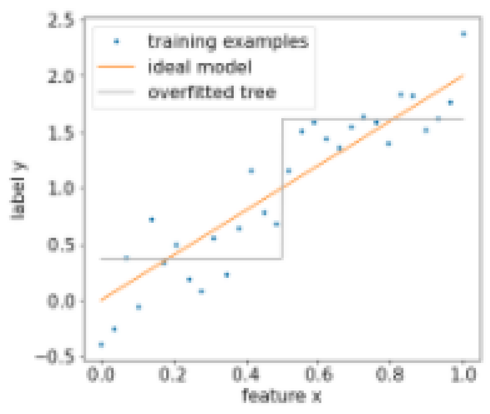
Overfitting and pruning

+ To limit overfitting a decision tree, apply one or both of the following regularization criteria while training the decision tree:

  + **Set a maximum depth:** Prevent decision trees from growing past a maximum depth, such as 10.
  + **Set a minimum number of examples in leaf:** A leaf with less than a certain number of examples will not be considered for splitting.

Overfitting and pruning

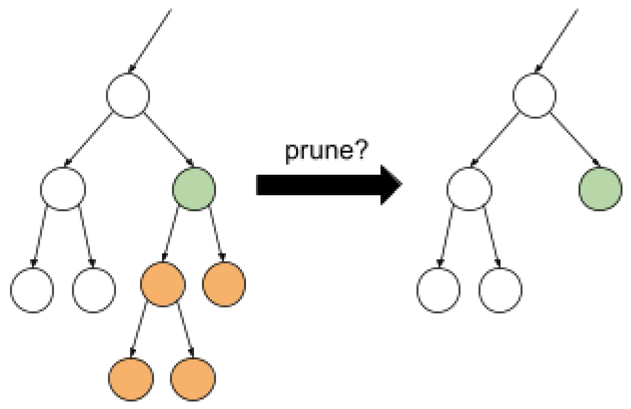
The following figure illustrates the effect of differing minimum number of examples per leaf. The model captures less of the noise.

minimum number of examples in leaf: 2	minimum number of examples in leaf: 5	minimum number of examples in leaf: 10
		

Overfitting and pruning

You can also regularize after training by selectively **removing (pruning) certain branches**, that is, by converting certain non-leaf nodes to leaves. A common solution to select the branches to remove is to use a validation dataset. That is, if removing a branch improves the quality of the model on the validation dataset, then the branch is removed.

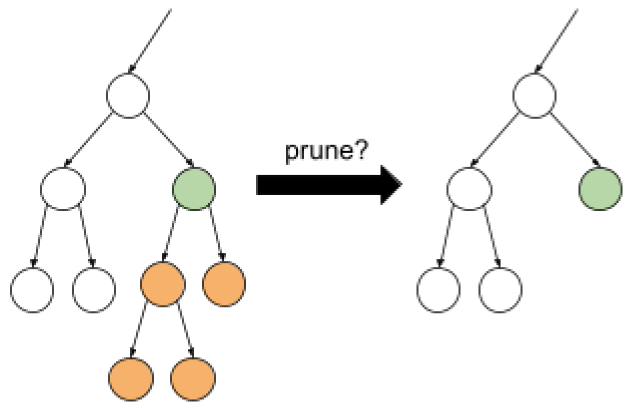
The following drawing illustrates this idea. Here, we test if the validation accuracy of the decision tree would be improved if the non-leaf green node was turned into a leaf; that is, pruning the orange nodes.



Pruning

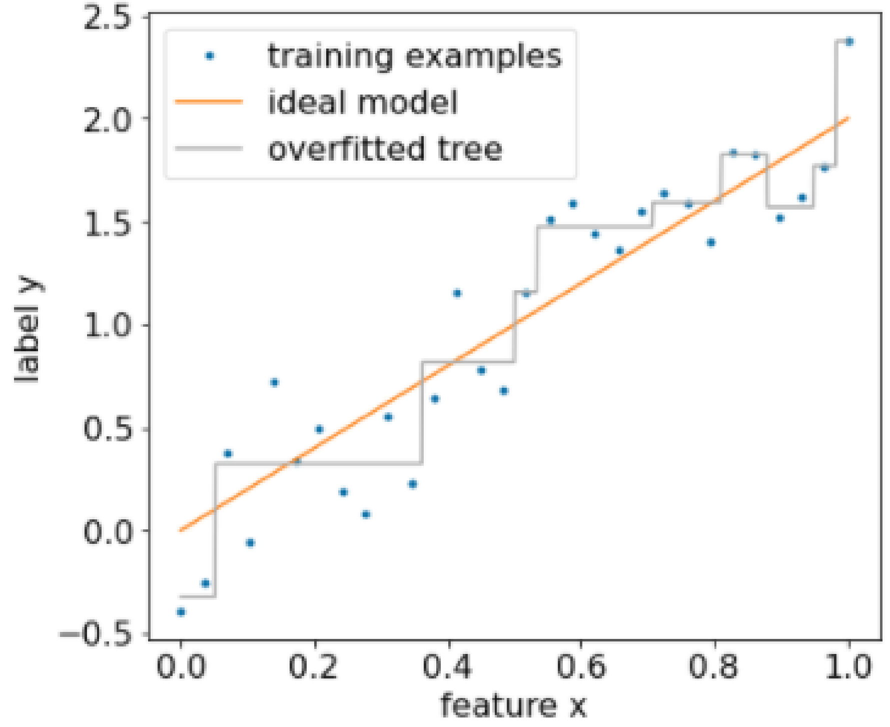
You can also regularize after training by selectively **removing (pruning) certain branches**, that is, by converting certain non-leaf nodes to leaves. A common solution to select the branches to remove is to use a validation dataset. That is, if removing a branch improves the quality of the model on the validation dataset, then the branch is removed.

The following drawing illustrates this idea. Here, we test if the validation accuracy of the decision tree would be improved if the non-leaf green node was turned into a leaf; that is, pruning the orange nodes.



Pruning

The following figure illustrates the effect of using 20% of the dataset as validation to prune the decision tree:



Pruning

Note that using a validation dataset reduces the number of examples available for the initial training of the decision tree.

Many model creators apply multiple criteria. For example, you could do all of the following:

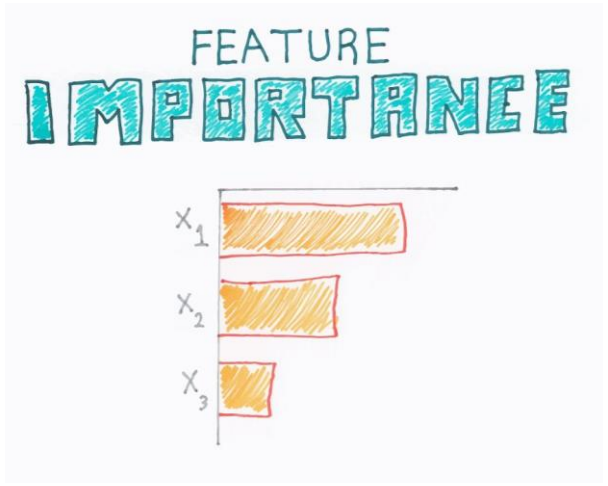
- Apply a minimum number of examples per leaf.
- Apply a maximum depth to limit the growth of the decision tree.
- Prune the decision tree

Feature importance

Variable importance (also known as feature importance) is a score that indicates how "important" a feature is to the model.

For example, if for a given model with two input features "f1" and "f2", the variable importances are {f1=5.8, f2=2.5}, then the feature "f1" is more "important" to the model than feature "f2".

As with other machine learning models, variable importance is a simple way to understand how a decision tree works



Feature importance

You can apply model agnostic variable importances such as permutation variable importances, to decision trees.

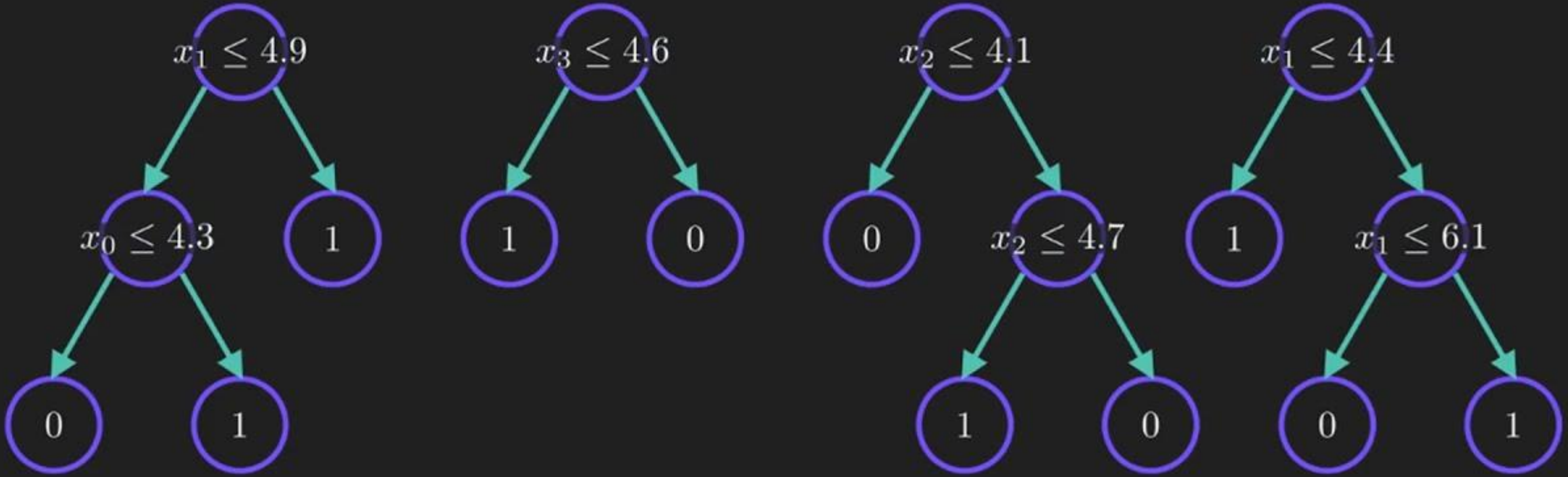
Decision trees also have specific variable importances, such as:

- The sum of the split score with a given variable.
- The number of nodes with a given variable.
- The average depth of the first occurrence of a feature across all the tree paths.

Furthermore, variable importances provide different types of information about:

- the model
- the dataset
- the training process

Random Forest



Random Forest - Introduction

A **random forest** (**RF**) is an ensemble of decision trees in which each decision tree is trained with a specific random noise.

Random forests are the most popular form of decision tree ensemble.

Random Forest - Ensemble?

In machine learning, an ensemble is a collection of models whose predictions are averaged (or aggregated in some way).

If the ensemble models are different enough without being too bad individually, the quality of the ensemble is generally better than the quality of each of the individual models. An ensemble requires more training and inference time than a single model. After all, you have to perform training and inference on multiple models instead of a single model.

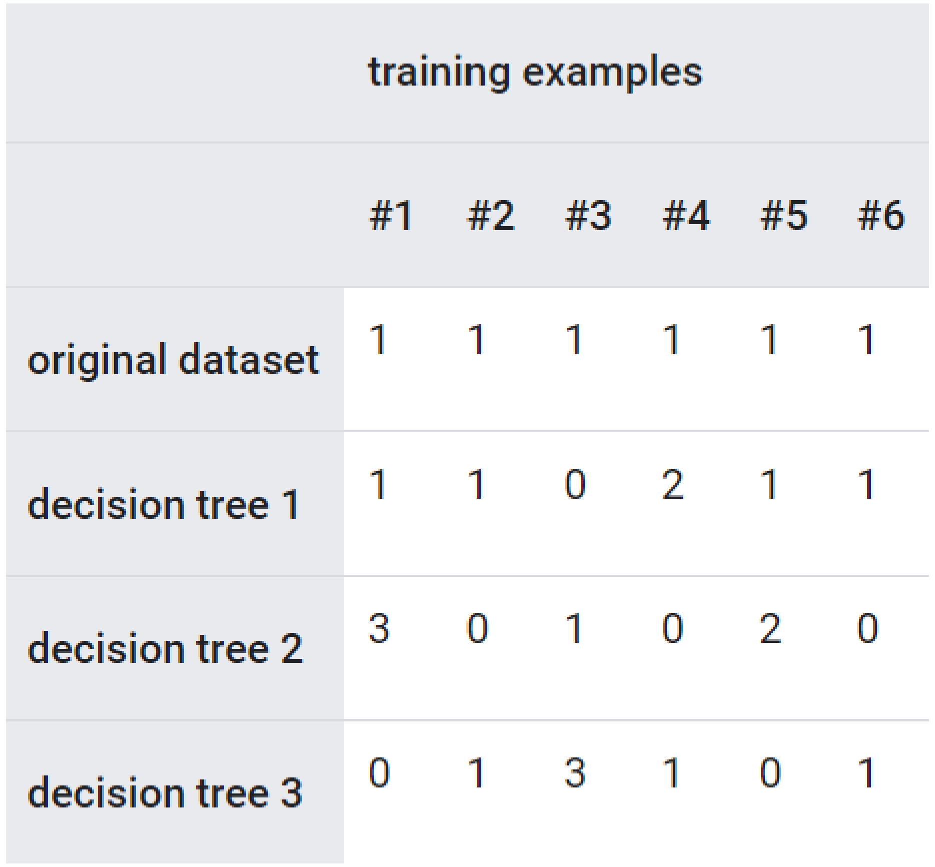
Random Forest - Bagging

- **Bagging** (**b**ootstrap **agg**regat**ing**) means training each decision tree on a random subset of the examples in the training set. In other words, each decision tree in the random forest is trained on a *different subset* of examples.
- Bagging is peculiar. Each decision tree is trained on the same *number* of examples as in the original training set. For example, if the original training set contains 60 examples, then each decision tree is trained on 60 examples. However, bagging only trains each decision tree on a *subset* (typically, 67%) of those examples. So, some of those 40 examples in the subset must be *reused* while training a given decision tree. This reuse is called training "with replacement."

Random Forest - Bagging

Table 6 shows how bagging could distribute six examples across three decision trees. Notice the following:

- Each decision tree trains on a total of six examples.
- Each decision tree trains on a different set of examples.
- Each decision tree reuses certain examples. For example, example #4 is used twice in training decision tree 1; therefore, the learned weight of example #4 is effectively doubled in decision tree 1.



	training examples					
	#1	#2	#3	#4	#5	#6
original dataset	1	1	1	1	1	1
decision tree 1	1	1	0	2	1	1
decision tree 2	3	0	1	0	2	0
decision tree 3	0	1	3	1	0	1

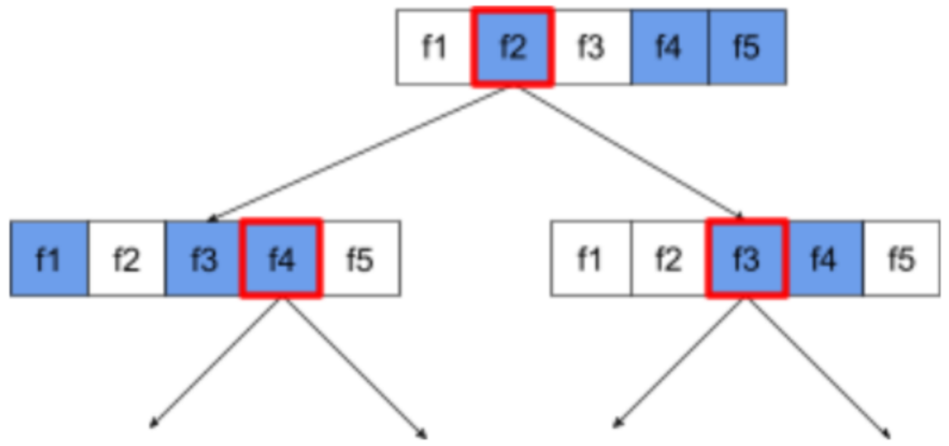
Random Forest – Attribute Sampling

Attribute sampling means that instead of looking for the best condition over all available features, only a random subset of features are tested at each node.

The set of tested features is sampled randomly at each node of the decision tree.

Random Forest – Attribute Sampling

- The following decision tree illustrates the attribute / feature sampling. Here a decision tree is trained on 5 features (f1-f5). The blue nodes represent the tested features while the white ones are not tested. The condition is built from the best tested features (represented with a red outline).
- The ratio of attribute sampling is an important regularization hyperparameter



Random Forest – No Pruning

Individual decision trees in a random forest are trained without pruning. This produces overly complex trees with poor predictive quality. Instead of regularizing individual trees, the trees are ensembled producing more accurate overall predictions.

Random Forest – Generalization

We expect the training and test accuracy of a random forest to differ. The training accuracy of a random forest is generally much higher (sometimes equal to 100%). However, a very high training accuracy in a random forest is normal and does not indicate that the random forest is overfitted.

The two sources of randomness (bagging and attribute sampling) ensure the relative independence between the decision trees. This independence corrects the overfitting of the individual decision trees.

Consequently, the ensemble is not overfitted.

Random Forest – Generalization

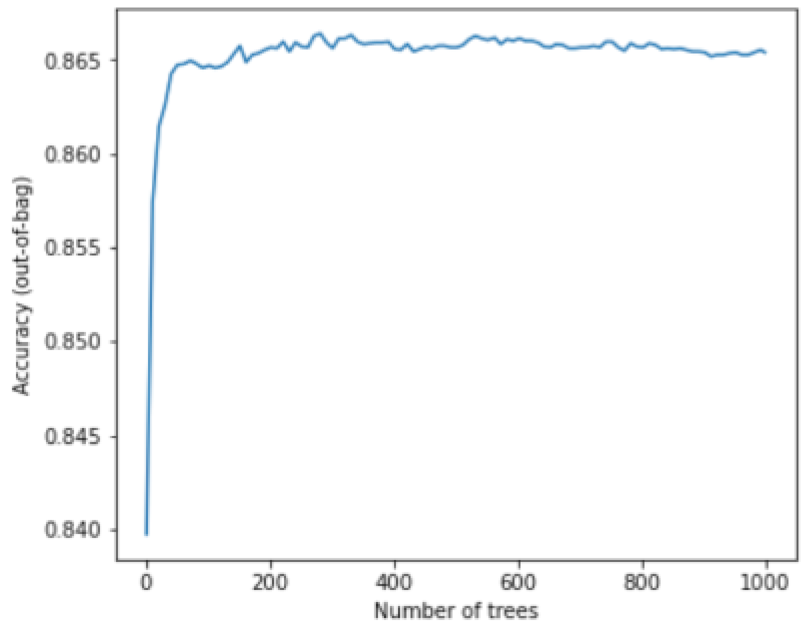
When training a random forest, as more decision trees are added, the error almost always decreases; that is, the quality of the model almost always improves.

Yes, adding more decision trees almost always reduces the error of the random forest. In other words, adding more decision trees cannot cause the random forest to overfit. At some point, the model just stops improving.

Leo Breiman famously said, "Random Forests do not overfit, as more trees are added".

Random Forest – Generalization

For example, the following plot shows the test evaluation of a random forest model as more decision trees are added. The accuracy rapidly improves until it plateaus around 0.865. However, adding more decision trees does not make accuracy decrease; in other words, the model does not overfit. This behavior is (mostly) always true and independent of the hyperparameters.



Random Forest – Pros and Cons

Pros:

- Like decision trees, random forests support natively numerical and categorical features and often do not need feature pre-processing.
- Because the decision trees are independent, random forests can be trained in parallel. Consequently, you can train random forests quickly.
- Random forests have default parameters that often give great results. Tuning those parameters often has little effect on the model.

Cons:

- Because decision trees are not pruned, they can be large. Models with more than 1M nodes are common. The size (and therefore inference speed) of the random forest can sometimes be an issue.
- Random forests cannot learn and reuse internal representations. Each decision tree (and each branch of each decision tree) must relearn the dataset pattern. In some datasets, notably non-tabular dataset (e.g. image, text), this leads random forests to worse results than other methods.

Random Forest and Decision Tree– Recap

- A decision tree is a model composed of a collection of conditions organized hierarchically in the shape of a tree. Conditions fall into various categories:
  - A binary condition has two possible outcomes. A non-binary condition has more than two possible outcomes.
- Training a decision tree involves searching for the best condition at each node. The splitter routine uses metrics such as information gain or Gini to determine the best condition.
- A decision forest is a mode made of multiple decision trees. The prediction of a decision forest is the aggregation of the predictions of its decision trees.
- A random forest is an ensemble of decision trees in which each decision tree is trained with a specific random noise.
- Bagging is a technique in which each decision tree in the random forest is trained on a different subset of examples.