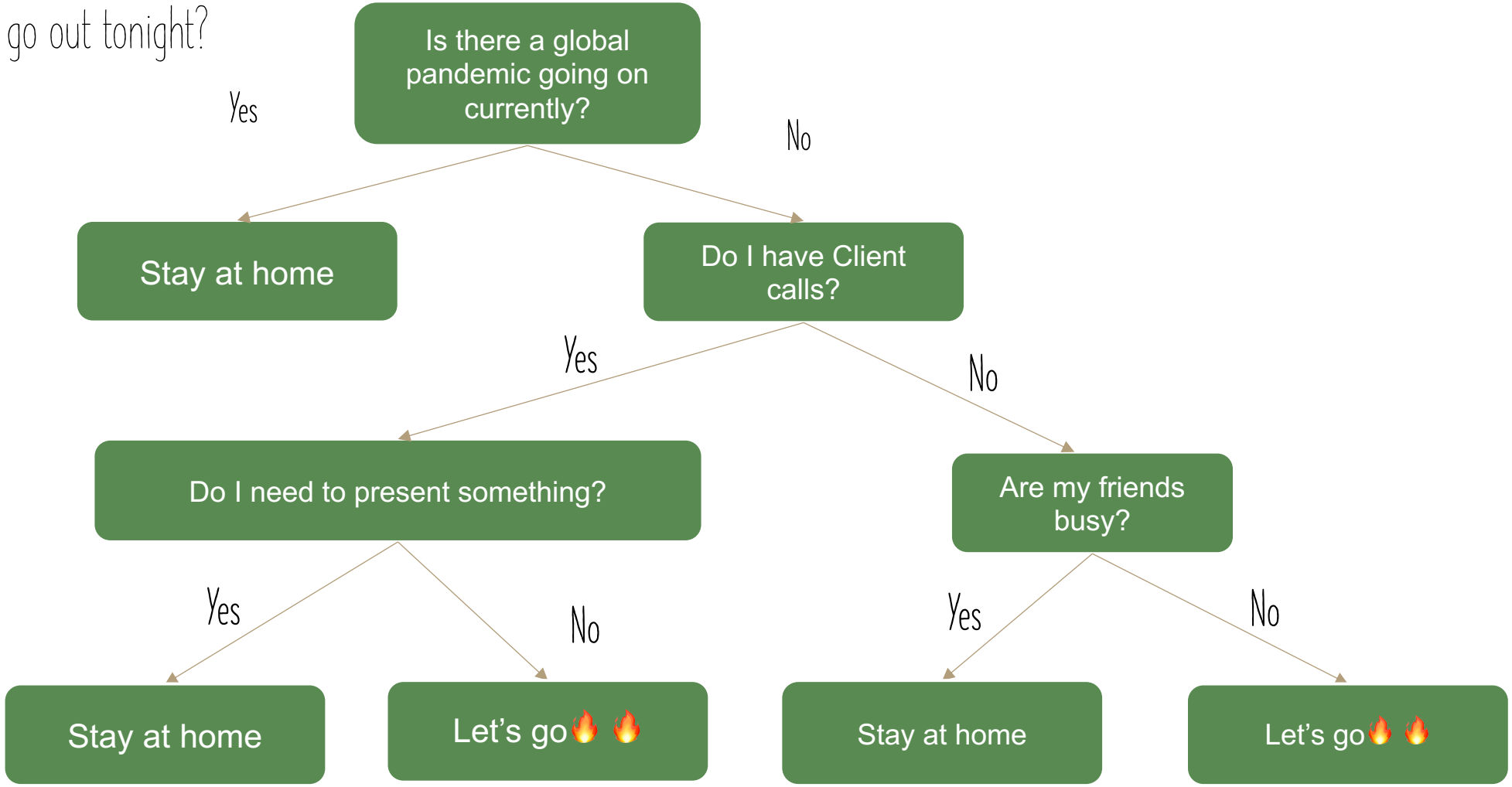
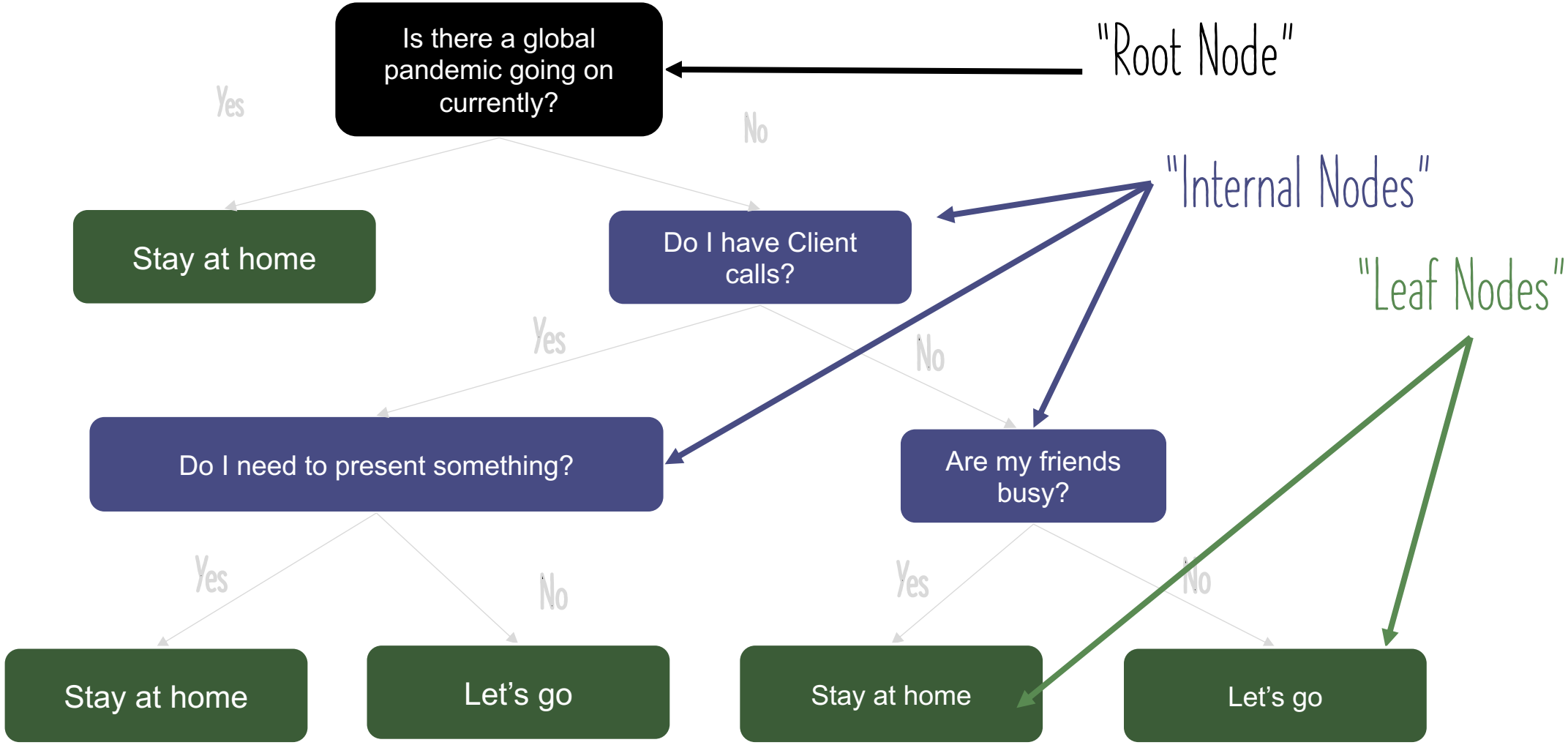

DECISION TREE ENSEMBLES

A SIMPLE DECISION TREE

Should I go out tonight?



JARGONS


Is there a global pandemic going on currently?
"Root Node"
Yes
No
Stay at home
Do I have Client calls?
"Internal Nodes"
"Leaf Nodes"
Yes
No
Do I need to present something?
Are my friends busy?
Yes
No
Yes
No
Stay at home
Let's go
Stay at home
Let's go

WHERE CAN THIS ALGORITHM BE APPLIED ?

- The Humble decision tree is also called by its more modern name: Classification and Regression Trees (CART)
- It can be applied for both classification and regression type settings
- It can have Numerical, Ordered and Categorical inputs as its variables
    - E.g:
        - High, Medium , Low (Ordered)
        - Blue/Red/Green, Yes/No (Categorical)

HOW DOES A DECISION TREE LEARN?

- Greedy Splitting / Recursive Binary Splitting

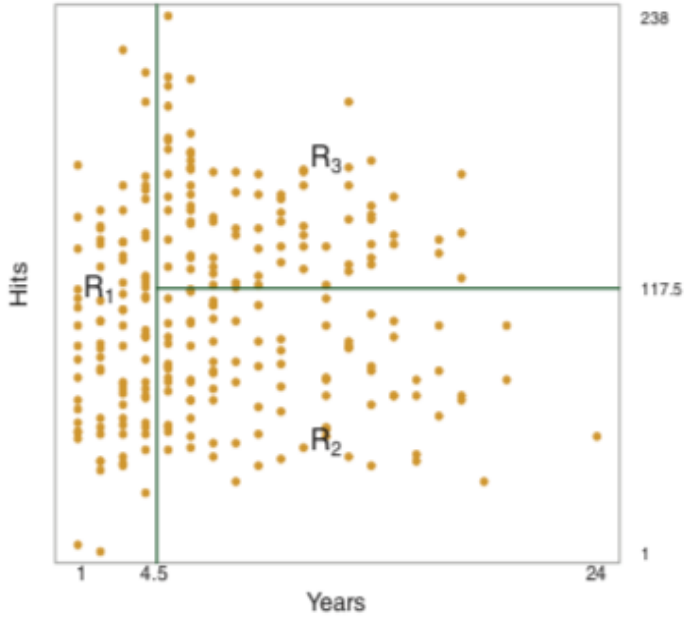
Example:

We use the 'Hitters' dataset to predict Salaries of Baseball Players based on 2 other variables: The Number of Years of experience and Number of Hits that particular player made in last season

Algorithm:

Goal: Minimize RSS (Residual Sum Of Squares): $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$

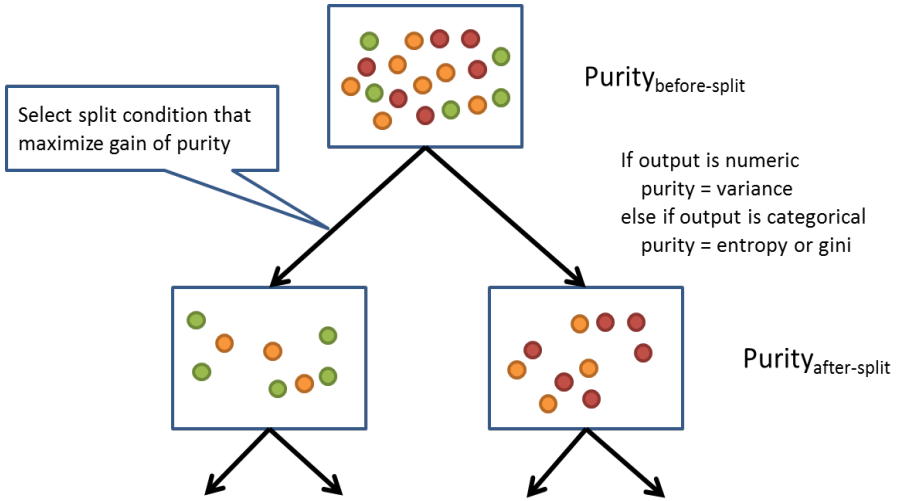
- Begin with Predicting average salaries for all observations
- Iterate through each value of the 2 variables to find a 'Split'. This split will divide the predictor space into 2 regions
- Each Split will reduce the RSS at every iteration. Next split is performed in the 2 sub regions after the split occurs.
- Continue iteration when a stoppage criterion is met. E.G: Continue unless each node contains no less than 5 observations or the tree reaches some maximum depth d.



HOW DOES THIS WORK IN CASE OF CLASSIFICATION?

Basic idea remains the same, in this case the observations are assigned the majority class present in its leaf node.

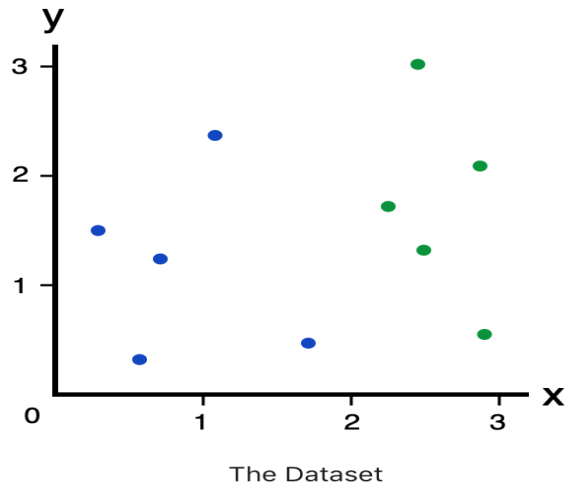
Goal: Instead of Minimizing RSS, we try to improve Node Purity. We try to minimize a measure called GINI Index



Gini Impurity: It is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset.

For a given split with C classes, it is given by: $G = \sum_{i=1}^{C} p(i) * (1 - p(i))$

INTERPRETING THE GINI INDEX



The Dataset

Right now, we have 1 branch with 5 blues and 5 greens.

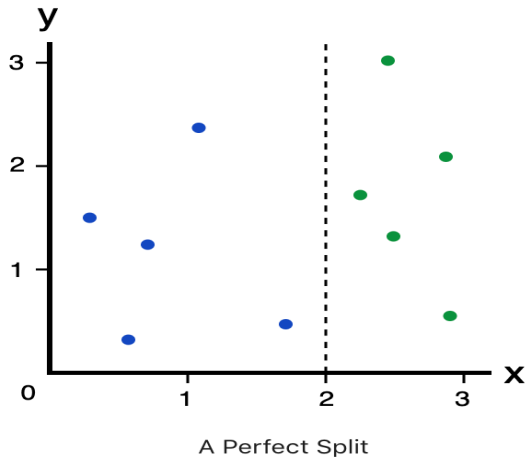
●●●●● ●●●●●

Let's calculate the Gini Impurity of our entire dataset. If the probability of picking a datapoint with class i is p(i) , then the Gini Impurity is calculated as $G = \sum_{i=1}^{C} p(i) * (1 - p(i))$

For the example above, we have C= 2 and p(blue) = 0.5 and p (green) = 0.5

GINI Index = 0.5*(1-0.5) + 0.5*(1-0.5) =0.5

Case 1: Perfect Split



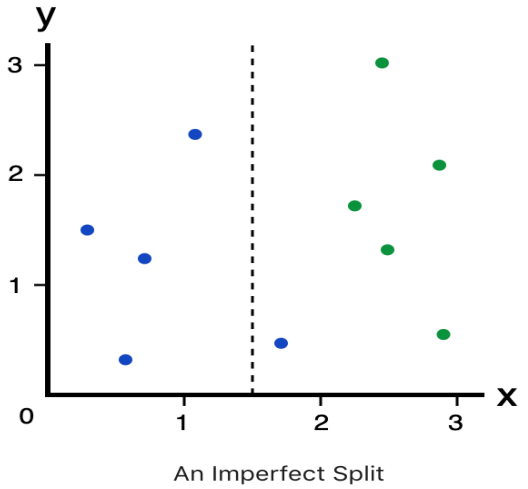
A Perfect Split

G_{Left} =1*(1-1) + 0*(1-0) = 0

G_{Right} =0*(1-0) + 1*(1-1) = 0

Both branches have 0 impurity! The perfect split turned a dataset with 0.5 impurity into 2 branches with 0 impurity!

Case 2: Imperfect Split



An Imperfect Split

G_{Left} =1*(1-1) + 0*(1-0) = 0

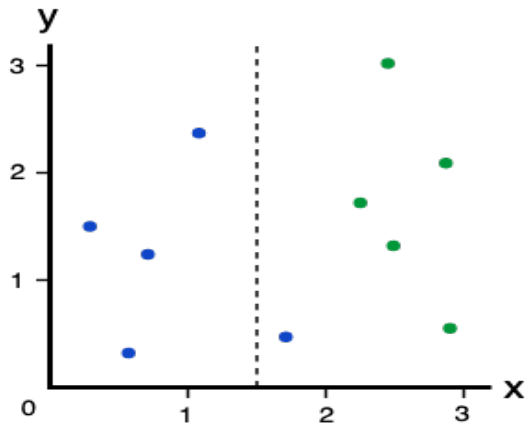
$G_{Right} = \frac{1}{6} * \left(1 - \frac{1}{6}\right) + \frac{5}{6} * \left(1 - \frac{5}{6}\right) = 0.278$

Left Branch has 0 impurity! However Right branch has a Gini Impurity of 0.278!

Source: https://victorzhou.com/blog/gini-impurity/

PICKING THE BEST SPLIT

Here's the imperfect split yet again



We've already calculated the Gini Impurities for:

- Before the split (the entire dataset): 0.50
- Left Branch: 0
- Right Branch: 0.278

Quality of split is given by weighing Impurities of each branch by number of elements it has.
Hence Gini Impurity = (0.4* 0) = (0.6 * 0.278) = 0.167

The amount of impurity we've "removed" with this split is:

0.5 - 0.167 = 0.333 ← Gain

This value (Gain) is used to pick the best split in a decision tree!

For example, it's easy to verify that the Gini Gain of the perfect split on our dataset is 0.5 > 0.333

ALGORITHM

The algorithm stops at a leaf node in any of the below situations:

- All examples in the leaf node are classified correctly by the model
- We cannot find an attribute to split upon.
- The split reduces the GINI Index less than some **ε** (the value for which has to be found experimentally).
- The tree reaches some maximum depth d (also has to be found experimentally).

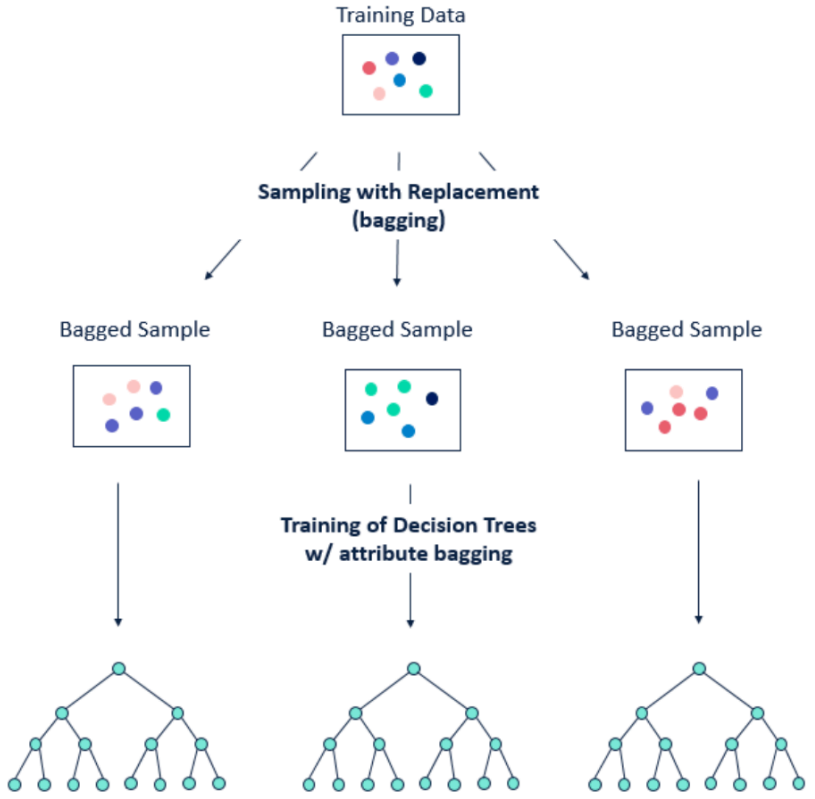
DISADVANTAGES OF DECISION TREES

- Even a small change in input data can at times, cause large changes in the tree structure
- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other more sophisticated regression and classification approaches
- Usually, as the number of levels increases in a decision tree, the model becomes vulnerable to high-variance and might overfit

However, by aggregating many decision trees, using methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved!!!

BAGGING

- Ensemble modeling is a process where multiple diverse models are created to predict an outcome, either by using many different modeling algorithms or using different training data sets.
- The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner.
- Bagging (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree.
- We end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.



ALGORITHM EXPLAINED

Bootstrapping is the process of generating bootstrapped samples from the given dataset. The samples are formulated by randomly drawing the data points with replacement.

1. In Bagging, the bootstrapped samples are first created.
2. Then, either a regression or classification algorithm is applied to each sample.
3. Finally,
    - In the case of **regression**, an average is taken over all the outputs predicted by the individual learners.
    - For **classification** either the most voted class is accepted (**hard-voting**), or the highest average of all the class probabilities is taken as the output (**soft-voting**). This is where **aggregation** comes into the picture.

Bootstrapping Samples

PROS, CONS AND HYPERPARAMETERS (BAGGED TREES)

Pros

- Bagging takes the advantage of ensemble learning wherein multiple weak learner outperform a single strong learner. It helps reduce variance and thus helps us avoid overfitting.
- Improves model accuracy

Cons

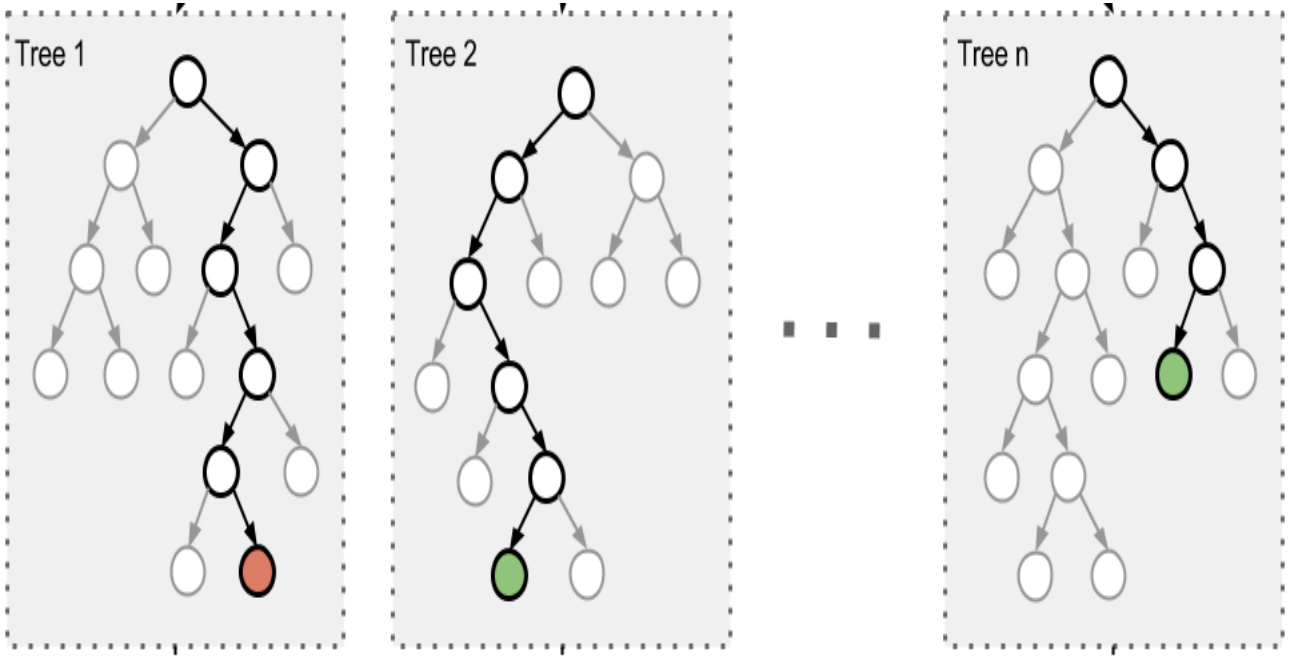
- There is loss of interpretability of the model. There can possibly be a problem of high bias if not modeled properly
- Computationally expensive
- Estimators are **correlated** with each other.

Important Hyper Parameters to tune

- Number of base estimators in the ensemble (number of trees)
- Number of samples to draw for each estimator (Minimum number of samples in each tree)
- Bootstrapped/ Non-Bootstrapped samples
- Max depth

RANDOM FOREST

- In **random forests**, each tree in the ensemble is built from a sample drawn with replacement (i.e. a bootstrap sample) from the training set. In addition, instead of using all the features, a random subset of features is selected, further randomizing the tree.



PROS, CONS AND HYPERPARAMETERS (RANDOM FOREST)

Pros

- The predictive performance can compete with the best supervised learning algorithms
- Reduction in overfitting: by averaging several trees, there is a significantly lower risk of overfitting.
- Less variance: By using multiple trees, you reduce the chance of stumbling across a classifier that doesn't perform well because of the relationship between the train and test data.

Cons

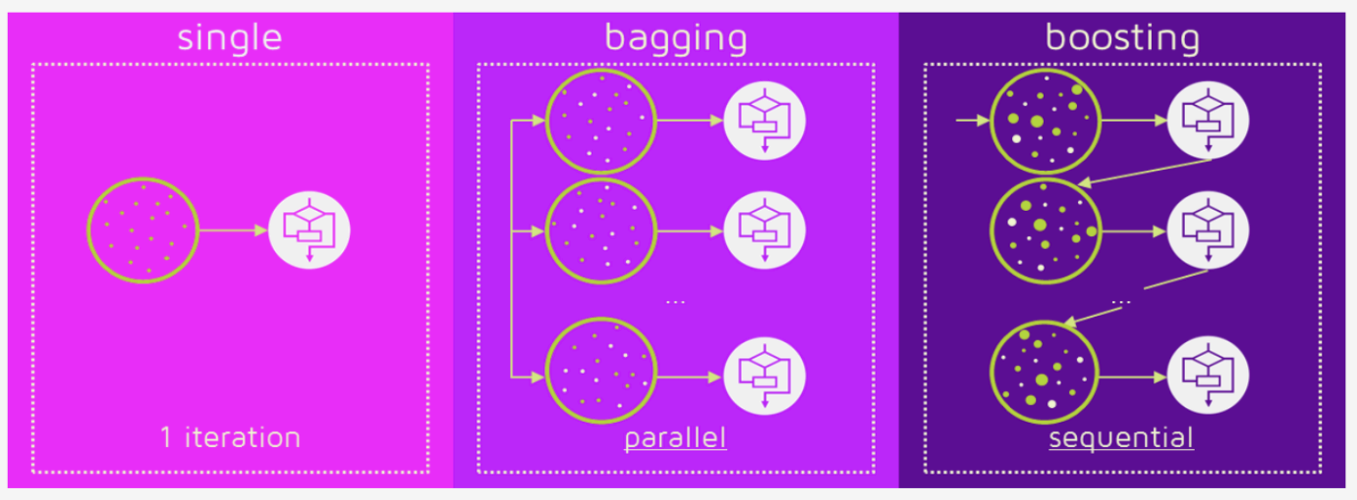
- It's more complex.
- It's hard to visualize the model or understand why it predicted something.
- It's more difficult to implement.

Important Hyperparameters to tune

- Number of base estimators in the ensemble (number of trees)
- Number of samples to draw for each estimator (Minimum number of samples in each tree)
- Bootstrapped/ Non-Bootstrapped samples
- Max depth

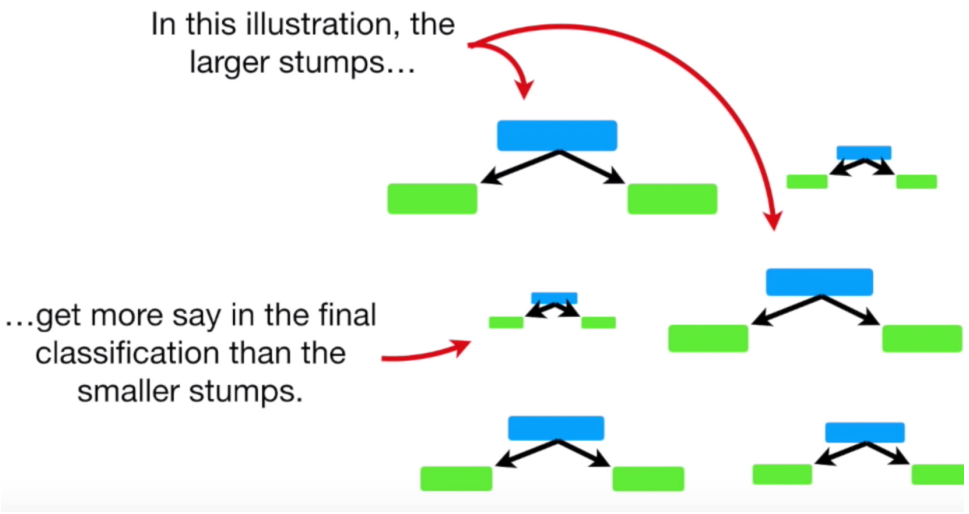
BOOSTING

- Boosting algorithms seek to improve the prediction power by training a sequence of weak models, each compensating the weaknesses of its predecessors. This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors.
- Types of Boosting:
  - Adaboost
  - Gradient Boost
  - XGBoost



ADABOOST (ADAPTIVE BOOSTING)

- Adaboost combines multiple weak learners into a single strong learner. The weak learners in AdaBoost are decision trees with a single split, called decision stumps.
- A tree with just a node and 2 leaves is called a Stump!
- In Adaboost, some stumps get more weight than others
- The errors that the first stump makes will influence how the second stump will be made and so on..



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

Stump

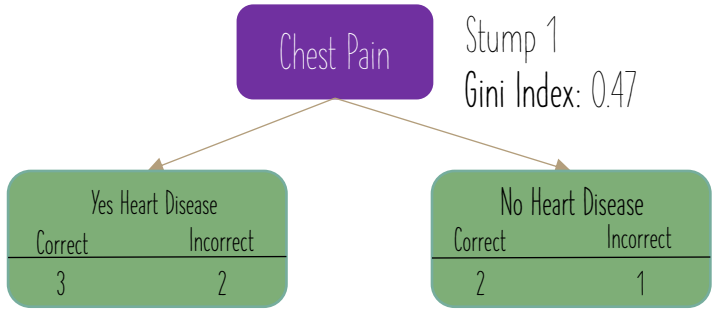
Source: StatQuest

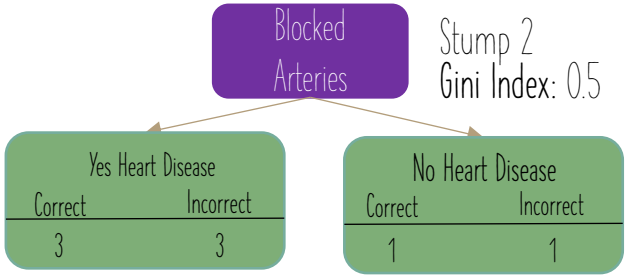
ADABOOST WORKING ALGORITHM - EXAMPLE (1/4)

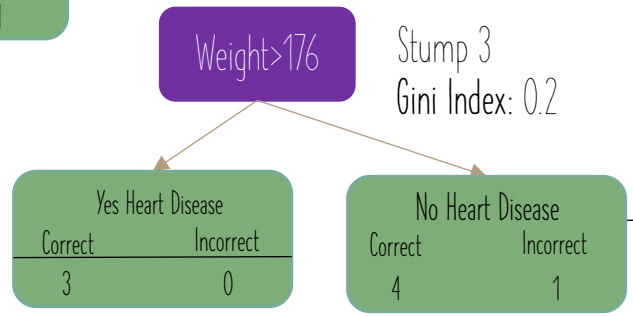
- We are trying to predict if a patient can suffer from a heart disease given information about Chest Pain, Blocked Arteries and Weight.

Chest Pain	Blocked Arteries	Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

How do we select the first stump?







ADABOOST WORKING ALGORITHM - EXAMPLE (2/4)

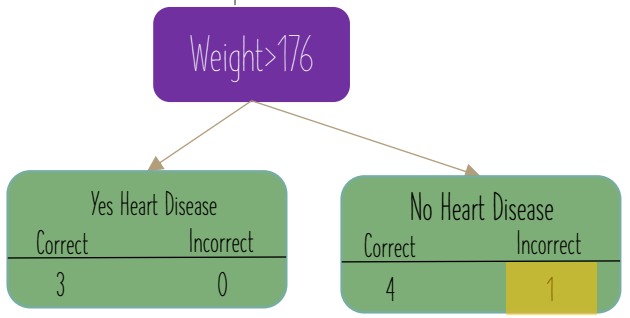
Chest Pain	Blocked Arteries	Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

How Much say will this stump get?

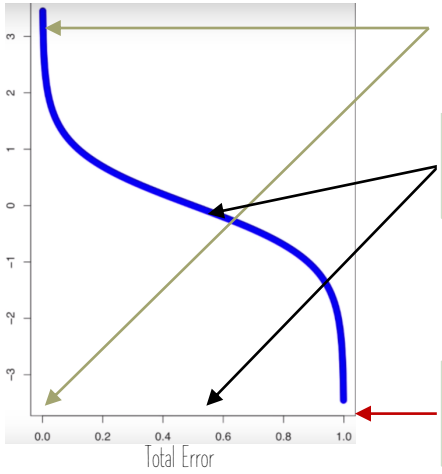
- Based on how well it classifies samples!

Total Error of a stump: Sum of the weights of the incorrectly classified samples

First Stump in the Forest



$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right) = \frac{1}{2}\left(Log\frac{1-\frac{1}{8}}{\frac{1}{8}}\right) = \frac{1}{2}\left(Log\frac{5}{3}\right) = 0.25$$



If the stump does a good job, it has a higher amount of say in the final prediction

If the stump has an error of 0.5, then the decision tree is no better than a randomized classification and it has no say in the final prediction

If a stump consistently gives you opposite results, it has Negative Amount of say which will reverse the stump's predictions

ADABOOST WORKING ALGORITHM - EXAMPLE (3/4)

Chest Pain	Blocked Arteries	Weight	Heart Disease	Sample Weight	Sample Weight 2 Normalized
Yes	Yes	205	Yes	0.125	0.114
No	Yes	180	Yes	0.125	0.114
Yes	No	210	Yes	0.125	0.114
Yes	Yes	167	Yes	0.125	0.203
No	Yes	156	No	0.125	0.114
No	Yes	125	No	0.125	0.114
Yes	No	168	No	0.125	0.114
Yes	Yes	172	No	0.125	0.114

Modify the weights so that the next tree will build on the errors of the first stump!

New Sample weight = $sample\ weight * e^{-Amount\ of\ Say}$

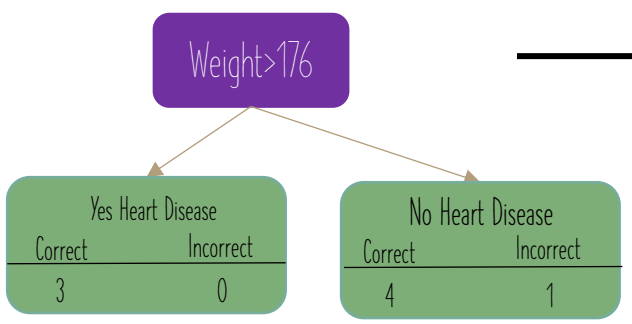
Normalized Sample Weights

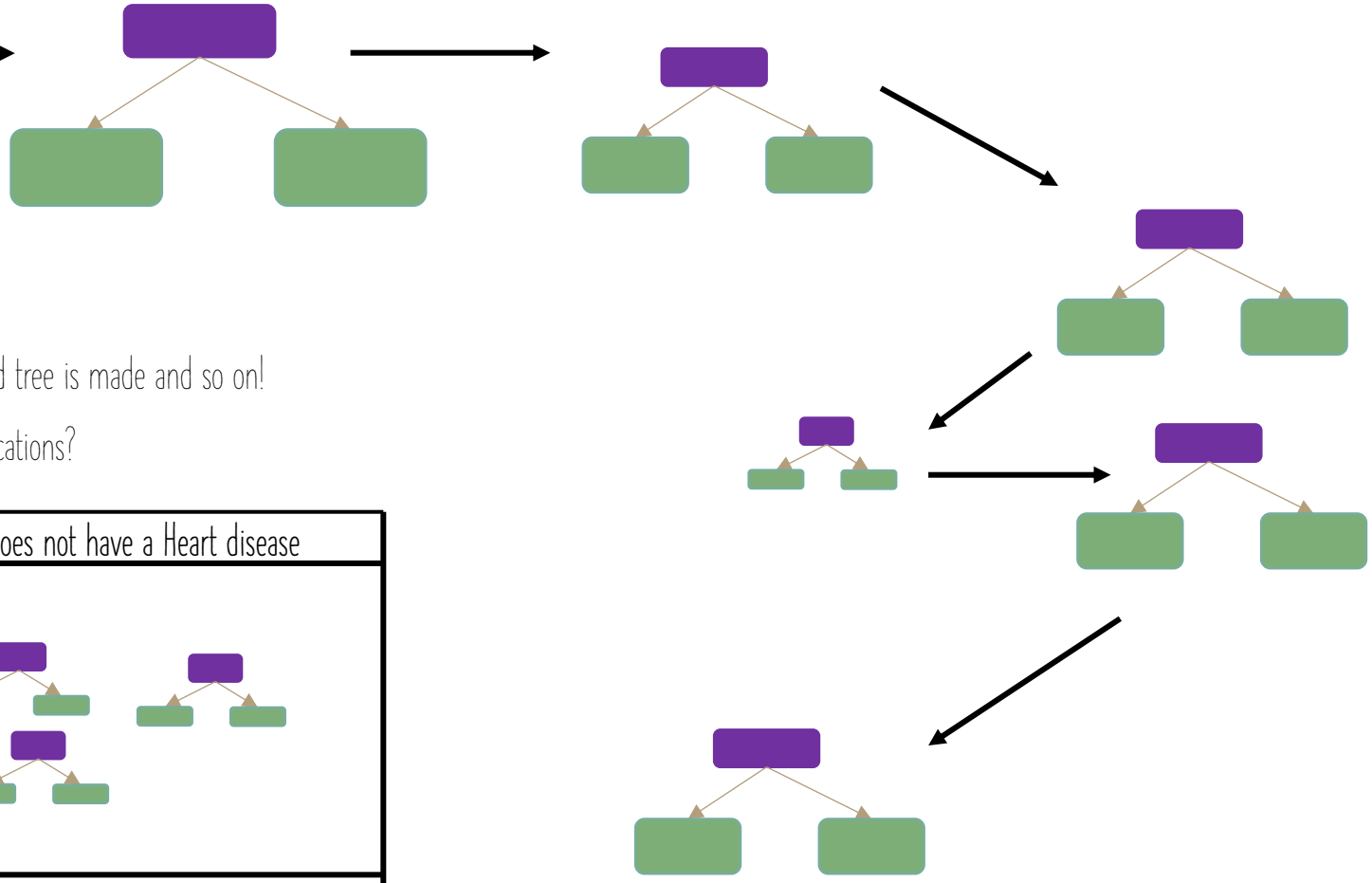
Now we can use the new sample weights to create the second stump!

New Sample weight = $sample\ weight * e^{Amount\ of\ Say}$

- We will use the weighted Gini Index to calculate which variable should the stump split next.
- The weighted Gini index will put more emphasis on correctly classifying the sample with higher weight.

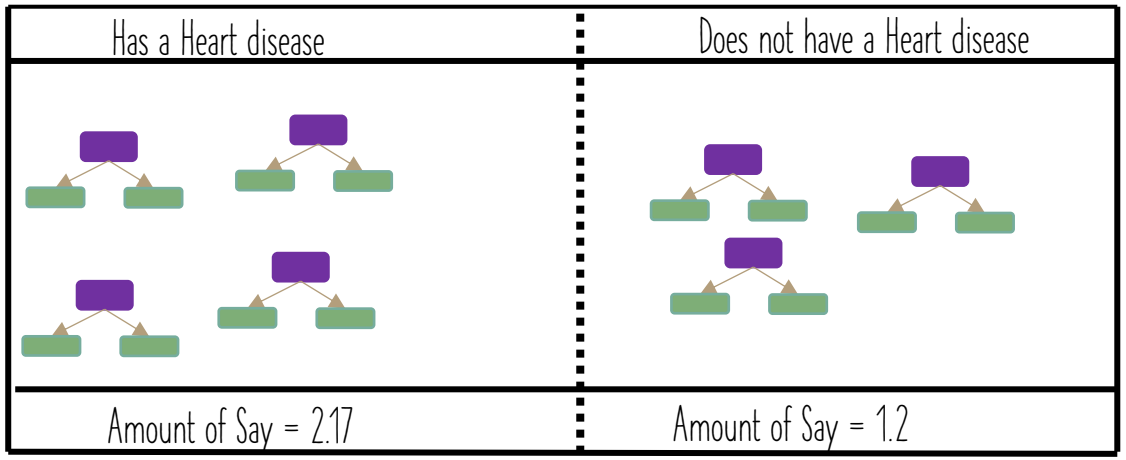
ADABOOST WORKING ALGORITHM - EXAMPLE (4/4)





- Errors that the first tree makes Influence how the second tree is made and so on!

How does a forest of stumps made by Adaboost make classifications?



PROS, CONS AND HYPERPARAMETERS (BOOSTED TREES)

Pros

- It has the flexibility to be combined with any machine learning algorithm
- Each tree/stump is weighted and not given equal say as compared to a Bagged/ Random Forest model

Cons

- Sensitive to noisy data and outliers
- Requires careful tuning of different hyper-parameters
- Weak classifiers being too weak can lead to low margins and overfitting

Important Hyperparameters to tune

- Number of weak learners to train
- Learning rate
- Maximum number of splits (depth of tree)
- Minimum Samples in a Split
- Maximum number of Leaf Nodes

HOW TO TUNE HYPERPARAMETERS?

- Hyperparameter Tuning: It is the process of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically weights) are learned by the ML algorithm
  - Grid Search
  - Random Search
  - Bayesian Optimization

Grid Search

- The traditional way of performing hyperparameter optimization has been *grid search*, or a *parameter sweep*, which is simply exhaustively sweeping through a manually specified subset of the hyperparameter space of a learning algorithm

N_estimators	500	800	1500	2000
Max_features	Auto	Sqrt	Log2	log10
Max_depth	10	20	30	40
Min_samples_split	5	10	15	20

Random Search

- Implements a randomized search over parameters, where each setting is sampled from a distribution over possible parameter values.
- Random search is surprisingly efficient compared to grid search. It will usually find a "close-enough" value in far fewer iterations.
- Random search is always run with a limit on the number of search iterations(n_iter).

Bayesian Optimization

- Bayesian optimization builds a probabilistic model of the function mapping from hyperparameter values to the objective evaluated on a validation set. By iteratively evaluating a promising hyperparameter configuration based on the current model, and then updating it, Bayesian optimization, aims to gather observations revealing as much information as possible about this function, in particular, the location of the optimum.
- Often most efficient algorithm computationally