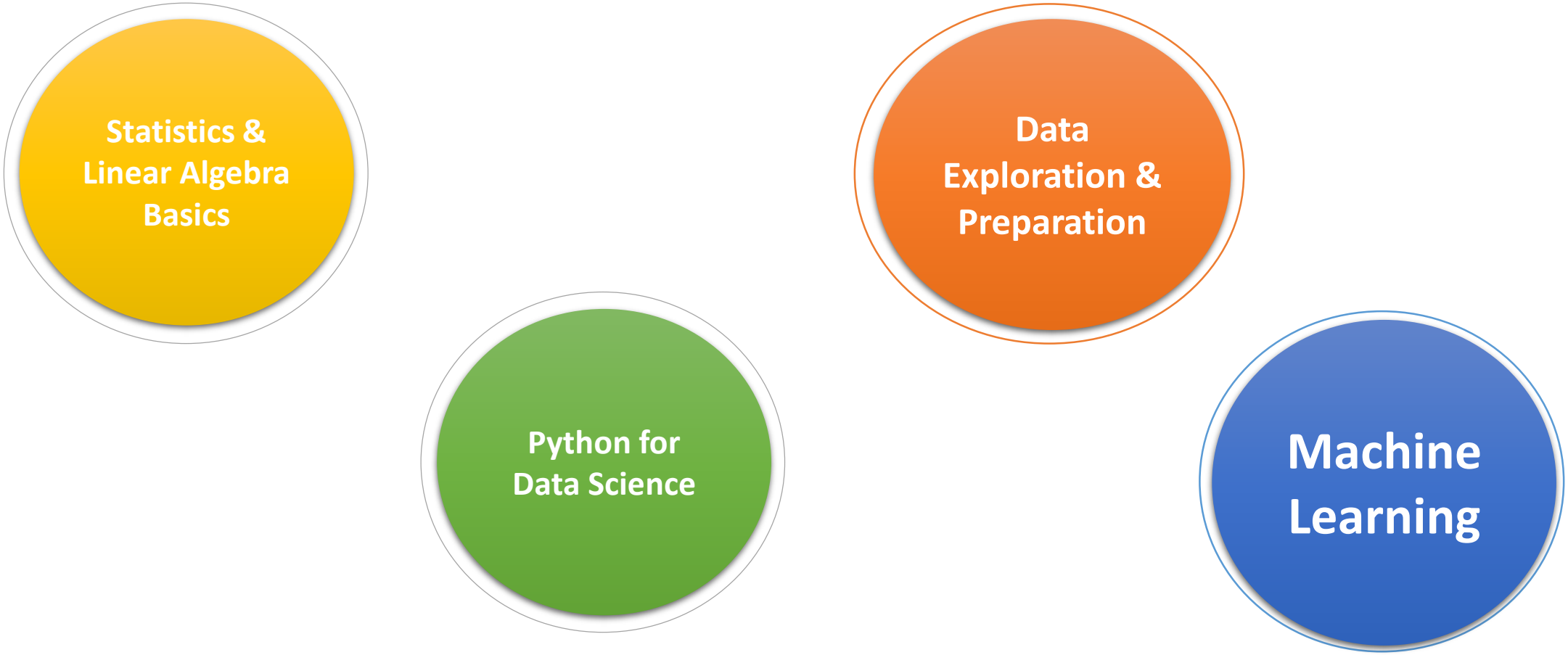
Data Science & Machine Learning

Eman Raslan

Course Schedule

Week #	Day1	Day2
1	24-Nov-2023	
2	1-Dec-2023	2-Dec-2023
3	8-Dec-2023	9-Dec-2023
4	15-Dec-2023	16-Dec-2023
5	22-Dec-2023	23-Dec-2023
6	29-Dec-2023	30-Dec-2023
7	5-Jan-2023	6-Jan-2023
8	12-Jan-2023	13-Jan-2023
	19-Jan-2023	
Project		

Course Agenda



Decision Tree

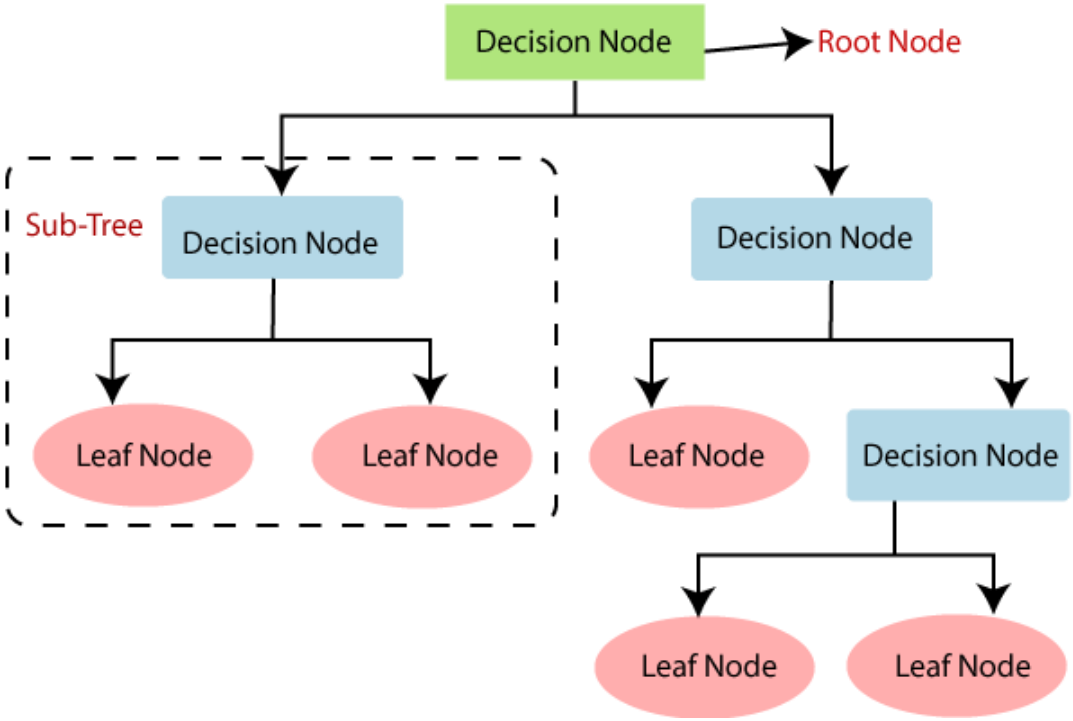
Decision Algorithm

Decision Tree algorithm

- Decision Tree algorithm belongs to the family of supervised learning algorithms.
- Unlike other supervised learning algorithms, decision tree algorithm can be used for solving **regression and classification problems** too.

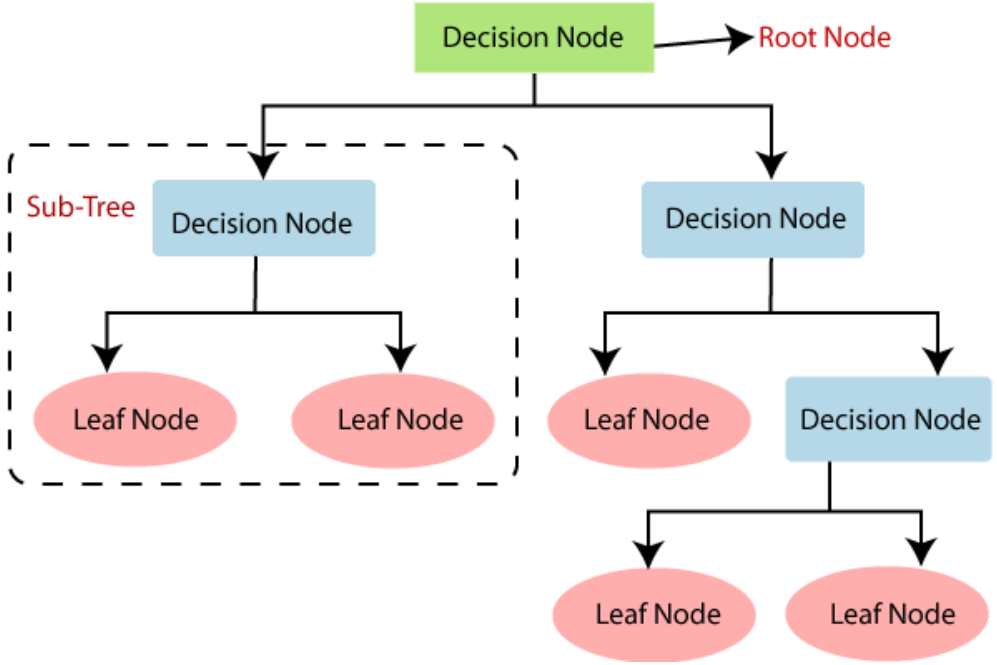
Decision Tree

- **A root node**: this is the node that begins the splitting process by finding the variable that best splits the target variable
- **Node purity**: Decision nodes are typically impure, or a mixture of both classes of the target variable (0,1 or green and red dots in the image). Pure nodes are those that have one class — hence the term pure. They either have green or red dots only in the image.
- **Decision nodes**: these are subsequent or intermediate nodes, where the target variable is again split further by other variables
- **Leaf nodes** or **terminal nodes:** are pure nodes, hence are used for making a prediction of a numerical or class is made.



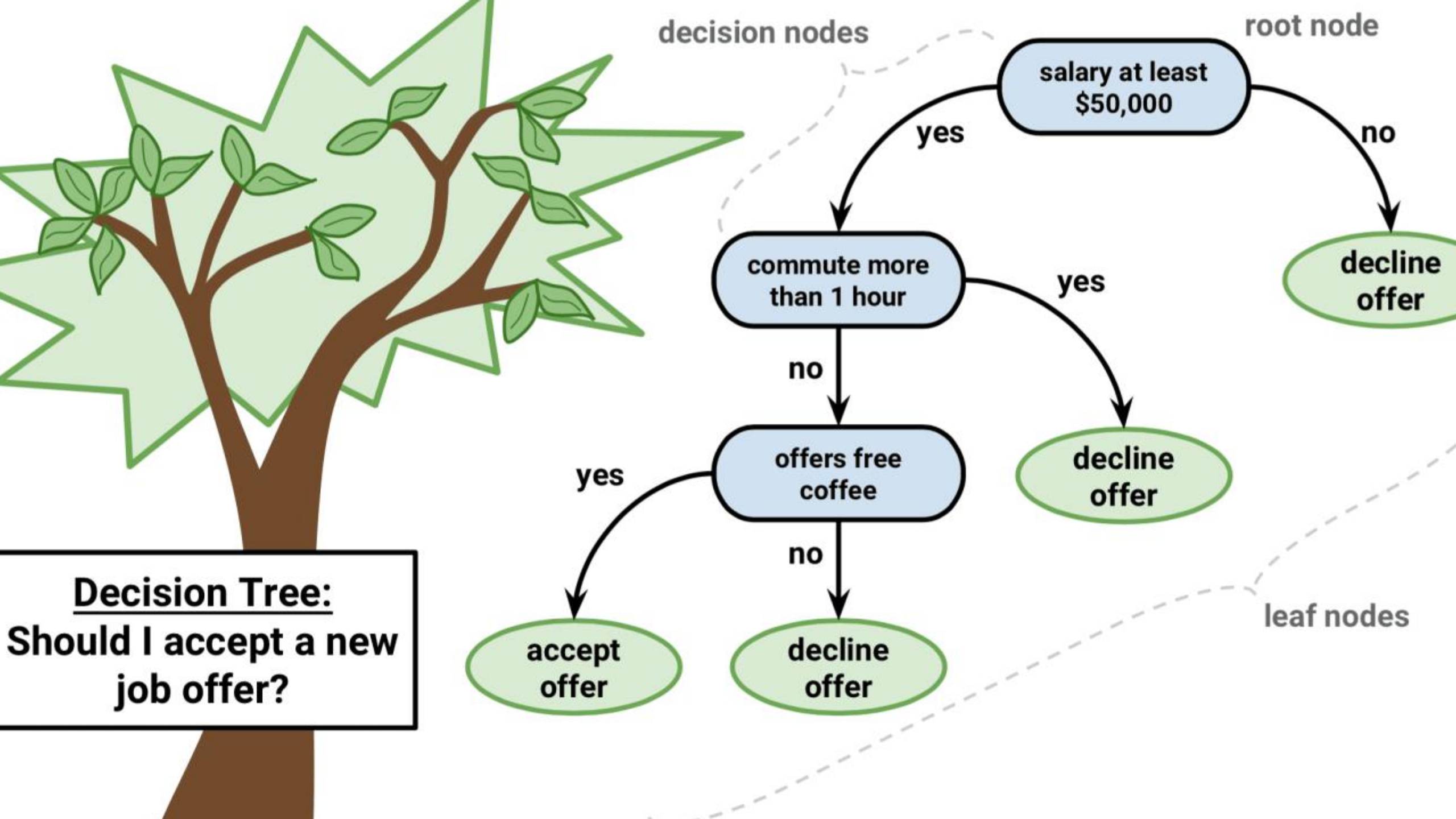
Decision Tree Algorithm Pseudocode

1. Place the best attribute of the dataset at the **root** of the tree.
2. Split the training set into **subsets**. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find **leaf nodes** in all the branches of the tree.



Decision Tree Algorithm Pseudocode

- Assign all training instances to the root of the tree. Set curent node to root node.
- For each attribute
  - Partition all data instances at the node by the value of the attribute.
  - Compute the information gain ratio from the partitioning.
- Identify feature that results in the greatest information gain ratio. Set this feature to be the splitting criterion at the current node.
  - If the best information gain ratio is 0, tag the current node as a leaf and return.
- Partition all instances according to attribute value of the best feature.
- Denote each partition as a child node of the current node.
- For each child node:
  - If the child node is “pure” (has instances from only one class) tag it as a leaf and return.
  - If not set the child node as the current node and recurse to step 2


decision nodes
root node
salary at least $50,000
yes
no
commute more than 1 hour
yes
decline offer
no
offers free coffee
yes
decline offer
yes
no
accept offer
decline offer
leaf nodes
Decision Tree:
Should I accept a new job offer?

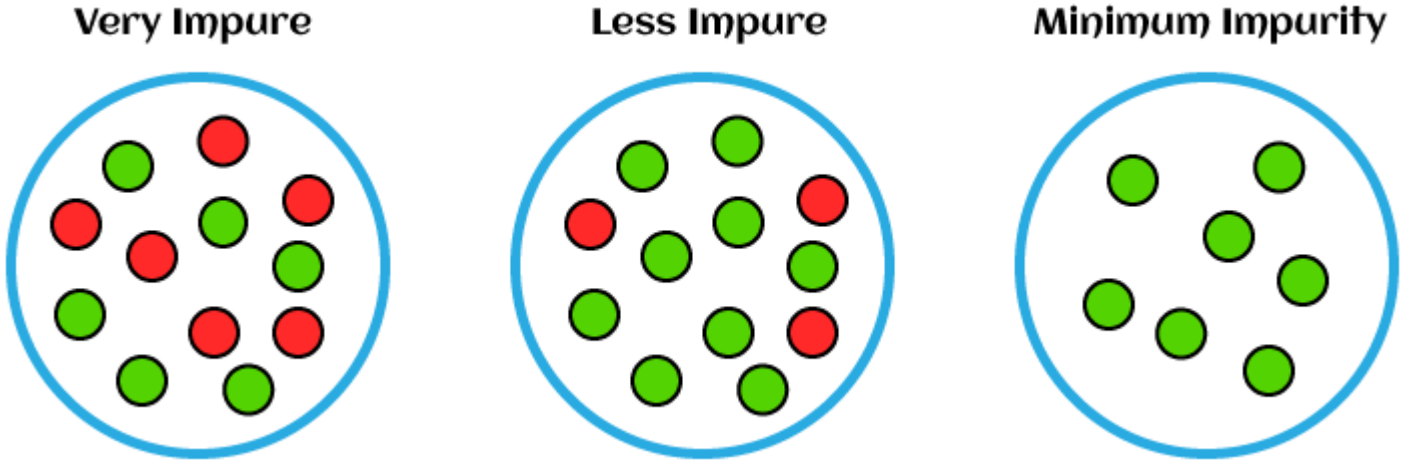
Splitting criteria

- In decision trees, There are some methods to determine the splitting criteria as We need something to measure and compare the impurity.
- Some of the methods are Gini Impurity, Entropy, Information Gain, Gain Ratio, Reduction in variance, Chi square .

Entropy

- **Entropy** is a scientific concept as well as a measurable physical property that is most commonly associated with a state of disorder, randomness, or uncertainty.
- In information theory, the **entropy** of a random variable is the average level of "surprise", or "uncertainty" inherent to the variable's possible outcomes.
- In the context of Decision Trees, entropy is a measure of disorder or impurity in a node.
- Thus, a node with more variable composition, such as 2Pass and 2 Fail would be considered to have higher Entropy than a node which has only pass or only fail.

Entropy



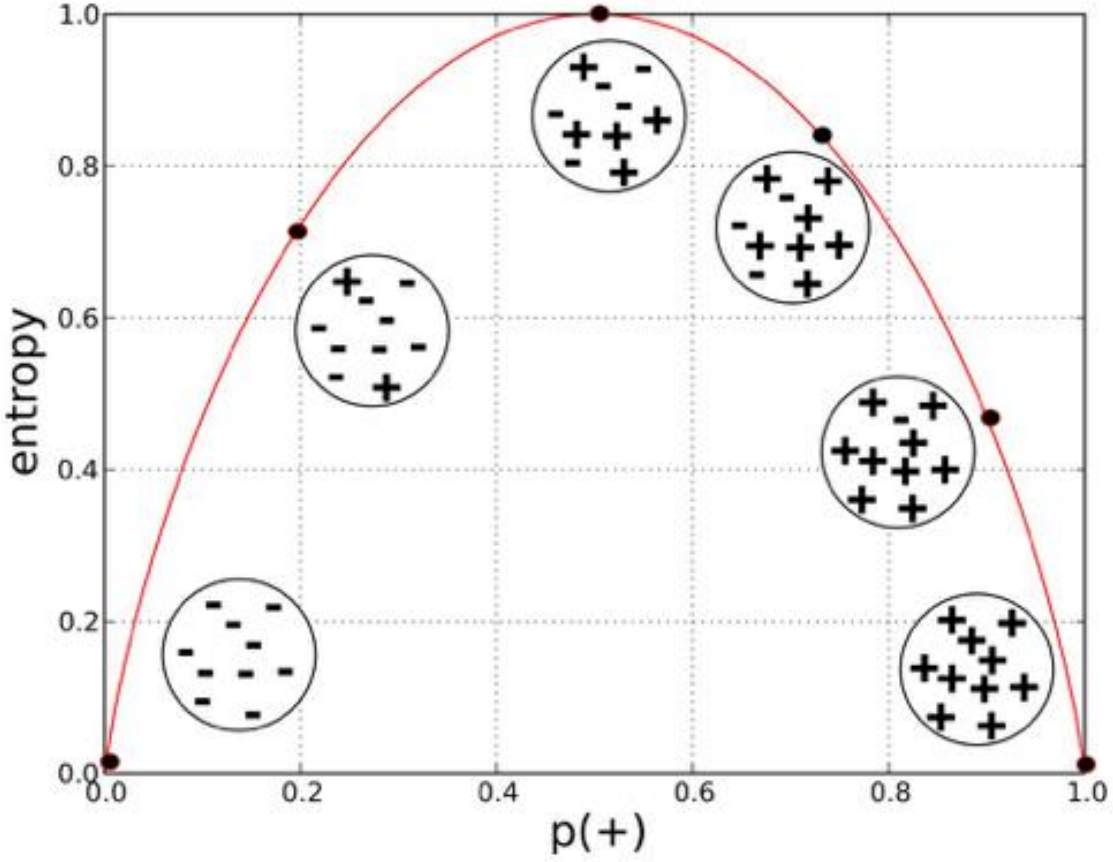
$$E = -\sum_{i=1}^{n} p_i \, log_2(p_i)$$

P_{pass} = Probability of passing/Total no. of instances = 9/15

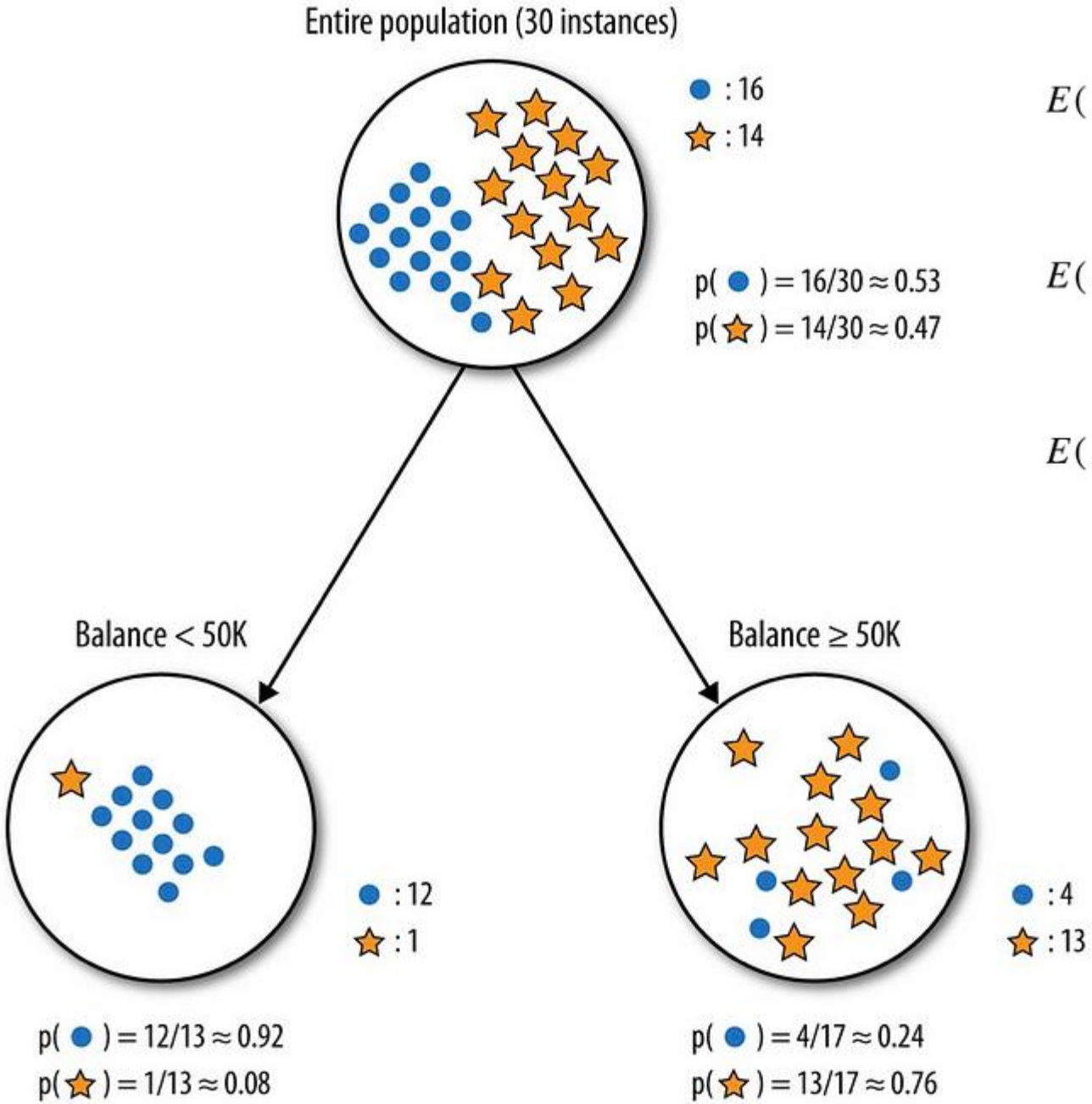
P_{fail}= Probability of failing/Total no. of instances = 6/15

$$E = -(P_{pass} \, log_2(P_{pass}) + P_{fail} \, log_2(P_{fail})$$

Entropy



- The maximum level of entropy or disorder is given by 1 and minimum entropy is given by a value 0.
- Leaf nodes which have all instances belonging to 1 class would have an entropy of 0.
- Whereas, the entropy for a node where the classes are divided equally would be 1.



$$E(Parent) = -\frac{16}{30}\log_2\left(\frac{16}{30}\right) - \frac{14}{30}\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(Balance < 50K) = -\frac{12}{13}\log_2\left(\frac{12}{13}\right) - \frac{1}{13}\log_2\left(\frac{1}{13}\right) \approx 0.39$$

$$E(Balance > 50K) = -\frac{4}{17}\log_2\left(\frac{4}{17}\right) - \frac{13}{17}\log_2\left(\frac{13}{17}\right) \approx 0.79$$

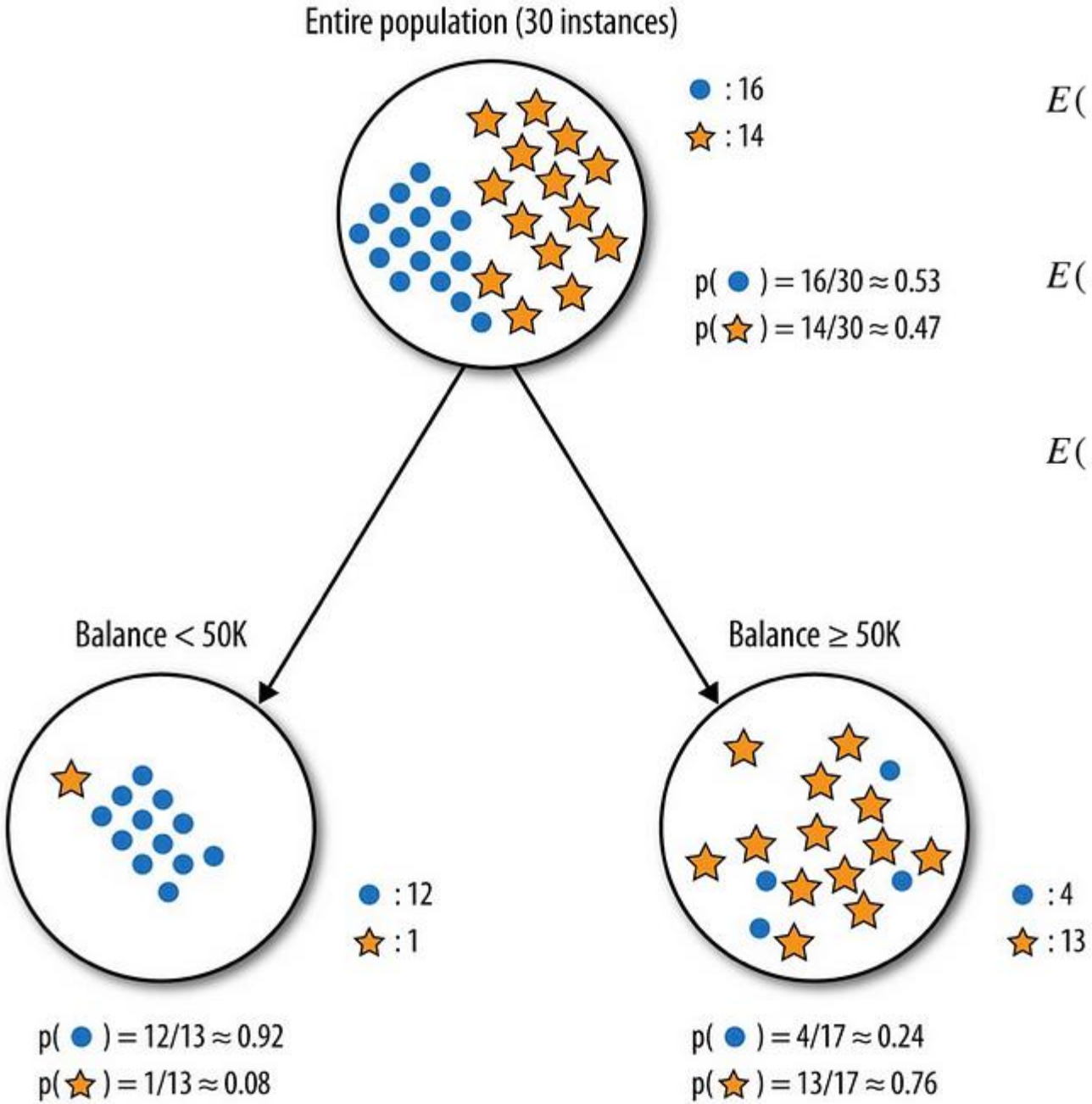
Information Gain

Which feature should we test to add a new node?

- Information is anything that decreases our uncertainty.
- The gain cannot be negative
- The feature with the largest expected amount is the best choice.

$$Information\ Gain = Entropy_{parent} - Entropy_{children}$$

$$IG(Y, X) = E(Y) - E(Y|X)$$



$$E(Parent) = -\frac{16}{30}\log_2\left(\frac{16}{30}\right) - \frac{14}{30}\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(Balance < 50K) = -\frac{12}{13}\log_2\left(\frac{12}{13}\right) - \frac{1}{13}\log_2\left(\frac{1}{13}\right) \approx 0.39$$

$$E(Balance > 50K) = -\frac{4}{17}\log_2\left(\frac{4}{17}\right) - \frac{13}{17}\log_2\left(\frac{13}{17}\right) \approx 0.79$$

Weighted Average of entropy for each node:

$$E(Balance) = \frac{13}{30} \times 0.39 + \frac{17}{30} \times 0.79$$

$$= 0.62$$

$$IG(Parent, Balance) = E(Parent) - E(Balance)$$

$$= 0.99 - 0.62$$

$$= \mathbf{0.37}$$

Information Gain

$$IG(t) = -\sum_{i=1}^{m} p(c_i)\log p(c_i)$$

$$+ p(t)\sum_{i=1}^{m} p(c_i \mid t)\log p(c_i \mid t) + p(\bar{t})\sum_{i=1}^{m} p(c_i \mid \bar{t})\log p(c_i \mid \bar{t})$$

c_i represents the i th category, $P(c_i)$ is the probability of the i th category.

$P(t)$ and $P(\bar{t})$ are the probabilities that the term t appears or not in the documents.

$P(c_i|t)$ is the conditional probability of the i th category given that term t appeared, and $P(c_i|\bar{t})$ is the conditional probability of the i th category given that term t does not appeared.

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy
Recent	High	Yes	Buy
Old	Low	No	Don't buy
Recent	High	No	Don't buy

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗

Root node

✓ 2 instances
✗ 2 instances

Calculating Entropy for the root node

$$E = -\left(P(\checkmark) * \log_2(P(\checkmark)) + P(\times) * \log_2(P(\times))\right)$$

Probability formula:

$$P(\checkmark) = \frac{\text{count of } \checkmark}{\text{total examples}}$$

$P(\checkmark) = 2/4 = 0.5$

$P(\times) = 2/4 = 0.5$

Plugging these values in the formula we get:

$$E = -\left(0.5 * \log_2(0.5) + 0.5 * \log_2(0.5)\right)$$

$$E = 1$$

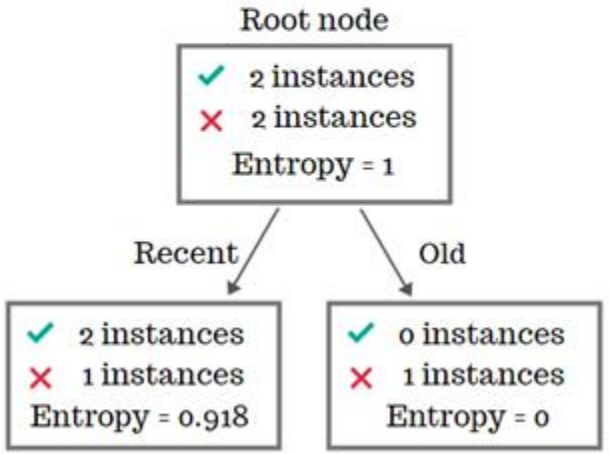
Information Gain (IG)

IG = Entropy (Parent) - weighted_avg * Entropy (Children)

Entropy (Parent) = 1

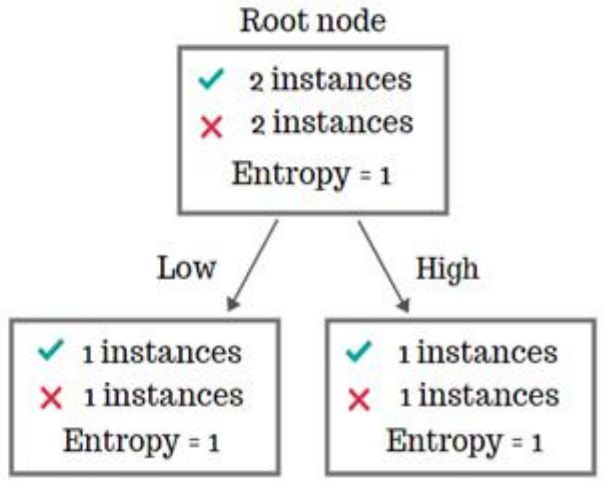
Information gain for Age

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗



Information gain for Milage

Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗

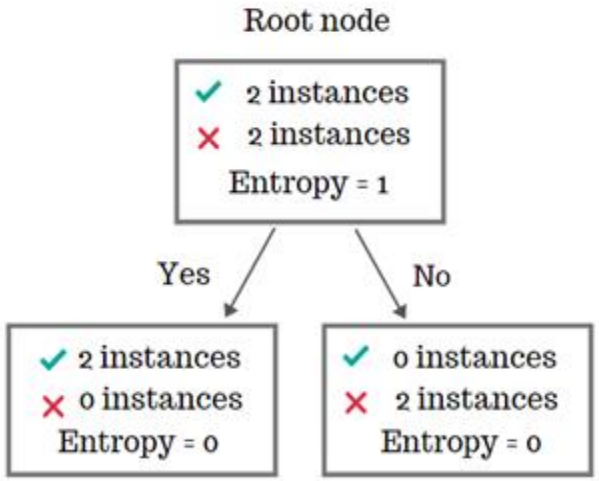


Children Entropy = 2/4 (1) + 2/4 (1) = 1

Information gain = 1 - 1 = 0

Information gain for Road Tested

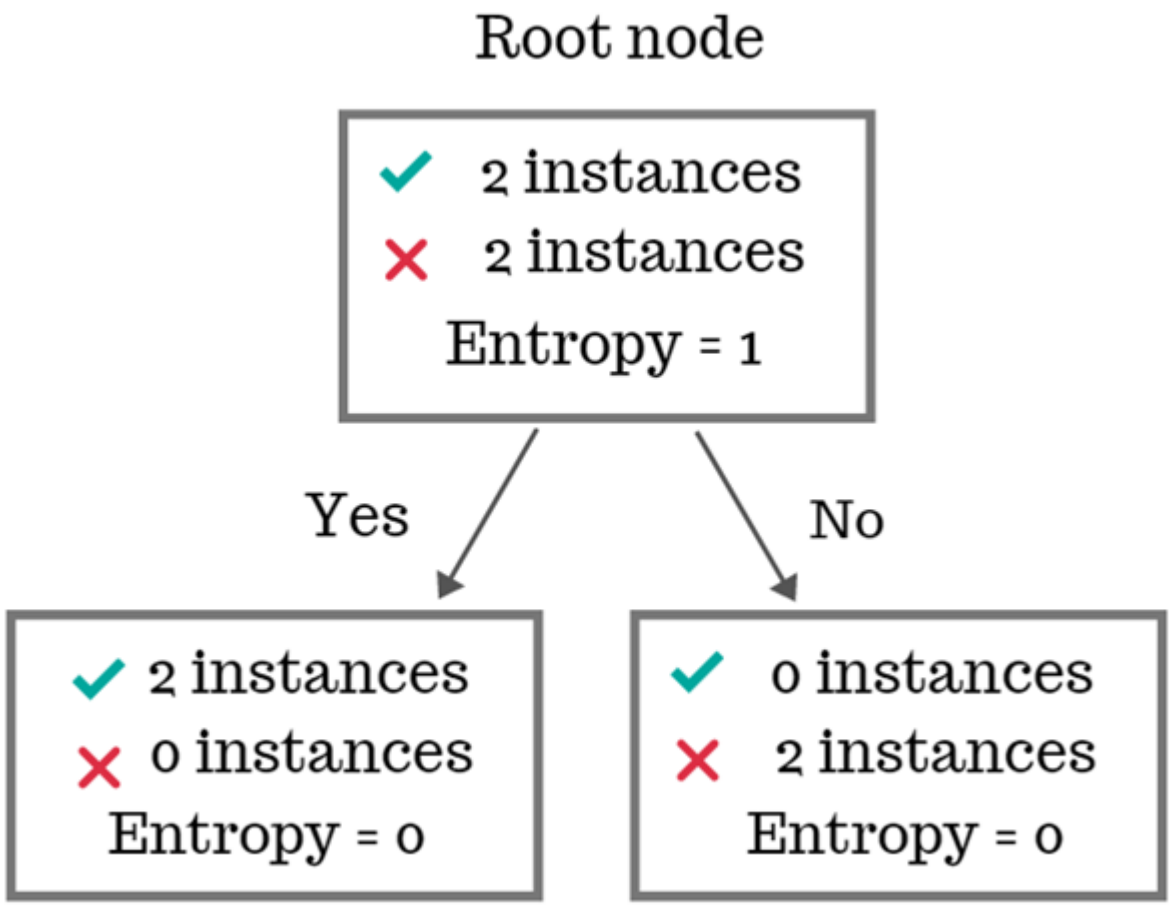
Age	Mileage	Road Tested	Buy
Recent	Low	Yes	Buy ✓
Recent	High	Yes	Buy ✓
Old	Low	No	Don't buy ✗
Recent	High	No	Don't buy ✗



Children Entropy = 2/4 (0) + 2/4 (0) = 0

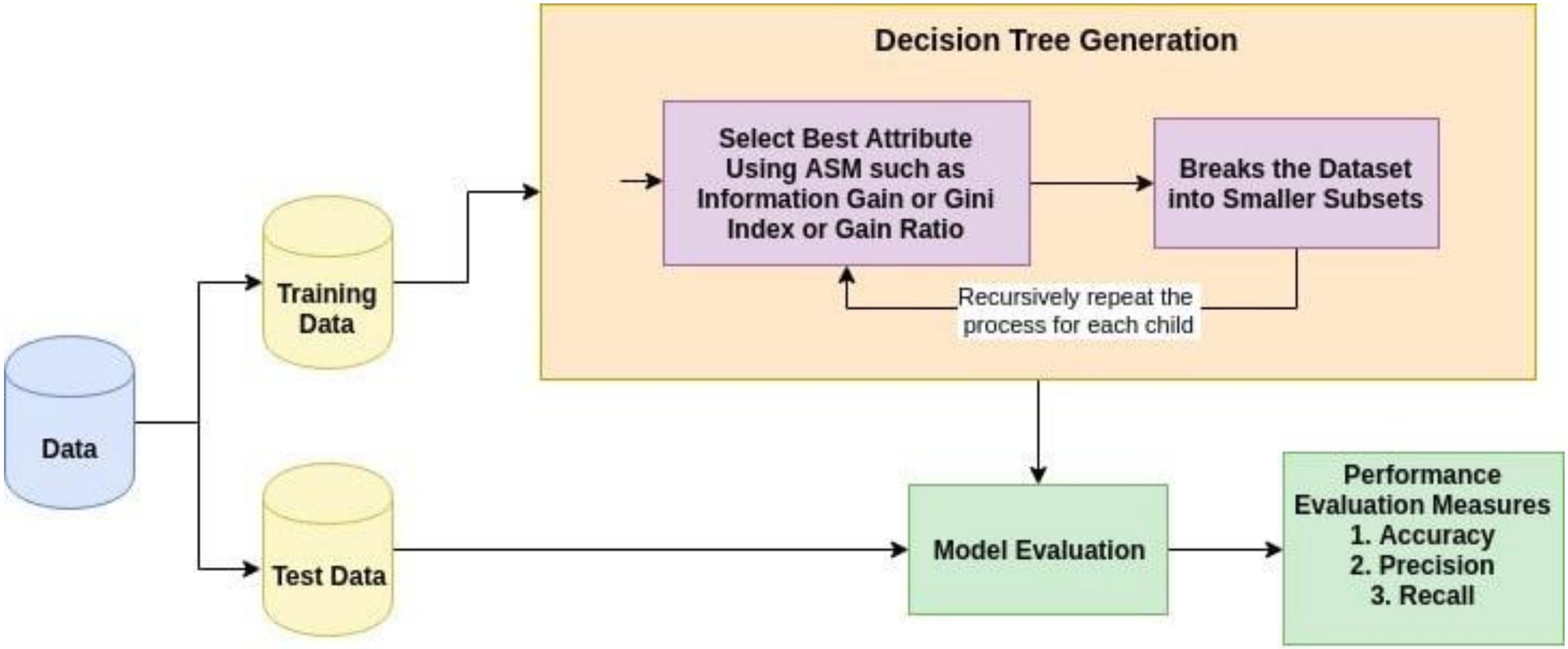
Information gain = 1 - 0 = 1

	Age	Mileage	Road Tested	Buy
	Recent	Low	Yes	Buy
	Recent	High	Yes	Buy
	Old	Low	No	Don't buy
	Recent	High	No	Don't buy
Info. Gain	0.3112	0	1	

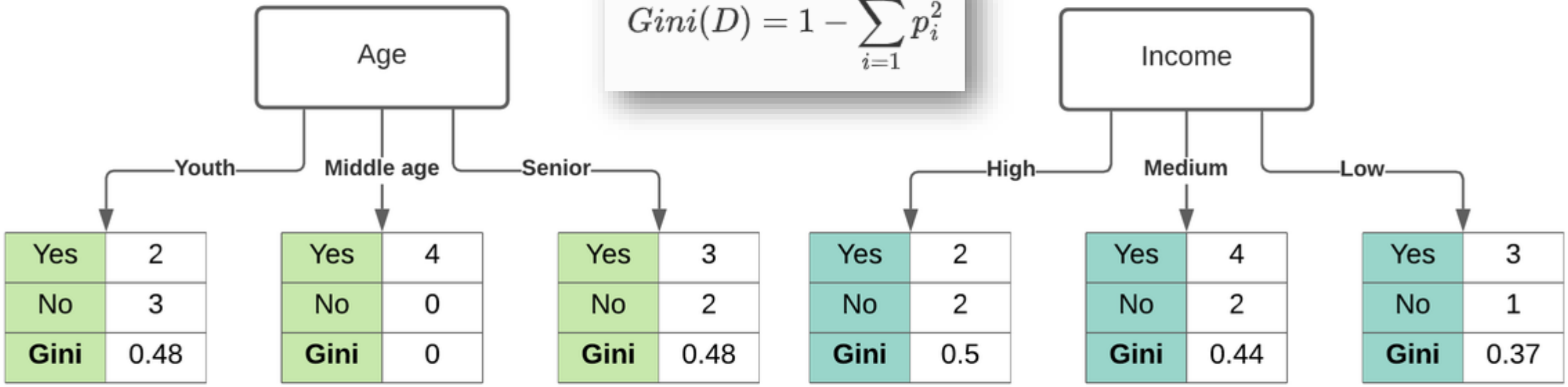


Since the entropy is zero at the leaf nodes, we'll stop splitting.
If this wasn't the case, then we would continue to find the information gain for the preceding parent and children nodes, until any one of the stopping criteria is met.

Decision Tree



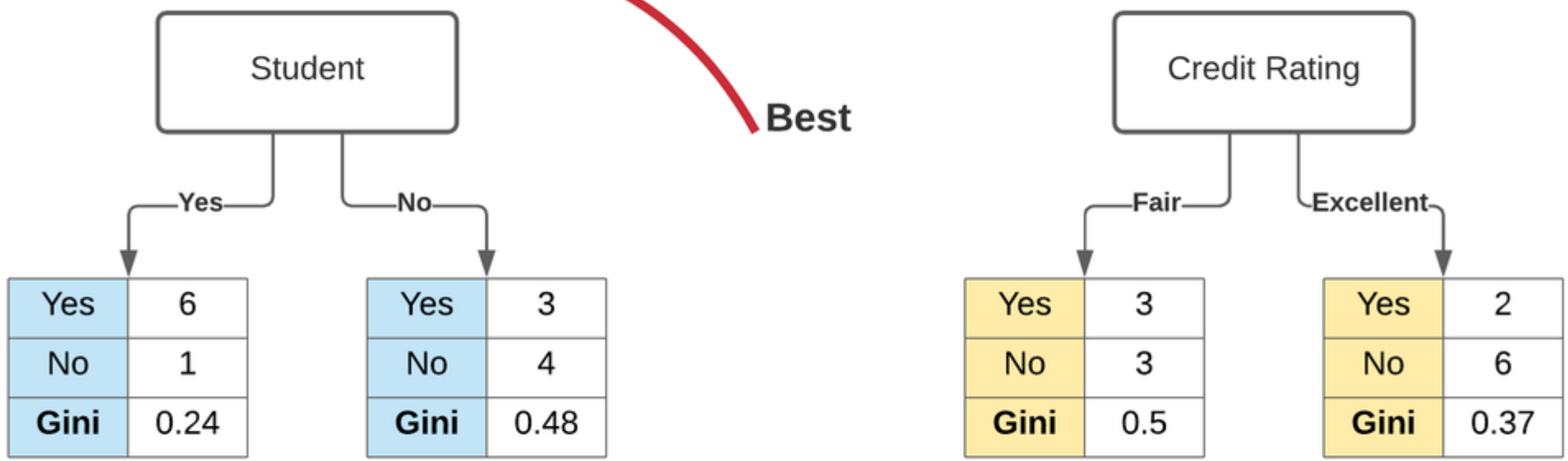
$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$



Gini Impurity for Age is 0.343

Gini Impurity for Income is 0.440

Best



Gini Impurity for Student is 0.367

Gini Impurity for Credit Rating is 0.429

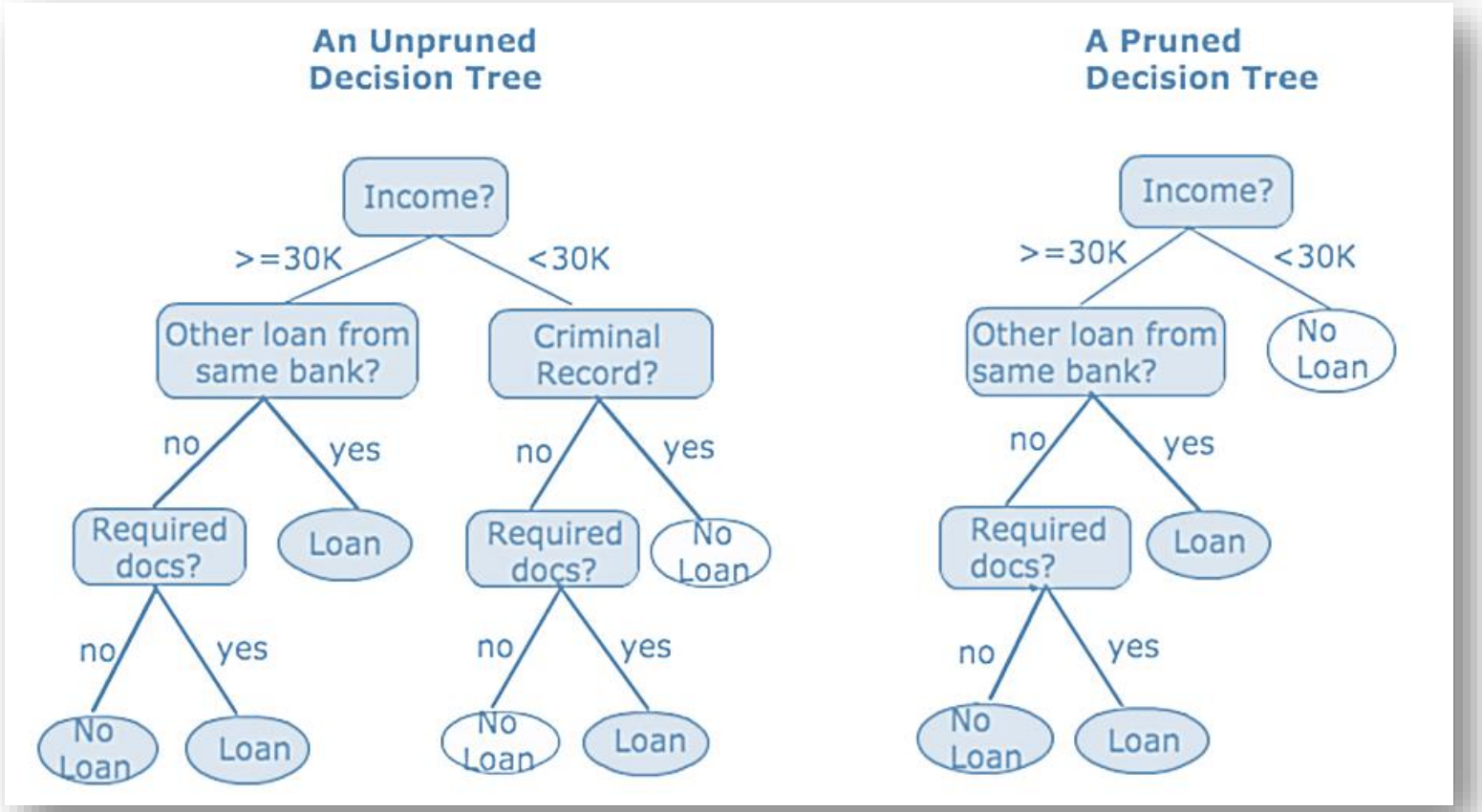
Advantages of Decision Tree

- Normalization or Scaling of data is not required
- Handles data with missing value
- Easy to explain and visualize (It results in a set of rules).
- Automatic feature selection i.e. irrelevant features won’t affect decision trees.
- The Number of hyper-parameters to be tuned is almost null.
- Work well with tabular data (structured data)

Disadvantages of Decision Tree

- Sensitive to data i.e. If data changes slightly, outcome also change to a very large extend.
- Requires more time in training
- There is a high probability of overfitting in Decision Tree.
- Generally, it gives low prediction accuracy for a dataset as compared to other machine learning algorithms.
- Calculations can become complex when there are many class labels.

Pruning decision trees



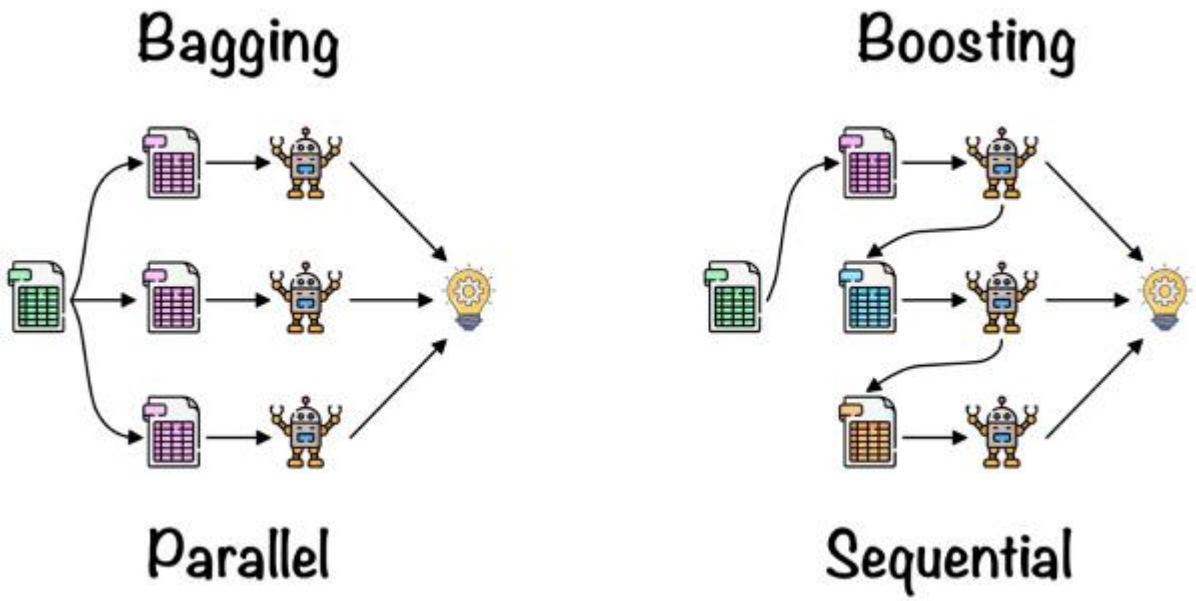
The process of IG-based pruning requires us to identify "twigs", nodes whose children are all leaves. "Pruning" a twig removes all of the leaves which are the children of the twig, and makes the twig a leaf.

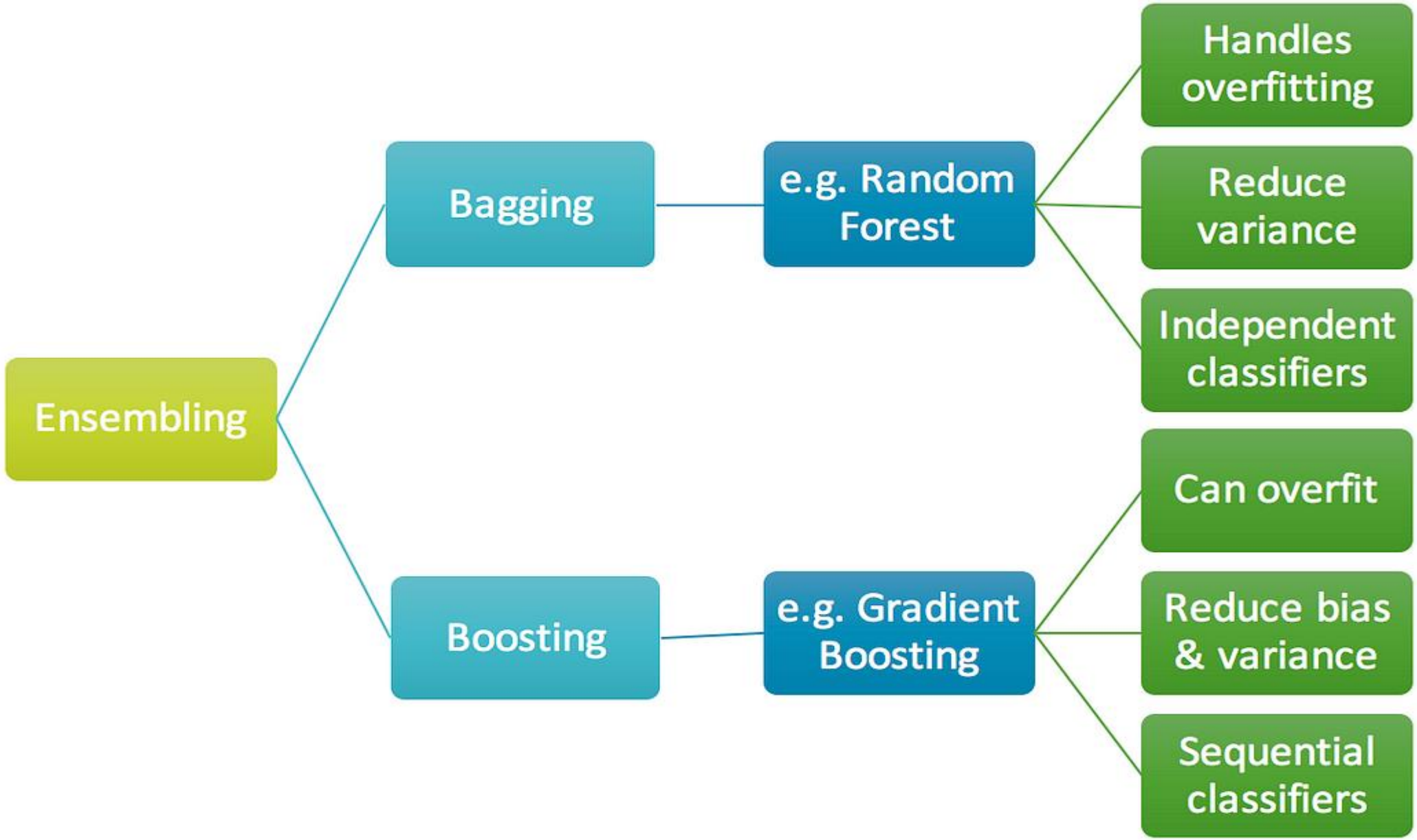
DecisionTreeClassifier in SkLearn

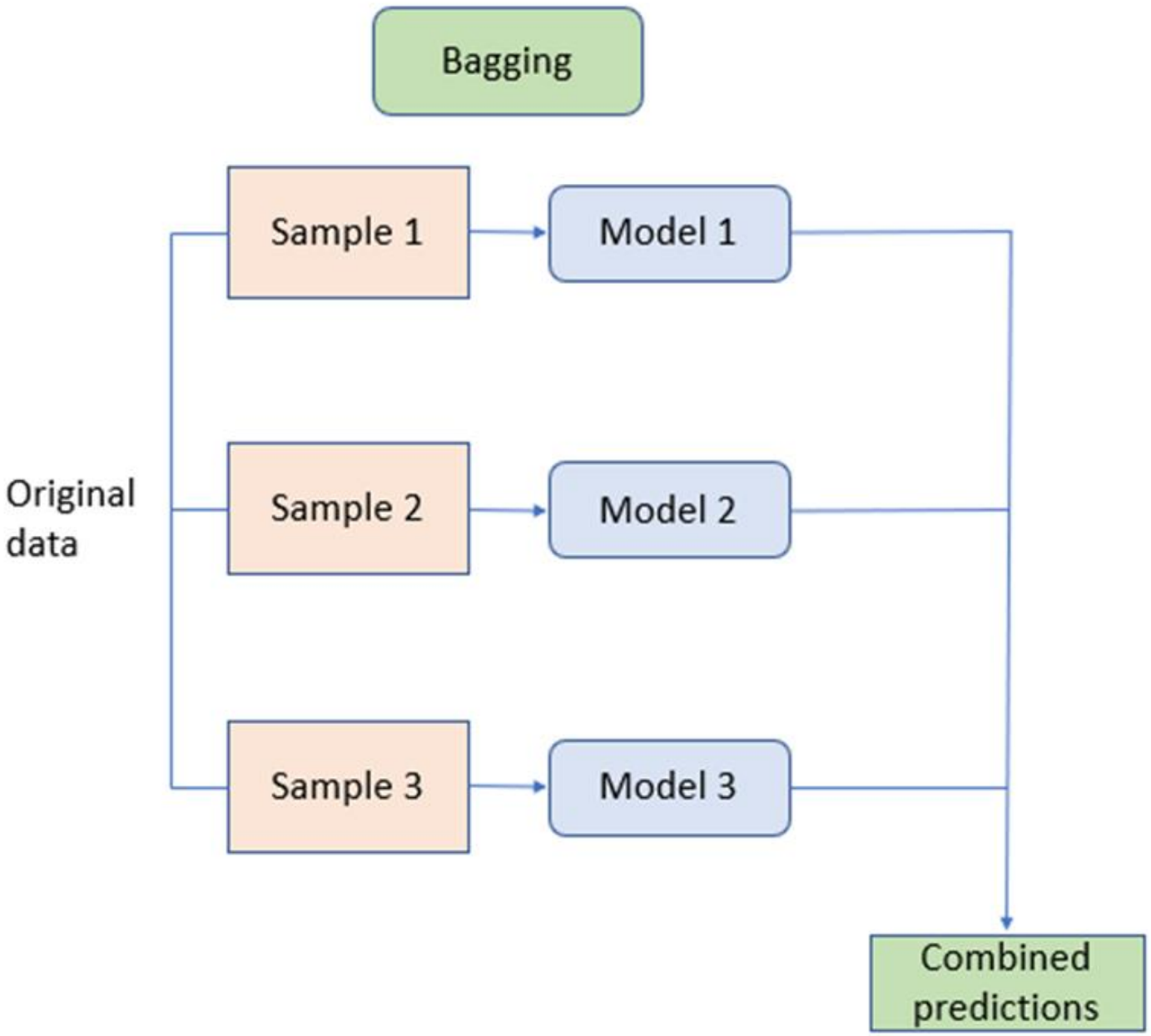
- The hyperparameters of the DecisionTreeClassifier in SkLearn include **max_depth**, **min_samples_leaf**, **min_samples_split** which can be tuned to early stop the growth of the tree and prevent the model from overfitting.
- The best way is to use the sklearn implementation of the GridSearchCV technique to find the best set of hyperparameters for a Decision Tree model.

Ensemble learning

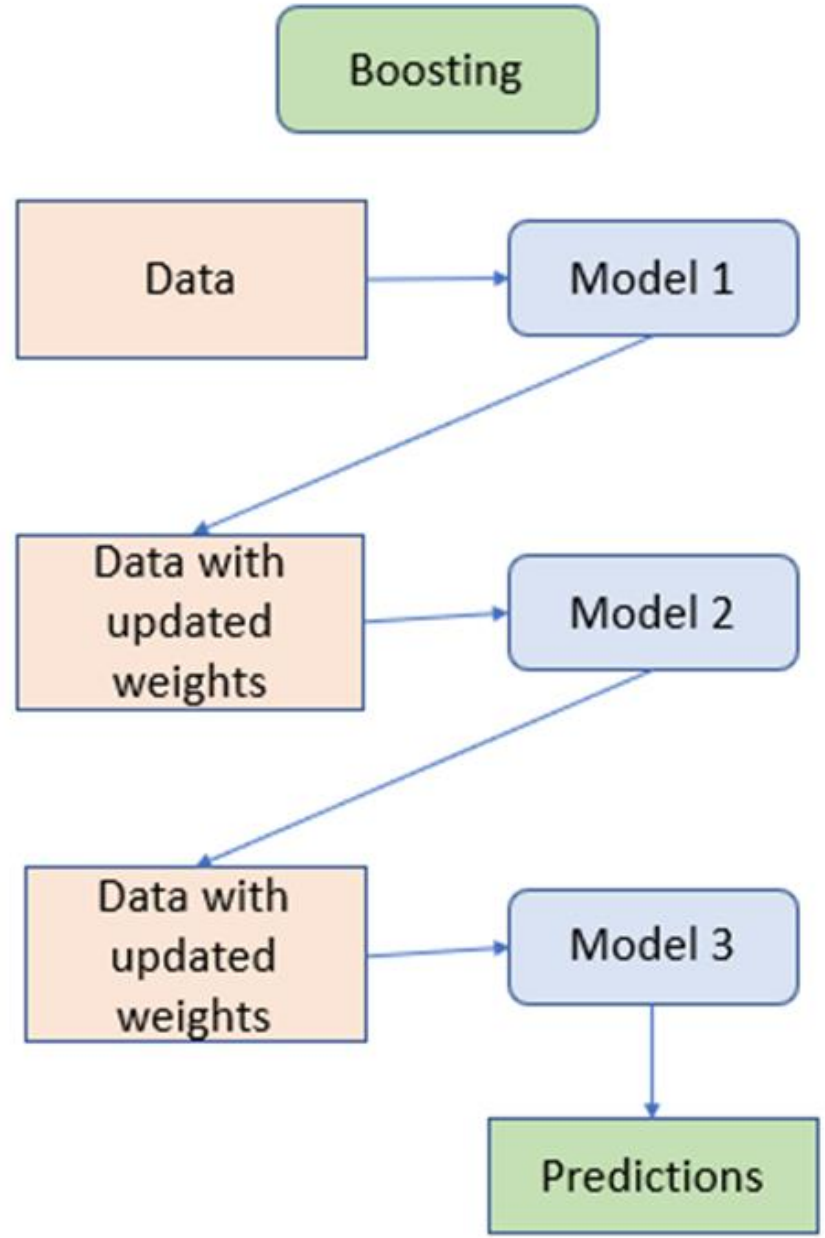
- Ensemble learning is a machine learning paradigm where multiple models (learners) are trained to solve the same problem.
- By using multiple learners, generalization ability of an ensemble can be much better than single learner.




Ensembling
Bagging
e.g. Random Forest
Handles overfitting
Reduce variance
Independent classifiers
Boosting
e.g. Gradient Boosting
Can overfit
Reduce bias & variance
Sequential classifiers



★ Bagging is used mainly in reducing variance and it is a parallel process

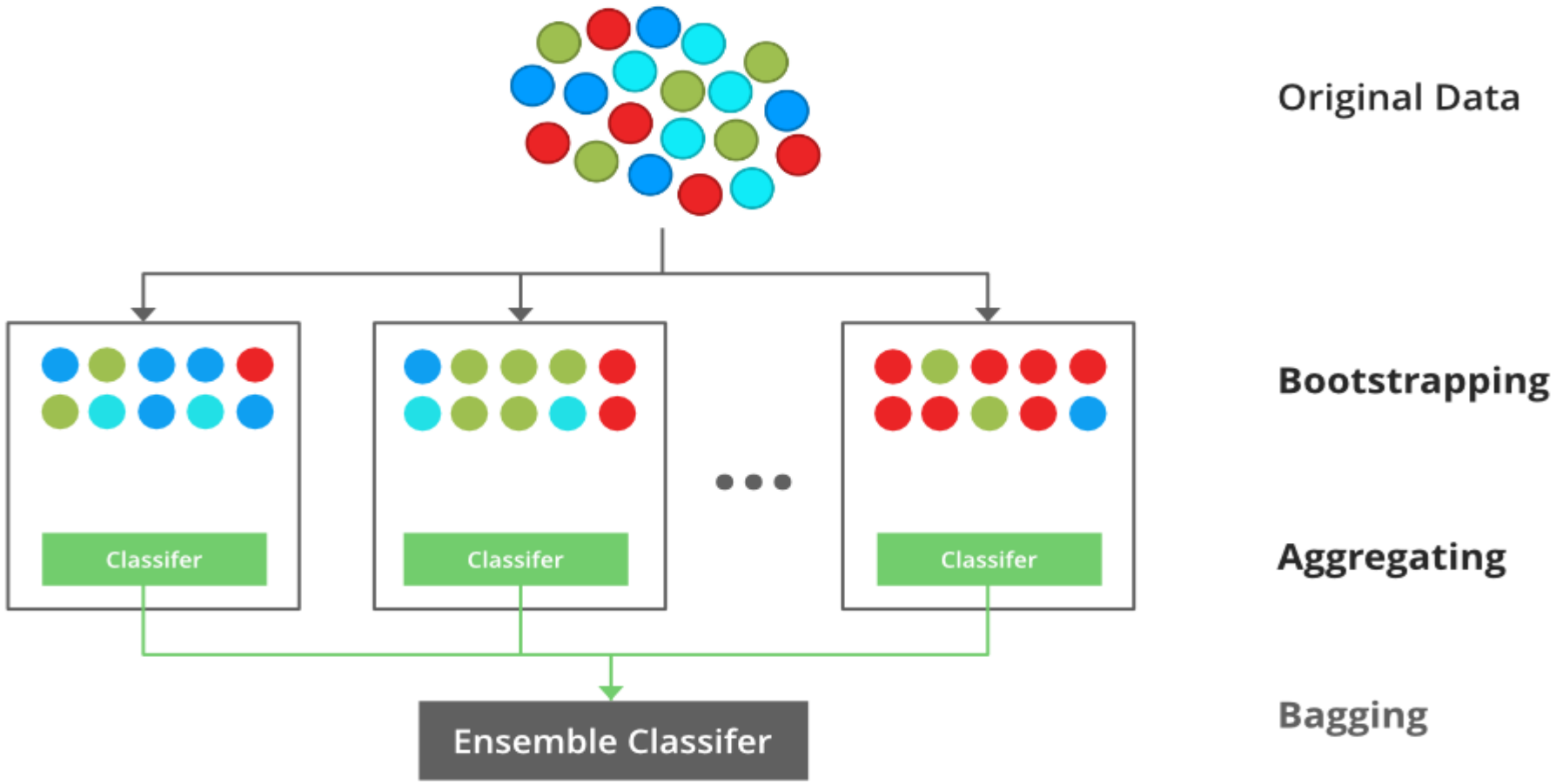


★ Boosting is used mainly in reducing bias and it is a sequential process

Ensemble Learning

- **Bagging** and **Boosting** are two types of **Ensemble Learning**. These two decrease the variance of a single estimate as they combine several estimates from different models. So the result may be a model with higher stability.
- **Bagging**: It is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.
- **Boosting**: It is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

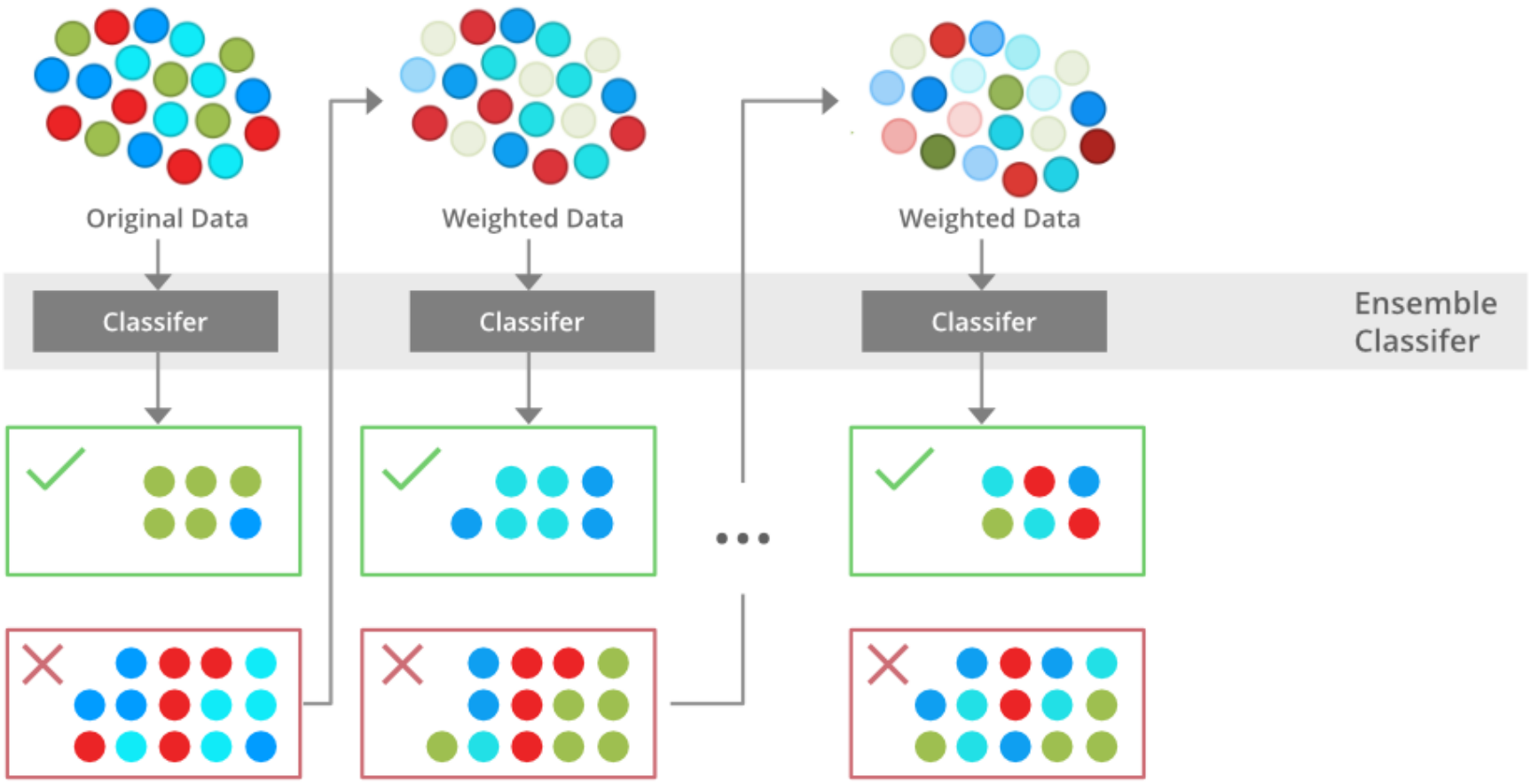
Bagging in Machine Learning



Bagging in Machine Learning

- **B**ootstrap **A**ggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression.
- It decreases the variance and helps to avoid overfitting. It is usually applied to decision tree methods.
- Bagging is a special case of the model averaging approach.

Boosting in Machine Learning

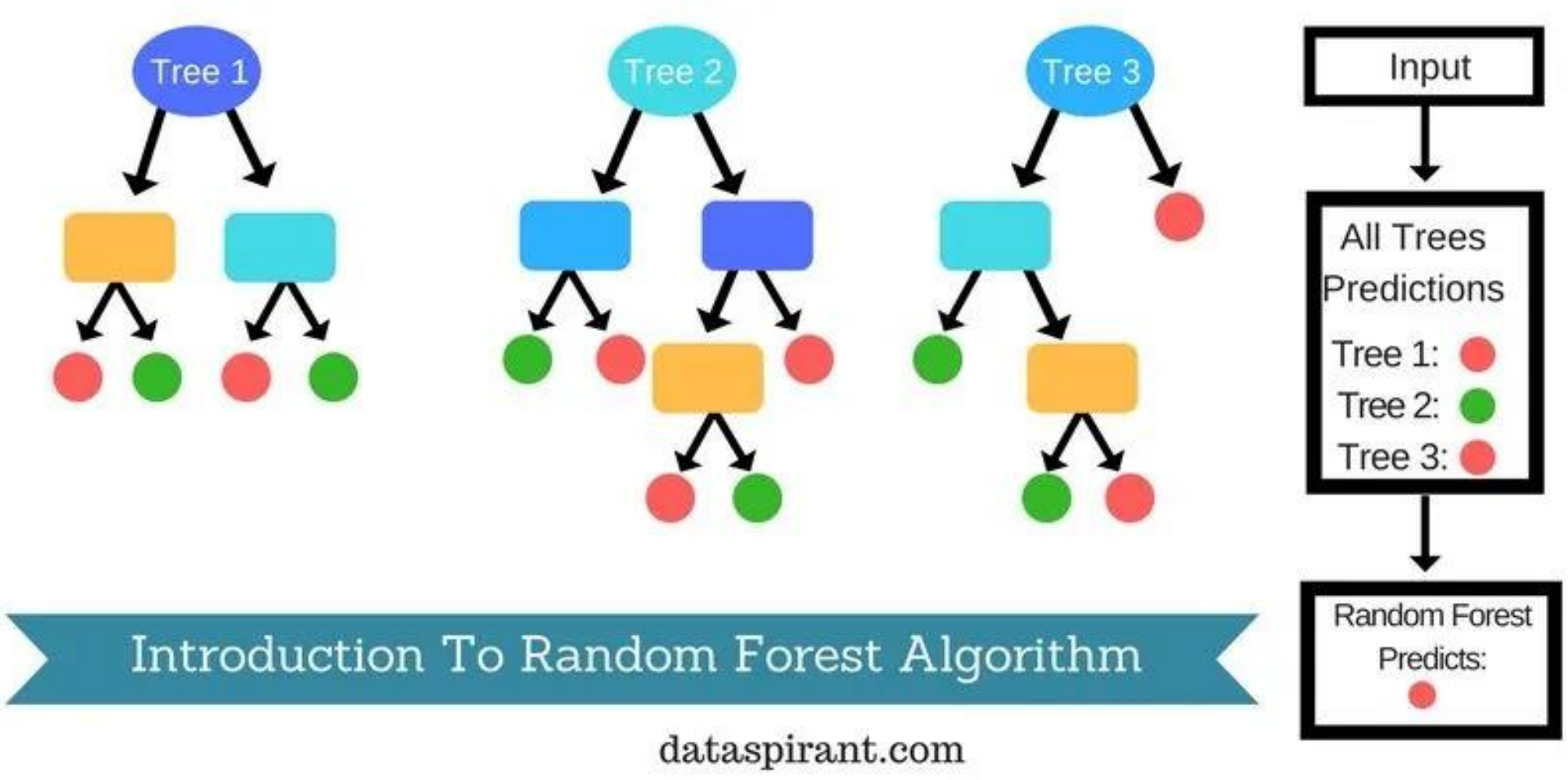


Boosting

- Boosting is the **ensemble learning method** where we build multiple weak learners (same algorithms) in a SEQUENTIAL manner.
- All these **weak learners** take the previous models' feedback to improve their power in accurately predicting the missclassified classes.
- Boosting algorithms are one of the best-performing algorithms among all the other Machine Learning algorithms with the best performance and higher accuracies.
- All the boosting algorithms work on the basis of learning from the errors of the previous model trained and tried avoiding the same mistakes made by the previously trained weak learning algorithm.

Random Forest

Random Forest



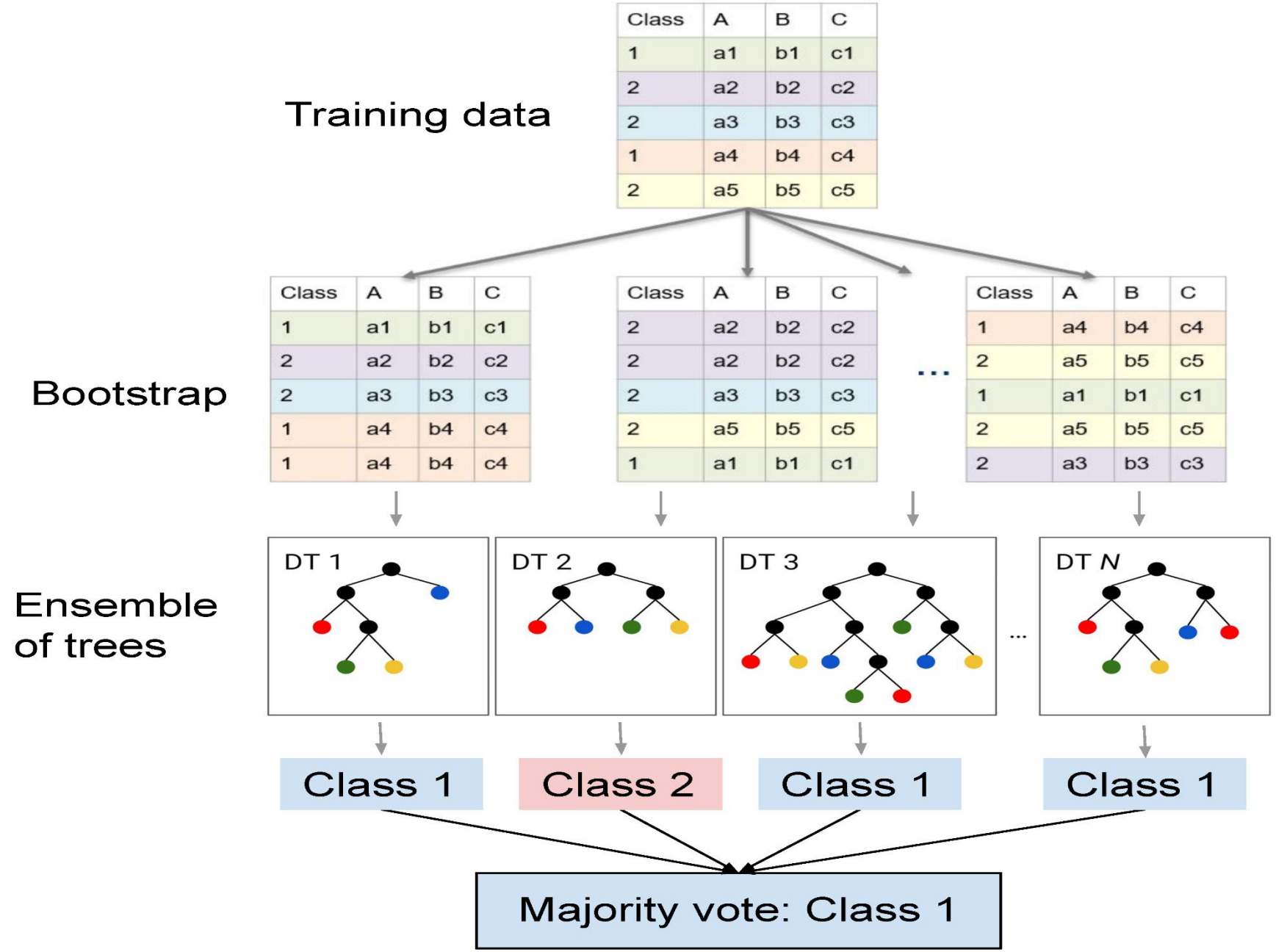
Random Forest

- The random forest algorithm is a supervised classification algorithm.
- As the name suggests, this algorithm creates the forest with a number of trees.
- In general, the **more trees in the forest** the more robust the forest looks like.
- In the same way in the random forest classifier, the **higher the number** of trees in the forest gives **the high the accuracy** results.
- In comparison, the random forest algorithm randomly selects observations and features to build several decision trees and then averages the results.

Why Random forest algorithm

- The same **random forest algorithm** or the random forest classifier can use for both classification and the regression task.
- Random forest classifier will **handle the missing** values.
- When we have more trees in the forest, a random forest classifier won't **overfit** the model.
- The random forest algorithm can be used for feature engineering.
  - This means identifying the most important features out of the available features from the training dataset.

Bagging


Training data
Class A B C
1 a1 b1 c1
2 a2 b2 c2
2 a3 b3 c3
1 a4 b4 c4
2 a5 b5 c5
Bootstrap
Class A B C
1 a1 b1 c1
2 a2 b2 c2
2 a3 b3 c3
1 a4 b4 c4
1 a4 b4 c4
Class A B C
2 a2 b2 c2
2 a2 b2 c2
2 a3 b3 c3
2 a5 b5 c5
1 a1 b1 c1
...
Class A B C
1 a4 b4 c4
2 a5 b5 c5
1 a1 b1 c1
2 a5 b5 c5
2 a3 b3 c3
Ensemble of trees
DT 1
DT 2
DT 3
...
DT N
Class 1
Class 2
Class 1
Class 1
Majority vote: Class 1

Dataset

Randomly Selected Data 1

Randomly Selected Data 2

Randomly Selected Data n

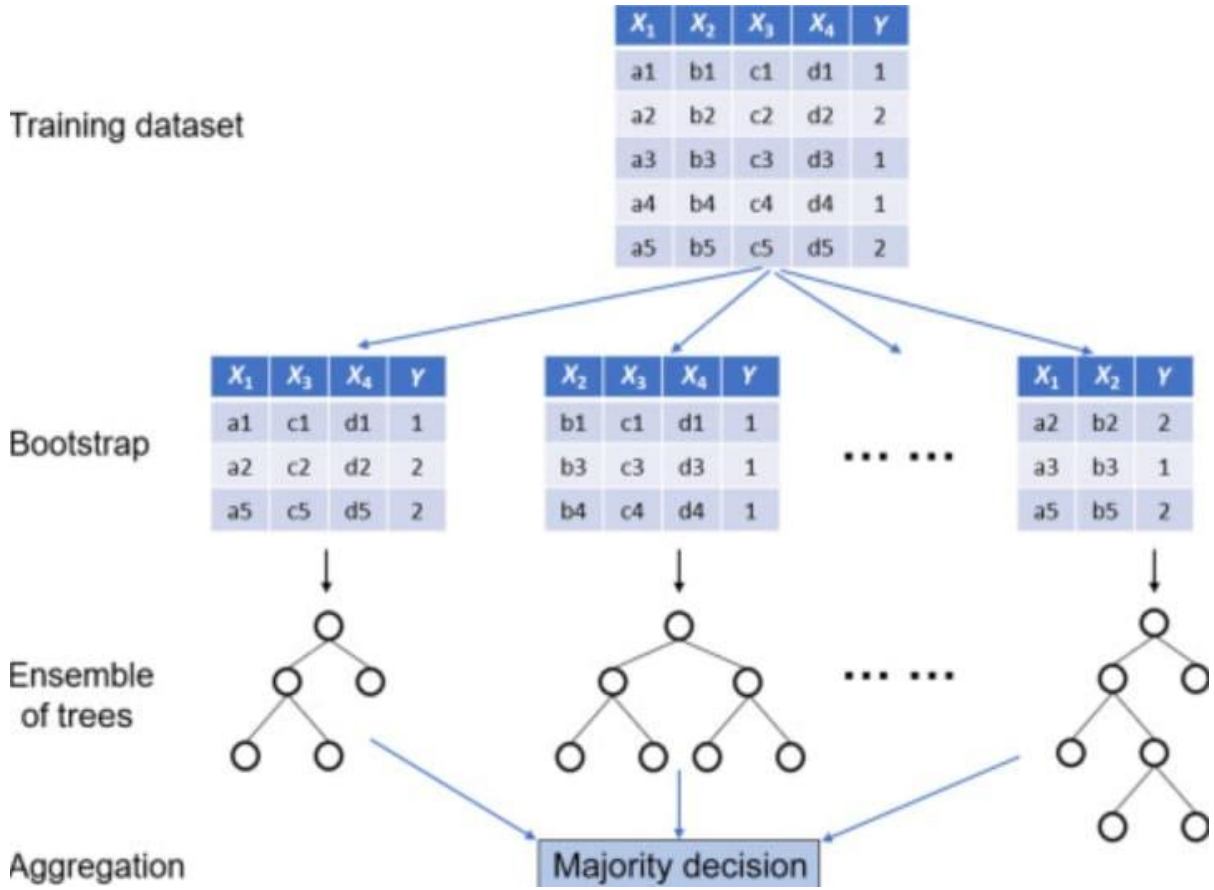
Decision tree 1

Decision tree 2

Decision tree n

Voting

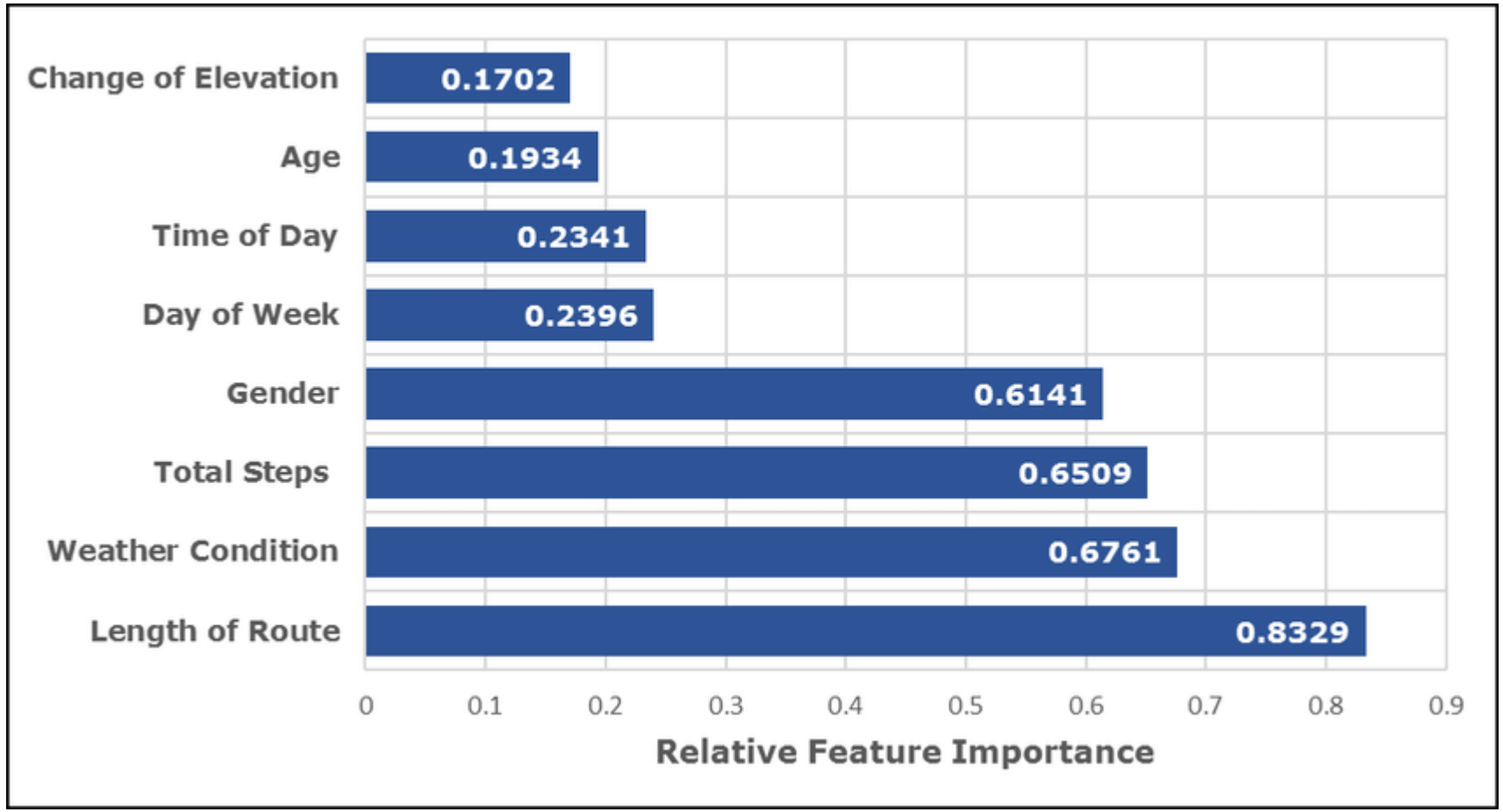
Prediction


Training dataset
X1 X2 X3 X4 Y
a1 b1 c1 d1 1
a2 b2 c2 d2 2
a3 b3 c3 d3 1
a4 b4 c4 d4 1
a5 b5 c5 d5 2
Bootstrap
X1 X3 X4 Y
a1 c1 d1 1
a2 c2 d2 2
a5 c5 d5 2
X2 X3 X4 Y
b1 c1 d1 1
b3 c3 d3 1
b4 c4 d4 1
X1 X2 Y
a2 b2 2
a3 b3 1
a5 b5 2
Ensemble of trees
Aggregation
Majority decision

Random Forest

1.Takes the **test features** and use the rules of each randomly created decision tree to predict the oucome and stores the predicted outcome (target)
2.Calculate the **votes** for each predicted target.
3.Consider the **high voted** predicted target as the **final prediction** from the random forest algorithm.

Feature Importance



RANDOM FOREST DISADVANTAGES

- Increased accuracy requires more trees
- More trees slow down model

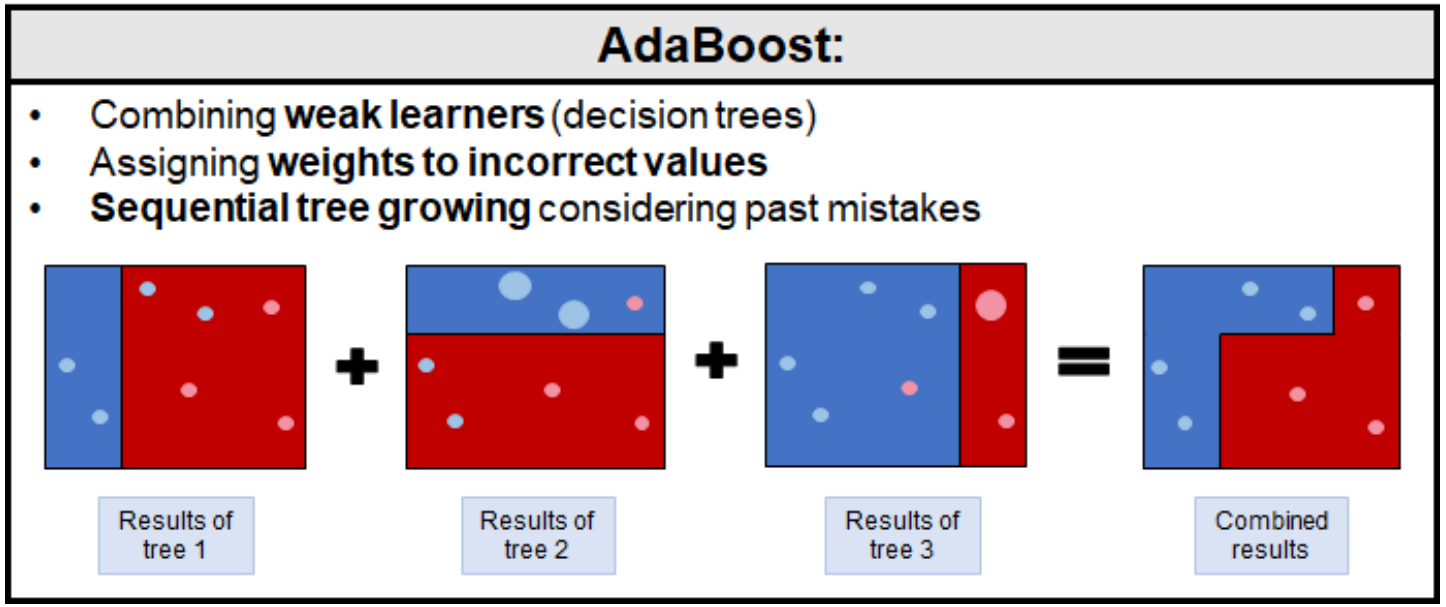
RANDOM FOREST IN Sklearn

- Firstly, there is the **n_estimators** hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions
- Another important hyperparameter is **max_features,** which is the maximum number of features random forest considers to split a node.
- The last important hyperparameter is **min_sample_leaf.** This determines the minimum number of leafs required to split an internal node.

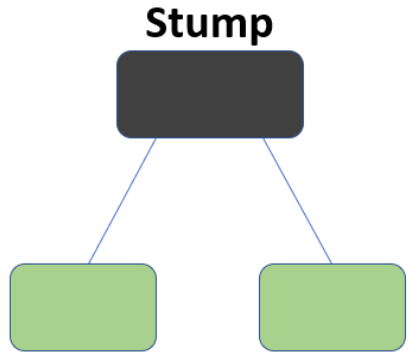
AdaBoost, GBoost and XGBoost

AdaBoost

- AdaBoost is a boosting algorithm, which also works on the principle of the stagewise addition method where multiple weak learners are used for getting strong learners.



What Makes AdaBoost Different



- AdaBoost is similar to Random Forests in the sense that the predictions are taken from many decision trees. However, there are three main differences that make AdaBoost unique:
  - First, AdaBoost creates a forest of stumps rather than trees. A stump is a tree that is made of only one node and two leaves.
  - Second, the stumps that are created are not equally weighted in the final decision. Stumps that create more error will have less say in the final decision.
  - Lastly, the order in which the stumps are made is important, because each stump aims to reduce the errors that the previous stump(s) made.

How AdaBoost Works

- **Step 1:** A weak classifier (e.g. a decision stump) is made on top of the training data based on the weighted samples. Here, the weights of each sample indicate how important it is to be correctly classified. Initially, for the first stump, we give all the samples equal weights.
- **Step 2:** We create a decision stump for each variable and see how well each stump classifies samples to their target classes.
- **Step 3:** More weight is assigned to the incorrectly classified samples so that they're classified correctly in the next decision stump. Weight is also assigned to each classifier based on the accuracy of the classifier, which means high accuracy = high weight!
- **Step 4:** Reiterate from Step 2 until all the data points have been correctly classified, the error function does not change, or the maximum iteration level has been reached.

How AdaBoost Works

- **Step 1** – First of all, these data points will be assigned some weights. Initially, all the weights will be equal.

Row No.	Gender	Age	Income	Illness	Sample Weights
1	Male	41	40000	Yes	1/5
2	Male	54	30000	No	1/5
3	Female	42	25000	No	1/5
4	Female	40	60000	Yes	1/5
5	Male	46	50000	Yes	1/5

The formula to calculate the sample weights is:

$$w(x_i, y_i) = \frac{1}{N}, \; i = 1, 2, \ldots . n$$

How AdaBoost Works

- **Step 2** – We start by seeing how well "*Gender*" classifies the samples and will see how the variables (Age, Income) classify the samples.
- We'll create a decision stump for each of the features and then calculate the ***Gini Index*** of each tree. The tree with the lowest Gini Index will be our first stump.
- Here in our dataset, let's say ***Gender*** has the lowest gini index, so it will be our first stump.

How AdaBoost Works

- **Step 3** – We'll now calculate the **"Amount of Say"** or **"Importance"** or **"Influence"** for this classifier in classifying the data points using this formula:

$$\frac{1}{2}\log\frac{1-Total\ Error}{Total\ Error}$$

- The total error is nothing but the summation of all the sample weights of misclassified data points.

Row No.	Gender	Age	Income	Illness	Sample Weights
1	Male	41	40000	Yes	1/5
2	Male	54	30000	No	1/5
3	Female	42	25000	No	1/5
4	Female	40	60000	Yes	1/5
5	Male	46	50000	Yes	1/5

How AdaBoost Works

- Here in our dataset, let's assume there is 1 wrong output, so our total error will be 1/5, and the alpha (performance of the stump) will be:

$$Performance\ of\ the\ stump\ =\ \frac{1}{2}\log_e\left(\frac{1-Total\ Error}{Total\ Error}\right)$$

$$\alpha\ =\ \frac{1}{2}\log_e\left(\frac{1-\frac{1}{5}}{\frac{1}{5}}\right)$$

$$\alpha\ =\ \frac{1}{2}\log_e\left(\frac{0.8}{0.2}\right)$$

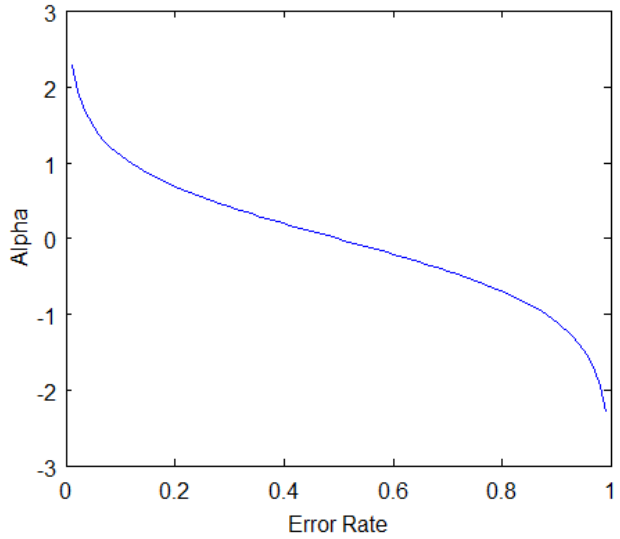
$$\alpha\ =\ \frac{1}{2}\log_e(4)\ =\ \frac{1}{2}*(1.38)$$

$$\alpha\ =\ 0.69$$

Note: Total error will always be between 0 and 1.
0 Indicates perfect stump, and 1 indicates horrible stump.

How AdaBoost Works

- From the graph, we can see that when there is no misclassification, then we have no error (Total Error = 0), so the “amount of say (alpha)” will be a large number.
- When the classifier predicts half right and half wrong, then the Total Error = 0.5, and the importance (amount of say) of the classifier will be 0.
- If all the samples have been incorrectly classified, then the error will be very high (approx. to 1), and hence our alpha value will be a negative integer.



How AdaBoost Works

- **Step 4** – We need to update the weights because if the same weights are applied to the next model, then the output received will be the same as what was received in the first model.
- The wrong predictions will be given more weight, whereas the correct predictions weights will be decreased.
- Now when we build our next model after updating the weights, more preference will be given to the points with higher weights.
- finally update the weights, and for this, we use the following formula:

$$New\ sample\ weight\ =\ old\ weight\ *\ e^{\pm Amount\ of\ say\ (\alpha)}$$

How AdaBoost Works

- The amount of say (alpha) will be ***negative*** when the sample is **correctly classified**.
- The amount of say (alpha) will be ***positive*** when the sample is **miss-classified.**

New weights for correctly classified samples are:

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(-0.69)$$

$$New\ sample\ weight\ =\ 0.2\ *\ 0.502\ =\ 0.1004$$

For wrongly classified samples, the updated weights will be:

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(0.69)$$

$$New\ sample\ weight\ =\ 0.2\ *\ 1.994\ =\ 0.3988$$

How AdaBoost Works

- Normalizing the sample weights, we get this dataset, and now the sum is equal to 1.

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004
2	Male	54	30000	No	1/5	0.1004
3	Female	42	25000	No	1/5	0.1004
4	Female	40	60000	Yes	1/5	0.3988
5	Male	46	50000	Yes	1/5	0.1004

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004/0.8004 =0.1254
2	Male	54	30000	No	1/5	0.1004/0.8004 =0.1254
3	Female	42	25000	No	1/5	0.1004/0.8004 =0.1254
4	Female	40	60000	Yes	1/5	0.3988/0.8004 =0.4982
5	Male	46	50000	Yes	1/5	0.1004/0.8004 =0.1254

How AdaBoost Works

- **Step 5** – Now, we need to make a new dataset to see if the errors decreased or not. For this, we will remove the "sample weights" and "new sample weights" columns and then, based on the "new sample weights," divide our data points into buckets.

Row No.	Gender	Age	Income	Illness	New Sample Weights	Buckets
1	Male	41	40000	Yes	0.1004/0.8004= 0.1254	0 to 0.1254
2	Male	54	30000	No	0.1004/0.8004= 0.1254	0.1254 to 0.2508
3	Female	42	25000	No	0.1004/0.8004= 0.1254	0.2508 to 0.3762
4	Female	40	60000	Yes	0.3988/0.8004= 0.4982	0.3762 to 0.8744
5	Male	46	50000	Yes	0.1004/0.8004= 0.1254	0.8744 to 0.9998

How AdaBoost Works

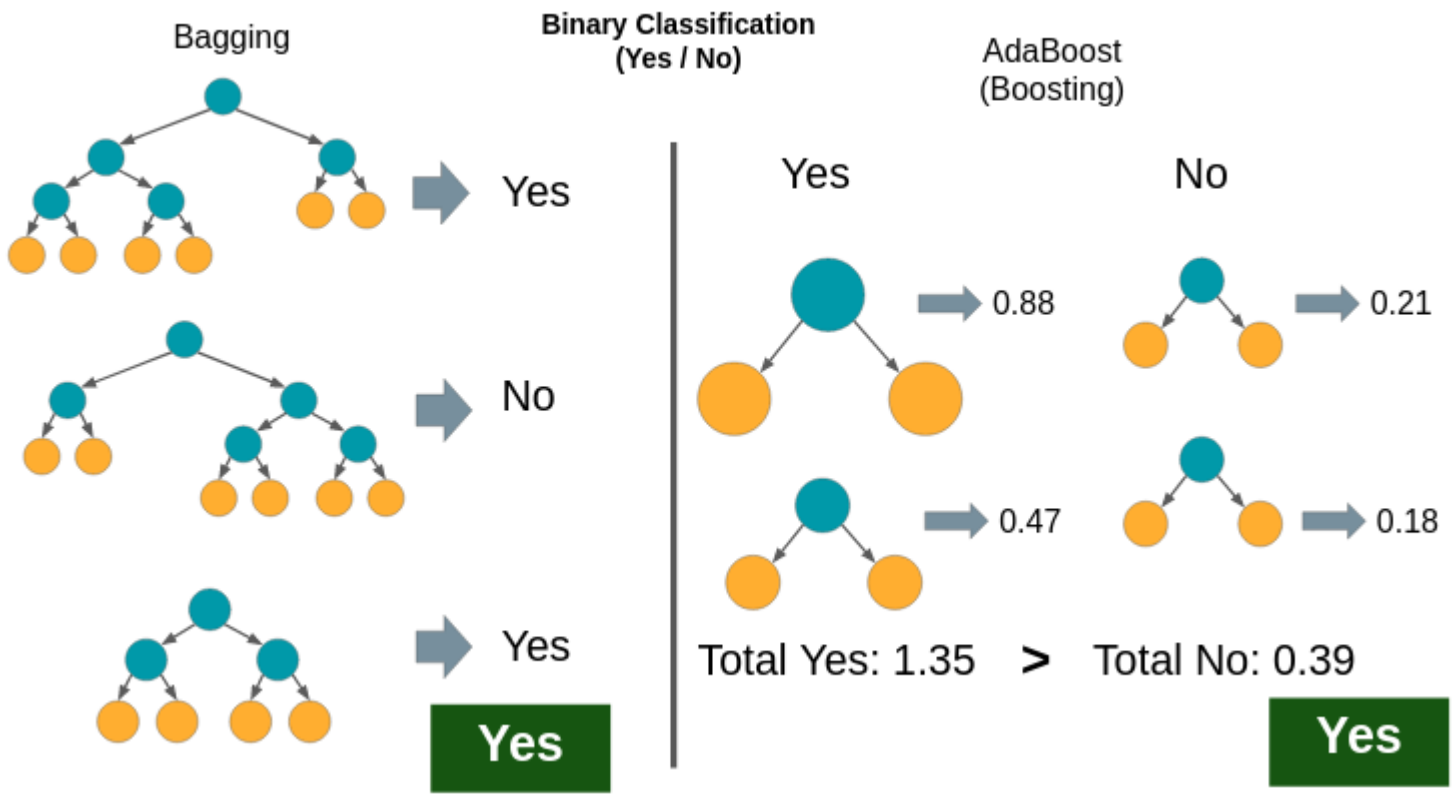
- **Step 6** – Now, what the algorithm does is selects random numbers from 0-1. Since incorrectly classified records have higher sample weights, the probability of selecting those records is very high.
- Suppose the 5 random numbers our algorithm take is 0.38,0.26,0.98,0.40,0.55.
- This comes out to be our new dataset, and we see the data point, which was wrongly classified, has been selected 3 times because it has a higher weight.

Row No.	Gender	Age	Income	Illness
1	Female	40	60000	Yes
2	Male	54	30000	No
3	Female	42	25000	No
4	Female	40	60000	Yes
5	Female	40	60000	Yes

How AdaBoost Works

- **Step 9** – Now this act as our new dataset, and we need to repeat all the above steps i.e.
  - Assign ***equal weights*** to all the data points.
  - Find the stump that does the ***best job classifying*** the new collection of samples by finding their Gini Index and selecting the one with the lowest Gini index.
  - Calculate the ***"Amount of Say"*** and ***"Total error"*** to update the previous sample weights.
  - Normalize the new sample weights.
  - Iterate through these steps until and unless a low training error is achieved.
- Suppose, with respect to our dataset, we have constructed 3 decision trees (DT1, DT2, DT3) in a ***sequential manner.*** If we send our **test data** now, it will pass through all the decision trees, and finally, we will see which class has the majority.

How AdaBoost Works



An Example of How AdaBoost Works

- Suppose we have the sample data below, with three features (x1, x2, x3) and an output (Y). *Note that T = True and F = False.*

x1	x2	x3	Y
T	T	F	T
T	T	F	F
T	F	F	T
T	T	T	F
F	T	F	F
T	F	F	T

An Example of How AdaBoost Works

- **Step 1: Assign a sample weight for each sample**

$$sample\ weight = \frac{1}{\#\ of\ samples}$$

x1	x2	x3	Y	Sample Weight
T	T	F	T	1/6
T	T	F	F	1/6
T	F	F	T	1/6
T	T	T	F	1/6
F	T	F	F	1/6
T	F	F	T	1/6

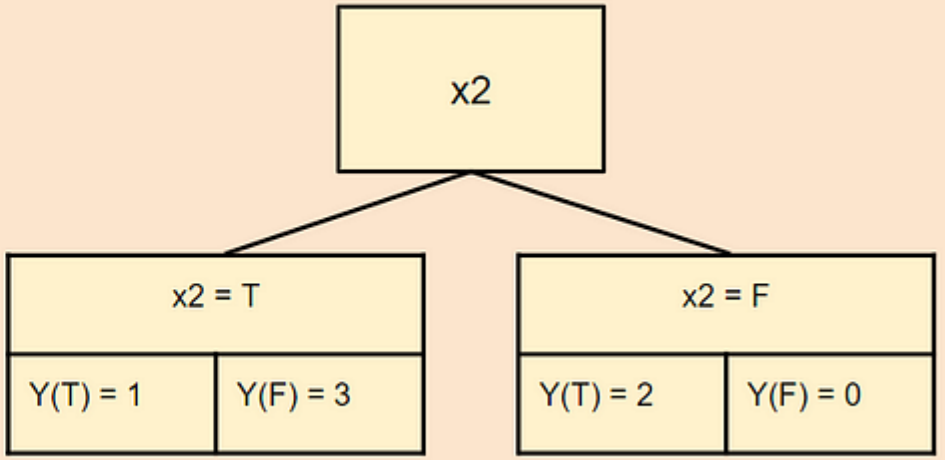
An Example of How AdaBoost Works

- **Step 2: Calculate the Gini Impurity for each variable**

This is done to determine which variable to use to create the first stump.

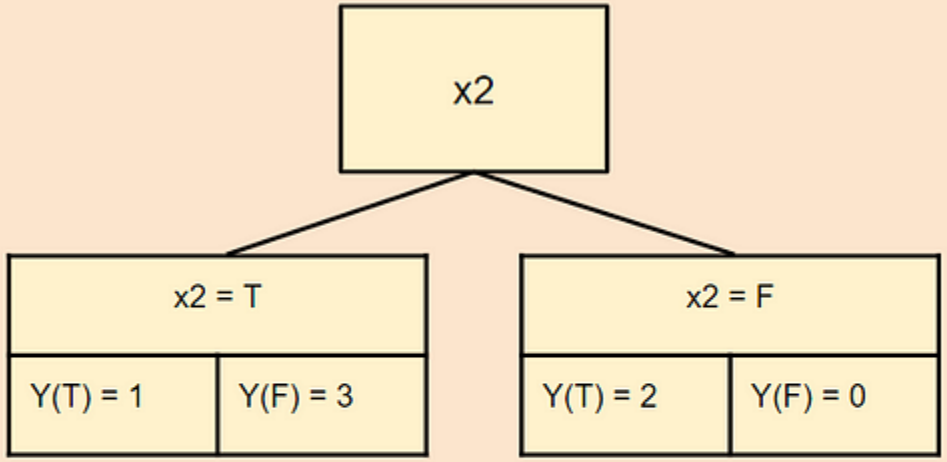
$$Gini\ Impurity = 1 - (the\ probability\ of\ True)^2 - (the\ probability\ of\ False)^2$$

x2	Y = T	Y = F
X2 = T	1	3
X2 = F	2	0



An Example of How AdaBoost Works

x2 has the lowest Gini Impurity, so x2 will be used to create the first stump.



$$Gini\ Impurity = 1 - (the\ probability\ of\ True)^2 - (the\ probability\ of\ False)^2$$

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

$$Gini\ Impurity = 0.375$$

$$Gini\ Impurity = 1 - (the\ probability\ of\ True)^2 - (the\ probability\ of\ False)^2$$

$$Gini\ Impurity = 1 - \left(\frac{2}{2+0}\right)^2 - \left(\frac{0}{2+0}\right)^2$$

$$Gini\ Impurity = 0$$

$$Total\ Impurity = 0.375\left(\frac{4}{4+2}\right) + 0\left(\frac{2}{4+2}\right)$$

$$Total\ Impurity = 0.25$$

the Gini Impurity for x2 = 0.25.

An Example of How AdaBoost Works

- **Step 3: Calculate the Amount of Say for the stump that was created**

Total Error is equal to the sum of the weights of the incorrectly classified samples.

Since one of the samples was incorrectly classified for x2, the total error is equal to 1/6.

$$Amount\ of\ say = \frac{1}{2}\log\left(\frac{1 - total\ error}{total\ error}\right)$$

$$Amount\ of\ say = \frac{1}{2}\log\left(\frac{1 - \frac{1}{6}}{\frac{1}{6}}\right) = 0.35$$

An Example of How AdaBoost Works

- **Step 4: Calculate the new sample weights for the next stump**

we're going to increase the sample weights of samples that were incorrectly classified and decrease the sample weights of samples that were correctly classified using the following equations:

$$New\ Sample\ Weight\ For\ \mathbf{Incorrect}\ Samples = Sample\ weight * e^{amount\ of\ say}$$

$$New\ Sample\ Weight\ For\ \mathbf{Correct}\ Samples = Sample\ weight * e^{-amount\ of\ say}$$

An Example of How AdaBoost Works

Need to divide each weight by 0.84

x1	x2	x3	Y	New Sample Weight	Normalized Sample Weights
T	T	F	T	0.24	0.29
T	T	F	F	0.12	0.14
T	F	F	T	0.12	0.14
T	T	T	F	0.12	0.14
F	T	F	F	0.12	0.14
T	F	F	T	0.12	0.14

Total: 0.84

An Example of How AdaBoost Works

- **Step 5: Create a bootstrapped dataset with the odds of each sample being chosen based on their new sample weights.**

x1	x2	x3	Y	Normalized Sample Weights
T	T	F	T	0.29
T	T	F	F	0.14
T	T	F	T	0.29
T	T	T	F	0.14
T	T	F	T	0.29
T	F	F	T	0.14

An Example of How AdaBoost Works

Once the new bootstrapped dataset is created, the samples are given equal weights again, and the process is repeated.

x1	x2	x3	Y	Sample Weights
T	T	F	T	1/6
T	T	F	F	1/6
T	T	F	T	1/6
T	T	T	F	1/6
T	T	F	T	1/6
T	F	F	T	1/6

An Example of How AdaBoost Works

- **Step 6: Repeat the process n number of times**
- Lastly, this process is repeated until n number of stumps are created, each with its own amount of say. Once this is done, the model is complete and new points can be classified.
- New points are classified by running them through all of the stumps and seeing how they're classified. Then, the amount of say is summed for each class, and the class with the higher amount of say is the classification of the new point.

Advantages of AdaBoost

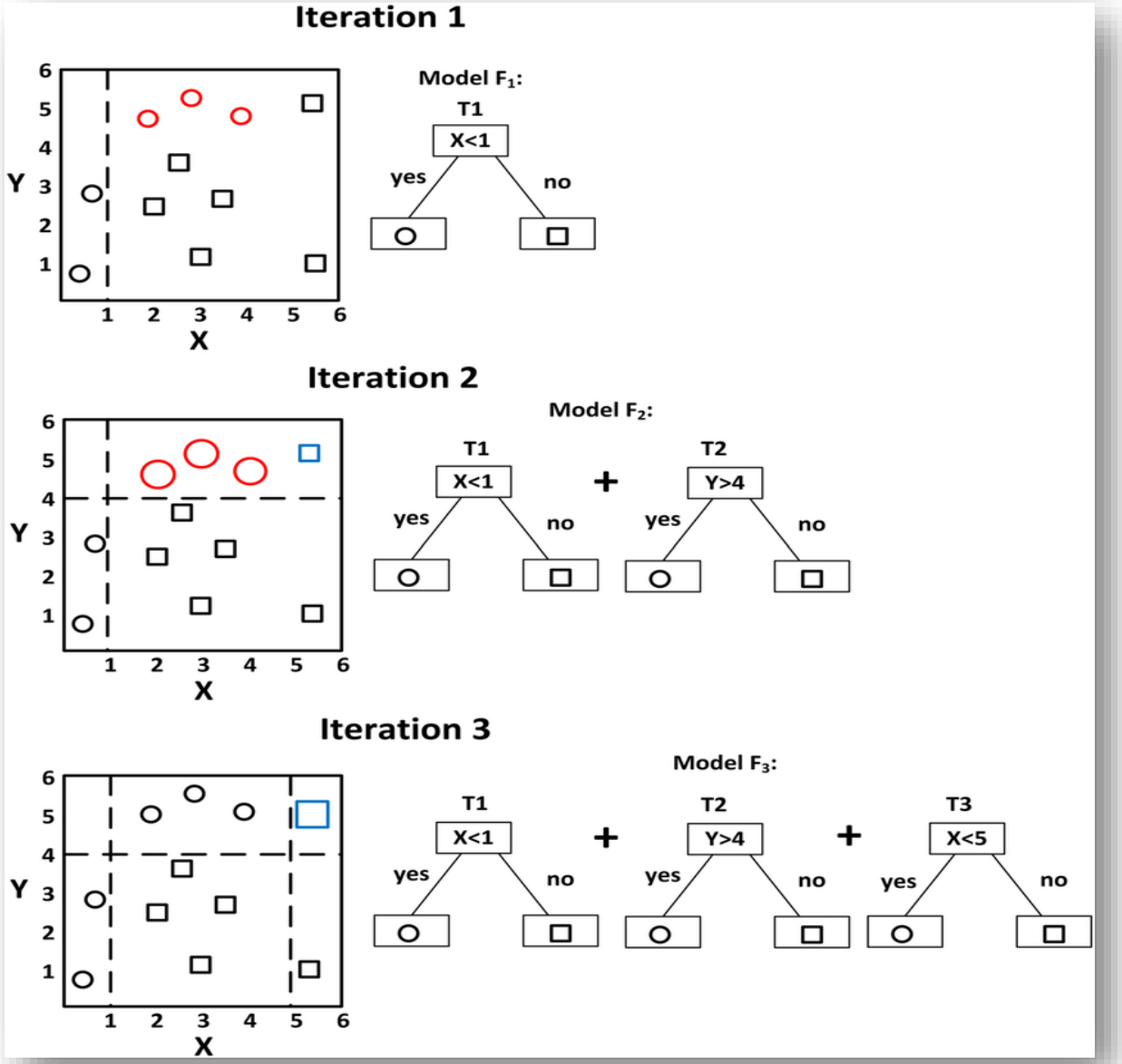
- it is easier to use with less need for tweaking parameters.
- Theoretically, AdaBoost is not prone to overfitting though there is no concrete proof for this.
- AdaBoost can be used to improve the accuracy of your weak classifiers hence making it flexible.
- It has now being extended beyond binary classification and has found use cases in text and image classification as well.

Disadvantages of AdaBoost

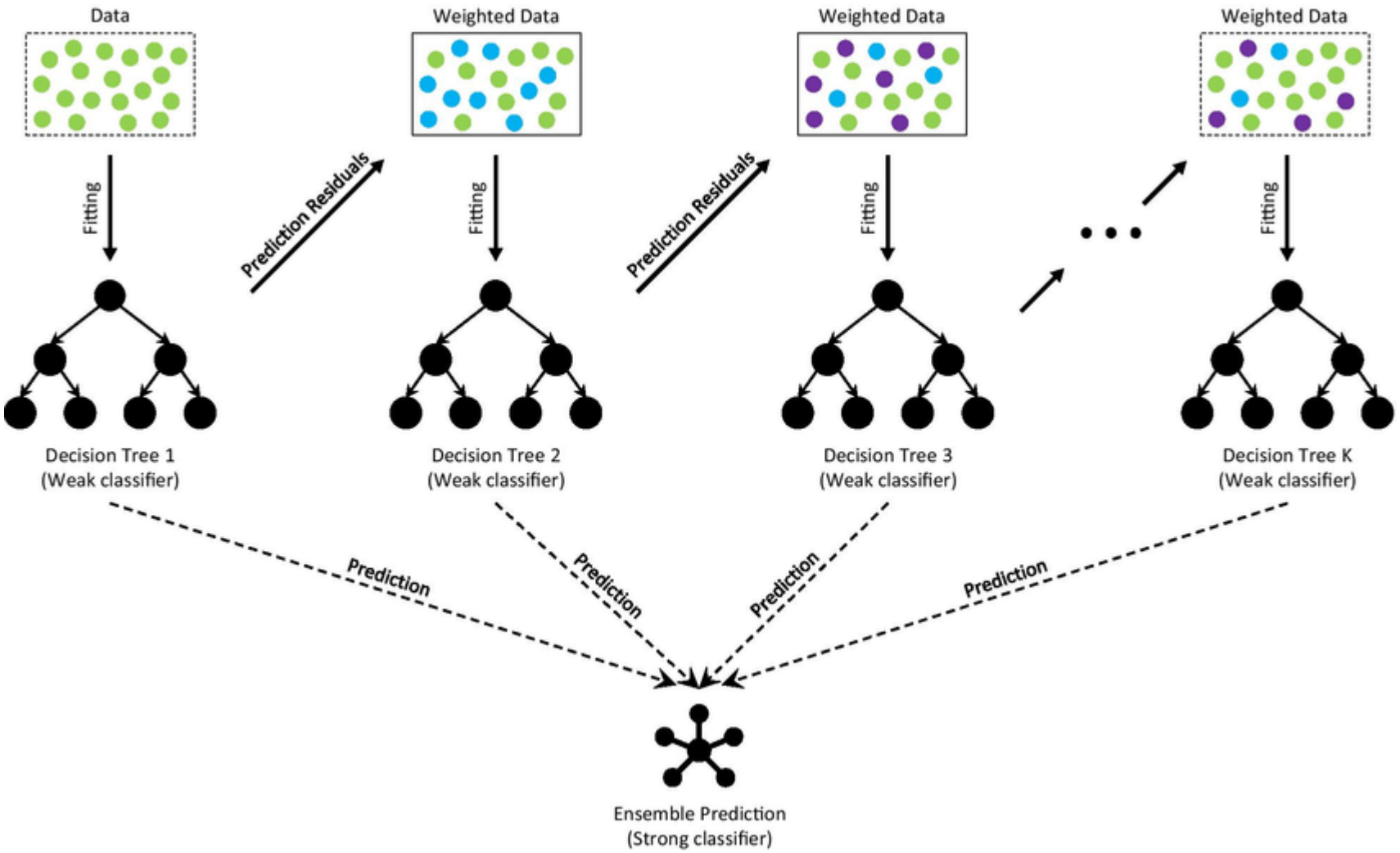
- Boosting technique learns progressively, it is important to ensure that you have quality data.
- AdaBoost is also extremely sensitive to Noisy data and outliers so if you do plan to use AdaBoost then it is highly recommended to eliminate them.
- AdaBoost has also been proven to be slower than XGBoost.

Gradient Boosting

- In brief, boosting uses sequences of decision trees that seek to reduce the residuals of the prior tree.
- In other words, each new tree uses the residual of the prior tree as the target variable for the current tree.
- In doing so, for each new tree, greater focus is given to the larger errors of the prior trees.
- After enough trees are created, the sum of all tree predictions is calculated to generate the final predicted value.



Gradient Boosting Decision Tree



Gradient Boosting components

- **Loss Function** - The role of the loss function is to estimate how good the model is at making predictions with the given data. For example, if we're trying to predict the weight of a person depending on some input variables (a regression problem), then the loss function would be something that helps us find the difference between the predicted weights and the observed weights.
- **Weak Learner** - A weak learner is one that classifies our data but does so poorly, perhaps no better than random guessing.
- **Additive Model** - This is the iterative and sequential approach of adding the trees (weak learners) one step at a time. After each iteration, we need to be closer to our final model. In other words, each iteration should reduce the value of our loss function.

Gradient Boosting **= Gradient Descent + Boosting**

- Gradient Boosting is the boosting algorithm that works on the principle of the stagewise addition method, where multiple weak learning algorithms are trained and a strong learner algorithm is used as a final model from the addition of multiple weak learning algorithms trained on the same dataset.
- In the gradient boosting algorithm, the first weak learner will not be trained on the dataset, it will simply return the mean of the particular column, and the residual for output of the first weak learner algorithm will be calculated which will be used as output or target column for next weak learning algorithm which is to be trained.
- Following the same pattern, the second weak learner will be trained and the residuals will be calculated which will be used as an output column again for the next weak learner, this is how this process will continue until we reach **zero residuals**.
- In gradient boosting the dataset should be in the form of numerical or categorical data and the loss function using which the residuals are calculated should be differential at all points.

Gradient Boosting Regressor

- **Step -1** The first step in gradient boosting is to build a base model to predict the observations in the training dataset. For simplicity we take an average of the target column and assume that to be the predicted value as shown below:

Row No.	Cylinder Number	Car Height	Engine Location	Price
1	Four	48.8	Front	12000
2	Six	48.8	Back	16500
3	Five	52.4	Back	15500
4	Four	54.3	Front	14000

Gradient Boosting Regressor

- **Step-2** The next step is to calculate the pseudo residuals which are (observed value – predicted value)

Row No.	Cylinder Number	Car Height	Engine Location	Price	Prediction 1	Residual 1
1	Four	48.8	Front	12000	14500	-2500
2	Six	48.8	Back	16500	14500	2000
3	Five	52.4	Back	15500	14500	1000
4	Four	54.3	Front	14000	14500	-500

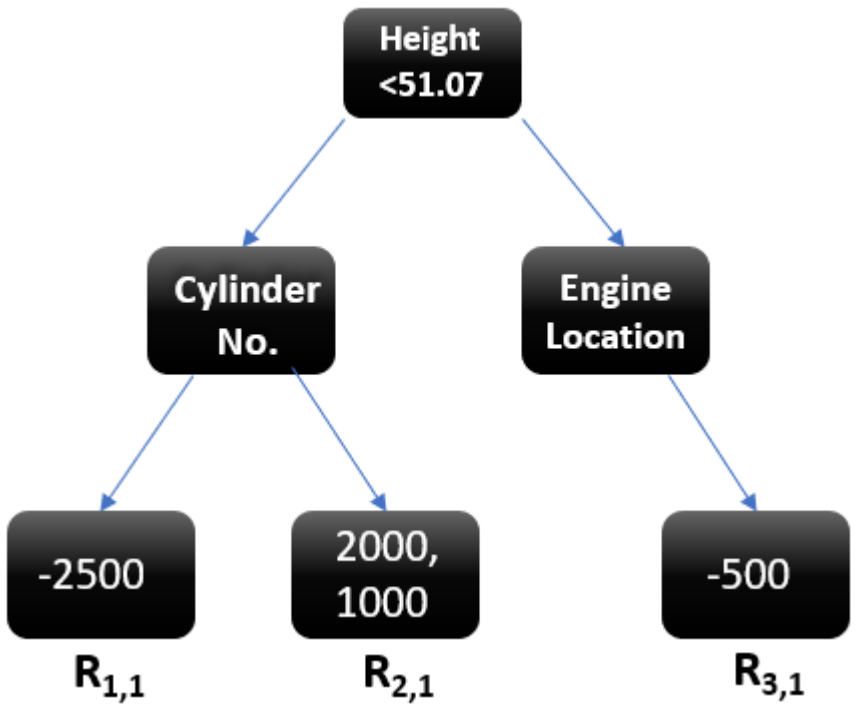
Gradient Boosting Regressor

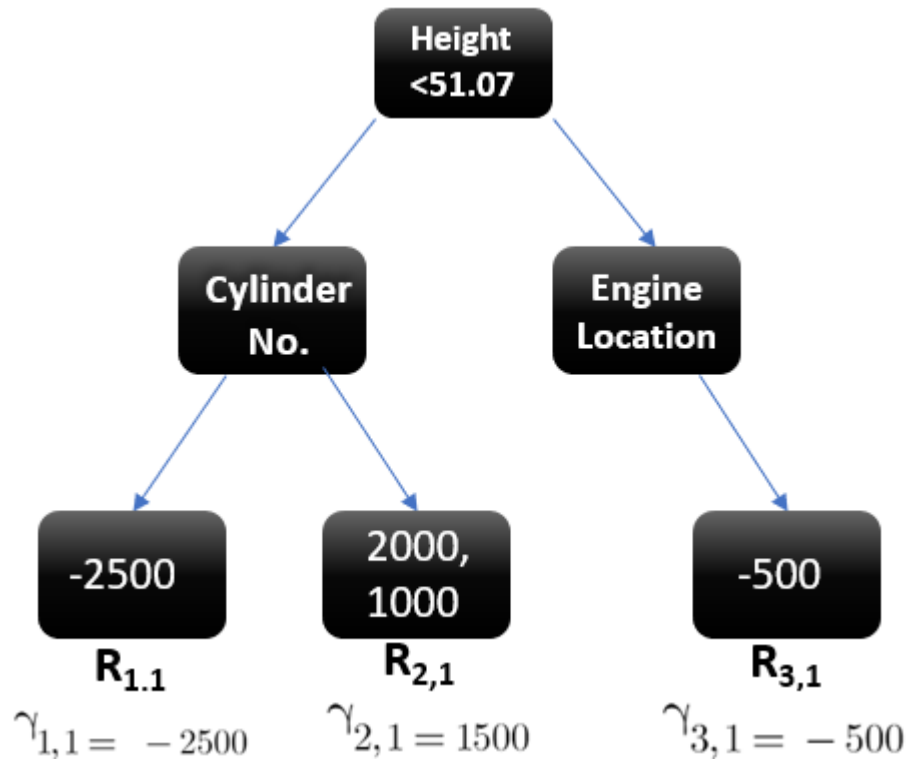
- Because we want to minimize these residuals and minimizing the residuals will eventually improve our model accuracy and prediction power.
- So, using the Residual as target and the original feature Cylinder number, cylinder height, and Engine location we will generate new predictions.

Gradient Boosting Regressor

- **Step- 4** In this step we find the output values for each leaf of our decision tree.
- That means there might be a case where 1 leaf gets more than 1 residual, hence we need to find the final output of all the leaves.
- TO find the output we can simply take the average of all the numbers in a leaf, doesn't matter if there is only 1 number or more than 1.

Row No.	Cylinder Number	Car Height	Engine Location	Price	Prediction 1	Residual 1
1	Four	48.8	Front	12000	14500	-2500
2	Six	48.8	Back	16500	14500	2000
3	Five	52.4	Back	15500	14500	1000
4	Four	54.3	Front	14000	14500	-500

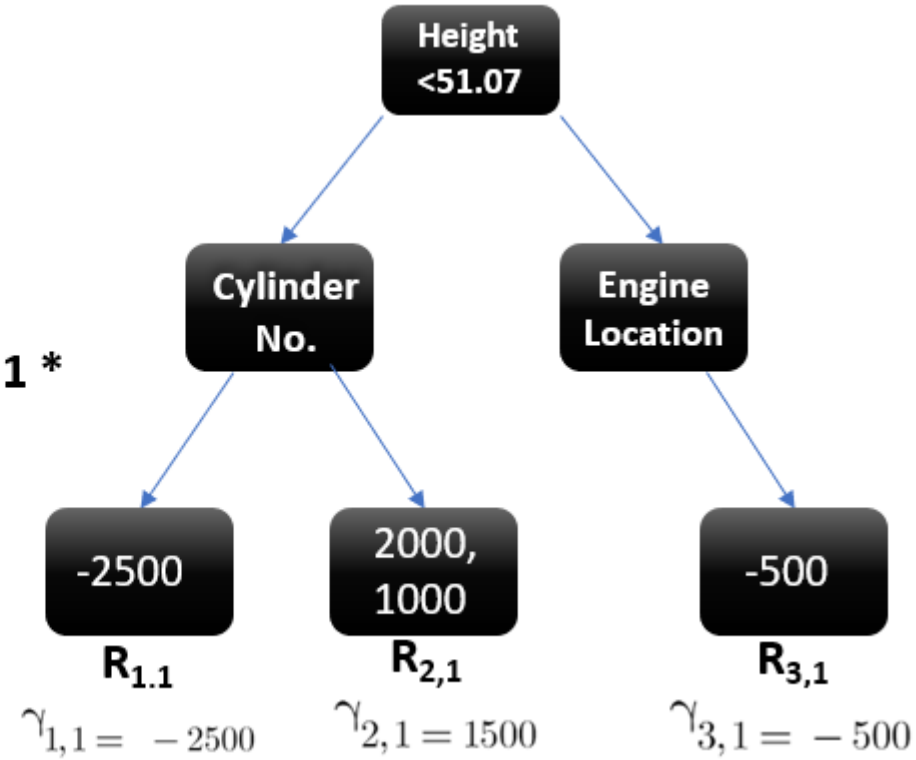




- **Step-5** This is finally the last step where we have to update the predictions of the previous model.

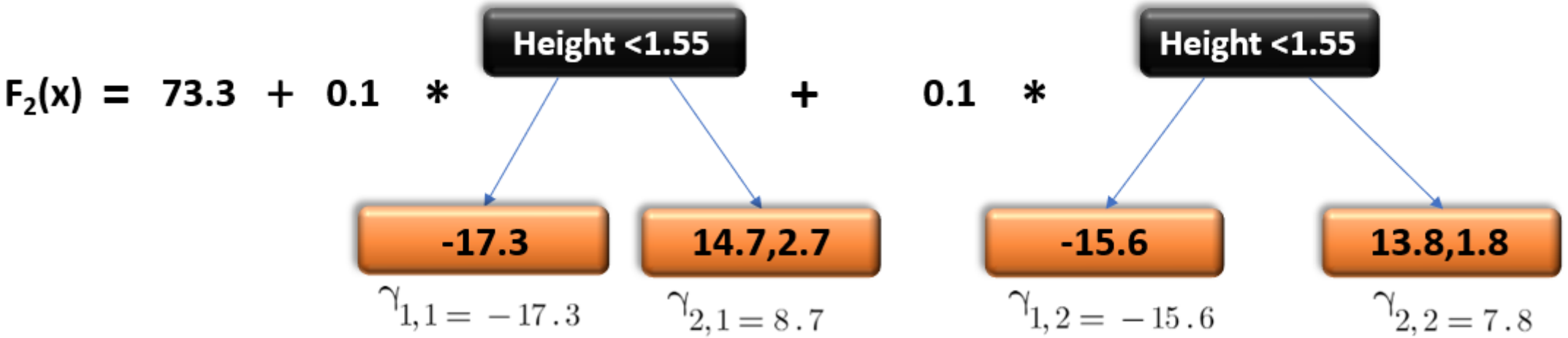
New Prediction = **Previous Prediction** + **Learning Rate** * **The tree made on Residuals**

New Prediction $F_1(x)$ = **14500** + **0.1** *



how this predicts for a new dataset

- If a new data point says height = 1.40 comes, it'll go through all the trees and then will give the prediction. Here we have only 2 trees hence the datapoint will go through these 2 trees and the final output will be **$F_2(x)$**.



Gradient Boost Classifier

Pclass	Age	Fare	Sex	Survived
3	22	7.25	male	0
1	38	71.2833	female	1
2	26	7.925	female	1
1	35	53.1001	female	1
3	8	21.07	male	0
3	27	11.133	female	1

Gradient Boost Classifier

$$log(\frac{survived}{notsurvived})$$

$$log(4/2) = 0.7$$

This becomes our initial leaf.

Gradient Boost Classifier

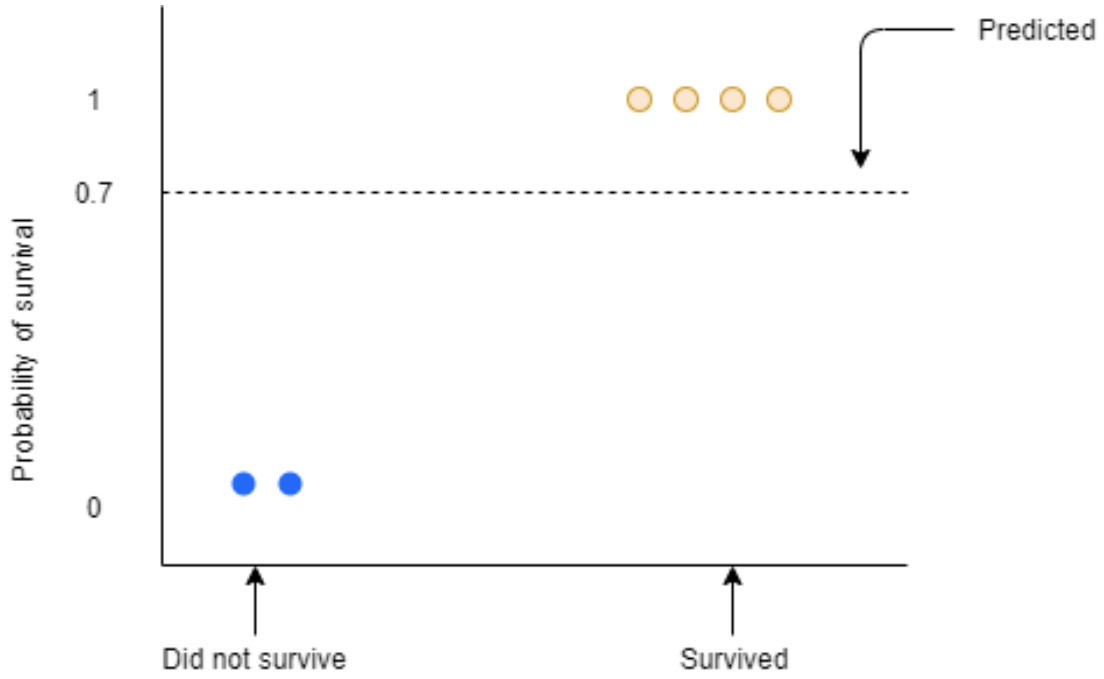
- The easiest way to use the log(odds) for classification is to convert it to a probability. To do so, we'll use this formula:

$$P(surviving) = \frac{e^{log(odds)}}{1 + e^{log(odds)}} = \frac{e^{0.2}}{1 + e^{0.2}} = 0.7$$

- If the probability of surviving is greater than 0.5, then we first classify everyone in the training dataset as survivors. (0.5 is a common threshold used for classification decisions made based on probability; note that the threshold can easily be taken as something else.)

Gradient Boost Classifier

- Now we need to calculate the **Pseudo Residual**, i.e, the difference between the observed value and the predicted value.



Gradient Boost Classifier

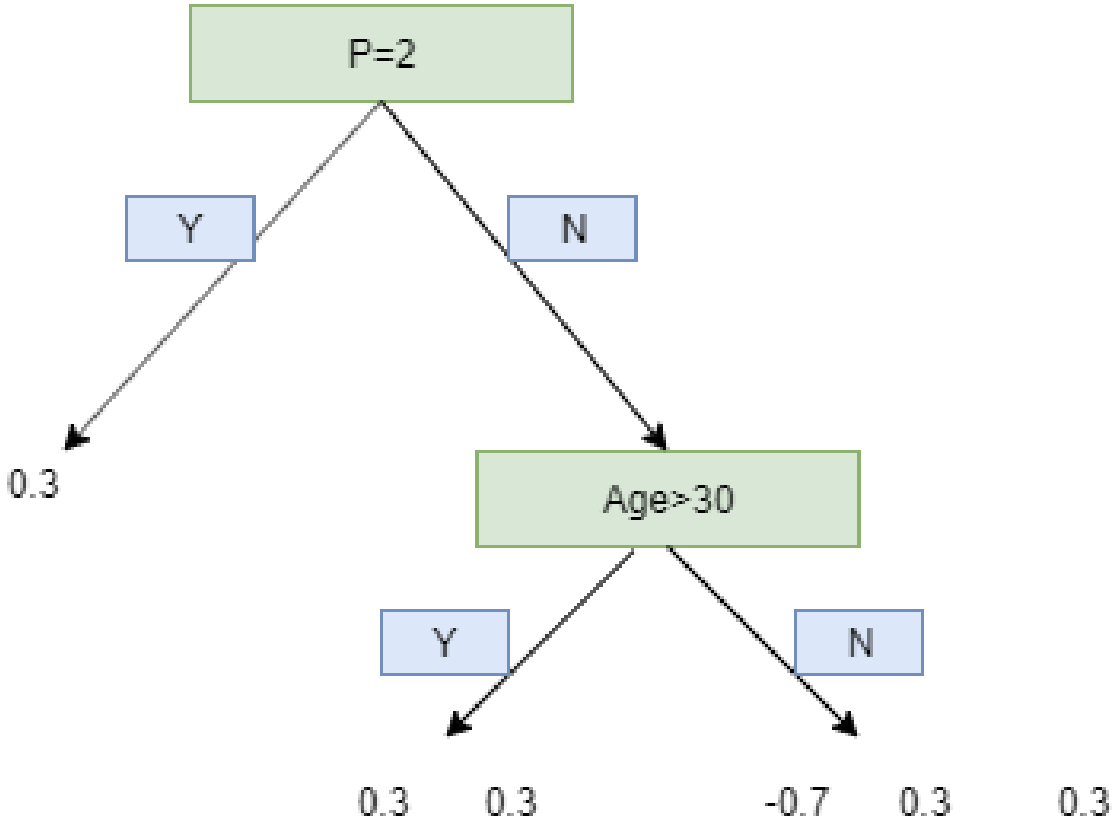
$$Residual = Observed - Predicted$$

Pclass	Age	Fare	Sex	Survived	Residual
3	22	7.25	male	0	-0.7
1	38	71.2833	female	1	0.3
2	26	7.925	female	1	0.3
1	35	53.1001	female	1	0.3
3	8	21.07	male	0	-0.7
3	27	11.133	female	1	0.3

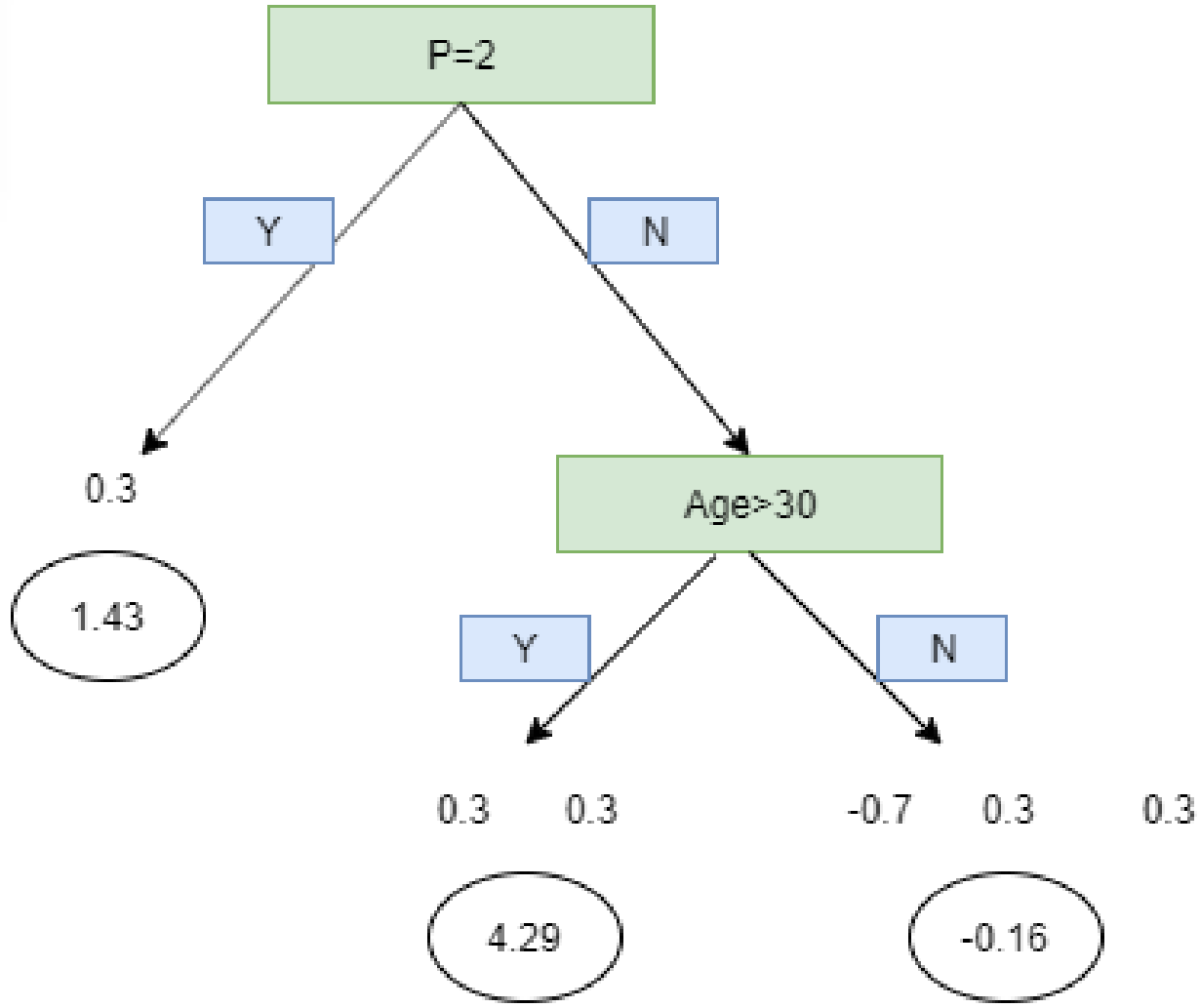
Gradient Boost Classifier

- We will use this residual to get the next tree.

Pclass	Age	Fare	Sex	Survived	Residual
3	22	7.25	male	0	-0.7
1	38	71.2833	female	1	0.3
2	26	7.925	female	1	0.3
1	35	53.1001	female	1	0.3
3	8	21.07	male	0	-0.7
3	27	11.133	female	1	0.3



$$\frac{\sum Residual}{\sum[PreviousProb * (1 - PreviousProb)]}$$



The first leaf

$$\frac{0.3}{[0.7 * (1 - 0.7)]} = 1.43$$

For the second leaf,

$$\frac{0.3 + 0.3}{[0.7 * (1 - 0.7)] + [0.7 * (1 - 0.7)]} = 4.29$$

Similarly, for the last leaf:

$$\frac{-0.7 + 0.3 + 0.3}{[0.7 * (1 - 0.7)] + [0.7 * (1 - 0.7)] + [0.7 * (1 - 0.7)]} = -0.16$$

Gradient Boost Classifier

- Now that we have transformed it, we can add our initial lead with our new tree with a learning rate.

$$OldTree + LearningRate * NewTree$$

- We can now calculate new log(odds) prediction and hence a new probability.

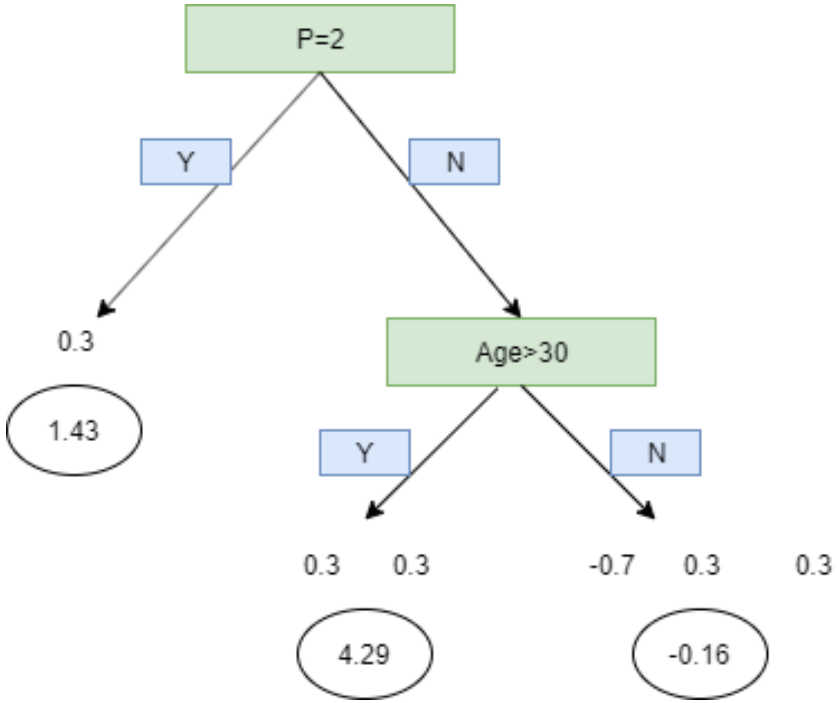
Gradient Boost Classifier

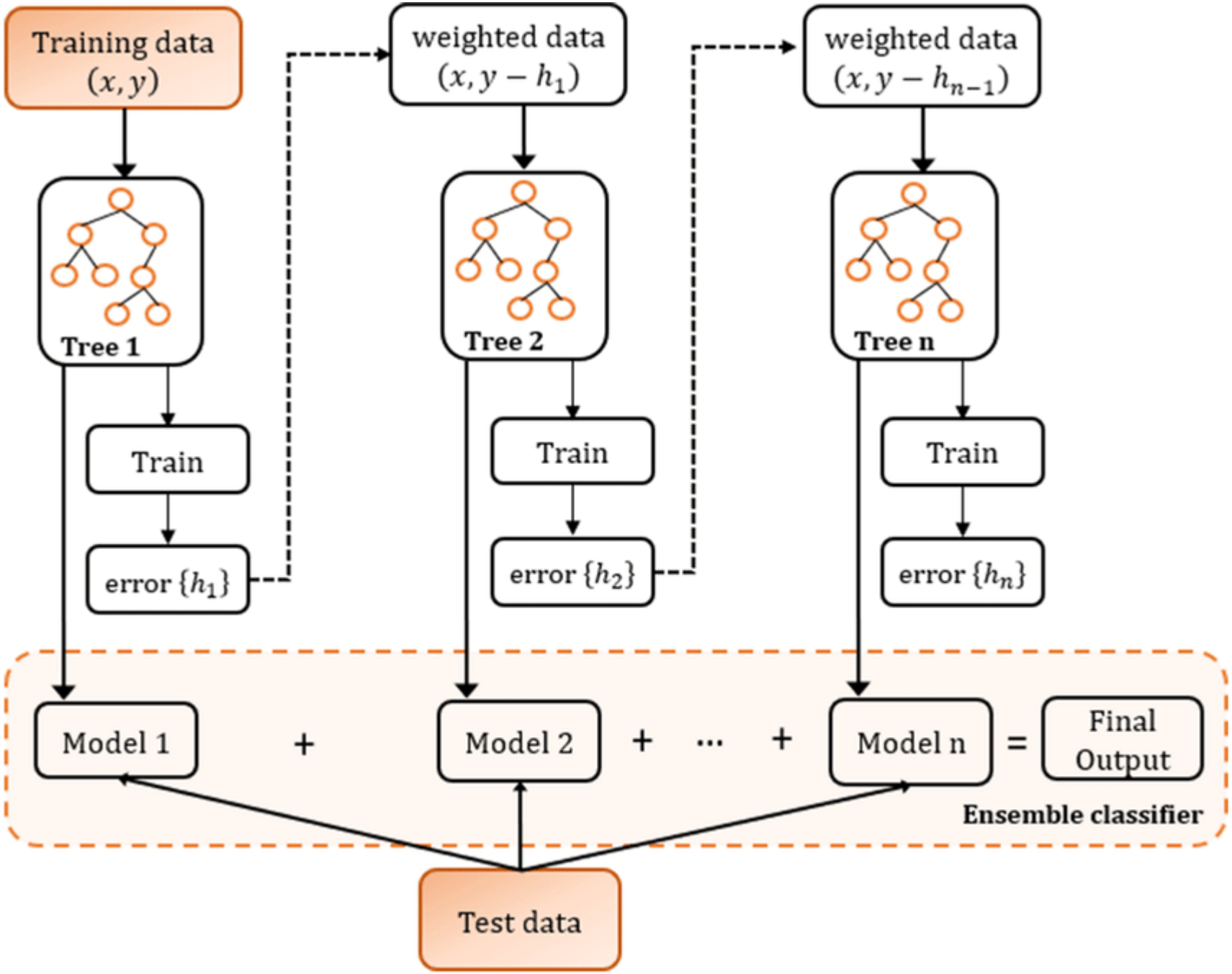
Pclass	Age	Fare	Sex	Survived	Residual
3	22	7.25	male	0	-0.7
1	38	71.2833	female	1	0.3
2	26	7.925	female	1	0.3
1	35	53.1001	female	1	0.3
3	8	21.07	male	0	-0.7
3	27	11.133	female	1	0.3

- For example, for the first passenger, Old Tree = 0.7. Learning Rate which remains the same for all records is equal to 0.1 and by scaling the new tree, we find its value to be -0.16. Hence, substituting in the formula we get:

$$0.7 + (0.1 * (-0.16)) = 0.684$$

- Similarly, we substitute and find the new log(odds) for each passenger and hence find the probability.
- Using the new probability, we will calculate the new residuals.
- This process repeats until we have made the maximum number of trees specified or the residuals get super small.




Training data
(x, y)
weighted data
$(x, y - h_1)$
weighted data
$(x, y - h_{n-1})$
Tree 1
Tree 2
Tree n
Train
Train
Train
error $\{h_1\}$
error $\{h_2\}$
error $\{h_n\}$
Model 1
+
Model 2
+
···
+
Model n
=
Final Output
Ensemble classifier
Test data

AdaBoost Vs. Gradient Boosting

- Both AdaBoost and Gradient Boost learn sequentially from a weak set of learners.
- A strong learner is obtained from the additive model of these weak learners. The main focus here is to learn from the shortcomings at each step in the iteration.
- AdaBoost requires users specify a set of weak learners (alternatively, it will randomly generate a set of weak learner before the real learning process).
- It increases the weights of the wrongly predicted instances and decreases the ones of the correctly predicted instances. The weak learner thus focuses more on the difficult instances. After being trained, the weak learner is added to the strong one according to its performance (so-called alpha weight). The higher it performs, the more it contributes to the strong learner.
- On the other hand, gradient boosting doesn't modify the sample distribution. Instead of training on a newly sampled distribution, the weak learner trains on the remaining errors of the strong learner.
- It is another way to give more importance to the difficult instances. At each iteration, the pseudo-residuals are computed and a weak learner is fitted to these pseudo-residuals.
- Then, the contribution of the weak learner to the strong one isn't computed according to its performance on the newly distributed sample but using a gradient descent optimization process. The computed contribution is the one minimizing the overall error of the strong learner.

AdaBoost Vs. Gradient Boosting

AdaBoost	Gradient Boosting
During each iteration in AdaBoost, the weights of incorrectly classified samples are increased, so that the next weak learner focuses more on these samples.	Gradient Boosting updates the weights by computing the negative gradient of the loss function with respect to the predicted output.
AdaBoost uses simple decision trees with one split known as the decision stumps of weak learners.	Gradient Boosting can use a wide range of base learners, such as decision trees, and linear models.
AdaBoost is more susceptible to noise and outliers in the data, as it assigns high weights to misclassified samples	Gradient Boosting is generally more robust, as it updates the weights based on the gradients, which are less sensitive to outliers.

Advantages of **Gradient Boosting**

- Frequently has remarkable predicting accuracy.
- Numerous choices for hyperparameter adjustment and the ability to optimize various loss functions.
- It frequently works well with numerical and categorical values without pre-processing the input.
- Deals with missing data; imputation is not necessary.

Disadvantages of **Gradient Boosting**

- Gradient Boosting classifier will keep getting better to reduce all inaccuracies. This may lead to overfitting and an overemphasis on outliers.
- Costly to compute since it frequently requires a large number of trees (>1000), which can be memory and time-consuming.
- Due to the high degree of flexibility, numerous variables interact and significantly affect how the technique behaves.

XGBoost

It has been a gold mine for kaggle competition winners.

The kaggle **avito challenge** 1st place winner **Owen Zhang** said,

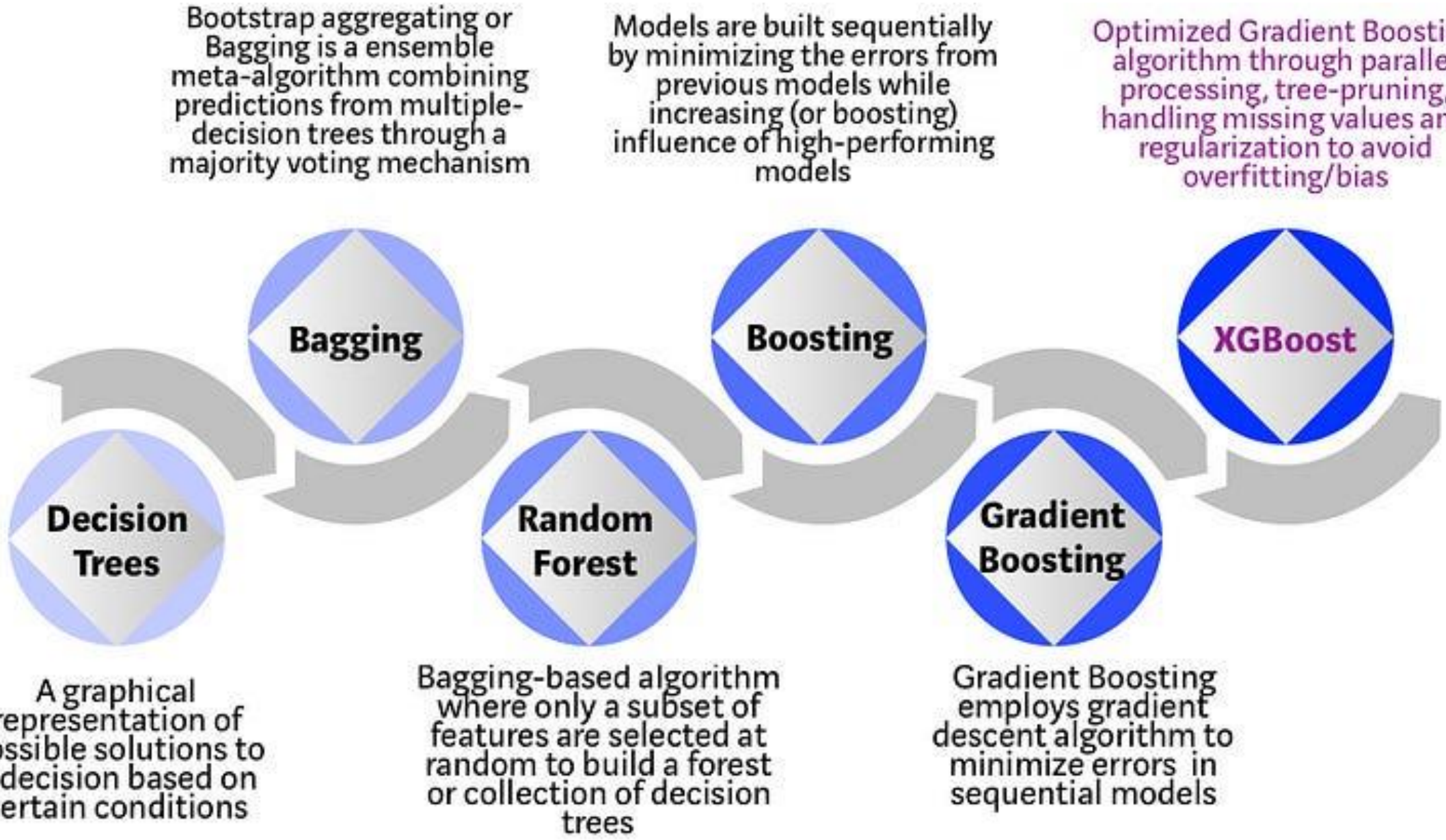
"When in doubt, just use XGBoost."

Whereas **Liberty mutual property challenge** 1st place winner **Qingchen wan** said,

.

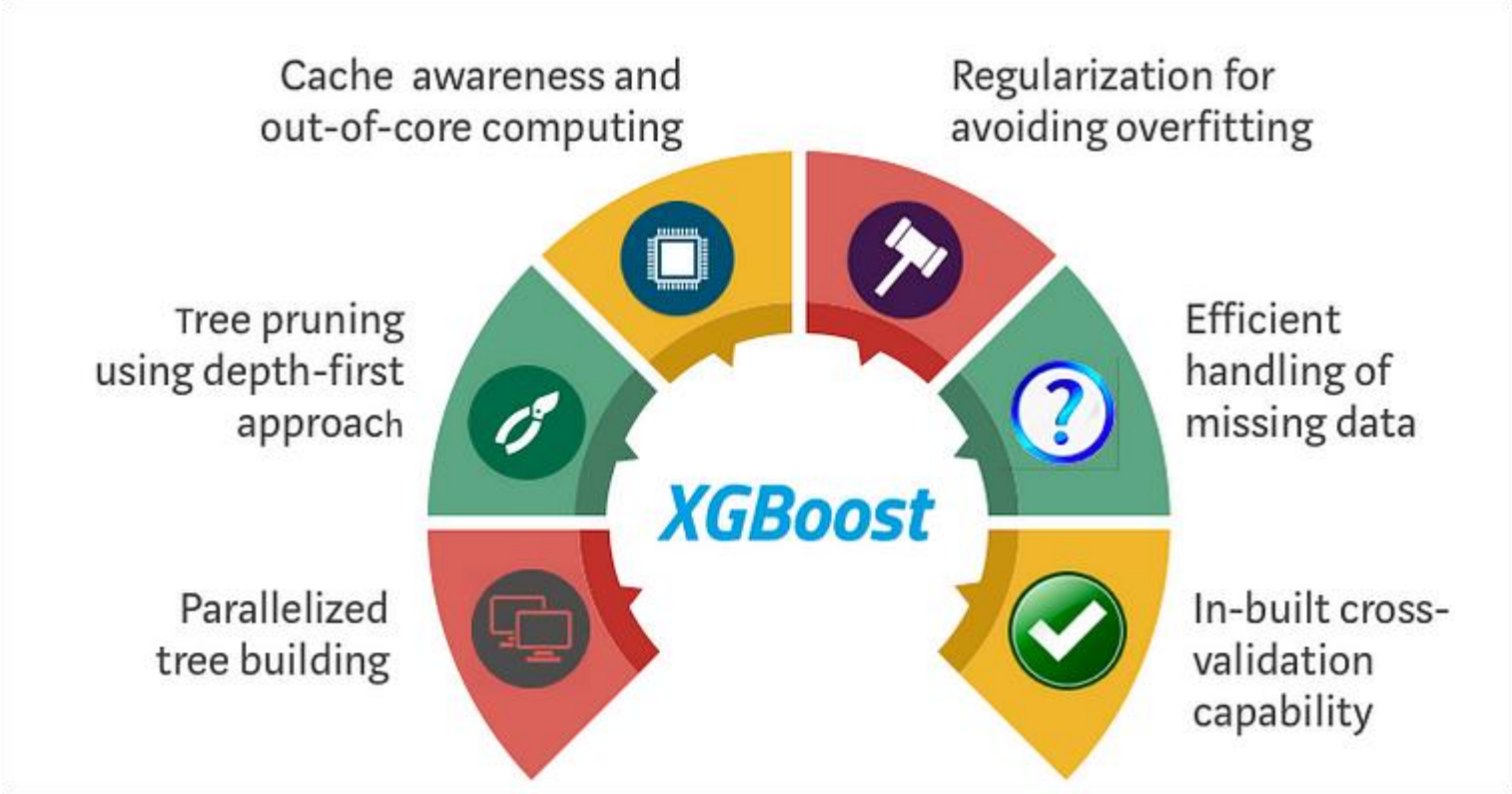
"I only used XGBoost."

The above two statements are enough to know the level impact of using the XGBoost algorithm in kaggle.


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

XGBoost

- XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.



System Optimization:

- **Parallelization**: XGBoost approaches the process of sequential tree building using parallelized implementation. This is possible due to the interchangeable nature of loops used for building base learners; the outer loop that enumerates the leaf nodes of a tree, and the second inner loop that calculates the features. This nesting of loops limits parallelization because without completing the inner loop (more computationally demanding of the two), the outer loop cannot be started. Therefore, to improve run time, the order of loops is interchanged using initialization through a global scan of all instances and sorting using parallel threads. This switch improves algorithmic performance by offsetting any parallelization overheads in computation.

- **Tree Pruning:** The stopping criterion for tree splitting within GBM framework is greedy in nature and depends on the negative loss criterion at the point of split. XGBoost uses 'max_depth' parameter as specified instead of criterion first, and starts pruning trees backward. This 'depth-first' approach improves computational performance significantly.

- **Hardware Optimization**: This algorithm has been designed to make efficient use of hardware resources. This is accomplished by cache awareness by allocating internal buffers in each thread to store gradient statistics. Further enhancements such as 'out-of-core' computing optimize available disk space while handling big data-frames that do not fit into memory.

Algorithmic Enhancements:

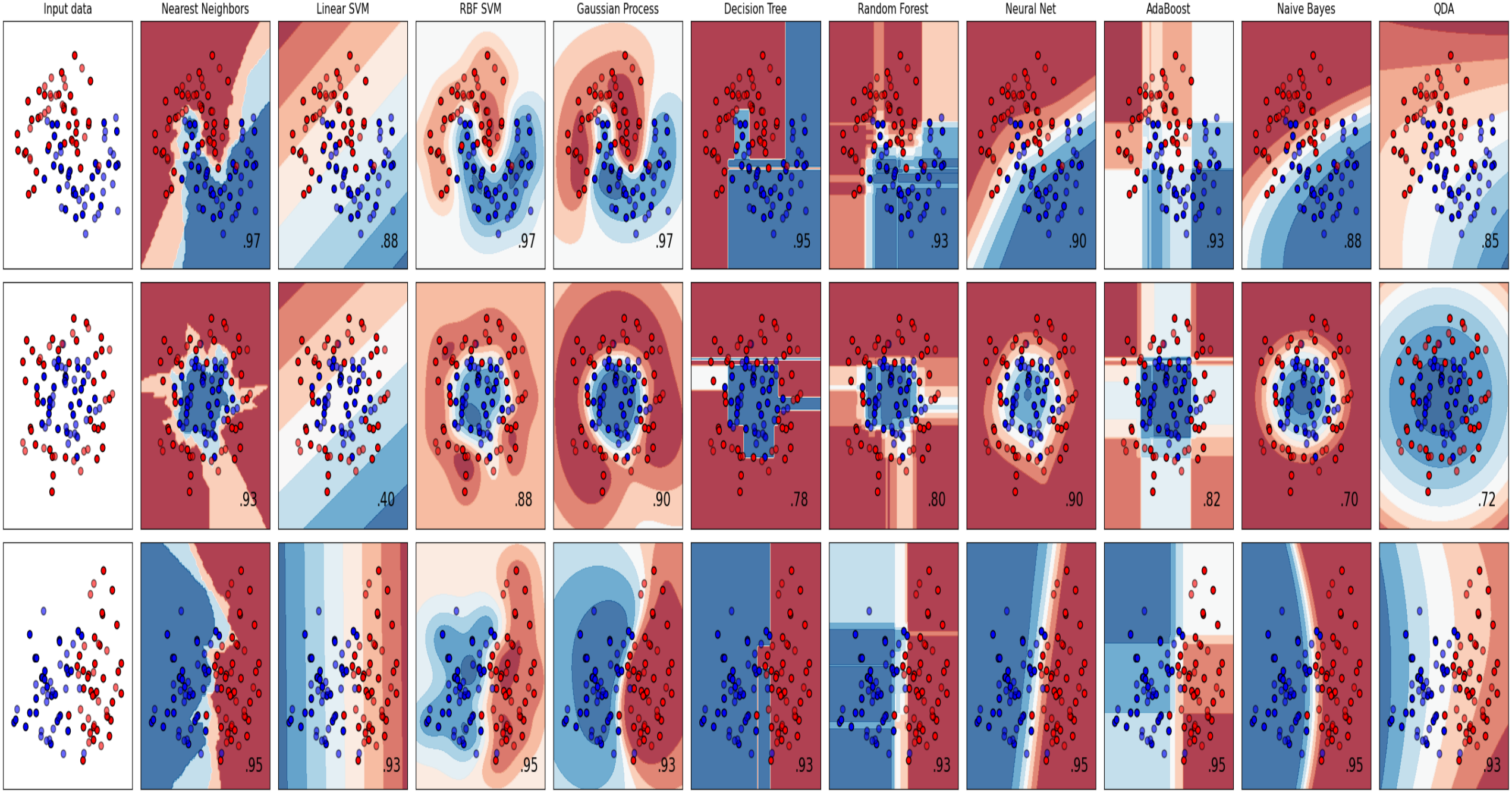
- **Regularization**: It penalizes more complex models through both LASSO (L1) and Ridge (L2) regularization to prevent overfitting.
- **Sparsity Awareness**: XGBoost naturally admits sparse features for inputs by automatically 'learning' best missing value depending on training loss and handles different types of sparsity patterns in the data more efficiently.
- **Weighted Quantile Sketch:** XGBoost employs the distributed weighted Quantile Sketch algorithm to effectively find the optimal split points among weighted datasets.
- **Cross-validation**: The algorithm comes with built-in cross-validation method at each iteration, taking away the need to explicitly program this search and to specify the exact number of boosting iterations required in a single run.

XGBoost

- XGBoost is also a boosting machine learning algorithm, which is the next version on top of the gradient boosting algorithm.
- The full name of the XGBoost algorithm is the eXtreme Gradient Boosting algorithm, as the name suggests it is an extreme version of the previous gradient boosting algorithm.
- The main difference between GradientBoosting is XGBoost is that XGboost uses a regularization technique in it. In simple words, it is a regularized form of the existing gradient-boosting algorithm.
- Due to this, XGBoost performs better than a normal gradient boosting algorithm and that is why it is much faster than that also.
- XGBoost (Extreme Gradient Boosting) is a specific implementation of GBM that introduces additional enhancements. XGBoost performs better than a normal gradient boosting algorithm and is much faster. A primary advantage of XGBoost over GBM are the greater set of hyper parameters that are able to be tuned.
- XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

Difference Between Boosting Algorithms

Algorithms	Gradient Boosting	AdaBoost	XGBoost	CatBoost	LightGBM
Year	–	1995	2014	2017	2017
Handling Categorical Variables	May require preprocessing like one-hot encoding	No	NO	Automatically handles categorical variables	No
Speed/Scalability	Moderate	Fast	Fast	Moderate	Fast
Memory Usage	Moderate	Low	Moderate	High	Low
Regularization	NO	No	Yes	Yes	Yes
Parallel Processing	No	No	Yes	Yes	Yes
GPU Support	No	No	Yes	Yes	Yes
Feature Importance	Available	Available	Available	Available	Available



Classifier comparison — scikit-learn 1.3.0 documentation

Bagging Vs. Boosting Vs. Stacking

	Bagging	Boosting	Stacking
Base-model Training	Parallel, independent	Sequential	Independent, and can be parallel
Base-model Type	Homogeneous	Homogeneous	Heterogeneous
Base-model Number	High	High	Low
Base-model Tree Depth	Deep	Shallow	NA
Base-model Weight	Equal weight	Higher weight for base-models with better performance	Higher weight for base-model predictions that contribute more to the meta-learner
Main Goal	Reduce variance	Reduce bias	Can be both depending on what algorithms are used for the base-models
Memory Use	High	Low	Depends on the base-model algorithm
CUP Use	High	Low	Depends on the base-model algorithm

S.NO	Bagging	Boosting
1.	The simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by the performance of previously built models.
5.	Different training data subsets are selected using row sampling with replacement and random sampling methods from the entire training dataset.	Every new subset contains the elements that were misclassified by previous models.
6.	Bagging tries to solve the over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	In this base classifiers are trained parallelly.	In this base classifiers are trained sequentially.
9	Example: The Random forest model uses Bagging.	Example: The AdaBoost uses Boosting techniques