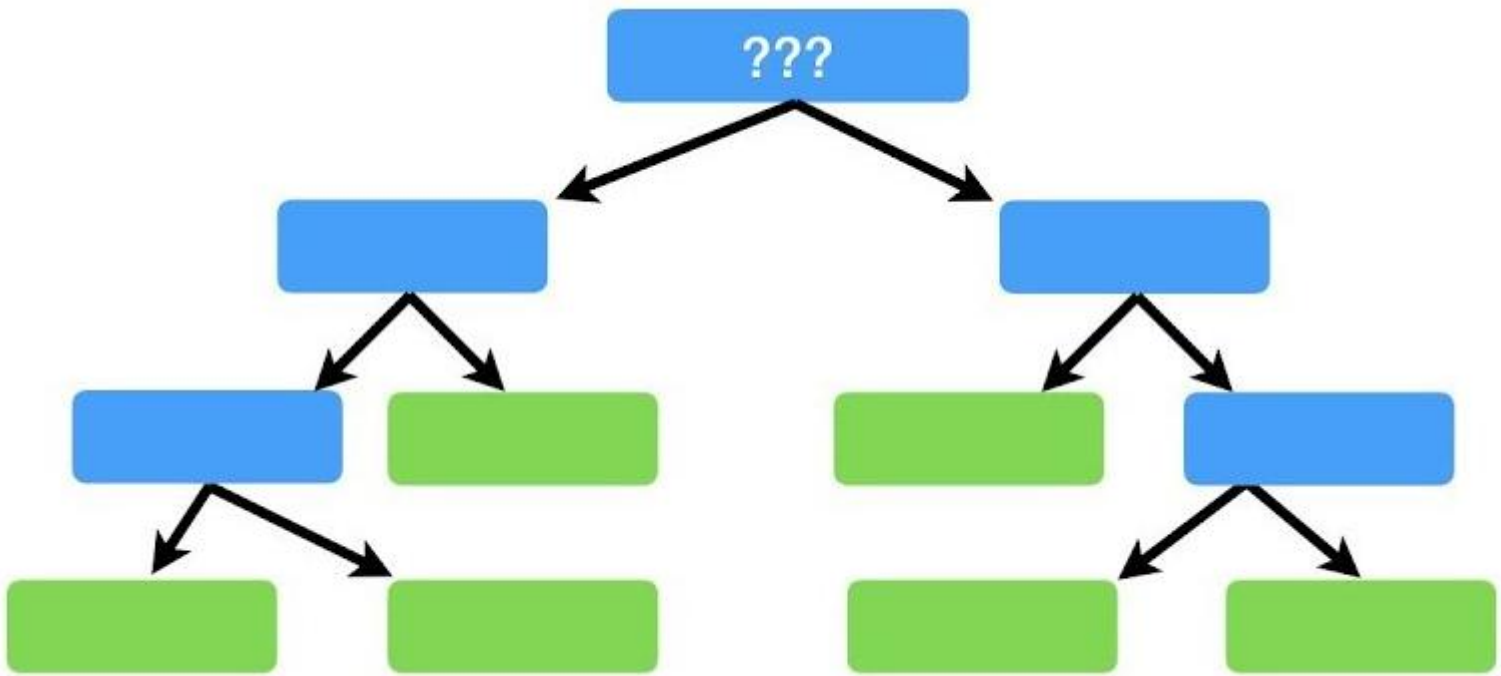
# Decision Tree

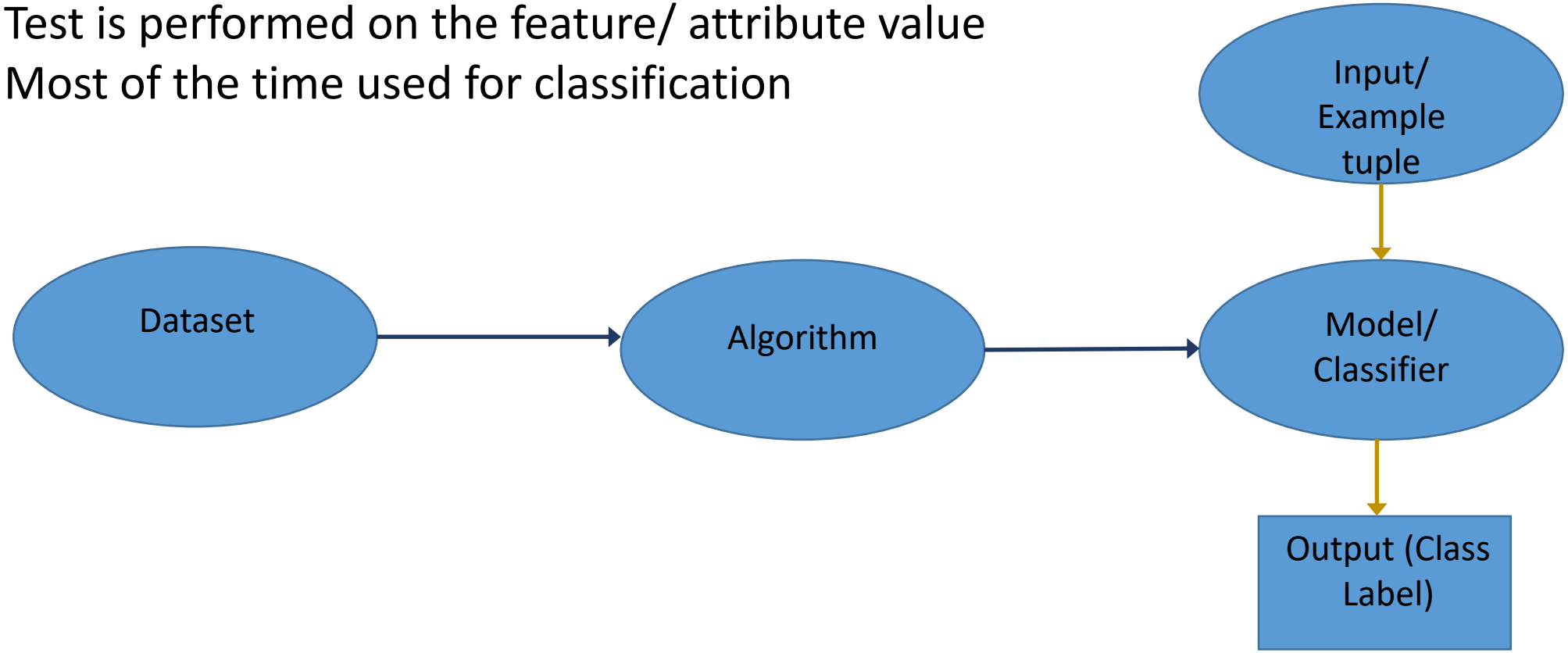


# Content…..

- Decision Tree Overview
- Decision Tree: Types of Methods
- ID3
- Gini Index
- Overfitting and Tree Pruining
- Strength
- Weakness

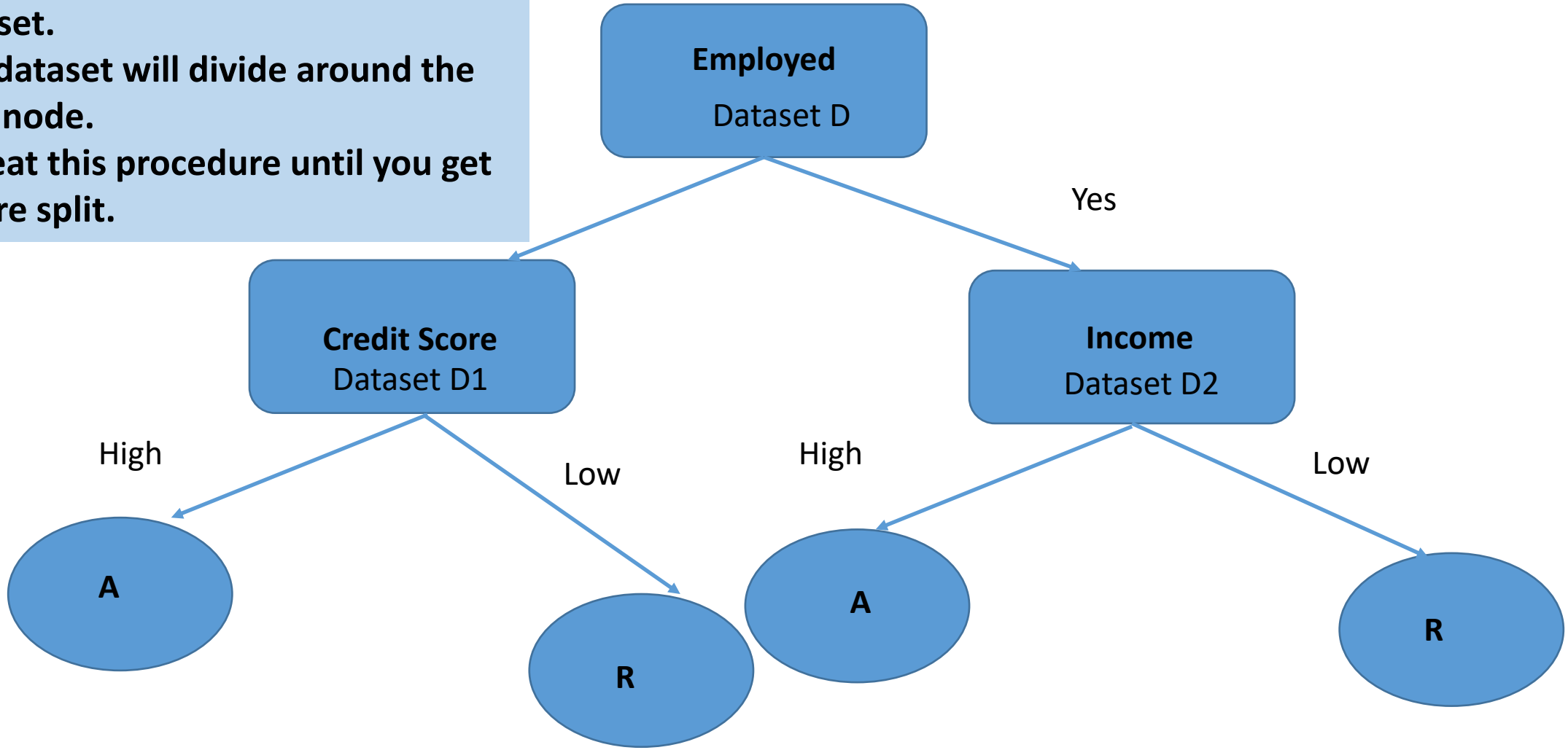
# Decision Tree: A Brief Idea

- A Tree Structured Classifier
- Also used for regression
- Internal Nodes – Decision nodes/ Tests
- Leaf nodes – Classification Values
- Test is performed on the feature/ attribute value
- Most of the time used for classification

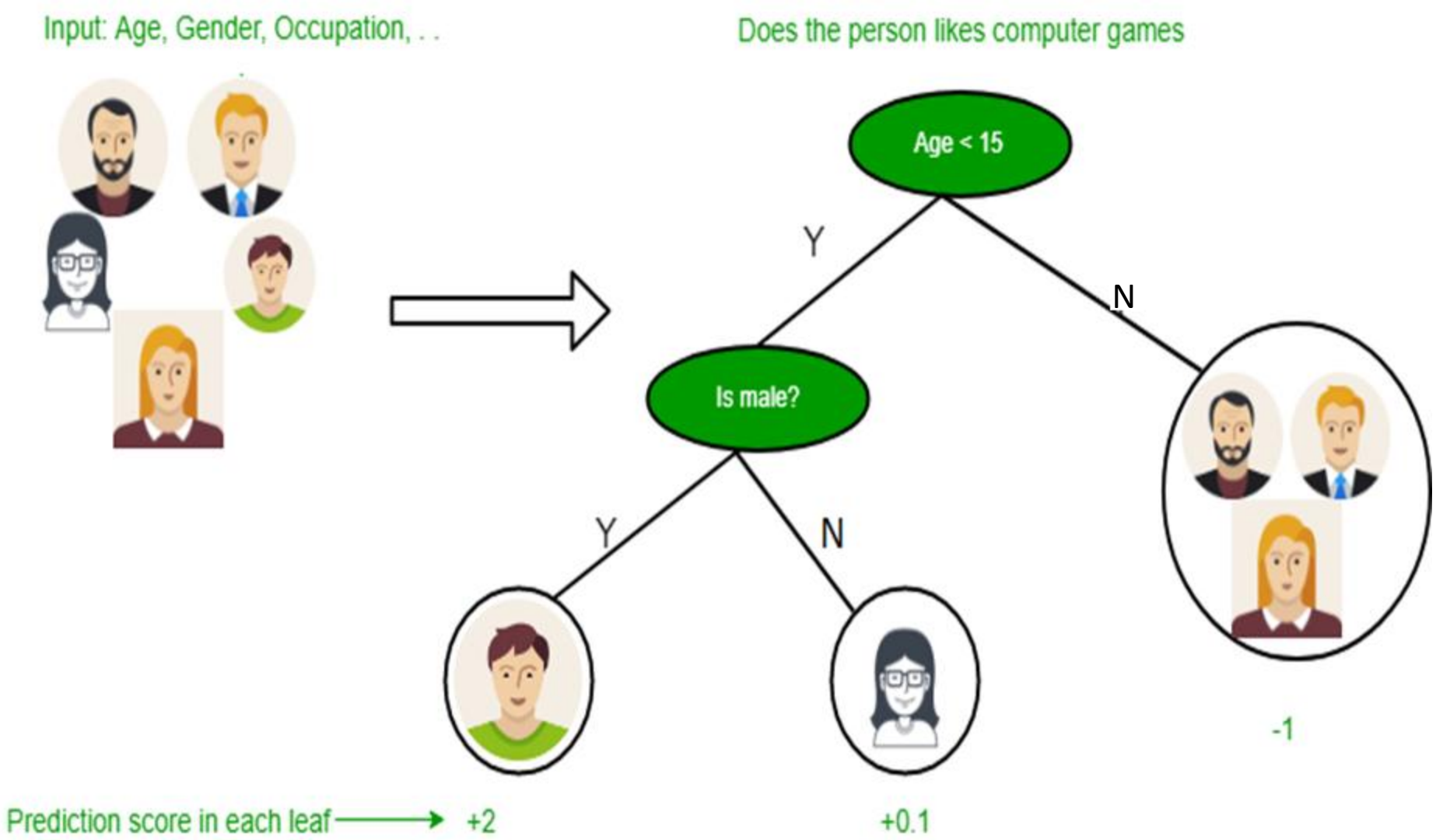


# Decision Tree: Example

1. **Give the root node the whole dataset.**
2. **The dataset will divide around the root node.**
3. **Repeat this procedure until you get a pure split.**



# Decision Tree: Example



# Decision Tree: Introduction

- The supervised learning category includes the decision tree method. The most powerful and widely used tool for categorization and prediction is the decision tree.

- A decision tree is a flowchart-like tree structure in which each internal node represents an attribute test, each branch reflects the test's conclusion, and each leaf node (class label) stores a class label.

- We may utilize decision trees to solve problems when the input and target attributes are both continuous and categorical.

# Decision Tree: Introduction

- The primary idea behind decision trees is to locate the descriptive features that contain the most "information" about the target feature and then partition the dataset along the values of these features, resulting in target feature values that are as pure as feasible.

- The most informative descriptive feature is the one that leaves the target feature as simple as possible.

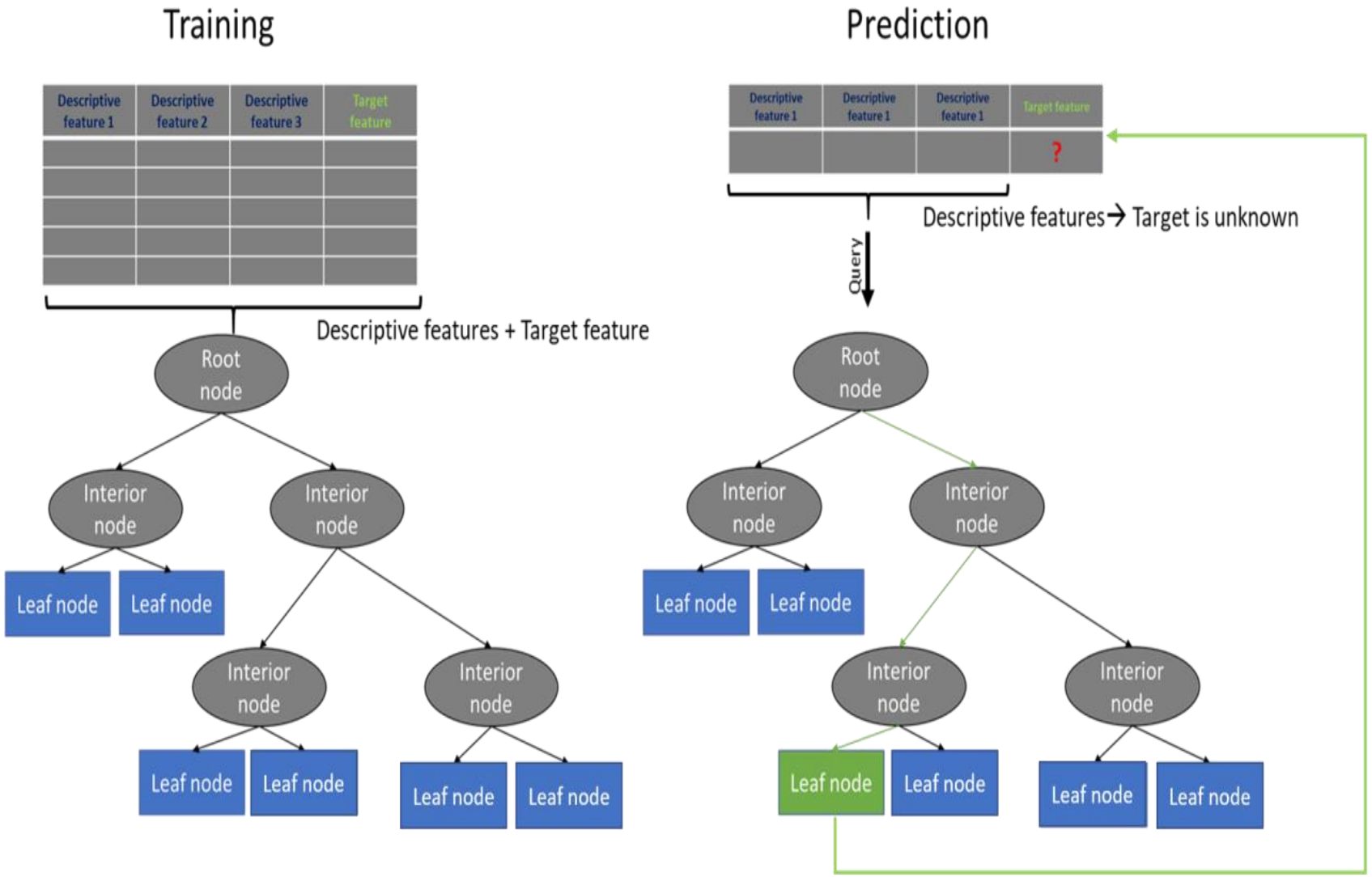
- This procedure of locating the "most informative" characteristic is repeated until a stopping condition is met, at which point we arrive at so-called leaf nodes.

- The predictions we'll make for new query instances provided to our trained model are stored in the leaf nodes.

# Decision Tree: Method of Splitting

1. Show a dataset with a number of training cases that are described by a number of descriptive attributes as well as a target feature.
2. Using a measure of information gain, train the decision tree model by constantly dividing the target feature along the values of the descriptive features during the training phase.
3. Continue to grow the tree until we reach a stopping point --> add leaf nodes to reflect the predictions we wish to make for future query instances.
4. Run down the tree until we reach the leaf nodes and show query instances to the tree.
5. DONE – with the query instance's response.

# Decision Tree: Method of Splitting



# Decision Tree: Types of Methods

**Algorithm used in decision trees:**

**1.ID3**

**2.Gini Index**

**Will be discussed here**

3.Chi-Square

4.Reduction in Variance

# Decision Tree: ID3

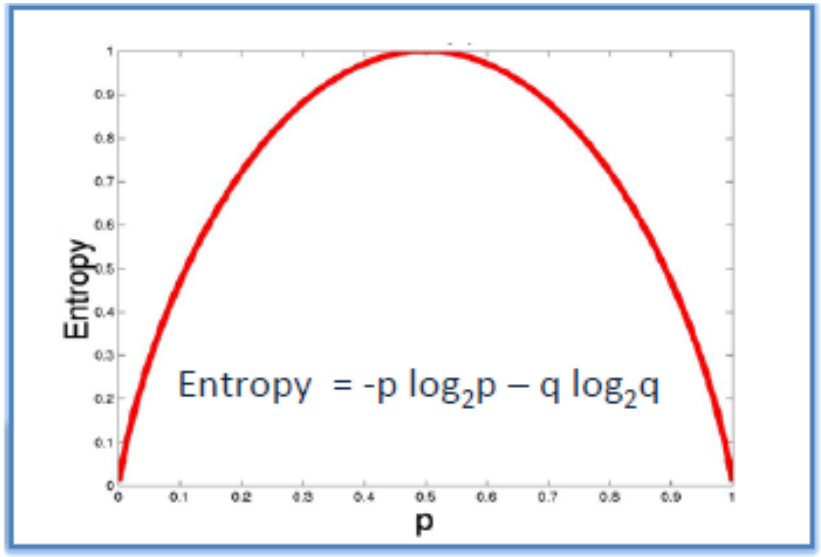
- ID3 is the name of the main decision tree-building algorithm.
- This method was created by J. R. Quinlan and uses a top-down, greedy search across the space of potential branches with no backtracking.
- To build a decision tree, ID3 employs Entropy and Information Gain.
- Entropy: A decision tree is constructed from the top down, starting with a root node, and entails splitting the data into subsets containing instances with comparable values (homogeneous).
- The ID3 algorithm calculates a sample's homogeneity using entropy. The entropy of a sample that is totally homogenous is zero, whereas a sample that is evenly split has an entropy of one.
- It is a metric for determining the purity of split. A pure subsplit is one in which all of the output belongs to a single category/ class.

# Decision Tree: ID3

Suppose I have feature f1 having 9 values (6 Yes and 3 No). If it is divided into two sets C1 (3 Yes and 3 No) and C2 (3 Yes and 0 No)

Entropy (C1) = -3/6*lg(3/6) – 3/6*lg(3/6)
= 1 bit (not a pure split)

Entropy (C2) = -3/3*lg(3/3) – 0/3*lg(0/3)
= 0 bit (pure split)




Entropy = -0.5 $\log_2 0.5$ – 0.5 $\log_2 0.5$ = 1

# *Algorithm for decision tree learning*

## *Basic algorithm (a greedy divide-and-conquer algorithm)*

- Assume attributes are categorical now
- Tree is constructed in a top-down recursive manner
- At start, all the training examples are at the root
- Examples are partitioned recursively based on selected attributes
- Attributes are selected on the basis of an impurity function (e.g., information gain)

## *Conditions for stopping partitioning*

- All examples for a given node belong to the same class
- There are no remaining attributes for further partitioning ? majority class is the leaf
- There are no examples left

# *Choose an attribute to partition data*

## *How chose the best attribute set?*

The objective is to reduce the impurity or uncertainty in data as much as possible

- A subset of data is *pure* if all instances belong to the same class.

The heuristic is to choose the attribute with the maximum *Information Gain* or *Gain Ratio* based on information theory.

## *Entropy of D*

Given a set of examples $D$ is possible to compute the original entropy of the dataset such as:

$$H[D] = -\sum_{j=1}^{|C|} P(c_j) log_2 P(c_j)$$

where $C$ is the set of desired class.

# Information Gain

## Entropy of an attribute $A_i$

If we make attribute $A_i$, with $v$ values, the root of the current tree, this will partition $D$ into $v$ subsets $D_1, D_2, \ldots, D_v$ . The expected entropy if $A_i$ is used as the current root:

$$H_{A_i}[D] = \sum_{j=1}^{v} \frac{|D_j|}{|D|} H[D_j]$$

## Information Gain

Information gained by selecting attribute $A_i$ to branch or to partition the data is given by the difference of *prior* entropy and the entropy of selected branch

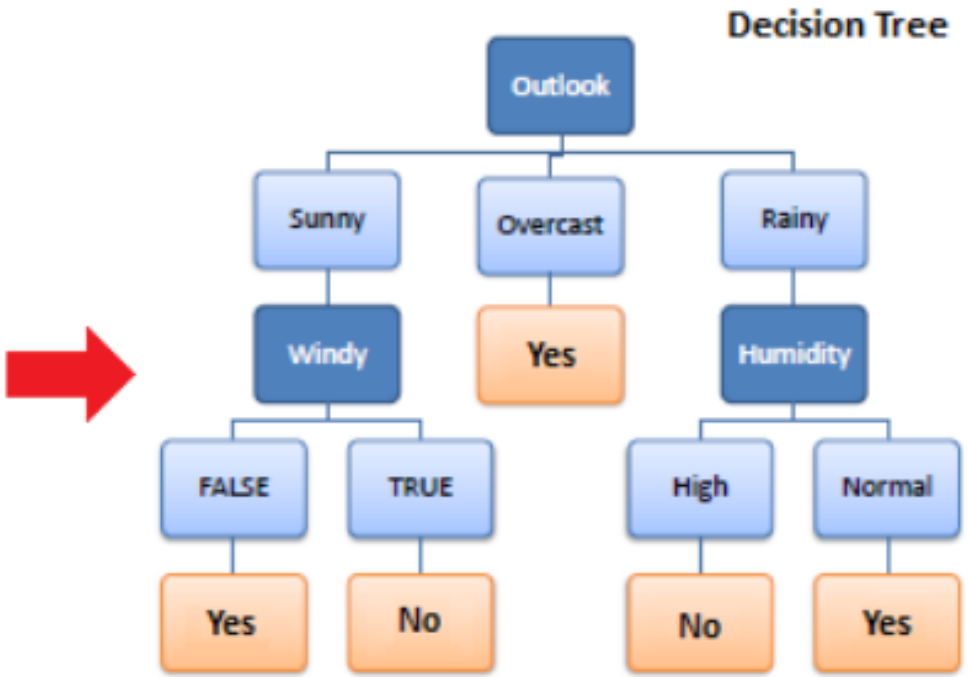
$$gain(D, A_i) = H[D] - H_{A_i}[D]$$

We choose the attribute with the *highest gain* to branch/split the current tree.

# Example

Predictors | Target

| Outlook | Temp | Humidity | Windy | Play Golf |
|---|---|---|---|---|
| Rainy | Hot | High | False | No |
| Rainy | Hot | High | True | No |
| Overcast | Hot | High | False | Yes |
| Sunny | Mild | High | False | Yes |
| Sunny | Cool | Normal | False | Yes |
| Sunny | Cool | Normal | True | No |
| Overcast | Cool | Normal | True | Yes |
| Rainy | Mild | High | False | No |
| Rainy | Cool | Normal | False | Yes |
| Sunny | Mild | Normal | False | Yes |
| Rainy | Mild | Normal | True | Yes |
| Overcast | Mild | High | True | Yes |
| Overcast | Hot | Normal | False | Yes |
| Sunny | Mild | High | True | No |



To create a decision tree, we must use frequency tables to determine two forms of entropy:

a) Entropy utilizing the target attribute's frequency table:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

| Play Golf | |
|---|---|
| Yes | No |
| 9 | 5 |

**Entropy(PlayGolf)** = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

b) Using the frequency table of other qualities to calculate entropy:

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

| | | Play Golf | | |
|---|---|---|---|---|
| | | Yes | No | |
| Outlook | Sunny | 3 | 2 | 5 |
| | Overcast | 4 | 0 | 4 |
| | Rainy | 2 | 3 | 5 |
| | | | | 14 |

**E**(PlayGolf, Outlook) = **P**(Sunny)*__E__(3,2) + **P**(Overcast)*__E__(4,0) + **P**(Rainy)*__E__(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

Information Gain: After a data set is divided on an attribute, the entropy decreases, resulting in information gain. Finding the attribute that offers the maximum information gain is the key to building a decision tree (i.e., the most homogeneous branches).
Step 1: Determine the target's entropy.

$$\begin{aligned}\textbf{Entropy(PlayGolf)} &= \text{Entropy } (5,9)\\ &= \text{Entropy } (0.36, 0.64)\\ &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64)\\ &= 0.94\end{aligned}$$

Step 2: The dataset is then segmented based on the various criteria. Each branch's entropy is computed. The total entropy for the split is then added proportionately. Before the split, the resultant entropy is deducted from the entropy. The Information Gain, or decrease in entropy, is the consequence.

| | | Play Golf | |
|---|---|---|---|
| | | Yes | No |
| Outlook | Sunny | 3 | 2 |
| | Overcast | 4 | 0 |
| | Rainy | 2 | 3 |
| Gain = 0.247 | | | |

| | | Play Golf | |
|---|---|---|---|
| | | Yes | No |
| Temp. | Hot | 2 | 2 |
| | Mild | 4 | 2 |
| | Cool | 3 | 1 |
| Gain = 0.029 | | | |

| | | Play Golf | |
|---|---|---|---|
| | | Yes | No |
| Humidity | High | 3 | 4 |
| | Normal | 6 | 1 |
| Gain = 0.152 | | | |

| | | Play Golf | |
|---|---|---|---|
| | | Yes | No |
| Windy | False | 6 | 2 |
| | True | 3 | 3 |
| Gain = 0.048 | | | |

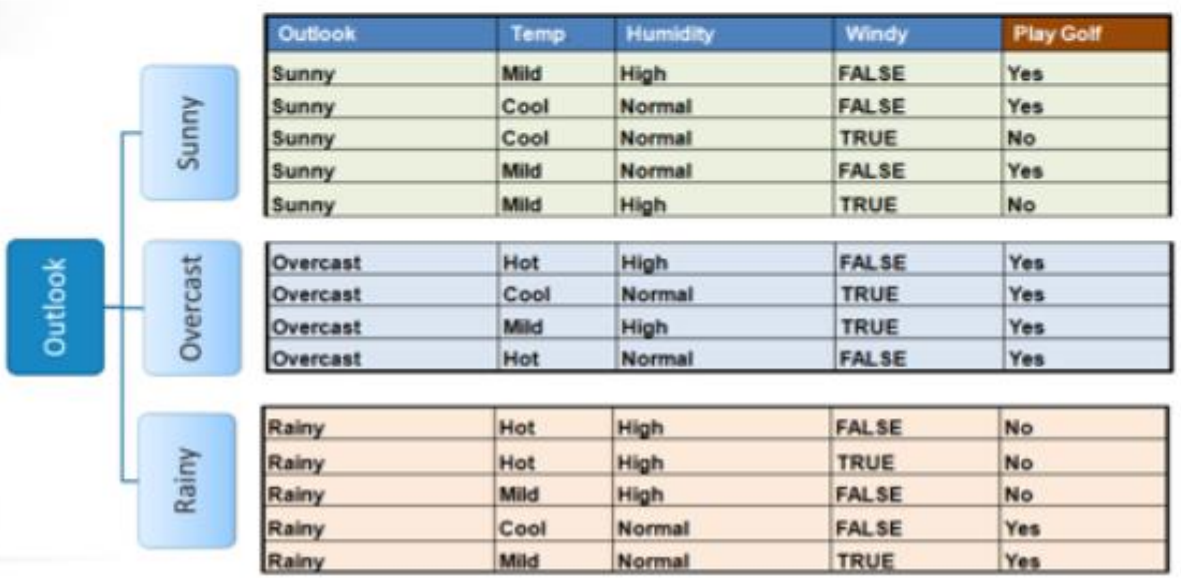
$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

**G**(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

*Step 3*: Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

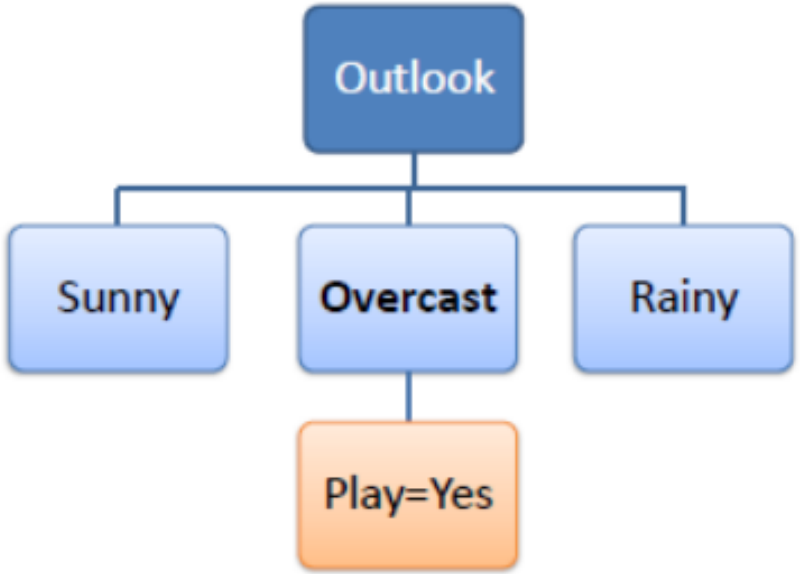
| ★ | | Play Golf | |
|---|---|---|---|
| | | Yes | No |
| Outlook | Sunny | 3 | 2 |
| | Overcast | 4 | 0 |
| | Rainy | 2 | 3 |
| Gain = 0.247 | | | |




| Outlook | Temp | Humidity | Windy | Play Golf |
|---|---|---|---|---|
| Sunny | Mild | High | FALSE | Yes |
| Sunny | Cool | Normal | FALSE | Yes |
| Sunny | Cool | Normal | TRUE | No |
| Sunny | Mild | Normal | FALSE | Yes |
| Sunny | Mild | High | TRUE | No |
| Overcast | Hot | High | FALSE | Yes |
| Overcast | Cool | Normal | TRUE | Yes |
| Overcast | Mild | High | TRUE | Yes |
| Overcast | Hot | Normal | FALSE | Yes |
| Rainy | Hot | High | FALSE | No |
| Rainy | Hot | High | TRUE | No |
| Rainy | Mild | High | FALSE | No |
| Rainy | Cool | Normal | FALSE | Yes |
| Rainy | Mild | Normal | TRUE | Yes |

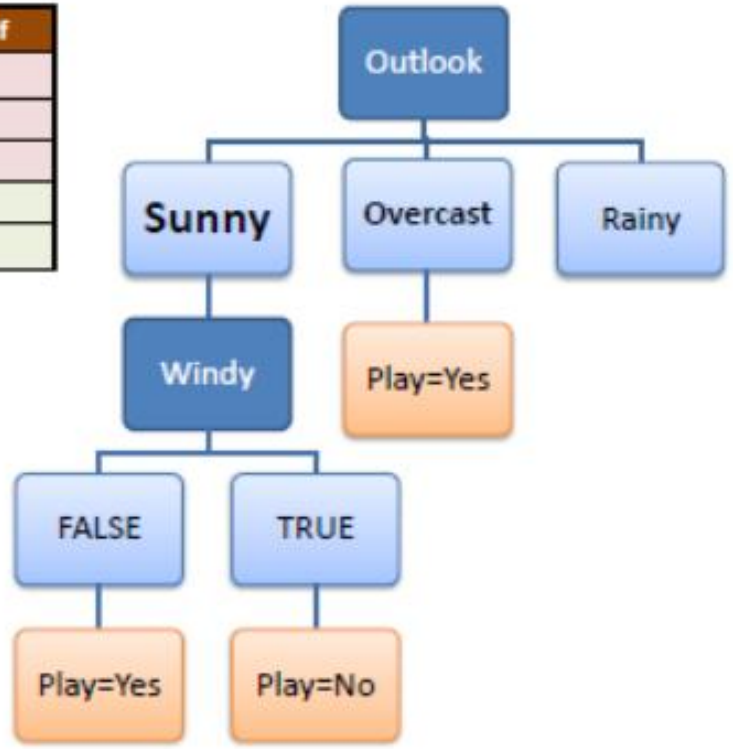
*Step 4a*: A branch with entropy of 0 is a leaf node.

| Temp | Humidity | Windy | Play Golf |
|---|---|---|---|
| Hot | High | FALSE | Yes |
| Cool | Normal | TRUE | Yes |
| Mild | High | TRUE | Yes |
| Hot | Normal | FALSE | Yes |



Step 4b: A branch with an entropy greater than 0 requires additional splitting.

| Temp | Humidity | Windy | Play Golf |
|---|---|---|---|
| Mild | High | FALSE | Yes |
| Cool | Normal | FALSE | Yes |
| Mild | Normal | FALSE | Yes |
| Cool | Normal | TRUE | No |
| Mild | High | TRUE | No |




Step 5: The ID3 algorithm is applied to the non-leaf branches in a recursive manner until all data has been categorised.

**Gini Index**

The Gini index states that if two objects are chosen at random from a population, they must belong to the same class, with a probability of one if the population is pure. It uses the category goal variables "Success" and "Failure" to work. It only does binary splits. The Gini approach is used by CART (Classification and Regression Tree) to construct binary splits.

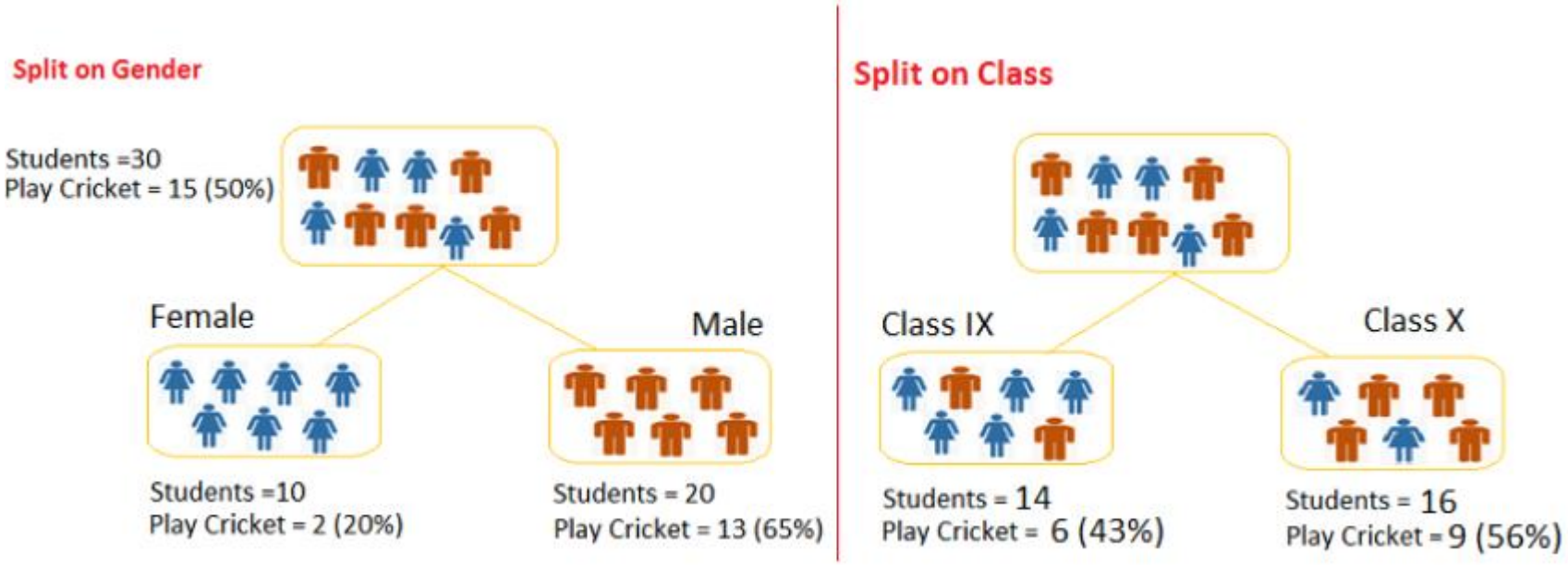
**Calculate Gini for a split**

1.Calculate Gini coefficients for sub-nodes using the formula sum of squared probability (p2+q2) for success and failure.

2.Calculate Gini for the split using the weighted Gini score of each split node.

## Example 1

As an example, suppose we wish to separate students based on a desired variable ( playing cricket or not ). We divided the population using two input factors, Gender and Class, in the picture below. Now, using the Gini index, I want to figure out which split produces the most homogenous sub-nodes.



**Split on Gender:**

1.Gini for sub-node Female = (0.2)*(0.2)+(0.8)*(0.8)=0.68

2.Gini for sub-node Male = (0.65)*(0.65)+(0.35)*(0.35)=0.55

3.Weighted Gini for Split Gender = (10/30)*0.68+(20/30)*0.55 = **0.59**

**Similar for Split on Class:**

1.Gini for sub-node Class IX = (0.43)*(0.43)+(0.57)*(0.57)=0.51

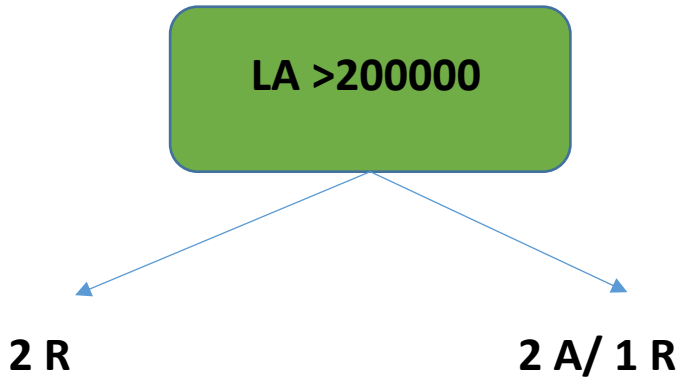
2.Gini for sub-node Class X = (0.56)*(0.56)+(0.44)*(0.44)=0.51

3.Weighted Gini for Split Class = (14/30)*0.51+(16/30)*0.51 = **0.51**

Above, you can see that Gini score for *Split on Gender* is higher than *Split on Class,* hence, the node split will take place on Gender.

# Example 2

| ID | Loan Amount | Loan Status |
|---|---|---|
| 1 | 100000 | R |
| 2 | 200000 | A |
| 3 | 250000 | R |
| 4 | 150000 | A |
| 5 | 300000 | R |




Gini Index for Left Branch:
[(0/2)*(0/2) + (2/2)*(2/2) = 1

Gini Index for Right Branch:
[(2/3)*(2/3) + (1/3)*(1/3) = 5/9

Final Gini Index = (2/5)*1 + (3/5)*(5/9)

Algorithm will compare the gini index for all the columns/ attributes, and wherever it is maximum, it will select that criteria for splitting. Vice versa is true if using (1- $\sum P^2$ ) for gini index calculation.

# Overfitting and Tree Pruning

- Many of the branches in a decision tree will represent abnormalities in the training data owing to noise or outliers.
- Overfitting the data is a problem that tree pruning approaches solve. Statistical metrics are often used in such procedures to eliminate the least-reliable branches.
- Pruned trees are smaller and less complicated, making them easier to understand. They are generally quicker and more accurate than unpruned trees in classifying independent test data (i.e., previously unseen tuples).

**Two approaches to avoid overfitting :**

1. Prepruning

2. Postpruning

# Overfitting and Tree Pruning

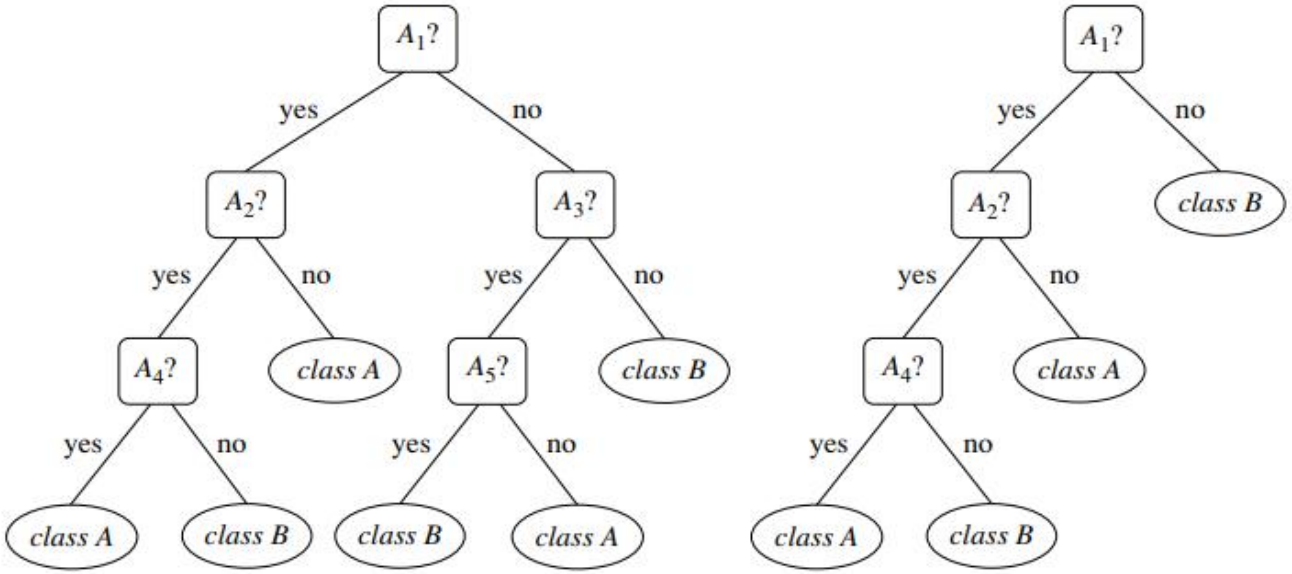
**Prepruning :**

- A tree is "pruned" by stopping its growth early in the prepruning method (e.g., by deciding not to further split or partition the subset of training tuples at a given node).
- When a node comes to a standstill, it transforms into a leaf. The most common class among the subset tuples, or the probability distribution of those tuples, may be stored in the leaf.
- Measures like statistical significance, information gain, Gini index, and others can be used to evaluate the quality of a split while building a tree.
- Further partitioning of the supplied subset is suspended if dividing the tuples at a node results in a split that falls below a predefined threshold.
- Choosing a suitable threshold, on the other hand, is challenging. High thresholds may lead to oversimplified trees, while low thresholds may lead to very little simplification.

# Overfitting and Tree Pruning

**Postpruning:**

The second and more popular method is postpruning, which involves the removal of subtrees from a "fully developed" tree. A subtree is pruned at a particular node by eliminating its branches and replacing them with leaves. The leaf is identified with the most common class in the subtree that is being replaced. For example, notice the subtree at node “A3?”




An unpruned decision tree and a pruned version of it.

Figure shows an unpruned tree. Assume that "class B" is the most prevalent class in this subtree. The subtree in question is trimmed in the pruned form of the tree by replacing it with the leaf "class B."

# Overfitting and Tree Pruning

**Postpruning:**

The postpruning strategy is shown by the cost complexity pruning algorithm employed in CART. The cost complexity of a tree is calculated using this method as a function of the number of leaves and the tree's mistake rate (where the error rate is the percentage of tuples misclassified by the tree). It all begins at the very bottom of the tree. It computes the cost complexity of the subtree at N and the cost complexity of the subtree at N if it were pruned for each internal node, N. (i.e., replaced by a leaf node). The two figures are compared and contrasted. If trimming the subtree at node N leads to a lower cost complexity, the subtree is trimmed. Otherwise, it is kept.

An independent test set is utilised to assess the correctness of each tree after creating a series of increasingly trimmed trees. It is preferable to use a decision tree that reduces the predicted error rate.

# Overfitting and Tree Pruning

We can prune trees depending on the amount of bits necessary to encode them rather than projected error rates. The tree with the smallest amount of encoding bits is the best trimmed. The Minimum Description Length (MDL) concept is used in this strategy. It does not require a separate set of samples, unlike cost complexity pruning.Prepruning and postpruning can also be done at the same time for a combined strategy. Although postpruning necessitates more calculation than prepruning, it usually results in a more stable tree. There hasn't been a single pruning strategy discovered to be better to all others. Although certain pruning approaches rely on the availability of extra data for pruning, when working with big databases, this is typically not an issue.

# Strengths of Decision Tree approach

- It's easy to comprehend and interpret. It is possible to visualise trees.It only takes a few minutes to prepare the data.
- Other procedures frequently need data normalisation, the creation of dummy variables, and the removal of blank values.
- This module, however, does not handle missing values.
- The cost of utilising the tree (that is, predicting data) is proportional to the amount of data points needed to train it.Capable of working with both numerical and category data.Statistical tests can be used to validate a model.
- As a result, the model's dependability may be accounted for.Even if the underlying model from which the data were created violates some of its assumptions, it still performs well.

# Weakness of Decision Tree approach

- Overly complicated trees can be created by decision-tree learners, which do not generalize the input well. This is referred to as overfitting. To prevent this problem, mechanisms like as pruning, establishing the minimum amount of samples required at a leaf node, and defining the maximum depth of the tree are required.

- Under numerous characteristics of optimality and even for basic notions, the issue of learning an optimal decision tree is known to be NP-complete. As a result, practical decision-tree learning algorithms rely on heuristic algorithms like the greedy algorithm, in which each node makes locally optimum judgments. Such algorithms can't promise that they'll give you the best decision tree in the world.

- This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

# THANK YOU