Decision Tree :→

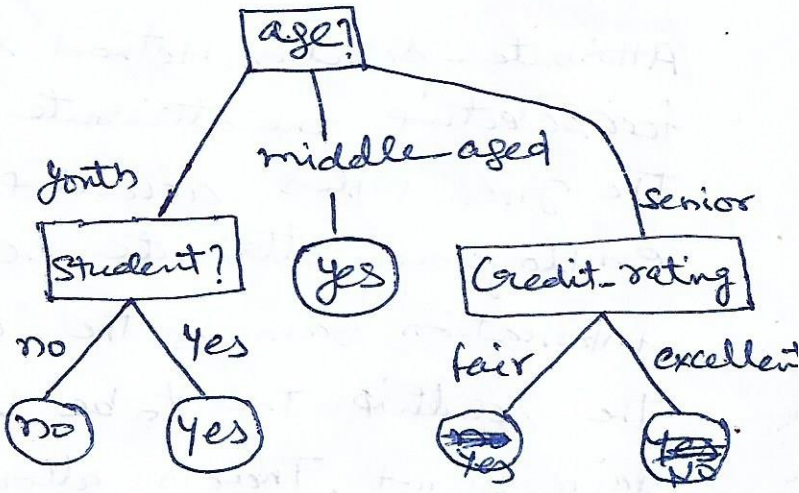
A decision Tree is a flow chart like structure, where each internal node (non leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.*

A typical decision Tree is shown in figure (a), it represents the concept buy-computer, i.e. it predicts whether a customer at AllElectronics is likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals. Some decision Tree algorithms produce only binary Trees (where each internal node branches to exactly Two other nodes), whereas others can produce non binary Trees.

"How are decision Trees used for classification?" Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision Tree. A path is ~~tested~~ Traced from the root to a leaf node, which holds the class prediction for that tuple. Decision Tree can easily be converted to classification rules. * root node of DT represents entire population or sample and this further get divided into two or more homogeneous sets.

Decision Tree induction :—

Decision Tree induction is the learning of decision Tree from class-labeled Training tuples. During the late 1970s and early 1980s. J. Ross Quinlan, a researcher in machine learning, Developed a decision Tree algorithm known as ID3 (Iterative Dichotomiser).



This work expanded on earlier work on concept learning systems, described by E. B. Hunt, J. Martin, and P. T. Stone. Quinlan later presented C4.5 (a successor of ID3), which became a benchmark to which newer supervised learning algorithms are often compared. In 1984, a group of statisticians published the book classification and Regression Trees (CART), which described the generation of binary decision trees.

ID3, C4.5, and CART adopt a greedy (i.e non backtracking) approach in which decision trees are constructed in a top-down recursive divide and conquer manner. Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subset as the tree is being built.

The Algorithm is called with three parameters: D, attribute_list, and attribute_selection_method. We refer to D as a Data Partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter attribute_list is a list of attributes describing the tuples. Attribute_selection_method specifies a heuristic procedure for selecting the attribute that "best" discriminates the given tuples according to class. This procedure employs an attribute selection measure such as Information gain or the Gini index. Gini index, enforce the resulting tree to be binary, others, like Information gain, do not, Therein allowing multiway splits.

The computational complexity of the algorithm given training set D is $O(n \times |D| \times \log |D|)$;

where n is the number of attributes describing the tuples in D and $|D|$ is the number of training tuples in D.

Attribute Selection Measures :→ It is also known as splitting rules because they determine how the tuples at a given node are to be split. An attribute selection measures is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes. If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class.

The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for measure is chosen as the splitting attribute for the given tuples. This section describes three popular attribute selection measures –

- Information gain
- gain ratio
- Gini index.

Type of decision Tree based on class label :- Types of decision Tree is based on the type of target variable we have, it can be of two types

(i) **Categorical variable decision Tree :-** Decision Tree which has categorical target variable then it is called as Categorical variable decision Tree.

Ex:- In above scenario of Student Problem, where the target variable was "Student will play cricket or not" i.e. Yes or No.

(ii) **Continuous variable decision Tree :-** Decision Tree has continuous target variable then it is called as continuous variable decision Tree.

Information gain :- ID3 uses information gain as its attribute selection measure. This measure is based on pioneering work by claude shannon on information theory, which studied the value or "information content" of messages. Let node N represent or hold the tuples of partition D. The attribute with highest information gain is chosen as the splitting attribute for node N. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflect the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple Tree is found.

The expected information needed to classify a tuple in D is given by

$$Info(D) = -\sum_{i=1}^{m} Pi \log_2(Pi),$$

or entropy (Parent) = " "

where P_i is the nonzero probability that an arbitrary tuple in D belongs to class C_i and is estimated by $\frac{|C_i, D|}{|D|}$.

A log function to the base 2 is used, because the information is encoded in bits.

$Info(D)$ is just the average amount of information needed to identify the class label of a tuple in D. $Info(D)$ is also known as the entropy of D.

How much more information would we still need (after the partitioning) to arrive at an exact classification? This amount is measured by

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

The term $\frac{|D_j|}{|D|}$ acts as the weight of the j^{th} partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information required, the greater the purity of the partitions.

Information gain is defined as the difference b/w the original information requirement (i.e, based on just the proportion of classes) and the new requirement (i.e. obtained after partitioning on A). That is

$$Gain(A) = Info(D) - Info_A(D)$$

Children

Info gain = entropy(Parent) − (weighted Avg) * entropy(k)

The term $\frac{|D_j|}{|D|}$ acts as the weight of the j^{th} partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A.

In other words, Gain(A) tells us how much would be gained by branching on A. It is the expected reduction in the information requirement caused by knowing the value of A. The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N. This is showing by the following example:-

Table:- Class labeled Training Tuples from the All electronics customer Database

RID	Age	Income	Student	credit_rating	class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle Aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle-Aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	Medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	Middle-aged	Medium	no	excellent	yes
13	middle-Aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

* Here are two distinct classes (i.e m=2). Let class C_1 correspond to yes and class C_2 correspond to no. There are nine tuples of class yes and five tuples of class no. A (root) node N is created for the tuples in D. To find the splitting criterion for these tuples, we

must compute the information gain of each attribute. We first use eq. (1) to compute the expected information needed to classify a tuple in D:

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits}$$

Next we need to compute the expected information requirement for each attribute. Let's start with the attribute age. We need to look at the distribution of yes and no tuples for each category of age. For the age category of "youth," there are two yes tuples and three no tuples. For the category "senior," there are three yes tuples and two no tuples. Using eq (1) the expected information needed to classify a tuple in D if the tuples are partitioned according to age is

$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{2}{5} - \frac{2}{5}\log_2\frac{2}{5}\right)$$

$$\Rightarrow 0.694 \text{ bits}$$

Here the gain in information from such a partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D)$$

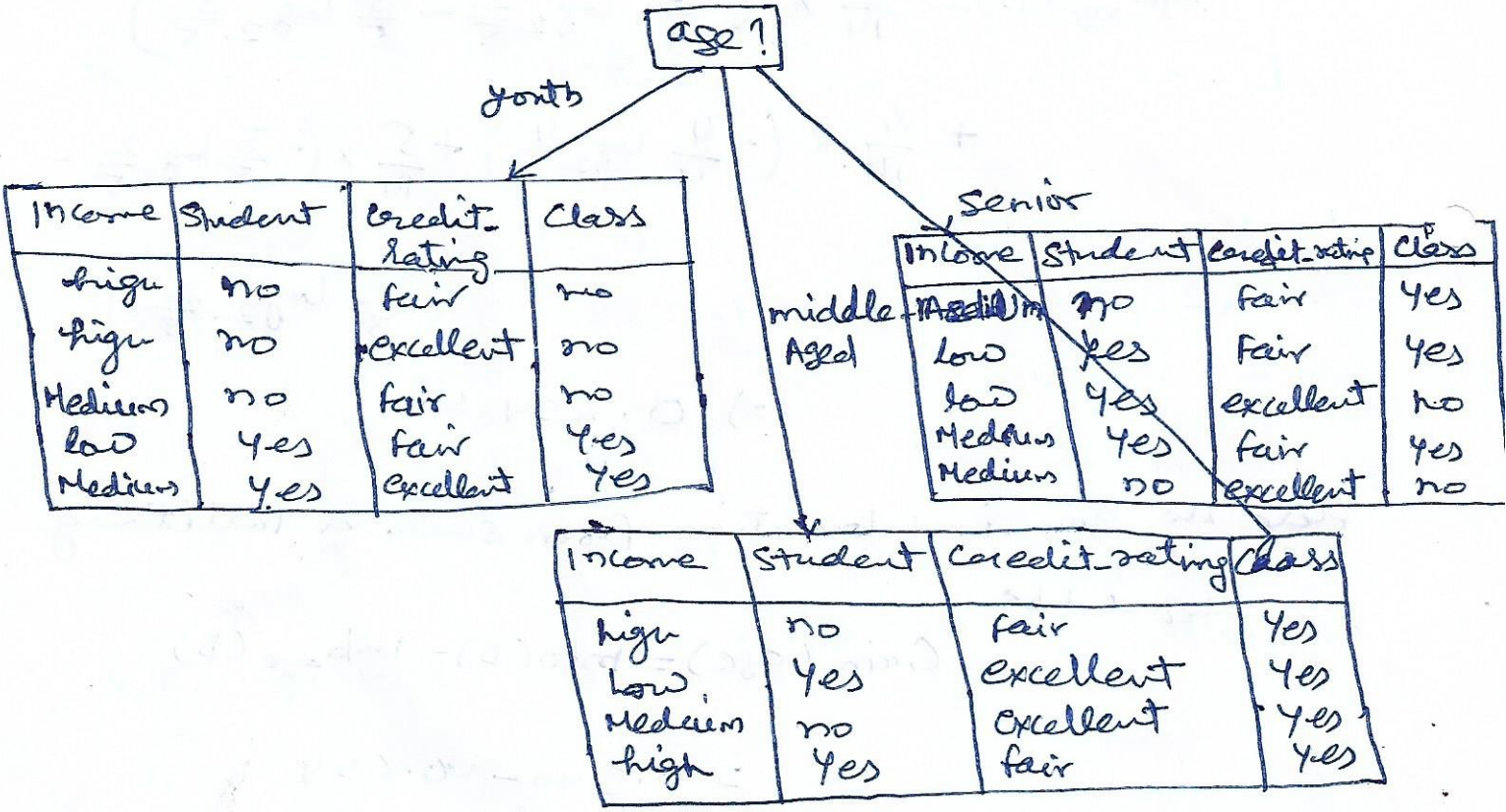
$$= 0.940 - 0.694$$

$$\Rightarrow 0.246 \text{ bits}$$

Similarly, we can compute Gain(Income) = 0.029 bits, Gain(Student) = 0.151 bits, and Gain(Credit_rating) = 0.048 bits. Because age has the highest information gain among the attributes, it is selected as the splitting attribute.

Node N is labeled with age, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as shown in figure (b).

Notice that the tuples falling into the Partitions for the AGE = middle_aged all belong to the same class. Because they all belong to class "yes" a leaf should ~~be~~ therefore be created at the end of this branch and labeled "yes". The final decision Tree returned by the algorithm was shown earlier in fig (a)



Income	Student	Credit_rating	Class
high	no	fair	no
high	no	excellent	no
Medium	no	fair	no
low	yes	fair	yes
Medium	yes	excellent	yes

Income	Student	Credit_rating	Class
high	no	fair	yes
low	yes	excellent	yes
Medium	no	excellent	yes
high	yes	fair	yes

Income	Student	Credit_rating	Class
Medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
Medium	yes	fair	yes
Medium	no	excellent	no

But how can we compute the information gain of an attribute that is continuous valued, unlike in the example? Suppose, instead, that we have an attribute A that is continuous-valued, rather than discreate valued. for such a scenario, we must determine the "best" split-point for A, where the split point is a threshold on A.

We first sort the values of A in increasing order. typically, the mid point ~~value~~ between each pair of adjacent values is considered as a possible split-point. Therefore, given v values of A, then v-1 possible split are evaluated. For Example, the midpoint between the value a_i and a_{i+1} of A is

$$\frac{a_i + a_{i+1}}{2}$$

If the value of A are sorted in advance, then determining the best split for A requires only one pass through the values. for each possible split for A, we evaluate $Info_A(D)$, where the number of partion is two, that is, $v = 2$ (or $j = 1, 2$) in eq — the point with the minimum expected information requirement for A is selected as the split-point for A. D_1 is the set of tuples in D satisfying $A \leq$ split-point, and D_2 is the set of tuples in D satisfying $A >$ split-point.

Important terminology related to Decision Tree

* How does a Tree decide where to split :- The decision ~~Tree~~ of making Strategic splits heavily affects a Tree's accuracy. The decision criterion is different for classification and Regression Trees.

* Information gain () :- The less impure node requires less information to describe it, and more impure node requires more information. Information Theory is measure to define this degree of disorganization in a system known as entropy. If the system is completely homogenous, then the entropy is zero and if the ~~system~~ sample is an equally divided (50% - 50%); it has entropy of one

Entropy can be calculated using formula :-

$$\text{Entropy} = -p \log_2 p - q \log_2 q$$

Here p and q are the probability of success and failure respectively in that node.

* Tree pruning :- when, we remove subnodes of a decision node. This process is called pruning. you can say opposite process of splitting

or

Pruning is a data compression technique in machine learning and search algorithms that reduces the size of decision Trees by removing sections of the Tree that are non-critical and redundant to classify instances. Pruning reduces the complexity of the final classifier, and hence improve the predictive accuracy by the reduction of overfitting. Here are two kinds of approaches

(i) Bottom-up ~~approach~~ Pruning :- These procedures start at the last node in the Tree (recursively upwards).

(ii) Top-down Pruning :- This method starts at the root of the Tree (following the structure below).