MACHINE LEARNING

UNIT-III

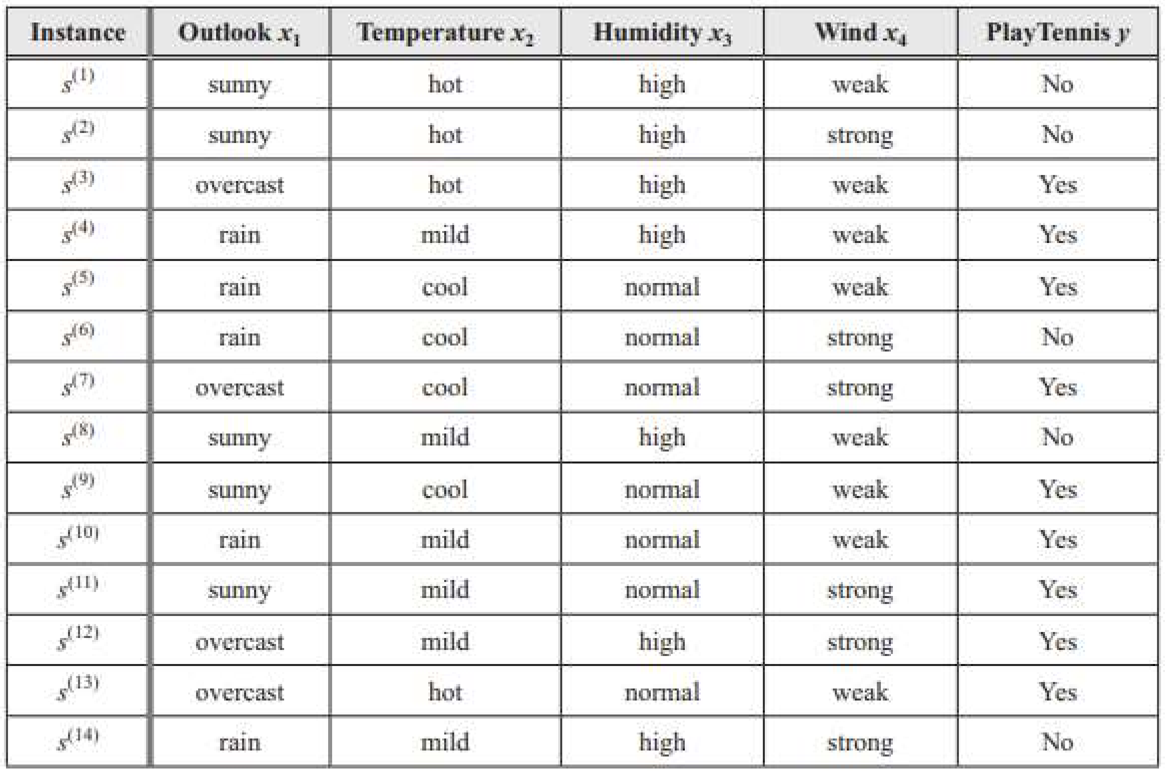
DECISION TREE LEARNING

5.1 INTRODUCTION

- Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.
- Learned trees can also be re-represented as sets of if-then rules to improve human readability. These learning methods are among the most popular of inductive inference algorithms and have been successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.
- A decision tree can be said to be a map of reasoning process. It uses a structure resembling that of a tree to describe a dataset and solutions can be visualized by following different pathways through the tree. It is a hierarchical set of rules explaining the way in which a large set of data can be divided into smaller data partitions.
- Each time a split takes place, the components of the resulting partitions become increasingly similar to one another with regard to the target. If we had to select a classification method capable of performing well across a wide range of situations without the analyst needing to put in effort, and easy for the customer to understand, the tree methodology would be the preferred choice. Several types of decision-free learning techniques are available with varying needs and abilities. Decision-tree learning is usually best suited to problems with the following features: • Patterns are described by a fixed set of attributes x_j; j = 1, 2, …, n, and each attribute x_j takes on a small number of disjoint possible values (categorical or numeric) $v_l x_j$; l = 1, 2, …, d_j. • The output variable y is a Boolean-valued function (binary classification problems) defined over the set S of patterns $\{s^{(i)}\} = \{x^{(i)}\}$; i = 1, 2, …, N. That is, y takes on values y_q ; q = 1, 2. If we assume y_1= 0 and y2=1, then y : S -> [0, 1].
- The training data is described by the dataset D of N patterns with corresponding observed outputs: D = $\{s^{(i)}, y^{(i)}\} = \{x^{(i)} y^{(i)}\}$; i = 1, 2, …,N.

5.2 EXAMPLE OF CLASSIFICATION DECISION TREE

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.
- Consider the weather data set:



Instance	Outlook x_1	Temperature x_2	Humidity x_3	Wind x_4	PlayTennis y
$s^{(1)}$	sunny	hot	high	weak	No
$s^{(2)}$	sunny	hot	high	strong	No
$s^{(3)}$	overcast	hot	high	weak	Yes
$s^{(4)}$	rain	mild	high	weak	Yes
$s^{(5)}$	rain	cool	normal	weak	Yes
$s^{(6)}$	rain	cool	normal	strong	No
$s^{(7)}$	overcast	cool	normal	strong	Yes
$s^{(8)}$	sunny	mild	high	weak	No
$s^{(9)}$	sunny	cool	normal	weak	Yes
$s^{(10)}$	rain	mild	normal	weak	Yes
$s^{(11)}$	sunny	mild	normal	strong	Yes
$s^{(12)}$	overcast	mild	high	strong	Yes
$s^{(13)}$	overcast	hot	normal	weak	Yes
$s^{(14)}$	rain	mild	high	strong	No

- Below figure illustrates a typical learned decision tree. This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis. For example, the instance

  (Outlook** = **Sunny, Temperature** = **Hot, Humidity** = **High, Wind** = **Strong)

  would be sorted down the leftmost branch of this decision tree and would therefore be classified as a negative instance (i.e., the tree predicts that ***PlayTennis** = **no).***

- In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions.
- For example, the decision tree shown in Figure **1** corresponds to the expression

(Outlook** = **Sunny Λ Humidity** = **Normal)

V ***(Outlook** = **Overcast)***

v ***(Outlook** = **Rain Λ Wind** = **Weak)***

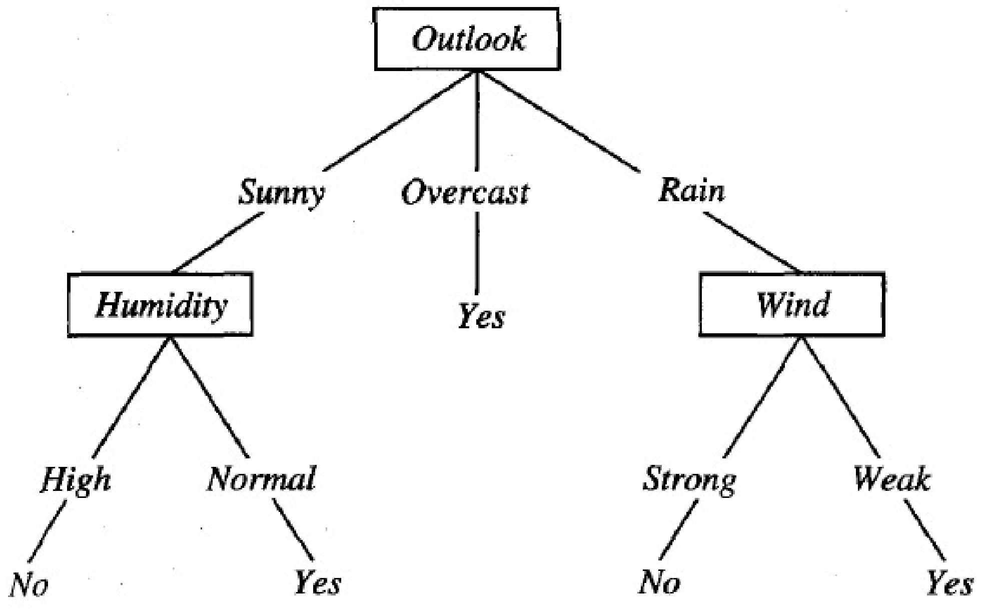


Fig: A decision tree for the concept *PlayTennis.*

3.3 MEASURES OF IMPURITY FOR EVALUATING SPLITS IN DECISION TREES

An impurity measure is a heuristic for selection of the splitting criterion that best separates a given dataset D of class-labeled training tuples into individual classes. If we divide D into smaller partitions as per the outcome of the splitting criterion, each partition should ideally be pure, with all the tuples falling into each partition belonging to the same class.
Three popular impurity measures are information gain, gain ratio, and Gini index.

5.3.1 Information Gain/Entropy reduction

Information theory has a measure for this, called entropy, which measures how disorganized a system is. The root node holds the entire dataset D which describes the patterns s $^{(1)}$, s $^{(2)}$, …, s $^{(N)}$ with their corresponding classes y_1 or y_2 (for a binary classification task). Imagine selecting a pattern at random from the dataset D and announcing that it belongs to class y_q. This message has the probability

$$P_q = \frac{freq\,(y_q, \mathcal{D})}{|\mathcal{D}|}$$

where freq(y_q, D) stands for the number of patterns in D that belong to class y_q and |D| denotes the total number of patterns in D (|D | = N). The expected information needed to classify a pattern in D is given by

$$Info\,(\mathcal{D}) = -\sum_{q=1}^{2} P_q \log_2(P_q)$$

A log function to the base 2 is used because information is encoded in bits. Info (D) is just the average amount of information needed to identify the class label of a pattern in D. Note that at this point, the information we have is solely based on the proportions of patterns in each class. Info (D) can also be expressed as entropy of D, denoted as Entropy (D).

$$Entropy\,(\mathcal{D}) = -\sum_{q=1}^{2} P_q \log_2 P_q$$

Associated with root node of the decision tree, Info(D) represents the expected amount of information that would be needed to specify whether a new instance should be classified as y_1 or y_2, given that the example reached the node. Info (D) is 0 if all patterns in D belong to the same class ($P_1 = 0, P_2 = 1$): $-P_1 \log_2 P_1 - P_2 \log_2 P_2 = 0$ (note that $0\log_2 0 = 0$). Info (D) is 1 when the collection D contains an equal number of Class 1 and Class 2 patterns ($P_1=1/2, P_2=1/2$), representing maximum heterogeneity (randomness) in the dataset: $-P_1 \log_2 P_1 - P_2 \log_2 P_2 = 1$. If the collection D contains unequal number of Class 1 and Class 2 patterns, Info (D) is between 0 and 1. It is, thus, a measure of impurity of the collection of examples.
To illustrate, we consider training set of Table 8.1 (Weather Data). It has nine examples of class Yes, and five examples of class No. Therefore,

$$Info(\mathcal{D}) = Entropy(\mathcal{D}) = -\tfrac{9}{14}\log_2\tfrac{9}{14} - \tfrac{5}{14}\log_2\tfrac{5}{14}$$

$$= 0.94 \text{ bits}$$

Root node with dataset D will therefore be a highly impure node. The training set D contains instances that belong to a mixture of classes (high entropy). In this situation, the idea of 'divide-and-conquer' strategy is to divide D into subsets of instances that are, or seem to be, heading towards single-class collection of instances.
Suppose we select attribute x_j for the root node. x_j has distinct values $v_l x_j$; $l = 1, \ldots, d_j$, as observed from the training data D. Attribute x_j can be used to split data D into l; $l = 1, \ldots, d_j$, partitions or subsets $\{D1, D2, \ldots, D_{d\,j}\}$, where D_l contains those patterns in D that have values $v_l x_j$ of x_j . These partitions would correspond to branches grown from the node. Ideally, we would like this partitioning to produce an exact classification, i.e., we would like each partition to be pure.
This amount is measured by

$$Info\,(\mathcal{D}, x_j) = \sum_{l=1}^{d_j} \frac{|\mathcal{D}_l|}{|\mathcal{D}|} \times Info(\mathcal{D}_l)$$

The term $|D_l|/|D|$ acts as the weight of l^{th} partition.

$Info(\mathcal{D}_l)$ is given by

$$Info\,(\mathcal{D}_l) = -\sum_{q=1}^{2} P_{ql} \log_2 P_{ql}$$

where P_{ql} is the probability that the arbitrary sample in subset D_l belongs to class y_q, and is estimated as

$$P_{ql} = \frac{freq\,(y_q, \mathcal{D}_l)}{|\mathcal{D}_l|}$$

Information gain is defined as the difference between the original information requirement (i.e, based on the partition of classes in the entire dataset D) and the new requirement (i.e., obtained after partitioning on xj). That is,

$$Gain(\mathcal{D}, x_j) = Info(\mathcal{D}) - Info(\mathcal{D}, x_j)$$

In other words, Gain (D, x_j) tells us how much would be gained by branching on x_j . It is the expected reduction in information requirement (expected reduction in entropy) by partitioning on x_j . The attribute xj with the highest information gain, Gain (D, x_j), is chosen as the splitting attribute at the root node. This is equivalent to saying that we want to partition on the attribute x_j that would do the best classification so that the amount of information still required (i.e., Info(D, x_j)) to finish classification task is minimal.
The information gain, Gain (D, x_j), measures the expected reduction in entropy, caused by partitioning the patterns in dataset D according to the attribute x_j .

$$\begin{aligned} Gain\ (\mathcal{D}, x_j) &= Entropy(\mathcal{D}) - Entropy\ (\mathcal{D}, x_j) \\ &= Entropy(\mathcal{D}) - \sum_{l=1}^{d_j} \frac{|\mathcal{D}_l|}{|\mathcal{D}|} \times Entropy(\mathcal{D}_l) \end{aligned}$$

where

$$Entropy(\mathcal{D}) = -\sum_{q=1}^{2} P_q \log_2 P_q ; P_q = \frac{freq\ (y_q, \mathcal{D})}{|\mathcal{D}|}$$

and

$$Entropy(\mathcal{D}_l) = -\sum_{q=1}^{2} P_{ql} \log_2 P_{ql} ; P_{ql} = \frac{freq\ (y_q, \mathcal{D}_l)}{|\mathcal{D}_l|}$$

5.3.2 Gain Ratio

It applies a kind of normalization to information gain using a ‘split information’ value defined analogously with Info(D, x_j) as

$$SplitInfo(\mathcal{D}, x_j) = -\sum_{l=1}^{d_j} \frac{|\mathcal{D}_l|}{|\mathcal{D}|} \log_2 \frac{|\mathcal{D}_l|}{|\mathcal{D}|}$$

This value is representative of the potential information derived from the division of the dataset, D, into d_j partitions matching with the dj values of the attribute x_j . For each value of x_j , the number of tuples possessing that value is considered with respect to the total number of tuples in D. This is different from information gain, which measures the information with respect to classification obtained on the basis of same partitioning. The gain ratio is defined as

$$GainRatio(\mathcal{D}, x_j) = \frac{Gain\ (\mathcal{D}, x_j)}{SplitInfo(\mathcal{D}, x_j)}$$

5.3.3 Gini Index

Gini index is used in CART.

$$Gini(\mathcal{D}) = 1 - \sum_{q=1}^{M} P_q^2$$

where P_q is the probability that a tuple in D belongs to class y_q, and is estimated by

$$P_q = \frac{freq(y_q, \mathcal{D})}{|\mathcal{D}|}$$

Gini index considers a binary split for each attribute. Let us first consider the case where xj is continuous-valued attribute having dj distinct values $v_l x_j$; l = 1, 2, …, d_j. It is common to take mid-point between each pair of (sorted) adjacent values as a possible split-point (It is a simple policy, although something might be gained by adoping a more sophisticated policy. One such policy will be discussed in the next section). The point giving the minimum Gini index for the attribute x_j is taken as its split-point

For a possible split-point of x_j , D_l is the number of tuples in D satisfying xj ≤ split-point, and D2 is the set of tuples satisfying xj > split-point. The reduction in impurity that would be incurred by a binary split on xj is

$$\Delta Gini(x_j) = Gini(\mathcal{D}) - Gini(\mathcal{D}, x_j)$$

$$Gini(\mathcal{D}, x_j) = \frac{|\mathcal{D}_1|}{|\mathcal{D}|} Gini(\mathcal{D}_1) + \frac{|\mathcal{D}_2|}{|\mathcal{D}|} Gini(\mathcal{D}_2)$$

The attribute that maximizes the reduction in impurity (or equivalently, has the minimum Gini index) is selected as the splitting attribute. Then one of these two parts (D_1, D_2) is divided in a similar manner by choosing a variable again and a split value for the variable. This process is continued till we get pure leaf nodes.

5.4 ID3, C4.5 AND CART DECISION TREES

The algorithms are clearly explained below

5.4.1.ID3 algorithm:

- Many algorithms have been developed for constructing the decision trees.
- The basic decision tree learning algorithm is ID3 which follows the top-down approach for constructing the decision tree.

5.4.2 Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- For this, ID3 algorithm uses information gain measure. The attribute with the highest information gain is chosen for testing at a node.
- ***Information gain*** measures how well a given attribute separates the training examples according to the target classification.
- In order to define information gain precisely, a measure called ***entropy*** is used.
- Entropy measures the impurity of an arbitrary collection of samples. (i.e. it measures the homogeneity of samples).

- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

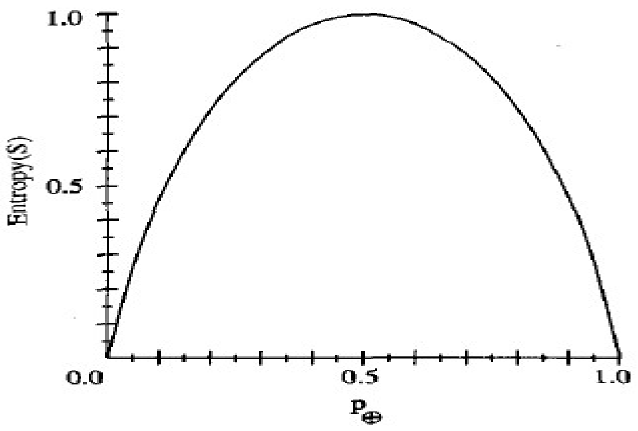
$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

where $p_{\oplus}$, is the proportion of positive examples in S and $p_{\ominus}$, is the proportion of negative examples in S.

- To illustrate, suppose S is a collection of 14 examples of some Boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this Boolean classification is

$$\begin{aligned} Entropy([9+, 5-]) &= -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) \\ &= 0.940 \end{aligned}$$

- Notice that the **entropy is 0** if all members of S belong to the same class. Also note that the **entropy is 1** when the collection contains an equal number of positive and negative examples.
- If the collection contains unequal numbers of positive and negative examples, the entropy is between **0** and 1. The following figure shows this.



- If the target classification has c classes, then the entropy of ***S*** relative to this c-wise classification is defined as

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

- where ***p***p_i is the proportion of ***S*** belonging to class ***i.***
- Having been defined entropy, now we can define the information gain,***Gain(S, A)*** of an attribute A, relative to a collection of samples ***S*** as

$$Gain(S, A) = Entropy(S) - Entropy_A(S)$$

Where $Entropy_A(S)$ is given as

$$\sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where ***Values(A)*** is the set of all possible values for attribute A, and ***S,*** is the subset of S for which attribute A has value v.

- As an example, let us construct the decision tree for the following data which shows training examples for the target concept ***PlayTennis***.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

First, compute ***Entropy(S)*** [S is the given data set. There are two classes yes = 9, No = 5].

$$\begin{aligned} Entropy([9+, 5-]) &= -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) \\ &= 0.940 \end{aligned}$$

Next, compute the information gain of each attribute in the data set.
$Entropy_{Outlook}(S)$ is given as

$$\sum_{v \in Values(Outlook)} \frac{|S_v|}{|S|} Entropy(S_v)$$

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

$$= 0.694$$

Therefore

$$Gain(S, Outlook) = \ Entropy(S) - Entropy_{Outlook}(S)$$

$$= \mathbf{0.940 - 0.694}$$
$$= \mathbf{0.246}$$

Similarly,
$Entropy_{Temperature}(S)$ is given as

$$\sum_{v \in Values(Temperature)}^{\square} \square \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= \frac{4}{14}\left(-\frac{2}{4}log\frac{2}{4} - \frac{2}{4}log\frac{2}{4}\right) + \frac{6}{14}\left(-\frac{4}{6}log\frac{4}{6} - \frac{2}{6}log\frac{2}{6}\right) + \frac{4}{14}\left(-\frac{3}{4}log\frac{3}{4} - \frac{1}{4}log\frac{1}{4}\right)$$

$$= 0.911$$

Therefore

$$Gain(S, Temperature) = \ Entropy(S) - Entropy_{Temperature}(S)$$

$$= \mathbf{0.940 - 0.911}$$
$$= \mathbf{0.029}$$

Similarly,
$Entropy_{Humidity}(S)$ is given as

$$\sum_{v \in Values(Humidity)}^{\square} \square \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= \frac{7}{14}\left(-\frac{3}{7}log\frac{3}{7} - \frac{4}{7}log\frac{4}{7}\right) + \frac{7}{14}\left(-\frac{6}{7}log\frac{6}{7} - \frac{1}{7}log\frac{1}{7}\right)$$

$$= 0.789$$

Therefore

$$Gain(S, Humidity) = \ Entropy(S) - Entropy_{Humidity}(S)$$

= 0.940 – 0.789
= 0.151

Similarly,
$Entropy_{Wind}(S)$ is given as

$$\sum_{v \in Values(Wind)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= \frac{8}{14}\left(-\frac{6}{8}log\frac{6}{8} - \frac{2}{8}log\frac{2}{8}\right) + \frac{6}{14}\left(-\frac{3}{6}log\frac{3}{6} - \frac{3}{6}log\frac{3}{6}\right)$$

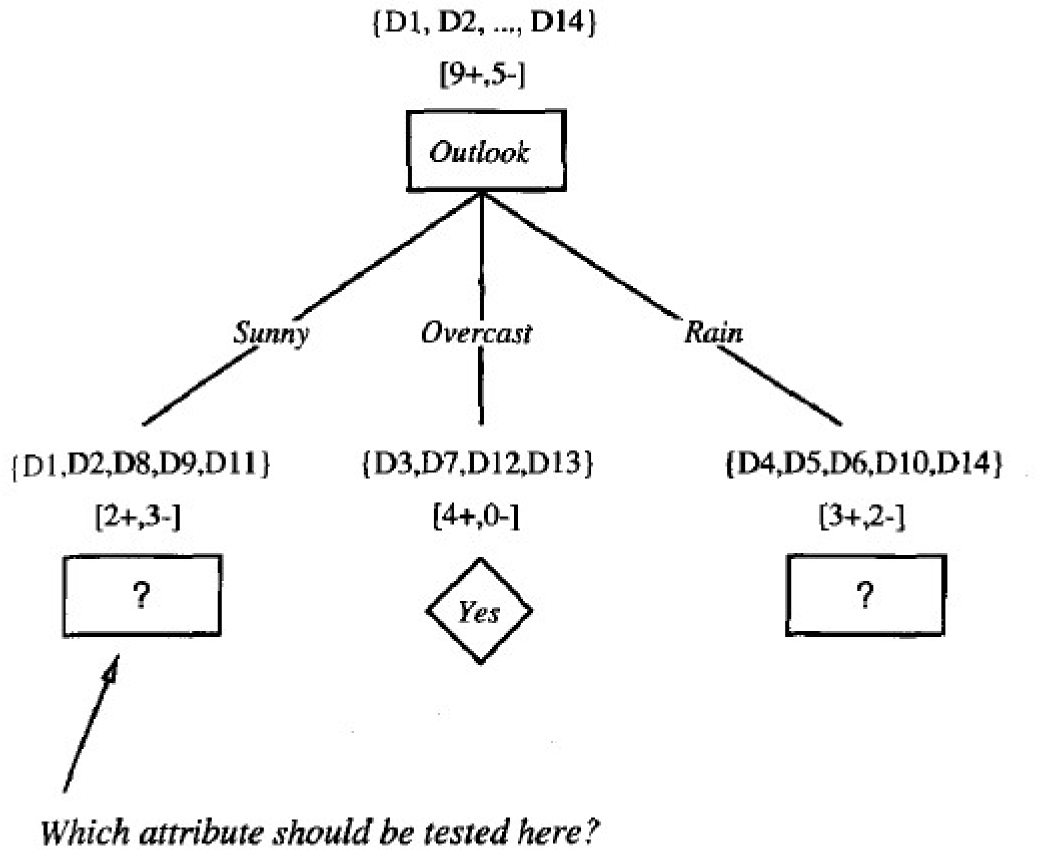
$$= 0.892$$

Therefore

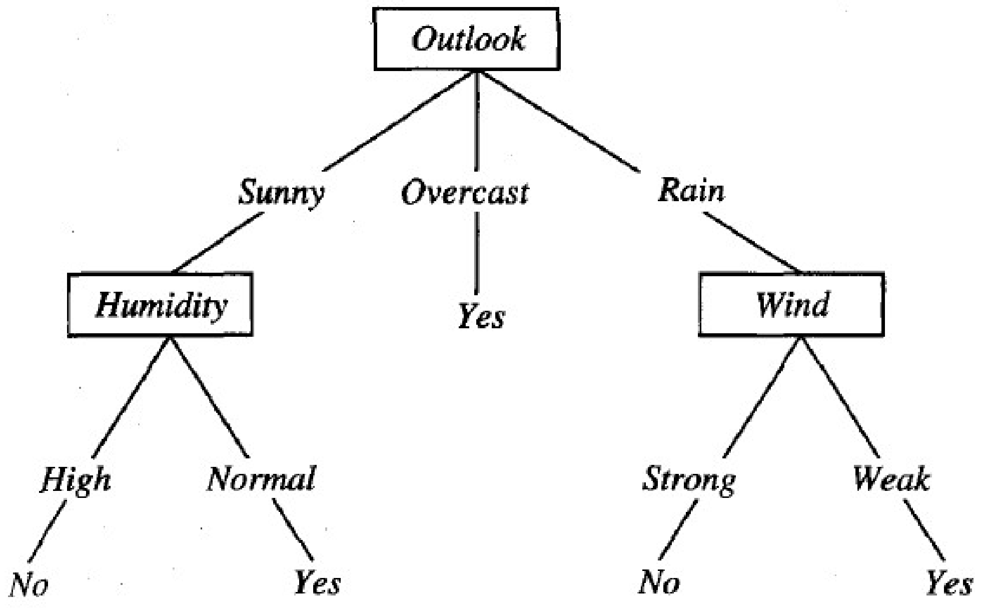
$$Gain(S, Wind) = Entropy(S) - Entropy_{Wind}(S)$$

= 0.940 – 0.892
= 0.048

Since, *Gain(S, Outlook)* is high so, the attribute *Outlook* is tested first and becomes the root for the decision tree. It then classifies S into three partitions [Let us say S1:leftmost subtree, S2:middle subtree, S3:rightmost subtree] (*because it has three different values: Sunny, Overcast, Rainy*) as shown below.



Now, we need to apply the same procedure to decide the root node for the leftmost subtree and for the rightmost subtree. However, observe that all the samples of middle tree are related to the same class (*Yes class*). So, we can create a leaf node here and label it with ***Yes.*** The final decision tree for the given data set, S, is given below.



After understanding this procedure, now we are ready to design the algorithm for the basic decision tree learning algorithm (ID3). This is given below.

ID3*((Examples, Targetattribute, Attributes)*

```
• Create a Root node for the tree
• If all Examples are positive, Return the single-node tree Root, with label = +
• If all Examples are negative, Return the single-node tree Root, with label = −
• If Attributes is empty, Return the single-node tree Root, with label = most common value of
  Target_attribute in Examples
• Otherwise Begin
    • $A \leftarrow$ the attribute from Attributes that best* classifies Examples
    • The decision attribute for Root $\leftarrow A$
    • For each possible value, $v_i$, of $A$,
        • Add a new tree branch below Root, corresponding to the test $A = v_i$
        • Let $Examples_{v_i}$ be the subset of Examples that have value $v_i$ for $A$
        • If $Examples_{v_i}$ is empty
            • Then below this new branch add a leaf node with label = most common
              value of Target_attribute in Examples
            • Else below this new branch add the subtree
                ID3($Examples_{v_i}$, Target_attribute, Attributes − {A}))
• End
• Return Root
```

5.4.2 The C4.5 Decision Tree

To illustrate the process of implementation of C4.5 decision tree, we consider the (toy) training set , in which there are four attributes: x_1 (Outlook), x_2 (Temperature), x_3 (Humidity), x_4 (Wind); and two classes for the output variable PlayTennis: Yes, No. The attributes x_1 and x_4 have categorical values, and the attributes x_2 and x_3 have continuous numeric values. Attribute x_1 (Outlook) has three categorical values: sunny, overcast and rain, and therefore the node labeled Outlook will have three branches. The node for categorical variable x_2 (Wind) will have two branches. The other two variables are continuous-valued, and as per the strategy followed in C4.5, the corresponding nodes will have binary splits. This will require discretization of these variables.

There are four choices of attribute at the root node: Outlook, Temperature, Humidity, and Wind. Tree stumps for the attribute Outlook (x_1) are shown above; a three-way split corresponding to the three categorical values of x_1.

Temperature (x_2) has continuous numeric values. For this attribute, we will select the best cut-point T_{x2} from its range of values by evaluating every candidate cut point. Examples are first sorted by increasing value of the attribute, and interval between each successive pair of values in the sorted sequence gives a potential cut-point. The cut-point that minimizes the entropy will never occur between two patterns of the same class. Therefore, it is necessary to consider potential divisions that separate patterns of different classes. For the weather data , this gives us eight potential cut-points: {64, 65, 70, 71, 72, 75, 80, 83}. Note that boundary points of the intervals between classes have been taken as the potential cut-points. Entropy for each of these cut-points is evaluated and the one that results in maximum information gain/gain ratio is selected as the split-value for the attribute x2. It follows that $x_2 = 83$ is the selected split-value

$$Entropy\ (\mathcal{D}) = 0.9402$$

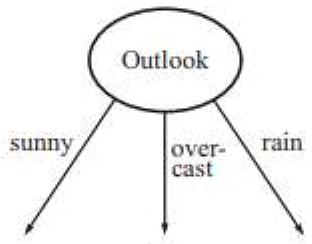
$$Entropy\ (\mathcal{D},\ T_{x_2} = 83) = 0.8268$$

$$Gain\ (\mathcal{D},\ T_{x_2} = 83) = 0.9402 - 0.8268 = 0.1134$$

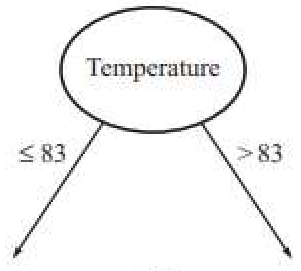
$$SplitInfo\ (\mathcal{D},\ T_{x_2} = 83) = 0.3711$$

$$GainRatio\ (\mathcal{D},\ T_{x_2} = 83) = \frac{0.1134}{0.3711} = 0.3053$$

Tree stump for x_2 (Temperature) is shown below. Evaluating all the four candidate root node variables, Outlook turns out to be the best choice with respect to entropy reduction/gain ratio measure.



(a)



(b)

Table shows the dataset for the branch sunny, obtained from the data D . Repeating the process described above on this dataset, we select Humidity as the daughter node with splitvalue = 70

$s^{(i)}$	Temperature	Humidity	Wind	Class
1	75	70	strong	Yes
2	80	90	strong	No
3	85	85	weak	No
4	72	95	weak	No
5	69	70	weak	Yes

The fully-grown C4.5 decision tree is shown below:

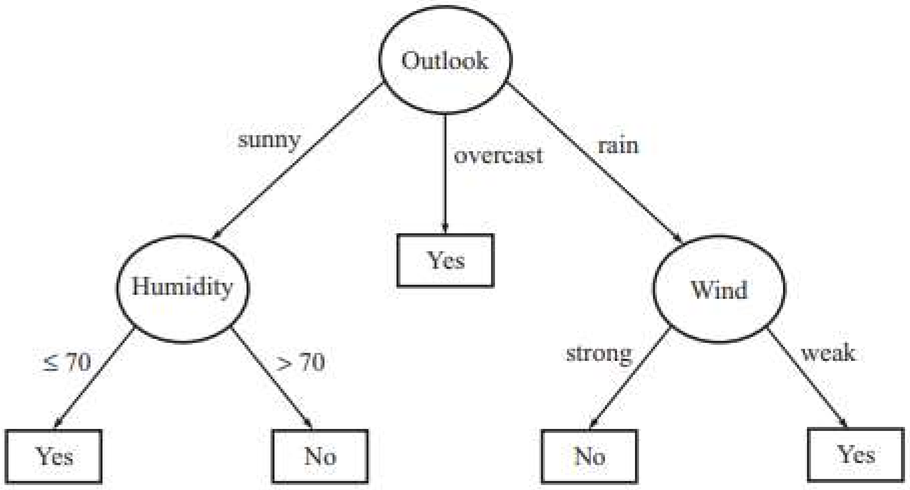


Fig: Decision tree for the weather data

5.4.3 CART Decision tree

Consider the data sample given below

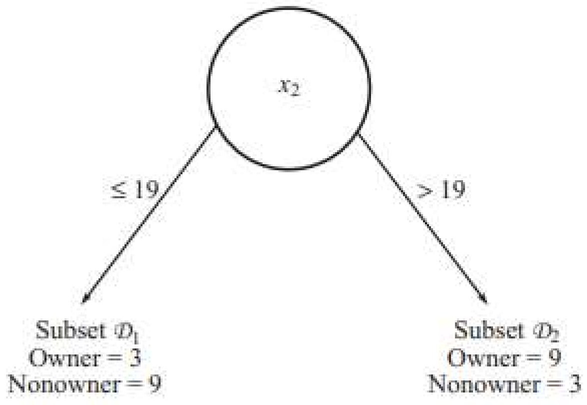
Household $s^{(i)}$	Income (\$ thousands) x_1	Lawn size (thousands ft^2)x_2	Ownership of a lawn tractor y
1	60	18.4	Owner
2	75	19.6	Nonowner
3	85.5	16.8	Owner

4	52.8	20.8	Nonowner
5	64.8	21.6	Owner
6	64.8	17.2	Nonowner
7	61.5	20.8	Owner
8	43.2	20.4	Nonowner
9	87	23.6	Owner
10	84	17.6	Nonowner
11	110.1	19.2	Owner
12	49.2	17.6	Nonowner
13	108	17.6	Owner
14	59.4	16	Nonowner
15	82.8	22.4	Owner
16	66	18.4	Nonowner
17	69	20	Owner
18	47.4	16.4	Nonowner
19	93	20.8	Owner
20	33	18.8	Nonowner
21	51	22	Owner
22	51	14	Nonowner
23	81	20	Owner
24	63	14.8	Nonowner

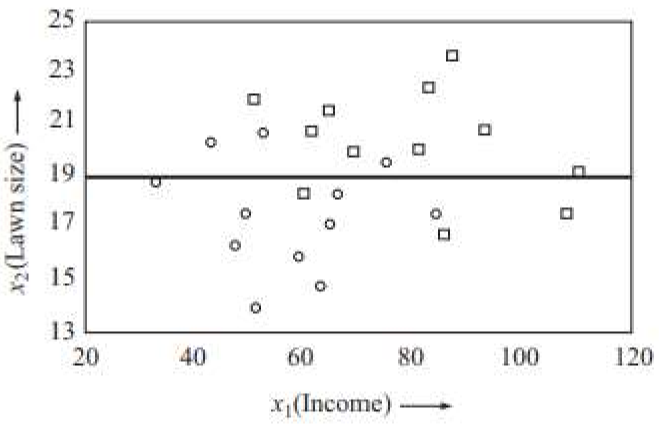
- There are N = 24 points in two dimensions given by the data sample; a pilot random sample of households in a city with respect to ownership of a lawn tractor.
- The dataset is set up for a predictive model—a lawn-tractor manufacturer would like to find a way of classifying households in a city into those likely to purchase a lawn tractor and those not likely to buy one. Input variables, x1 = Income and x2 = Lawn size, are recorded for 24 households, and the target variable y = Ownership of a lawn tractor, is assigned to each household.
- The dataset is balanced, containing equal numbers of Owner/Nonowner households. When searching for a binary split on a continuous-valued input variable, midpoints between the consecutive values may be treated as candidate values for the split. The candidate split points for the variable x_1 (Income) are {38.1, 45.3, 50.1, …, 109.5}, and those for x_2 (Lawn size) are {14.4, 15.4, 16.2, …, 23}. We need to rank the candidate split points according to how much they reduce impurity (heterogeneity) in the resulting subsets after the split. Note that the total dataset D given in above table has the highest impurity. With respect to Gini index as impurity measure,

$$Gini(\mathcal{D}) = 1 - \sum_{q=1}^{2} P_q^2; \; P_q = \frac{freq(y_q, \mathcal{D})}{|\mathcal{D}|}$$
$$= 1 - (0.5)^2 - (0.5)^2$$
$$= 0.5$$

- It can easily be verified that the Gini index impurity measure is at its peak when Pq = 0.5, i.e., when the data is perfectly balanced. Calculating Gini index for all the candidate split points for both x_1 and x_2 variables, and ranking them according to how much they reduce impurity, we choose x_2 (Lawn size) for the first split with a splitting value of 19. The (x_1, x_2) space is now divided into two rectangles, one with x2 ≤ 19 and the other with x2 > 19. This is illustrated as



(a) Tree stumps after the first split



(b) Scatter plot after the first split

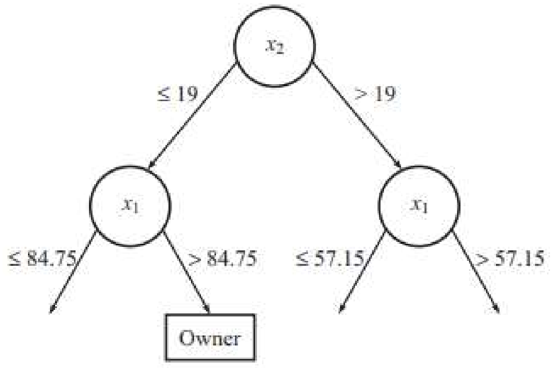
$$Gini(\mathcal{D}, x_2) = \frac{|\mathcal{D}_1|}{|\mathcal{D}|} \times Gini(\mathcal{D}_1) + \frac{|\mathcal{D}_2|}{|\mathcal{D}|} \times Gini(\mathcal{D}_2)$$

$$= \tfrac{12}{24} \times Gini(\mathcal{D}_1) + \tfrac{12}{24} \times Gini(\mathcal{D}_2)$$

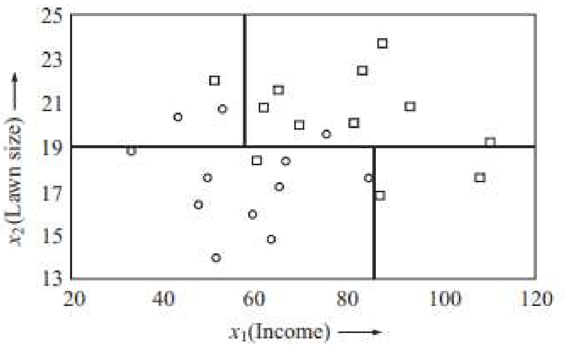
$$= \tfrac{12}{24}\left(1-\left(\tfrac{3}{12}\right)^2-\left(\tfrac{9}{12}\right)^2\right)+\tfrac{12}{24}\left(1-\left(\tfrac{9}{12}\right)^2-\left(\tfrac{3}{12}\right)^2\right)$$

$$= \tfrac{1}{2} \times 0.375 + \tfrac{1}{2} \times 0.375 = 0.375$$

- Thus, the Gini impurity index decreased from 0.5 before the split to 0.375 after the split. Each of the rectangles created by the split is more homogeneous than the rectangle before the split. The upper rectangle contains points that are mostly Owners and the lower rectangle contains mostly Nonowners. By comparing the reduction in impurity across all possible splits in all possible attributes, the next split is chosen. The next split is found to be on the x_1 (Income) variable at the value 84.75. Below figure shows that once again the tree procedure has actually chosen to split a rectangle to increase the purity of the resulting rectangles. The left lower rectangle ($x1 \leq 84.75$, $x2 \leq 19$) has all points that are Nonowners with one exception, whereas the right lower rectangle ($x1 > 84.75$, $x2 \leq 19$) consists exclusively of Owners.



(a) Tree stumps after first three splits



(b) Scatter plot after first three splits

If we continue partitioning till all the branches hit leaf nodes, each rectangle will have data points from just one of the two classes.

5.5 PRUNING THE TREE

- 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.
- The fundamental algorithm for decision trees continues to grow the tree by splitting nodes as long as new divisions generate daughter nodes that increase purity. Such a tree has undergone optimization for the training set. Therefore, elimination of any leaf nodes will simply lead to an increase in the error rate of the tree on the training set. But this certainly does not mean that the entire tree with pure leaf nodes also performs the best on new data!
- Pruning is the procedure that decreases the size of decision trees. It can decrease the risk of overfitting by defining the size of the tree or eliminating areas of the tree that support little power. Pruning supports by trimming the branches that follow anomalies in the training information because of noise or outliers and supports the original tree in a method that enhances the generalization efficiency of the tree,
- Various methods generally use statistical measures to delete the least reliable departments, frequently resulting in quicker classification and an improvement in the capability of the tree to properly classify independent test data.
- There are two approaches to tree pruning which are as follows −

5.5.1 Pre-pruning Approach

- In the pre-pruning approach, a tree is “pruned” by labored its construction early (e.g., by determining not to further divide or partition the subset of training samples at a provided node).
- Upon halting, the node turns into a leaf. The leaf can influence the most common class between the subset samples, or the probability distribution of those samples.
- When making a tree, measures including statistical significance, information gain, etc., can be used to create the generosity of a split. If partitioning the samples at a node can result in a split that declines below a pre-specified threshold, then partitioning of the given subset is halted. There are problems in selecting an appropriate threshold. High thresholds can result in oversimplified trees, while low thresholds can result in very little simplification.

5.5.2 Post-pruning Approach

The post-pruning approach eliminates branches from a “completely grown” tree.

- A tree node is pruned by eliminating its branches. The price complexity pruning algorithm is an instance of the post-pruning approach. The pruned node turns into a leaf and is labeled by the most common class between its previous branches.

- For each non-leaf node in the tree, the algorithm computes the expected error rate that can appear if the subtree at that node were shortened. Next, the expected error rate appearing if the node were not pruned is computed using the error rates for each branch, connected by weighting according to the dimension of observations along each branch. If pruning the node leads to a higher expected error rate, then the subtree is preserved. Therefore, it is pruned.
- After creating a set of increasingly pruned trees, an independent test set can estimate the efficiency of each tree. The decision tree that diminishes the expected error cost is preferred.

5.6 METRICS FOR ASSESSING CLASSIFICATION ACCURACY

- Evaluation metrics are tied to machine learning tasks. There are different metrics for the tasks of classification and regression. Some metrics, like precision-recall, are useful for multiple tasks. Classification and regression are examples of supervised learning, which constitutes a majority of machine learning applications.
- Using different metrics for performance evaluation, we should be able to improve our model’s overall predictive power before we roll it out for production on unseen data. Without doing a proper evaluation of the Machine Learning model by using different evaluation metrics, and only depending on accuracy, can lead to a problem when the respective model is deployed on unseen data and may end in poor predictions.
- There are many ways for measuring classification performance. Accuracy, confusion matrix, log-loss, and AUC-ROC are some of the most popular metrics. Precision-recall is a widely used metrics for classification problems.

i. Accuracy

Accuracy simply measures how often the classifier correctly predicts. We can define accuracy as the ratio of the number of correct predictions and the total number of predictions.

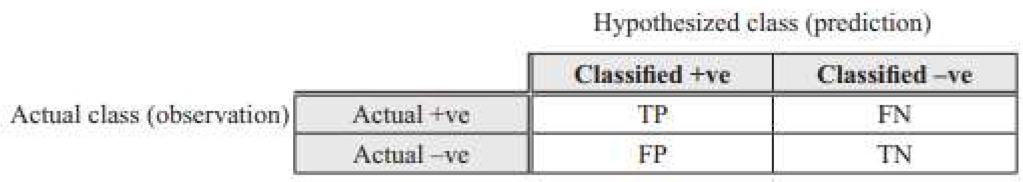
Accuracy=(TP+TN)/(TP+TN+FP+FN)

Where TP=True Positive, TN=True Negative, FP=False Positive, FN=False Negative

ii. Confusion Matrix

Confusion Matrix is a performance measurement for the machine learning classification problems where the output can be two or more classes. It is a table with combinations of predicted and actual values.

A confusion matrix is defined as thetable that is often used to describe the performance of a classification model on a set of the test data for which the true values are known.



		Hypothesized class (prediction)	
		Classified +ve	Classified –ve
Actual class (observation)	Actual +ve	TP	FN
	Actual –ve	FP	TN

Fig: Confusion matrix

$(+ve, +ve) \rightarrow TP$

$(-ve, +ve) \rightarrow FP$

$(+ve, -ve) \rightarrow FN$

$(-ve, -ve) \rightarrow TN$

iii. Precision

Precision explains how many of the correctly predicted cases actually turned out to be positive. Precision is useful in the cases where False Positive is a higher concern than False Negatives.

“Precision for a label is defined as the number of true positives divided by the number of predicted positives”.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

iv. Recall (Sensitivity)—Recall explains how many of the actual positive cases we were able to predict correctly with our model. It is a useful metric in cases where False Negative is of higher concern than False Positive.

Recall for a label is defined as the number of true positives divided by the total number of actual positives.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

v. F1 Score

It gives a combined idea about Precision and Recall metrics. It is maximum when Precision is equal to Recall.

F1 Score is the harmonic mean of precision and recall.

$$F1 = 2.\ \frac{Precision\ \times Recall}{Precision + Recall}$$

The F1 score punishes extreme values more. F1 Score could be an effective evaluation metric in the following cases:

- When FP and FN are equally costly.
- Adding more data doesn’t effectively change the outcome
- True Negative is high