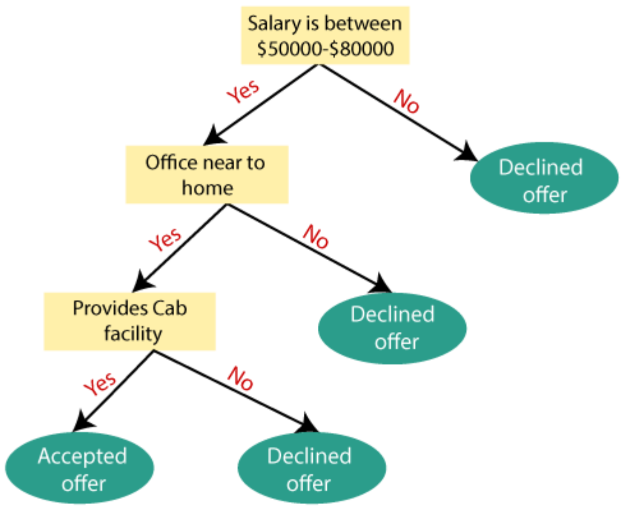
Module – II
Decision Tree Learning

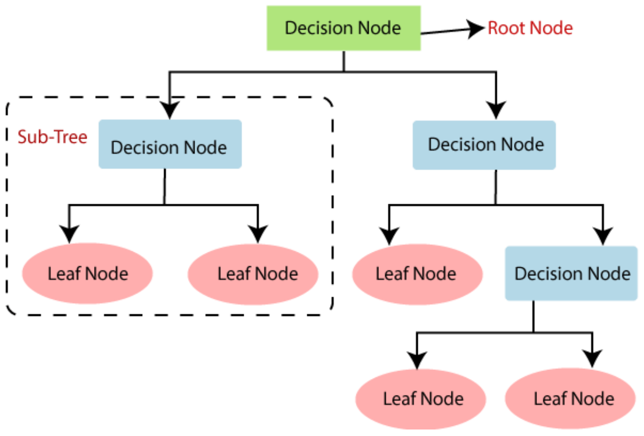
- Decision Tree Representation
- Appropriate Problems for Decision Tree Learning
- Basic Decision Tree Learning Algorithm
- Problems based on ID3 algorithm
- Issues in Decision Tree Learning

Decision Tree Representation

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. The process is then repeated for the subtree rooted at the new node.



Decision Tree Representation



What is a Decision Tree?

It is a tree in which each branch node represents a choice between a number of alternatives, and each leaf node represents a decision. It is a type of supervised learning algorithm. A decision tree is a tree where:

- Each node represents a feature (Attribute)
- Each branch represents a decision (Rule)
- Each leaf represents an outcome.

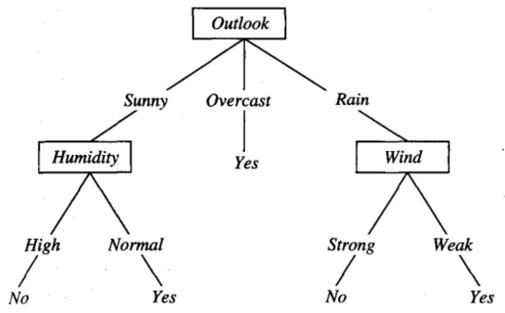
Decision Tree Components

Root node: It refers to the start of the decision tree with maximum split (information gain)

Node: Node is a condition with multiple outcomes in the tree

Leaf: This is the final decision (end point) of a node from the condition (question)/

In general, decision tree represents a disjunction of conjunction 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.



Expression: (Outlook=Sunny ^ Humidity=Normal) ˅ (Outlook=Overcast) ˅ (Outlook=Rain ^ Wind=Weak)

Appropriate Problems for Decision Tree Learning

Decision Tree Learning is generally best suited to problems with the following characteristics:

1. Instances are represented by attribute-value pairs: Instances are described by a fixed set of attributes (Ex. Temperature) and their values (ex. Hot). The easiest situation for decision tree learning is when each attribute takes on a small number of disjoint possible values (Ex. Hot, mild, cold).
2. The target function has discrete output values: The decision tree assigns a boolean classification (Ex. Yes or no) to each example. Decision tree methods easily extend to learning functions with more than two possible output values.
3. Disjunctive descriptions may be required: As noted above, decision trees naturally represent disjunctive expressions.
4. The training data may contain errors: Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.
5. The training data may contain missing attribute values: Decision tree methods can be used when some training examples have unknown values (Ex. If the Humidity of the day is known for only some of the training examples).

Decision tree learning has been applied to problems such as learning to classify medical patients by their disease, equipment malfunctions by their cause, and loan applicants by their likelihood of defaulting on payments. Such problems, in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as classification problems.

The Basic Decision Tree Algorithm

The widely decision tree algorithms are:

1. ID3 – Iterative Dichotomiser3
2. C4.5 – Successor of ID3
3. CART – Classification and Regression Tree

ID3: Learns the decision trees by constructing them top-down, beginning with the question, "which attribute should be tested at the root of the tree"?. To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree. A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute). The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree. This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices.

ID3 Algorithm

ID3 (Examples, Target_attribute, Attributes)

Examples are the training examples

Target attribute – is the attribute whose value is to be predicted by the tree

Attributes – is a list of other attributes that may be tested by the learned decision tree.

The algorithm returns a decision tree that correctly classifies the given examples.

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. We would like to select the attribute that is most useful for classifying examples. What is a good quantitative measure of the worth of an attribute? We will define a statistical property, called **information gain**, that measures how well a given attribute separates the training examples according to their target classification. ID3 uses this information gain measure to select among the candidate attributes at each step while growing the tree.

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

In order to define information gain precisely, we begin by defining a measure commonly used in information theory, called **entropy,** that characterizes the (im)purity of an arbitrary collection of examples. 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⊕, is the proportion of positive examples in S and pθ, is the proportion of negative examples in S. In all calculations involving entropy we define 0 log 0 to be 0.

```
ID3(Examples, Target_attribute, Attributes)
    Examples are the training examples. Target_attribute is the attribute whose value is to be
    predicted by the tree. Attributes is a list of other attributes that may be tested by the learned
    decision tree. Returns a decision tree that correctly classifies the given Examples.
• 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 ← the attribute from Attributes that best* classifies Examples
    • The decision attribute for Root ← 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
```

[Problems solved in class]

Issues in decision tree learning

1. Avoiding overfitting the data
2. Incorporating continuous-valued data
3. Alternative measures for selecting attribute
4. Handling training examples with missing attribute values
5. Handling attributes with differing costs

Avoiding Overfitting the data

What is Overfitting? – Definition – Given a hypothesis space H, a hypothesis h ϵ H is said to overfit the training data if there exists some alternative hypothesis h' ϵ H, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances. In other words,

Training error (h) < training error (h')

Testing error (h) > Testing error (h')

The hypotheses h is performing better over the training set than the hypothesis h'. But h is not performing well over the testing data set. The model has learnt more than required because of which it is wrongly predicting.

Overfitting with respect to ID3

ID3 algorithm grows each branch of the tree deeply enough to perfectly classify the training examples. It may create overfitting when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. Overfitting happens when the learning algorithm continues to develop hypotheses that reduce error on training set whereas increase the error on testing set.

Training and validation set approach – Available data are separated into two sets: Training set and validation set. Training set is used to form the learned hypothesis. Validation set is used to evaluate the accuracy of this hypothesis over subsequent data and in particular, to evaluate the impact of pruning this hypothesis. Validation set provides a safety check against overfitting of the training set. Validation set should be large enough to provide a statistically significant sample of the instances. Common heuristic is one-third of the available examples for validation set, other two-thirds for training.

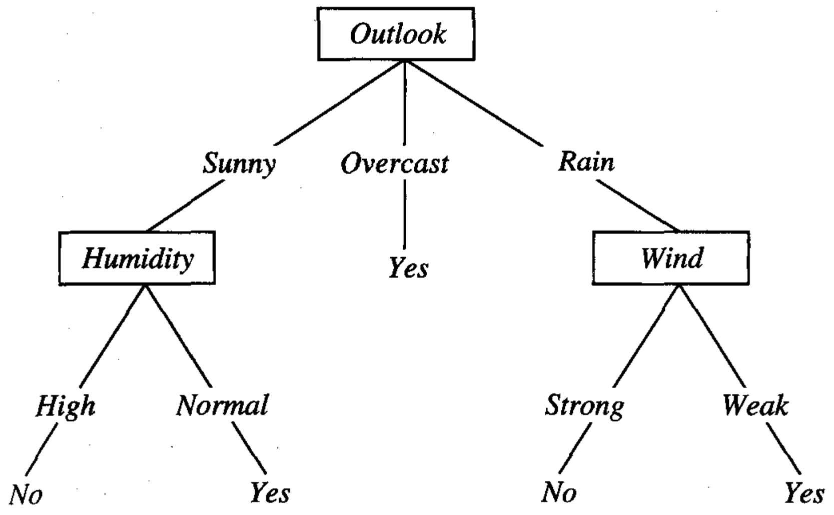
Reduced Error Pruning

Considers each of the decision nodes in the tree to be candidate for pruning. Pruning a decision node consists of removing the subtree rooted at the node, making it a leaf node, and assigning it the most common classification of the training examples. Nodes are removed only if the resulting pruned tree performs better than the original tree for the validation set. Pruning of nodes continues until further pruning is harmful i.e., it decreases the accuracy of the tree over the validation set. Using a separate set of data to guide pruning is an effective approach provided a large amount of data is available. The major drawback of this approach is when data is limited, withholding part of it for the validation set reduces even further the number of examples available for training.

Rule Post-pruning

- Infer the decision tree from the training set, growing the tree until the training data is fit and allowing overfitting to occur.
- Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.
- Prune each rule by removing any preconditions so that accuracy is improved.
- Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

Consider the following decision tree:



In rule post pruning, one rule is generated for each leaf node in the tree. Each attribute test along the path from the root to the leaf becomes a rule antecedent (precondition) and the classification at the leaf node becomes the rule consequent (postcondition). For example, the leftmost path of the tree in the above figure is translated into the rule:

IF $(Outlook = Sunny) \wedge (Humidity = High)$
THEN $PlayTennis = No$

Next, each such rule is pruned by removing any antecedent, or precondition, whose removal does not worsen its estimated accuracy. Given the above rule, for example, rule post-pruning would consider removing the preconditions (Outlook = Sunny) and (Humidity = High). It would select whichever of these pruning steps produced the greatest improvement in estimated rule accuracy, then consider pruning the second precondition as a further pruning step. No pruning step is performed if it reduces the estimated rule accuracy.

Incorporating continuous-valued attributes

ID3 is restricted to attributes that take on a discrete set of values. First, the target attribute whose value is predicted by the learned tree must be discrete valued. Second, the attributes tested in the decision nodes of the tree must also be discrete valued. This second restriction can easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree. This can be accomplished by dynamically defining new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals. For an attribute A that is continuous-valued, the algorithm can dynamically create a new boolean attribute A, that is true if A < c and false otherwise. The only question is how to select the best value for the threshold c.

As an example, suppose we wish to include the continuous-valued attribute *Temperature.*

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

What threshold-based boolean attribute should be defined based on Temperature? Clearly, we would like to pick a threshold, c, that produces the greatest information gain. By sorting the examples according to the continuous attribute A, then identifying adjacent examples that differ in their target classification, we can generate a set of candidate thresholds midway between the corresponding values of A. These candidate thresholds can then be evaluated by computing the information gain associated with each.

In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes: (48 + 60)/2, and (80 + 90)/2. The information gain can then be computed for each of the candidate attributes, Temperature > 54, and Temperature > 85, and the best can be selected (Temperature > 54). This dynamically created boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

Alternative measures for selecting attributes

There is a natural bias in the information gain measure that favors attributes with many values over those with few values. As an extreme example, consider the attribute Date, which has a very large number of possible values (e.g., March 4, 1979). If we were to add this attribute to the data set, it would have the highest information gain of any of the attributes. This is because Date alone perfectly predicts the target attribute over the training data. Thus, it would be selected as the decision attribute for the root node of the tree and lead to a (quite broad) tree of depth one, which perfectly classifies the training data. Of course, this decision tree would fare poorly on subsequent examples, because it is not a useful predictor despite the fact that it perfectly separates the training data.

One way to avoid this difficulty is to select decision attributes based on some measure other than information gain. One alternative measure that has been used successfully is the **gain ratio**. The gain ratio measure penalizes attributes such as Date by incorporating a term, called **split information**, that is sensitive to how broadly and uniformly the attribute splits the data:

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

where S1 through S, are the c subsets of examples resulting from partitioning S by the c-valued attribute A.

The **Gain Ratio** measure is defined in terms of the earlier Gain measure, as well as this Split Information, as follows:

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

If **Split Information** is high, then uncertainty is high. This will reduce **GainRatio** so that attribute will not be selected as root node for splitting.

Handling Training Examples with Missing Attribute Values

In certain cases, the available data may be missing values for some attributes. For example, in a medical domain in which we wish to predict patient outcome based on various laboratory tests, it may be that the lab test Blood-Test-Result is available only for a subset of the patients. In such cases, it is common to estimate the missing attribute value based on other examples for which this attribute has a known value.

Consider the situation in which **Gain(S, A)** is to be calculated at node n in the decision tree to evaluate whether the attribute A is the best attribute to test at this decision node. Suppose that (x, c(x)) is one of the training examples in S and that the value A(x) is unknown. One strategy for dealing with the missing attribute value is to assign it the value that is most common among training examples at node n. Alternatively, we might assign it the most common value among examples at node n that have the classification c(x). The elaborated training example using this estimated value for A(x) can then be used directly by the existing decision tree learning algorithm.

Handling Attributes with Differing Costs

In some learning tasks the instance attributes may have associated costs. For example, in learning to classify medical diseases we might describe patients in terms of attributes such as Temperature, BiopsyResult, Pulse, BloodTestResults, etc. These attributes vary significantly in their costs. For instance, all the records will have Temperature and Pulse values. But not all records will have BiopsyResult and BloodTestResult values. Therefore we assign a low-cost for Temperature and Pulse but a high cost for BiopsyResult and BloodTestResult. High-cost attributes will have missing values. In such tasks, we would prefer decision trees that use low-cost attributes where possible, relying on high-cost attributes only when needed to produce reliable classifications.

ID3 can be modified to consider attribute costs by introducing a cost term into the attribute selection measure:

$$\frac{Gain^2(S, A)}{Cost(A)}$$

Such cost-sensitive measures do not guarantee finding an optimal cost-sensitive decision tree, they do bias the search in favor of low-cost attributes.