Module 2

DECISION TREE LEARNING

3.1 INTRODUCTION

- Decision tree learning is one of the most widely used and practical methods for inductive inference.
- 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.

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

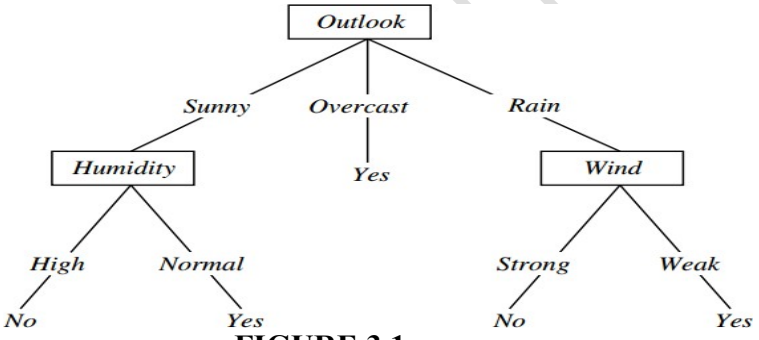


FIGURE 3.1

A decision tree for the concept *PlayTennis.*

An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf (in this case, *Yes* or *No).*

This tree classifies Saturday mornings according to whether or not they are suitable for playing tennis. For example, the instance would be sorted down the left-most branch of this decision tree and would therefore be classified as a negative instance (i.e., the tree predicts that ***PlayTennis*** = ***no).***

(Outlook = ***Sunny, Temperature*** = ***Hot, Humidity*** = ***High, Wind*** = ***Strong)*** 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 3.1 corresponds to the expression

$$(Outlook = Sunny \wedge Humidity = Normal)$$
$$\vee \quad (Outlook = Overcast)$$
$$\vee \quad (Outlook = Rain \wedge Wind = Weak)$$

3.3 APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

Decision tree learning is generally best suited to problems with the following characteristics:

- ***Instances are represented by attribute-value pairs.*** Instances are described by a fixed set of attributes (e.g., ***Temperature)*** and their values (e.g., ***Hot).*** The easiest situation for decision tree learning is when each attribute takes on a small number of disjoint possible values (e.g., ***Hot, Mild, and Cold). Decision tree a***llow handling real-valued attributes as well (e.g., representing ***Temperature*** numerically).
- ***The target function has discrete output values.*** The decision assigns a boolean classification (e.g., ***yes*** or ***no)*** to each example. Decision tree methods easily extend to learning functions with more than two possible output values. A more substantial extension allows learning target functions with

real-valued outputs, though the application of decision trees in this setting is less common.

- ***Disjunctive descriptions may be required.*** As noted above, decision trees naturally represent disjunctive expressions.
- ***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.
- ***The training data may contain missing attribute values.*** Decision tree methods can be used even when some training examples have unknown values (e.g., if the ***Humidity*** of the day is known for only some of the training examples).

3.3 THE BASIC DECISION TREE LEARNING ALGORITHM

- Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees.
- The basic algorithm for decision tree learning, corresponding approximately to the ID3 algorithm.

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.
- 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 the proportion of positive examples in S and, is the proportion of negative examples in S.
- In all calculations involving entropy we define 0 log 0 to be 0.

ID3 ALGORITHM

Summary of the ***ID3*** algorithm specialized to learning boolean-valued functions. ***ID3*** is a greedy algorithm that grows the tree top-down, at each node selecting the attribute that best classifies the local training examples. This process continues until the tree perfectly classifies the training examples, or until all attributes have been used.

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*

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

$$Entropy([9+, 5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$$
$$= 0.940$$

- Notice that the entropy is 0 if all members of S belong to the

same class. For example, if all members are positive ($p_{\oplus} = 1$), then p is 0, and Entropy(S)=-1.$\log_2$1 -0.$\log_2$0= -1.0-0.$\log_2$ 0 = 0.

- Note 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. Below figure shows the form of the entropy function relative to a Boolean classification, as **,** varies between **0** and 1.

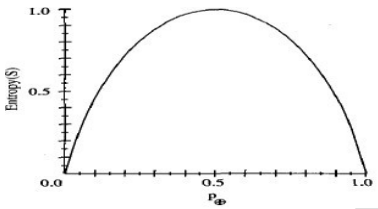


FIGURE 3.2
The entropy function relative to a boolean classification, as the proportion, $p_{\oplus}$, of positive examples varies between 0 and 1.

Information Gain

- Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data.
- The measure we will use, called ***information gain,*** is simply the expected reduction in entropy caused by partitioning the examples according to this attribute.
- More precisely, the information gain, ***Gain(S, A)*** of ***an*** attribute **A,** relative to a collection of examples ***S,*** is defined as

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

- For example, suppose ***S*** is a collection of training-example days described by attributes including ***Wind,*** which can have the values ***Weak*** or ***Strong.***
- As before, assume ***S*** is a collection containing ***14*** examples, [9+, 5-]. Of these 14 examples, suppose 6 of the positive and 2 of the negative examples have ***Wind*** = ***Weak,*** and the

remainder have ***Wind*** = ***Strong.***

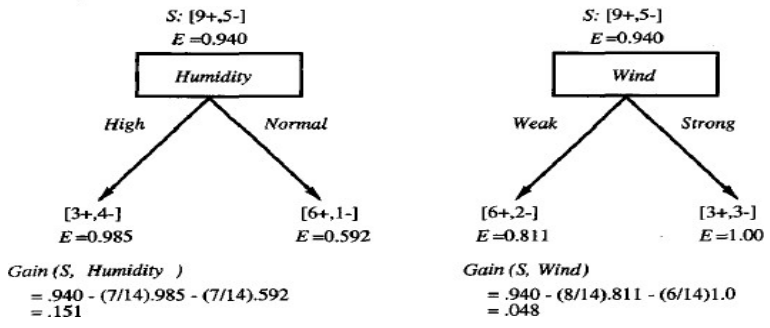
- The information gain due to sorting the original ***14*** examples by the attribute ***Wind*** may then be calculated as

$$
\begin{aligned}
Values(Wind) &= Weak, Strong \\
S &= [9+, 5-] \\
S_{Weak} &\leftarrow [6+, 2-] \\
S_{Strong} &\leftarrow [3+, 3-] \\
Gain(S, Wind) &= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v) \\
&= Entropy(S) - (8/14) Entropy(S_{Weak}) \\
&\quad - (6/14) Entropy(S_{Strong}) \\
&= 0.940 - (8/14)0.811 - (6/14)1.00 \\
&= 0.048
\end{aligned}
$$

Which attribute is the best classifier?

Figure 3.3

Humidity provides greater information gain than ***Wind,*** relative to the target classification. Here, E stands for entropy and ***S*** for the original collection of examples. Given an initial collection ***S*** of 9 positive and 5 negative examples, **[9+,** 5-], sorting these by their ***Humidity*** produces collections of [3+, 4-] ***(Humidity*** = ***High)*** and **[6+,** 1-] ***(Humidity*** = ***Normal).*** The information gained by this partitioning is 0.151,



compared to a gain of only **.048** for the attribute ***Wind.***

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

Table 3.2: Training examples for the target concept ***PlayTennis***

The information gain values for all four attributes are
Gain(S, Outlook) = 0.246
Gain(S, Humidity) = 0.151
Gain(S, Wind) = 0.048
Gain(S, Temperature) = 0.029
where ***S*** denotes the collection of training examples

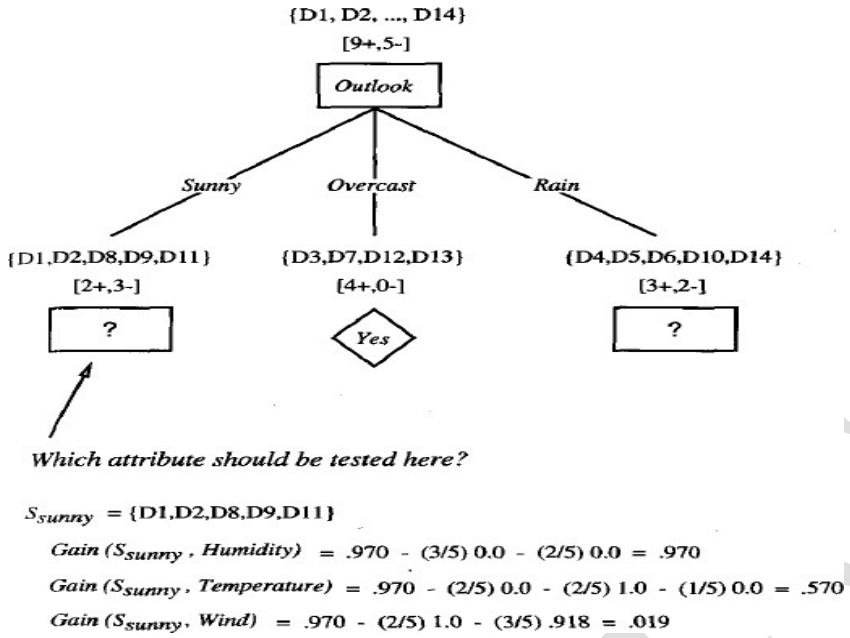


Fig 3.4: The partially learned decision tree resulting from the first step of ID3.

The training examples are sorted to the corresponding descendant nodes. The *Overcast* descendant has only positive examples and therefore becomes a leaf node with classification *Yes*. The other two nodes will be further expanded, by selecting the attribute with highest information gain relative to the new subsets of examples.

3.5 HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

- **ID3** in its pure form performs no backtracking in its search. Once it, selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice. Therefore, it is susceptible to the usual risks of hill-climbing search without backtracking: converging to locally optimal solutions that are not globally optimal.
- Locally optimal solution may be less desirable than trees that would have been encountered along a different branch of the search.

- **ID3** uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis.

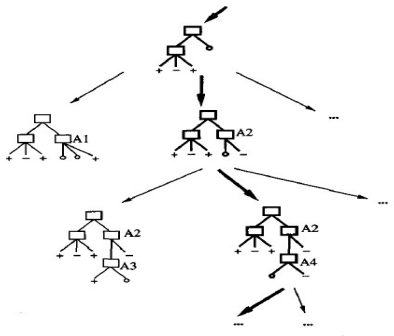


FIGURE 3.5
Hypothesis space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the *information gain heuristic.*

- **ID3** can be easily extended to handle noisy training data by modifying its termination criterion to accept hypotheses that imperfectly fit the training data.

3.7 ISSUES IN DECISION TREE LEARNING

- Incorporating Continuous-Valued Attributes
- Alternative Measures for Selecting Attributes
- Handling Training Examples with Missing Attribute Values
- Handling Attributes with Differing Costs.
- Avoiding Over-fitting the Data
  - Reduced Error Pruning
  - Rule Post-Pruning

Incorporating Continuous-Valued Attributes

In particular, 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.

Suppose further that the training examples associated with a particular node in the decision tree have the following values for ***Temperature*** and the target attribute ***PlayTennis.***

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

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.***

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, $Temperature_{>54}$ $Temperature_{>85}$ the best can be selected ($Temperature_{>54}$)

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.

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 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

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

Handling Training Examples with Missing Attribute Values

In certain cases, the available data may be missing values for some attributes.

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).***

A second, more complex procedure is to assign a probability to each of the possible values of ***A*** rather than simply assigning the most common value to ***A(x).*** These probabilities can be estimated again based on the observed frequencies of the various values for ***A*** among the examples at node ***n.***

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, both in terms of monetary cost and cost to patient comfort. 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.

More efficient recognition strategies are learned, without sacrificing classification accuracy, by replacing the information gain attribute selection measure by the following measure

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

Avoiding Over-fitting the Data

We will say that a hypothesis overfits the training examples if some other hypothesis that fits the training examples less well actually performs better over the entire distribution of instances (i.e., including instances beyond the training set).

> **Definition: Given a hypothesis space H, a hypothesis h ∈ H is said to over-fit the training data if there exists some alternative hypothesis h' ∈ 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.

There are several approaches to avoiding overfitting in decision tree learning. These can be grouped into two classes:

- Approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data,
- Approaches that allow the tree to over-fit the data, and then post-prune the tree.

Regardless of whether the correct tree size is found by stopping early or by post-pruning, a key question is what criterion is to be used to determine the correct final tree size. Approaches include:

- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.
- Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set.
- Use an explicit measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized.

REDUCED ERROR PRUNING

Pruning a decision node consists of removing the sub tree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node. Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set.

RULE POST-PRUNING

Rule post-pruning involves the following steps:

1. Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.

2. 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.

3. Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.

4. Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.