MODULE -2

DECISION TREE LEARNING

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.

DECISION TREE REPRESENTATION

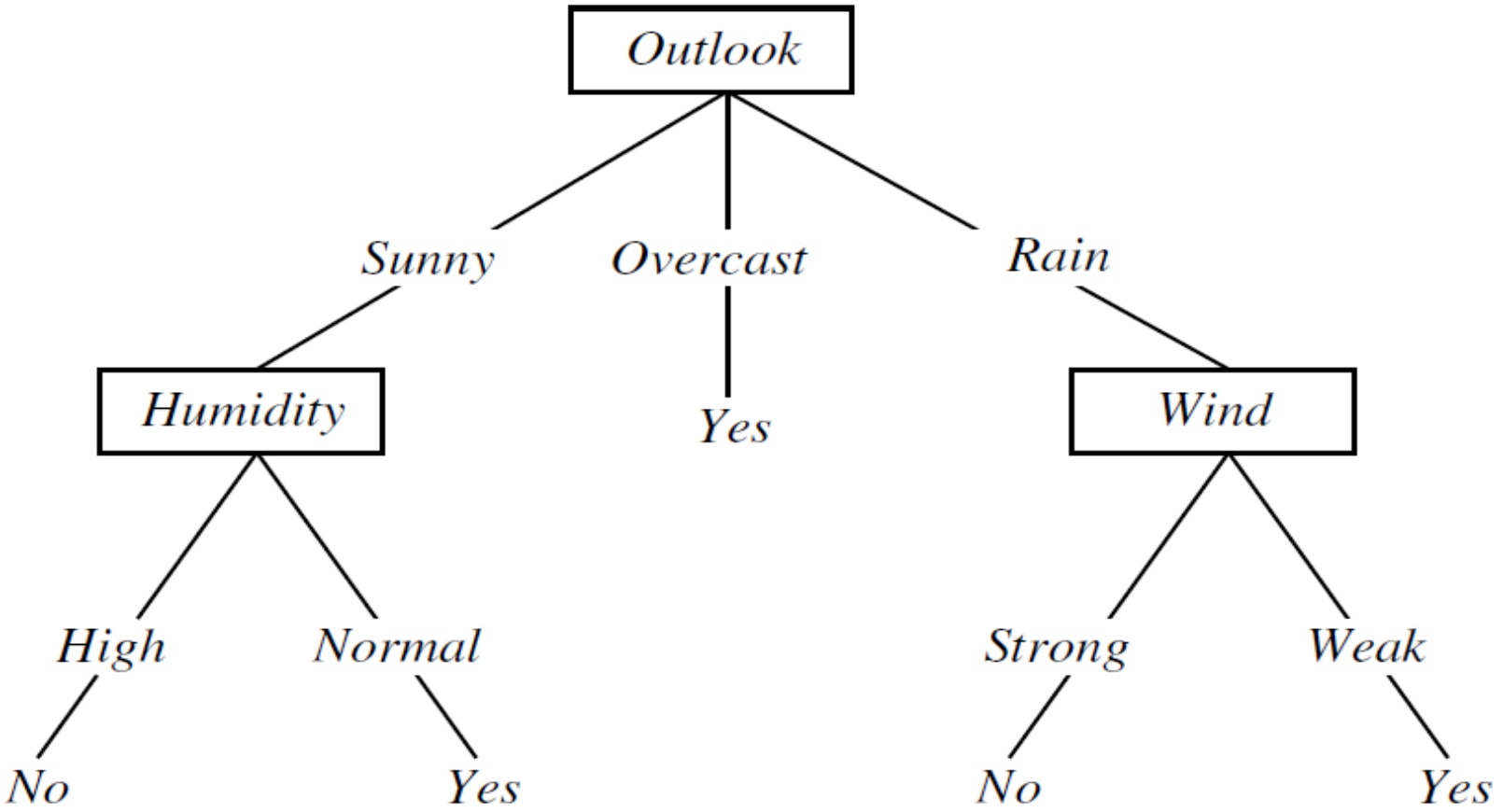


FIGURE: A decision tree for the concept ***PlayTennis***. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf

- 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.

- 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 above figure corresponds to the 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 and their values
2. ***The target function has discrete output values*** – The decision tree 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.
3. ***Disjunctive descriptions may be required***

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 even when some training examples have unknown values

- 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 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. This approach is exemplified by the ID3 algorithm and its successor C4.5

What is the ID3 algorithm?

- ID3 stands for Iterative Dichotomiser 3
- ID3 is a precursor to the C4.5 Algorithm.
- The ID3 algorithm was invented by Ross Quinlan in 1975
- Used to generate a decision tree from a given data set by employing a top-down, greedy search, to test each attribute at every node of the tree.
- The resulting tree is used to classify future samples.

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. 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 $Examples_{vi}$, be the subset of Examples that have value v_i for *A*
    - If $Examples_{vi}$, 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($Examples_{vi}$, Targe_tattribute, Attributes – {A}))
- End
- Return Root

* The best attribute is the one with highest information gain

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.
- A statistical property called ***information gain*** that measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses ***information gain*** measure to select among the candidate attributes at each step while growing the tree.

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

- To define information gain, we begin by defining a measure called entropy. *Entropy measures the impurity of a 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

p_- is the proportion of negative examples in S.

Example: Entropy

- 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

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

- The entropy is 0 if all members of S belong to the same class
- 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

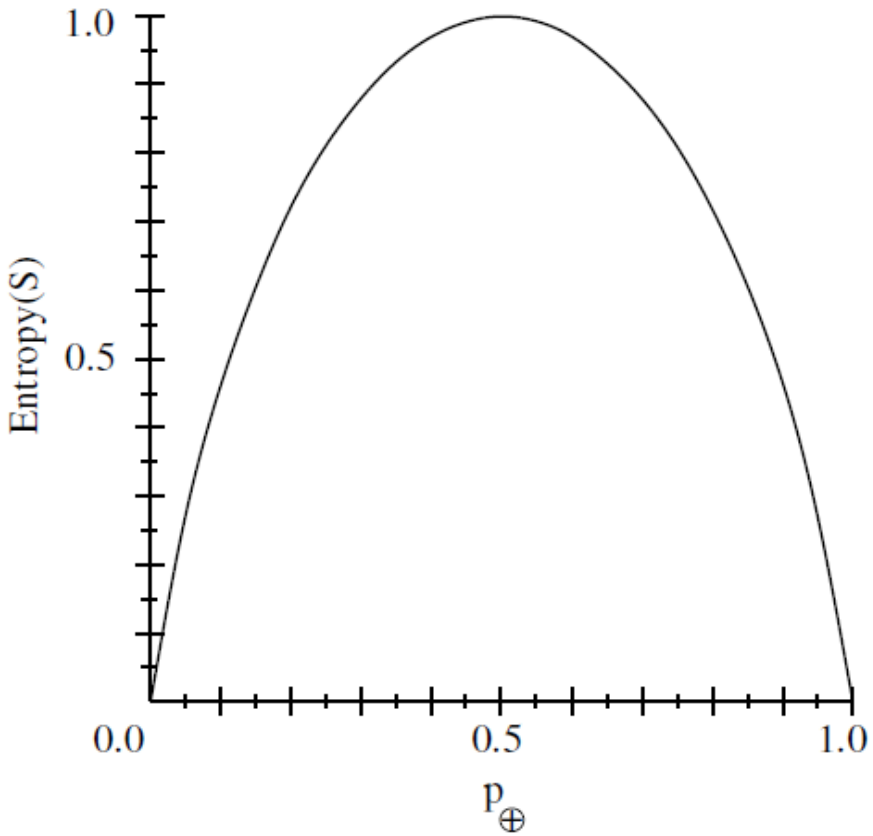


FIGURE The entropy function relative to a boolean classification, as the proportion, $p_{\oplus}$, of positive examples varies between 0 and 1.

INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY

- ***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

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

Example: Information gain

Let, $Values(Wind) = \{Weak, Strong\}$

$S = [9+, 5-]$

$S_{Weak} = [6+, 2-]$

$S_{Strong} = [3+, 3-]$

Information gain of attribute *Wind*:

$$Gain(S, Wind) = Entropy(S) - 8/14\ Entropy\ (S_{Weak}) - 6/14\ Entropy\ (S_{Strong})$$

$$= 0.94 - (8/14)* 0.811 - (6/14) *1.00$$

$$= 0.048$$

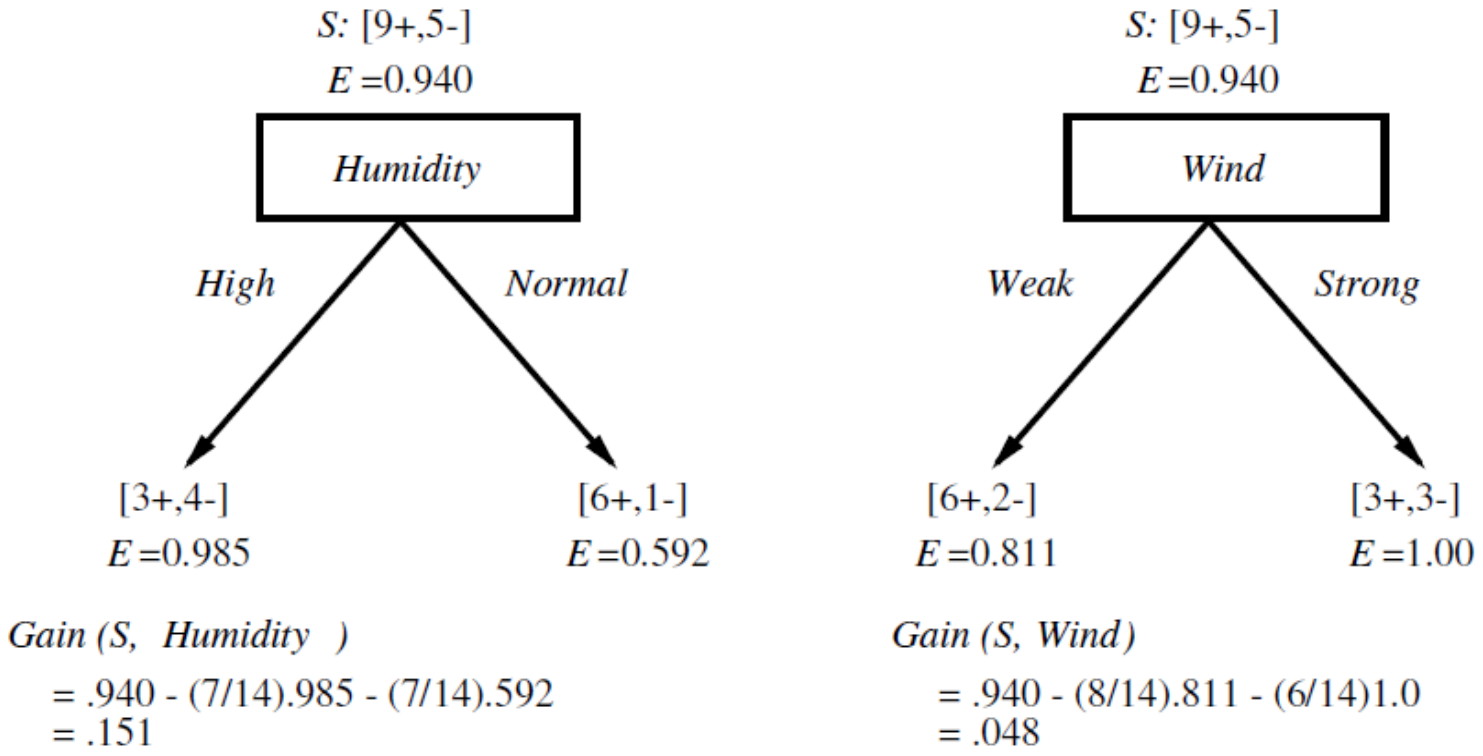
An Illustrative Example

- To illustrate the operation of ID3, consider the learning task represented by the training examples of below table.
- Here the target attribute ***PlayTennis***, which can have values ***yes*** or ***no*** for different days.
- Consider the first step through the algorithm, in which the topmost node of the decision tree is created.

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

ID3 determines the information gain for each candidate attribute (i.e., Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain

Which attribute is the best classifier?



Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592
= .151

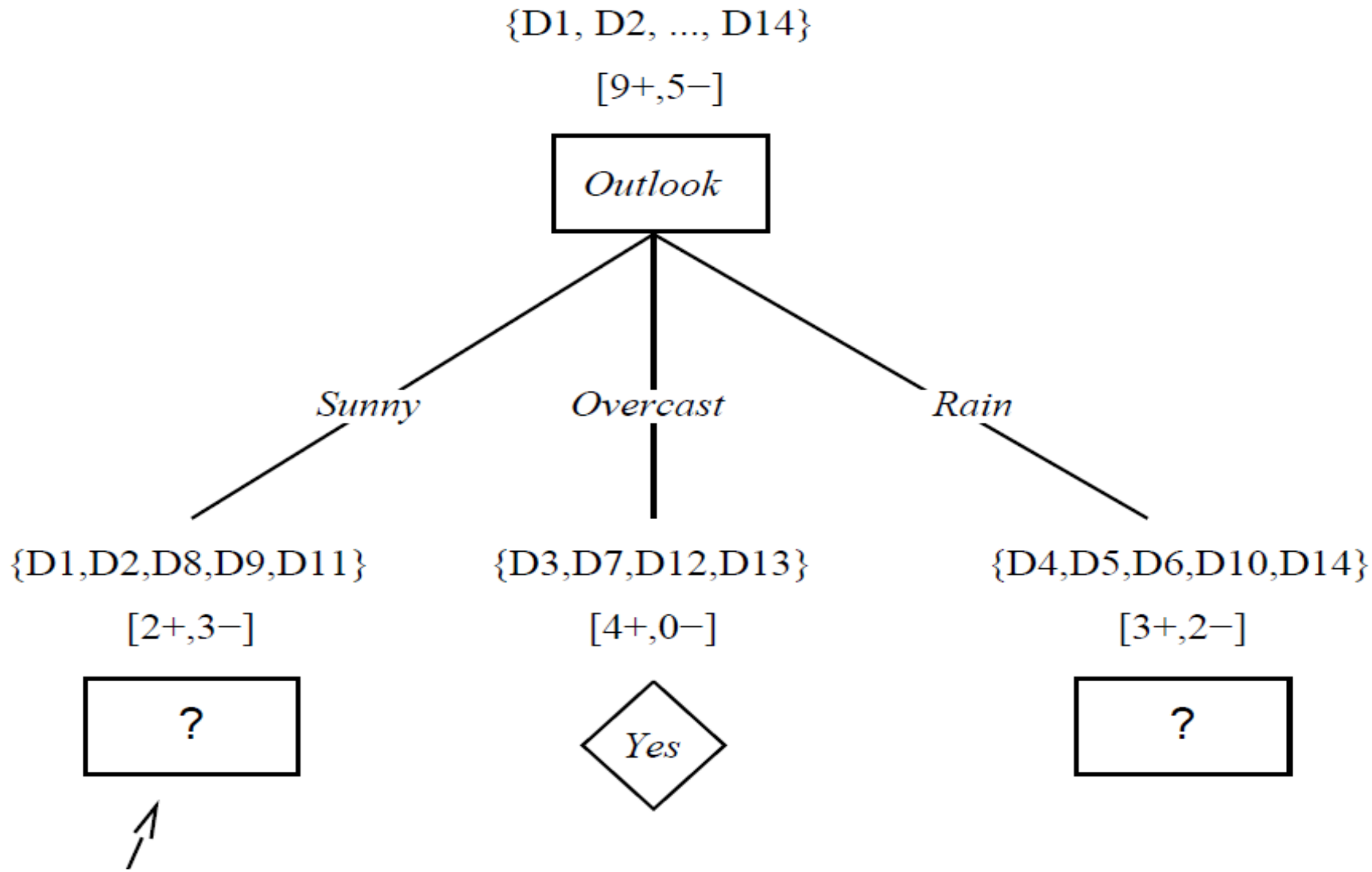
Gain (S, Wind)

= .940 - (8/14).811 - (6/14)1.0
= .048

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

- According to the information gain measure, the ***Outlook*** attribute provides the best prediction of the target attribute, ***PlayTennis***, over the training examples. Therefore, ***Outlook*** is selected as the decision attribute for the root node, and branches are created below the root for each of its possible values i.e., Sunny, Overcast, and Rain.


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
{D1,D2,D8,D9,D11}
[2+,3−]
{D3,D7,D12,D13}
[4+,0−]
{D4,D5,D6,D10,D14}
[3+,2−]
?
Yes
?
Which attribute should be tested here?

$S_{sunny} = \{D1, D2, D8, D9, D11\}$

$Gain\ (S_{sunny}, Humidity) = .970 - (3/5)\ 0.0 - (2/5)\ 0.0 = .970$

$Gain\ (S_{sunny}, Temperature) = .970 - (2/5)\ 0.0 - (2/5)\ 1.0 - (1/5)\ 0.0 = .570$

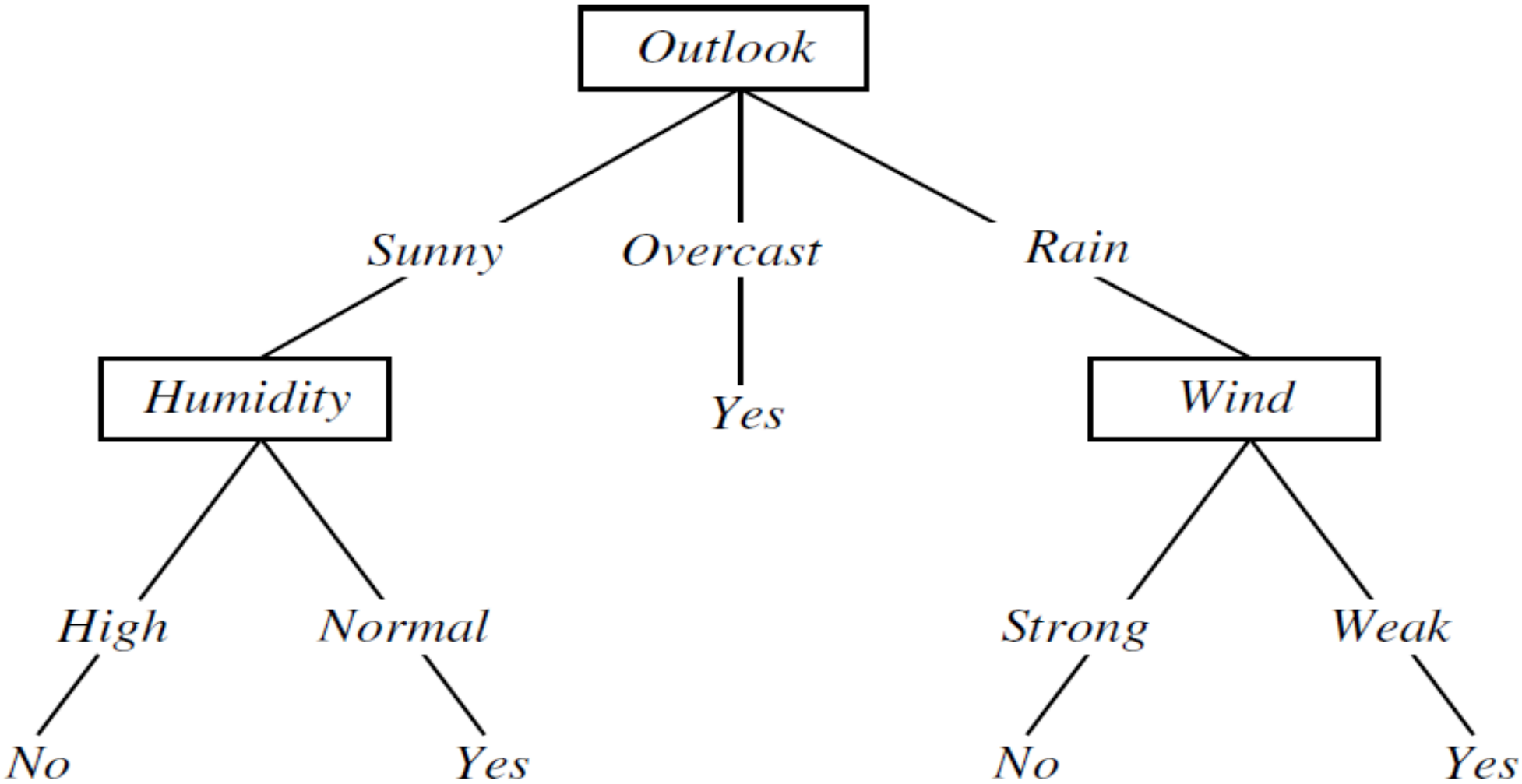
$Gain\ (S_{sunny}, Wind) = .970 - (2/5)\ 1.0 - (3/5)\ .918 = .019$

$S_{Rain} = \{ D4, D5, D6, D10, D14\}$

$Gain\ (S_{Rain}, Humidity) = 0.970 - (2/5)1.0 - (3/5)0.917 = 0.019$

$Gain\ (S_{Rain}, Temperature) = 0.970 - (0/5)0.0 - (3/5)0.918 - (2/5)1.0 = 0.019$

$Gain\ (S_{Rain}, Wind) = 0.970 - (3/5)0.0 - (2/5)0.0 = 0.970$


Outlook
Sunny
Overcast
Rain
Humidity
Yes
Wind
High
Normal
Strong
Weak
No
Yes
No
Yes

HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples.
- The hypothesis space searched by ID3 is the set of possible decision trees.
- ID3 performs a ***simple-to complex, hill-climbing search*** through this hypothesis space, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data

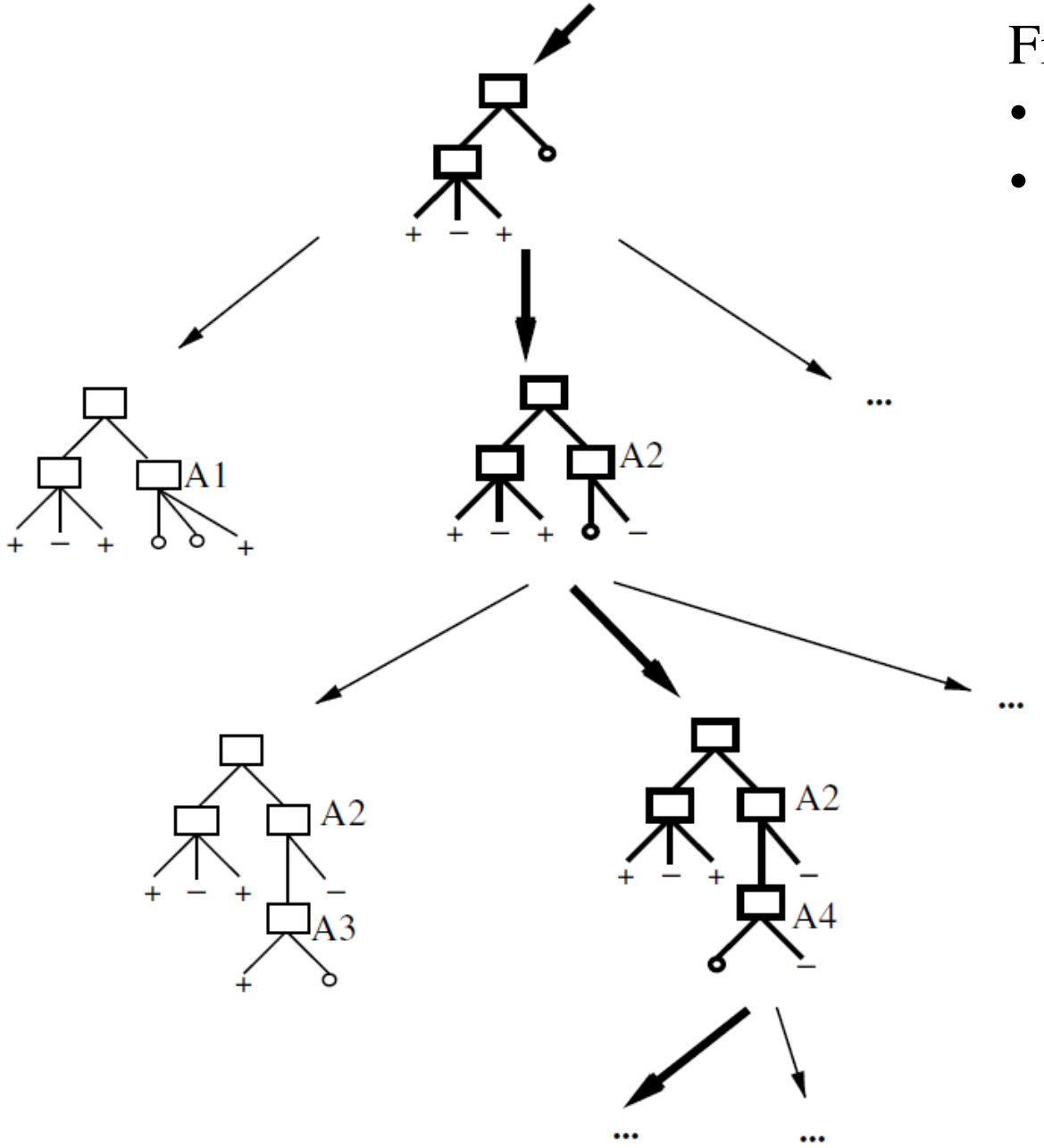


Figure:

- 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

By viewing ID3 in terms of its search space and search strategy, we can get some insight into its capabilities and limitations

1. ID3's hypothesis space of all decision trees is a ***complete*** space of finite discrete-valued functions, relative to the available attributes. Because every finite discrete-valued function can be represented by some decision tree

- ID3 avoids one of the major risks of methods that ***search incomplete hypothesis spaces*** : that the hypothesis space might not contain the target function.

2. ID3 maintains ***only a single current hypothesis*** as it searches through the space of decision trees.

For example, with the earlier version space candidate elimination method, which maintains the set of ***all*** hypotheses consistent with the available training examples.

By determining only a single hypothesis, ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses.

For example, it does not have the ability to determine how many alternative decision trees are consistent with the available training data, or to pose new instance queries that optimally resolve among these competing hypotheses

3. **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.

- In the case of **ID3,** a locally optimal solution corresponds to the decision tree it selects along the single search path it explores. However, this locally optimal solution may be less desirable than trees that would have been encountered along a different branch of the search.

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

- One advantage of using statistical properties of all the examples is that the resulting search is much ***less sensitive to errors*** in individual training examples.
- **ID3** can be easily extended to handle noisy training data by modifying its termination criterion to accept hypotheses that imperfectly fit the training data.

INDUCTIVE BIAS IN DECISION TREE LEARNING

Inductive bias is the set of assumptions that, together with the training data, deductively justify the classifications assigned by the learner to future instances

Given a collection of training examples, there are typically many decision trees consistent with these examples. Which of these decision trees does ID3 choose?

ID3 search strategy

(a) selects in favour of shorter trees over longer ones

(b) selects trees that place the attributes with highest information gain closest to the root.

Approximate inductive bias of ID3: Shorter trees are preferred over larger trees

- Consider an algorithm that begins with the empty tree and searches ***breadth first*** through progressively more complex trees.
- First considering all trees of depth 1, then all trees of depth 2, etc.
- Once it finds a decision tree consistent with the training data, it returns the smallest consistent tree at that search depth (e.g., the tree with the fewest nodes).
- Let us call this breadth-first search algorithm BFS-ID3.
- BFS-ID3 finds a shortest decision tree and thus exhibits the bias "shorter trees are preferred over longer trees.

A closer approximation to the inductive bias of ID3: Shorter trees are preferred over longer trees. Trees that place high information gain attributes close to the root are preferred over those that do not.

- ID3 can be viewed as an efficient approximation to BFS-ID3, using a greedy heuristic search to attempt to find the shortest tree without conducting the entire breadth-first search through the hypothesis space.
- Because ID3 uses the information gain heuristic and a hill climbing strategy, it exhibits a more complex bias than BFS-ID3.
- In particular, it does not always find the shortest consistent tree, and it is biased to favour trees that place attributes with high information gain closest to the root.

Restriction Biases and Preference Biases

Difference between the types of inductive bias exhibited by ID3 and by the CANDIDATE-ELIMINATION Algorithm.

ID3

- ID3 searches a complete hypothesis space
- It searches incompletely through this space, from simple to complex hypotheses, until its termination condition is met
- Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy. Its hypothesis space introduces no additional bias

CANDIDATE-ELIMINATION Algorithm

- The version space CANDIDATE-ELIMINATION Algorithm searches an incomplete hypothesis space
- It searches this space completely, finding every hypothesis consistent with the training data.
- Its inductive bias is solely a consequence of the expressive power of its hypothesis representation. Its search strategy introduces no additional bias

Restriction Biases and Preference Biases

- The inductive bias of ID3 is a *preference* for certain hypotheses over others (e.g., preference for shorter hypotheses over larger hypotheses), with no hard restriction on the hypotheses that can be eventually enumerated. This form of bias is called a ***preference bias*** or ***a search bias***.

- The bias of the CANDIDATE ELIMINATION algorithm is in the form of a *categorical* restriction on the set of hypotheses considered. This form of bias is typically called a ***restriction bias*** or ***a language bias***.

Which type of inductive bias is preferred in order to generalize beyond the training data, a preference bias or restriction bias?

- A preference bias is more desirable than a restriction bias, because it allows the learner to work within a complete hypothesis space that is assured to contain the unknown target function.
- In contrast, a restriction bias that strictly limits the set of potential hypotheses is generally less desirable, because it introduces the possibility of excluding the unknown target function altogether.

Occam's razor

Occam's razor: is the problem-solving principle that the simplest solution tends to be the right one. When presented with competing hypotheses to solve a problem, one should select the solution with the fewest assumptions.

Occam's razor: "***Prefer the simplest hypothesis that fits the data***".

Why Prefer Short Hypotheses ?

Argument in favour:

Fewer short hypotheses than long ones:

- Short hypotheses fits the training data which are *less likely* to be *coincident*
- Longer hypotheses fits the training data might be *coincident.*

Many complex hypotheses that fit the current training data but fail to generalize correctly to subsequent data.

Argument opposed:

- There are few small trees, and our priori chance of finding one consistent with an arbitrary set of data is therefore small. The difficulty here is that there are very many small sets of hypotheses that one can define *but understood by fewer learner*.
- The size of a hypothesis is determined by the representation used ***internally*** by the learner. Occam's razor will produce *two different hypotheses from the same training examples when it is applied by two learners*, both justifying their contradictory conclusions by Occam's razor. On this basis we might be tempted to reject Occam's razor altogether.

ISSUES IN DECISION TREE LEARNING

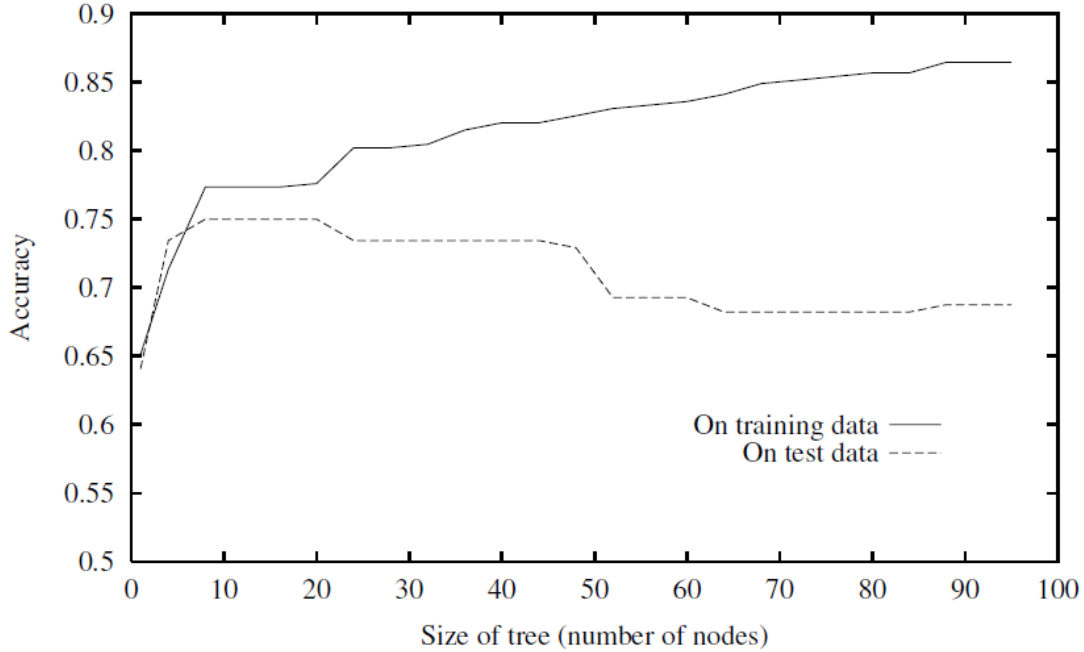
1. Avoiding Overfitting the Data
    - Reduced error pruning
    - Rule post-pruning
2. Incorporating Continuous-Valued Attributes
3. Alternative Measures for Selecting Attributes
4. Handling Training Examples with Missing Attribute Values
5. Handling Attributes with Differing Costs

1. Avoiding Overfitting the Data

- The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties 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. This algorithm can produce trees that ***overfit*** the training examples.

- ***Definition - Overfit:*** Given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, 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.

- The below figure illustrates the impact of overfitting in a typical application of decision tree learning.



- The **horizontal axis** of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed. The **vertical axis** indicates the accuracy of predictions made by the tree.
- The **solid line** shows the accuracy of the decision tree over the training examples. The **broken line** shows accuracy measured over an independent set of test example
- The **accuracy** of the tree over the **training examples increases** monotonically as the tree is grown. The **accuracy** measured over the independent test examples **first increases**, **then decreases**.

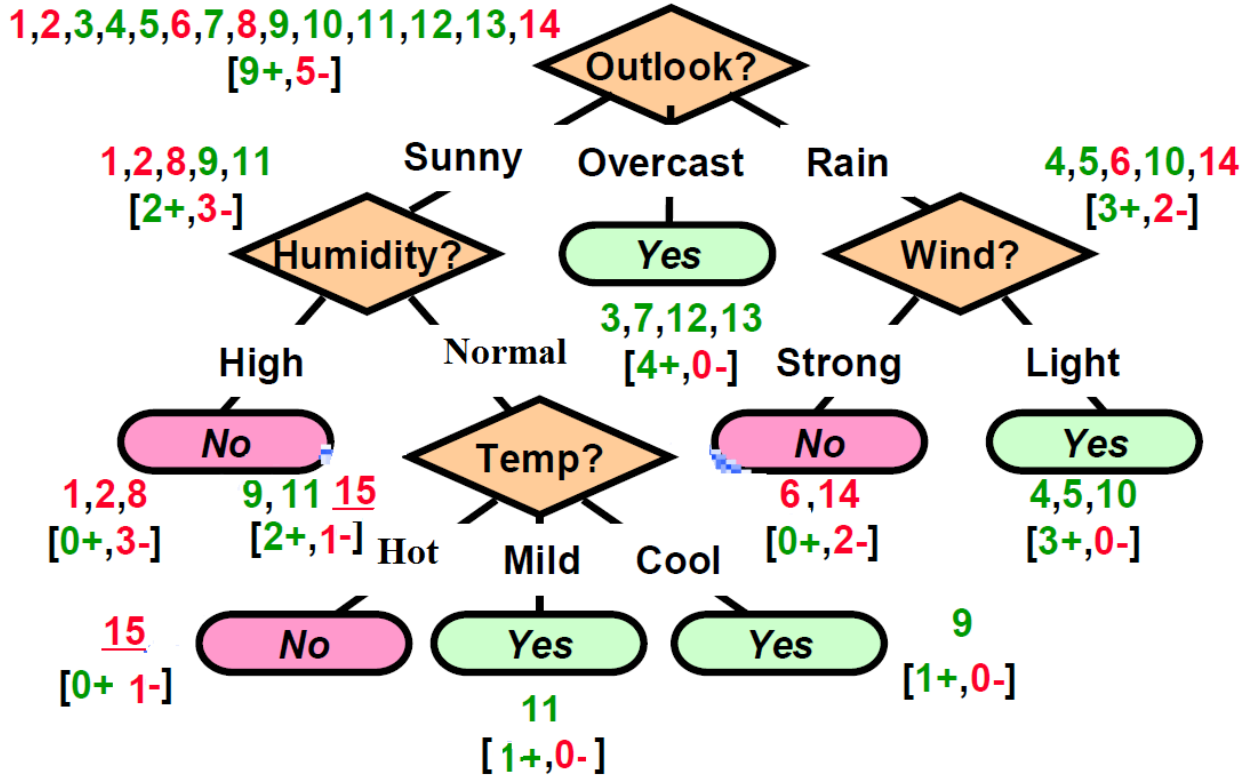
How can it be possible for tree h to fit the training examples better than h', but for it to perform more poorly over subsequent examples?

1. Overfitting can occur when the training examples contain random errors or noise
2. When small numbers of examples are associated with leaf nodes.

Noisy Training Example

Example 15: *<Sunny, Hot, Normal, Strong, ->*

- Example is noisy because the correct label is +
- Previously constructed tree misclassifies it



Approaches to avoiding overfitting in decision tree learning

- **Pre-pruning (avoidance):** Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- **Post-pruning (recovery):** Allow the tree to overfit the data, and then post-prune the tree

Criterion used to determine the correct final tree size

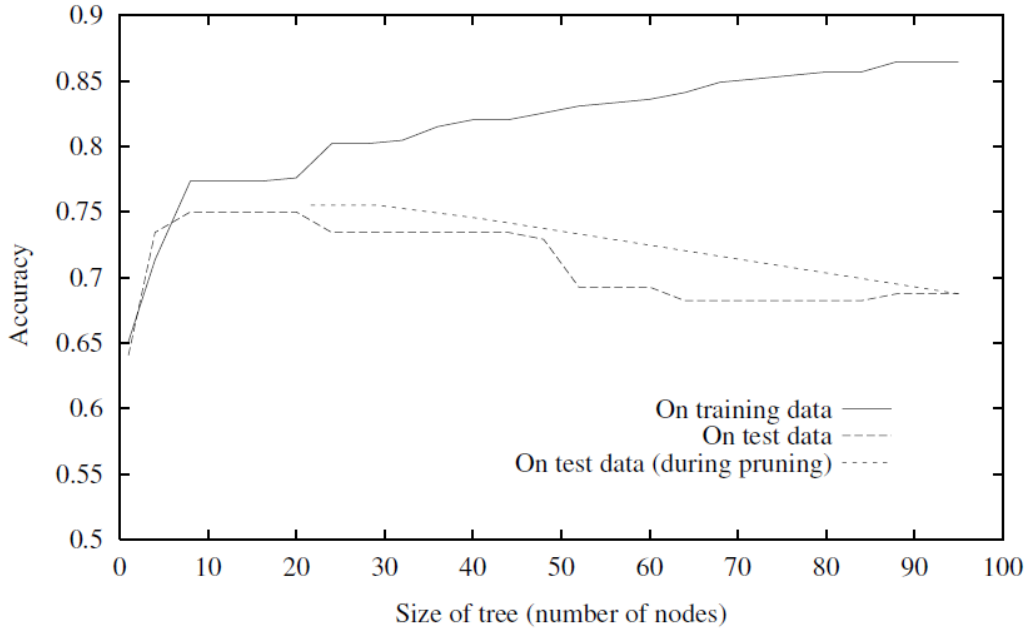
- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree
- Use all the available data for training, but apply ***a statistical test*** to estimate whether expanding (or pruning) a particular node is likely to produce an improvement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

$$MDL - Minimize : size(tree) + size\ (misclassifications(tree))$$

Reduced-Error Pruning

- ***Reduced-error pruning***, is to consider each of the decision nodes in the tree to be candidates for pruning
- ***Pruning*** a decision node consists of removing the subtree 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.
- Reduced error pruning has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set

The impact of reduced-error pruning on the accuracy of the decision tree is illustrated in below figure



- The additional line in figure shows ***accuracy over the test examples*** as the tree is pruned. When pruning begins, the tree is at its maximum size and lowest accuracy over the test set. As pruning proceeds, the number of nodes is reduced and accuracy over the test set increases.
- The available data has been split into three subsets: the training examples, the validation examples used for pruning the tree, and a set of test examples used to provide an unbiased estimate of accuracy over future unseen examples. The plot shows accuracy over the training and test sets.

Pros and Cons

Pro: Produces smallest version of most accurate T (subtree of T)

Con: Uses less data to construct T

Can afford to hold out $D_{validation}$?. If not (data is too limited), may make error worse (insufficient D_{train})

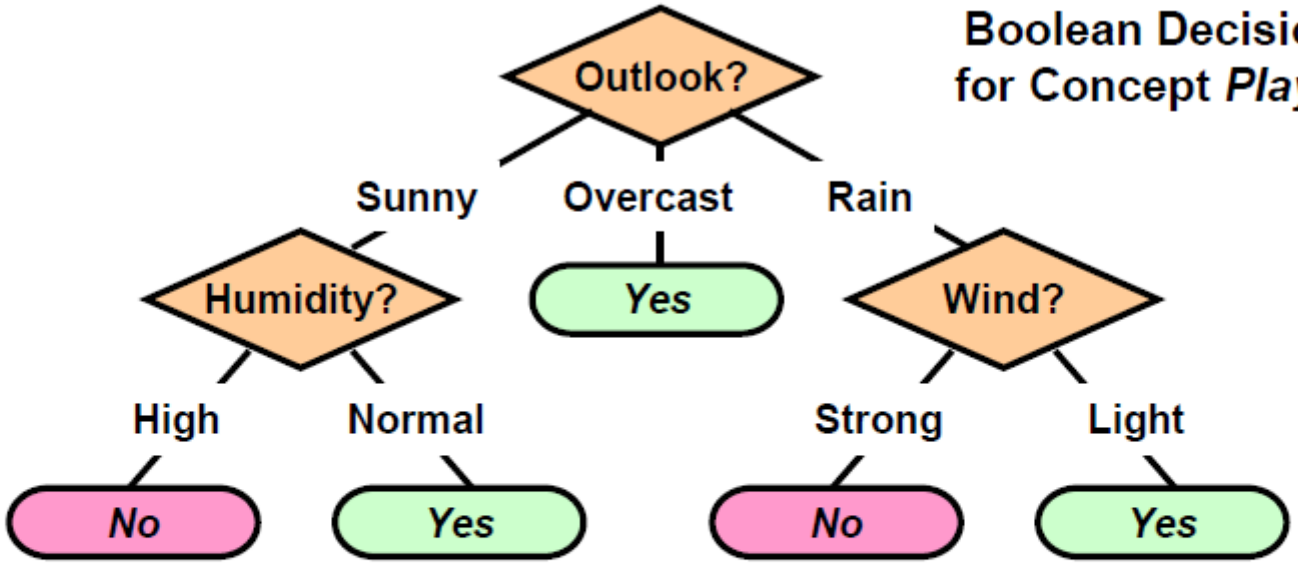
Rule Post-Pruning

Rule post-pruning is successful method for finding high accuracy hypotheses

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.

Converting a Decision Tree into Rules



Boolean Decision Tree for Concept *PlayTennis*

Example

- IF (*Outlook* = *Sunny*) ∧ (*Humidity* = *High*) THEN *PlayTennis* = *No*
- IF (*Outlook* = *Sunny*) ∧ (*Humidity* = *Normal*) THEN *PlayTennis* = *Yes*
- …

For example, consider the decision tree. The leftmost path of the tree in below figure is translated into the rule.

IF (Outlook = Sunny) ^ (Humidity = High)

THEN *PlayTennis* = No

Given the above rule, 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.

There are three main advantages by converting the decision tree to rules before pruning

- Converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path.
- Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves. Thus, it avoid messy bookkeeping issues such as how to reorganize the tree if the root node is pruned while retaining part of the subtree below this test.
- Converting to rules improves readability. Rules are often easier for to understand.

2. Incorporating Continuous-Valued Attributes

Continuous-valued decision attributes can be incorporated into the learned tree.

There are two methods for Handling Continuous Attributes

1. Define new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals.

E.g., {high ≡ Temp > 35º C, med ≡ 10º C < Temp ≤ 35º C, low ≡ Temp ≤ 10º C}

2. Using thresholds for splitting nodes

e.g., A ≤ a produces subsets A ≤ a and A > a

What threshold-based boolean attribute should be defined based on Temperature?

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

Pick a threshold, ***c,*** that produces the greatest information gain

- 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 $_{>54}$)

3. Alternative Measures for Selecting Attributes

The problem is if attributes with many values, ***Gain*** will select it ?

Example: consider the attribute ***Date***, which has a very large number of possible values. (e.g., March 4, 1979).

- If this attribute is added to the ***PlayTennis*** data, 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 tree of depth one, which perfectly classifies the training data.
- This decision tree with root node ***Date*** is not a useful predictor because it perfectly separates the training data, but poorly predict on subsequent examples.

One Approach: Use *GainRatio* instead of *Gain*

- The gain ratio measure penalizes attributes by incorporating a split information, that is sensitive to how broadly and uniformly the attribute splits the data

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

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

- where S_i is subset of S, for which attribute A has value v_i

4. Handling Training Examples with Missing Attribute Values

The data which is available may contain missing values for some attributes

Example: Medical diagnosis

- <*Fever* = *true*, *Blood-Pressure* = *normal*, …, *Blood-Test* = *?*, …>
- Sometimes values truly unknown, sometimes low priority (or cost too high)

Example : PlayTennis

Day	Outlook	Temperature	Humidity	Wind	*PlayTennis*?
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	???	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

Strategies for dealing with the missing attribute value

- If node n test A, assign most common value of A among other training examples sorted to node ***n***
- Assign most common value of A among other training examples with same target value
- Assign a probability p_i to each of the possible values v_i of ***A*** rather than simply assigning the most common value to ***A(x)***

5. Handling Attributes with Differing Costs

In some learning tasks the instance attributes may have associated costs.

For example:

- In learning to classify medical diseases, the patients described 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
- Decision trees use low-cost attributes where possible, depends only on high-cost attributes only when needed to produce reliable classifications

How to Learn A Consistent Tree with Low Expected Cost?

One approach is replace Gain by ***Cost-Normalized-Gain***

Examples of normalization functions

- Tan and Schlimmer

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

- Nunez

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

where $w \in [0, 1]$ determines importance of cost