DECISION TREE LEARNING

- The problem of ***inducting*** general functions from training examples is central to ***learning***.
- ***Decision tree learning*** is one of the most widely used methods for *inductive inference.*[1]
  - It approximates discrete-valued functions (learned function is represented by a decision tree), is robust to noisy data, and is capable of learning disjunctive expressions.
- ***Learned trees can also be represented as sets of if-then rules*** to improve human interpretation.
- We study a family of ***decision tree learning algorithms*** that includes widely used algorithms such as ***ID3*** and ***C4.5***.
  - ***They search the complete hypothesis space.***
  - ***Their inductive bias is a preference for small trees over large trees.***
- These methods are popular and have been applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

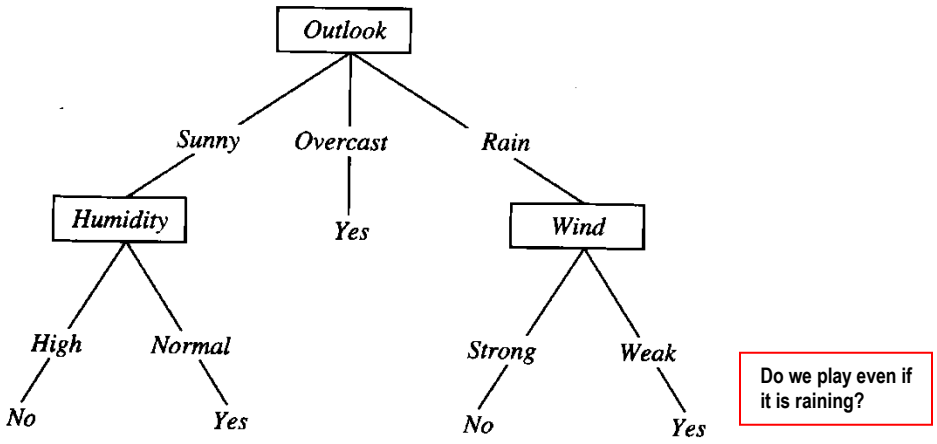


Figure 1. A decision tree for the concept *PlayTennis*. It classifies Saturday mornings according to their suitability for playing tennis.

DECISION TREE REPRESENTATION

- ***Decision trees "classify instances" by "sorting" them down the tree***
  - Sorts from the root to some leaf node that provides the classification of the instance
- **Each tree node 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.
- Process is repeated for the subtree rooted at the new node.
- Example: Figure 1 illustrates a typical decision tree. It classifies Saturday mornings according to whether they are suitable for playing tennis. For example, the instance

$$\langle Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong \rangle$$

[1] *Inductive learning* (or learning from examples) is the task of identifying regularities in some given set of training examples with little or no knowledge about the domain from which the examples are drawn [Michalski, 1983].

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 *Play Tennis* = *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**.
- Example: The decision tree shown above corresponds to the expression:

$$(Outlook = Sunny \wedge Humidity = Normal)$$
$$\vee (Outlook = Overcast) \qquad \Rightarrow \text{Conditions for "Play Tennis"}$$
$$\vee (Outlook = Rain \wedge Wind = Weak)$$

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*). It is preferable if each attribute takes on a small number of disjoint possible values (e.g., *Hot, Mild, and Cold*). However, extensions to the basic algorithm allow handling real-valued attributes as well (e.g., representing *Temperature* numerically).
  - *Target function has discrete output values*. The decision tree shown earlier assigns a Boolean classification (e.g., yes or no) to each instance. 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 (e.g., *CART*), 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.
  - *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.
  - *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).

THE BASIC DECISION TREE LEARNING ALGORITHM

- Most **decision tree learning algorithms are variations on a core algorithm** that employs a ***top-down greedy search*** through the space of possible decision trees
- Approach is exemplified by the ***ID3 algorithm*** (Quinlan, 1986) and its successor ***C4.5*** (Quinlan, 1993)

ID3 Algorithm

- ID3 learns decision trees by constructing them top-down, beginning with the question "**which attribute should be tested at the root of the tree**?"
  - It does so by **evaluating each instance attribute 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**
  - 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.
- A simplified version of the algorithm, specialized to learning Boolean-valued functions is described in Table 1.

Table 1. ID3 Decision Tree Learning Algorithm

ID3(*Examples, Target-attribute, Attributes*)

Examples are 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*

*The best attribute is the one with highest information gain (as defined in Eqn. (4)).

Which Attribute Is the Best Classifier?

- Central choice in the ID3 algorithm is **selecting the 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***, which 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 (Non)Homogeneity of Examples

- To define information gain precisely, we define a measure commonly used in information theory, called ***entropy***, that characterizes (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

$$Entropy(S) \equiv -P_+ \log_2 P_+ - P_- \log_2 P_- \quad (1)$$

where P_+ is the proportion of positive examples in S and P_- is the proportion of negative examples in S. In all calculations involving entropy we define 0×log(0) to be 0.

- Example: Suppose S is a collection of 14 examples of some Boolean concept, including 9 positive and 5 negative examples (we adopt the notation [9+, 5-] to summarize such a sample). Entropy of S relative to this Boolean classification is

$$Entropy([9+,5-]) = -(9/14)\log 2(9/14) - (5/14)\log 2(5/14) = 0.940 \quad (2)$$

Note: Entropy is 0 if all members of S belong to same class. Entropy is 1 if collection contains an equal number of positive and negative examples. ***Entropy is bounded between 0 and 1.***

- Figure 2 below plots entropy function relative to a boolean classification, as P_+ varies between 0 and 1.
- One interpretation of **entropy** from information theory is that it **specifies the minimum number of bits of information needed to encode** the classification of an arbitrary member of S (i.e., a member of S drawn at random with uniform probability).
  - **If P_+ is 1**, the receiver knows any drawn example will be positive, so **no message need be sent**, and the **entropy is zero**.
  - **If P_+ is 0.5**, **one bit is required to indicate whether the drawn example is +ve or -ve**.
  - If P_+ is 0.8, a collection of messages can be encoded using on average less than 1 bit per message by assigning shorter codes to collections of positive examples and longer codes to less likely negative examples.

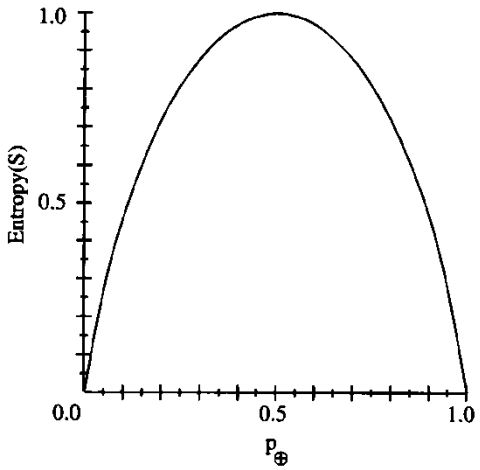


Figure 2. The entropy function relative to a boolean classification, as the proportion, P_+, of positive examples varies between 0 and 1.

- We have discussed entropy for the special case where the target classification is Boolean.
- **More generally**, if the target attribute can take on c different values, then the **entropy of S relative to this c-wise classification is defined as**

$$Entropy(S) = \sum_{i=1}^{c} -P_i \log_2 P_i \quad (3)$$

where P_i is the proportion of S belonging to class i.

Note: Logarithm is still base 2 because entropy is a measure of the expected encoding length measured in bits. ***If the target attribute can take on c possible values, the entropy can be as large as $\log_2 c$.***

Information Gain Measures the Expected Reduction in Entropy

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

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \left(\frac{|S_v|}{|S|} Entropy\ (S_v) \right) \qquad (4)$$

where $Values(A)$ is the set of all possible values for attribute A and S_v is the subset of S for which attribute A has value v (i.e., $S_v = \{S \mid A(s) = v\}$).

- Note: First term in Eqn. (4) is just the entropy of the original collection S and second term is the expected value of the entropy after S is partitioned using attribute A.
  - Expected entropy described by the second term is simply the sum of the entropies of each subset S_v, weighted by the fraction of examples $|S_v|/|S|$ that belong to S_v.
- $Gain(S, A)$ **is expected reduction in entropy caused by knowing the value of attribute** A**.**
- Put another way, $Gain(S, A)$ is the information provided about the target function value, given the value of attribute A.
- $Gain(S, A)$ is the number of bits saved when encoding the target value of an arbitrary member of S, by knowing the value of attribute A.
- 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 has *Wind* = *Strong*.
  - Information gain due to sorting the original 14 examples by the attribute *Wind* may then be calculated as

$$Values(Wind) = Weak, Strong$$

$$S = [9+, 5-]$$

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

$$S_{Strong} \leftarrow [3+, 3-]$$

$$Gain(S, A) = Entropy(S) - \sum_{v \in \{Weak, Strong\}} \left(\frac{|S_v|}{|S|} Entropy(S_v) \right)$$

$$= Entropy(S) - (8/12) Entropy(S_{Weak}) - (6/14) Entropy(S_{Strong}) \qquad (8/14)$$

$$= 0.940 - (8/12)0.811 - (6/14)1.00$$

$$= 0.048$$

- Information gain is precisely the measure used by ID3 to select the best attribute at each step in growing the tree. See example below.

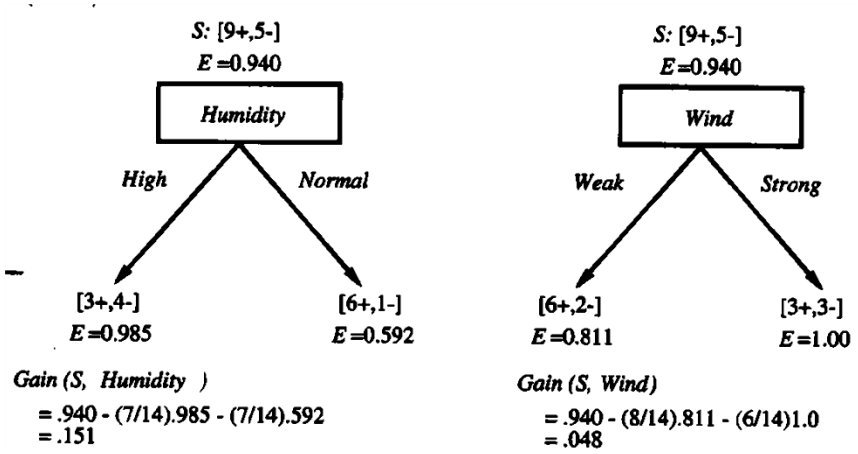


Figure 3. *Humidity* provides greater information gain than *Wind,* relative to the target classification. Here, *E* stands for entropy and *S* for the original collection of examples. Given an initial collection *S* of 9 positive and 5 negative examples, [9+, 5-], sorting these by their *Humidity* produces collections of [3+.4-] (*Humidity* = *High*) and [6+. I-] (*Humidity* = *Normal*). The information gained by this partitioning is 0.151, compared to a gain of only 0.048 for the attribute *Wind*.

An Illustrative Example:

- To illustrate the operation of ID3, consider the learning task represented by the training examples of the table below.
- Here the target attribute *PlayTennis*, which can have values *yes* or *no* for different Saturday mornings, is to be predicted based on other attributes of the morning in question.

Table 2. 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

- Consider the first step through the algorithm, creation of the topmost node of the decision tree.
- Which attribute should be tested first in the tree? ID3 determines the information gain for each candidate attribute (i.e., *Outlook, Temperature, Humidity, and Wind*), then selects the one with highest information gain.

- The computation of information gain for two of these attributes is shown in Figure 3. The information gain values for all four attributes are

$$Gain(S, Outlook) = 0.246$$

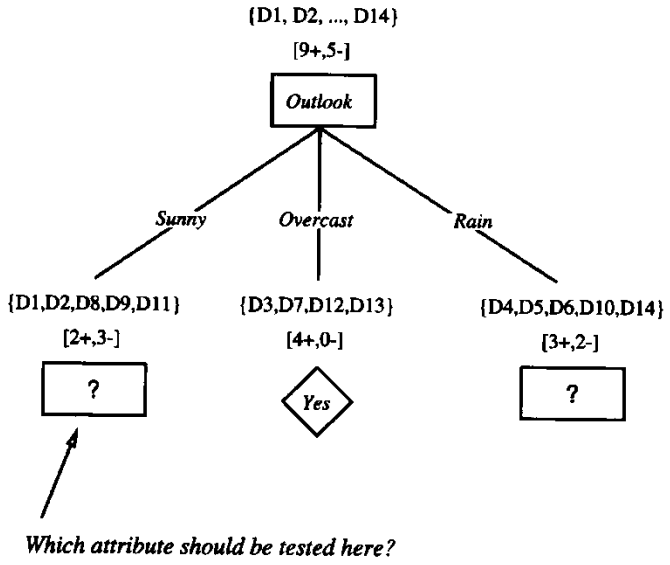
$$Gain(S, Humidity) = 0.151$$

$$Gain(S, Wind) = 0.048$$

$$Gain(S, Temperature) = 0.029$$

where S denotes the collection of training examples from Table 2.

- 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*).
- The resulting partial decision tree is shown in Figure 4, along with the training examples sorted to each new descendant node.



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

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

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

Figure 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.

- Note that every example for which *Outlook* = *Overcast* is also a positive example of *PlayTennis*. Therefore, this node of the tree becomes a leaf node with the classification *PlayTennis* = *Yes*.
- In contrast, the descendants corresponding to *Outlook* = *Sunny* and *Outlook* = *Rain* still have nonzero entropy, and the decision tree will be further elaborated below these nodes.

- The process of selecting a new attribute and partitioning the training examples is repeated for each non-terminal descendant node, this time using only examples associated with that node.
- Attributes that have been incorporated higher in the tree are excluded, so that any given attribute can appear at most once along any path through the tree.
- Process continues for each new leaf node until either of two conditions is met: (1) every attribute has already been included along this path through the tree, or (2) the training examples associated with this leaf node all have the same target attribute value (i.e., entropy is zero).
- The final decision tree learned by ID3 from the 14 training examples is shown in Figure 1.

HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

- ID3 performs a 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.
- The evaluation function that guides this hill-climbing search is the information gain measure. This search is depicted in Figure 5.

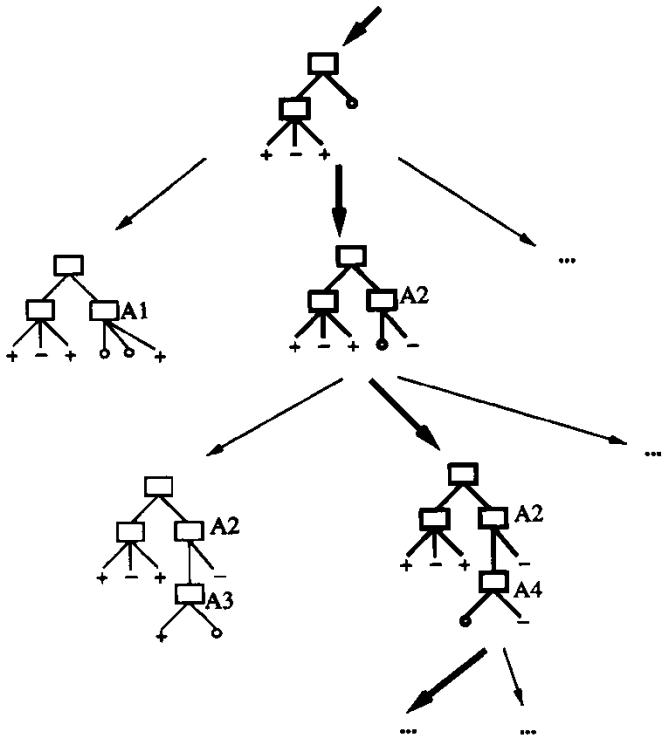


Figure 5. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the information gain heuristic.

- By viewing ID3 in terms of its search space and search strategy, we can get some insight into its capabilities and limitations.
- 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 (such as methods that consider only conjunctive hypotheses): that the hypothesis space might not contain the target function.
- ID3 maintains a single current hypothesis as it searches through the space of decision trees.
- **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.
  - ***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.
- However, this locally optimal solution may be less desirable than trees that would have been encountered along a different branch of the search.
- Below we discuss an extension that adds a form of backtracking (post-pruning the decision tree).

- ID3 uses "all" training examples at each step (batch training vs. pattern by pattern training) in the search to make statistically based decisions regarding how to refine its current hypothesis.
  - This contrasts with methods that make decisions incrementally, based on individual training examples.
  - One advantage of using statistical properties of all the examples (e.g., information gain) is that the resulting search is much less sensitive to errors in individual training example.
  - ***ID3 can be 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

- What is the policy by which ID3 generalizes from observed training examples to classify unseen instances? In other words, **what is its inductive bias**? [2]
- **Given a collection of training examples, there are typically many decision trees consistent with these examples.**
- Describing the inductive bias of ID3 therefore consists of describing the basis by which it chooses one of these consistent hypotheses over the others.
- Which decision tree does ID3 choose? **It chooses the first acceptable tree it encounters in its simple-to-complex, hill-climbing search through the space of possible trees**.
- Roughly speaking, then, the ID3 search strategy
  a) selects in favor of shorter trees over longer ones, and
  b) selects trees that place the attributes with highest information gain closest to the root.
- Because of the subtle interaction between the attribute selection heuristic used by ID3 and the particular training examples it encounters, it is difficult to characterize precisely the inductive bias exhibited by ID3.
- However, we can approximately characterize its bias as a preference for short decision trees over complex trees.

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

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

ISSUES IN DECISION TREE LEARNING

- Practical issues in learning decision trees include determining **how deeply to grow** the decision tree, **handling continuous attributes**, choosing an appropriate attribute selection measure, handling training data with **missing attribute values**, handling **attributes with differing costs**, and improving computational efficiency.
- **ID3 has itself been extended to address most of these issues, with the resulting system renamed C4.5** (Quinlan, 1993).

Avoiding Overfitting the Data

- The algorithm described in Table 1 **grows each branch of the tree to perfectly classify the training examples**.
  - This can lead to difficulties if 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
  - In these cases, ID3 **can produce trees that *overfit* the training examples**
- 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 $h \in H$ 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.
- Figure 6 illustrates the impact of overfitting in a typical application of decision tree learning.

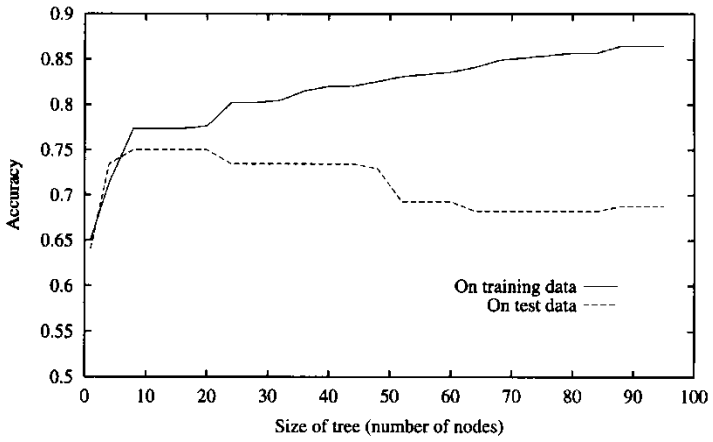


Figure 6. Overfitting in decision tree learning.

- In this case, the ID3 algorithm is applied to the task of learning which medical patients have a form of diabetes.
- As can be seen, once the tree size exceeds approximately 25 nodes, further elaboration of the tree decreases its accuracy over the test examples despite increasing its accuracy on the training examples.

- Overfilling is possible even when the training data are noise-free, especially when small numbers of examples are associated with leaf nodes.
- 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 overfit the data, and then **post-prune the tree**.
- Although the first of these approaches might seem more direct, the second approach of **post-pruning overfit trees has been found to be more successful in practice**. This is due to the difficulty in the first approach of estimating precisely when to stop growing the tree.

Incorporating Continuous-Valued Attributes

- Our initial definition of **ID3 is restricted to attributes that take on a discrete set of values**.
- **First**, the **target attribute** (whose value is predicted by the tree) **must be discrete valued**.
- **Second**, the **attributes tested in 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**. In particular, for an attribute A that is continuous-valued, the algorithm can dynamically create a new Boolean attribute A_c 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* in describing the training example days in the learning task of Table 2.
- 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**

- 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.
- It can be shown that the value of c that maximizes information gain must always lie at such a boundary (Fayyad, 1991).
- 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.
- Fayyad and Irani (1993) discuss an extension to this approach that **splits the continuous attribute into multiple intervals rather than just two intervals** based on a single threshold.
- Utgoff and Brodley (1991) and Murthy et al. (1994) discuss approaches that **define features by thresholding linear combinations of several continuous-valued attributes**.

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 in Table 2, 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, *Date* 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.
- What is wrong with the attribute *Date*? Simply put, it has so many possible values that it is bound to separate the training examples into very small subsets. Because of this, it will have a very high information gain relative to the training examples, despite being a very poor predictor of the target function over unseen instances.
- 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*** (Quinlan 1986).
- 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:

$$SplitInformation(S,A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \qquad (5)$$

where S_1 through S_c are the c subsets of examples resulting from partitioning S by the c-valued attribute A.

- Note that *SplitInformation* is actually the entropy of S with respect to the values of attribute A. This is in contrast to our previous uses of entropy, in which we considered only the entropy of S with respect to the target attribute whose value is to be predicted by the learned tree.

  The *GainRatio* measure is defined in terms of the earlier *Gain* measure, as well as this *SplitInformation*, as follows

$$GainRatio(S,A) \equiv \frac{Gain(S,A)}{SplitInformation(S,A)} \qquad (6)$$

- Notice that the *SplitInformation* term discourages the selection of attributes with many uniformly distributed values.

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 $\langle x, c(x) \rangle$ 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. This strategy is examined by Mingers (1989a).
- 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.
- For example, given a Boolean attribute A, if node n contains six known examples with $A=1$ and four with $A=0$, then we would say the probability that $A(x)=1$ is 0.6, and the probability that $A(x)=0$ is 0.4.
- A fractional 0.6 of instance x is now distributed down the branch for $A=1$, and a fractional 0.4 of x down the other tree branch.
- These fractional examples are used for the purpose of computing information *Gain* and can be further subdivided at subsequent branches of the tree if a second missing attribute value must be tested.
- This same fractioning of examples can also be applied after learning, to classify new instances whose attribute values are unknown.
- In this case, the classification of the new instance is simply the most probable classification, computed by summing the weights of the instance fragments classified in different ways at the leaf nodes of the tree. This method for handling missing attribute values is used in C4.5 (Quinlan, 1993).

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.
- ID3 can be modified to take into account attribute costs by introducing a cost term into the attribute selection measure.
- For example, we might divide the *Gain* by the cost of the attribute, so that lower-cost attributes would be preferred.
- While 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.
- Tan and Schlimmer (1990) and Tan (1993) describe one such approach and apply it to a robot perception task in which the robot must learn to classify different objects according to how they can be grasped by the robot's manipulator.
- In this case the attributes correspond to different sensor readings obtained by a movable sonar on the robot.
- Attribute cost is measured by the number of seconds required to obtain the attribute value by positioning and operating the sonar.
- They demonstrate that more efficient recognition strategies are learned, without sacrificing classification accuracy, by replacing the information gain attribute selection measure by the following measure

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

- Nunez (1988) describes a related approach and its application to learning medical diagnosis rules.
- Here the attributes are different symptoms and laboratory tests with differing costs. His system uses a somewhat different attribute selection measure

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

where $w \in [0,1]$ is a constant that determines the relative importance of cost versus information gain.

- Nunez (1991) presents an empirical comparison of these two approaches over a range of tasks.

Optimal Decision Trees

- Prevalent tree-building methods (such as ctree and rpart available in R) utilize only a single variable in each split, which limits the expressiveness and, in some cases, the predictive accuracy of the tree model.
- New research is seeking to develop "optimal" decision trees that aim to address the gap by providing an optimization-based framework to exploit multivariate splits and enable more expressive, comprehensible and accurate tree models.
- **Sample "R" Package**: **bsnsing** | https://rdrr.io/github/profyliu/bsnsing/
  - **Author**: Dr. Yanchao Liu, Industrial & Systems Engineering, Wayne State University
  - A mixed integer program (MIP) is solved at each node to identify the optimal combination of features used to split the node.
  - Currently, supported MIP solvers include CPLEX (commercial) and lpSolve (free).
  - This dynamically created Boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

SUMMARY AND FURTHER READING

- The main points of this chapter include:
  - Decision tree learning provides a practical method for concept learning and for learning other discrete-valued functions. The ID3 family of algorithms infers decision trees by growing them from the root downward, greedily selecting the next best attribute for each new decision branch added to the tree.
  - ID3 searches a complete hypothesis space (i.e., the space of decision trees can represent any discrete-valued function defined over discrete-valued instances). It thereby avoids the major difficulty associated with approaches that consider only restricted sets of hypotheses: that the target function might not be present in the hypothesis space.
  - The inductive bias implicit in ID3 includes a *preference* for smaller trees; that is, its search through the hypothesis space grows the tree only as large as needed in order to classify the available training examples.
  - Overfitting the training data is an important issue in decision tree learning. Because the training examples are only a sample of all possible instances, it is possible to add branches to the tree that improve performance on the training examples while decreasing performance on other instances outside this set. Methods for post-pruning the decision tree are therefore important to avoid overfitting in decision tree learning (and other inductive inference methods that employ a preference bias).
  - A large variety of extensions to the basic ID3 algorithm has been developed by different researchers. These include methods for post-pruning trees, handling real-valued attributes, accommodating training examples with missing attribute values, incrementally refining decision trees as new training examples become available, using attribute selection measures other than information gain, and considering costs associated with instance attributes.
- Among the earliest work on decision tree learning is Hunt's Concept Learning System (CLS) (Hunt et al. 1966) and Friedman and Breiman's work resulting in the CART system (Friedman 1977; Breiman et al. 1984).
- Quinlan's ID3 system (Quinlan 1979, 1983) forms the basis for the discussion in this document.
- Other early work on decision tree learning includes ASSISTANT (Kononenko et al. 1984; Cestnik et al. 1987).
- Implementations of decision tree induction algorithms are now commercially available on many computer platforms.
- For further details on decision tree induction, an excellent book by Quinlan (1993) discusses many practical issues and provides executable code for C4.5.
- Mingers (1989a) and Buntine and Niblett (1992) provide two experimental studies comparing different attribute-selection measures.
- Mingers (1989b) and Malerba et al. (1995) provide studies of different pruning strategies.
- Experiments comparing decision tree learning and other learning methods can be found in numerous papers, including (Dietterich et al. 1995; Shavlik et al. 1991).