CS601PC: MACHINE LEARNING

III Year B.Tech. CSE II-Sem.

Prerequisites

1. Data Structures
2. Knowledge on statistical methods

Course Objectives

1. This course explains machine learning techniques such as decision tree learning, Bayesian learning etc.
2. To understand computational learning theory.
3. To study the pattern comparison techniques.

Course Outcomes

1. Understand the concepts of computational intelligence like machine learning
2. Ability to get the skill to apply machine learning techniques to address the real time problems in different areas
3. Understand the Neural Networks and its usage in machine learning application.

UNIT - I

Introduction - Well-posed learning problems, designing a learning system, Perspectives and issues in machine learning.

Concept learning and the general to specific ordering – introduction, a concept learning task, concept learning as search, find-S: finding a maximally specific hypothesis, version spaces and the candidate elimination algorithm, remarks on version spaces and candidate elimination, inductive bias**.**

Decision Tree Learning – Introduction, decision tree representation, appropriate problems for decision tree learning, the basic decision tree learning algorithm, hypothesis space search in decision tree learning, inductive bias in decision tree learning, issues in decision tree learning.

UNIT - II

Artificial Neural Networks-1– Introduction, neural network representation, appropriate problems for neural network learning, perceptions, multilayer networks and the back-propagation algorithm.

Artificial Neural Networks-2- Remarks on the Back-Propagation algorithm, An illustrative example: face recognition, advanced topics in artificial neural networks.

Evaluation Hypotheses – Motivation, estimation hypothesis accuracy, basics of sampling theory, a general approach for deriving confidence intervals, difference in error of two hypotheses, comparing learning algorithms.

UNIT - III

Bayesian learning – Introduction, Bayes theorem, Bayes theorem and concept learning, Maximum Likelihood and least squared error hypotheses, maximum likelihood hypotheses for predicting probabilities, minimum description length principle, Bayes optimal classifier, Gibs algorithm, Naïve Bayes classifier, an example: learning to classify text, Bayesian belief networks, the EM algorithm.

Computational learning theory – Introduction, probably learning an approximately correct hypothesis, sample complexity for finite hypothesis space, sample complexity for infinite hypothesis spaces, the mistake bound model of learning.

Instance-Based Learning- Introduction, *k*-nearest neighbour algorithm, locally weighted regression, radial basis functions, case-based reasoning, remarks on lazy and eager learning.

UNIT- IV

Genetic Algorithms – Motivation, Genetic algorithms, an illustrative example, hypothesis space search, genetic programming, models of evolution and learning, parallelizing genetic algorithms.

Learning Sets of Rules – Introduction, sequential covering algorithms, learning rule sets: summary, learning First-Order rules, learning sets of First-Order rules: FOIL, Induction as inverted deduction, inverting resolution.

Reinforcement Learning – Introduction, the learning task, *Q*–learning, non-deterministic, rewards and actions, temporal difference learning, generalizing from examples, relationship to dynamic programming.

UNIT - V

Analytical Learning-1- Introduction, learning with perfect domain theories: PROLOG-EBG, remarks on explanation-based learning, explanation-based learning of search control knowledge.

Analytical Learning-2-Using prior knowledge to alter the search objective, using prior knowledge to augment search operators.

Combining Inductive and Analytical Learning – Motivation, inductive-analytical approaches to learning, using prior knowledge to initialize the hypothesis.

TEXT BOOKS:

1. Machine Learning – Tom M. Mitchell, - MGH

REFERENCES:

1. Machine Learning: An Algorithmic Perspective, Stephen Marshland, Taylor & Francis.

UNIT - I

Introduction - Well-posed learning problems, designing a learning system, Perspectives and issues in machine learning.

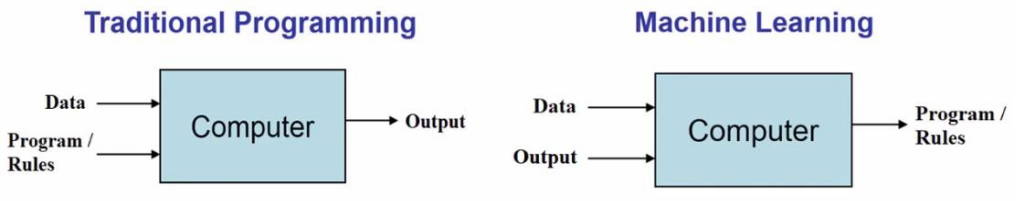
Concept learning and the general to specific ordering – introduction, a concept learning task, concept learning as search, find-S: finding a maximally specific hypothesis, version spaces and the candidate elimination algorithm, remarks on version spaces and candidate elimination, inductive bias**.**

Decision Tree Learning – Introduction, decision tree representation, appropriate problems for decision tree learning, the basic decision tree learning algorithm, hypothesis space search in decision tree learning, inductive bias in decision tree learning, issues in decision tree learning.

INTRODUCTION

MACHINE LEARNING:

Machine Learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (ie., progressively improve performance on a specific task) with the data, without being explicitly programmed.



WELL-POSED LEARNING PROBLEMS:

Definition: A computer program is said to **learn** from experience E with respect to some class of tasks ***T*** and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

A well-defined learning problem requires a well-specified task, performance metric, and source of training experience.

A checkers learning problem:

- **Task T:** playing checkers
- **Performance measure P:** percent of games won against opponents
- **Training experience E:** playing practice games against itself

A handwriting recognition learning problem:

- **Task T:** recognizing and classifying handwritten words within images
- **Performance measure *P:*** percent of words correctly classified
- **Training experience E:** a database of handwritten words with given classifications

A robot driving learning problem: (Autonomous Vehicle)

- **Task T:** driving on public four-lane highways using vision sensors
- **Performance measure P:** average distance traveled before an error (as judged by human supervisor)
- **Training experience E:** a sequence of images and steering commands recorded while observing a human driver

Learning problem	**Task T**	**Performance measure P**	**Training Experience E**
Studying books if human as machine	Getting more marks	40 marks/60 marks/80 marks	MID-I/ MID-II/ Pre-final Examination
Travel	Go to Job place	30 min/20 min/10min	Identifying Speed break, turning and traffic
Playing checkers	Playing against opponents to win the game	Percentage of games won against opponents.	Playing practice games against itself.
Hand written recognition	recognizing and classifying handwritten words within images	percent of words correctly classified	a database of handwritten words with given classifications
Robot Driving	driving on public four-lane highways using vision sensors	average distance traveled before an error (as judged by human supervisor)	A sequence of images and steering commands recorded while observing a human driver

DESIGNING A LEARNING SYSTEM:

Let us consider designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament. We adopt the obvious performance measure: the percent of games it wins in this world tournament. The sequence of steps to design as follows:

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    a. Estimating training values
    b. Adjusting the weights
5. The Final Design

1. Choosing the Training Experience:

- The first design choice is to choose the type of training experience from which our system will learn. (Ex. Driving a car for learner)
- The type of training experience available can have a significant impact on success or failure of the learner. (Ex. Listening audio and getting knowledge and drive a car)

There are **three attributes** which impact on success or failure of the learner:

1. **One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system.**

    In learning to play checkers, the system might learn from ***direct*** training examples consisting of individual checkers board states and the correct move for each. (Ex. Trainer giving instructions to the learner when driving a car/Teaching in class)

    Alternatively, it might have available only ***indirect*** information consisting of the move sequences and final outcomes of various games played. (Ex. Trainer not there. Recorded audio explains the process of driving.)

2. **A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples.**

    The learner might rely on the teacher to select informative board states and to provide the correct move for each. Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move. ***Or*** the learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with no teacher present.

    (Ex. Trainer is telling what are all we have to do like switch on car, put the gear and accelerator in a sequence. If any mistake, trainer will correct.

    We have to drive. Trainer will explain when mistake occur.

    Trainer sit in backside. Trainer would not explain. Learner has to drive the car.

    Teaching in class – Tutorial class :solving when need - Assignment)

3. **A third important attribute of the training experience is how well it represents the distribution of examples over which the final system performance P must be measured.**

In our checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament. If its training experience E consists only of games played against itself, there is an obvious danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested. For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion. Such situations are problematic because mastery of one distribution of examples will not necessary lead to strong performance over some other distribution.

(Ex. Drive only in college ground – Few days
After that You have to tell to drive hill area, High way, traffic area – distribution - Performance will improve.
Textbook – Reference books - Videos)

To proceed with our design, let us decide that our system will train by playing games against itself and allows the system to generate as much training data as time permits. This has the advantage that no external trainer need be present. We now have a fully specified learning task.

2. Choosing the Target Function:

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

Let us begin with a checkers-playing program that can generate the ***legal*** moves from any board state. The program needs only to learn how to choose the ***best*** move from among these legal moves. This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a **priori**(set of legal moves are available, chose best move).

Let us call this function ***ChooseMove*** and use the notation

$$\textbf{\textit{ChooseMove}} : B \rightarrow M$$

to indicate that this function accepts as input any board from the set of legal board states *B* and produces as output some move from the set of legal moves ***M.*** The choice of the target function will therefore be a key design choice.

Let us call this **target function *V*** and again use the notation

$$\boldsymbol{V} : B \rightarrow R$$

to denote that ***V*** maps any legal board state from the set B to some real value (we use *R* to denote the set of real numbers). We intend for this target function ***V*** to assign **higher scores** to better board states. If the system can successfully learn such a target function ***V,*** then it can easily use it to select the **best move** from any current board position. This will make it easier to design a training algorithm.

Let us therefore define the target value ***V(b)*** for an arbitrary board state ***b*** in *B*, ***as*** follows:
1. if ***b*** is a final board state that is won, then ***V(b)*** = 100
2. if b is a final board state that is lost, then ***V(b)*** = -100
3. if b is a final board state that is drawn, then ***V(b)*** = ***0***
4. if b is a not a final state in the game, then V(b) = V(b^l), where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game (assuming the opponent plays optimally, as well).

In fact, we often expect learning algorithms to acquire only some ***approximation*** to the target function, and for this reason the process of learning the target function is often called **f*unction approximation.*** In the current discussion we will use the symbol $\hat{V}$ to refer to the function that is actually learned by our program, to distinguish it from the ideal target function V.

3. Choosing a Representation for the Target Function:

Now that we have specified the ideal target function V, we must choose a representation that the learning program will use to describe the function $\hat{V}$ that it will learn. Let us choose a simple representation:
the function $\hat{V}$ will be calculated as a linear combination of the following board features:

- ***xl:*** the number of black pieces on the board
- ***x2:*** the number of red pieces on the board
- ***x3:*** the number of black kings on the board
- x4**:** the number of red kings on the board
- ***x5:*** the number of black pieces threatened by red
- ***x6:*** the number of red pieces threatened by black

Thus, our learning program will represent $\hat{V}$ (b) as a linear function of the Form

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

where

- ***wo*** through ***w6*** are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights ***w1*** through ***w6*** will determine the relative importance of the various board features in determining the value of the board.
- The weight ***wo*** will provide an additive constant to the board value.

To summarize our design choices thus far, we have

Partial design of a checkers learning program:

- **Task T:** playing checkers
- **Performance measure *P:*** percent of games won in the world tournament
- **Training experience E:** games played against itself
- **Target function:** V: Board $\longrightarrow$ R
- **Target function representation:**

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

4.Choosing a Function Approximation Algorithm:

- In order to learn the target function $\hat{V}$ we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b.
- Each training example is an ordered pair of the form (b, $V_{train}(b)$).
- For instance, the following training example describes a board state b in which black has won the game (note ***x2*** = 0 indicates that **red** has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore **+100.**

$$\langle\langle x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0\rangle, +100\rangle$$

Function ApproximationProcedures:

a) ESTIMATING TRAINING VALUES:

- A Simple approach for estimating training values for intermediate board states is to assign the training value of $V_{train}(b)$ for any intermediate board state ***b*** to be $\hat{V}$ ***(Successor(b))***
  where $\hat{V}$ is the learner's current approximation to ***V*** and where ***Successor(b)*** denotes the next board state following ***b*** for which it is again the program's turn to move (i.e., the board state following the program's move and the opponent's response). This rule for estimating training values can be summarized as

  Rule for estimating training values.

$$V_{train}(b) \leftarrow \hat{V}(Successor(b))$$

b) ADJUSTING THE WEIGHTS:

Specify the learning algorithm for choosing the weights ***wi*** to best fit the set of training examples ***{ (b,V train(b))}.***

A first step we must define what we mean by the ***best fit*** to the training data.

- One common approach is to define the best hypothesis, or set of weights, as that which minimizes the square error E between the training values and the values predicted by the hypothesis $\hat{V}$ **.**

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{ training examples}} (V_{train}(b) - \hat{V}(b))^2$$

Thus, minimizing the sum of squared errors is equivalent to finding the most probable hypothesis given the observed training data.
The least mean squares (LMS) algorithm or **LMS** training rule is used to minimize the squared error E by adjusting weights in a small amount.

LMS weight update rule.
For each training example *(b,V train(b))*

- Use the current weights to calculate $\hat{V}$ (b)
- For each weight wi, update it as

$$w_i \leftarrow w_i + \eta \ (V_{train}(b) - \hat{V}(b)) \ x_i$$

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.
Working of weight update rule:

- When the error $(V_{train}(b) - \hat{V}(b))$ is zero, no weights are changed.
- When $(V_{train}(b) - \hat{V}(b))$ is positive (i.e., when $\hat{V}(b)$ is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}(b)$, reducing the error.
- If the value of some feature x_i is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

5. The Final Design:

The final design of our checkers learning system can be naturally described by four distinct program modules. These four modules, summarized in Figure 1.1, are as follows:

- The **Performance System** is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output. In our case, the strategy used by the Performance System to select its next move at each step is determined by the learned $\hat{V}$ evaluation function.
- The **Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function.

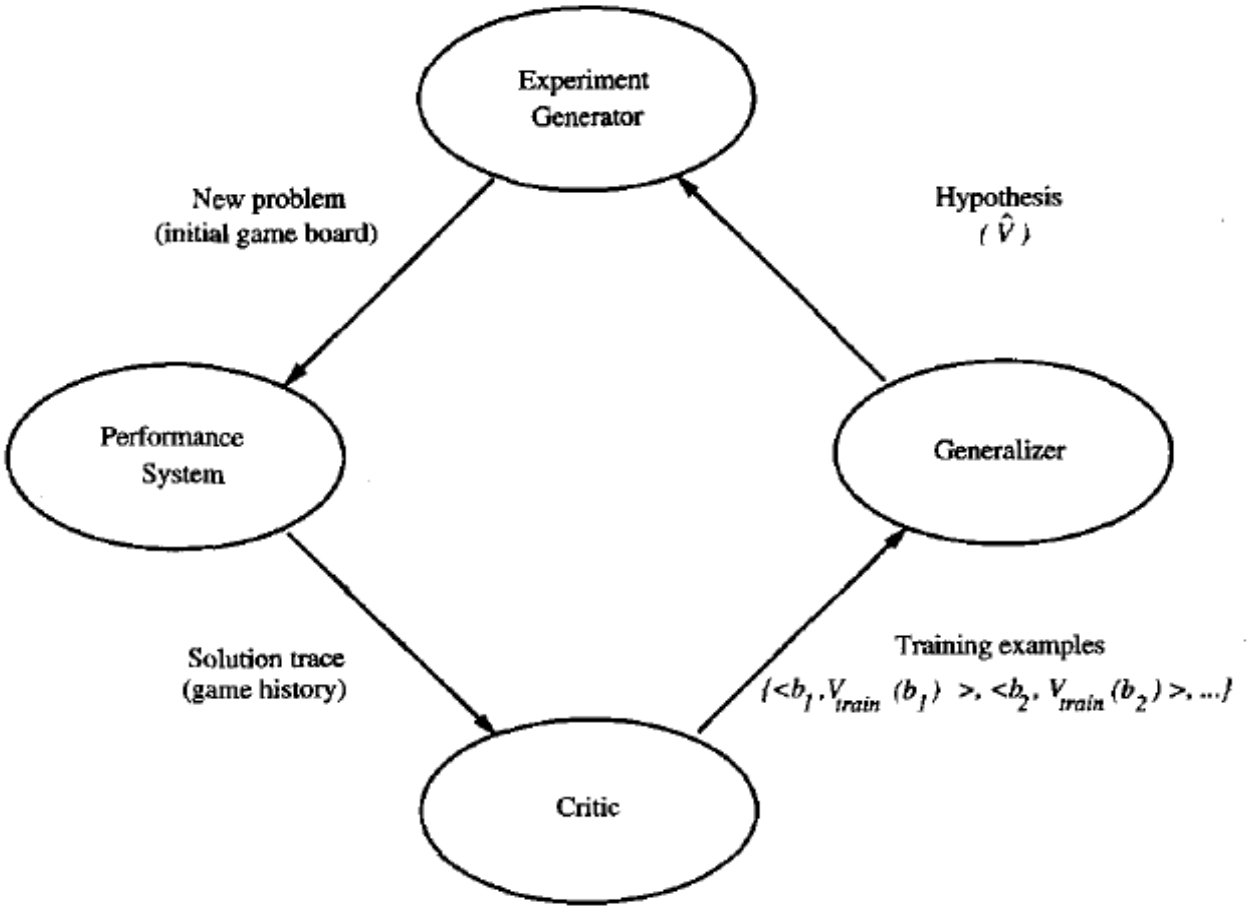


FIGURE 1.1
Final design of the checkers learning program.

- The **Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. In our example, the Generalizer corresponds to the LMS algorithm, and the output hypothesis is the function $\hat{V}$ described by the learned weights ***wo,*** . . . , ***W6.***
- The **Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

The sequence of design choices made for the checkers program is summarized in Figure **1.2.**

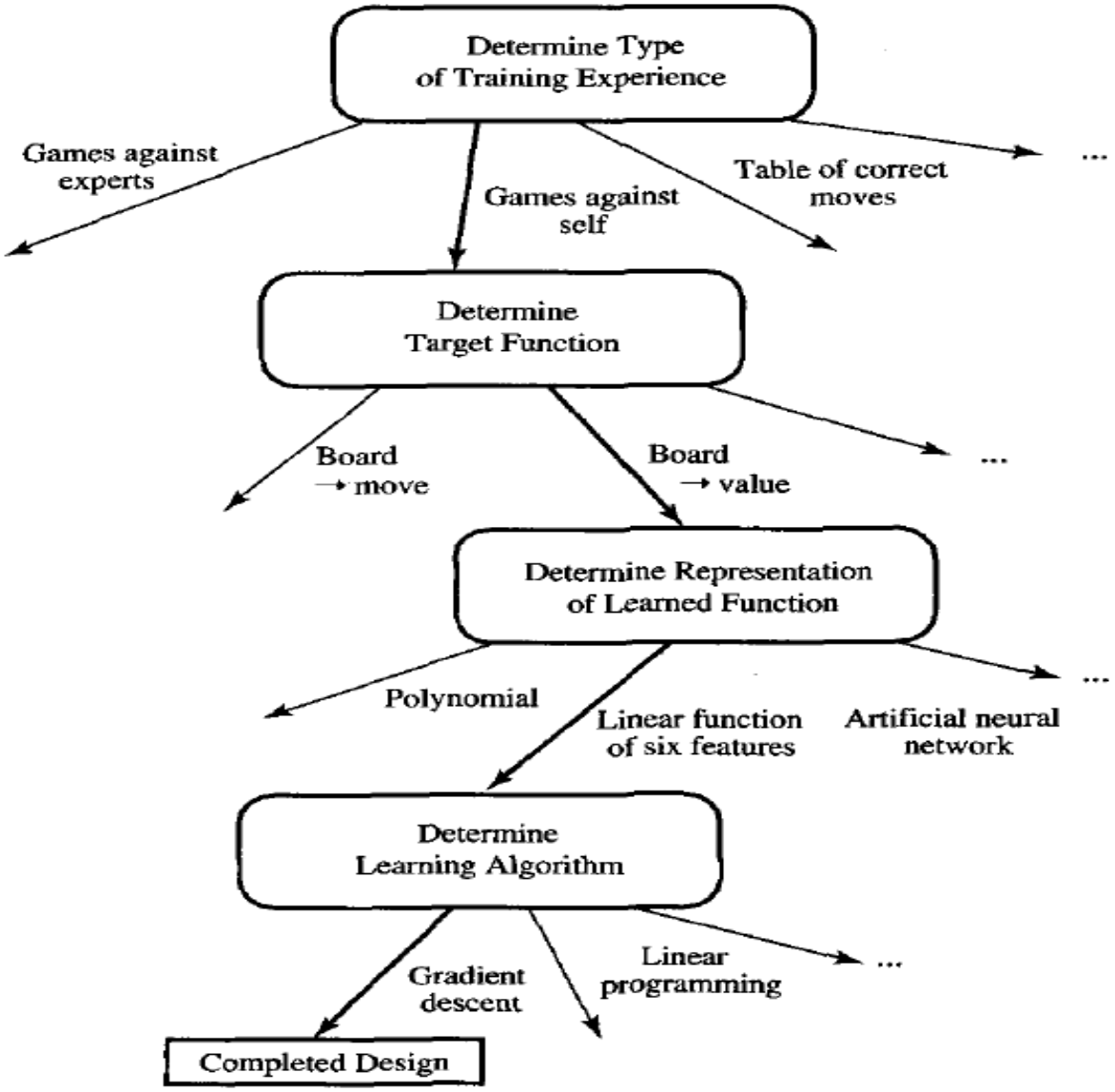


FIGURE 1.2
Summary of choices in designing the checkers learning program..

PERSPECTIVES AND ISSUES IN MACHINE LEARNING:

- One useful perspective on machine learning is that it involves searching a **very large space** of possible hypotheses to determine one that **best fits** the observed data and any prior knowledge held by the learner.
- The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples.
- The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value. This

algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.

- Many of the algorithms that search a hypothesis space defined by some underlying representation (e.g., linear functions, logical descriptions, decision trees, artificial neural networks). These different hypothesis representations are appropriate for learning different kinds of target functions. For each of these hypothesis representations, the corresponding learning algorithm takes advantage of a different underlying structure to organize the search through the hypothesis space.

Issues in Machine Learning:

The field of machine learning is concerned with answering questions such as the following:

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Concept learning and the general to specific ordering

- ✓ Introduction,
- ✓ Concept learning task,
- ✓ Concept learning as search,
- ✓ Find-S and finding a maximally specific hypothesis,
- ✓ Version spaces and the candidate elimination algorithm,
- ✓ Remarks on version spaces and candidate elimination,
- ✓ inductive bias.

CONCEPT LEARNING AND THE GENERAL TO SPECIFIC ORDERING

INTRODUCTION:

Concept learning: Inferring **a** Boolean-valued function from training examples of its input and output.

Much of learning involves acquiring general concepts from specific training examples. People, for example, continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc. Each such concept can be viewed as describing some subset of objects or events defined over a larger set (e.g., the subset of animals that constitute birds – Animal, Vehicle : General - Car, Cat, Dog:Specific). Car is subset of vehicle, cat is subset of Animal.

Alternatively, each concept can be thought of as a boolean-valued function defined over this larger set (e.g., a function defined over all animals, whose value is true for birds and false for other animals).

A CONCEPT LEARNING TASK:

Consider the example task of learning the target concept
"days on which my friend Aldo enjoys his favorite water sport."

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

TABLE 2.1
Positive and negative training examples for the target concept *EnjoySport*.

Table 2.1 describes a set of example days, each represented by a set of ***attributes.*** The attribute ***EnjoySport*** indicates whether or not Aldo enjoys his favorite water sport on this day. The task is to learn to predict the value of ***Enjoy Sport*** for an arbitrary day, based on the values of its other attributes.

Let each hypothesis be a vector of six constraints, specifying the values of the six attributes ***Sky, AirTemp, Humidity, Wind, Water,*** and ***Forecast.*** For each attribute, the hypothesis will either

- indicate by a "?' that any value is acceptable for this attribute,
- specify a single required value (e.g., ***Warm)*** for the attribute, or

- indicate by a *"φ"* that no value is acceptable.

If some instance x satisfies all the constraints of hypothesis h, then h classifies x as a positive example (h(x) = 1).

- The hypothesis that Aldo enjoys his favorite sport only on cold days with high humidity (independent of the values of the other attributes) is represented by the expression (?, ***Cold,*** High, ?, ?, ?)
- The most general hypothesis-that every day is a positive example-is represented by (?, ?, ?, ?, ?, ?)
- The most specific possible hypothesis-that no day is a positive example-is represented by **(*φ*, *φ*, *φ*, *φ*, *φ*, *φ*)**

Notation

- The set of items over which the concept is defined, is called the set of instances, which we denote by X.
- In the current example, X is the set of all possible days, each represented by the attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast.
- The concept or function to be learned is called the target concept, which we denote by c.
- In general, c can be any boolean valued function defined over the instances X; that is, c : X $\rightarrow$ **(*O,*** 1).
- In the current example, the target concept corresponds to the value of the attribute EnjoySport (i.e., c(x) = 1 if EnjoySport = Yes, and c(x) = **0** if EnjoySport = No).

- **Given:**
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values *Sunny*, *Cloudy*, and *Rainy*),
    - *AirTemp* (with values *Warm* and *Cold*),
    - *Humidity* (with values *Normal* and *High*),
    - *Wind* (with values *Strong* and *Weak*),
    - *Water* (with values *Warm* and *Cool*), and
    - *Forecast* (with values *Same* and *Change*).
  - Hypotheses H: Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), "Ø" (no value is acceptable), or a specific value.
  - Target concept c: $EnjoySport : X \rightarrow \{0, 1\}$
  - Training examples D: Positive and negative examples of the target function (see Table 2.1).
- **Determine:**
  - A hypothesis h in H such that $h(x) = c(x)$ for all x in X.

TABLE 2.2
The *EnjoySport* concept learning task.

- When learning the target concept, the learner is presented a set of ***training examples,*** each consisting of an instance ***x*** from X, along with its target concept value ***c(x)*** (e.g., the training examples in Table 2.1).
- Instances for which ***c(x)*** = 1 are called ***positive examples,*** or members of the target concept.
- Instances for which ***C(X)*** = 0 are called ***negative examples,*** or nonmembers of the target concept.
- Ehe ordered pair ***(x, c(x))*** describe the training example consisting of the instance **x** and its target concept value ***c(x).***
- The symbol ***D*** used to denote the set of available training examples.
- The symbol H used to denote the set of ***all possible hypotheses.***
- Each hypothesis ***h*** in H represents a boolean-valued function defined over X; that is, ***h*** : X $\longrightarrow${0, 1). The goal of the learner is to find a hypothesis ***h*** such that ***h(x)*** = ***c(x)*** for all ***x*** in X.

The Inductive Learning Hypothesis

Inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data. Lacking any further information, our assumption is that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data.

The inductive learning hypothesis: Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

CONCEPT LEARNING AS SEARCH:

- Concept learning can be viewed as the task of searching through a large space of hypotheses.
- The goal of this search is to find the hypothesis that best fits the training examples.

Consider, for example, the instances X and hypotheses H in the ***EnjoySport*** learning task. Given that the attribute ***Sky*** has three possible values, and that ***AirTemp, Humidity, Wind, Water,*** and ***Forecast*** each have two possible values, the instance space X contains exactly 3.2.2.2.2.2 = 96 distinct instances. **A** similar calculation shows that there are 5.4.4.4.4.4 = 5120 (by ? + φ) syntactically distinct hypotheses within H.

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Every hypothesis containing one or more " φ " symbols represents the empty set of instances; that is, it classifies every instance as negative. Therefore, the number of semantically distinct hypotheses is only 1+ **(4.3.3.3.3.3) = 973. ie,** 1 by φ + *add of ?*

If we view learning as a search problem, then the learning algorithms will examine the different strategies for searching very large or infinite hypothesis spaces, to find the hypotheses that best fit the training data.

General-to-Specific Ordering of Hypotheses:

Algorithms for concept learning organize the search through the hypothesis space by relying on a very useful structure**: a general-to-specific ordering of hypotheses**.

To illustrate the general-to-specific ordering, consider the two hypotheses

h1 = (Sunny, ?, ?, Strong, ?, ?)

h2 = (Sunny, ?, ?, ?, ?, ?)

Now consider the sets of instances that are classified positive by hl and by h2. Because h2 imposes fewer constraints on the instance, it classifies more instances as positive. In fact, any instance classified positive by hl will also be classified positive by h2. Therefore, we say that h2 is **more general than(** $\geq_g$)hl.

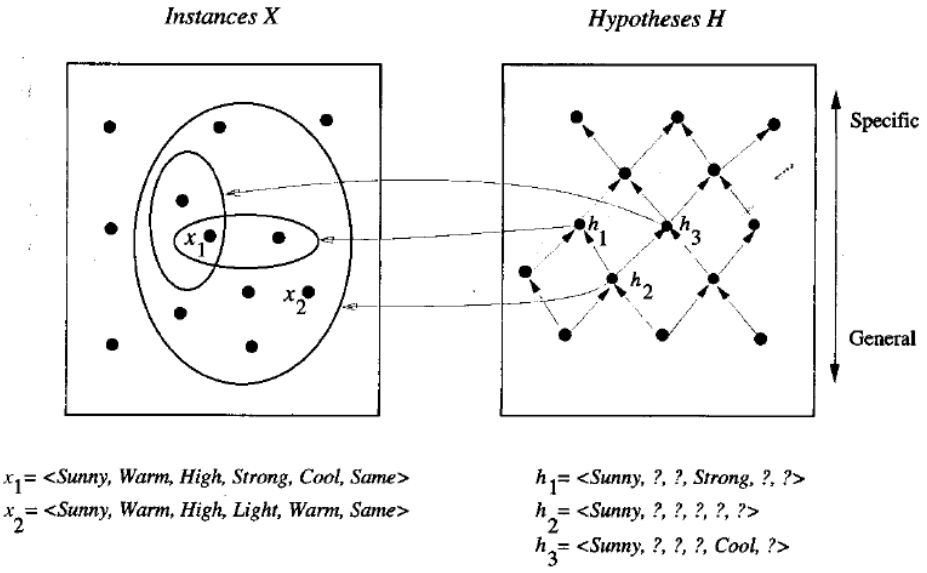


FIGURE 2.1

Instances, hypotheses, and the *more_general_than* relation. The box on the left represents the set X of all instances, the box on the right the set H of all hypotheses. Each hypothesis corresponds to some subset of X—the subset of instances that it classifies positive. The arrows connecting hypotheses represent the *more_general_than* relation, with the arrow pointing toward the less general hypothesis. Note the subset of instances characterized by h_2 subsumes the subset characterized by h_1, hence h_2 is *more_general_than* h_1.

Definition: Let h_j and h_k be boolean-valued functions defined over X. Then h_j is **more_general_than_or_equal_to** h_k (written $h_j \geq_g h_k$) if and only if

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

We will also find it useful to consider cases where one hypothesis is strictly more general than the other. As noted earlier, hypothesis ***h2*** is more general than ***hl*** because every instance that satisfies ***hl*** also satisfies ***h2.*** Similarly, ***h2*** is more general than ***h3.*** If some instance x satisfies all the constraints of hypothesis h, then h classifies x as a positive example (h(x)=1).

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS:

How can we use the *more-general-than* partial (as opposed to total) ordering to organize the search for a hypothesis consistent with the observed training examples?

One way is to begin with the most specific possible hypothesis in H, then generalize this hypothesis each time it fails to cover an observed positive training example. To be more precise about how the partial ordering is used, consider the FIND-S algorithm defined in Table 2.3.

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   - For each attribute constraint a_i in h
     If the constraint a_i is satisfied by x
     Then do nothing
     Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

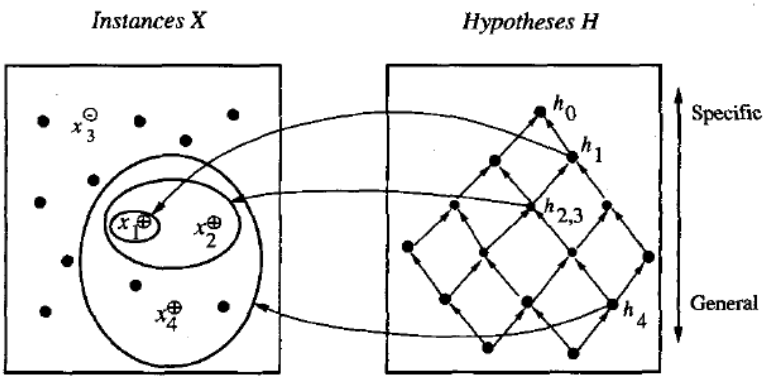
TABLE 2.3
FIND-S Algorithm.

To illustrate this algorithm, assume the learner is given the sequence of training examples from Table 2.1 for the *EnjoySport* task.

- The first step of FINDS is to initialize *h* to the most specific hypothesis in H

$$h \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

- In particular, none of the "φ" constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example; namely, the attribute values for this training example.



x_1 = <Sunny Warm Normal Strong Warm Same>, +
x_2 = <Sunny Warm High Strong Warm Same>, +
x_3 = <Rainy Cold High Strong Warm Change>, -
x_4 = <Sunny Warm High Strong Cool Change>, +

h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <Sunny Warm Normal Strong Warm Same>
h_2 = <Sunny Warm ? Strong Warm Same>
h_3 = <Sunny Warm ? Strong Warm Same>
h_4 = <Sunny Warm ? Strong ? ? >

FIGURE 2.2
The hypothesis space search performed by FIND-S. The search begins (h_0) with the most specific hypothesis in H, then considers increasingly general hypotheses (h_1 through h_4) as mandated by the training examples. In the instance space diagram, positive training examples are denoted by "+," negative by "–," and instances that have not been presented as training examples are denoted by a solid circle.

This *h* is still very specific; the second training example is positive in this case and forces the algorithm to further generalize *h*, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example. Upon encountering the third training example-in this case a negative example- the algorithm makes no change to *h.* In fact, the FIND-S algorithm simply *ignores every negative example.*

To complete our trace of FIND-S, the fourth (positive) example leads to a further generalization of **h ⟵ (Sunny, Warm, ?, Strong, ?, ?)**

The FIND-S algorithm illustrates one way in which the ***more-general than*** partial ordering can be used to organize the search for an acceptable hypothesis.

The search moves from hypothesis to hypothesis, searching from the most specific to progressively more general hypotheses along one chain of the partial ordering. Figure 2.2 illustrates this search in terms of the instance and hypothesis spaces.

At each step, the hypothesis is generalized only as far as necessary to cover the new positive example. Therefore, at each stage the hypothesis is the most specific hypothesis consistent with the training examples observed up to this point (hence the name FIND-S).

FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples.

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM:

- This section describes a second approach to concept learning, the CANDIDATE ELIMINATION algorithm, that addresses several of the limitations of FIND-S.
- Notice that although FIND-S outputs a hypothesis from H, that is consistent with the training examples, this is just one of many hypotheses from H that might fit the training data equally well. The key idea in the CANDIDATE ELIMINATION algorithm is to output a description of the set of ***all hypotheses consistent with the training examples.***

2.5.1 Representation

Definition: A hypothesis h is **consistent** with a set of training examples *D* if and only if ***h(x)*** = ***c(x)*** for each example (x, ***c(x))*** in ***D.***

$$Consistent(h, D) \equiv (\forall \langle x, c(x) \rangle \in D)\ h(x) = c(x)$$

Notice the key difference between this definition of ***consistent*** and our earlier definition of ***satisfies.***

An example *x* is said to ***satisfy*** hypothesis ***h*** when ***h(x)*** = 1, regardless of whether x is a positive or negative example of the target concept. However, whether such an example is ***consistent*** with ***h*** depends on the target concept, and in particular, whether ***h(x)*** = ***c(x).***

The CANDIDATE ELIMINATION algorithm represents the set of *all* hypotheses consistent with the observed training examples. This subset of all hypotheses is called the **version space** with respect to the hypothesis space H and the training examples D.

Definition: The **version space**, denoted $VS_{H,D}$, with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with the training examples in D.

$$VS_{H,D} \equiv \{h \in H | Consistent(h, D)\}$$

The LIST-THEN-ELIMINATlON Algorithm:

- The LIST-THEN-ELIMINAION Algorithm first initializes the version space to contain all hypotheses in H, then eliminates any hypothesis found inconsistent with any training example.

The LIST-THEN-ELIMINATE Algorithm

1. $VersionSpace \leftarrow$ a list containing every hypothesis in H
2. For each training example, $\langle x, c(x) \rangle$
   remove from $VersionSpace$ any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in $VersionSpace$

TABLE 2.4
The LIST-THEN-ELIMINATE algorithm.

2.5.3 A More Compact Representation for Version Spaces

- The version space is represented by its most general and least general members.
- These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

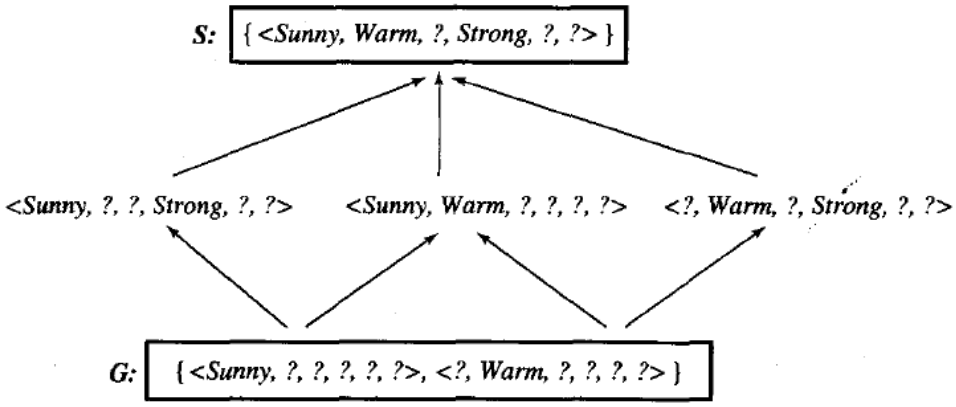


FIGURE 2.3
A version space with its general and specific boundary sets. The version space includes all six hypotheses shown here, but can be represented more simply by S and G. Arrows indicate instances of the *more_general_than* relation. This is the version space for the *EnjoySport* concept learning problem and training examples described in Table 2.1.

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

To illustrate this representation for version spaces, consider again the En*joysport* concept learning problem described in Table 2.2. Recall that given the four training examples from Table 2.1, FIND-S outputs the hypothesis *h = (Sunny, Warm,* ?, *Strong,* ?, ?)

In fact, this is just one of six different hypotheses from H that are consistent with these training examples. All six hypotheses are shown in Figure 2.3. They constitute the version space relative to this set of data and this hypothesis representation.

The arrows among these six hypotheses in Figure 2.3 indicate instances of the *more-general than* relation.

The CANDIDATE-ELIMINATlON algorithm represents the version space by storing only its most general members (labeled G in Figure 2.3) and its most specific (labeled *S* in the figure).

Definition: The **general boundary** G, with respect to hypothesis space H and training data D, is the set of maximally general members of H consistent with D.

$$G \equiv \{g \in H | Consistent(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge Consistent(g', D)]\}$$

Definition: The **specific boundary** S, with respect to hypothesis space H and training data D, is the set of minimally general (i.e., maximally specific) members of H consistent with D.

$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

CANDIDATE-ELIMINATION Algorithm:

- The CANDIDATE-ELIMINATION Algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.
- It begins by initializing the version space to the set of all hypotheses in H; that is, by initializing the ***G* boundary set** to contain the **most general hypothesis** in H

  $G_o \leftarrow \{<?, ?, ?, ?, ?, ?>\}$

  and initializing the ***S* boundary set** to contain the **most specific (least general) hypothesis** $S_0 \leftarrow \{ <\varphi, \varphi, \varphi, \varphi, \varphi, \varphi>\}$

Eliminate from the version space if any hypotheses found inconsistent with the training examples. This algorithm is summarized in Table 2.5.

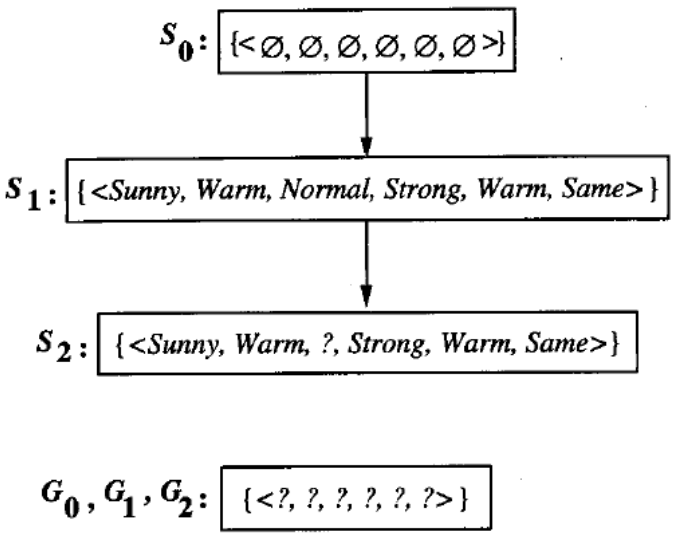
Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

TABLE 2.5
CANDIDATE-ELIMINATION algorithm using version spaces. Notice the duality in how positive and negative examples influence S and G.

An Illustrative Example:

Figure 2.4 traces the CANDIDATE-ELIMINATION Algorithm applied to the first two training examples from Table 2.1. As described above, the boundary sets are first initialized to G_o and S_o, the most general and most specific hypotheses in H, respectively.



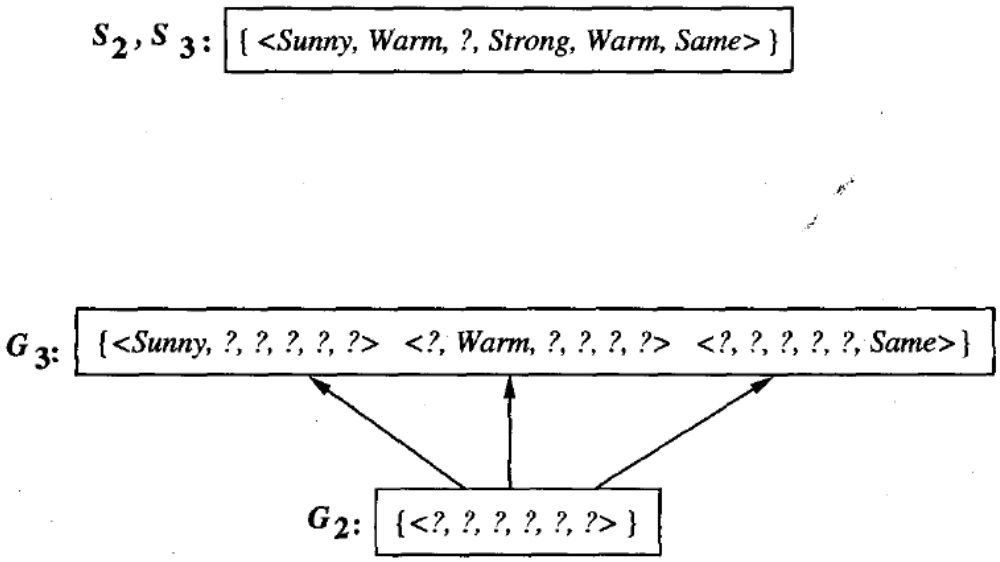
Training examples:

1. <*Sunny, Warm, Normal, Strong, Warm, Same*>, *Enjoy Sport = Yes*

2. <*Sunny, Warm, High, Strong, Warm, Same*>, *Enjoy Sport = Yes*

FIGURE 2.4
CANDIDATE-ELIMINATION Trace 1. S_0 and G_0 are the initial boundary sets corresponding to the most specific and most general hypotheses. Training examples 1 and 2 force the S boundary to become more general, as in the FIND-S algorithm. They have no effect on the G boundary.

- When the first training example is presented (a positive example in this case), the CANDIDATE-ELIMINATION Algorithm checks the *S* boundary and finds that it is overly specific. The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example. This revised boundary is shown as S_1 in Figure 2.4.
- No update of the G boundary is needed in response to this training example because G_o correctly covers this example. When the second training example (also positive) is observed, it has a similar effect of generalizing ***S*** further to $\boldsymbol{S_2}$, leaving G again unchanged (i.e., $G_2 = \mathbf{G_1} = G_0$). Notice the processing of these first two positive examples is very similar to the processing performed by the FIND-S algorithm.
- Consider the third training example, shown in Figure 2.5. This negative example reveals that the **G** boundary of the version space is very general; that is, the hypothesis in **G** incorrectly predicts that this new example is a positive example. The hypothesis in the **G** boundary must therefore be specialized until it correctly classifies this new negative example.

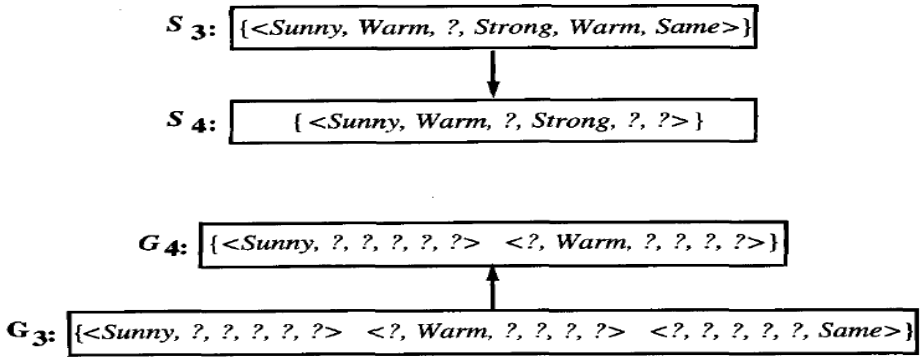


Training Example:

3. *<Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No*

FIGURE 2.5
CANDIDATE-ELIMINATION Trace 2. Training example 3 is a negative example that forces the G_2 boundary to be specialized to G_3. Note several alternative maximally general hypotheses are included in G_3.

- The fourth training example, as shown in Figure 2.6, further generalizes the S boundary of the version space.



Training Example:

4. *<Sunny, Warm, High, Strong, Cool, Change>, EnjoySport = Yes*

FIGURE 2.6
CANDIDATE-ELIMINATION Trace 3. The positive training example generalizes the S boundary, from S_3 to S_4. One member of G_3 must also be deleted, because it is no longer more general than the S_4 boundary.

- After processing these four examples, the boundary sets **S4** and **G4** delimit the version space of all hypotheses consistent with the set of incrementally observed training examples. The entire version space, including those hypotheses bounded by **S4** and **G4,** is shown in Figure 2.7.

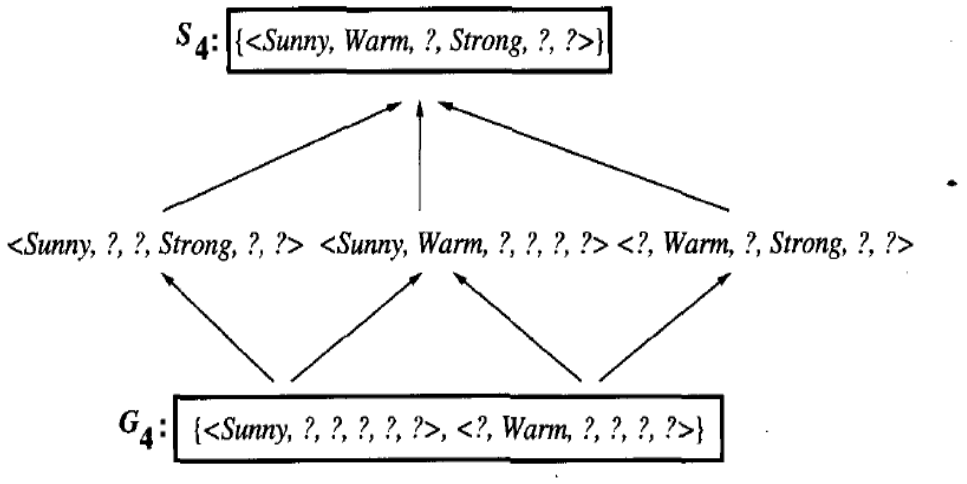


FIGURE 2.7
The final version space for the *EnjoySport* concept learning problem and training examples described earlier.

REMARKS ON VERSION SPACES AND CANDIDATE-ELIMINATION:

Will the CANDIDATE-ELIMINATION Algorithm Converge to the Correct Hypothesis?

The version space learned by the **CANDIDATE-ELIMINATION Algorithm** will converge toward the hypothesis that correctly describes the target concept, provided

(1) There are no errors in the training examples, and
(2) There is some hypothesis in H that correctly describes the target concept.

What Training Example Should the Learner Request Next?

Up to this point we have assumed that training examples are provided to the learner by some external teacher. Suppose instead that the learner is allowed to conduct experiments in which it chooses the next instance, then obtains the correct classification for this instance from an external oracle (e.g., nature or a teacher).

How Can Partially Learned Concepts Be Used?

Suppose that no additional training examples are available beyond the four in our example above, but that the learner is now required to classify new instances that it has not yet observed. Even though the version space of Figure **2.3** still contains multiple hypotheses, indicating that the target concept has not yet been fully learned, it is possible to classify

certain examples with the same degree of confidence as if the target concept had been uniquely identified. To illustrate, suppose the learner is asked to classify the four new instances shown in Table **2.6.**

Instance	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
A	Sunny	Warm	Normal	Strong	Cool	Change	?
B	Rainy	Cold	Normal	Light	Warm	Same	?
C	Sunny	Warm	Normal	Light	Warm	Same	?
D	Sunny	Cold	Normal	Strong	Warm	Same	?

TABLE 2.6
New instances to be classified.

- The learner can classify instance A as positive with the same confidence it would have if it had already converged to the single, correct target concept.
- Similarly, instance B is classified as a negative instance by every hypothesis in the version space.
- Instance C presents a different situation. Half of the version space hypotheses classify it as positive and half classify it as negative. Thus, the learner cannot classify this example with confidence until further training examples are available.
- Finally, instance D is classified as positive by two of the version space hypotheses and negative by the other four hypotheses. In this case we have less confidence in the classification than in the unambiguous cases of instances A and B. Still, the vote is in favor of a negative classification.
- Furthermore, the proportion of hypotheses voting positive can be interpreted as the **probability** that this instance is positive given the training data.

INDUCTIVE BIAS:

The constraints on the Hypothesis space are called **Inductive bias**.

The fundamental questions for inductive reference:

1. What if the target concept is not contained in the hypothesis space?
2. Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis?
3. How does the size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances?
4. How does the size of the hypothesis space influence the number of training examples that must be observed?

1. A Biased Hypothesis Space (Effect of incomplete hypothesis space)

Consider again the ***EnjoySpor***t example which represent target function

"Sky = ***Sunny*** or ***Sky*** = ***Cloudy."***

In fact, given the following three training examples of this disjunctive hypothesis, our algorithm would find that there are zero hypotheses in the version space.

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Cool	Change	Yes
2	Cloudy	Warm	Normal	Strong	Cool	Change	Yes
3	Rainy	Warm	Normal	Strong	Cool	Change	No

To see why there are no hypotheses consistent with these three examples, note that the most specific hypothesis consistent with the first two examples ***and representable in the given hypothesis space H*** is ***S2*** : (?, ***Warm, Normal, Strong, Cool, Change)***

This hypothesis, although it is the maximally specific hypothesis from H that is consistent with the first two examples, is already overly general: it incorrectly covers the third (negative) training example. The problem is that we have biased the learner to consider only conjunctive hypotheses. In this case we require a more expressive hypothesis space.

2. An Unbiased Learner (Incomplete hypothesis space)

- If c is not in H, then consider generalizing representation of H to contain c.
- The size of the instance space X of days described by the six available attributes is 96. Recall that there are 96 instances in EnjoySport; hence there are 296 possible hypothesis space H.
- Can do this by using full propositional calculus with AND, OR, NOT
- Hence H defined only by conjunctions of attributes is biased (containing only 973 h's)
- Let us reformulate the ***Enjoysport*** learning task in an unbiased way by defining a new hypothesis space ***H'*** that can represent every subset of instances; that is, let H' correspond to the power set of X.
- One way to define such an H' is to allow arbitrary disjunctions, conjunctions, and negations of our earlier hypotheses.

For instance, the target concept ***"Sky*** = ***Sunny*** or ***Sky*** = ***Cloudy"*** could then be described as

h = (Sunny, ?, ?, ?, ?, ?) V ***(Cloudy,*** ?, ?, ?, ?, ?)

$$S : \{(x_1 \vee x_2 \vee x_3)\}$$

$$G : \{\neg(x_4 \vee x_5)\}$$

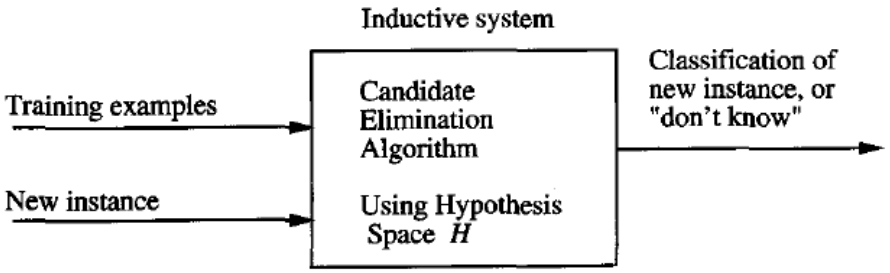
3. The Futility of Bias - Free Learning

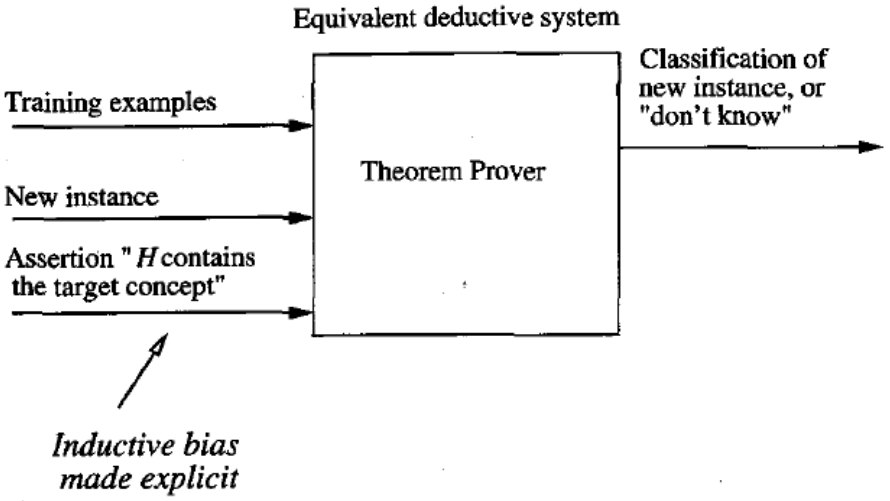
Definition:

Consider a concept Learning algorithm L for the set of instances X.

- Let c be an arbitrary concept defined over X
- **Let** $D_c = \{(x, c(x))\}$ be an arbitrary set of training examples of c.
- **Let L**(x_i, D_c) denote the classification assigned to the instance x_i by L after training on the data D_c.
- The inductive bias of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$





DECISION TREE LEARNING:

- ✓ Introduction,
- ✓ decision tree representation,
- ✓ appropriate problems for decision tree learning,
- ✓ the basic decision tree learning algorithm,
- ✓ hypothesis space search in decision tree learning,
- ✓ inductive bias in decision tree learning,
- ✓ issues in decision tree learning.

INTRODUCTION:

- Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.
- Learned trees can also be re-represented as sets of if-then rules to improve human readability.

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*** illustrates a typical learned decision tree. This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis.

For example, the instance

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

would be sorted down the leftmost branch of this decision tree and would therefore be classified as a negative instance (i.e., the tree predicts that ***PlayTennis = no).*** This tree and the example used in Table ***3.2***

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*.

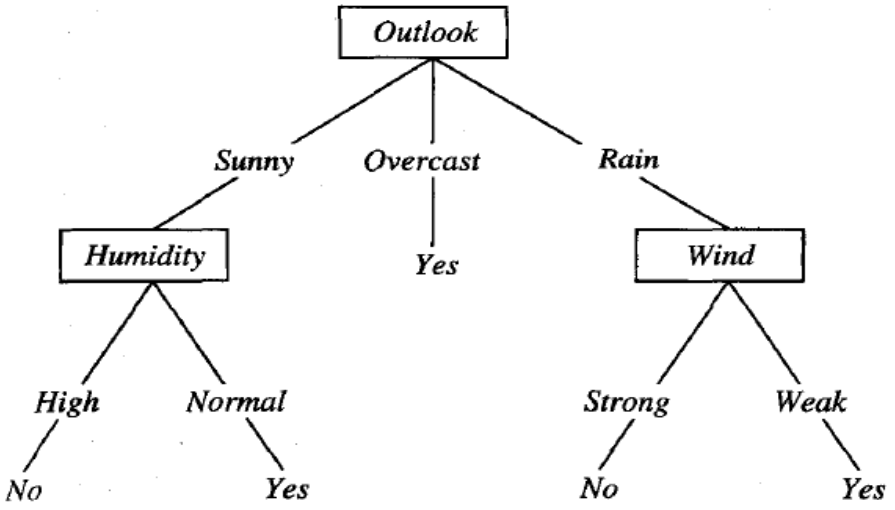


FIGURE 3.1
A decision tree for the concept *PlayTennis*. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf (in this case, *Yes* or *No*). This tree classifies Saturday mornings according to whether or not they are suitable for playing tennis.

- 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.

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

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).**
- **The target function has discrete output values.** The decision tree in Figure 3.1 assigns a boolean classification (e.g., **yes** or **no)** to each example.
- **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).

Decision tree learning has therefore 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:

The basic algorithm for decision tree learning is ID3 algorithm which learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?' To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples.

The best attribute is selected and used as the test at the root node of the tree. A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute).

The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.

A simplified version of the algorithm, specialized to learning boolean-valued functions (i.e., concept learning), is described in Table 3.1.

Which Attribute Is the Best Classifier?

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

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

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

In all calculations involving entropy we define 0 log 0 to be 0.

To illustrate, 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 of data). 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 $p_{\ominus}$ 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.

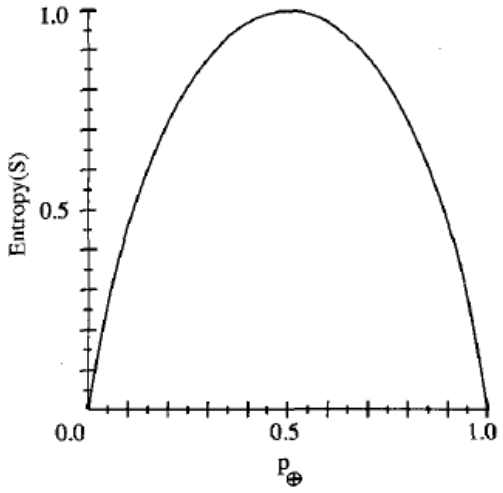


FIGURE 3.2
The entropy function relative to a boolean classification, as the proportion, $p_{\oplus}$, of positive examples varies between 0 and 1.

Figure 3.2 shows the form of the entropy function relative to a boolean classification, as $p_{\oplus}$ varies between **0** and 1.

The entropy of ***S*** is defined as

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

where ***pi*** is the proportion of ***S*** belonging to class ***i.***

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

* The best attribute is the one with highest *information gain*, as defined in Equation (3.4).

TABLE 3.1
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.

INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY:

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

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

where

- ***Values(A)*** is the set of all possible values for attribute A
- S_v is the subset of S for which attribute A has value v.
- The first term in Equation is just the entropy of the original collection ***S***
- The second term is the expected value of the entropy after ***S*** is partitioned using attribute ***A.***

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

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

Information gain is precisely the measure used by ID3 to select the best attribute at each step in growing the tree. The use of information gain to evaluate the relevance of attributes is summarized in Figure ***3.3.*** In this figure the information gain of two different attributes, ***Humidity*** and ***Wind,*** is computed in order to determine which is the better attribute for classifying the training examples shown in Table ***3.2.***

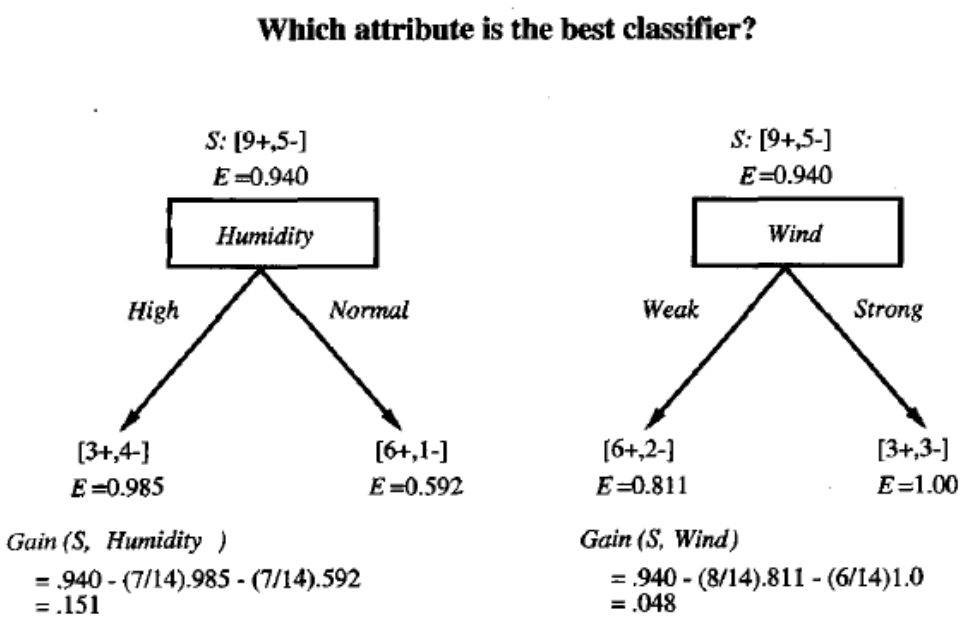


FIGURE 3.3
Humidity provides greater information gain than *Wind*, relative to the target classification. Here, E stands for entropy and S for the original collection of examples. Given an initial collection S of 9 positive and 5 negative examples, [9+, 5−], sorting these by their *Humidity* produces collections of [3+, 4−] ($Humidity = High$) and [6+, 1−] ($Humidity = Normal$). The information gained by this partitioning is .151, compared to a gain of only .048 for the attribute *Wind*.

An Illustrative Example

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.

Consider the first step through the algorithm, in which the topmost node of the decision tree is created. 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 3.3. The information gain values for all four attributes are

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

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

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

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

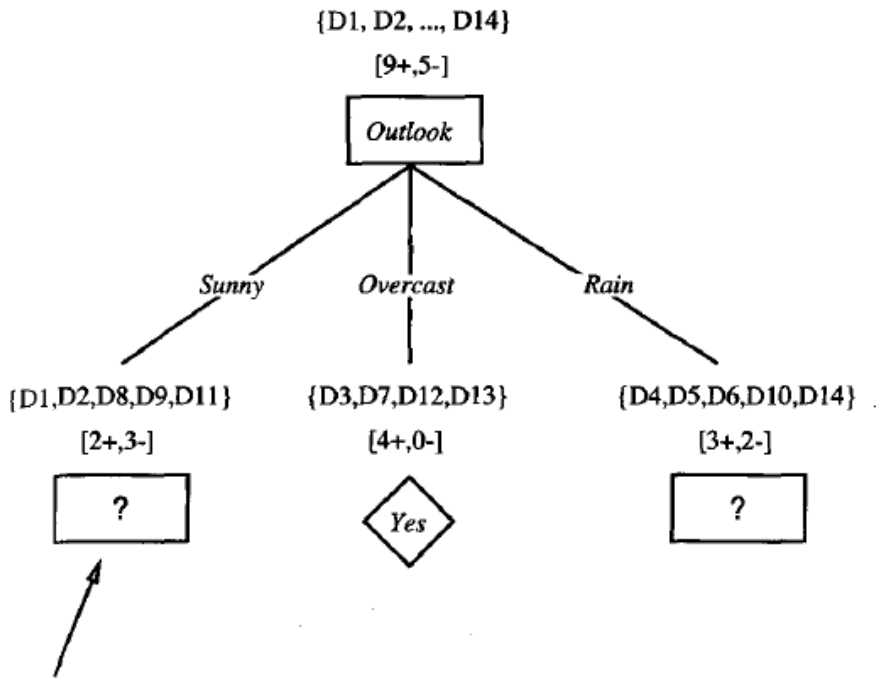
where ***S*** denotes the collection of training examples from Table 3.2.

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*.

- 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 3.4, along with the training examples sorted to each new descendant node.

- 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.



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

FIGURE 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.

This 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., their entropy is zero).

The final decision tree learned by ID3 from the 14 training examples of Table 3.2 is shown in Figure 3.1.

HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING:

- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples.
- 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.
- The evaluation function that guides this hill-climbing search is the information gain measure. This search is depicted in Figure 3.5.

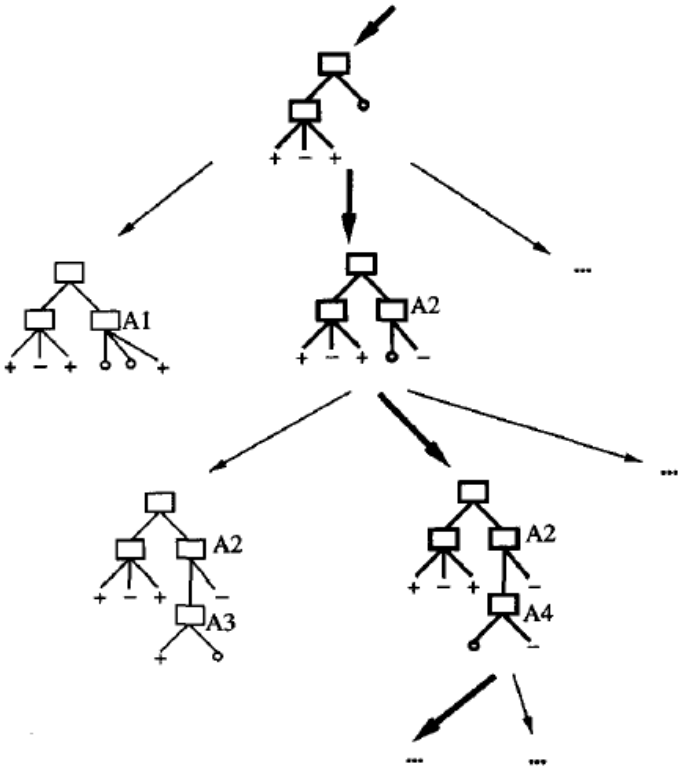


FIGURE 3.5
Hypothesis space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the information gain heuristic.

Capabilities and limitations By viewing **ID3** in terms of its search space and search strategy are

- **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.

- **ID3** maintains only 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.
- **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.
- **ID3** uses all training examples at each step 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.

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.
- Inductive bias of ID3 therefore consists of describing the basis by which it chooses one of these consistent hypotheses over the others.
- **A closer approximation to the inductive bias of ID3:**
  Shorter trees are preferred over longer trees. Selects trees that place high information gain attributes close to the root.

Restriction Biases and Preference Biases

- There is an interesting difference between the types of inductive bias exhibited by ID3 and by the CANDIDATE-ELIMINATlON Algorithm.
- ID3 searches incompletely through this space, from simple to complex hypotheses, until its termination condition is met (e.g., until it finds a hypothesis consistent with the data). Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy.
- The inductive bias of ID3 is thus a preference for certain hypotheses over others (e.g., for shorter hypotheses), with no hard restriction on the hypotheses that can be eventually enumerated. This form of bias is typically called a preference bias (or, alternatively, a search bias).
- The bias of the CANDIDATEELIMINATION 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, alternatively, a language 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.

- 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.

Why Prefer Short Hypotheses?

- Occam's razor: Prefer the simplest hypothesis that fits the data.
- There are fewer short hypotheses than long ones (based on straightforward combinatorial arguments), it is less likely that one will find a short hypothesis that coincidentally fits the training data.
- Why should we believe that the small set of hypotheses consisting of decision trees with ***short descriptions*** should be any more relevant than the multitude of other small sets of hypotheses that we might define?
- A second problem with the above argument for Occam's razor is that the size of a hypothesis is determined by the particular representation used ***internally*** by the learner.
- This last argument shows that Occam's razor will produce two different hypotheses from the same training examples when it is applied by two learners that perceive these examples in terms of different internal representations.
- Occam's razor supports the scenario that examines the question of which internal representations might arise from a process of evolution and natural selection.

ISSUES IN DECISION TREE LEARNING: (ASSIGNMENT-1)

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. Below we discuss each of these issues and extensions to the basic ID3 algorithm that address them.

1. Avoiding Overfitting the Data

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.

Example, the ID3 algorithm is applied to the task of learning which medical patients have a form of diabetes. 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, whereas the broken line shows accuracy measured over an independent

set of test examples (not included in the training set). Predictably, the accuracy of the tree over the training examples increases monotonically as the tree is grown. However, the accuracy measured over the independent test examples first increases, then decreases. 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.

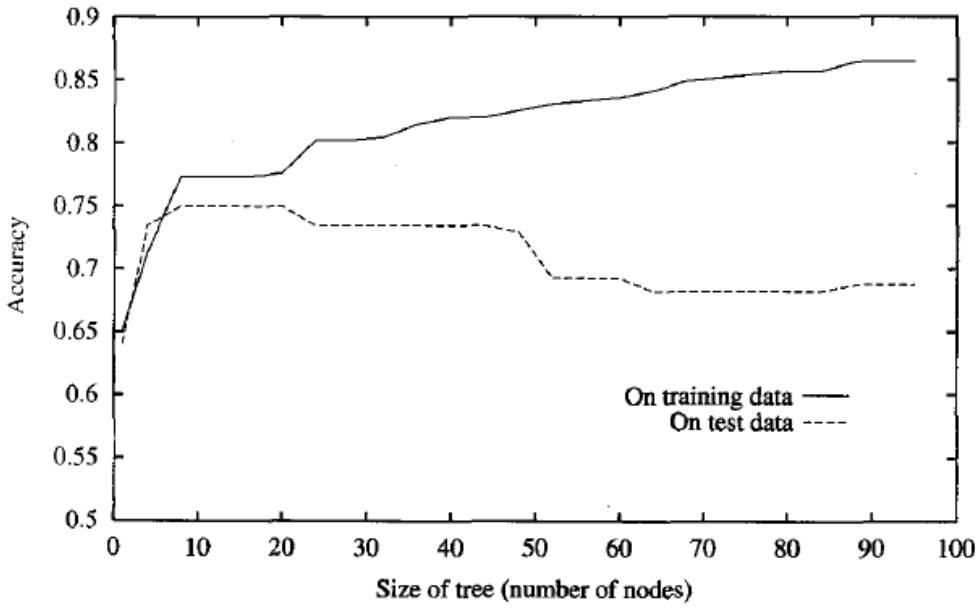


FIGURE 3.6
Overfitting in decision tree learning. As ID3 adds new nodes to grow the decision tree, the accuracy of the tree measured over the training examples increases monotonically. However, when measured over a set of test examples independent of the training examples, accuracy first increases, then decreases. Software and data for experimenting with variations on this plot are available on the World Wide Web at http://www.cs.cmu.edu/~tom/mlbook.html.

Approaches to avoid over fitting are

- 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.

A key question is what criterion is to be 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 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.

1.1 REDUCED ERROR 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.

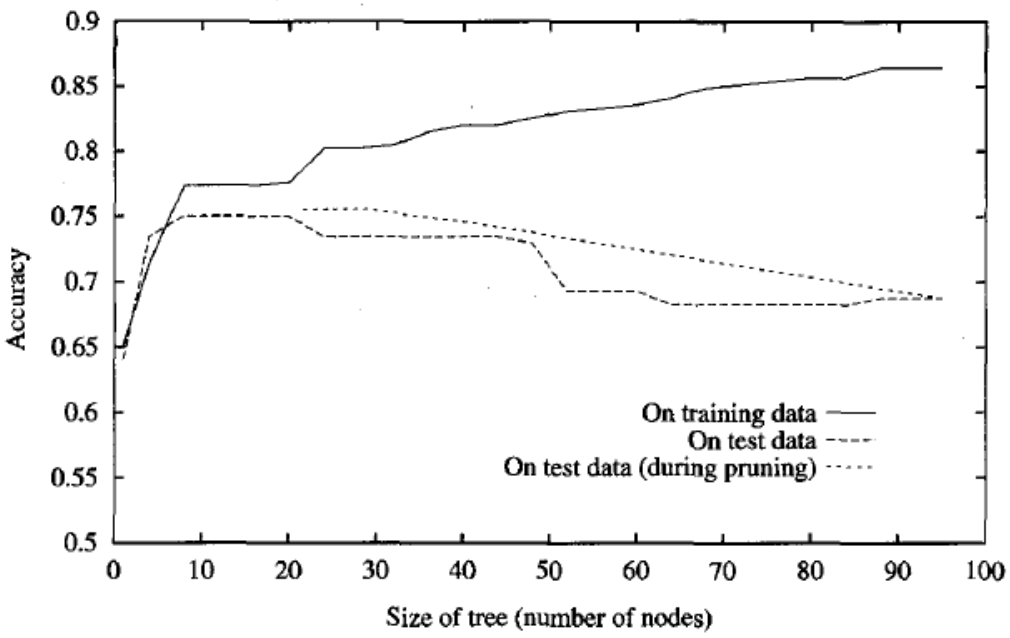


FIGURE 3.7
Effect of reduced-error pruning in decision tree learning. This plot shows the same curves of training and test set accuracy as in Figure 3.6. In addition, it shows the impact of reduced error pruning of the tree produced by ID3. Notice the increase in accuracy over the test set as nodes are pruned from the tree. Here, the validation set used for pruning is distinct from both the training and test sets.

1.2 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.

To illustrate, consider again the decision tree in Figure 3.1. In rule postpruning, one rule is generated for each leaf node in the tree. Each attribute test along the path from the root to the leaf becomes a rule antecedent (precondition) and the classification at the leaf node becomes the rule consequent (postcondition). For example, the leftmost path of the tree in Figure 3.1 is translated into the rule

IF (Outlook = Sunny) A (Humidity = High)
THEN PlayTennis = No

Advantages of converting decision tree to rules before pruning are:

There are three main advantages.

- Converting to rules allows distinguishing among the different contexts in which a decision node is used. if the tree itself were pruned, the only two choices would be to remove the decision node completely, or to retain it in its original form.
- 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, we 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

- 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 learned tree must be discrete valued.
- Second, the attributes tested in the decision nodes of the tree must also be discrete valued. This second restriction can easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree.
- This can be accomplished by dynamically defining new discretevalued 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, that is true if A < c and false otherwise.

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

3. 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.

4. Handling Training Examples with Missing Attribute Values

- It is common to estimate the missing attribute value based on other examples for which this attribute has a known value.
- 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.***
- **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).***

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 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 Gpin 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.

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