Data Mining

Lesson 2

Inductive Learning

MSc in Computer Science

University of New York Tirana

Assoc. Prof. Dr. Marenglen Biba

Data Mining: Content

- Introduction to data mining and machine learning
- Inductive learning
- Decision trees
- Rule induction
- Instance-based learning
- Bayesian learning
- Neural networks
- Support vector machines
- Other machine learning models
- Engineering data mining tasks

Class Outline

- PART I: Concept Learning and Inductive Bias
- PART II: Practical Data Mining: Introduction to Concepts, Instances, and Attributes
  - Weka
  - Oracle Data Miner

PART I

Concept Learning

Intro to Inductive Learning

- The problem of inducing general functions from specific training examples is central to learning.
- In this lecture we will consider concept learning: acquiring the definition of a general category given a sample of positive and negative training examples of the category.
- Concept learning can be formulated as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples.
- In many cases this search can be efficiently organized by taking advantage of a naturally occurring structure over the hypothesis space — a general-to-specific ordering of hypotheses.

Concepts

- *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.
- 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).

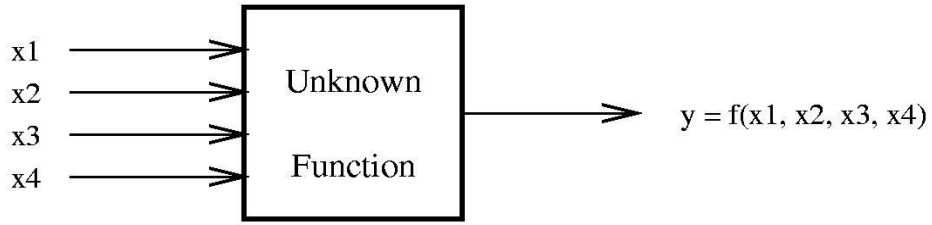
Concept Learning

- In this lecture we consider the problem of automatically inferring the general definition of some concept, given examples labeled as members or nonmembers of the concept.

- This task is commonly referred to as concept learning, or approximating a boolean-valued function from examples.

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

A Learning Problem



Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Slide from P. Domingos UW-CSE

Supervised Learning

- **Given:** Training examples $\langle \mathbf{x}, f(\mathbf{x}) \rangle$ for some unknown function f.
- **Find:** A good approximation to f.

Example Applications

- **Credit risk assessment**

  $\mathbf{x}$: Properties of customer and proposed purchase.

  $f(\mathbf{x})$: Approve purchase or not.

- **Disease diagnosis**

  $\mathbf{x}$: Properties of patient (symptoms, lab tests)

  $f(\mathbf{x})$: Disease (or maybe, recommended therapy)

- **Face recognition**

  $\mathbf{x}$: Bitmap picture of person's face

  $f(\mathbf{x})$: Name of the person.

- **Automatic Steering**

  $\mathbf{x}$: Bitmap picture of road surface in front of car.

  $f(\mathbf{x})$: Degrees to turn the steering wheel.

Slide from P. Domingos UW-CSE

Appropriate Applications for Supervised Learning

- **Situations where there is no human expert**

  $\mathbf{x}$: Bond graph for a new molecule.

  $f(\mathbf{x})$: Predicted binding strength to AIDS protease molecule.

- **Situations where humans can perform the task but can't describe how they do it.**

  $\mathbf{x}$: Bitmap picture of hand-written character

  $f(\mathbf{x})$: Ascii code of the character

- **Situations where the desired function is changing frequently**

  $\mathbf{x}$: Description of stock prices and trades for last 10 days.

  $f(\mathbf{x})$: Recommended stock transactions

- **Situations where each user needs a customized function f**

  $\mathbf{x}$: Incoming email message.

  $f(\mathbf{x})$: Importance score for presenting to user (or deleting without presenting).

Slide from P. Domingos UW-CSE

Example: A concept learning task

- Let’s start our discussion of concept learning, with an example of learning the target concept "days on which one can play water sport."
- We have 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 EnjoySport for an arbitrary day, based on the values of its other attributes.

Positive and Negative Training Examples

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*.

Hypothesis representation

- What hypothesis representation shall we provide to the learner in this case?
- Let us begin by considering a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.
- In particular, 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 "0" that no value is acceptable.

Examples satisfying hypothesis

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

  (?, ?, ?, ?, ?, ?)
- and the most specific possible hypothesis — that no day is a positive example — is represented by

  (0, 0, 0, 0,0, 0).

Defining a Concept Learning Task

- To summarize, the EnjoySport concept learning task requires learning the set of days for which EnjoySport = yes, describing this set by a conjunction of constraints over the instance attributes.
- In general, any concept learning task can be described by:
  - the set of instances over which the target function is defined
  - the target function
  - the set of candidate hypotheses considered by the learner
  - and the set of available training examples.

Defining a Concept Learning task

- **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.

Hypothesis space

- Given a set of training examples of the target concept c, the problem faced by the learner is to hypothesize or estimate c.
- We use the symbol H to denote the set of all possible hypotheses or hypothesis space that the learner may consider regarding the identity of the target concept.
- Usually H is determined by the human designer's choice of hypothesis representation.
- In general, each hypothesis h in H represents a boolean-valued function defined over X, that is, h : X -> {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 Assumption

- Notice that although the learning task is to determine a hypothesis h identical to the target concept c over the entire set of instances X, the only information available about c is its value over the training examples.
- Therefore, 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.
- This is the fundamental assumption of inductive learning, and we will have much more to say about it throughout this course.

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 implicitly defined by the hypothesis representation.

- The goal of this search is to find the hypothesis that best fits the training examples.

Number of hypothesis

- It is important to note that by selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn.
- 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 Air Temp, 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 syntactically distinct hypotheses within H.
- Our EnjoySport example is a very simple learning task, with a relatively small, finite hypothesis space.
- Most practical learning tasks involve much larger, sometimes infinite, hypothesis spaces.

General-to-Specific Ordering of Hypotheses

- Many algorithms for concept learning organize the search through the hypothesis space by relying on a very useful structure that exists for any concept learning problem: a general-to-specific ordering of hypotheses.

- By taking advantage of this naturally occurring structure over the hypothesis space, we can design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis.

General-to-specific ordering

- 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 h1 and by h2.
- Because h2 imposes fewer constraints on the instance, it classifies more instances as positive.
- In fact, any instance classified positive by h1 will also be classified positive by h2.
- Therefore, we say that h2 is more general than h1.

Th “more general than" relationship

- The intuitive "more general than" relationship between hypotheses can be defined more precisely as follows.
- First, for any instance x in X and hypothesis h_i in H, we say that x satisfies h if and only if h(x) = 1.
- We now define the more-general-than-or-equal-to relation in terms of the sets of instances that satisfy the two hypotheses:
  - Given hypotheses h_j and h_k, h_j is more-general-than-or-equal-to h_k if and only if any instance that satisfies h_k also satisfies h_j.

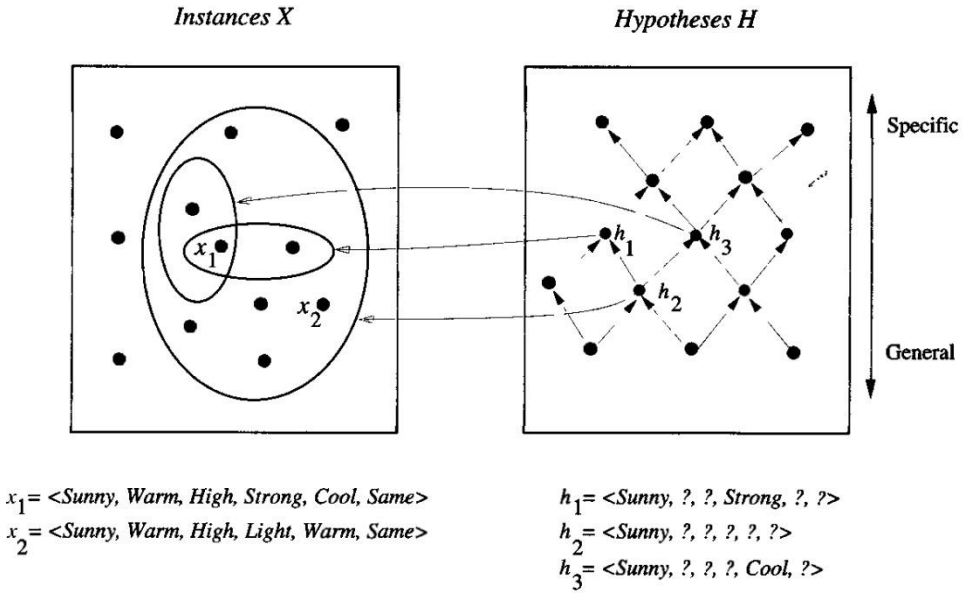
Definition of more-general-than

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 $h_j =>_g$ hk) if and only if

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

Instances and hypothesis



- 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 h2 subsumes the subset characterized by h1, hence h2 is more-general-than h1.

Finding a Maximally Specific Hypothesis

- *Important: How can we use the more-general-than partial 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.
- We say that a hypothesis "covers" a positive example if it correctly classifies the example as positive.

The Find-S Algorithm

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.

Illustration of Find-S

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*.

Illustration of Find-S

- To illustrate this algorithm, assume the learner is given the sequence of training examples from Table 2.1 for the Enjoy Sport task.
- The first step of Find-S is to initialize h to the most specific hypothesis in H

  h <- (0, 0, 0, 0,0, 0)
- Upon observing the first training example from Table 2.1, which happens to be a positive example, it becomes clear that our hypothesis is too specific.
- In particular, none of the "0" 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.
- h <- {Sunny, Warm, Normal, Strong, Warm, Same)

Illustration of Find-S

- This h is still very specific; it asserts that all instances are negative except for the single positive training example we have observed.
- Next, the second training example (also positive in this case) 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.
- The refined hypothesis in this case is:

  h <— {Sunny, Warm, ?, Strong, Warm, Same).

Illustration of Find-S

- 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.
- While this may at first seem strange, notice that in the current case our hypothesis h is already consistent with the new negative example (i.e., h correctly classifies this example as negative), and hence no revision is needed.

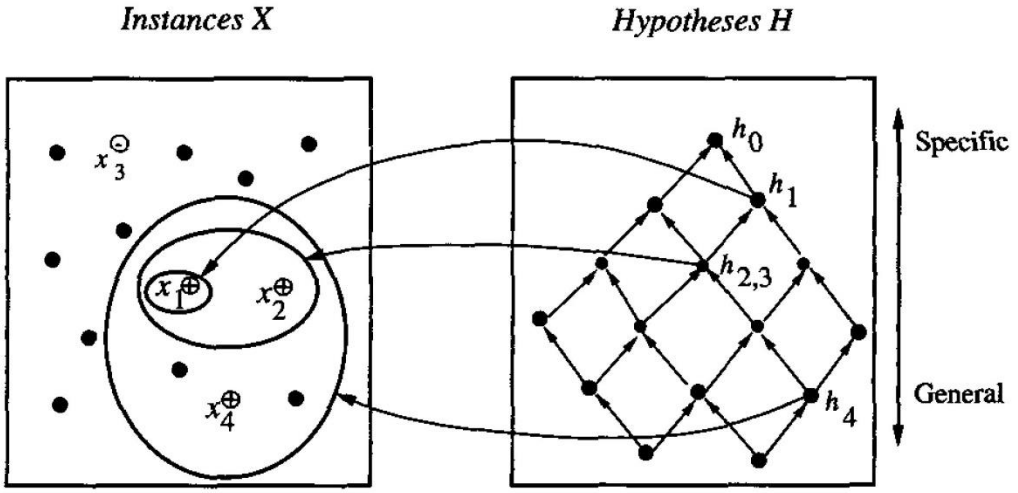
- *In the general case, as long as we assume that the hypothesis space H contains a hypothesis that describes the true target concept c and that the training data contains no errors, then the current hypothesis h can never require a revision in response to a negative example*.

Illustration of Find-S

- To complete our trace of Find-S, the fourth (positive) example leads to a further generalization of h:

  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.
- 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).

Hypothesis Search Space performed by Find-S



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.

Properties of Find-S

- The key property of the Find-S algorithm is that for hypothesis spaces described by conjunctions of attribute constraints (such as H for the EnjoySport task), Find-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples.
- Its final hypothesis will also be consistent with the negative examples provided:
  - the correct target concept is contained in H
  - the training examples are correct.
- However, there are several questions still left unanswered by this learning algorithm, such as:

Convergence of Find-S

- Has the learner converged to the correct target concept?

- Although Find-S will find a hypothesis consistent with the training data, it has no way to determine:
  - whether it has found the only hypothesis in H consistent with the data (i.e., the correct target concept), or
  - whether there are many other consistent hypotheses as well.

- We would prefer a learning algorithm that could determine whether it had converged and,
  - if not, at least characterize its uncertainty regarding the true identity of the target concept.

Preference of the most specific hypothesis

- Why prefer the most specific hypothesis?

- In case there are multiple hypotheses consistent with the training examples, Find-S will find the most specific.

- It is unclear whether we should prefer this hypothesis over, say, the most general, or some other hypothesis of intermediate generality.

Consistence of Training Examples

- Are the training examples consistent?
- In most practical learning problems there is some chance that the training examples will contain at least some errors or noise.
- Such inconsistent sets of training examples can severely mislead Find-S, given the fact that it ignores negative examples.
- We would prefer an algorithm that could at least detect when the training data is inconsistent and, preferably, accommodate such errors.

Are there more specific hypothesis?

- What if there are several maximally specific consistent hypotheses?
- In the hypothesis language H for the EnjoySport task, there is always a unique, most specific hypothesis consistent with any set of positive examples.
- However, for other hypothesis spaces there can be several maximally specific hypotheses consistent with the data.
- In this case, Find-S must be extended to allow it to backtrack on its choices of how to generalize the hypothesis, to accommodate the possibility that the target concept lies along a different branch of the partial ordering than the branch it has selected.
- Furthermore, we can define hypothesis spaces for which there is no maximally specific consistent hypothesis.

Version Spaces And The Candidate-Elimination Algorithm

The Candidate-Elimination algorithm

- The Candidate-Elimination algorithm 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.
- Surprisingly, the Candidate-Elimination algorithm computes the description of this set without explicitly enumerating all of its members.
- This is accomplished by again using the more-general-than partial ordering, this time to maintain a compact representation of the set of consistent hypotheses and to incrementally refine this representation as each new training example is encountered.

Noisy data

- The Candidate-Elimination algorithm has been applied to problems such as learning regularities in chemical mass spectroscopy (Mitchell 1979) and learning control rules for heuristic search (Mitchell et al. 1983).
- Nevertheless, practical applications of the Candidate-Elimination and Find-S algorithms are limited by the fact that they both perform poorly when given noisy training data.
- More importantly for our purposes here, the Candidate-Elimination algorithm provides a useful conceptual framework for introducing several fundamental issues in machine learning.
- In the remainder of this lecture we present the algorithm and discuss these issues.
- In the next lessons, we will examine learning algorithms that are used more frequently with noisy training data.

Definition of hypothesis consistence

- The Candidate-Elimination algorithm finds all describable hypotheses that are consistent with the observed training examples.
- In order to define this algorithm precisely, we begin with a few basic definitions.
- First, let us say that a hypothesis is consistent with the training examples if it correctly classifies these examples.
- 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.

Version space

- 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, because it contains all plausible versions of the target concept.
- *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-Eliminate Algorithm

- The List-Then-Eliminate algorithm first initializes the version space to contain all hypotheses in H, then eliminates any hypothesis found inconsistent with any training example.
- The version space of candidate hypotheses thus shrinks as more examples are observed, until ideally just one hypothesis remains that is consistent with all the observed examples.
- This, presumably, is the desired target concept.
- If insufficient data is available to narrow the version space to a single hypothesis, then the algorithm can output the entire set of hypotheses consistent with the observed data.

The List-Then-Eliminate Algorithm

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.

The List-Then-Eliminate Algorithm

- In principle, the List-Then-Eliminate algorithm can be applied whenever the hypothesis space H is finite.

- It has many advantages, including the fact that it is guaranteed to output all hypotheses consistent with the training data.

- Unfortunately, it requires exhaustively enumerating all hypotheses in H — an unrealistic requirement for all but the most trivial hypothesis spaces.

A More Compact Representation for Version Spaces

- The Candidate-Elimination algorithm works on the same principle as the above List-Then-Eliminate algorithm.
- However, it employs a much more compact representation of the version space.
- In particular, 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.

Boundary Sets

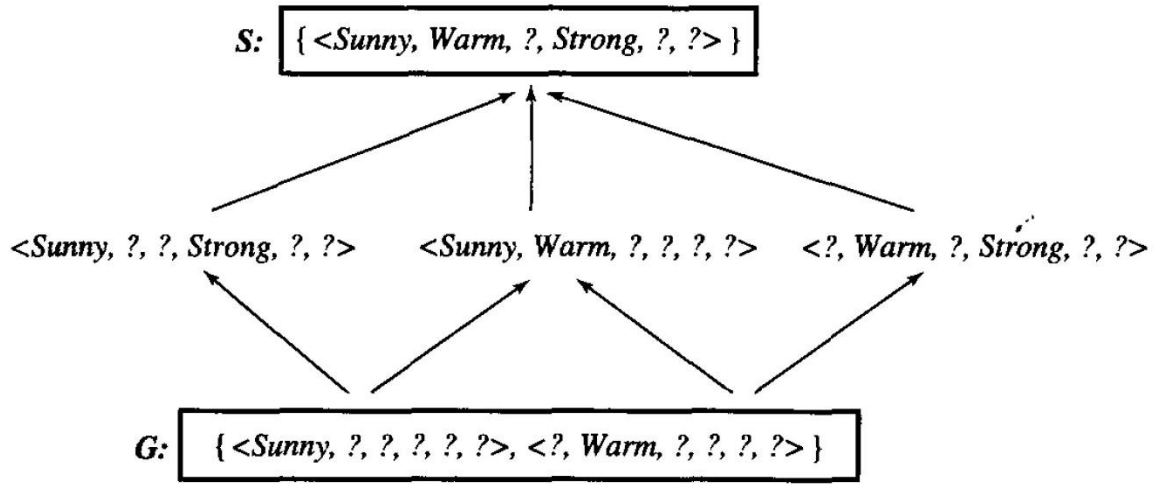


FIGURE 2.3
A version space with its general and specific boundary sets. The version space includes all six hypotheses shown here, but can be represented more simply by S and G. Arrows indicate instances of the *more_general_than* relation. This is the version space for the *EnjoySport* concept learning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

Example of version spaces

- To illustrate this representation for version spaces, consider again the EnjoySport 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 (previous slide)
- 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.

Version space of CE algorithm

- The Candidate-Elimination algorithm represents the version space by storing only its most general members (labeled G) and its most specific (labeled S).

- *Given only these two sets S and G, it is possible to enumerate all members of the version space as needed by generating the hypotheses that lie between these two sets in the general-to-specific partial ordering over hypotheses.*

Definition of boundaries

- *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.

- *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.

Composition of the version space

- As long as the sets G and S are well defined, they completely specify the version space.

- In particular, we can show that the version space is precisely the set of hypotheses contained in G, plus those contained in S, plus those that lie between G and S in the partially ordered hypothesis space.

Theorem of the version space

- **Theorem 2.1.** Version space representation theorem.
- Let X be an arbitrary set of instances and let H be a set of boolean-valued hypotheses defined over X. Let c : X -> {0,1} be an arbitrary target concept defined over X, and let D be an arbitrary set of training examples {{x, c(x))}.
- For all X, H, c, and D such that S and G are well defined,

$$VS_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s)\}$$

Candidate-Elimination Learning 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

$$G_0 \leftarrow \{(?, ?, ?, ?, ?, ?)\}$$

- and initializing the S boundary set to contain the most specific (least general) hypothesis

$$S_0 \leftarrow \{(0,0,0,0, 0,0)\}$$

Candidate-Elimination Learning Algorithm

- These two boundary sets delimit the entire hypothesis space, because every other hypothesis in H is both more general than S_0 and more specific than G_0.
- As each training example is considered, the S and G boundary sets are generalized and specialized, respectively, to eliminate from the version space any hypotheses found inconsistent with the new training example.
- After all examples have been processed, the computed version space contains all the hypotheses consistent with these examples and only these hypotheses.

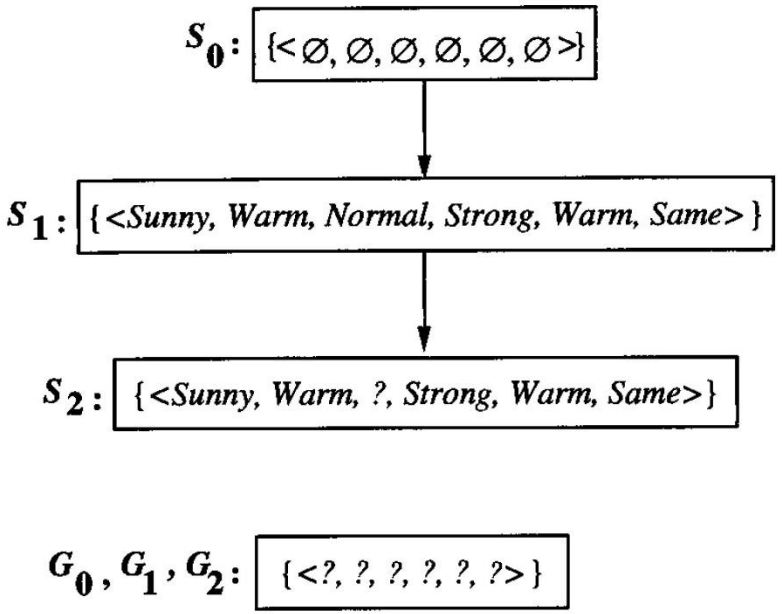
Candidate-Elimination Learning Algorithm

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.

Example



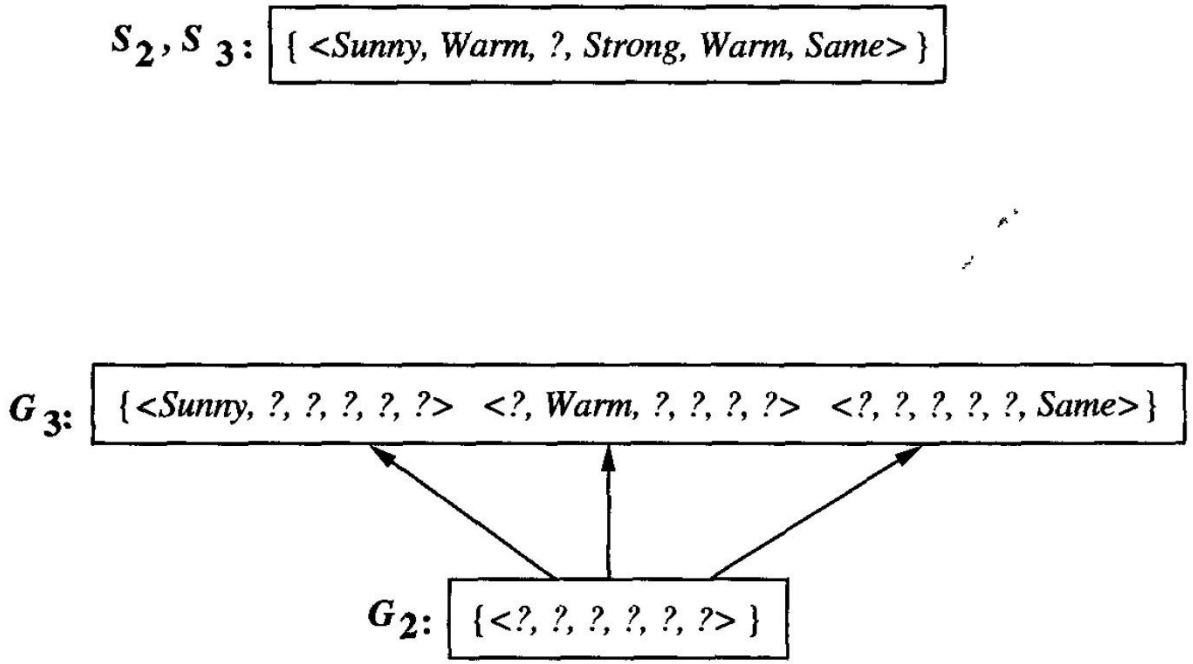
Training examples:

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.

Example

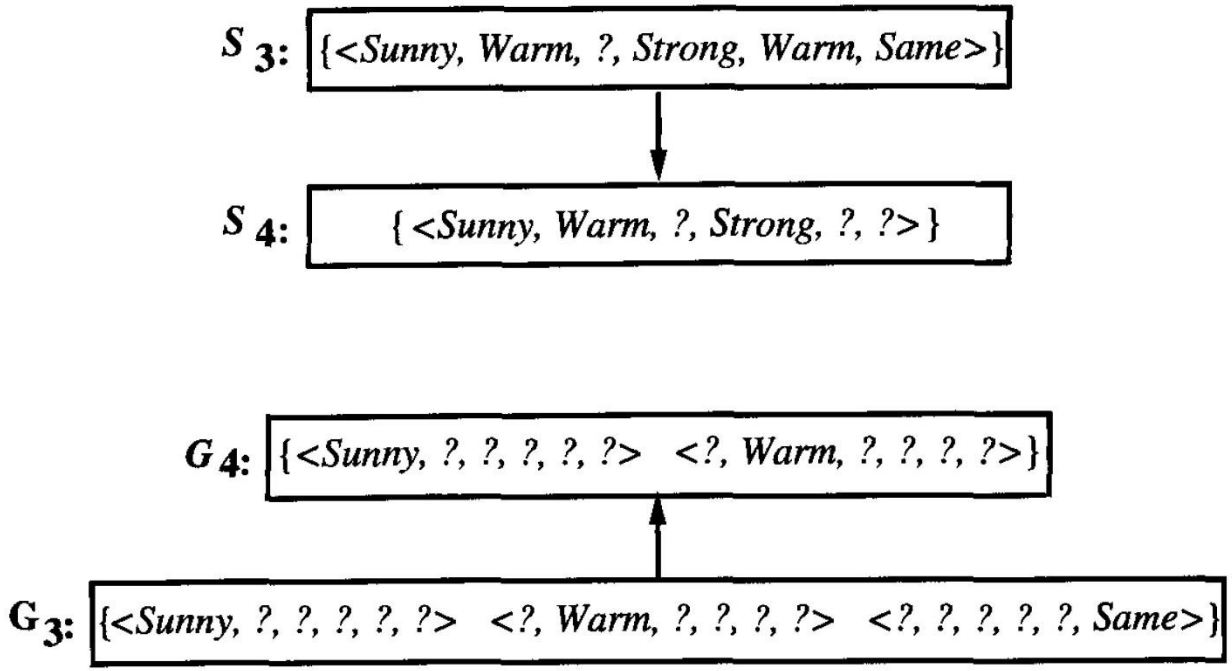


Training Example:

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.

Example



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.

Example

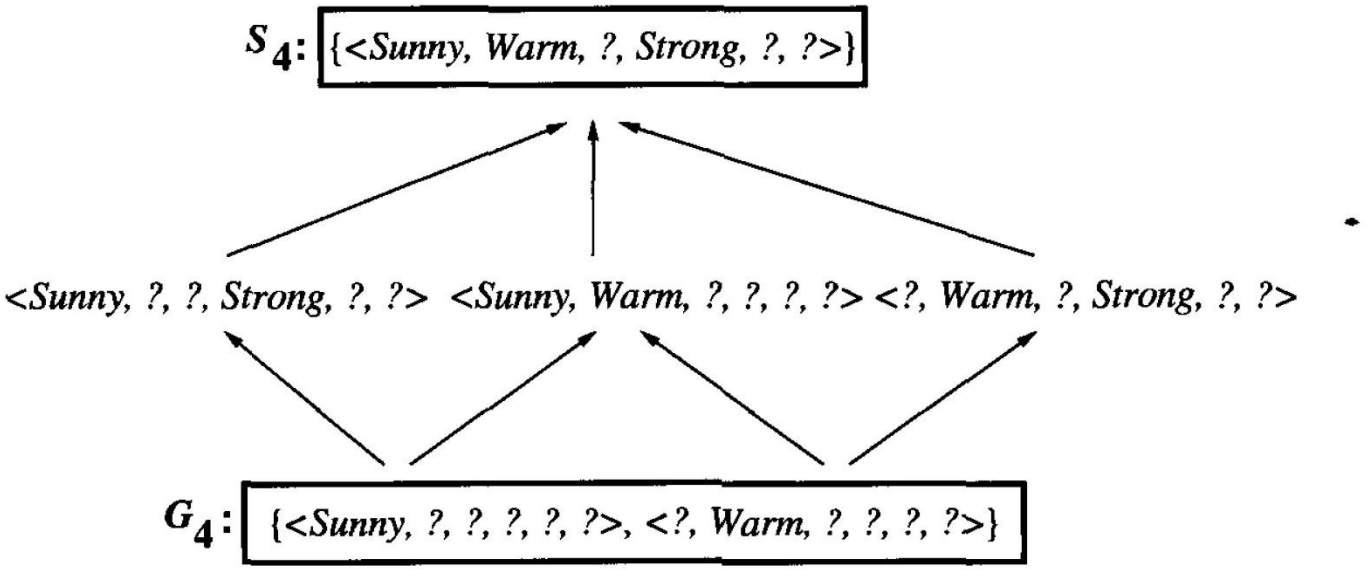


FIGURE 2.7
The final version space for the *EnjoySport* concept learning problem and training examples described earlier.

Remarks on Version Spaces And Candidate-Elimination

- 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 will happen if the training data contains errors?
- Suppose, for example, that the second training example above is incorrectly presented as a negative example instead of a positive example.

What if training data contains errors?

- Unfortunately, in this case the algorithm is certain to remove the correct target concept from the version space!

- Because it will remove every hypothesis that is inconsistent with each training example, it will eliminate the true target concept from the version space as soon as this false negative example is encountered.

- Of course, given sufficient additional training data the learner will eventually detect an inconsistency by noticing that the S and G boundary sets eventually converge to an empty version space.

What if the target concept cannot be described?

- A similar symptom, an empty version space, will appear when the training examples are correct, but the target concept cannot be described in the hypothesis representation (e.g., if the target concept is a disjunction of feature attributes and the hypothesis space supports only conjunctive descriptions).

Inductive Bias

Inductive Bias

- What if the target concept is not contained in the hypothesis space?
- Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis?
- How does the size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances?
- How does the size of the hypothesis space influence the number of training examples that must be observed?
- These are fundamental questions for inductive inference in general.
- Here we examine them in the context of the Candidate-Elimination algorithm.

A Biased Hypothesis Space

- Suppose we wish to assure that the hypothesis space contains the unknown target concept.
- The obvious solution is to enrich the hypothesis space to include every possible hypothesis.
- To illustrate, consider again the EnjoySport example in which we restricted the hypothesis space to include only conjunctions of attribute values.
- Because of this restriction, the hypothesis space is unable to represent even simple disjunctive target concepts such as "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

A Biased Hypothesis Space

- 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.

An Unbiased Learner

- The obvious solution to the problem of assuring that the target concept is in the hypothesis space H is to provide a hypothesis space capable of representing every teachable concept, that is, it is capable of representing every possible subset of the instances X.
- In general, the set of all subsets of a set X is called the power set of X.
- In the EnjoySport learning task, for example, the size of the instance space X of days described by the six available attributes is 96.
- How many possible concepts can be defined over this set of instances? In other words, how large is the power set of X?

An Unbiased Learner

- In general, the number of distinct subsets that can be defined over a set X containing |X| elements (i.e., the size of the power set of X) is $2^{|X|}$.

- Thus, there are 2^{96}, or approximately 10^{28} distinct target concepts that could be defined over this instance space and that our learner might be called upon to learn.

- Recall that our conjunctive hypothesis space is able to represent only 973 of these — a very biased hypothesis space indeed!

Unbiased hypothesis space

- Given an unbiased hypothesis space, we can safely use the Candidate-Elimination algorithm without worrying that the target concept might not be expressible.

- However, while this hypothesis space eliminates any problems of expressiveness, it unfortunately raises a new, equally difficult problem:
  - *our concept learning algorithm is now completely unable to generalize beyond the observed examples!*

Fundamental property of inductive inference

- *Property: A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.*
- In fact, the only reason that the Candidate-Elimination algorithm was able to generalize beyond the observed training examples in our original formulation of the EnjoySport task is that it was biased by the implicit assumption that the target concept could be represented by a conjunction of attribute values.
- In cases where this assumption is correct (and the training examples are error-free), its classification of new instances will also be correct.
- If this assumption is incorrect, however, it is certain that the Candidate-Elimination algorithm will misclassify at least some instances from X.

Inductive Bias

- Because inductive learning requires some form of prior assumptions, or inductive bias, we will find it useful to characterize different learning approaches by the inductive bias they employ.

- Let us define this notion of inductive bias more precisely.

- The key idea we wish to capture here is the policy by which the learner generalizes beyond the observed training data, to infer the classification of new instances.

Definition of inductive bias

- Consider a concept learning algorithm L for the set of instances X.
- Let c be an arbitrary concept defined over X, and 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;*

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)] \qquad (2.1)$$

- The notation y |- z indicates that z follows deductively from y (i.e., that z is provable from y).

Inductive bias of Candidate-Elimination algorithm

- *The target concept c is contained in the given hypothesis space H.*

Inductive Bias

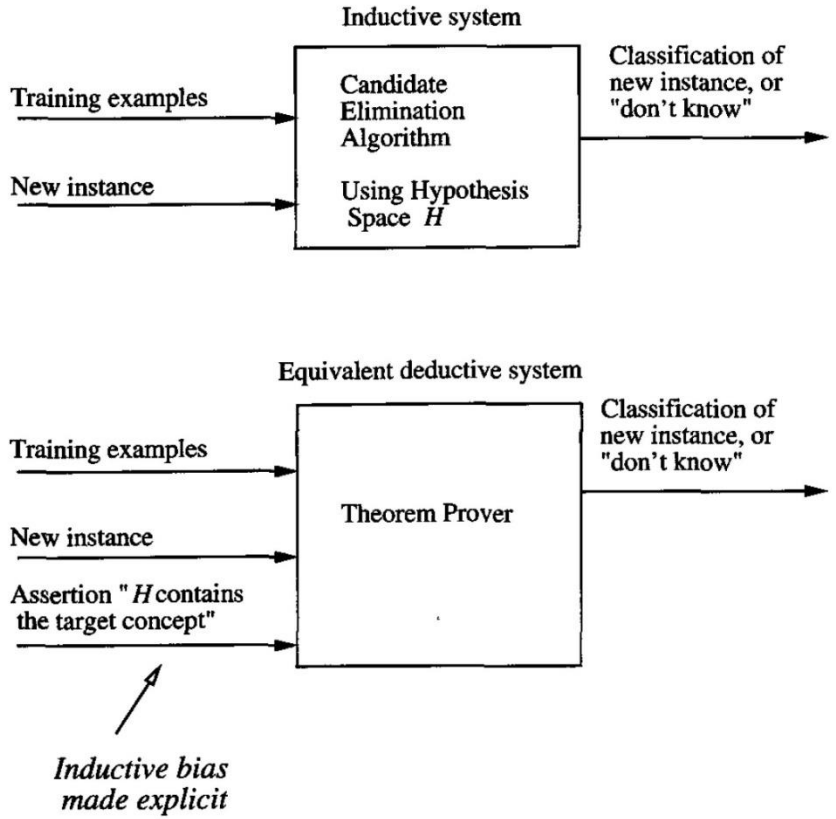


FIGURE 2.8
Modeling inductive systems by equivalent deductive systems. The input-output behavior of the CANDIDATE-ELIMINATION algorithm using a hypothesis space H is identical to that of a deductive theorem prover utilizing the assertion "H contains the target concept." This assertion is therefore called the *inductive bias* of the CANDIDATE-ELIMINATION algorithm. Characterizing inductive systems by their inductive bias allows modeling them by their equivalent deductive systems. This provides a way to compare inductive systems according to their policies for generalizing beyond the observed training data.

Learners

1. Rote-Learner: Learning corresponds simply to storing each observed training example in memory. Subsequent instances are classified by looking them up in memory. If the instance is found in memory, the stored classification is returned. Otherwise, the system refuses to classify the new instance.
2. Candidate-Elimination algorithm: New instances are classified only in the case where all members of the current version space agree on the classification. Otherwise, the system refuses to classify the new instance.
3. Find-S: This algorithm, described earlier, finds the most specific hypothesis consistent with the training examples. It then uses this hypothesis to classify all subsequent instances.

Learners and inductive bias

- The Rote-Learner has no inductive bias. The classifications it provides for new instances follow deductively from the observed training examples, with no additional assumptions required.
- The Candidate-Elimination algorithm has a stronger inductive bias: that the target concept can be represented in its hypothesis space. Because it has a stronger bias, it will classify some instances that the Rote-Learner will not. Of course the correctness of such classifications will depend completely on the correctness of this inductive bias.
- The Find-S algorithm has an even stronger inductive bias. In addition to the assumption that the target concept can be described in its hypothesis space, it has an additional inductive bias assumption: that all instances are negative instances unless the opposite is entailed by its other knowledge.

Key Issues in Machine Learning

- **What are good hypothesis spaces?**
  Which spaces have been useful in practical applications and why?
- **What algorithms can work with these spaces?**
  Are there general design principles for machine learning algorithms?
- **How can we optimize accuracy on future data points?**
  This is sometimes called the "problem of overfitting".
- **How can we have confidence in the results?**
  How much training data is required to find accurate hypotheses? (the *statistical question*)
- **Are some learning problems computationally intractable?**
  (the *computational question*)
- **How can we formulate application problems as machine learning problems?** (the *engineering question*)

Part II

Practical Data Mining:

Introduction to Concepts, Instances, and Attributes

Weka session

The ARFF format of WEKA

- ARFF: the attribute-relation file format

```
% ARFF file for the weather data with some numeric features
%
@relation weather

@attribute outlook { sunny, overcast, rainy }
@attribute temperature numeric
@attribute humidity numeric
@attribute windy { true, false }
@attribute play? { yes, no }

@data
%
% 14 instances
%
sunny, 85, 85, false, no
sunny, 80, 90, true, no
overcast, 83, 86, false, yes
rainy, 70, 96, false, yes
rainy, 68, 80, false, yes
rainy, 65, 70, true, no
overcast, 64, 65, true, yes
sunny, 72, 95, false, no
sunny, 69, 70, false, yes
rainy, 75, 80, false, yes
sunny, 75, 70, true, yes
overcast, 72, 90, true, yes
overcast, 81, 75, false, yes
rainy, 71, 91, true, no
```

Figure 2.2 ARFF file for the weather data.

Oracle session

- Oracle Data Miner Set Up
  - Tutorial

Readings for this part

- Machine Learning. T. Mitchell
  - Chapter 2
- Data Mining
  - Chapter 2.
- Weka
  - Manual
- Oracle
  - Tutorial