1. WELL-POSED LEARNING PROBLEMS

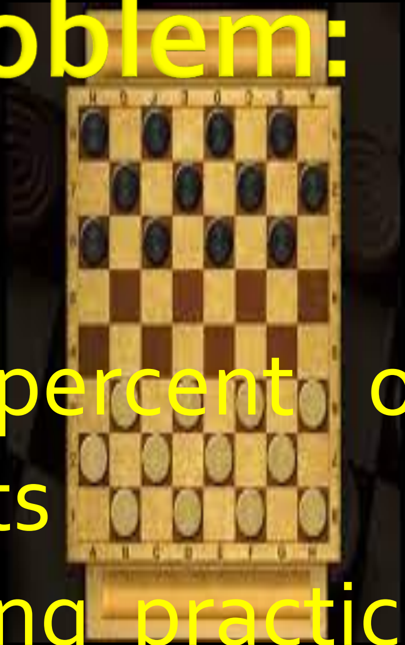
- Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P,

  if its performance at tasks in T, as measured by P, improves with experience E.

In general, to have a well-defined learning problem, we must identity these three features:

- the class of tasks,
- the measure of performance to be improved,
- and the source of 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:

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

To better filter emails as spam or not

- Task – Classifying emails as spam or not
- Performance Measure – The fraction of emails accurately classified as spam or not spam
- Experience – Observing you label emails as spam or not spam

Fruit Prediction Problem

- Task – forecasting different fruits for recognition
- Performance Measure – able to predict maximum variety of fruits
- Experience – training machine with the largest datasets of fruits images

2. DESIGNING A LEARNING SYSTEM

1) Choosing the Training Experience
2) Choosing the Target Function
3) Choosing a Representation for the Target Function
4) Choosing a Function Approximation Algorithm
    1) ESTIMATING TRAINING VALUES
    2) ADJUSTING THE WEIGHTS
5) The Final Design

2. DESIGNING A LEARNING SYSTEM

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

1. Choosing the Training Experience

- There are three attributes to choose the training experience:

1. Type of training experience
2. The degree to which the learner controls
3. Representation of the distribution examples

1. Choosing the Training Experience

- **3 Attributes:**

1. **The first design choice:** To choose the **type of training experience** from which our system will learn.
2. **A second important attribute: The degree to which the learner controls** the sequence of training examples.
3. **A third important attribute of the training experience: how well it represents the distribution** of examples over which the final system performance P must be measured.

1.a. Choose the Type of Training Experience:

- The type of training experience available can have a significant impact on success or failure of the learner.
- The key attribute is whether the training experience provides:
  - **Direct feedback** or
  - **Indirect feedback** regarding the choices made by the performance system.

Example:

- **Direct training examples:** consisting of individual checkers board states and the correct move for each.
- **Indirect** information consisting of the move sequences and final outcomes of various games played.
- **Credit Assignment:** It can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves.
- **Hence, learning from direct training feedback is typically easier than learning from indirect feedback.**

1.b. 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.
- The learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.
- 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.

1.c. 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.
- Mastery of one distribution of examples will not necessary lead to strong performance over some other distribution.

2. Choosing the Target Function

- **What type of knowledge will be learned and how this will be used by the performance program?**
- checkers-playing program :
  - It 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.
- To choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.
- **Target function:** ***ChooseMove***
- Use the notation **ChooseMove : B -> 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 target function such as Choose Move turns out to be very difficult to learn given the kind of indirect training experience available to the system, an alternative target function is then an evaluation function that assigns a numerical score to any given board state, **$V: B \rightarrow R$**
- **What is the value of the target function V for any given board state?**
- 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**

3. Choosing a Representation for the Target Function

- The ideal target function : V
- Representation for the target function: $\hat{V}$
- Allow the program to represent $\hat{V}$ using a large table with a distinct entry specifying the
  - value for each distinct board state.
  - using a collection of rules that match against features of the board state, or
  - a quadratic polynomial function of predefined board features, or
  - an artificial neural network.

- The function 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 (i.e., which can be captured on red's next turn)
- 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

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

- where w0 through W6 are numerical coefficients, or weights

4.Choosing a Function Approximation Algorithm

- To learn the target function V' need set of training examples, each describes specific board state b and the training value $V_{train}(b)$
- Each training example is the ordered pair:

  $(b, V_{train}(b))$
- Example:
  - board state b in which black has won the game(x2=0, the number of red pieces)
  - $V_{train}(b)$ is +100
  - ((x1=3,x2=0,x3=1,x4=0,x5=0,x6=0),+100)

- Estimating Training Values: Vtrain (b) ← V' (Successor (b)).
- Adjusting the weights: To specify the learning algorithm for choosing the weights wi to best fit the set of training examples {}

The Final Design

- The final design of our checkers learning systems can be naturally described by four distinct program modules that represent the central components in many learning

3. PERSPECTIVES AND ISSUES IN MACHINE LEARNING

- One useful perspective on machine learning is that it involves
  - searching a very large space of possible **hypotheses** (assumption/imagination/explanation for anything) to determine one that best fits the observed data and any prior knowledge held by the learner.

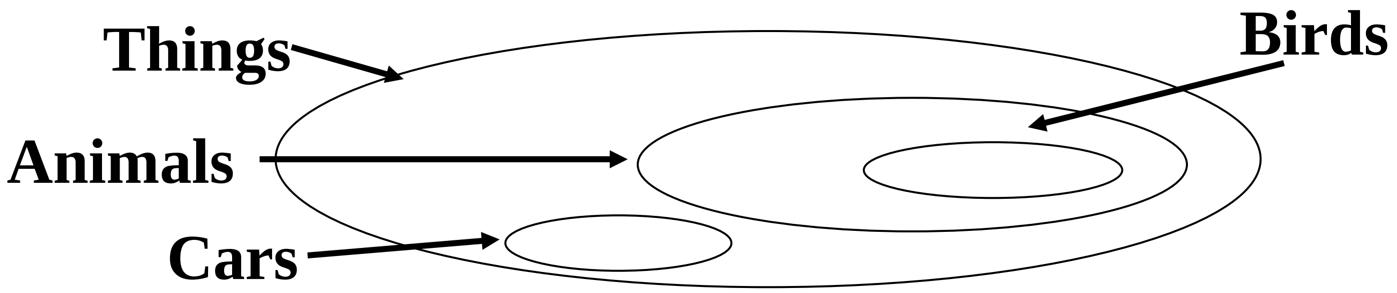
Issues in Machine Learning

Our checkers example raises a number of generic questions about machine learning.

- 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?

What is a Concept?

- A Concept is a subset of objects or events defined over a larger set [Example: The concept of a *bird* is the subset of all objects (i.e., the set of all *things* or all *animals*) that belong to the category of bird.]



- Alternatively, a concept is a **boolean valued function** defined over this larger set [Example: a function defined over all *animals* whose value is *true* for birds and *false* for every other animal].

What is Concept-Learning?

Given a set of examples labeled as members or non-members of a concept, *concept-learning* consists of automatically inferring the general definition of this concept.

In other words, *concept-learning* consists of approximating a boolean valued function from training examples of its input and output.

Concept Learning task

- **Concept:** Good Days for Water Sports (values: Yes, No)
- **Attributes/Features:**
  - Sky (values: Sunny, Cloudy, Rainy)
  - AirTemp (values: Warm, Cold)
  - Humidity (values: Normal, High)
  - Wind (values: Strong, Weak)
  - Water (Warm, Cool)
  - Forecast (values: Same, Change)

class

- **Example of a Training Point:**

<Sunny, Warm, High, Strong, Warm, Same, Yes>

Example of a Concept Learning task

Database:

Day	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*WaterSport*
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
							class

Chosen Hypothesis Representation:

Conjunction of constraints on each attribute where:

- "?" means "any value is acceptable"
- "0" means "no value is acceptable"

Example of a hypothesis: <?,Cold,High,?,?,?>

(If the air temperature is cold and the humidity high then it is a good day for water sports)

Example of a Concept Learning task

- **Goal:** To infer the "best" concept-description from the set of all possible hypotheses ("best" means "which best generalizes to all (known or unknown) elements of the instance space".

  ➡ . concept-learning is an ill-defined task)
- **Most General Hypothesis:** Everyday is a good day for water sports <?,?,?,?,?,?>
- **Most Specific Hypothesis:** No day is a good day for water sports <0,0,0,0,0,0>

Terminology and Notation

- The set of items over which the concept is defined is called the set of ***instances*** (denoted by X)
- The concept to be learned is called the *Target Concept* (denoted by c: X--> {0,1})
- The set of ***Training Examples*** is a set of instances, x, along with their target concept value c(x).
- Members of the concept (instances for which c(x)=1) are called ***positive examples***.
- Nonmembers of the concept (instances for which c(x)=0) are called ***negative examples***.
- H represents the set of ***all possible hypotheses***. H is determined by the human designer's choice of a hypothesis representation.
- **The goal of concept-learning is to find a hypothesis h:X --> {0,1} such that h(x)=c(x) for all x in X.**

The Inductive Learning Hypothesis

- 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.
- Inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data.

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.
- Selecting a Hypothesis Representation is an important step since it restricts (or *biases*) the space that can be searched.
- For example, the hypothesis "If the air temperature is cold **or** the humidity high then it is a good day for water sports" cannot be expressed in our chosen representation.

Example: Concept Learning as Search

- The instances X and hypotheses H in the EnjoySport learning task.
- Given that the attribute
  - Sky has three possible values
  - 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.**
- **syntactically distinct hypotheses within H:**
  - A similar calculation shows that there are 5.4.4.4.4.4 = 5120.
- Notice, however, that 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.
- 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. If we view learning as a se

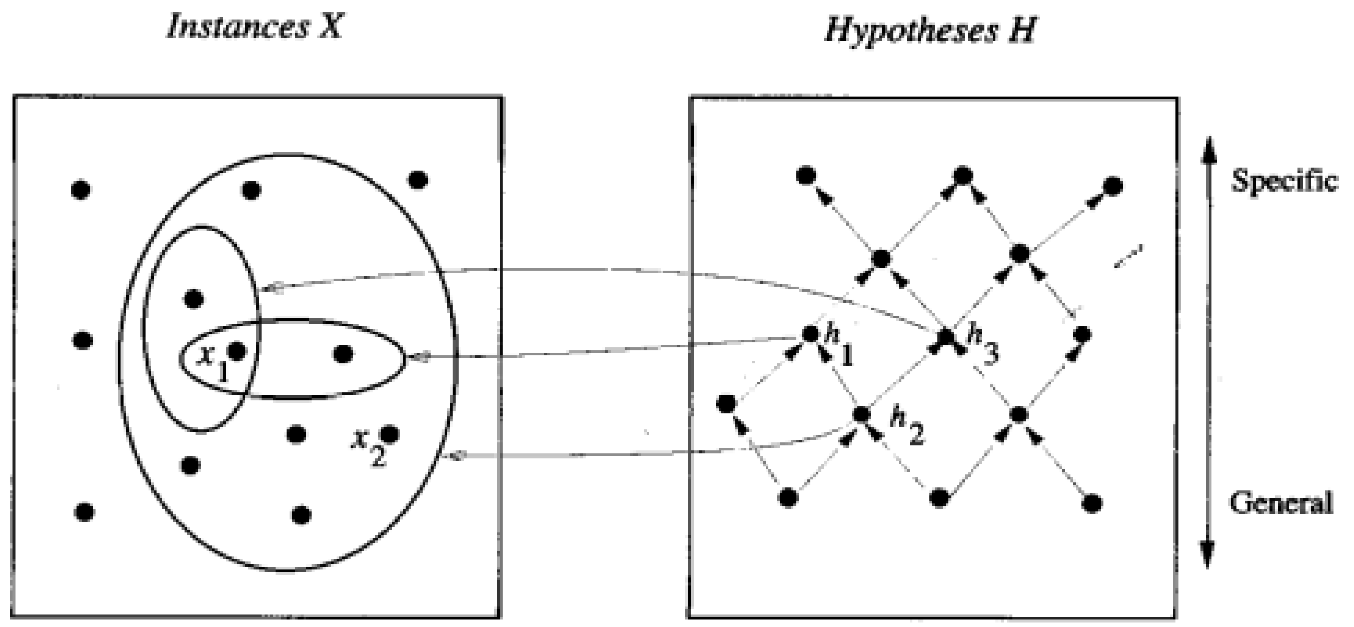
General to Specific Ordering of Hypotheses

- **Definition:** Let *hj* and *hk* be boolean-valued functions defined over X. Then *hj* is **more-general-than-or-equal-to** *hk* <u>iff</u> For all x in X, [(*hk*(x) = 1) --> (*hj*(x)=1)]

Example:

- ***h1* = <Sunny,?,?,Strong,?,?>**
- ***h2* = <Sunny,?,?,?,?,?>**

- Every instance that are classified as positive by *h1* will also be classified as positive by *h2* in our example data set. Therefore *h2* is more general than *h1*.
- We also use the ideas of **“strictly”-more-general-than,** and **more-specific-than** (illustration [Mitchell, p. 25])



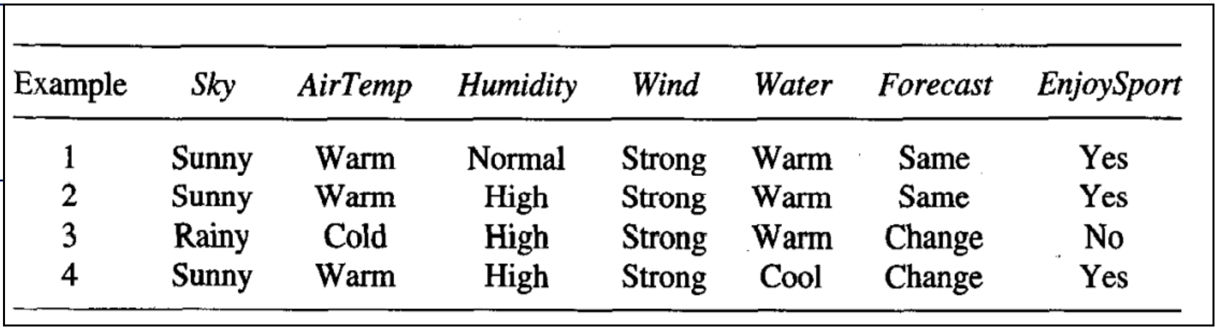
x_1= <Sunny, Warm, High, Strong, Cool, Same>
x_2= <Sunny, Warm, High, Light, Warm, Same>

h_1= <Sunny, ?, ?, Strong, ?, ?>
h_2= <Sunny, ?, ?, ?, ?, ?>
h_3= <Sunny, ?, ?, ?, Cool, ?>

Find-S, a Maximally Specific Hypothesis Learning Algorithm

```
Initialize $h$ to the most specific hypothesis in $H$
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 $x$
Output hypothesis $h$
```

Example:

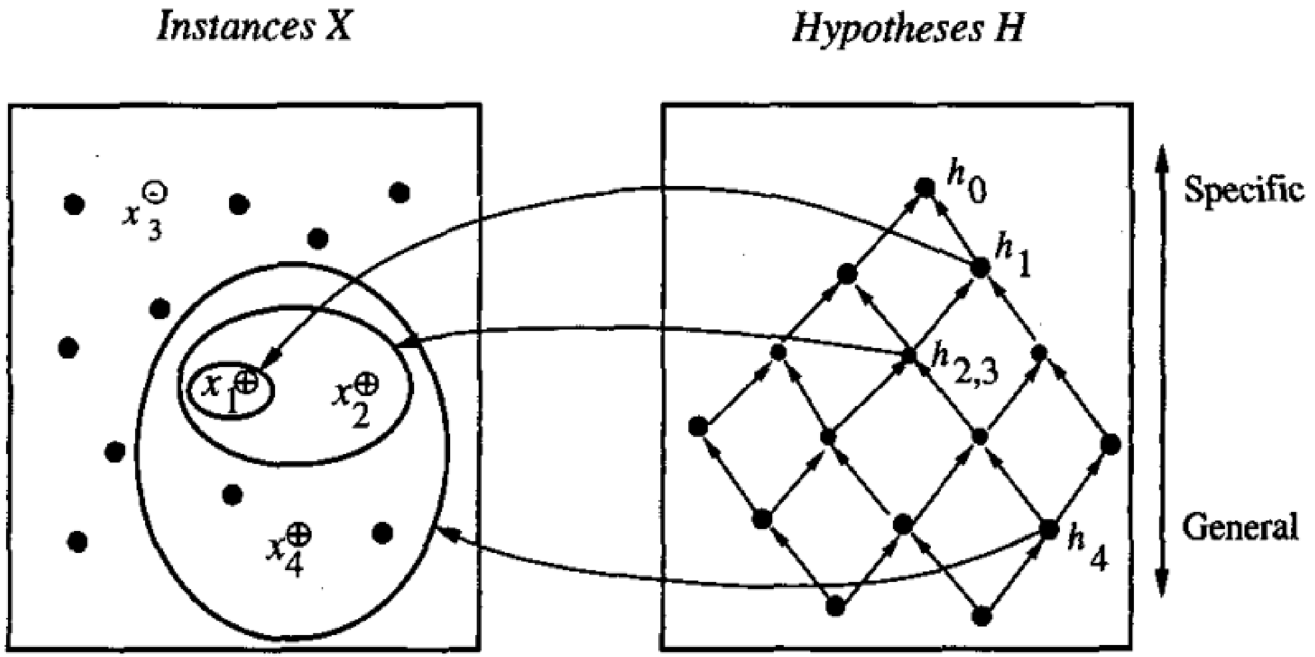


Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- The first step of FIND-S is to initialize h to the most specific hypothesis in H

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

- It becomes clear that our hypothesis is too specific. In particular, none of the " Φ " constraints in h are satisfied by this example,
- so each is replaced by the next more general constraint:



h_0 = <∅, ∅, ∅, ∅, ∅, ∅>

x_1 = <*Sunny Warm Normal Strong Warm Same*>, +	h_1 = <*Sunny Warm Normal Strong Warm Same*>
x_2 = <*Sunny Warm High Strong Warm Same*>, +	h_2 = <*Sunny Warm ? Strong Warm Same*>
x_3 = <*Rainy Cold High Strong Warm Change*>, -	h_3 = <*Sunny Warm ? Strong Warm Same*>
x_4 = <*Sunny Warm High Strong Cool Change*>, +	h_4 = <*Sunny Warm ? Strong ? ?* >

Shortcomings of Find-S

- Although Find-S finds a hypothesis consistent with the training data, it does not indicate whether that is the only one available
- Is it a good strategy to prefer the most specific hypothesis?
- What if the training set is inconsistent (*noisy*)?
- What if there are several maximally specific consistent hypotheses? Find-S cannot backtrack!

Version Spaces and the Candidate-Elimination Algorithm

The CANDIDATE-ELIMINATION algorithm provides a useful conceptual framework for introducing several fundamental issues in machine learning.

1. Representation
2. The LIST-THEN-ELIMINATE Algorithm
3. A More Compact Representation for Version Spaces
4. CANDIDATE-ELIMINATION Learning Algorithm
5. An Illustrative Example

Version Spaces and the Candidate-Elimination Algorithm

Representation:

- **Definition:** A hypothesis *h* is **consistent** with a set of training examples *D* iff *h(x)* = *c(x)* for each example *<x,c(x)>* in *D*.
  - Consistent(h,D)=(for all (x,c(x)) belongs to D) h(x)=c(x)
- **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*.
  - $VS_{H,D}$={h belongs to H| Consistent(h,D)}
- **NB:** While a Version Space can be exhaustively enumerated, a more compact representation is preferred.

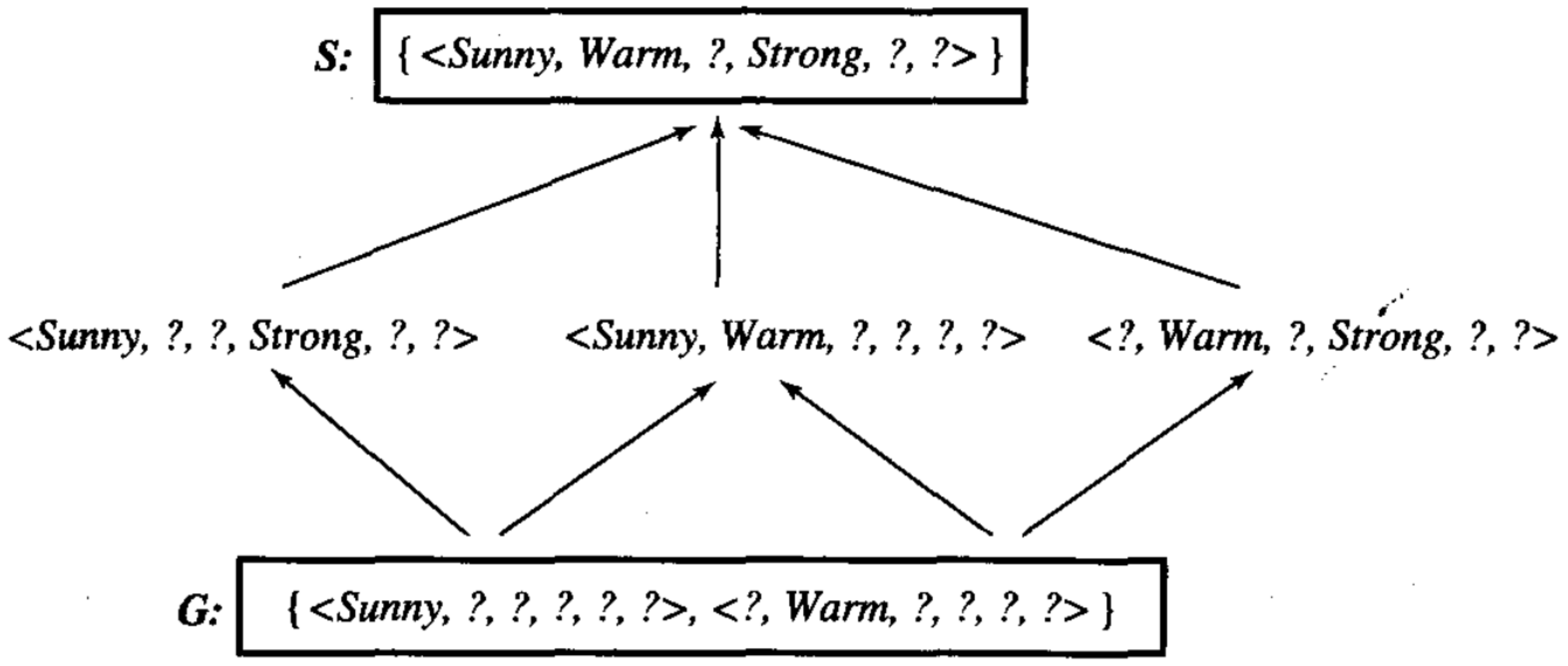
The LIST-THEN-ELIMINATE Algorithm

1. VersionSpace ← a list containing every hypothesis in H
2. For each training example, (x, c(x)) remove from VersionSpace any hypothesis h for which h(x) != c(x)
3. Output the list of hypotheses in VersionSpace

A Compact Representation for Version Spaces

- Instead of enumerating all the hypotheses consistent with a training set, we can represent its **most specific** and **most general** boundaries. The hypotheses included in-between these two boundaries can be generated as needed.
- **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*.

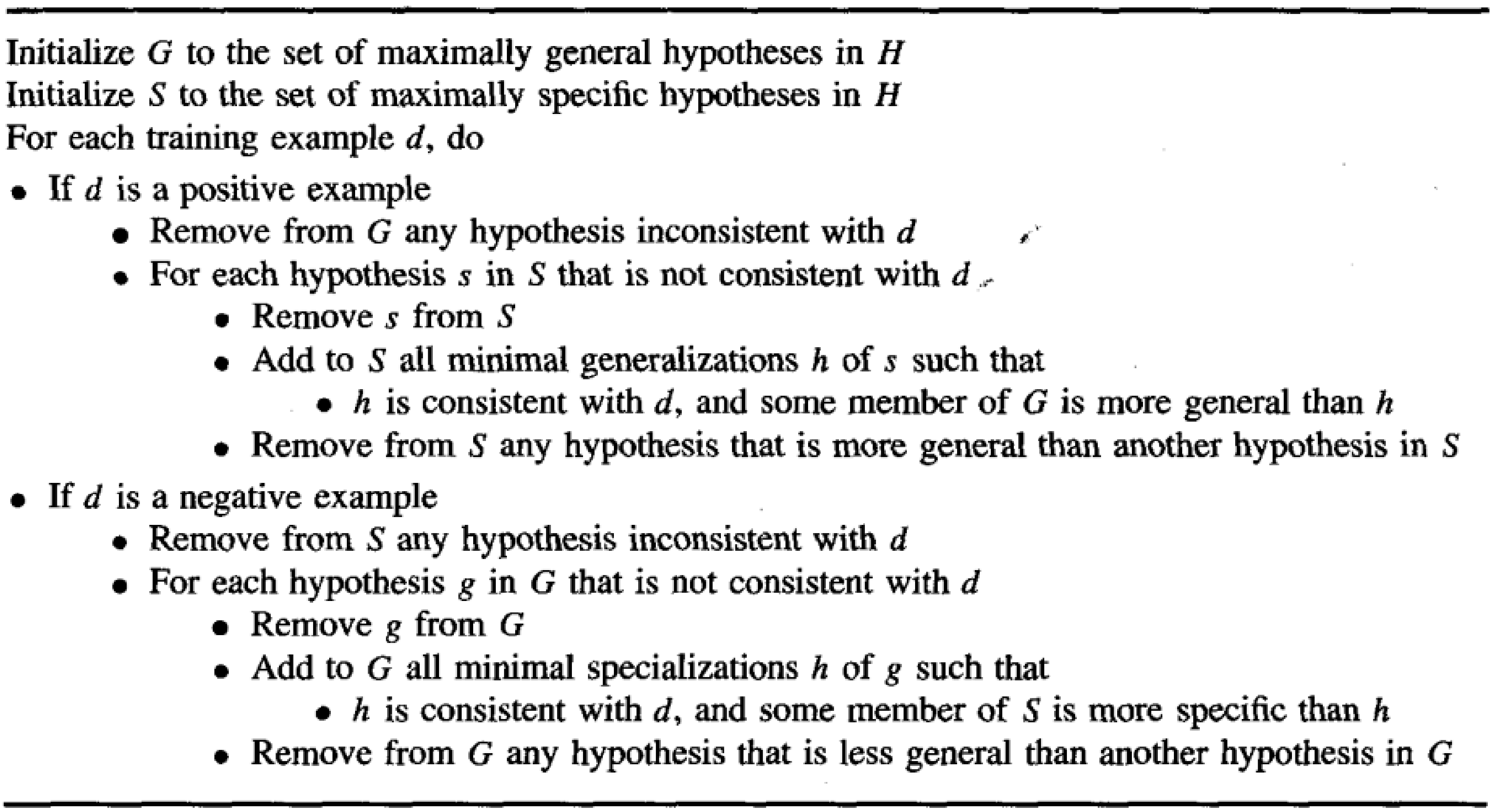
A More Compact Representation for Version Spaces



4. Candidate-Elimination Learning Algorithm

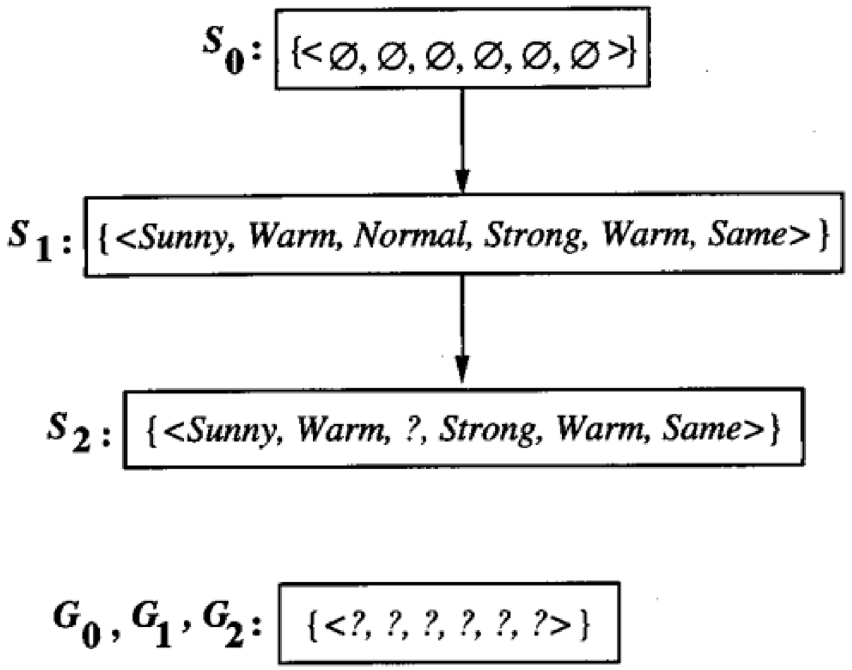
- The candidate-Elimination algorithm computes the version space containing all (and only those) hypotheses from *H* that are consistent with an observed sequence of training examples.

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 $G$
```

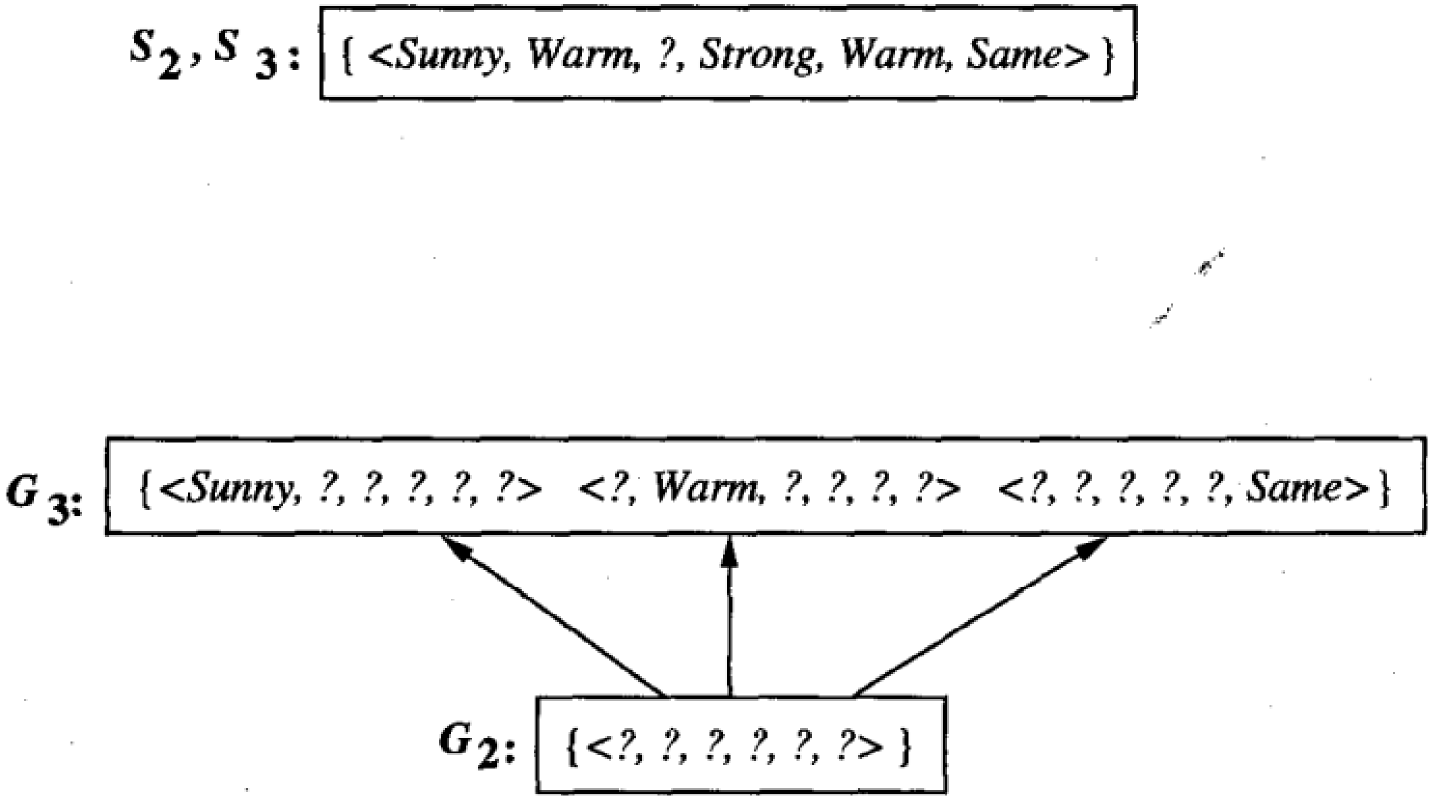
5. An Illustrative Example


S_0: {<∅, ∅, ∅, ∅, ∅, ∅>}
S_1: {<Sunny, Warm, Normal, Strong, Warm, Same>}
S_2: {<Sunny, Warm, ?, Strong, Warm, Same>}
G_0, G_1, G_2: {<?, ?, ?, ?, ?, ?>}

Training examples:

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

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



Training Example:

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

$$\begin{tikzcd}
S_3: \boxed{\{\langle \textit{Sunny, Warm, ?, Strong, Warm, Same}\rangle\}} \arrow[d] \\
S_4: \boxed{\{\langle \textit{Sunny, Warm, ?, Strong, ?, ?}\rangle\}}
\end{tikzcd}$$

$$\begin{tikzcd}
G_4: \boxed{\{\langle \textit{Sunny, ?, ?, ?, ?, ?}\rangle \quad \langle \textit{?, Warm, ?, ?, ?, ?}\rangle\}} \\
G_3: \boxed{\{\langle \textit{Sunny, ?, ?, ?, ?, ?}\rangle \quad \langle \textit{?, Warm, ?, ?, ?, ?}\rangle \quad \langle \textit{?, ?, ?, ?, ?, Same}\rangle\}} \arrow[u]
\end{tikzcd}$$

Training Example:

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

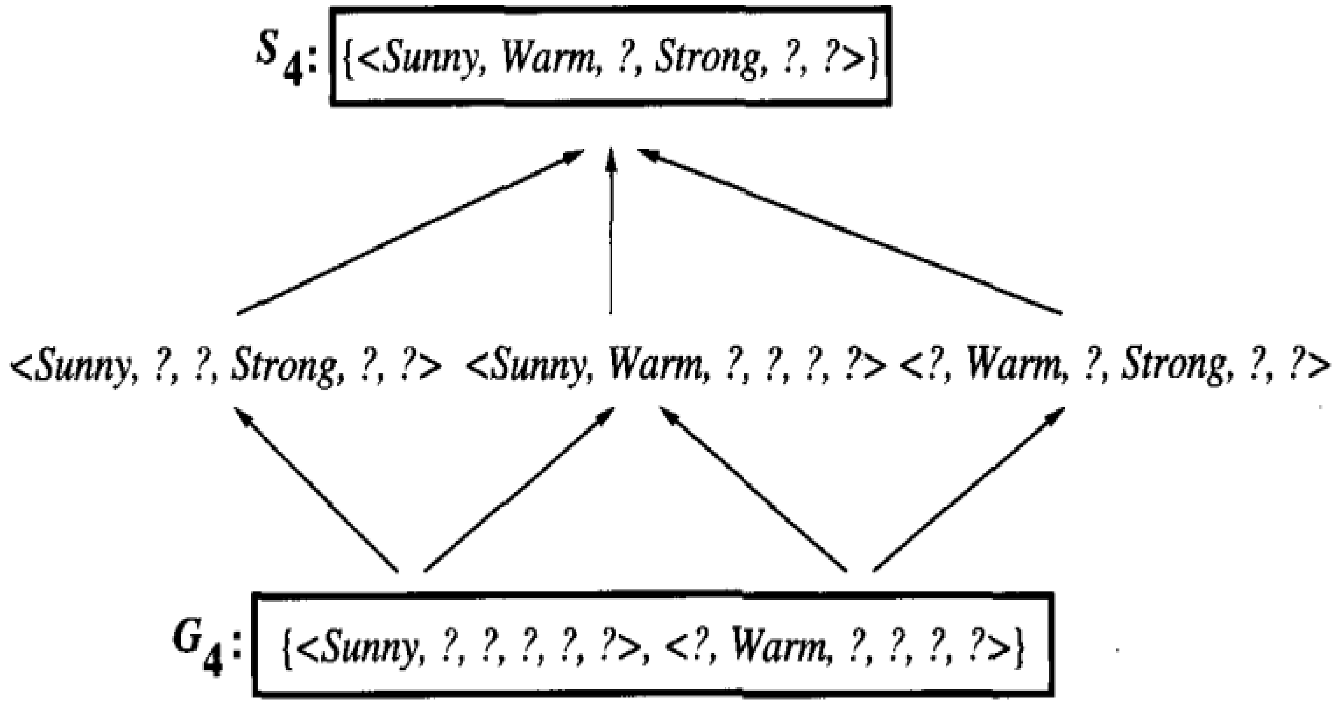


FIGURE 2.7
The final version space for the *EnjoySport* concept learning problem and training examples described earlier.

Remarks on Version Spaces and Candidate-Elimination

1. 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;
- (2) There is some hypothesis in *H* that correctly describes the target concept.
- Convergence can be speeded up by presenting the data in a strategic order. The best examples are those that satisfy exactly half of the hypotheses in the current version space.
- Version-Spaces can be used to assign certainty scores to the classification of new examples
- 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.

Remarks on Version Spaces and Candidate-Elimination

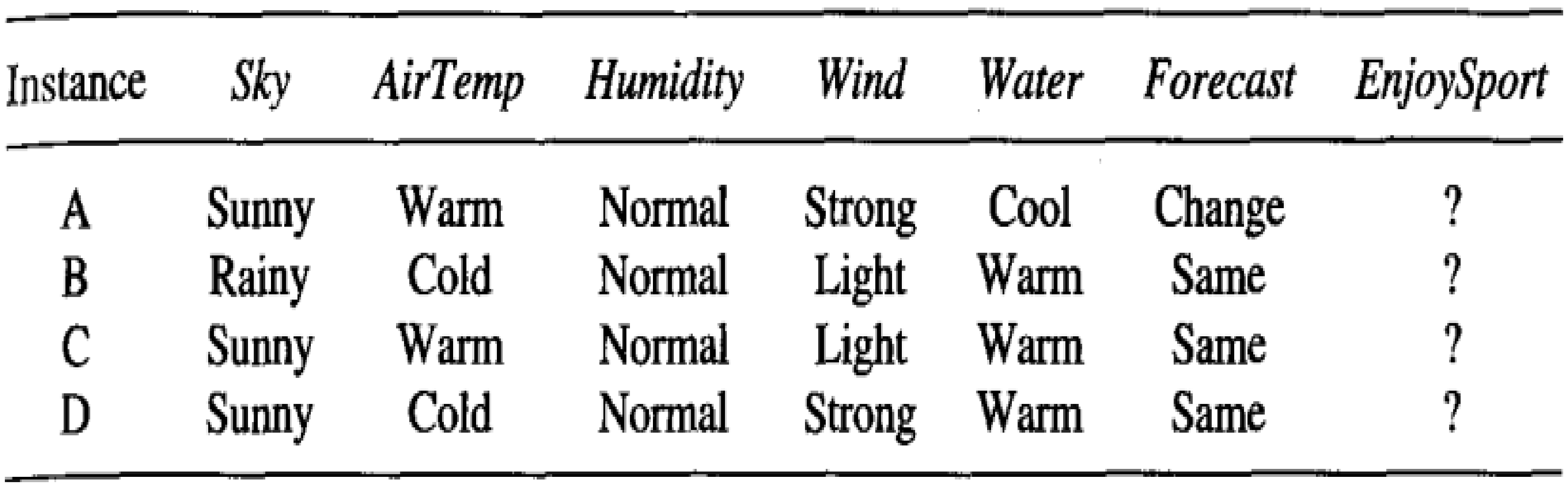
2. What Training Example Should the Learner Request Next?

<Sunny, Warm, Normal, Light, Warm, Same>

- -this instance satisfies three of the six hypotheses in the current version space
- If the trainer classifies this instance as a positive example, the S boundary of the version space can then be generalized. Alternatively, if the trainer indicates that this is a negative example, the G boundary can then be specialized. Either way, the learner will succeed in learning more about the true identity of the target concept, shrinking the version space from six hypotheses to half this number.

Remarks on Version Spaces and Candidate-Elimination

3. How Can Partially Learned Concepts Be Used?



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.