Concept Learning

- Inducing general functions from specific training examples is a main issue of machine learning.
- **Concept Learning:** Acquiring the definition of a general category from given sample positive and negative training examples of the category.
- *Concept Learning* can seen as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples.
- The hypothesis space has a *general-to-specific ordering* of hypotheses, and the search can be efficiently organized by taking advantage of a naturally occurring structure over the hypothesis space.

Concept Learning

- A Formal Definition for Concept Learning:

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

- An example for concept-learning is the learning of bird-concept from the given examples of birds (positive examples) and non-birds (negative examples).
- We are trying to learn the definition of a concept from given examples.

A Concept Learning Task – Enjoy Sport
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	Warm	Change	**YES**

ATTRIBUTES **CONCEPT**

- A set of example days, and each is described by six attributes.
- The task is to learn to predict the value of EnjoySport for arbitrary day, based on the values of its attribute values.

EnjoySport – Hypothesis Representation

- **Each hypothesis consists of a conjuction of constraints on the instance attributes**.
- Each hypothesis will be a vector of six constraints, specifying the values of the six attributes
  - (Sky, AirTemp, Humidity, Wind, Water, and Forecast).
- Each attribute will be:

  ? - indicating any value is acceptable for the attribute (**don't care**)

  single value – specifying a single required value (ex. Warm) (**specific**)

  0 - indicating no value is acceptable for the attribute (**no value**)

Hypothesis Representation

- A hypothesis:

  Sky *AirTemp* *Humidity* *Wind* *Water* *Forecast*

  < Sunny, ? , ? , Strong , ? , Same >

- *The most general hypothesis* – that every day is a positive example

  <?, ?, ?, ?, ?, ?>

- *The most specific hypothesis* – that no day is a positive example

  <0, 0, 0, 0, 0, 0>

- *EnjoySport concept learning task* requires learning the sets of days for which EnjoySport=yes, describing this set by a conjunction of constraints over the instance attributes.

EnjoySport Concept Learning Task

Given

- ***Instances X*** : set of all possible days, each described by the attributes
  - Sky – (values: Sunny, Cloudy, Rainy)
  - AirTemp – (values: Warm, Cold)
  - Humidity – (values: Normal, High)
  - Wind – (values: Strong, Weak)
  - Water – (values: Warm, Cold)
  - Forecast – (values: Same, Change)
- ***Target Concept (Function) c*** : EnjoySport : X → {0,1}
- ***Hypotheses H*** : Each hypothesis is described by a conjunction of constraints on the attributes.
- ***Training Examples D*** : positive and negative examples of the target function

Determine

- A hypothesis h in H such that h(x) = c(x) for all x in D.

The Inductive Learning Hypothesis

- Although the learning task is to determine a hypothesis h identical to the target concept cover 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.
  - 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.

- **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.
- 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.

Enjoy Sport - Hypothesis Space

- Sky has 3 possible values, and other 5 attributes have 2 possible values.
- There are 96 (= 3.2.2.2.2.2) distinct instances in X.
- There are 5120 (=5.4.4.4.4.4) *syntactically distinct hypotheses* in H.
  - Two more values for attributes: ? and 0
- Every hypothesis containing one or more 0 symbols represents the empty set of instances; that is, it classifies every instance as negative.
- There are 973 (= 1 + 4.3.3.3.3.3) *semantically distinct hypotheses* in H.
  - Only one more value for attributes: ?, and one hypothesis representing empty set of instances.
- Although EnjoySport has small, finite hypothesis space, most learning tasks have much larger (even infinite) hypothesis spaces.
  - We need efficient search algorithms on the hypothesis spaces.

General-to-Specific Ordering of Hypotheses

- Many algorithms for concept learning organize the search through the hypothesis space by relying on 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.

- Consider 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 instancesas 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* hl.

More-General-Than Relation

- For any instance x in X and hypothesis h in H, we say that x satisfies h if and only if h(x) = 1.

- **More-General-Than-Or-Equal Relation:**

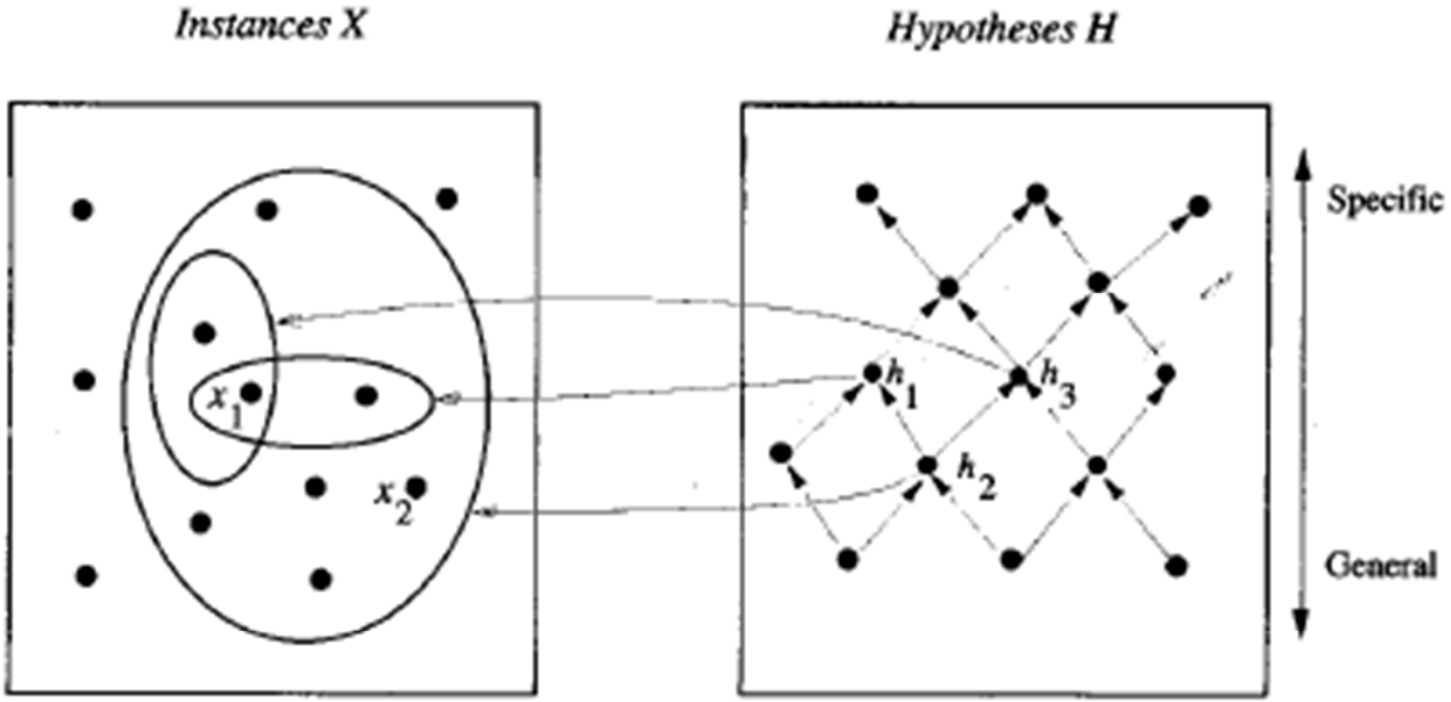
  Let h1 and h2 be two boolean-valued functions defined over X.

  Then h1 is ***more-general-than-or-equal-to*** h2 (written h1 $\geq$ h2)

  if and only if any instance that satisfies h2 also satisfies h1.

- h1 is ***more-general-than*** h2 (h1 > h2) if and only if h1$\geq$h2 is true and h2$\geq$h1 is false. We also say h2 is ***more-specific-than*** h1.

More-General-Relation



- h2 > h1 and h2 > h3
- But there is no more-general relation between h1 and h3

FIND-S Algorithm

- FIND-S Algorithm starts from the most specific hypothesis and generalize it by considering only positive examples.
- FIND-S algorithm ignores negative examples.
  - As long as the hypothesis space contains a hypothesis that describes the true target concept, and the training data contains no errors, ignoring negative examples does not cause to any problem.
- FIND-S algorithm finds the most specific hypothesis within H that is consistent with the positive training examples.
  - The final hypothesis will also be consistent with negative examples if the correct target concept is in H, and the training examples are correct.

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, in h

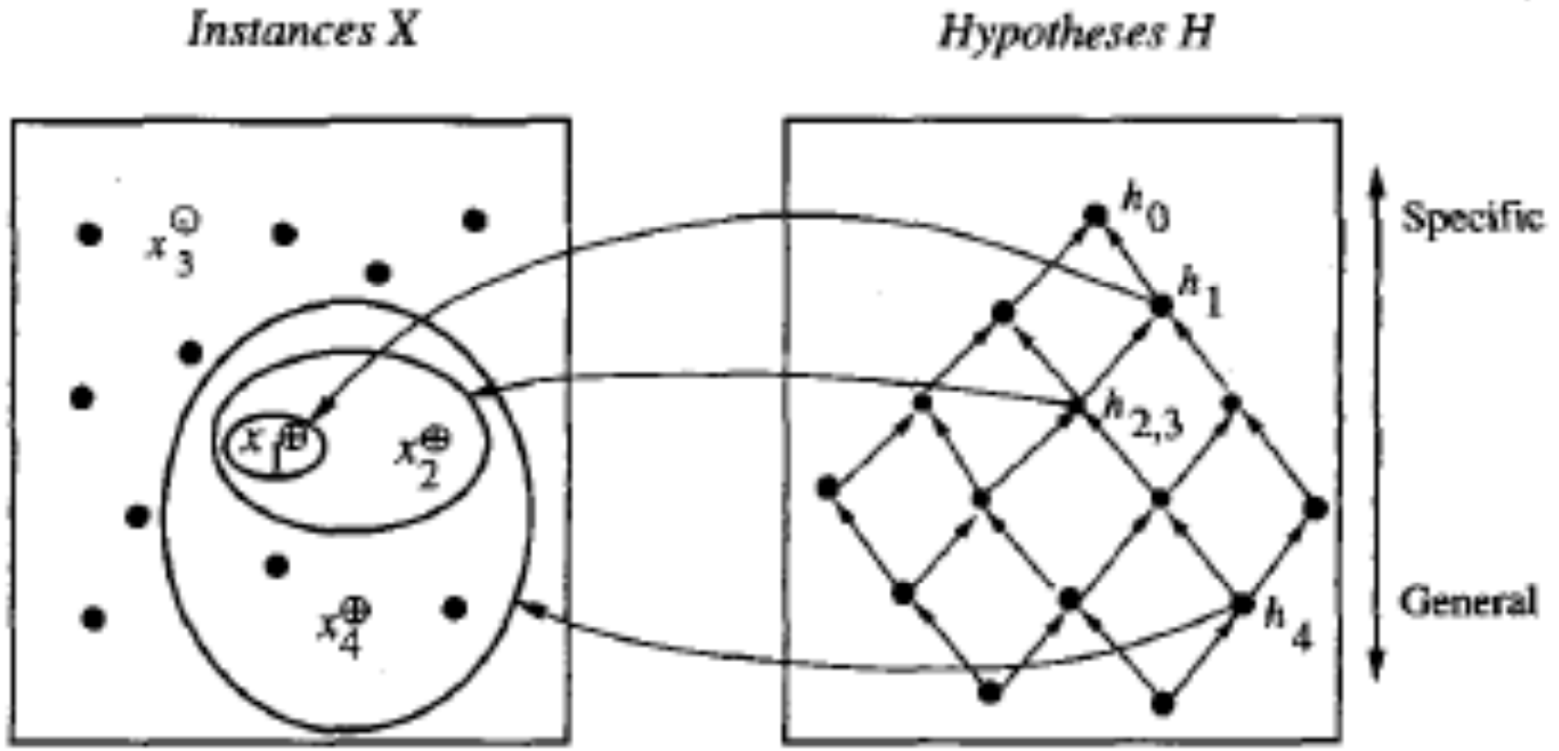
      If the constraint a, is satisfied by x

      Then do nothing

      Else replace a, in h by the next more general constraint that is satisfied by x

3. Output hypothesis h

FIND-S Algorithm - 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 ? ?*>

Unanswered Questions by FIND-S Algorithm

- Has FIND-S 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.

- 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.

Unanswered Questions by FIND-S Algorithm

- 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.
- 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.

Candidate-Elimination Algorithm

- 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.
  - Candidate-Elimination algorithm computes the description of this set without explicitly enumerating all of its members.
  - This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses.

Consistent Hypothesis

A hypothesis h is **consistent** with a set of training examples D of target concept c if and only if $h(x) = c(x)$ for each training example $\langle x, c(x) \rangle$ in D.

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

- The key difference between this definition of ***consistent*** and ***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).***

Version Spaces

- 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.

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

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

List-Then-Eliminate Algorithm

- List-Then-Eliminate algorithm 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.
  - Presumably, this 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.
- 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.

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 $VersionSpace$

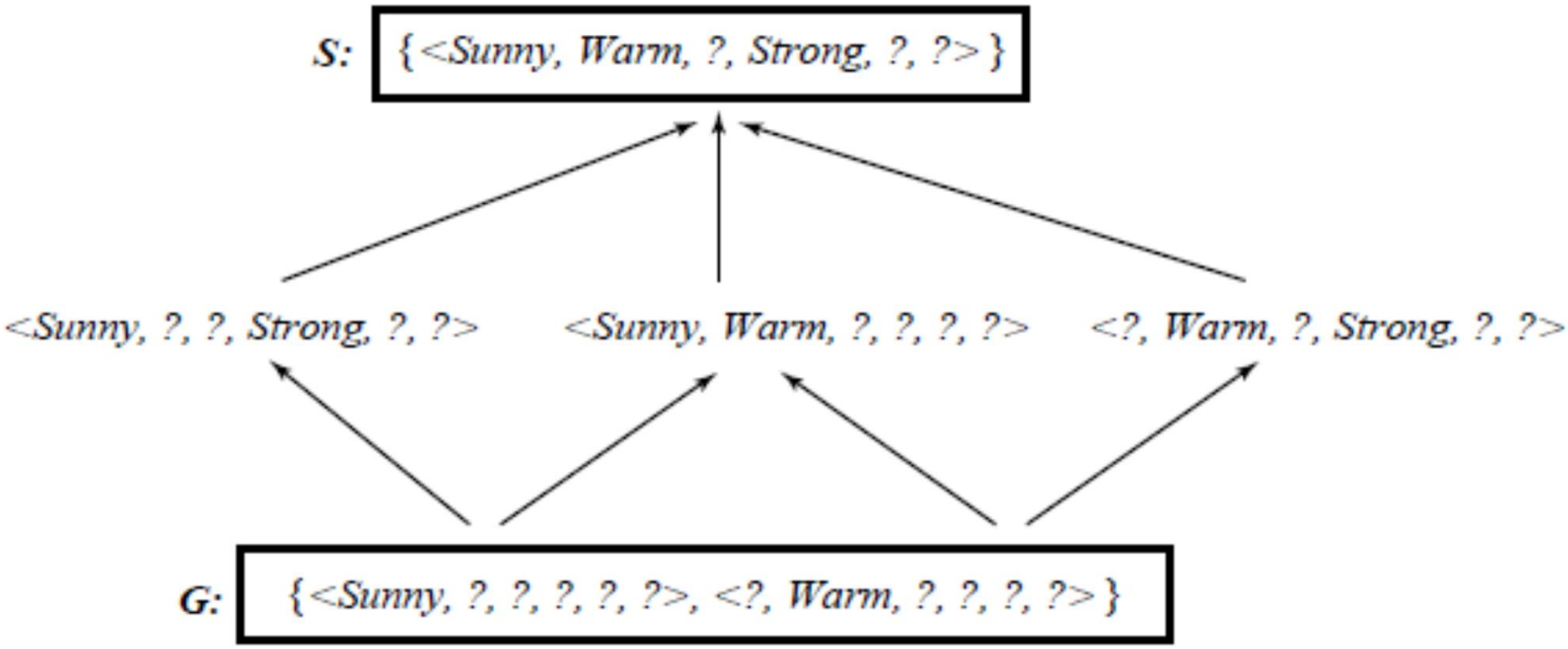
Compact Representation of Version Spaces

- A version space can be represented with its *general* and *specific boundary sets*.
- The Candidate-Elimination algorithm represents the version space by storing only its most general members G and its most specific members S.
- Given only these two sets S and G, it is possible to enumerate all members of a version space by generating hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses.
- Every member of the version space lies between these boundaries

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

where x ≥y means x is more general or equal to y.

Example Version Space



- 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.***

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

  G0 ← { <?, ?, ?, ?, ?, ?> }

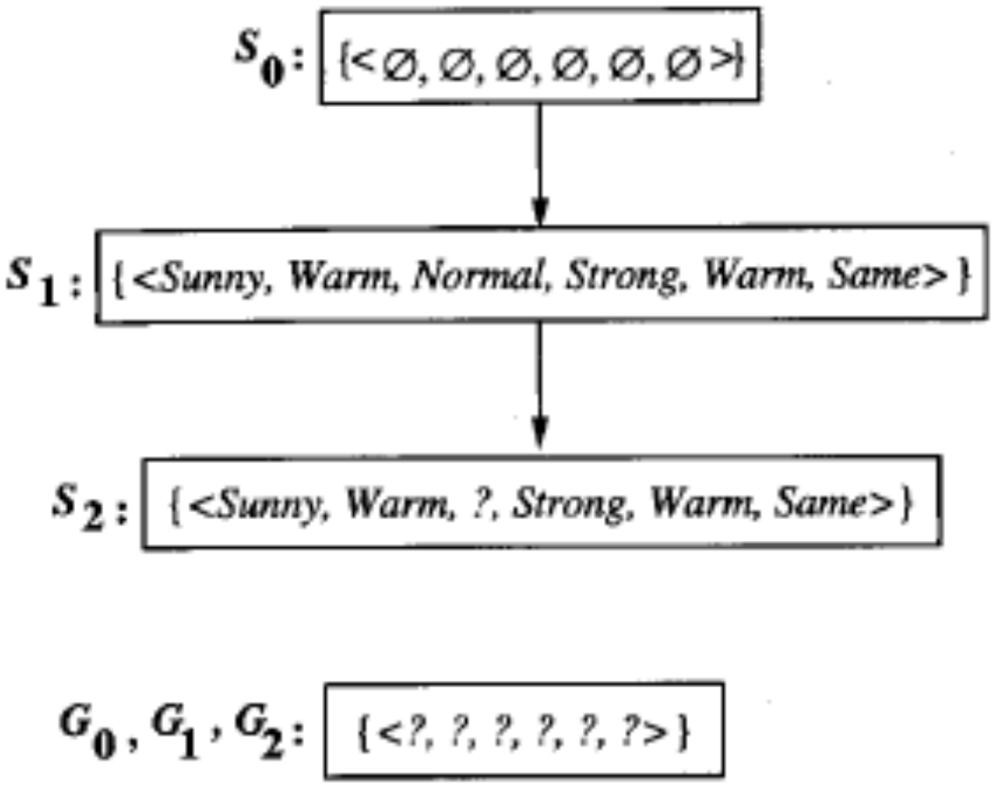
  and initializing the S boundary set to contain the most specific hypothesis

  S0 ← { <0, 0, 0, 0, 0, 0> }
- These two boundary sets delimit the entire hypothesis space, because every other hypothesis in H is both more general than S0 and more specific than G0.
- 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 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

Candidate-Elimination Algorithm - Example



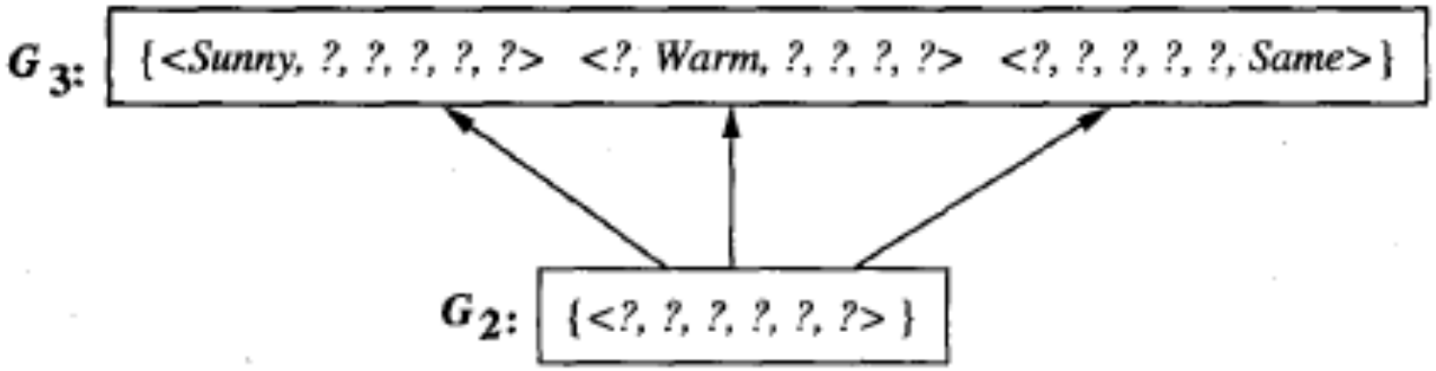
- *S0* and *G0* are the initial boundary sets corresponding to the most specific and most general hypotheses.
- Training examples 1 and 2 force the *S* boundary to become more general.
- They have no effect on the *G* boundary

Training examples:

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

Candidate-Elimination Algorithm - Example

S_2, S_3: { <Sunny, Warm, ?, Strong, Warm, Same> }



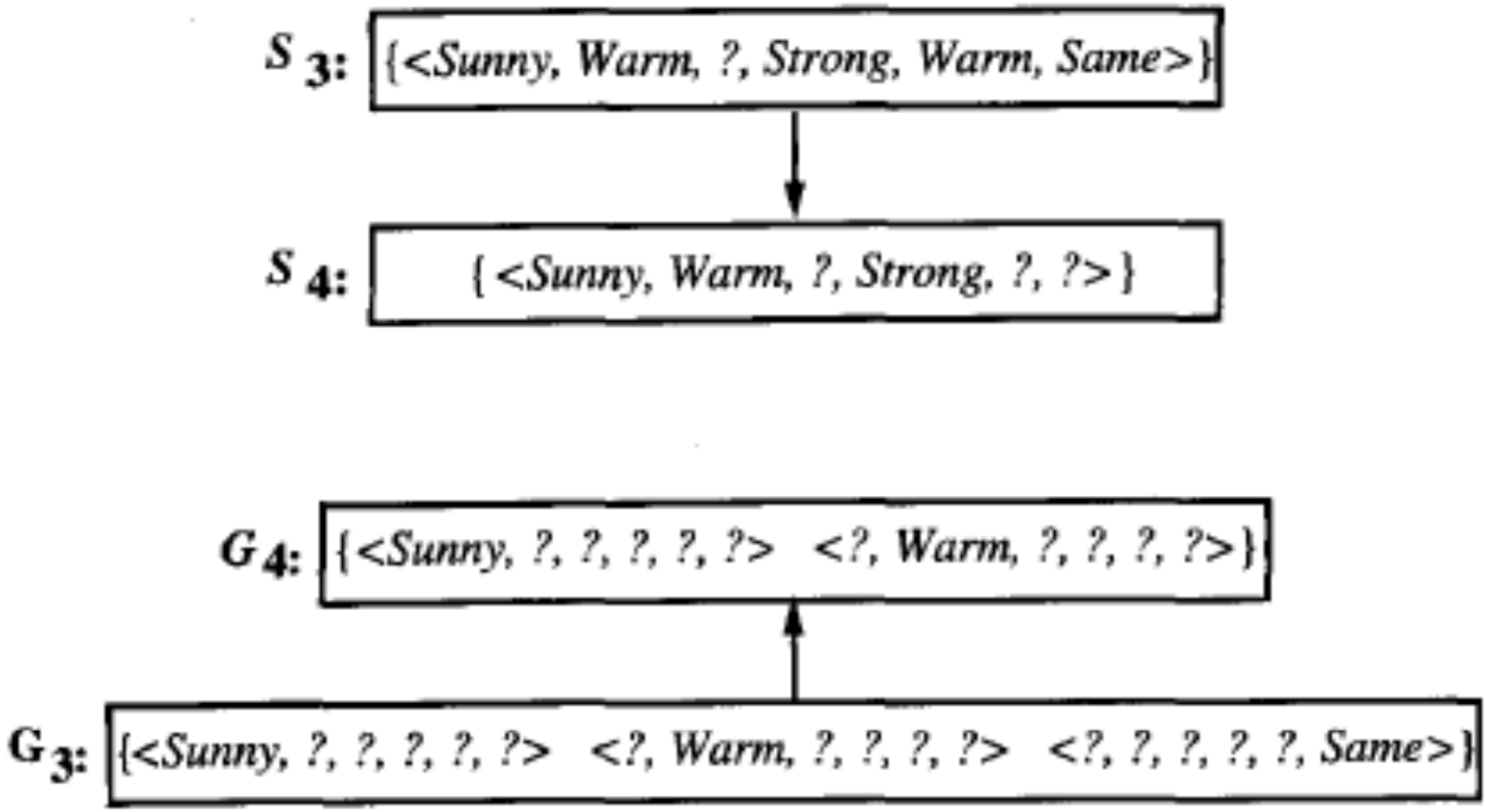
Training Example:

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

Candidate-Elimination Algorithm - Example

- Given that there are six attributes that could be specified to specialize **G2,** why are there only three new hypotheses in **G3?**
- For example, the hypothesis ***h*** = <?, ?, ***Normal,*** ?, ?, ?> is a minimal specialization of **G2** that correctly labels the new example as a negative example, but it is not included in **G3.**
  - The reason this hypothesis is excluded is that it is inconsistent with S2.
  - The algorithm determines this simply by noting that ***h*** is not more general than the current specific boundary, **S2.**
- In fact, the **S** boundary of the version space forms a summary of the previously encountered positive examples that can be used to determine whether any given hypothesis is consistent with these examples.
- The G boundary summarizes the information from previously encountered negative examples. Any hypothesis more specific than G is assured to be consistent with past negative examples

Candidate-Elimination Algorithm - Example



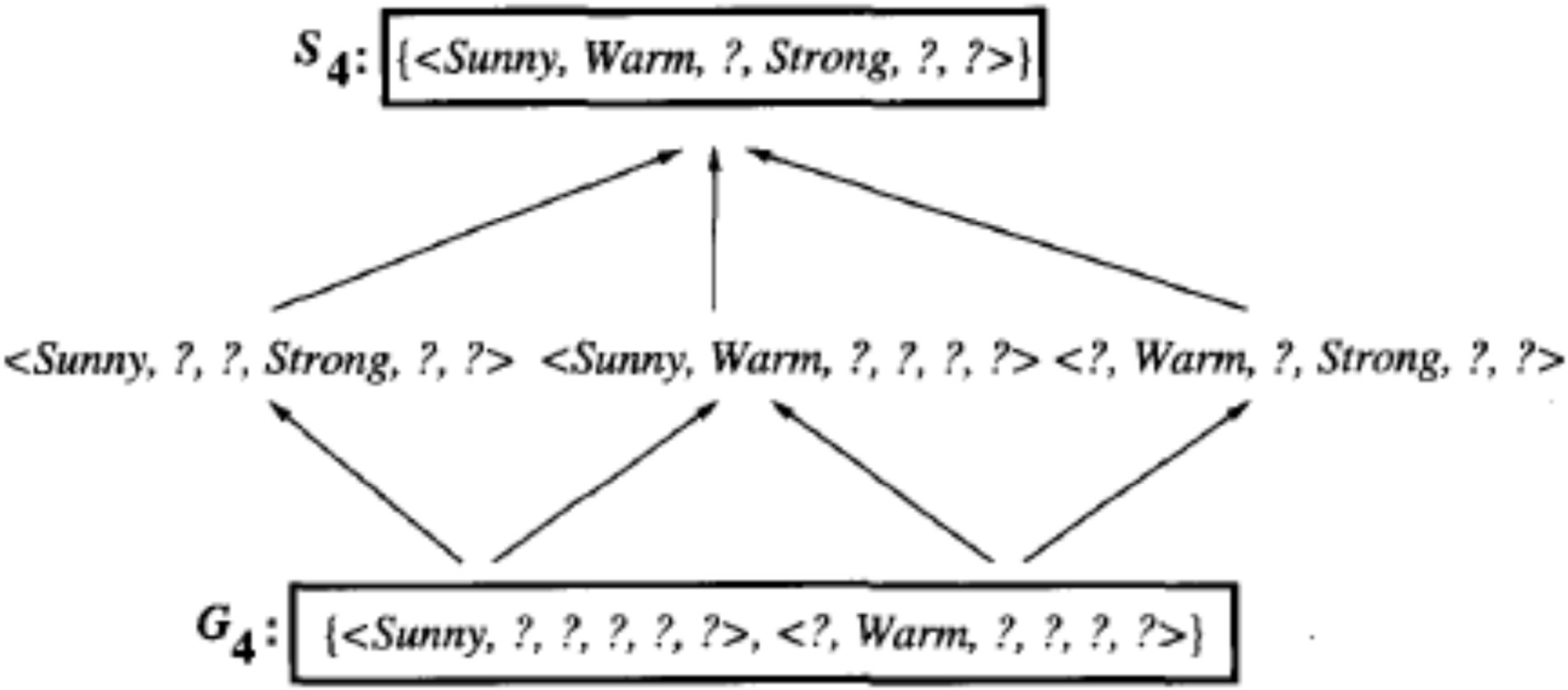
Training Example:

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

Candidate-Elimination Algorithm - Example

- The fourth training example further generalizes the S boundary of the version space.
- It also results in removing one member of the G boundary, because this member fails to cover the new positive example.
  - To understand the rationale for this step, it is useful to consider why the offending hypothesis must be removed from G.
  - Notice it cannot be specialized, because specializing it would not make it cover the new example.
  - It also cannot be generalized, because by the definition of G, any more general hypothesis will cover at least one negative training example.
  - Therefore, the hypothesis must be dropped from the G boundary, thereby removing an entire branch of the partial ordering from the version space of hypotheses remaining under consideration

Candidate-Elimination Algorithm – Example
Final Version Space



Candidate-Elimination Algorithm – Example
Final Version Space

- 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.
- This learned version space is independent of the sequence in which the training examples are presented (because in the end it contains all hypotheses consistent with the set of examples).
- As further training data is encountered, the S and G boundaries will move monotonically closer to each other, delimiting a smaller and smaller version space of candidate hypotheses.

Will Candidate-Elimination Algorithm Converge to Correct Hypothesis?

- The version space learned by the Candidate-Elimination Algorithm will converge toward the hypothesis that correctly describes the target concept, provided
  - There are no errors in the training examples, and
  - there is some hypothesis in H that correctly describes the target concept.
- What will happen if the training data contains errors?
  - The algorithm removes the correct target concept from the version space.
  - S and G boundary sets eventually converge to an empty version space if sufficient additional training data is available.
  - Such an empty version space indicates that there is no hypothesis in H consistent with all observed training examples.
- A similar symptom 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

What Training Example Should the Learner Request Next?

- 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).
  - This scenario covers situations in which the learner may conduct experiments in nature or in which a teacher is available to provide the correct classification.
  - We use the term query to refer to such instances constructed by the learner, which are then classified by an external oracle.
- Considering the version space learned from the four training examples of the EnjoySport concept.
  - What would be a good query for the learner to pose at this point?
  - What is a good query strategy in general?

What Training Example Should the Learner Request Next?

- The learner should attempt to discriminate among the alternative competing hypotheses in its current version space.
  - Therefore, it should choose an instance that would be classified positive by some of these hypotheses, but negative by others.
  - One such instance is <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.
- In general, the optimal query strategy for a concept learner is to generate instances that satisfy exactly half the hypotheses in the current version space.
- When this is possible, the size of the version space is reduced by half with each new example, and the correct target concept can therefore be found with only $\lceil \log_2 |VS| \rceil$ experiments.

How Can Partially Learned Concepts Be Used?

- Even though the learned version space 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.

- Let us assume that the followings are new instances to be classified:

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	?

How Can Partially Learned Concepts Be Used?

- ***Instance A*** was is classified as a positive instance by every hypothesis in the current version space.
- Because the hypotheses in the version space unanimously agree that this is a positive instance, 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.
- Regardless of which hypothesis in the version space is eventually found to be the correct target concept, it is already clear that it will classify instance A as a positive example.
- Notice furthermore that we need not enumerate every hypothesis in the version space in order to test whether each classifies the instance as positive.
  - This condition will be met if and only if the instance satisfies every member of S.
  - The reason is that every other hypothesis in the version space is at least as general as some member of S.
  - By our definition of more-general-than, if the new instance satisfies all members of S it must also satisfy each of these more general hypotheses.

How Can Partially Learned Concepts Be Used?

- ***Instance B*** is classified as a negative instance by every hypothesis in the version space.
  - This instance can therefore be safely classified as negative, given the partially learned concept.
  - An efficient test for this condition is that the instance satisfies none of the members of G.
- Half of the version space hypotheses classify ***instance C*** as positive and half classify it as negative.
  - Thus, the learner cannot classify this example with confidence until further training examples are available.
- ***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, and one approach we could take would be to output the majority vote, perhaps with a confidence rating indicating how close the vote was.

Inductive Bias - Fundamental Questions for Inductive Inference

- The Candidate-Elimination Algorithm will converge toward the true target concept provided it is given accurate training examples and provided its initial hypothesis space contains the target concept.

- 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?

Inductive Bias - A Biased Hypothesis Space

- In EnjoySport example, 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."

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

- From first two examples ➔ S2 : <?, Warm, Normal, Strong, Cool, Change>
  - This is inconsistent with third examples, and there are no hypotheses consistent with these three examples

PROBLEM: We have biased the learner to consider only conjunctive hypotheses.
➔ We require a more expressive hypothesis space.

Inductive Bias - 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.
  - Every possible subset of the instances X ➔ ***the power set of X.***

- What is the size of the hypothesis space H (the power set of X) ?
  - In EnjoySport, the size of the instance space X is 96.
  - The size of the power set of X is $2^{|X|}$ ➔ The size of H is 2^{96}
  - Our conjunctive hypothesis space is able to represent only 973of these hypotheses.

    ➔ a very biased hypothesis space

Inductive Bias - An Unbiased Learner : Problem

- Let the hypothesis space H to be the power set of X.
  - A hypothesis can be represented with disjunctions, conjunctions, and negations of our earlier hypotheses.
  - The target concept "Sky = Sunny or Sky = Cloudy" could then be described as

    <Sunny, ?, ?, ?, ?, ?> $\vee$ <Cloudy, ?, ?, ?, ?, ?>

NEW PROBLEM: our concept learning algorithm is now completely unable to generalize beyond the observed examples.

- three positive examples (xl,x2,x3) and two negative examples (x4,x5) to the learner.
- S : { x1 $\vee$ x2 $\vee$ x3 } and G : { $\neg$ (x4 $\vee$ x5) } ➔ NO GENERALIZATION
- Therefore, the only examples that will be unambiguously classified by S and G are the observed training examples themselves.

Inductive Bias – Fundamental Property of Inductive Inference

- **A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.**

- **Inductive Leap**: A learner should be able to generalize training data using prior assumptions in order to classify unseen instances.
- The generalization is known as **inductive leap** and our prior assumptions are the **inductive bias** of the learner.
- Inductive Bias (prior assumptions) of Candidate-Elimination Algorithm is that the target concept can be represented by a conjunction of attribute values, the target concept is contained in the hypothesis space and training examples are correct.

Inductive Bias – Formal Definition

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 ***Dc = {<x , c(x)>}*** be an arbitrary set of training examples of ***c***.

Let ***L(xi, Dc)*** denote the classification assigned to the instance ***xi*** by ***L*** after training on the data ***Dc.***

The **inductive bias** of ***L*** is any minimal set of assertions ***B*** such that for any target concept ***c*** and corresponding training examples ***Dc*** the following formula holds.

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

Inductive Bias – Three Learning Algorithms

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.

Inductive Bias: No inductive bias

CANDIDATE-ELIMINATION: 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.

Inductive Bias: the target concept can be represented in its hypothesis space.

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.

Inductive Bias: the target concept can be represented in its hypothesis space, and all instances are negative instances unless the opposite is entailed by its other know1edge.

Concept Learning - Summary

- Concept learning can be seen as a problem of searching through a large predefined space of potential hypotheses.
- The general-to-specific partial ordering of hypotheses provides a useful structure for organizing the search through the hypothesis space.
- The FIND-S algorithm utilizes this general-to-specific ordering, performing a specific-to-general search through the hypothesis space along one branch of the partial ordering, to find the most specific hypothesis consistent with the training examples.
- The CANDIDATE-ELIMINATION algorithm utilizes this general-to-specific ordering to compute the version space (the set of all hypotheses consistent with the training data) by incrementally computing the sets of maximally specific (S) and maximally general (G) hypotheses.

Concept Learning - Summary

- Because the S and G sets delimit the entire set of hypotheses consistent with the data, they provide the learner with a description of its uncertainty regarding the exact identity of the target concept. This version space of alternative hypotheses can be examined
  - to determine whether the learner has converged to the target concept,
  - to determine when the training data are inconsistent,
  - to generate informative queries to further refine the version space, and
  - to determine which unseen instances can be unambiguously classified based on the partially learned concept.
- The CANDIDATE-ELIMINATION algorithm is not robust to noisy data or to situations in which the unknown target concept is not expressible in the provided hypothesis space.

Concept Learning - Summary

- Inductive learning algorithms are able to classify unseen examples only because of their implicit inductive bias for selecting one consistent hypothesis over another.
- If the hypothesis space is enriched to the point where there is a hypothesis corresponding to every possible subset of instances (the power set of the instances), this will remove any inductive bias from the CANDIDATE-ELIMINATION algorithm .
  - Unfortunately, this also removes the ability to classify any instance beyond the observed training examples.
  - An unbiased learner cannot make inductive leaps to classify unseen examples.