MODULE 2

INTRODUCTION

Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn-to improve automatically with experience-the impact would be dramatic.

- Imagine computers learning from medical records which treatments are most effective for new diseases
- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
- Personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper

A successful understanding of how to make computers learn would open up many new uses of computers and new levels of competence and customization

Some successful applications of machine learning

- Learning to recognize spoken words
- Learning to drive an autonomous vehicle
- Learning to classify new astronomical structures
- Learning to play world-class backgammon

Why is Machine Learning Important?

- Some tasks cannot be defined well, except by examples (e.g., recognizing people).
- Relationships and correlations can be hidden within large amounts of data. Machine Learning/Data Mining may be able to find these relationships.
- Human designers often produce machines that do not work as well as desired in the environments in which they are used.
- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
- Environments change over time.
- New knowledge about tasks is constantly being discovered by humans. It may be difficult to continuously re-design systems “by hand”.

CONCEPT LEARNING

- Learning involves acquiring general concepts from specific training examples. Example: People 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. (Example: A function defined over all animals, whose value is true for birds and false for other animals).

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

A CONCEPT LEARNING TASK

Consider the example task of learning the target concept "Days on which ***Aldo*** enjoys his favorite water sport"

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

Table: Positive and negative training examples for the target concept ***EnjoySport.***

The task is to learn to predict the value of ***EnjoySport*** for an arbitrary day, based on the values of its other attributes?

What hypothesis representation is provided to the learner?

- Let's consider a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.
- Let each hypothesis be a vector of six constraints, specifying the values of the six attributes *Sky, AirTemp, Humidity, Wind, Water*, and *Forecast.*

For each attribute, the hypothesis will either

- Indicate by a "?' that any value is acceptable for this attribute,
- Specify a single required value (e.g., Warm) for the attribute, or
- Indicate by a "Φ" that no value is acceptable

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

The hypothesis that ***PERSON*** enjoys his favorite sport only on cold days with high humidity is represented by the expression

(?, Cold, High, ?, ?, ?)

The most general hypothesis-that every day is a positive example-is represented by

(?, ?, ?, ?, ?, ?)

The most specific possible hypothesis-that no day is a positive example-is represented by

(Φ, Φ, Φ, Φ, Φ, Φ)

Notation

- The set of items over which the concept is defined is called the ***set of instances***, which is denoted by X.

Example: X is the set of all possible days, each represented by the attributes: Sky, AirTemp, Humidity, Wind, Water, and Forecast

- The concept or function to be learned is called the ***target concept***, which is denoted by c.

c can be any Boolean valued function defined over the instances X

$$c: X \rightarrow \{O, 1\}$$

Example: The target concept corresponds to the value of the attribute ***EnjoySport*** (i.e., c(x) = 1 if ***EnjoySport*** = Yes, and c(x) = 0 if ***EnjoySport*** = No).

- Instances for which c(x) = 1 are called ***positive examples***, or members of the target concept.
- Instances for which c(x) = 0 are called ***negative examples***, or non-members of the target concept.
- The ordered pair (x, c(x)) to describe the training example consisting of the instance x and its target ***concept value c(x).***
- ***D*** to denote the set of available training examples

- The symbol ***H*** to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. Each hypothesis ***h*** in ***H*** represents a Boolean-valued function defined over ***X***

$$h: X \rightarrow \{O, 1\}$$

The goal of the learner is to find a hypothesis h such that h(x) = c(x) for all x in X.

- 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),
    - *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), "Φ" (no value is acceptable), or a specific value.

  - Target concept *c*: ***EnjoySport*** : X → {0, 1}
  - 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 X.

Table: The ***EnjoySport*** concept learning task.

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.

Example:
Consider the instances X and hypotheses H in the *EnjoySport* learning task. The attribute Sky has three possible values, and *AirTemp, Humidity, Wind, Water, Forecast* each have two possible values, the instance space X contains exactly

3.2.2.2.2.2 = 96 distinct instances

5.4.4.4.4.4 = 5120 syntactically distinct hypotheses within H.

Every hypothesis containing one or more "Φ" symbols represents the empty set of instances; that is, it classifies every instance as negative.

1 + (4.3.3.3.3.3) = 973. Semantically distinct hypotheses

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

h_1 = (Sunny, ?, ?, Strong, ?, ?)

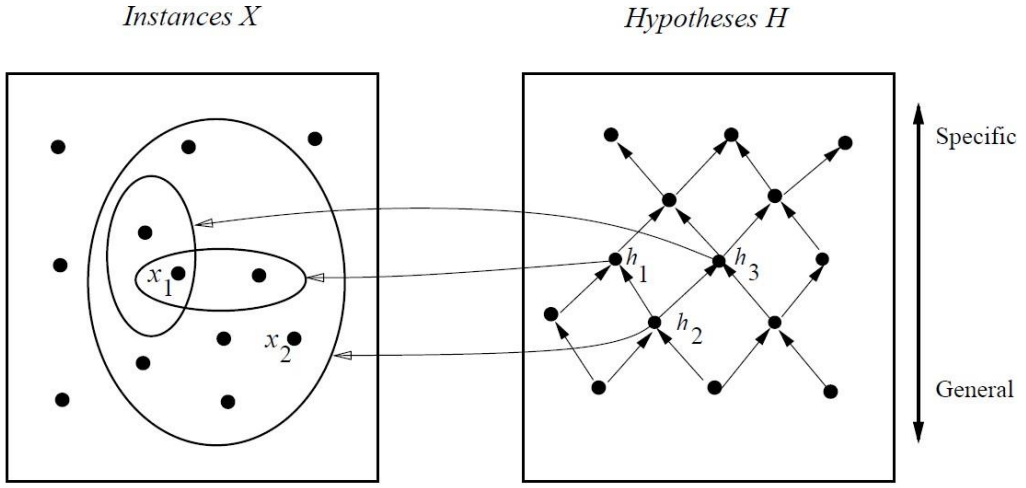
h_2 = (Sunny, ?, ?, ?, ?, ?)

- Consider the sets of instances that are classified positive by h_l and by h_2.
- h_2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by h_l will also be classified positive by h_2. Therefore, h_2 is more general than h_l.

Given hypotheses h_j and h_k, h_j is more-general-than or- equal do h_k if and only if any instance that satisfies h_k also satisfies h_i

Definition: Let h_j and h_k be Boolean-valued functions defined over X. Then h_j is ***more general-than-or-equal-to*** h_k (written $h_j \geq h_k$) if and only if

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



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, ?>

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

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

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*

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

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 - (Ø, Ø, Ø, Ø, Ø, Ø)

- Consider the first training example

  x_1 = <Sunny Warm Normal Strong Warm Same>, +

  Observing the first training example, it is clear that hypothesis *h* is too specific. None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next *more general constraint* that fits the example

  h_1 = <Sunny Warm Normal Strong Warm Same>

- Consider the second training example

  x_2 = <Sunny, Warm, High, Strong, Warm, Same>, +

  The second training example 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

  h_2 = <Sunny Warm ? Strong Warm Same>

- Consider the third training example

  x3 = <Rainy, Cold, High, Strong, Warm, Change>, -

  Upon encountering the third training the algorithm makes no change to h. The FIND-S algorithm simply ignores every negative example.

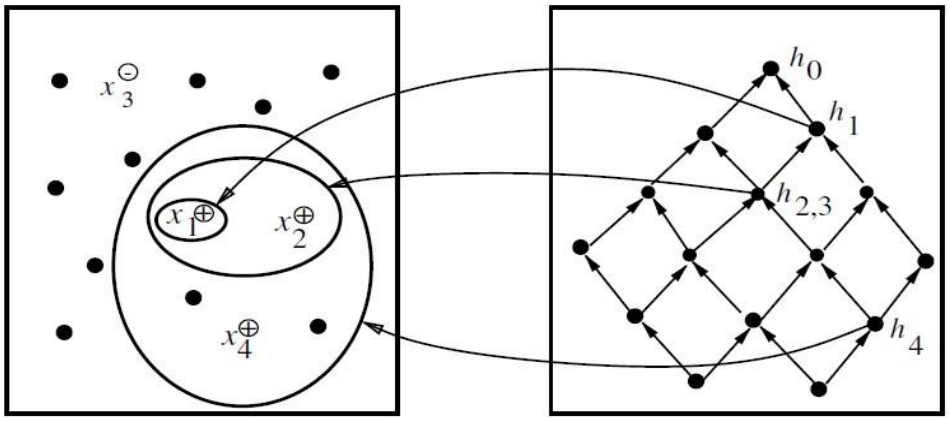
  h_3 = < Sunny Warm ? Strong Warm Same>

- Consider the fourth training example

  x_4 = <Sunny Warm High Strong Cool Change>, +

  The fourth example leads to a further generalization of h

  h_4 = < Sunny Warm ? Strong ? ? >



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 Sam
h_2 = <Sunny Warm ? Strong Warm Same>
h_3 = <Sunny Warm ? Strong Warm Same>
h_4 = <Sunny Warm ? Strong ? ? >

The key property of the FIND-S algorithm

- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples
- FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H, and provided the training examples are correct.

Unanswered by FIND-S

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all ***hypotheses consistent with the training examples***

Representation

Definition: consistent- 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.***

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

Note difference between definitions 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.
- An example *x* is said to ***consistent*** with hypothesis ***h*** iff ***h(x)*** = ***c(x)***

Definition: version space- 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

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

The LIST-THEN-ELIMINATION algorithm

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

1. ***VersionSpace c*** 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***

The LIST-THEN-ELIMINATE Algorithm

- List-Then-Eliminate works in principle, so long as version space is finite.
- However, since it requires exhaustive enumeration of all hypotheses in practice it is not feasible.

A More Compact Representation for Version Spaces

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

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

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

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

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

Theorem: Version Space representation theorem

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 →{O, 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,

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

To Prove:

1. Every h satisfying the right hand side of the above expression is in $\boldsymbol{VS_{H, D}}$

2. Every member of $\boldsymbol{VS_{H, D}}$ satisfies the right-hand side of the expression

Sketch of proof:

1. let g, h, s be arbitrary members of G, H, S respectively with $g \geq_g h \geq_g s$
   - By the definition of ***S,*** **s** must be satisfied by all positive examples in D. Because $h \geq_g s$, h must also be satisfied by all positive examples in D.
   - By the definition of ***G,*** g cannot be satisfied by any negative example in D, and because $g \geq_g h$ h cannot be satisfied by any negative example in D. Because h is satisfied by all positive examples in D and by no negative examples in D, h is consistent with D, and therefore h is a member of $VS_{H,D}$.
2. It can be proven by assuming some h in $VS_{H,D}$,that does not satisfy the right-hand side of the expression, then showing that this leads to an inconsistency

CANDIDATE-ELIMINATION Learning Algorithm

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

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- ELIMINTION algorithm using version spaces

An Illustrative 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

CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H

$G_0 \langle ?, ?, ?, ?, ?, ? \rangle$

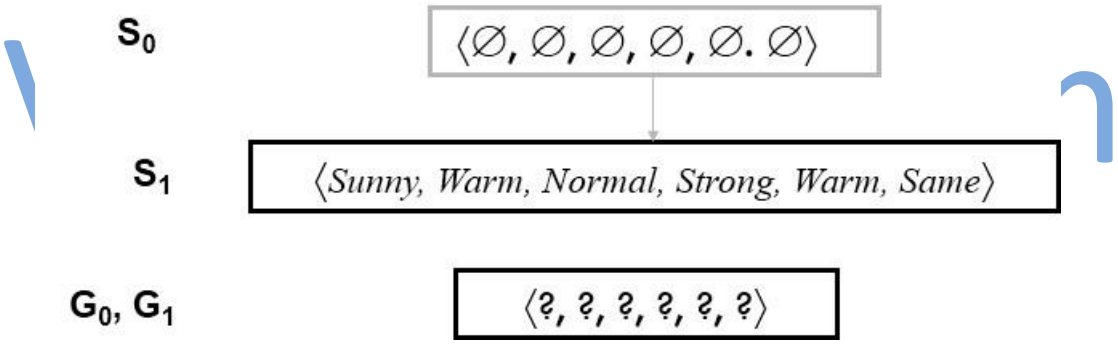
Initializing the S boundary set to contain the most specific (least general) hypothesis

$S_0 \langle \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle$

- When the first training example is presented, the CANDIDATE-ELIMINTION algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.
- The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example
- No update of the G boundary is needed in response to this training example because G_o correctly covers this example

For training example d,

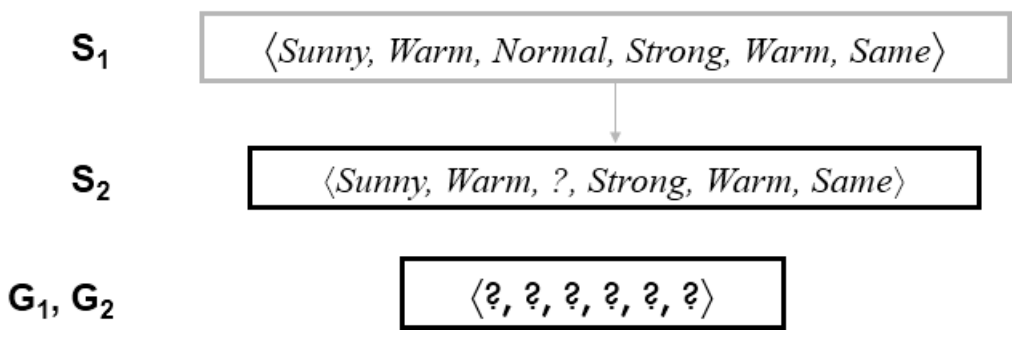
⟨Sunny, Warm, Normal, Strong, Warm, Same ⟩ +



- When the second training example is observed, it has a similar effect of generalizing S further to S_2, leaving G again unchanged i.e., $G_2 = G_1 = G_0$

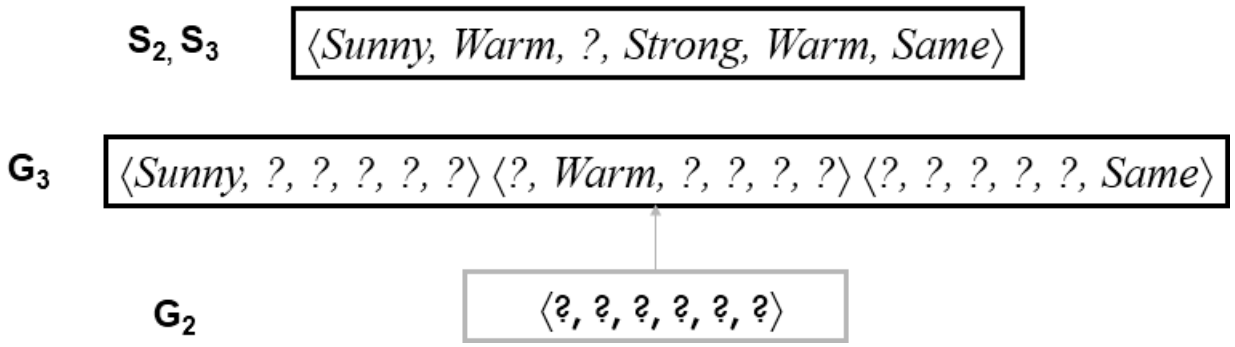
For training example d,

⟨Sunny, Warm, High, Strong, Warm, Same⟩ +



- Consider the third training example. This negative example reveals that the G boundary of the version space is overly general, that is, the hypothesis in G incorrectly predicts that this new example is a positive example.
- The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example

For training example d, ⟨Rainy, Cold, High, Strong, Warm, Change⟩ –

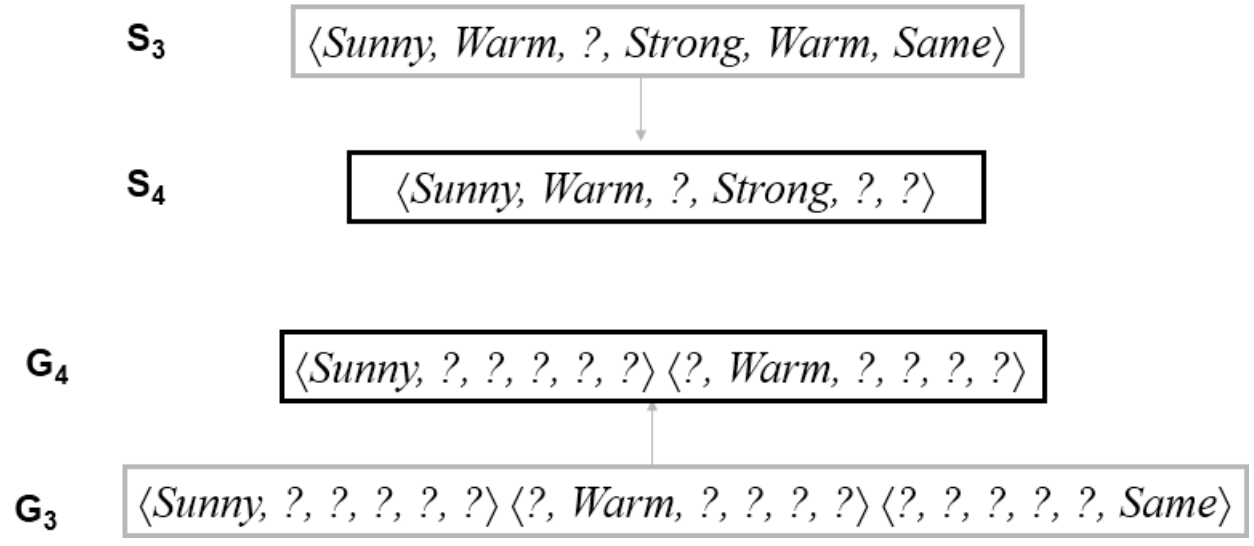


Given that there are six attributes that could be specified to specialize G_2, why are there only three new hypotheses in G_3?

For example, the hypothesis h = (?, ?, Normal, ?, ?, ?) is a minimal specialization of G_2 that correctly labels the new example as a negative example, but it is not included in G_3. The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples

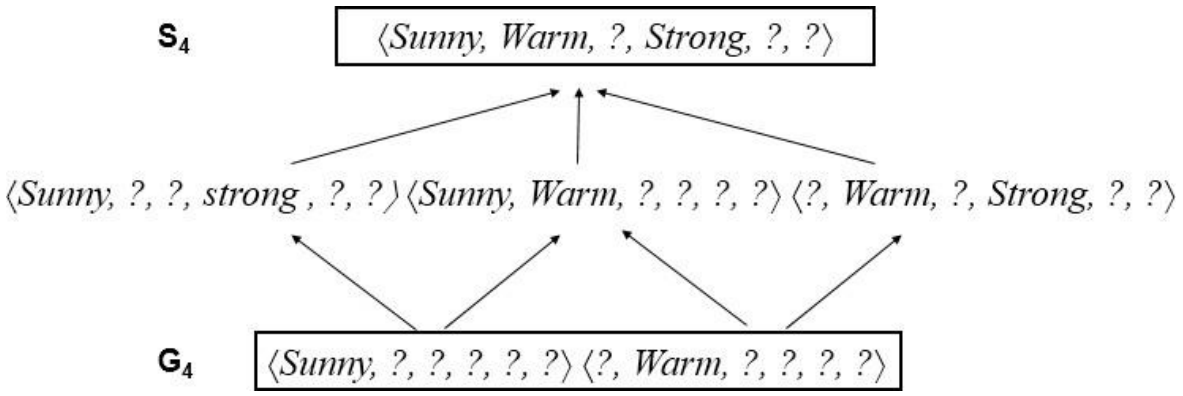
- Consider the fourth training example.

For training example d, ⟨Sunny, Warm, High, Strong, Cool Change⟩ +



- This positive 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

After processing these four examples, the boundary sets S_4 and G_4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



INDUCTIVE BIAS

The fundamental questions for inductive inference

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

These fundamental questions are examined in the context of the CANDIDATE-ELIMINTION algorithm

A Biased Hypothesis Space

- Suppose the target concept is not contained in the hypothesis space *H*, then obvious solution is to enrich the hypothesis space to include every possible hypothesis.
  - Consider the ***EnjoySport*** example in which the hypothesis space is restricted 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."
  - The following three training examples of disjunctive hypothesis, the algorithm would find that there are zero hypotheses in the version space

⟨Sunny Warm Normal Strong Cool Change⟩	Y
⟨Cloudy Warm Normal Strong Cool Change⟩	Y
⟨Rainy Warm Normal Strong Cool Change⟩	N

  - If Candidate Elimination algorithm is applied, then it end up with empty Version Space. After first two training example
    S= ⟨? Warm Normal Strong Cool Change⟩
  - This new hypothesis is overly general and it incorrectly covers the third negative training example! So H does not include the appropriate c.
  - In this case, a more expressive hypothesis space is required.

An Unbiased Learner

- The 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 representing every possible subset of the instances X.
- The set of all subsets of a set X is called the power set of X

  - In the ***EnjoySport*** learning task the size of the instance space X of days described by the six attributes is 96 instances.
  - Thus, there are 2^{96} distinct target concepts that could be defined over this instance space and learner might be called upon to learn.
  - The conjunctive hypothesis space is able to represent only 973 of these - a biased hypothesis space indeed

  - Let us reformulate the ***EnjoySport*** learning task in an unbiased way by defining a new hypothesis space H' that can represent every subset of instances
  - The target concept "Sky = Sunny or Sky = Cloudy" could then be described as

(Sunny, ?, ?, ?, ?, ?) v (Cloudy, ?, ?, ?, ?, ?)

The Futility of Bias-Free Learning

Inductive learning requires some form of prior assumptions, or inductive bias

Definition:

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

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

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

The below figure explains

- Modelling 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 modelling 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.