Concept learning

- Inferring a boolean-valued function from training examples of its input and output.
- Concept learning is also known as binary classification.
- Example: "EnjoySport"

Suppose want to learn target concept days on which Fred enjoys his favorite *water sport*

Concept learning …

- Input is a set of examples, one per day
  - describing the day in terms of a set of *attributes*
  - indicating (yes/no) whether Fred enjoyed his sport that day

Example	Sky	Temp	Humid	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

- Task: learn to predict the value of *EnjoySport* for an arbitrary day, given values of other attributes

Concept learning …

- Suppose hypotheses take form of conjunctions of constraints on instance attributes – e.g. specify allowed values of:
  Sky, Temp, Humid, Wind, Water, Forecast
  and suppose constraints take one of three forms:
  - *?* – any value acceptable
  - `Specified_value` – specific value required, e.g. *Warm* for the *Temp* attribute
  - $\emptyset$ – no value acceptable

- Then, hypotheses can be represented as vectors of such constraints.
  E.g. $\langle ?, Cold, High, ?, ?, ? \rangle$ – represents hypothesis
  Fred enjoys sport only on cold days with high humidity

Concept learning …

- Can describe the concept learning setting as follows.
  Given:
  - X a set of *instances* over which the concept is to be defined (each represented, e.g., as a vector of attribute values)
  - a *target function* or *concept* to be learned:

$$c : X \rightarrow \{0, 1\}$$

  - a set D of *training examples*, each of the form $\langle x, c(x) \rangle$ where $x \in X$ and $c(x)$ is the target concept value for x
    (Note: instances in D for which $c(x) = 1$ are called *positive examples*, those for which $c(x) = 0$ are *negative examples*)

Concept learning …

Find:

– a *hypothesis*, or estimate, of c.
I.e. supposing H is the set of all hypotheses, find $h \in H$, where $h : X \rightarrow \{0,1\}$ such that

$$h(x) = c(x) \text{ for all } x \in X$$

- Thus, concept learning can be viewed as **search** over the space of hypotheses, as defined by the hypothesis representation.

- Note: we want to find h identical to c over all of X **but**, only have information about c for training examples D.

- Inductive learning algorithms can only guarantee that hypotheses fit *training data*.

Concept learning …

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.

General-to-Specific Ordering of Hypotheses

- **Problem**: How should we search this space to find the target concept?

  A Solution: Start with the most specific hypothesis and, considering each training example in turn, generalise towards the most general hypothesis, stopping at the first hypothesis that 'covers' the training examples (FIND-S)

Concept learning …

– the most specific hypothesis:

$$\langle\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset\rangle \text{ (\textit{Fred enjoys sport on no day})}$$

– the most general hypothesis:

$$\langle ?,?,?,?,?,?\rangle \text{ (\textit{Fred enjoys sport on every day})}$$

- Intuitively a hypothesis h_i is more general than another h_j if
  - every instance that h_j classifies as positive h_i also classifies as positive, and
  - h_i classifies instances as positive that h_j does not

  E.g.

$$\begin{aligned} h_1 &= \langle Sunny, ?, ?, Strong, ?, ?\rangle \\ h_2 &= \langle Sunny, ?, ?, ?, ?, ?\rangle \end{aligned}$$

  h_2 is more general than h_1

Concept learning …

- More formally, instance x **satisfies** hypothesis h iff $h(x) = 1$.

  Define partial ordering relation *more_general_than_or_equal_to* holding between two hypotheses h_i and h_j in terms of the sets of instances that satisfy them:

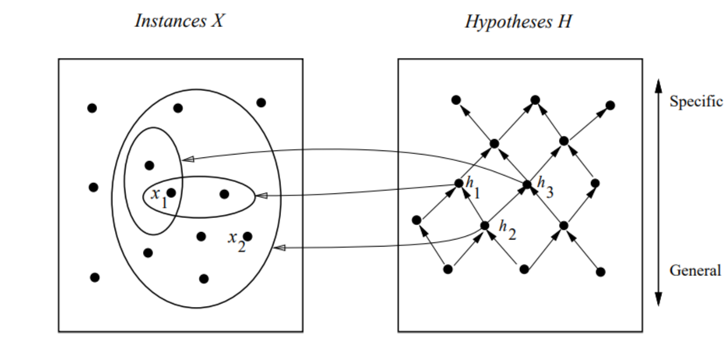
  h_i is *more_general_than_or_equal_to* h_j iff every instance that satisfies h_j also satisfies h_i

  or

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

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

Concept learning …



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, ?>

- Note that in the example:

$$h_2 \geq_g h_1 \qquad h_2 \geq_g h_3$$
$$h_1 \ngeq_g h_3 \qquad h_3 \ngeq_g h_1$$

FIND-S:
Finding a Maximally Specific Hypothesis

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 in h 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

FIND-S …

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 ? ?* >

Note: negative training instances completely ignored by FIND-S

FIND-S …

- For hypothesis spaces described by conjunctions of attribute constraints (e.g. *H* for *EnjoySport*), FIND-S is guaranteed to output the most specific hypothesis in *H* consistent with positive training examples

- FIND-S also guaranteed to output hypothesis consistent with negative examples provided
  - correct target concept is in *H*
  - training examples are correct

FIND-S …

But, there are problems with FIND-S:

– may be multiple hypotheses consistent with the training data – FIND-S will find one, but give no indication of whether there may be others

– FIND-S always proposes maximally specific hypothesis – why prefer this to, e.g., maximally general?

– FIND-S has serious problems when training examples are inconsistent which frequently happens with noisy "real" data

Version Spaces + Candidate Elimination

- One limitation of the FIND-S algorithm is that it outputs just one hypothesis consistent with the training data – there might be many.

  To overcome this, introduce notion of **version space** and algorithms to compute it.

- 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 **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.

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

Version Spaces + Candidate Elimination

- Note difference between definitions of *consistent* and *satisfies*:
  - an example x *satisfies* hypothesis h when $h(x) = 1$, regardless of whether x is $+$ve or $-$ve example of target concept
  - an example x is *consistent* with hypothesis h iff $h(x) = c(x)$

The List-Then-Eliminate Algorithm

- Can represent version space by listing all members.
- Leads to LIST-THEN-ELIMINATE concept learning 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 $VersionSpace$

- 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.

The Candidate-Elimination Algorithm

- The CANDIDATE-ELIMINATION algorithm is similar to LIST-THEN-ELIMINATE algorithm but uses a more compact representation of version space.
  - represents version space by its most general and most specific members

- The CANDIDATE-ELIMINATION algorithm represents the version space by recording only the most general members (G) and its most specific members (S)
  - other intermediate members in general-to-specific ordering can be generated as needed

The Candidate-Elimination Algorithm

- The **General boundary**, G, of version space $VS_{H,D}$ is the set of its maximally general members

- The **Specific boundary**, S, of version space $VS_{H,D}$ is the set of its maximally specific members

- **Version Space Representation Theorem**
  Every member of the version space lies between these boundaries

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

  where $x \geq_g y$ means x is more general or equal to y

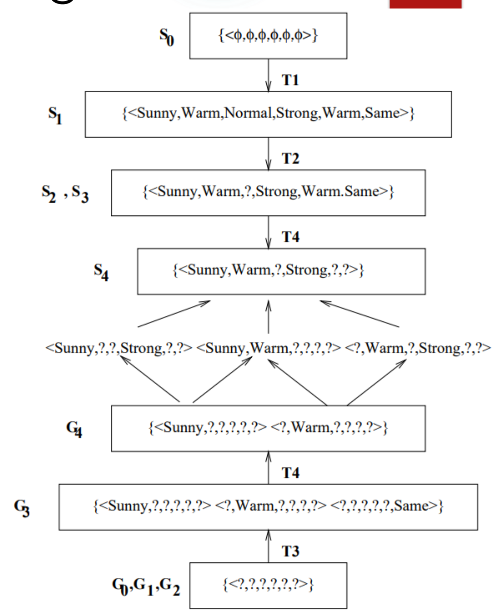
The Candidate-Elimination Algorithm

- Intuitively, CANDIDATE-ELIMINATION algorithm proceeds by
  - initialising G and S to the maximally general and maximally specific hypotheses in H
  - considering each training example in turn and
    - using positive examples to drive the maximally specific boundary up
    - using negative examples to drive the maximally general boundary down

$G \leftarrow$ maximally general hypotheses in H
$S \leftarrow$ 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
      1. h is consistent with d, and
      2. 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
      1. h is consistent with d, and
      2. some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

The Candidate-Elimination Algorithm



Training Examples:

T1: $\langle Sunny, Warm, Normal, Strong, Warm, Same \rangle, Yes$

T2: $\langle Sunny, Warm, High, Strong, Warm, Same \rangle, Yes$

T3: $\langle Rainy, Cold, High, Strong, Warm, Change \rangle, No$

T4: $\langle Sunny, Warm, High, Strong, Cool, Change \rangle, Yes$

The Candidate-Elimination Algorithm

- Version space learned by CANDIDATE-ELIMINATION algorithm will converge towards correct hypothesis provided:
  - no errors in training examples
  - there is a hypothesis in H that describes target concept

  In such cases algorithm may converge to empty version space
- If algorithm can request next training example (e.g. from teacher) can increase speed of convergence by requesting examples that split the version space
  - E.g. T5: $\langle Sunny, Warm, Normal, Light, Warm, Same \rangle$ satisfies 3 hypotheses in previous example
    * If T5 positive, S generalised, 3 hypotheses eliminated
    * If T5 negative, G specialised, 3 hypotheses eliminated

The Candidate-Elimination Algorithm

- As noted, version space learned by CANDIDATE-ELIMINATION algorithm will converge towards correct hypothesis provided:
  - no errors in training examples
  - there is a hypothesis in H that describes target concept