K. SRINIVASA RAO

EURCS 606
AI & MACHINE LEARNING
3/4 B.Tech. II Semester

UNIT-I

3.12.2014

Intelligence: The ability to reason, to trigger new thoughts, to perceive and learn is "Intelligence".

Artificial Intelligence: AI is a branch of computer science concerned with the study and creation of computer systems that exhibit some form of intelligence: systems that learn new concepts and tasks, systems that can understand a natural language or perceive and comprehend a visual scene, and systems that perform other types of feats that require human types of intelligence.

An understanding of AI requires an understanding of related terms such as intelligence, reasoning, knowledge, cognition, learning, etc.

Problem areas (Task Domains) of AI: (From AI by Rich & Knight)

(Mundane – earthly, ordinary, common)

Mundane Tasks
- Perception
  - Vision
  - Speech
- Natural Language
  - Understanding
  - Generation
  - Translation
- Commonsense reasoning
- Robot control

Formal Tasks
- Games
  - chess
  - Backgammon
  - Checkers
  - Go
- Mathematics
  - Geometry
  - Logic
  - Integral Calculus
  - Proving properties of programs

Expert Tasks
- Engineering
  - Design
  - Fault Finding
  - manufacturing planning
- Scientific analysis
- Medical diagnosis
- Financial analysis

(From: 'Machine Learning' by Tom M. Mitchell)
Chapter 1. Introduction (all topics)

Machine Learning (ML):
Learning by computers:

Exs:
1. Computers learning from medical records which treatments are most effective for new diseases.
2. Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
3. Personal software assistants learning the interests of users reading daily newspapers.

- ML algorithms are being used in Data Mining.

Applications of ML:

1. Learning to recognize spoken words (SPHINX system)
2. Learning to drive autonomous vehicle on public highways (ALVINN system)
3. Learning to classify new astronomical structures (Decision Tree learning algorithms have been used by NASA to learn how to classify celestial objects)
4. Learning to play world-class backgammon (Famous program: TD-GAMMON learns by playing 10,00,000 practice games against itself)

- ML is a multidisciplinary field.
  It draws on results from:
  - AI
  - Probability and Statistics
  - Computational Complexity Theory
  - Control Theory
  - Information Theory
  - Philosophy
  - Psychology
  - Neurobiology, and other fields.

Well-posed Learning Problems:

(posed : asserted or stated)

Machine Learning (ML) - 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.

Learning Problems:

	Task T	Performance measure P	Training experience E
1. Checkers	Playing checkers	Percent of games won against opponents	Playing practice games against itself
2. Handwriting recognition	Recognizing and classifying handwritten words with images	Percent of words correctly classified	A database of handwritten words with given classifications
3. Robot driving	Driving on public four-lane highways using vision sensors	Average distance traveled before an error (as judged by human overseer)	A sequence of images and steering commands recorded while observing a human driver

Designing a Learning System

(Design choices)

1. Choosing the Training Experience

8-puzzle (or) chess → states

- The type of training experience available can have a significant impact on success or failure of the learner.
- One key attribute is whether the training experience provides direct or indirect feedback.

Direct information:

In learning to play checkers, the direct training examples consist of individual checker board states and the correct move for each.

Indirect information:

It consists of the move sequences and

final outcomes of various games played.

7.12.14 Case 1 (direct information) is easier than the second case.

In case 2, the learner faces an additional problem of 'credit assignment', or determining the degree to which each move in the sequence deserves credit or blame for the final outcome.

Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal.

The 2nd important attribute of the training experience is the 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.

Alternatively, the learner might itself propose board states and ask the teacher for the correct move (for confusing states).

Or the learner may have complete control over both the board states and (indirect) training classifications.

A 3rd important attribute is how well it represents the distribution of examples over which the final system performance P must be measured.

In general, learning is most reliable when the training examples follow a distribution similar to that of future test examples.

If the training experience E consists only of games played against itself, then this experience may not be completely sufficient to solve future problems.

A checkers learning problem:

Task T
Performance measure P
Training experience E
} as mentioned earlier (in Pg. 4)

In order to complete the design of the learning system, we must now choose:

1. the exact type of knowledge to be learned
2. a representation for this target knowledge
3. a learning mechanism.

2. Choosing the Target Function

(to determine exactly what type of knowledge will be learned and how this will be used by the performance program)

Ex: A checkers-playing program that can generate the legal moves.

For the type of information to be learned, we can design a program or a function, that chooses the best move for any given board state.

For ex; ChooseMove : B → M

where B — is the set of legal board states
M — " " legal moves and
ChooseMove — " function that accepts any one legal board state as input and produces one legal move as output.

ChooseMove — is the Target function here.

(For more idea, refer to Chap:2 in AI (Rich & Knight); idea of states, current state, new state and rules).

An alternative target function that is easier to learn (than ChooseMove) is an evaluation function that assigns a numerical score to any given board state.

Ex: $V : B \rightarrow \mathbb{R}$

Function V maps any legal board state from the set B to some real value ($\mathbb{R}$ is the set of real numbers).

We want to assign higher scores to better board states using the target function V.

If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position. This can be accomplished by generating the legal successor board states from a given state and then using V to choose the best successor state and therefore the best legal move.

Values of Target function V:

Suppose b is an arbitrary board state.

1. If b is a final board state that is won, then $V(b) = 100$.
2. If " " " lost, then $V(b) = -100$.
3. If " " " drawn, then $V(b) = 0$.
4. If b is not a final state in the game, then $V(b) = V(b')$, where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game (assuming the opponent plays optimally, as well).

This recursive definition specifies a value of V(b) for every board state b, but it is not efficiently computable by the checkers playing program.

Hence we say that it is a nonoperational definition. We want an operational definition of V which can evaluate states and select moves within realistic time bounds.

So we need to discover an operational description of the ideal target function V.

✓ The process of learning the target function is often called function approximation (as the learning algorithms acquire only some approximation to the target function).

- We use the symbol $\hat{V}$ to refer to the function that is actually learned by our program, to distinguish it from V.

3. Choosing a Representation for the Target function

There are many options.

Represent $\hat{V}$ using:

(i) A large table with a distinct entry specifying the value for each distinct board state.

(ii) A collection of rules that match against features of the board state.

(iii) A quadratic polynomial function of predefined board features.

(iv) An artificial neural network.

We choose a simple representation for any given board state. The function $\hat{v}$ will be calculated as a linear combination of the following board features:

- x_1 : the no. of black pieces on the board.
- x_2 : the " red " "
- x_3 : the " black kings "
- x_4 : the " red " "
- x_5 : the no. of black pieces threated by red (which can be captured on red's next turn).
- x_6 : the no. of red pieces threated by black.

Now $\hat{v}(b)$ as a linear function:

$$\hat{v}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

where w_0 through w_6 are numerical coefficients (or weights) to be chosen by the learning algorithm.

Learned values of w_1 through w_6 will determine the relative importance of the various board features, in determining the value of the board.

w_0 will provide an additive constant to the board value.

Partial design of a checkers-learning program:

- Task T
- Performance measure P
- Training experience E

} as mentioned earlier

- Target function: V : Board $\rightarrow \mathbb{R}$

- Target function representation

$$\hat{V}(b) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5 + \omega_6 x_6$$

First three — correspond to the specification of the learning task

Final two — constitute design choices for the implementation of the learning program.

9.12.14

4. Choosing a Function Approximation Algorithm

- In order to learn the target function $\hat{V}$, we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b.

ie, each training example is an ordered pair of the form: $\langle b, V_{train}(b) \rangle$.

Ex: b is a board state in which black has won the game.

Hence $V_{train}(b) = +100$.

Suppose the ordered pair is:

$\langle\langle x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0\rangle, +100\rangle$

($x_2=0$ indicates that there are no red pieces)

Below is the procedure

- that derives such training examples from the indirect training experience,
- then adjusts the weights w_i to best fit these training examples.

(i) Estimating Training Values :

- The learner has indirect training experience.
So, the only information available to the learner is whether the game was eventually won or lost.
- Therefore, we require training examples that assign specific scores to specific board states.
This assignment is easy for the states that correspond to the end of the game.
But how to assign scores to intermediate states?

The approach :

Suppose b – any intermediate board state

$V_{train}(b)$ – training value of b

Successor (b) – next board state following b for which it is again the program's turn to move.
(ie, the board state following the program's move and the opponent's response)

$\hat{V}$ – learner's current approximation to V

Then, the

Rule for estimating training values :

$$V_{train}(b) \leftarrow \hat{V}(\text{Successor}(b))$$

ie, we use estimates of the value of Successor(b) to estimate the value of the board state b.

- we can see that this will make sense if $\hat{V}$ tends to be more accurate for board states closer to game's end.

14.12.14

(ii) Adjusting the weights :

Our learning algorithm has to choose the weights w_i to best fit the set of training examples $\{\langle b, V_{train}(b)\rangle\}$.

Best fit means — defining the best hypothesis (or set of weights) that minimizes the squared error E between the training values and the values predicted by the hypothesis $\hat{V}$.

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{training examples}} (V_{train}(b) - \hat{V}(b))^2$$

- Several algorithms are known for finding weights of a linear function that minimize E defined in this way.
- Here we require an algorithm that will incrementally refine the weights as new training examples become available and that will be robust to errors in these estimated training values.
- One such algorithm is called "Least Mean Squares" or "LMS training rule".
- For each observed training example, it adjusts the weights a small amount in the direction that reduces the error on this training example.

(LMS performs a stochastic gradient-descent search through the space of possible hypotheses (weight values) to minimize the squared error E.)

LMS weight update rule

For each training example $\langle b, V_{train}(b)\rangle$,

- Use the current weights to calculate $\hat{V}(b)$
- For each weight w_i, update it as

$$w_i \leftarrow w_i + \eta\,(V_{train}(b) - \hat{V}(b))\,x_i$$

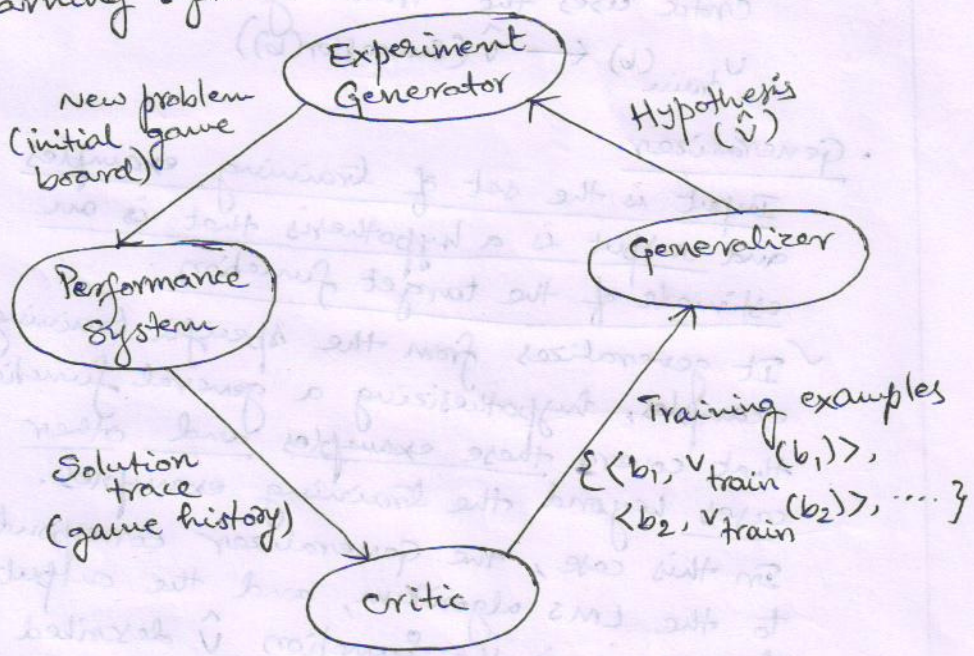
Here,
η — is a small constant (ex: 0.1) that moderates the size of the weight update.

when the error $(V_{train}(b) - \hat{V}(b))$ is
zero — no weights are changed
positive — (when $\hat{V}(b)$ is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}(b)$, reducing the error.

If the value of some feature x_i is zero, then its weight is not altered regardless of the error. (ie, $w_i \leftarrow w_i + 0$)

5. The Final Design

It has four distinct program modules that represent the central components in many learning systems.



- **Performance System**

This module must solve the given performance task (ie, playing checkers), by using the learned target function(s).

It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

It selects its next move at each step as determined by the learned $\hat{V}$ evaluation function. Its performance improves if the function becomes increasingly accurate.

- **Critic**

This takes trace of the game as input and produces a set of training examples as output.

Each training example corresponds to some game state.

Critic uses the training rule:

$$V_{train}(b) \leftarrow \hat{V}(Successor(b))$$

- **Generalizer**

Input is the set of training examples and output is a hypothesis that is an estimate of the target function.

✓ It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

In this case, the Generalizer corresponds to the LMS algorithm, and the output hypothesis is the function $\hat{V}$ described by the learned weights $w_0, \ldots, w_6$.

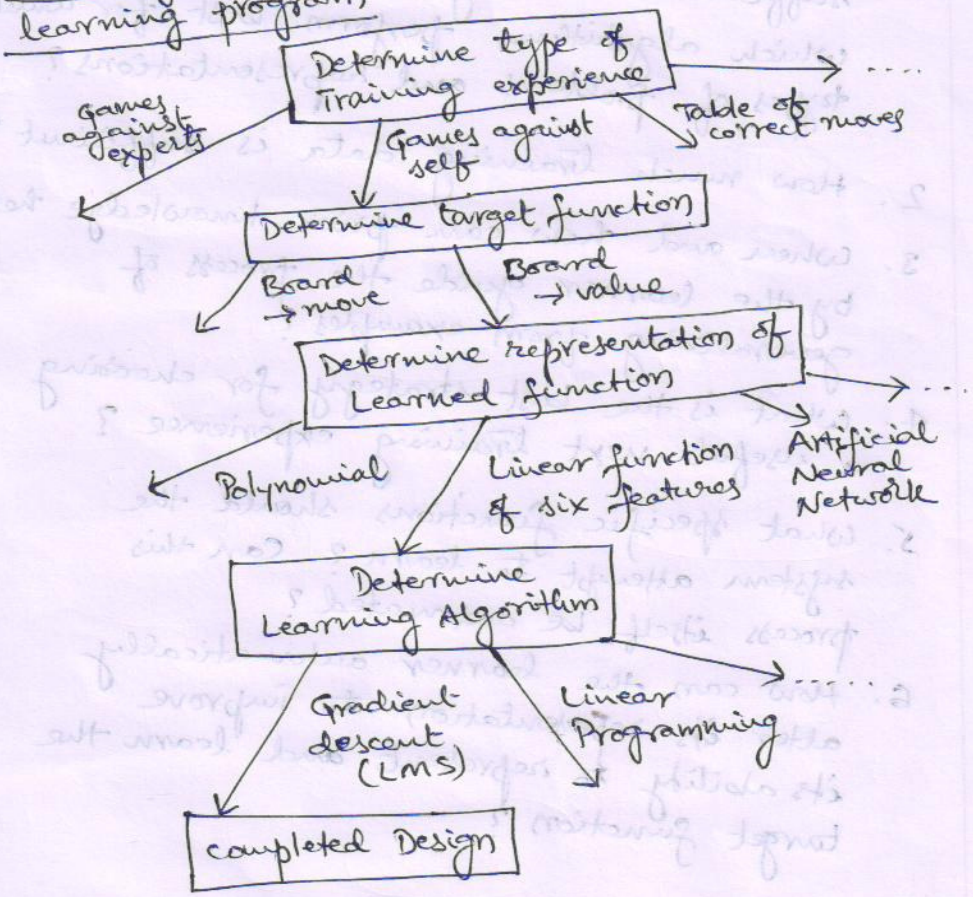
• Experiment Generator

Input is the current hypothesis (currently learned function) and output is a new problem (ie, initial board state) for the Performance system to explore.

Its role is to pick new practice problems that will maximize the learning rate of the overall system.

In our example, this module follows a very simple strategy: it always proposes the same initial game board to begin a new game.

Summary of choices in designing the checkers learning program



Our learning program designed in this way, may not beat the human checkers world champion. This is because the linear function representation for $\hat{V}$ is too simple.

Issues in Machine Learning:

The field of ML is concerned with answering questions such as the following:

1. 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?
2. How much training data is sufficient?
3. When and how can prior knowledge held by the learner guide the process of generalizing from examples?
4. What is the best strategy for choosing a useful next training experience?
5. What specific functions should the system attempt to learn? Can this process itself be automated?
6. How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

— *** —

K. Srinivasa Rao
Assoc. Prof.
Dept. of CSE, GIT, GU.

16.12.14

Chapter 2

CONCEPT LEARNING AND THE GENERAL-TO-SPECIFIC ORDERING

Concept — Describing some subset of objects or events defined over a larger set.

Ex: Set of Student names: A, B, C, X, Y, Z
Set of Dept. names: D1, D2, D3
Set of college names: C1, C2

Concepts: {A, D2, C1} {C, D2, C1}
{B, D1, C2} :

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

Concept Learning Task

Table 1

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

Positive and Negative training examples for the target concept EnjoySport.

- Table 1 describes a set of example days, each represented by a set of attributes.
- The attribute EnjoySport indicates whether or not Aldo (a person) enjoys this favourite water sport on this day.

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

25.12.14

Hypothesis Representation for the Learner:

Each hypothesis consists of a conjunction of constraints on the instance attributes.

Here, each hypothesis is a vector of six constraints, specifying the values of the six attributes (from Sky to Forecast).

Each attribute can have a:

- Specific value (ex: warm)
- "?" (indicates that any value is acceptable)
- "ϕ" (indicates that no value is acceptable)

Positive Example:

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

Ex: The hypothesis that Allo enjoys his favourite sport only on cold days with high humidity (independent of the values of other attributes) is:

$\langle ?, \text{Cold}, \text{High}, ?, ?, ? \rangle$

For this EnjoySport is 'Yes' and this is a positive example.

- Every day is a Positive example (the most general hypothesis) is represented by:

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

- No day is a Positive example (the most specific possible hypothesis) is:

  $\langle \phi, \phi, \phi, \phi, \phi, \phi \rangle$

The EnjoySport concept Learning Task:

It requires learning the set of days for which EnjoySport = yes.

The General Form:

Given

- Instances X: Possible days, each described by the attributes
  - Sky (with possible values Sunny, Cloudy and Rainy)
  - AirTemp (with values Warm and Cold)
  - Humidity (with values Normal and High)
  - Wind (with values Strong and Weak)
  - Water (with values Warm and Cool), and
  - Forecast (with values Same and Change).
- Hypotheses H: Each hypothesis is described by a conjunction of constraints on the attributes Sky, AirTemp, Humidity, Wind, Water and Forecast.

  The constraints may be:

  "?" (any value is acceptable),

  "ϕ" (no value is acceptable) or

  a specific value.

- Target concept c: EnjoySport : $X \rightarrow \{0,1\}$.
- Training Examples D: Positive and Negative examples of the target function (see Table 1).

Determine

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

X — the set of instances (each instance is x)

H — " hypotheses (each hypothesis is h)

C — the Target concept (the concept or function to be learned)

— any boolean-valued function defined over the instances X; that is $c: X \rightarrow \{0,1\}$.

($c(x) = 1$ if EnjoySport = Yes and $c(x) = 0$ if EnjoySport = No).

D — the set of training examples (each consisting of an instance x from X, along with its target concept value $c(x)$)

Positive examples (or members of the target concept) — instances for which $c(x) = 1$.

Negative examples (or nonmembers of the target concept) — instances for which $c(x) = 0$.

- Given D and c, the problem faced by the learner is to hypothesize, or estimate c.

  Hypothesis h (in H) is a boolean-valued function defined over X;

  ie, $h: X \rightarrow \{0, 1\}$.

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

The Inductive Learning 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".

- Inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data.
- Our assumption is that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed 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.

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

Ex: EnjoySport Learning Task

Suppose consider:

Attributes	Possible values
Sky	3
AirTemp	2
Humidity	2
Wind	2
Water	2
Forecast	2

Then the instance space X contains exactly $3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96$ distinct instances.

This 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.

✓ Our interest is in algorithms capable of efficiently searching very large or infinite hypothesis spaces, to find the hypotheses that best fit the training data.

General-to-Specific Ordering of Hypotheses:

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

Ex: (for General-to-specific ordering)

Consider the hypotheses:

$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$

$h_2 = \langle Sunny, ?, ?, ?, ?, ? \rangle$

h_2 imposes fewer constraints on the instance and hence it classifies more instances as positive. In fact, any instance classified positive by h_1 will also be classified positive by h_2. Therefore, we say that 'h_2 is more general than h_1'.

more-general-than-or-equal-to relationship:

suppose x is an instance in X and h is a hypothesis in H.

we say that x satisfies h iff $h(x) = 1$.

Definition:

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

Complete 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 $h_j \geq_g h_k$) if and only if

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

strictly more-general-than:

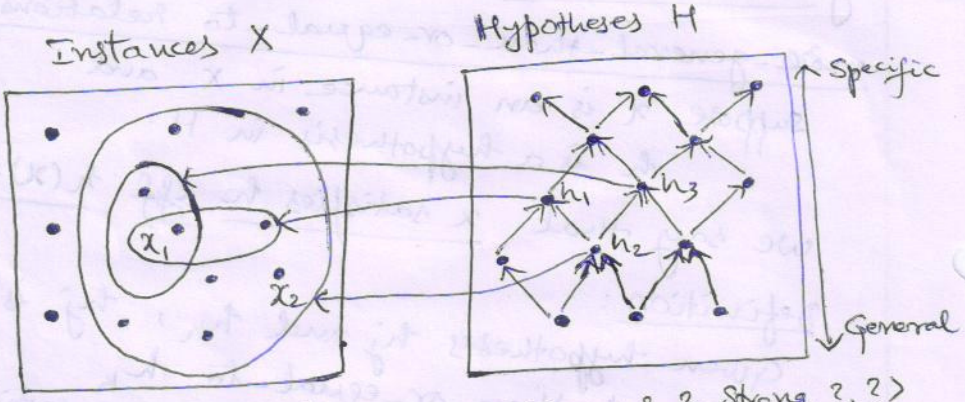
h_j is strictly more-general-than h_k (written $h_j >_g h_k$) if and only if

$$(h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j).$$

more_specific_than : (inverse definition)

h_j is more_specific_than h_k when h_k is more_general_than h_j.

Ex: Consider the following figure (to illustrate the above).



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, ?>

27.12.14

From the above figure,

- h_2 is more general than h_1 (because every instance that satisfies h_1 also satisfies h_2)
- h_2 is more general than h_3.
- Neither h_1 nor h_3 is more general than the other. (Although the instances satisfied by these two hypotheses intersect, neither set subsumes the other.)
- The $\geq_g$ and $>_g$ relations are defined independent of the target concept. (they depend only on the instances and not on the classification of those instances.)

- Formally, the $\geq_g$ relation defines a partial order over the hypothesis space H (the relation is reflexive, antisymmetric and transitive).

  (informally, there may be h_1 and h_3 such that $h_1 \not\geq_g h_3$ and $h_3 \not\geq_g h_1$)

FIND-S: Finding a Maximally Specific Hypothesis

Introduction:

To organize the search for a hypothesis consistent with the observed training examples, begin with the most specific possible hypothesis in H, then generalize this hypothesis each time it fails to cover an observed positive training example.

("covering" a positive example means if the hypothesis correctly classifies the example as positive.)

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.

Explanation:

From Table 1 of EnjoySport Task, consider the most specific hypothesis in H (as step 1 of algorithm),

$h \leftarrow \langle \phi, \phi, \phi, \phi, \phi, \phi \rangle$

- Now, none of the "ϕ" constraints in h are satisfied by the first training example, so each is replaced by the next more general constraint the fits the example. So,

$h \leftarrow \langle$Sunny, Warm, Normal, Strong, Warm, Same$\rangle$

h classifies training example 1 as positive, but h is still very specific.

(this is 'do nothing' part of step 2 of the algorithm)

- Next, the training example 2 (also positive) forces the algorithm to further generalize h.

So h will be:

$h \leftarrow \langle$Sunny, Warm, ?, Strong, Warm, Same$\rangle$

(only the value of 3rd attribute differs in the training examples 1 and 2; hence make it general by writing '?')

Now h can classify training examples 1 and 2 as positive.

28.12.14

- The third training example is negative. The algorithm makes no change to h.

In fact, FIND-S Algorithm simply ignores every negative example.

In the current case, h is already consistent with the new negative example (ie, h correctly classifies this example as negative) and hence

no revision is needed.

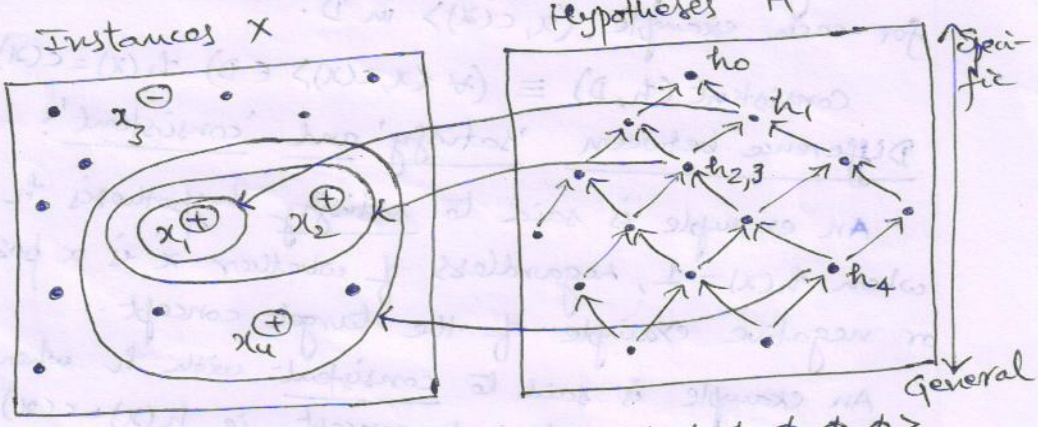
- The fourth (positive) example leads to a further generalization of h.

  $h \leftarrow$ <Sunny, Warm, ?, Strong, ?, ?>

Conclusion:

- The FIND-S algorithm illustrates how the more-general-than partial ordering can be used to organize the search for an acceptable hypothesis.
- The search moves from hypothesis to hypothesis, searching from the most specific to progressively more general hypotheses.

Consider the following figure:



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, ?, ?>

- At each step, the hypothesis is generalized only as far as necessary to cover the new positive example.
- Therefore, at each stage, the hypothesis is the "most specific hypothesis" consistent with the training examples observed up to this point (hence the name FIND-S).

Version Spaces and the Candidate-Elimination Algorithm:

Consistency - Definition:

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

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

Difference between 'satisfy' and 'consistent':

An example 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 is said to consistent with h when it depends on the target concept, ie, $h(x) = c(x)$.

Version Space:

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 the training examples in D.

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

The List-Then-Eliminate Algorithm:

This algorithm first initializes the version space to contain all hypotheses in H, then eliminates any hypothesis found inconsistent with any training example.

The version space of candidate hypotheses thus shrinks as more examples are observed. Finally, just one hypothesis remains that is consistent with all the observed examples.

- This algorithm can be applied whenever H is finite.
- Advantage is, it is guaranteed to output all hypotheses consistent with the training data.
- Disadvantage is, it exhaustively enumerates all hypotheses in H.

Algorithm:

1. Version space ← a list containing every hypothesis in H.
2. For each training example $\langle x, c(x) \rangle$, remove from version space any hypothesis h for which $h(x) \neq c(x)$.
3. Output the list of hypotheses in Version Space.

The Candidate-Elimination Algorithm:

- It is the second approach to concept learning. It addresses several of the limitations of FIND-S.
- 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.

It can do this without explicitly enumerating all of its members.
(using the more-general-than partial ordering)

2.1.15

- It employs a more compact representation of the version space.
- The version space is represented by its most general and least general (specific) members. These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

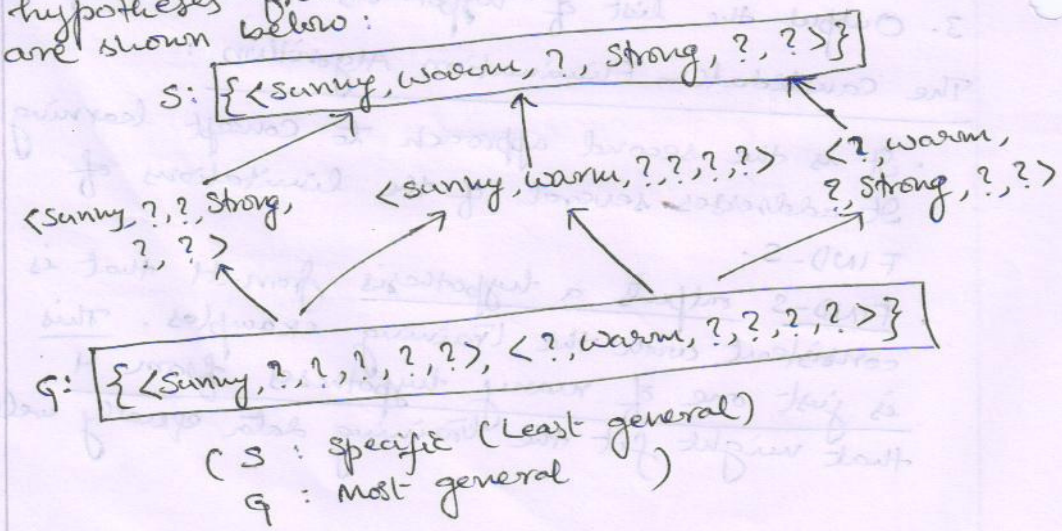
Representation of Version Space:

Ex: EnjoySport concept learning task

FIND-S outputs the hypothesis (earlier)

$h = \langle$sunny, warm, ?, strong, ?, ?$\rangle$

In fact, this is just one of six different hypotheses from H that are consistent. These are shown below:



(S : specific (Least general)
G : Most general)

- The candidate-Elimination algorithm represents the version space by storing only its most-general members (G) and its most-specific (S).
- Given S and G, it is possible to enumerate all members of the version space that lie between these two sets.

Definitions of G and S:

The general boundary G, w.r.t. the hypothesis space H and training data D, is the set of maximally general members of H consistent with D.

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

The specific boundary S, w.r.t. hypothesis space H and training data D, is the set of minimally general (ie, maximally specific) members of H consistent with D.

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

The Algorithm:

1. Initialize G to the set of maximally general hypotheses in H.
2. Initialize S to the set of maximally specific hypotheses in H.

3. 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.

3.1.15

Working of Candidate-Elimination Algorithm:

Example: Data of Table 1.

1. The Algorithm initializes G and S as specified below:

$G_0 \leftarrow \{\langle ?, ?, ?, ?, ?, ? \rangle\}$ (most general hypothesis)

$S_0 \leftarrow \{\langle \phi, \phi, \phi, \phi, \phi, \phi \rangle\}$ (most specific hypothesis)

(The version space initially contains these two hypotheses. As the algorithm progresses, new hypotheses that are consistent with the training examples will be added and any hypotheses that are inconsistent will be removed from the version space.)

Now, the version space is:

S_0: $\{\langle \phi, \phi, \phi, \phi, \phi, \phi \rangle\}$

G_0: $\{\langle ?, ?, ?, ?, ?, ? \rangle\}$

2. Consider 1st training example (Positive).

Algorithm checks the S boundary and finds that it is overly specific — it fails to cover this.

So, S boundary is revised to 'least more general hypothesis' that covers example 1.

No update of G boundary is required.

S_1: $\{\langle Sunny, Warm, Normal, Strong, Warm, Same \rangle\}$

G_1: $\{\langle ?, ?, ?, ?, ?, ? \rangle\}$

3. Consider 2nd training example (Positive).

S_1 requires generalization to cover examples 1 and 2. G_1 requires no further update.

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

G_2 : {<?, ?, ?, ?, ?, ?>}

4. Consider 3rd training example (Negative).

Negative training examples play the complimentary role of forcing the G boundary to become increasingly specific.

The hypothesis in G_2 incorrectly classifies the 3rd example as positive. Therefore it must be specialized until it correctly classifies this new negative example.

G can include three alternatives (minimally more specific hypotheses) as shown below:

S_2, S_3 : {<Sunny, warm, ?, Strong, warm, same>}

G_3 : {<Sunny, ?, ?, ?, ?, ?> <?, warm, ?, ?, ?, ?> <?, ?, ?, ?, ?, same>}

G_2 : {<?, ?, ?, ?, ?, ?>}

In G_3, all the three hypotheses classify training examples 1 and 2 as positive and example 3 as negative.

In fact, G_2 can be specialized in Six different ways, corresponding to Six attributes. They are:

1. <Sunny, ?, ?, ?, ?, ?>
2. <?, warm, ?, ?, ?, ?>
3. <?, ?, Normal, ?, ?, ?>
4. <?, ?, ?, Strong, ?, ?>
5. <?, ?, ?, ?, warm, ?>
6. <?, ?, ?, ?, ?, Same>

Among these, only 1, 2 and 6 are specified in G_3. Others are not included. The reason is:

1, 2, 6 — can classify examples 1, and 2 as positive and 3 as negative.

3 — classifies 1 as positive (✓), 2 as negative (x) and 3 as negative (✓).

4 — classifies 1 as positive (✓), 2 as positive (✓) and 3 also as positive (x).

5 — classifies 1 as positive (✓), 2 as positive (✓) and 3 also as positive (x).

5. Consider 4th training example (Positive).

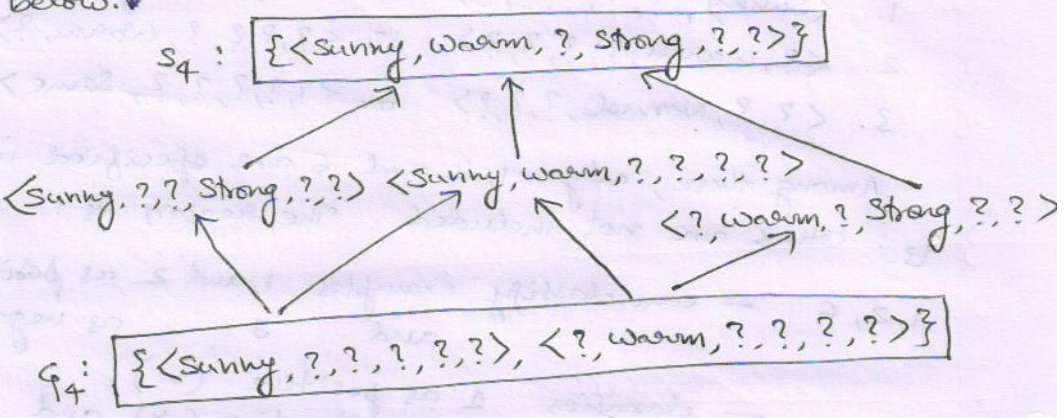
The 4th example further generalizes the S boundary and also results in removing one member of G boundary (<?, ?, ?, ?, ?, Same>), because this member fails to cover the new positive example.

Now the version space is:

S_4 : {<Sunny, warm, ?, Strong, ?, ?>}

G_4 : {<Sunny, ?, ?, ?, ?, ?> <?, warm, ?, ?, ?, ?>}

The entire version space, including those hypotheses bounded by S_4 and G_4, is shown below.



- 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.

Remarks on Version Spaces and Candidate-Elimination Algorithm :

1. Will the 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, and

(2) there is some hypothesis in H that correctly describes the target concept.

Explanation:

For (1): If the training data contains errors.

Suppose 2nd training example is incorrectly presented as a negative example instead of a positive example.

Then all the six hypotheses will be removed from the version space because each of those is inconsistent with this 2nd example. 2nd example is satisfied by each hypothesis but incorrectly classified as positive.

Hence the S and G boundary sets eventually converge to an empty version space.

For (2): Suppose the training examples are correct but the target concept cannot be described in the hypothesis representation — then also the algorithm will not converge.

(Ex: if the target concept is a disjunction of feature attributes and the hypothesis space supports only conjunctive descriptions.)

Hence we can say that the algorithm will converge when the given (1) and (2) are true.

2. What training example should the learner request next?

Consider the instance:

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

This satisfies three of the six hypotheses in the current version space (see Fig. in Pg. 30).

If the trainer classifies this instance as a positive example, the S boundary of the version space can then be generalized.

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.

3. How can partially learned concepts be used?

Suppose the first four training examples are available to the learner.

Now it is required to classify new instances that it has not yet observed.

Consider: New instances to be classified

Table 2

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	?

A – classified as positive by all the six hypotheses
(hence EnjoySport = yes (learned))

B – classified as negative by all the six hypotheses
(hence EnjoySport = no (learned))

C – classified as positive by three hypotheses,
and " negative " " .
(the learner cannot classify this example with confidence until further training examples are available.)

D – classified as positive by two hypotheses,
and " negative " four " .

(Since negative classification is high, the learner may classify this as negative.)

Inductive Bias

With the help of the two conditions specified above, the candidate-Elimination algorithm will converge toward the true target concept.

(Target concept : The days on which Aldo enjoys water sport.)

Consider the fundamental questions arise for inductive inference in general (given below):

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 the 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 are examined here, in the context of the 'candidate-Elimination Algorithm'.

The conclusions can be applied to any concept learning system.

1. A Biased Hypothesis Space :

Suppose the hypothesis space contains the unknown target concept.

The obvious solution is to enrich the hypothesis space to include every possible hypothesis.

In our 'EnjoySport' example, the hypothesis space is restricted to include only conjunctions of attribute values.

Because of this, the hypothesis space is unable to represent even simple disjunctive target concepts such as:

"Sky = Sunny or Sky = Cloudy".

Consider the following training examples:

Table 3

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

Our algorithm returns zero hypotheses for the above disjunctive target concept.

Consider the hypothesis:

S_2 : <?, Warm, Normal, Strong, Cool, Change>

This is the maximally specific hypothesis from H, but —

- it is consistent with the first two examples only.
- it incorrectly covers the third (negative) example.

The problem is that we have biased the learner to consider only conjunctive hypotheses. (We require a more expressive hypothesis space.)

2. An Unbiased Learner :

The solution to the above problem is :
provide a hypothesis space capable of representing every teachable concept.
(ie, representing every possible subset of the instances X.)

Power Set — the set of all subsets of a set X.

In our example,
the size of the instance space $X = 96$. $(3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2)$
The size of the Power set of $X = 2^{|X|}$
$= 2^{96}$ (or approx. 10^{28} distinct target concepts)

Ex: If $Z = \{a, b, c\}$, the Power set of Z consists of 2^3 elements. [$\{\phi\}, \{a,b,c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a\}, \{b\}, \{c\}$]

In our example of X,
there are $5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5,120$ syntactically distinct hypotheses within H.
ie, if we include ϕ and ? values for each attribute. (ex. for Sky: Sunny, Cloudy, Rainy, ϕ, ?
AirTemp : warm, cold, ϕ, ?
and so on.)

But every hypothesis containing one or more "ϕ" symbols represents the empty set of instances;
ie, it classifies every instance as negative.

∴ The no. of Semantically distinct hypotheses
$= 1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973$.

(ie, for Sky: <Sunny, –, –, –, –, –> ✓
<Cloudy, –, –, –, –, –> ✓
<Rainy, –, –, –, –, –> ✓ } 4 only
<?, –, –, –, –, –> ✓
<ϕ, –, –, –, –, –> Remove

Similarly for other attributes.

So, $4 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 = 972$

Instances containing one "ϕ" value or more than one "ϕ" value are collectively taken as 1.

Hence total $= 1 + 972 = 973$)

So, even though X can represent 2^{96} instances, the hypothesis space is able to represent only 973 of these (a very biased hypothesis space indeed).

Let us reformulate the EnjoySport learning task.

Suppose H' is — new hypothesis space
correspond to power set of X

Allow arbitrary conjunctions, disjunctions and negations of our earlier hypotheses in H'.

Ex: The target concept "Sky = Sunny or Sky = Cloudy" could then be represented as:

$\langle Sunny, ?, ?, ?, ?, ? \rangle \vee \langle Cloudy, ?, ?, ?, ?, ? \rangle$

Now we can safely use Candidate-Elimination algorithm.

H' eliminates any problems of expressibility, but raises another problem:

Our learning algorithm is now completely unable to generalize beyond the observed examples.

Suppose x_1, x_2, x_3 — are positive examples
x_4, x_5 — " negative " .

Then S and G will be:

$S: \{ (x_1 \vee x_2 \vee x_3) \}$

$G: \{ \neg(x_4 \vee x_5) \}$

This is a very expressive hypothesis representation. Therefore, the only examples that will be unambiguously classified by S and G are the observed training examples themselves.

In order to converge to a single, final target concept, we will have to present every single instance in X as a training example.

5.1.15

3. The Futility of Bias-Free Learning

Inductive learning requires some form of prior assumptions (or inductive bias).

One such assumption is that the target concept could be represented by a conjunction of attribute values.

In cases where this assumption is correct (and the training examples are error-free), its classification of new instances will also be correct. (If this assumption is incorrect, it is certain that the Candidate-Elimination algorithm will misclassify at least some instances from X.)

Defining the notion of Inductive Bias:

Suppose: L – a learning algorithm

$D_c = \{\langle x, c(x)\rangle\}$ – set of training data of some arbitrary concept c.

x_i – new instance to be classified by L

$L(x_i, D_c)$ — denote the classification (ex: positive or negative) that L assigns to x_i after learning from D_c.

$y \succ z$ — indicates that z is inductively inferred from y.

Then, the inductive inference step performed by L is:

$$(D_c \wedge x_i) \succ L(x_i, D_c)$$

For our EnjoySport concept learning Task,

L — is the Candidate-Elimination algorithm

D_c — training data from Table 1

x_i — <Sunny, Warm, Normal, Strong, Cool, Change>

Then the inductive inference concludes that

$$L(x_i, D_c) = (\text{EnjoySport} = \text{yes})$$

Induction: Any form of reasoning in which the conclusion, though supported by the premises, does not follow from them necessarily.

Deduction: A process of reasoning in which a conclusion follows necessarily from the premises presented, so that the conclusion cannot be false if the premises are true.

Because L is an inductive learning algorithm, the classification $L(x_i, D_c)$ need not follow deductively from the training data D_c and the new instance x_i.

- If we add additional assumptions to $D_c \wedge x_i$, then $L(x_i, D_c)$ would follow deductively.
- We define the inductive bias of L as this set of additional assumptions.

More precisely, we define the inductive bias of L to be the set of assumptions B such that for all new instances x_i,

$$(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)$$

where the notation $y \vdash z$ indicates that z follows deductively from y (ie, z is provable from y).

Thus, we define the inductive bias of a learner as the set of additional assumptions B sufficient to justify its inductive inferences as deductive inferences.

Defn. – Inductive Bias:

The inductive bias of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c

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

Inductive Bias of Candidate-Elimination algorithm :

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

- Inductive learning algorithms are able to classify unseen examples only because of their implicit inductive bias for selecting one consistent hypothesis over another. The bias associated with the Candidate-Elimination algorithm is that the target concept can be found in the provided hypothesis space ($c \in H$). The output hypotheses and classifications of subsequent instances follow deductively from this assumption together with the observed training data.

— * * * —

K. Srinivasa Rao
Assoc. Prof.
Dept. of CSE, GIT, GU.