An introduction to Machine Learning

- The term Machine Learning was coined by Arthur Samuel in 1959, an American pioneer in the field of computer gaming and artificial intelligence, and stated that “it gives computers the ability to learn without being explicitly programmed”.
- In 1997, Tom Mitchell gave a “well-posed” mathematical and relational definition that

“A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

- Within the field of data analytics, machine learning is used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics.
- These analytical models allow researchers, data scientists, engineers, and analysts to “produce reliable, repeatable decisions and results” and uncover “hidden insights” through learning from historical relationships and trends in the data set(input).

Applications of Machine Learning

Learning to recognize spoken words.

Most successful speech recognition systems employ machine learning in some form.

For example, the SPHINX system (e.g., Lee 1989) learns speaker-specific strategies for recognizing the primitive sounds (phonemes) and words from the observed speech signal.

Neural network learning methods (e.g., Waibel et al. 1989) and methods for learning hidden Markov models (e.g., Lee 1989) are effective for automatically customizing to, individual speakers, vocabularies, microphone characteristics, background noise, etc.

Learning to drive an autonomous vehicle.

Machine learning methods have been used to train computer-controlled vehicles to steer correctly when driving on a variety of road types.

For example, the ALVINN system (Pomerleau 1989) has used its learned strategies to drive unassisted at 70 miles per hour for 90 miles on public highways among other cars.

Recommendation Engine.

You browse through the travel agency website and search for a hotel. When you look at a specific hotel, just below the hotel description there is a section titled “You might also like these hotels”.

Contd…

Learning to classify new astronomical structures.

Machine learning methods have been applied to a variety of large databases to learn general regularities implicit in the data.

For example, decision tree learning algorithms have been used by NASA to learn how to classify celestial objects from the second Palomar Observatory Sky Survey (Fayyad et al. 1995).

This system is now used to automatically classify all objects in the Sky Survey, which consists of three terabytes of image data.

Learning to play world-class backgammon.

The most successful computer programs for playing games such as backgammon are based on machine learning algorithms.

For example, the world's top computer program for backgammon, TD-GAMMON (Tesauro 1992, 1995). learned its strategy by playing over one million practice games against itself.

It now plays at a level competitive with the human world champion.

WELL-POSED LEARNING PROBLEMS

“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”.

For example, a computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself.

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

the class of tasks

the measure of performance to be improved

the source of experience.

Examples of well-posed problems

A checkers learning problem:

Task T: playing checkers

Performance measure P: percent of games won against opponents

Training experience E: playing practice games against itself

A handwriting recognition learning problem:

Task T: recognizing and classifying handwritten words within images

Performance measure P: percent of words correctly classified

Training experience E: a database of handwritten words with given classifications

A robot driving learning problem:

Task T: driving on public four-lane highways using vision sensors

Performance measure P: average distance travelled before an error (as judged by human overseer)

Training experience E: a sequence of images and steering commands recorded while observing a human driver

Disciplines influence on Machine Learning

Artificial intelligence

Learning symbolic representations of concepts.

Machine learning as a search problem.

Learning as an approach to improving problem solving.

Using prior knowledge together with training data to guide learning.

Bayesian methods

Bayes' theorem as the basis for calculating probabilities of hypotheses.

The naive Bayes classifier.

Algorithms for estimating values of unobserved variables.

Computational complexity theory

Theoretical bounds on the inherent complexity of different learning tasks, measured in terms of the computational effort, number of training examples, number of mistakes, etc. required in order to learn.

Control theory

Procedures that learn to control processes in order to optimize predefined objectives and that learn to predict the next state of the process they are controlling.

Contd…

Information theory

Measures of entropy and information content.

Minimum description length approaches to learning.

Optimal codes and their relationship to optimal training sequences for encoding a hypothesis.

Philosophy

Occam's razor, suggesting that the simplest hypothesis is the best.

Analysis of the justification for generalizing beyond observed data.

Psychology and neurobiology

The power law of practice, which states that over a very broad range of learning problems, people's response time improves with practice according to a power law.

Neurobiological studies motivating artificial neural network models of learning.

Statistics

Characterization of errors (e.g., bias and variance) that occur when estimating the accuracy of a hypothesis based on a limited sample of data. Confidence intervals, statistical tests.

DESIGNING A LEARNING SYSTEM

- **Choosing the Training Experience**
- **Choosing the Target Function**
- **Choosing a Representation for the Target Function**
- **Choosing a Function Approximation Algorithm for Target Function**

Choosing the Training Experience

a) The **first design choice** is to choose the **type of training experience** from which our system will learn.

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 regarding the choices made by the performance system.

Direct : In learning to play checkers, the system might learn from direct training examples consisting of individual checkers board states and the correct move for each.

Indirect: Available only indirect information consisting of the move sequences and final outcomes of various games played.

credit assignment, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome is difficult.

Contd…

- b) A **second** important attribute of the training experience is the **degree to which the learner controls** the sequence of training examples.
- For example, 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 that it finds particularly **confusing and ask the teacher** for the correct move.
- The learner may have **complete control** over both the board states and training classifications, as it does when it learns by playing against itself with no teacher present.

Contd…

- c) A **third** important attribute of the training experience is **how well it represents the distribution** of examples over which the final system performance P must be measured.
- Learning is most reliable when the training examples follow a distribution similar to that of future test examples.
- In our checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.
- If its training experience E consists only of games played against itself, there is an obvious danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.
- For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion.

Contd…

- To proceed with our design of Checkers problem, let us consider that our system will train by playing games against itself.
- This has the advantage that no external trainer need to be present, and it therefore allows the system to generate as much training data as time permits.
- Now define a fully specified learning task as given below for **A checkers learning problem**
- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: games played against itself
- In order to complete the design of the learning system, we must now choose
    1. the exact type of knowledge to be, learned i.e. target function
    2. a representation for this target knowledge
    3. a learning mechanism or algorithm

Choosing the Target Function

- The second design choice is to determine exactly **what type of knowledge** will be learned and how this will be used by the performance program
- Let us begin with a checkers-playing program that can generate the legal moves from any board state.
- The program needs only to learn how to choose the best move from among these legal moves.
- This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known.
- Many optimization problems fall into this class, such as the problems of scheduling and controlling manufacturing processes where the available manufacturing steps are well understood, but the best strategy for sequencing them is not.

Contd…

- Learn a program, or function, that chooses the best move for any given board state.
- Let us call this function ChooseMove and use the notation

  ChooseMove : B -> M

- This function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.
- The choice of the target function is key for a design choice.

Contd…

- A target function that is easier to learn is an evaluation function that assigns a numerical score to any given board state.
- Let us call this target function V and use the notation V : B ->R to denote that V maps any legal board state from the set B to some real value R (we use R to denote the set of real numbers).
- The target function V will assign higher scores to better board states.
- 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 successor board state produced by every legal move, then using V to choose the best successor state and therefore the best legal move

Contd…

Let us define the target value V(b) for an arbitrary board state b in B, as follows:

- 1. if b is a final board state that is won, then V(b) = 100
- 2. if b is a final board state that is lost, then V(b) = -100
- 3. if b is a final board state that is drawn, then V(b) = 0
- 4. if b is a not a final state in the game, then V(b) = V(b'), 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).

Choosing a Representation for the Target Function

- For any given board state, the function V^ will be calculated as a linear combination of the following board features:
- x1: the number of black pieces on the board
- x2: the number of red pieces on the board
- x3: the number of black kings on the board
- x4: the number of red kings on the board
- x5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- X6: the number of red pieces threatened by black
- our learning program will represent V^(b) as a linear function of the form

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

- where w0 through w6 are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights w1 through w6 will determine the relative importance of the various board features in determining the value of the board, whereas the weight w0 will provide an additive constant to the board value.

- The elaborated formulation of the learning problem **by choosing a type of training experience, a target function to be learned, and a representation for this target function** for a checkers learning program as given below:
- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: games played against itself
- Target function: V:Board -> R
- Target function representation

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

- The first three items above correspond to the specification of the learning task, whereas the final two items constitute design choices for the implementation of the learning program.
- The net effect of this set of design choices is to reduce the problem of learning a checkers strategy to the problem of learning values for the coefficients w0 through w6 in the target function representation.

Choosing a Function Approximation Algorithm

- In order to learn the target function f we require a set of training examples, each describing a specific board state b and the training value Vtrain(b) for b.
- In other words, each training example is an ordered pair of the form (b, Vtrain(b)).
- For instance, the following training example describes a board state b in which black has won the game

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

- (note x2 = 0 indicates that red has no remaining pieces) and for which the target function value Vtrain(b) is therefore +100.

ESTIMATING TRAINING VALUES

- one simple approach is to assign the training value of Vtrain(b) for any intermediate board state b to be V(Successor(b)), where b is the learner's current approximation to V and where Successor(b) denotes the next board state following b for which it is again the program's turn to move
- Rule for estimating training values $V_{train}(b) \leftarrow \hat{V}(Successor(b))$
- We are using estimates of the value of the Successor(b) to estimate the value of board state b.
- Intuitively, we can see this will make sense if it tends to be more accurate for board states closer to game's end.

$\hat{V}$

ADJUSTING THE WEIGHTS

- "Specify the learning algorithm for choosing the weights wi to best fit the set of training examples {(b, Vtrain(b))}".
- First step is to define "what is meant by the best fit to the training data?".
- One common approach is to define the best hypothesis, or set of weights, is that which minimizes the square error E between the training values and the values predicted by the hypothesis V.

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

- In checkers case, it requires an algorithm that will incrementally refine the weights as new training examples become available and that will be robust to errors.
- One such algorithm is called the **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 the training example.
- This algorithm can be viewed as performing a stochastic gradient-descent search through the space of possible hypotheses (weight values) to minimize the squared error E.

Contd…

- The LMS algorithm is defined as follows:

LMS weight update rule.

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

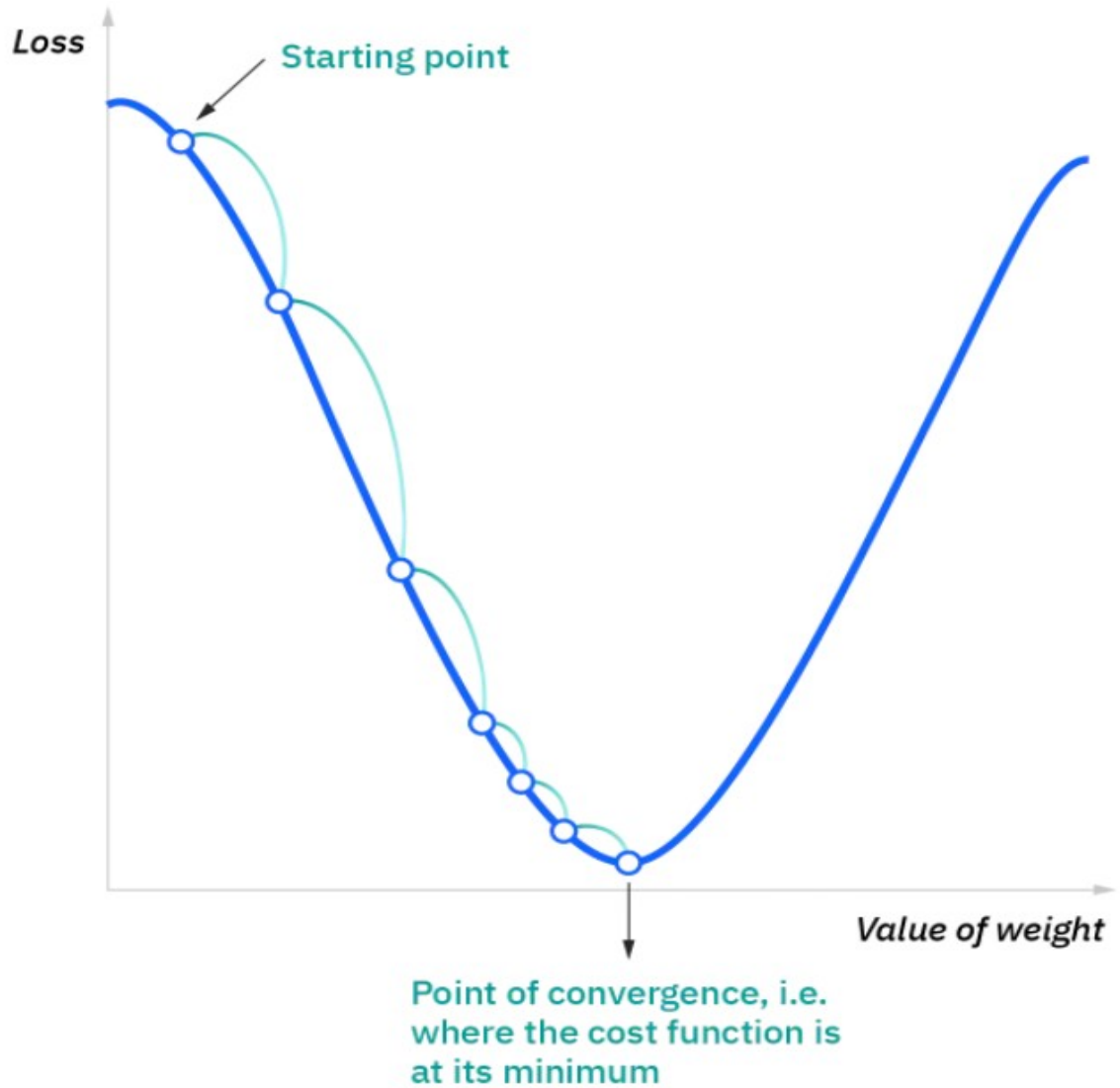
- Use the current weights to calculate $\hat{V}(b)$
- For each weight w_i, update it as

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

- Here ή is a small constant (e.g., 0.1) that moderates the size of the weight update.
- To get an intuitive understanding for why this weight update rule works, notice that when the error (Vtrain(b) - c(b)) is zero, no weights are changed.
- When (Vtrain(b) – c(b)) is positive (i.e., when c(b) is too low), then each weight is increased in proportion to the value of its corresponding feature.

Gradient Descent

- Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks.
- Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates.
- Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error.
- Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.


Loss
Starting point
Value of weight
Point of convergence, i.e.
where the cost function is
at its minimum

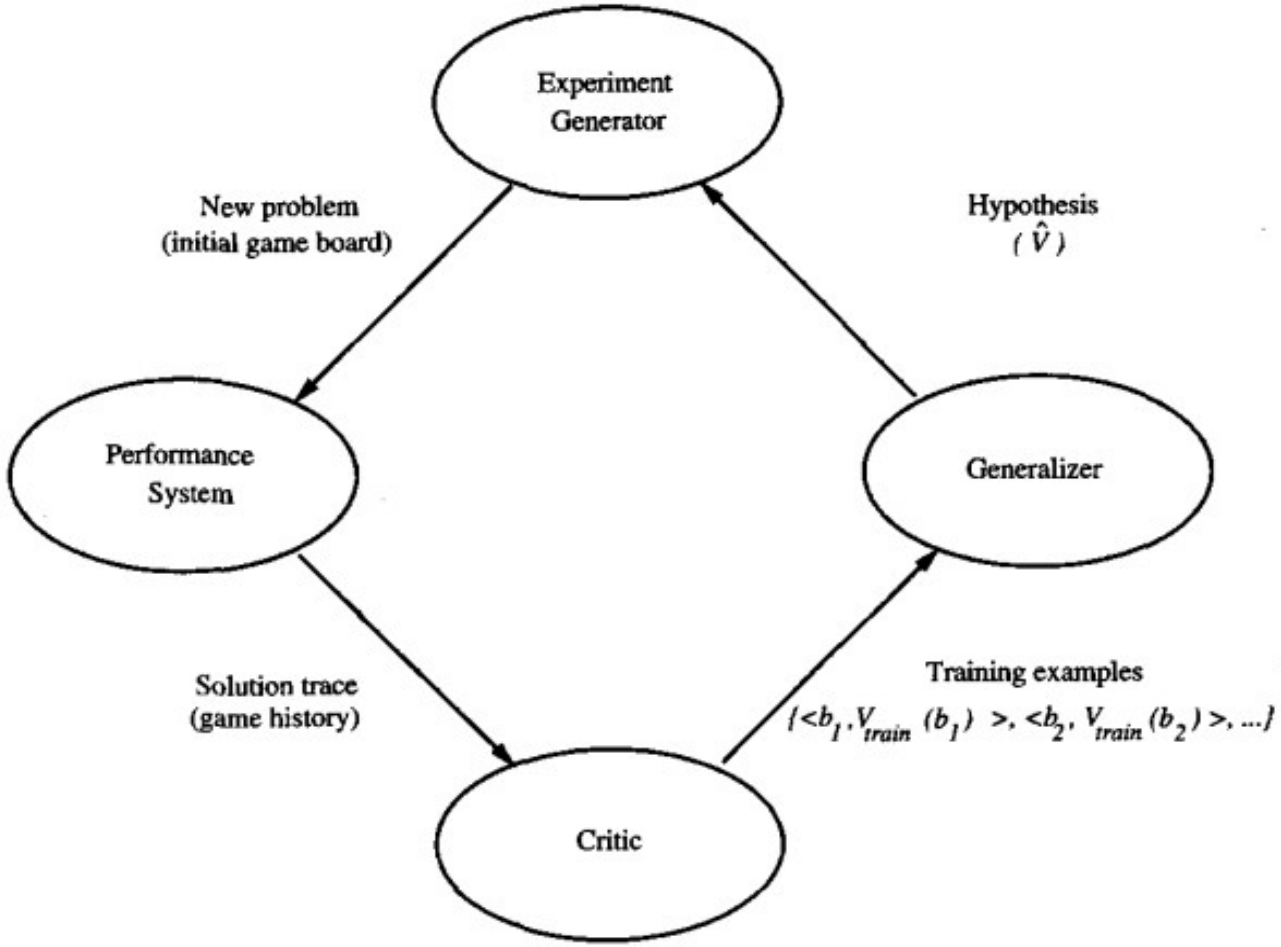
The Final Design

- The final design of checkers learning system can be described by four distinct program modules that represent the central components in many learning systems.
- **The Performance System** is the module that must solve the given performance task
- In case of 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.
- In this case, the strategy used by the Performance System to select its next move at each step is determined by the learned p evaluation function. Therefore, expect its performance to improve as this evaluation function becomes increasingly accurate.
- **The Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function.
- As shown in the diagram, each training example in this case corresponds to some game state in the trace, along with an estimate Vtrain, of the target function value for this example.

Contd…

- **The Generalizer** takes as input the training examples and produces an output hypothesis that is its 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 our example, the Generalizer corresponds to the LMS algorithm, and the output hypothesis is the function f described by the learned weights wo, . . . , w6.
- **The Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem (i.e., 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, the Experiment Generator follows a very simple strategy: It always proposes the same initial game board to begin a new game. More sophisticated strategies could involve creating board positions designed to explore particular regions of the state space.

System Design



PERSPECTIVES AND ISSUES IN MACHINE LEARNING

- One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner.
- For example, consider the space of hypotheses that could in principle be output by the checkers learner.
- This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights w0 through w6.
- The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples.
- The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value.
- This algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.

Issues in Machine Learning

- Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient?
- Can prior knowledge be helpful even when it is only approximately correct?
- How does the choice of strategy alter the complexity of the learning problem?
- What specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

CONCEPT LEARNING

- Most of the learning involves acquiring general concepts from specific training examples.
- People, for example, 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 (e.g., the subset of animals that constitute birds).
- Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set (e.g., a function defined over all animals, whose value is true for birds and false for other animals).
- **Concept learning** is 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 favourite 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

Contd...

- The attribute EnjoySport indicates whether or not Aldo enjoys his 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.
- **hypothesis representation**
- Let us begin by considering 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
- indicate by a "0" that no value is acceptable.

Contd…

- If some instance x satisfies all the constraints of hypothesis h, then h classifies x as a positive example (h(x) = 1).
- To illustrate, the hypothesis that Aldo enjoys his favourite sport only on cold days with high humidity (independent of the values of the other attributes) 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

  (0,0,0,0,0,0)
- The EnjoySport concept learning task requires learning the set of days for which EnjoySport = yes, describing this set by a conjunction of constraints over the instance attributes.
- In general, any concept learning task can be described by the set of instances over which the target function is defined, the target function, the set of candidate hypotheses considered by the learner, and the set of available training examples

Notation

- The set of items over which the concept is defined is called the set of instances, which we denote by X.
- In the current 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 we denote by c.
- In general, c can be any Boolean valued function defined over the instances X

  c : X -> {O, 1).
- In the current 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).
- Given a set of training examples of the target concept c, the problem faced by the learner is to hypothesize, or estimate, c.
- We use the symbol H to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept.
- In general, each hypothesis h in H represents a Boolean-valued function defined over X; that is, h : X -> {O, 1). The goal of the learner is to find a hypothesis h such that h(x) = c(x) for all x in X.

The EnjoySport concept learning task

- **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), "$\emptyset$" (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 2.1).
- **Determine:**
  - A hypothesis h in H such that $h(x) = c(x)$ for all x in X.

The Inductive Learning Hypothesis

- Although the learning task is to determine a hypothesis h identical to the target concept c over the entire set of instances X, the only information available about c is its value over the training examples.
- Therefore, 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.
- Consider, for example, the instances X and hypotheses H in the EnjoySport learning task. Given that the attribute Sky has three possible values, and that AirTemp, Humidity, Wind, Water, and Forecast each have two possible values, the instance space X contains exactly 3 .2 2 .2 2 .2 = 96 distinct instances.
- A similar calculation shows that there are 5.4.4.4.4.4 = 5120 syntactically distinct hypotheses within H.
- However, that every hypothesis containing one or more “0" symbols represents the empty set of instances; that is, it

- classifies every instance as negative.
- Therefore, the number of semantically distinct hypotheses is only 1 + (4.3.3.3.3.3) = 973.
- Our EnjoySport example is a very simple learning task, with a relatively small, finite hypothesis space. Most practical learning tasks involve much larger, sometimes infinite, hypothesis spaces.

General-to-Specific Ordering of Hypotheses

- Many algorithms for concept learning organize the search through a very useful structure known as: 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.
- To illustrate the general-to-specific ordering, consider the two hypotheses
- h1 = (Sunny, ?, ?, Strong, ?, ?)
- h2 = (Sunny, ?, ?, ?, ?, ?)
- Now consider the sets of instances that are classified positive by h1 and by h2. Because h2 imposes fewer constraints on the instance, it classifies more instances as positive. In fact, any instance classified positive by h1 will also be classified positive by h2.
- Therefore, we say that h2 is more general than h1.

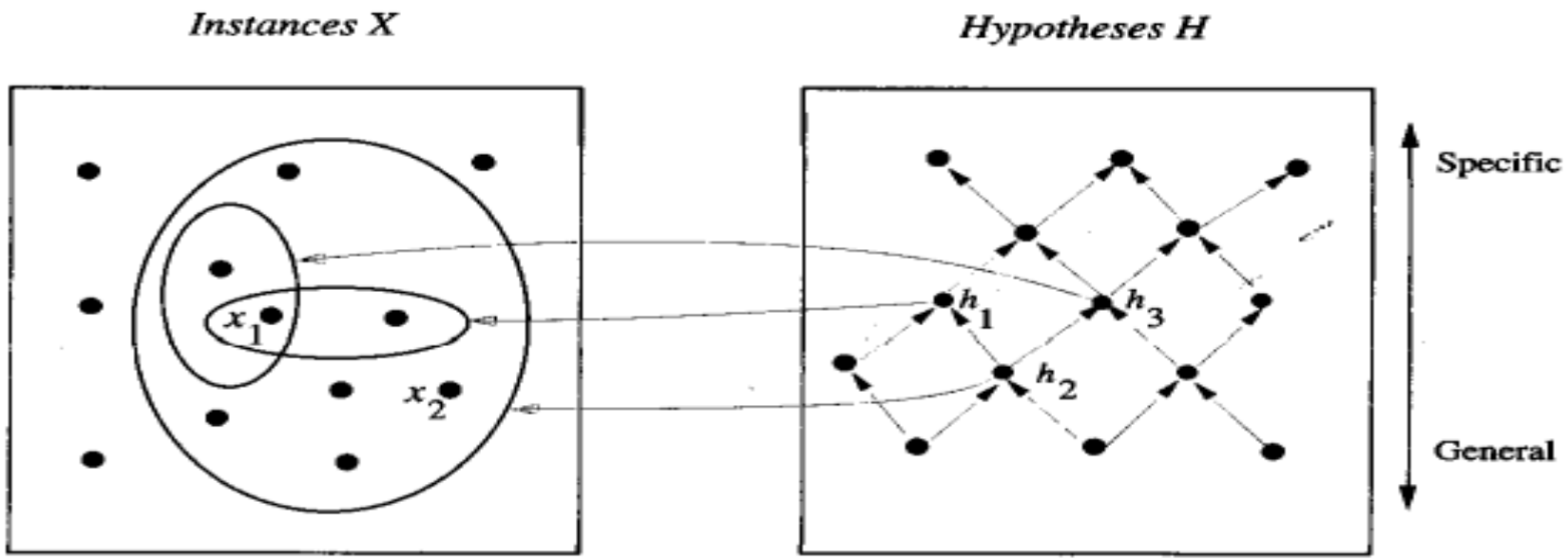
Contd…

- First, for any instance x in X and hypothesis h in H, we say that x satisfies h if and only if h(x) = 1.
- We now define the more-general_than_or_equal_to relation in terms of the sets of instances that satisfy the two hypotheses:
- 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.

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

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

Contd…

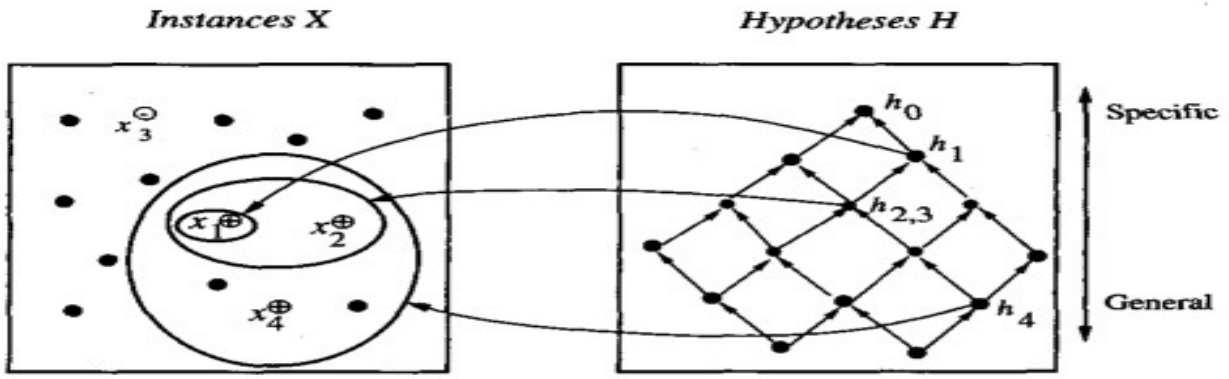


x_1= <Sunny, Warm, High, Strong, Cool, Same>
x_2= <Sunny, Warm, High, Light, Warm, Same>

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

FIND-S: 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 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



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 ? ? >

Steps Involved In Find-S

1. Start with the most specific hypothesis.
   h = {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}
2. Take the next example and if it is negative, then no changes occur to the hypothesis.
3. If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.
4. Keep repeating the above steps till all the training examples are complete.
5. After we have completed all the training examples we will have the final hypothesis when can use to classify the new examples.

Data set having the data about which particular seeds are poisonous.

EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

ATTRIBUTES ON WHICH THE CONCEPT DEPENDS ON — CONCEPT

First, we consider the hypothesis to be a more specific hypothesis. Hence, our hypothesis would be :

h = {ϕ, ϕ, ϕ, ϕ,}

Consider example 1 :

The data in example 1 is { GREEN, HARD, NO, WRINKLED }. We see that our initial hypothesis is more specific and we have to generalize it for this example. Hence, the hypothesis becomes :

h = { GREEN, HARD, NO, WRINKLED }

Consider example 2 :

Here we see that this example has a negative outcome. Hence we neglect this example and our hypothesis remains the same.

h = { GREEN, HARD, NO, WRINKLED }

Contd...

- **Consider example 3 :**
  Here we see that this example has a negative outcome. Hence we neglect this example and our hypothesis remains the same.
  h = { GREEN, HARD, NO, WRINKLED }
- **Consider example 4 :**
  The data present in example 4 is { ORANGE, HARD, NO, WRINKLED }. We compare every single attribute with the initial data and if any mismatch is found we replace that particular attribute with a general case (” ? ”). After doing the process the hypothesis becomes :
  h = { ?, HARD, NO, WRINKLED }
- **Consider example 5 :**
  The data present in example 5 is { GREEN, SOFT, YES, SMOOTH }. We compare every single attribute with the initial data and if any mismatch is found we replace that particular attribute with a general case (” ? ”). After doing the process the hypothesis becomes :
  h = { ?, ?, ?, ? }
  Since we have reached a point where all the attributes in our hypothesis have the general condition, example 6 and example 7 would result in the same hypothesizes with all general attributes.
  h = { ?, ?, ?, ? }
- Hence, for the given data the final hypothesis would be :
  Final Hyposthesis: h = { ?, ?, ?, ? }

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

- The CANDIDATE-ELIMINATION algorithm, addresses several limitations of FIND-S.
- Although 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.
- The 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, to maintain a compact representation of the set of consistent hypotheses and to incrementally refine this representation as each new training example is encountered.

Representation

- Let us say that a hypothesis is consistent with the training examples if it correctly classifies these examples.

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.

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

- The key difference between this definition of consistent and satisfies is
- 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).
- 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 possible versions of the target concept.

For Example:

Example	Citations	Size	InLibrary	Price	Editions	Buy
1	Some	Small	No	Affordable	One	No
2	Many	Big	No	Expensive	Many	Yes

h1 = (?, ?, No, ?, Many) – Consistent Hypothesis as it is consistent with all the training examples

h2 = (?, ?, No, ?, ?) – Inconsistent Hypothesis as it is inconsistent with first training example

The LIST-THEN-ELIMINATE Algorithm

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

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

- One way to represent the version space is simply to list all of its members. This leads to a simple learning algorithm known as List-Then-Eliminate.
- The LIST-THEN-ELIMINATE 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, until ideally just one hypothesis remains that is consistent with all the observed examples.
- This, presumably, 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.

Contd…

- The LIST-THEN-ELIMINATE algorithm can be applied whenever the hypothesis space H is finite.
- It is guaranteed to output all hypotheses consistent with the training data

---

The 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 $VersionSpace$

A More Compact Representation for Version Spaces

- The CANDIDATE-ELIMINATION algorithm works on the same principle as the above LIST-THEN-ELIMINATE algorithm.
- However, it employs a much more compact representation of the version space.
- In particular, 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.
- The CANDIDATE-ELIMINATION algorithm represents the version space by storing only its most general members (labeled G) and its most specific (labeled S).
- Given only these two sets S and G, it is possible to enumerate all members of the version space as needed by generating the hypotheses that lie between these two sets in the general-to-specific partial ordering over hypotheses.

Contd…

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.

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

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.

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

- **Version space representation 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

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

Contd…

- 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
- Go -> {(?, ?, ?, ?, ?, ?)}
- and initializing the S boundary set to contain the most specific (least general) hypothesis
- So -> ((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 So and more specific than Go
- 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

Contd…

Consider the dataset given below:

Sky	Temperature	Humid	Wind	Water	Forest	Output
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

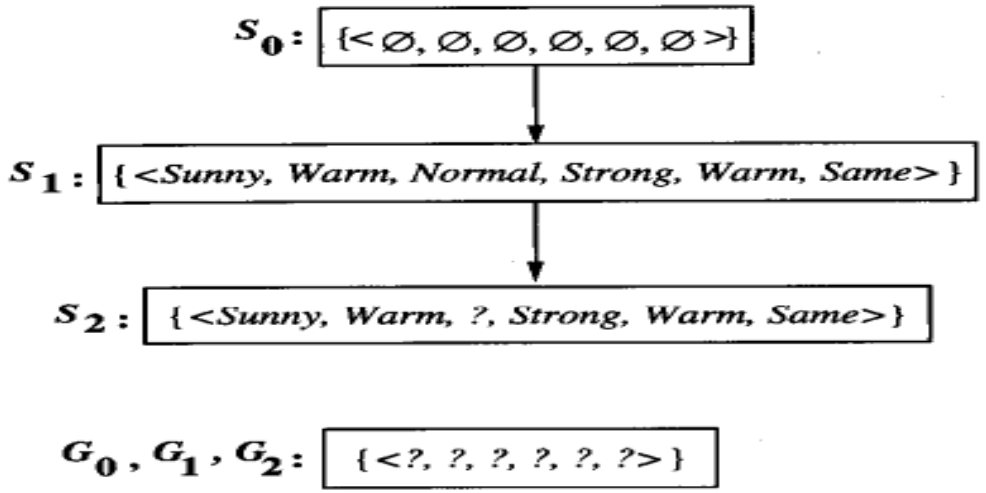
G0= (?,?,?,?,?,?)
S0= (0,0,0,0,0,0)
For –ve example move specific to general
For +ve examples move general to specific
Consider Example 1: Yes= +ve so move general to specific
G1= (sunny,wam,normal,strong, warm, same)
S1= (0,0,0,0,0,0)
Consider Example 2: Yes= +ve so move general to specific
G2= (sunny,wam, ?, strong, warm, same)
S2= (0,0,0,0,0,0)
Consider Example 3: No= -ve so move specific to general
G3= (sunny,wam, ?, strong, warm, same)
S3=(Sunny, ?,?,?,?,?) (?,Warm, ?,?,?,?) (?,?,?,?,?,Same)
Consider Example 4: Yes= +ve so move general to specific
G4=(sunny,wam, ?, strong,?,?)
S4=(Sunny, ?,?,?,?,?) (?,Warm, ?,?,?,?)

Example	Citations	Size	InLibrary	Price	Editions	Buy
1	Some	Small	No	Affordable	One	No
2	Many	Big	No	Expensive	Many	Yes
3	Many	Medium	No	Expensive	Few	Yes
4	Many	Small	No	Affordable	Many	Yes

G0= (?,?,?,?,?,?)
S0= (0,0,0,0,0,0)
For –ve example move specific to general
For +ve examples move general to specific
Consider Example 1: No= -ve so move specific to general
G1= (?,?,?,?,?,?)
S1=(Some, ?, ?,?,?) (?, Small, ?,?,?) (?,?,No,?,?) (?,?,?.Affordable, ?) (?,?,?,?,one)
Consider Example 2: Yes= +ve so move general to specific
G2=(Many, Big, No, Expensive, Many)
S2==(Some, ?, ?,?,?) (?, Small, ?,?,?) (?,?,No,?,?) (?,?,?.Affordable, ?) (?,?,?,?,one)
Consider Example 3: Yes= +ve so move general to specific
G3= (Many, ?,No, Expensive, ?)
S3=(Some, ?, ?,?,?) (?, Small, ?,?,?) (?,?,No,?,?) (?,?,?.Affordable, ?) (?,?,?,?,one)

Consider Example 4: Yes= +ve so move general to specific
G4= (Many, ?,No,?, ?)
S4=(?,?,No,?,?)

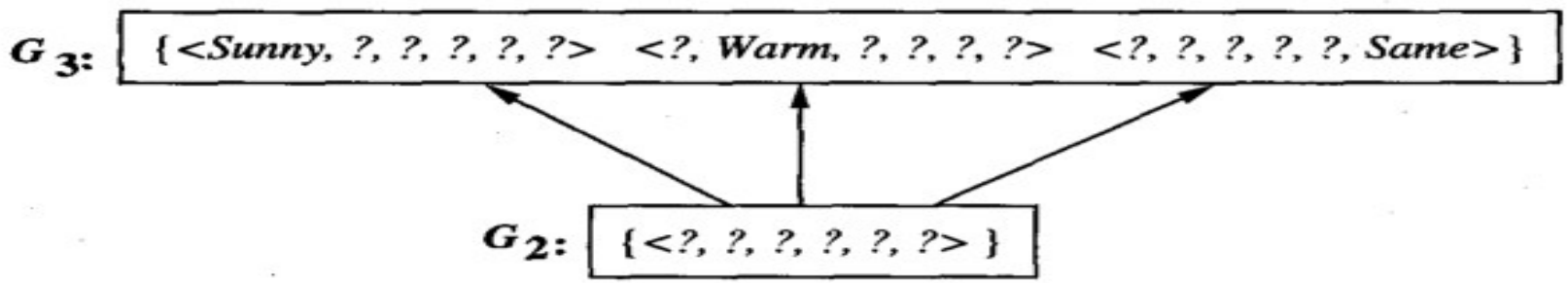
An Illustrative Example



Training examples:

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

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



Training Example:

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

INDUCTIVE BIAS

- As discussed above, 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.

A Biased Hypothesis Space

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

To see why there are no hypotheses consistent with these three examples, note that the most specific hypothesis consistent with the first two examples *and representable in the given hypothesis space H* is

$$S_2 : \langle ?, Warm, Normal, Strong, Cool, Change \rangle$$

- This hypothesis, although it is the maximally specific hypothesis from H that is consistent with the first two examples, is already overly general: it incorrectly covers the third (negative) training example.
- The problem is that we have biased the learner to consider only conjunctive hypotheses.

An Unbiased Learner