Unit-I

Course Objectives

- This course explains machine learning techniques such as decision tree learning, Bayesian learning etc.
- To understand computational learning theory.
- To study the pattern comparison techniques.

Course Outcomes

- Understand the concepts of computational intelligence like machine learning
- Ability to get the skill to apply machine learning techniques to address the real time problems in different areas
- Understand the Neural Networks and its usage in machine learning application.

Unit-I

- Introduction
- Concept Learning and General to Specific Ordering
- Decision Tree Learning

Introduction

- Well-posed Learning Problems
- Designing a Learning System
- Perspectives and Issues in Machine Learning

Introduction

- A successful understanding of how to make computers learn opens up new uses of computers and new levels of competence and customization.
- Algorithms have been invented that are effective for certain types of learning tasks, and a theoretical understanding of learning has begun to emerge.
- As our understanding of computers continue to mature, it seems very clear that machine learning will revolutionize the way we use computers.

Introduction-Some Successful Applications of Machine Learning

- Learning to recognize spoken words.

  All of the 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. Similar techniques have potential applications in many signal-interpretation problems.

- 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. Similar techniques have possible applications in many sensor-based control problems.

- 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 terrabytes 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. Similar techniques have applications in many practical problems where very large search spaces must be examined efficiently.

Introduction-Some disciplines and examples of their 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.
- 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.

Well-Posed Learning Problems

Definition of Machine Learning:

- 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.

Well-Posed Learning Problems

In general to have a well defined learning problem the following three features are to be identified:

- The class of tasks
- The measure of performance to be improved
- The source of experience

Well-Posed Learning Problems-Examples

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

Well-Posed Learning Problems-Examples

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

Designing A Learning System

Here we design a program to learn to **play checkers** so that we understand some of the basic design issues and approaches to machine learning. We take the following steps:

- Choosing the Training Experience
- Choosing the Target function
- Choosing a representation for the target function
- Choosing a function approximation algorithm
  - Estimating Training values
  - Adjusting the weights
- The Final Design

Choosing the Training Experience

The type of training experience available can have a significant impact on success or failure of the learner. There are three important attributes of training experience. They are:

- Whether the training experience provides direct or indirect feedback
- The degree to which the learner controls the sequence of training examples
- Representation of the distribution of examples

Direct or Indirect Feedback

- **Direct Feedback:** Learning from *direct* training examples consisting of individual checkers board states and the correct moves for each.
- **Indirect Feedback:** Learning from indirect information consisting of the move sequences and final outcomes of various games played.
- Indirect feedback has a problem of credit assignment.

Sequencing of Training Examples (settings for learning)

- The learner might rely on the teacher to select informative board states and to provide correct moves for each of them.
- The learner might propose board states that are confusing and ask the teacher for the appropriate move.
- The learner might take complete control over both the board states and the correct moves while playing against itself.

Representation of the distribution of examples

- Learning is most reliable when the training examples follow a distribution that is similar to the future text examples.

Choosing the Training Experience

- In our design we decide that our system will train by playing games against itself. Hence no external trainer need to be present and the system is allowed to as much training data as time permits. Now we have a fully specified learning task:

A checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: playing practice games against itself

Choosing the Target Function

- We now need to determine What type of knowledge will be learned and how this will be used by the performance program.
- We begin with a checkers-playing program that can generate legal moves from any board state.
- The program needs to learn how to choose the best move from among the legal moves.
- Many optimization problems fall into this class.

Choosing the Target Function

- To solve this problem we need a function that chooses the best move for any given board state. The target function is defined as follows:

  ChooseMove: B□M

- This function accepts as input any board from the set of legal board states B and produces as output an appropriate move M from the set of legal moves.
- Now the problem of improving performance P at task T is reduced to the problem of learning some target function ChooseMove.
- Given the kind of indirect training experience the target function ChooseMove is very difficult to learn.

Choosing the Target Function

- An alternative target function that will be easy to learn is an evaluation function that assigns a numerical score to any given board state. Let us call the function V and it is defined as follows:

$$V: B \rightarrow R$$

- This function maps a legal board state to some real value.

Choosing the Target Function

- Let us now define the target value V(b) for an arbitrary board state b in B.
  - If b is final board state that is won, then V(b)=100
  - If b is final board state that is lost, then V(b)=-100
  - If b is a final board state that is drawn, then V(b)=0
  - 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.
- This definition of the function is a **nonoperational** definition because
  - The last case requires searching ahead for the optimal line of play until the end of the game, and
  - This definition is not efficiently computable

Choosing the Target Function

- The goal here is to discover an operational description of V; that is a description that is useful in evaluating states and selecting moves for the checkers game.
- Now the learning task is reduced to the problem of discovering an **operational** description of the ideal target function V.
- We actually expect learning algorithms to acquire some approximation to the target function, and therefore learning a target function is often called as **function approximation**.
- We will use the symbol $\hat{V}$ to refer to the function that is actually learned, to distinguish it from the ideal target function V.

Choosing a Representation for the Target Function

- The program can be allowed to represent $\hat{V}$ using:
  - A large table with a distinct entry specifying the value for each distinct board state.
  - We could allow it to represent $\hat{V}$ using a collection of rules that match against features of the board state.
  - A quadratic polynomial function of predefined board features.
  - An artificial neural network.
- The choice of representation of the function approximation is a tradeoff between:
  - A very expensive representation to allow representing as close an approximation as possible to the ideal target function.
  - The more expensive the representation the more training data the program will require.

Choosing a Representation for the Target Function

- Here in this example we choose a simple representation in which for any given board state, the function $\hat{v}$ 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
  - X6: the number of red pieces threatened by black.
- Our learning program will represent $\hat{v}$ (b) as a linear function of the form:
- $$\hat{v}(b) = w_0+w_1x_1+w_2x_2+w_3x_3+w_4x_4+w_5x_5+w_6x_6$$

Choosing a Representation for the Target Function

- The partial design of the checkers learning program is as follows:
- 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: B □ R
- Target function representation:

$\hat{v}(b) = w_0+w_1x_1+w_2x_2+w_3x_3+w_4x_4+w_5x_5+w_6x_6$

Choosing a function approximation algorithm

- A set of training examples describing a given board state b and the training value $V_{train}(b)$are needed to learn the target function $\hat{V}$
- Each training example is represented as an ordered pair $<b,V_{train}(b)>$
- An example training pair is as follows:

$$<<x_1=0,x_2=4,x_3=0,x_4=1,x_5=0,x_6=0>,+100>$$

- There are two steps to be taken as part of the function approximation algorithm
  - Estimating training values
  - Adjusting the weights

Estimating Training Values

- The only information available to the learner as far as the example checkers is concerned is whether the game is eventually won or lost.
- Here we require training examples which assign scores to board states.
- The most difficult task is assigning scores to intermediate board states that occur before the game's end.
- The rule for estimating training values can be summarized as follows:

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

Adjusting the Weights

- The remaining task is to specify the algorithm for choosing the weights w_i to best fit to the set of training examples $\langle b, V_{train}(b) \rangle$ just estimated.
- One very common approach is the estimate the weights so that the squared error E between the training values and the predicted values is minimized.

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

- Here we need to minimize E.

Adjusting the Weights

- Here we use an algorithm called as least mean square (LMS) to update the weights.

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

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

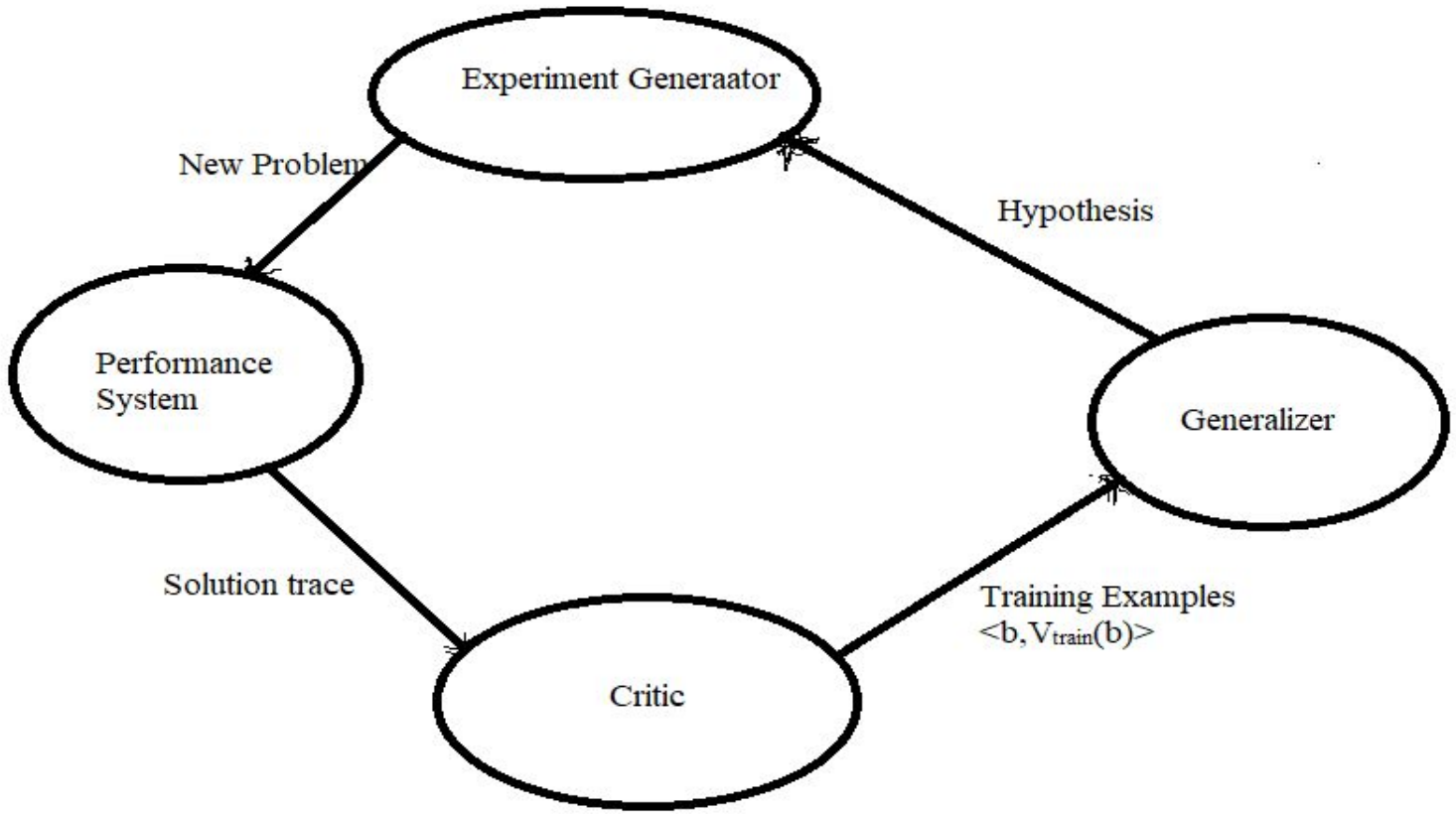
- Where η is a small constant that moderates the size of the weight update.

The Final Design

- The Final design of our checkers learning system is characterized by four distinct modules namely:
- The Performance System
- The Critic
- Generalizer
- Experiment Generator

The Final Design

Diagrammatic representation of the relationship between the four modules of our checkers learning system



The Performance System

- This is the module that solves the given performance task like the checkers game using the learned target function.
- It takes a new problem as input and produces a trace of the solution as the output.
- In the case of our checkers game the performance system chooses the next move which is decided by the e $\hat{V}$ luation function.
- The performance improve as the evaluation function becomes increasingly accurate.

The Critic

- This module takes as input the solution trace of the game and produces training examples of the target function as the output.
- In the case of our checkers game the critic corresponds to the rule for producing the training examples.

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

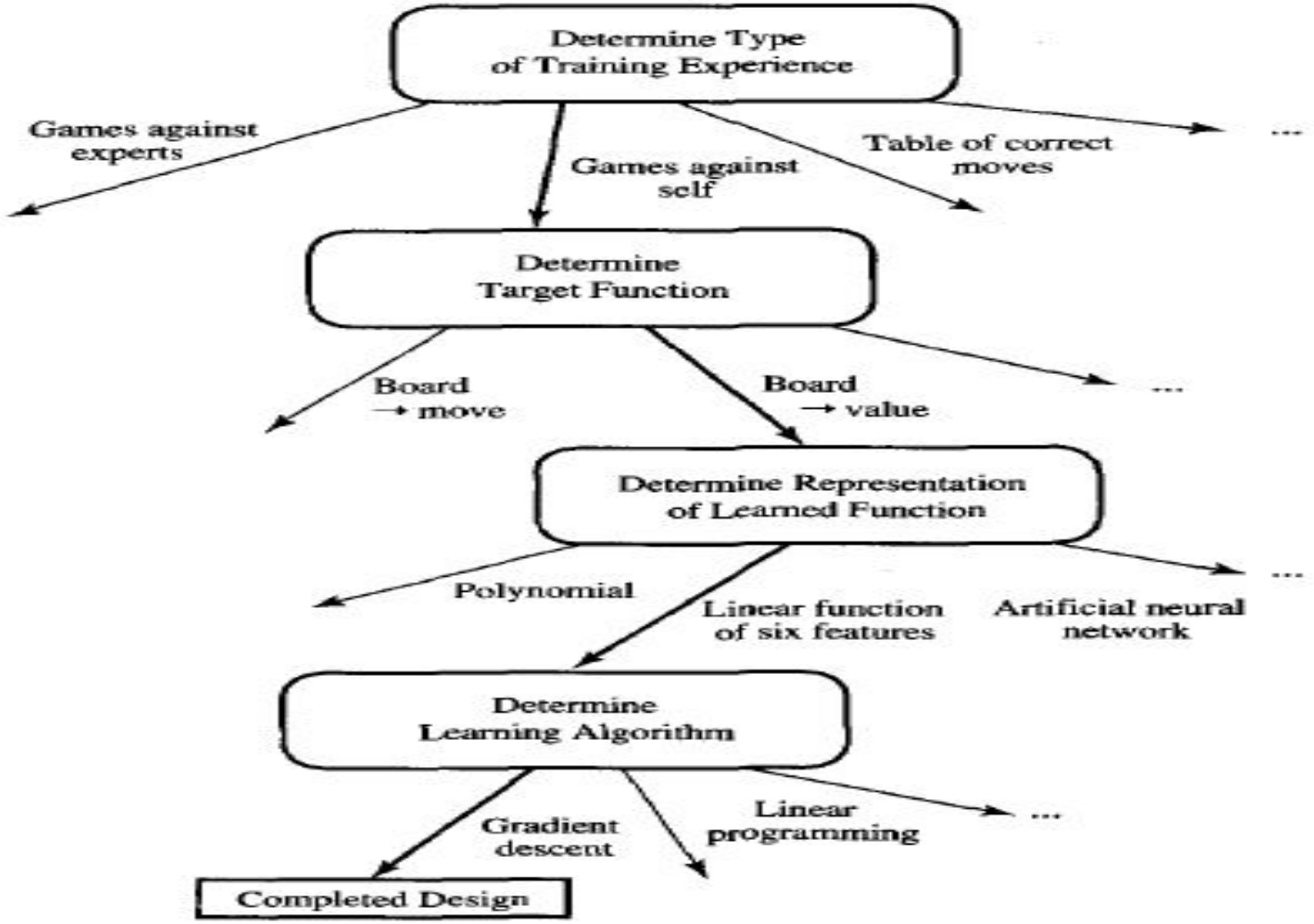
The Generalizer

- This module takes the training examples as input and produces the hypothesis which is the estimate of the target function as the output.
- It generalizes from the given specific examples.
- In the case of our checkers problem this module uses the LMS algorithm and generalizes $\hat{V}$ described by the learned weights $w_0, w_1, w_2, w_3, w_4, w_5, w_6$.

Experiment Generator

- This module takes as input the current hypothesis and produces a new problem as output for the performance system to explore.
- In the current checkers problem the current hypothesis is $\hat{V}$ and the new problem is the checkers game with the board's initial state.
- Our experiment generator is a very simple module which outputs the same board with the initial state each time.
- A better experiment generator module would create board positions to explore particular regions of the state space.

Summary of choices in designing the checkers learning problem



Perspectives and Issues in Machine Learning

- In machine learning there is a very large space of possible hypothesis to be searched.
- The job of machine learning algorithm is to search for the one that best fits the large observed space and any prior knowledge held by the learner.
- In the checkers game example the LMS algorithm fits the weights each time the hypothesized evaluation function predicts a value that differs from the training value.

Perspectives and Issues in Machine Learning

Issues in Machine Learning

- What algorithms exist for learning general target functions from specific training examples?
- How much training data is sufficient?
- When and how does the prior knowledge held by the learner guide the process of generalizing from examples?
- What is the best strategy for choosing a useful next training experience?
- How does the choice of this strategy alter the complexity of the learning problem?
- What specific functions should the system attempt to learn? Can this process be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Concept Learning and the General to Specific Ordering

- Introduction
- A Concept Learning Task
- Concept Learning as Search
- Find-S: Finding a Maximally Specific Ordering of Hypothesis
- Version Spaces and the Candidate Elimination Algorithm
- Remarks on Version Spaces and Candidate Elimination
- Inductive Bias

Introduction

- Concept learning can be thought of as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples.
- The activity of learning mostly involves acquiring general concepts from specific training examples which is called as concept learning.
- Each concept can be thought of as a Boolean-valued function defined over a large set. (Eg: a function defined over all animals, whose value is defined as true for birds and false for other animals).
- Concept learning is all about inferring a Boolean valued function from training examples of its input and output.

A Concept Learning Task

- Description of Concept Learning Task
- Notation
- The Inductive Learning Hypothesis

Description of A Concept Learning Task

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

- Let us consider the example task of learning the target concept "the days on which I enjoy my favorite water sport". The above table describes a few examples.
- Let us look at the hypothesis representation for the learner.
- Here we have a very simple representation in which each hypothesis consists of a conjugation on the instance attributes.
- For each attribute the hypothesis will be indicated by:
  - a "?" indicating any value is accepted for this attribute
  - a "Ø" indicating no value is accepted
  - or specify a single required value for the attribute
- For example: (rainy,?,high,?,?,?) ,(?,?,?,?,?,?) and (Ø, Ø, Ø, Ø, Ø, Ø)

Description of A Concept Learning Task

- Any concept learning task can be described by:
  - The set of instances over which the target function is defined (X)
  - The target function (c)
  - The set of candidate hypothesis considered by the learner, and (H)
  - The set of available training examples (D)

Notation

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

The Inductive Learning Hypothesis

- The learning task in the previous example is to determine a hypothesis h identical to the target concept c over the entire set of instances X.
- The information available about c is only its value over the training examples.
- The inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data.
- The inductive learning hypothesis is any hypothesis found to approximate the target function well over a sufficiently large set of training examples. It will also approximate the target function well over other unobserved examples.

Concept Learning as Search

- Concept Learning as Search (Description)
- General to Specific Ordering of Hypothesis

Description of Concept Learning as Search

- Concept learning is a task of searching through a large space of hypothesis implicitly defined by the hypothesis representation.
- The goal is to find the hypothesis that best fits the training examples.
- It is the selection of the hypothesis representation that defines the space of all hypothesis the program can ever represent and therefore can ever learn.
- Let us consider the set of instances X and hypothesis H in the previous example of EnjoySport.
- The instance space contains a total of 3*2*2*2*2*2 =96 distinct instances and the hypothesis space contains a total of 5*4*4*4*4*4=5120 syntactically distinct hypothesis.

Description of Concept Learning as Search

- We can observe that every hypothesis containing one or more “Ø” represent the empty set of instances, which classify every instance as negative.
- The number of semantically distinct hypothesis is 1+(4*3*3*3*3*3)=973.
- If learning is viewed as a search problem the study of learning algorithms will examine different strategies for searching the hypothesis space.
- The learning algorithm will be particularly interested in algorithms capable of efficiently searching infinite space of hypothesis to find the hypothesis that best fits the training data.

General to Specific Ordering of Hypothesis

- There is a naturally occurring structure that exists for any concept learning problem: a general-to-specific ordering of hypothesis.
- This structure helps us in designing learning algorithms that exhaustively search even infinite hypothesis space without explicitly enumerating every hypothesis.
- Let us consider two hypothesis:
  - h_1=(sunny,?,?,strong,?,?)
  - h_2=(sunny,?,?,?,?,?)
- h_2 imposes fewer constraints on the instances and therefore classifies more instances as positive. We can say that h_2 is **more general than** h_1.

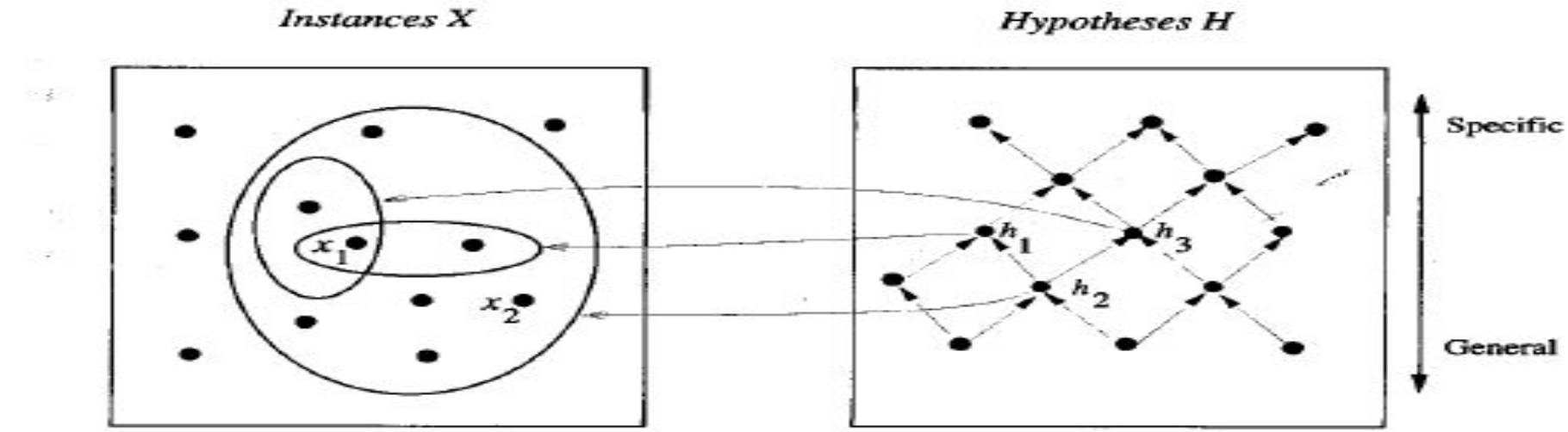
General to Specific Ordering of Hypothesis

- The definition of “more general than” relationship between hypothesis can be defined as follows:
- For any instance x in X and hypothesis h in H, x **satisfies** h iff h(x)=1
- Definition of “more general than or equal to”

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

- Now we say that h_j is **more general than** h_k written ($h_j >_g h_k$) iff ($h_j >=_g h_k$) and (h_k !>=$_g$ h_j).
- We can say that h_k is more specific than h_j and h_j is more general than h_k.

General to Specific Ordering of Hypothesis



The relation $>=_g$ defines a partial order over the hypothesis space (the relation is reflexive, antisymmetric and transitive).

The $>=_g$ relation is important because it provides a useful structure over the hypothesis space H for any concept learning problem.

Find-S: Finding A Maximally Specific Hypothesis

- The best way to use "the more general than" partial ordering to organize the search for the best hypothesis is to begin with the most specific possible hypothesis in H, then generalize it each time it fails to cover an observed positive training example.
- Find-S is an algorithm which uses this partial ordering very effectively.

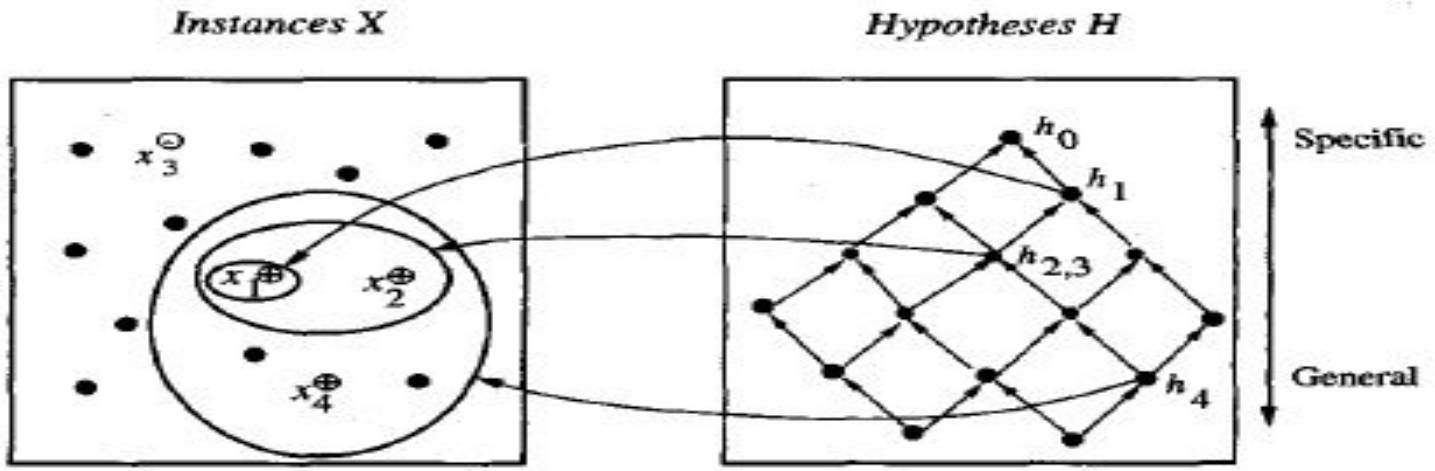
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   - For each attribute constraint a_i in h
     If the constraint a_i is satisfied by x
     Then do nothing
     Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Find-S: Finding A Maximally Specific Hypothesis

- The first step of Find-S is to initialize h to the most specific hypothesis in H (for the example EnjoySport).
  - h←(∅, ∅, ∅, ∅, ∅, ∅)
- After observing the first training example the algorithm finds a more general constraint that fits the example.
  - h←(sunny,warm,normal,strong,warm,same)
- After observing the second training example the algorithm finds a more more general constraint.
  - h←(sunny,warm,?,strong,warm,same)
- The third training example is ignored because the Find-S algorithm ignores every negative example.
- The fourth example leads to a further generalization of h.
  - h←(sunny,warm,?,strong,?,?)

Find-S: Finding A Maximally Specific Hypothesis

- The Find-S algorithm is an algorithm which illustrates a way in which the more general than partial ordering can be used to find an acceptable hypothesis.



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 ? ? >

Find-S: Finding A Maximally Specific Hypothesis

- There are a few questions left unanswered by the Find-S algorithm such as:
  - Has the learner converged to the correct target concept?
  - Why prefer the most specific hypothesis?
  - Are the training examples consistent?
  - What if there are several maximally specific consistent hypothesis?

Version Spaces and the CANDIDATE-ELIMINATION Algorithm

- Introduction
- Representation
- The List-Then-Eliminate Algorithm
- A more compact representation for version spaces
- CANDIDATE-ELIMINATION Learning Algorithm
- An Illustrative Example

Version Spaces and the CANDIDATE-ELIMINATION Algorithm (Introduction)

- A new approach to concept learning is CANDIDTE-ELIMINATION algorithm which addresses several limitations of the Find-S algorithm
- The idea in CANDIDATE-ELIMINATION algorithm is to output a description of the set of all hypotheses consistent with the training examples.
- CANDIDATE-ELIMINATION provides a useful conceptual framework for introducing several fundamental issues in machine learning.

Representation

- The CANDIDATE-ELIMINATION algorithm finds all describable hypotheses that are consistent with the observed training examples.
- Let us first try to understand what is being consistent with the observed training 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.

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

Representation

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.

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

The LIST-THEN-ELIMINATE Algorithm

The LIST-THEN-ELIMINATE Algorithm

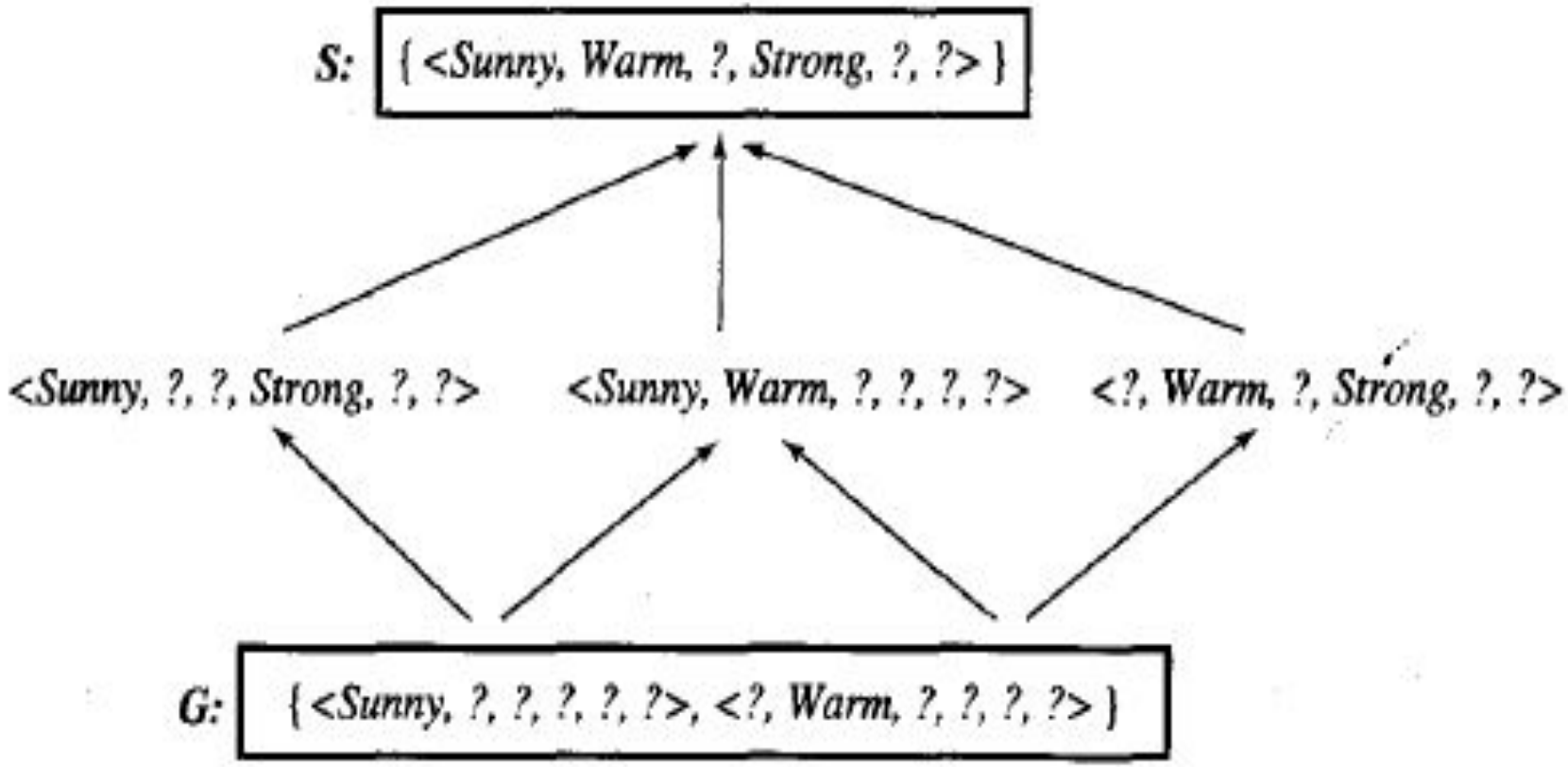
1. $VersionSpace \leftarrow$ a list containing every hypothesis in H
2. For each training example, $\langle x, c(x) \rangle$

   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 of Version Spaces

- The CANDIDATE-ELIMINATION algorithm employs a better representation of the version space.
- The version space is represented by its most general and least general members.
- These members form the general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

A more Compact Representation of Version Spaces



A more Compact Representation of Version Spaces

- It Is intuitively probable that we can represent the version space in terms of its most specific and most general members.
- Here are the definitions for the boundary sets G and S.

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.

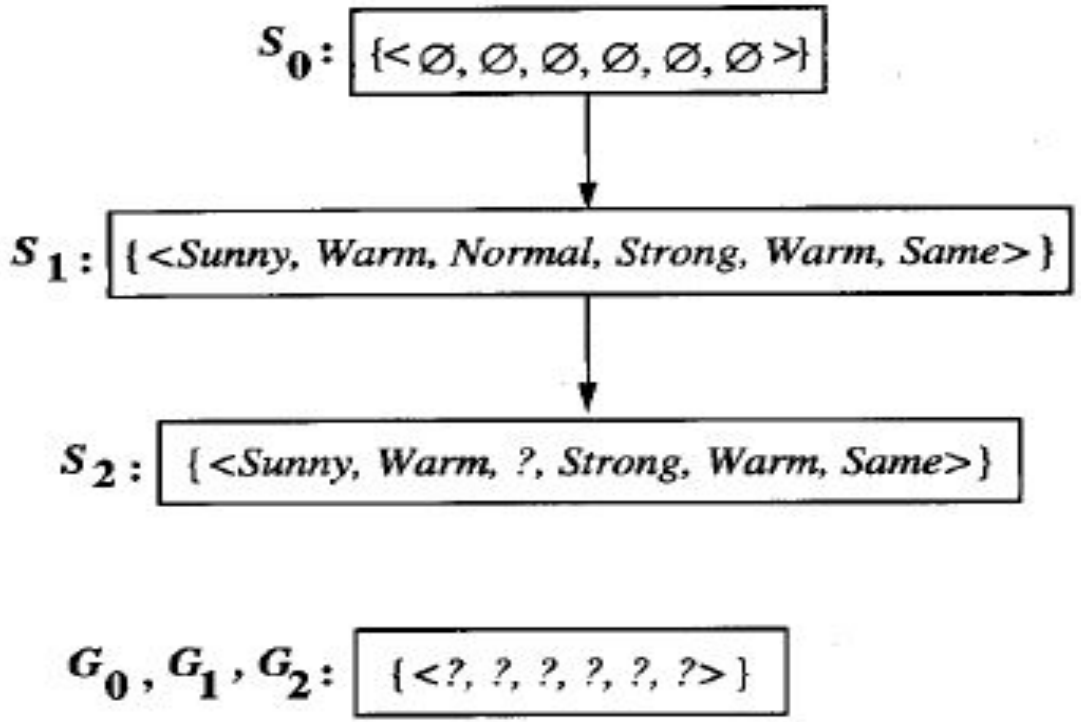
$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

CANDIDATE-ELIMINATION Learning 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

An Illustrative Example

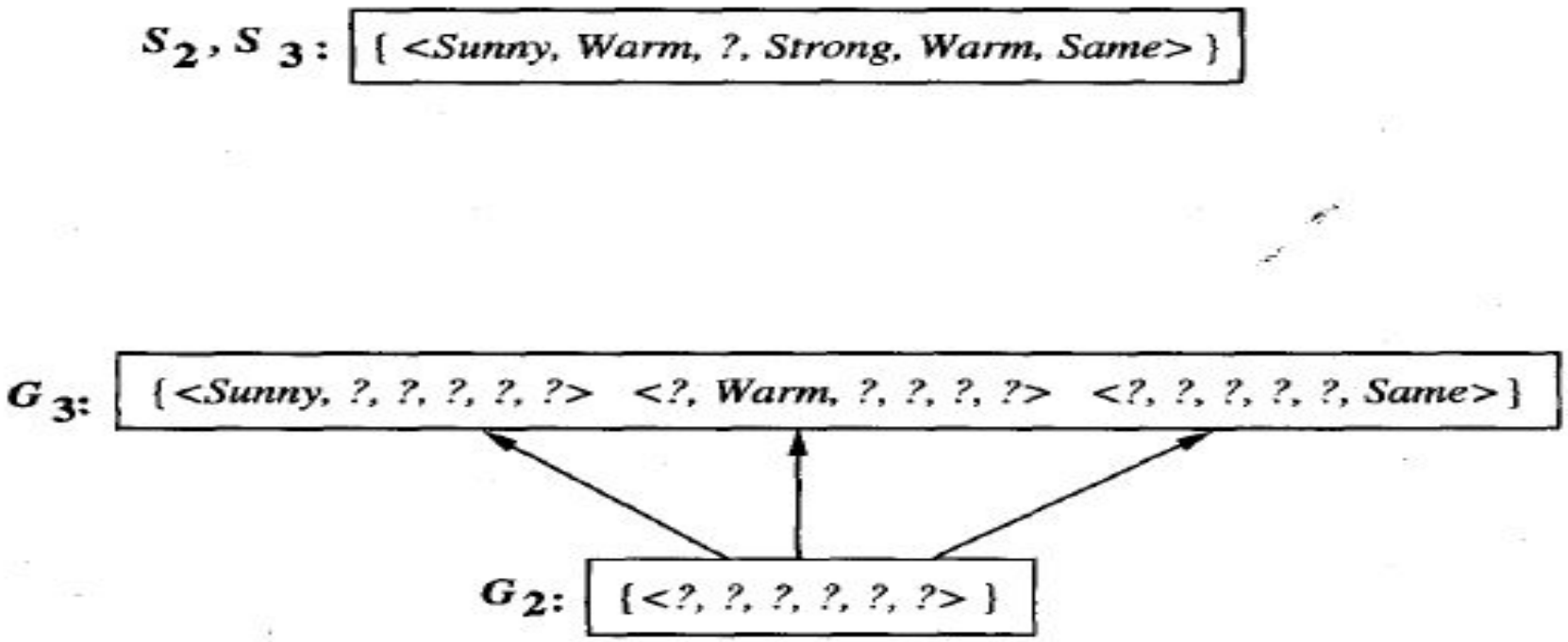


Training examples:

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

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

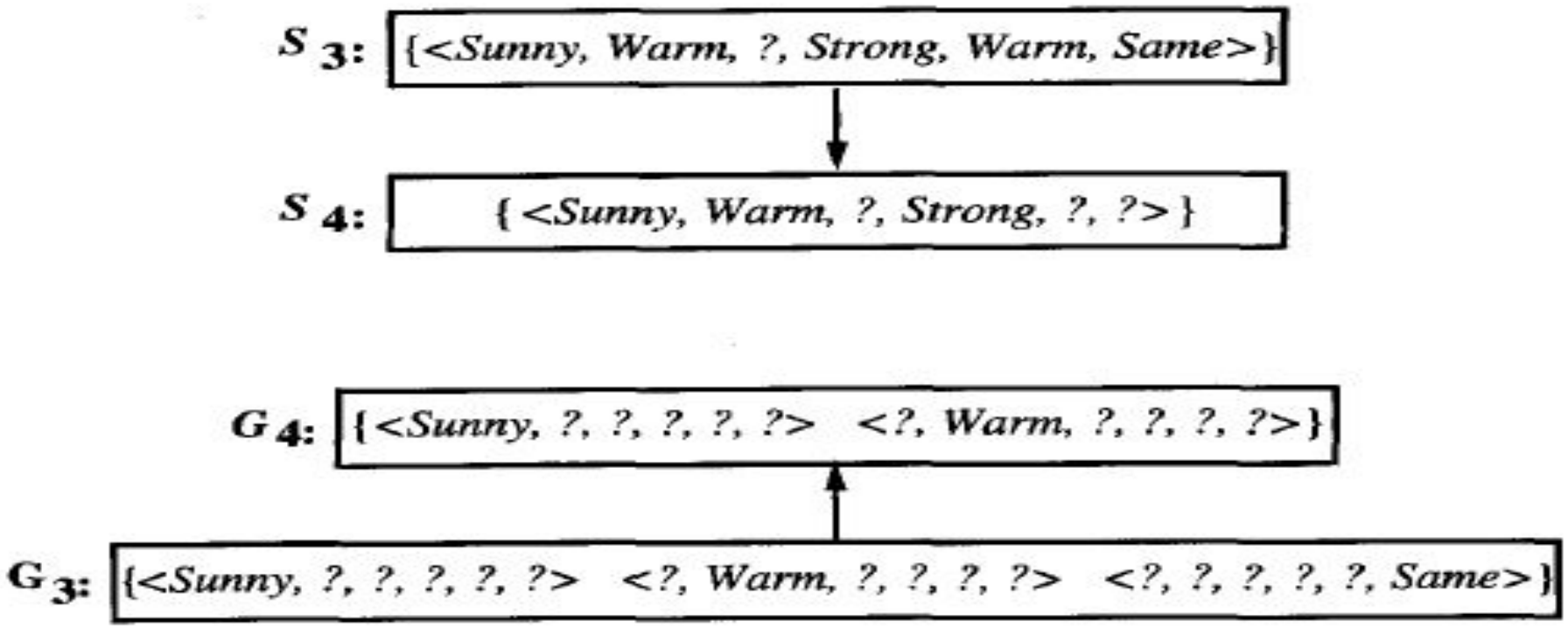
An Illustrative Example



Training Example:

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

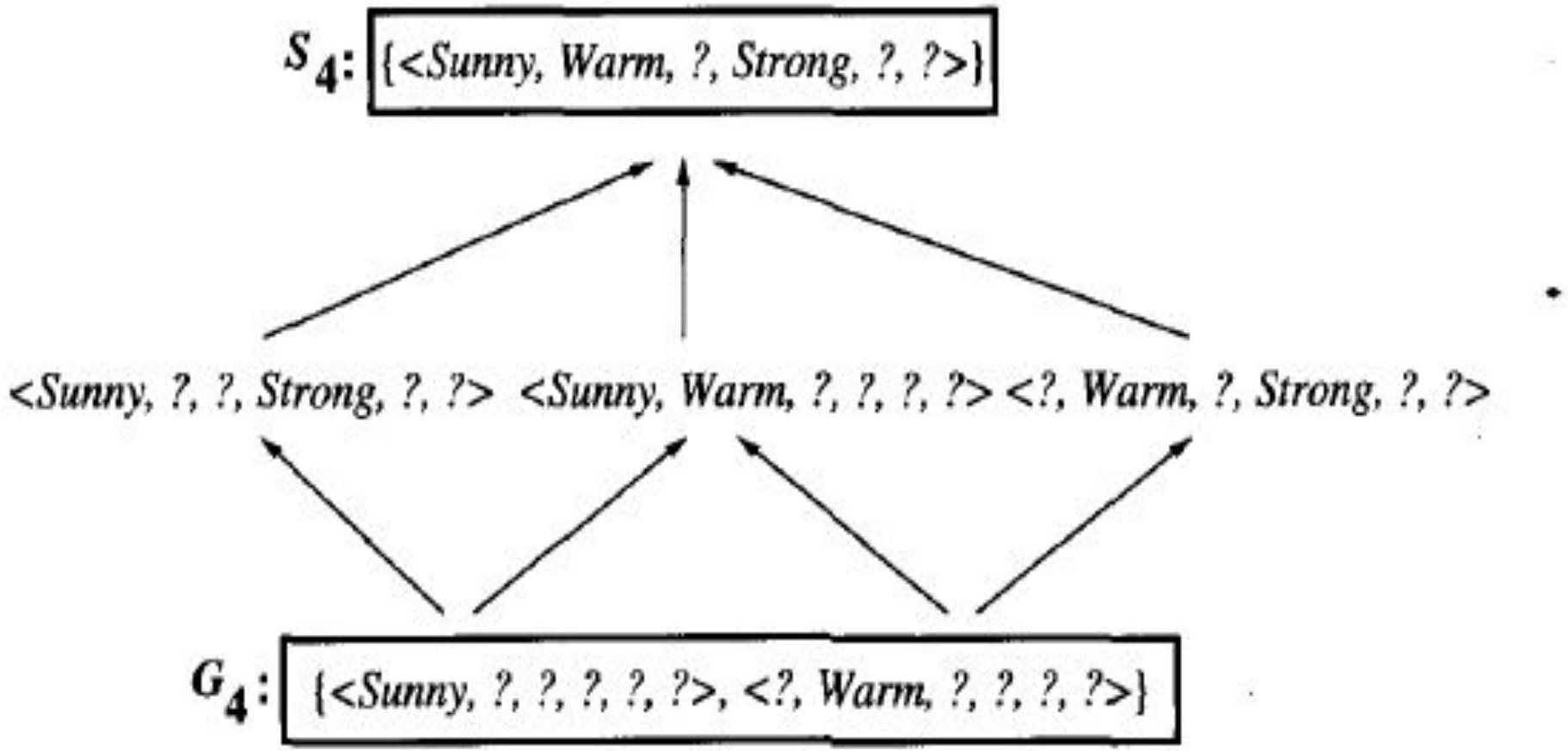
An Illustrative Example



Training Example:

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

An Illustrative Example



Remarks on Version Spaces and CANDIDATE-ELIMINATION

- Will the CANDIDATE-ELIMINATION Algorithm converge to the correct hypothesis?
- What training example should the learner request next?
- How can partially learned concepts be used?

Will the CANDIDATE-ELIMINATION Algorithm converge to the correct hypothesis

- The version space learned by the CANDIDATE-ELIMINATION algorithm will converge to the hypothesis that correctly represents the target concept, provided:
  - There are no errors in the training examples
  - There is some hypothesis in H that correctly describes the target concept.
- The target concept is exactly learned if the S and G boundaries converge to a single, identical hypothesis.
- Incase there are errors in the training data the S and G boundaries converge to an empty version space.
- Similarly though the training examples are correct if the hypothesis representation cannot describe the target concept then also the algorithm will not converge to the target concept.

What Training Example should the learner request next

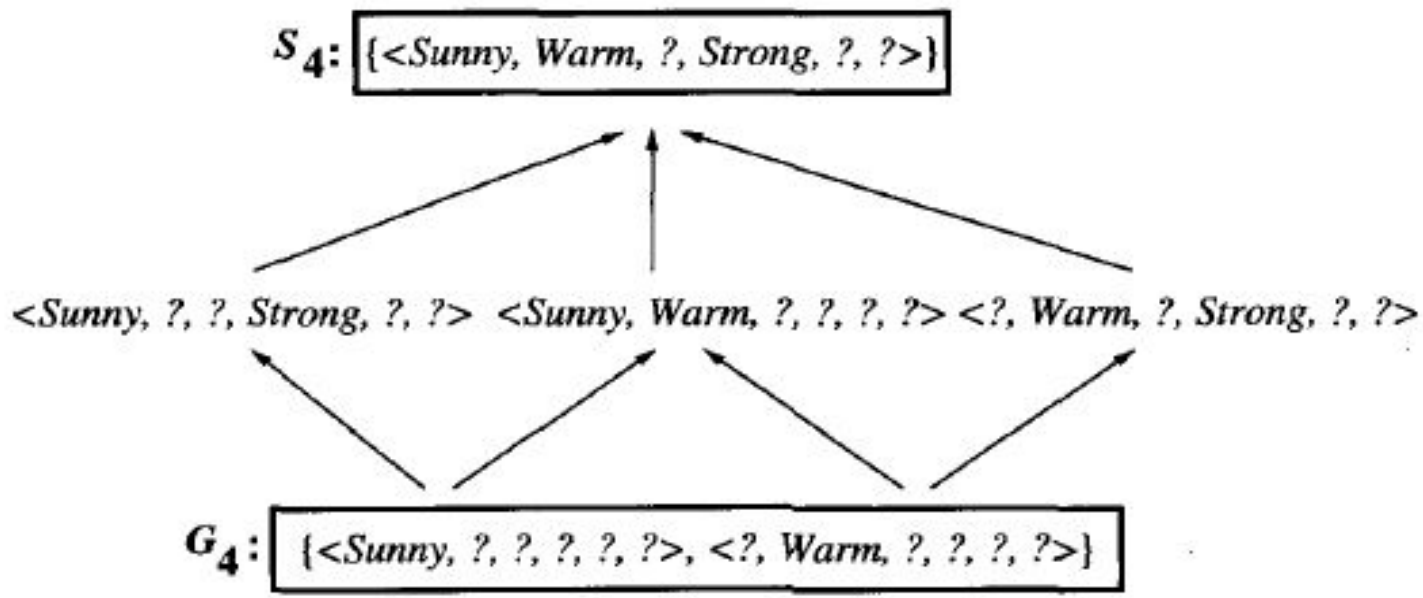
- Suppose that the learner is allowed to conduct experiments in which it chooses the next instance, then obtains the correct classification for this instance from an external oracle (e.g., nature or a teacher).
- The term ***query*** is used to refer to such instances constructed by the learner, which are then classified by an external oracle.
- Let us look for a query in the example EnjoySport concept chosen by the learner.
  - *(Sunny, Warm, Normal, Light, Warm, Same)*
- If the trainer classifies this instance as positive S boundary of the version space can be generalized.
- If the trainer classifies it as negative the G boundary can be specialized.

What Training Example should the learner request next

- The optimal query strategy for a concept learner is to generate instances that satisfy exactly half of the hypothesis in the current version space.
- The correct target concept can then be found in $\log_2 VS$ experiments.

How can partially learned concept be used

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	?



Inductive Bias

- Inductive Bias Introduction
- A Biased Hypothesis Space
- An Unbiased Learner
- The Futility of Bias-Free Learning

Inductive Bias Introduction

- The **CANDIDATE-ELIMINATION** algorithm will converge towards the true target concept provided it is given accurate training examples and provided its initial hypothesis space contains the target concept.
- What if the target concept is not contained in the hypothesis space?
- Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis?
- How does the size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances?
- How does the size of the hypothesis space influence the number of training examples that must be observed?

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

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

- Earlier we restricted the hypothesis space to include only conjunctions of attribute values.
- Because of this restriction, the hypothesis space is unable to represent even simple disjunctive target concepts such as ***"Sky*** = ***Sunny*** or ***Sky*** = ***Cloudy."***

An Unbiased Learner

- To ensure that the target concept is in the hypothesis space H, the hypothesis space must be able to represent every teachable concept.
- In our EnjoySport example there are 2^{96} distinct target concepts that could be defined over the instance space.
- Our conjugative hypothesis is only able to represent 973 of these.
- We need to reformulate the EnjoySport learning task by including disjunctions, conjunctions and negations.
- For example the target concept “sky=sunny or sky=cloudy” can be represented as :

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

An Unbiased Learner

- Let us assume we present three positive examples (x1,x2,x3) and two negative examples (x4,x5) to the learner.
- The S boundary and the G boundary will be as follows:

$$S : \{(x_1 \vee x_2 \vee x_3)\} \qquad G : \{\neg(x_4 \vee x_5)\}$$

- The only examples that will be unambiguously classified by S and G are only the observed training examples.
- To converge to a single final target concept we need to present all the all the instances of X as examples.

The Futility of Bias Free Learner

- A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.
- Inductive learning requires some form of prior assumptions, or inductive bias, we will find it useful to characterize different learning approaches by the inductive bias they employ.
- The inductive inference step preformed by the learner can be described as follows: $(D_c \wedge x_i) \succ L(x_i, D_c)$

- Where the notation $y \succ z$ indicates that z is inductively inferred from y.

The Futility of Bias Free Learner

- We define the inductive bias of L to be the set of assumptions B such that for all new instances xi:

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

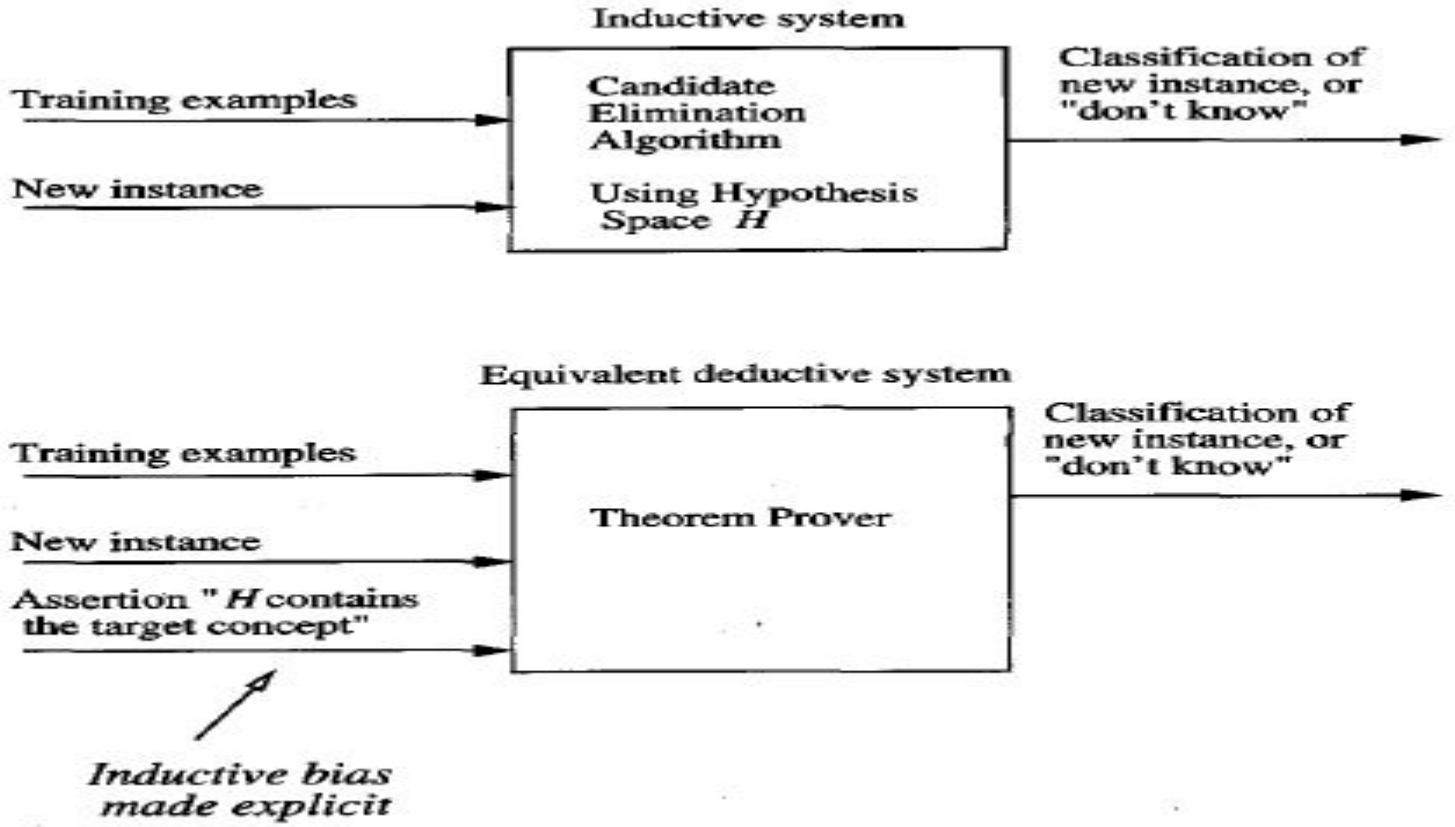
- Where the natation $y \vdash z$ indicates that z is deductively inferred from y.

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

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)] \qquad (2.1)$$

The Futility of Bias Free Learner

- The inductive bias of the CANDIDATE-ELIMINATION algorithm is that the target concept is contained in the given hypothesis.



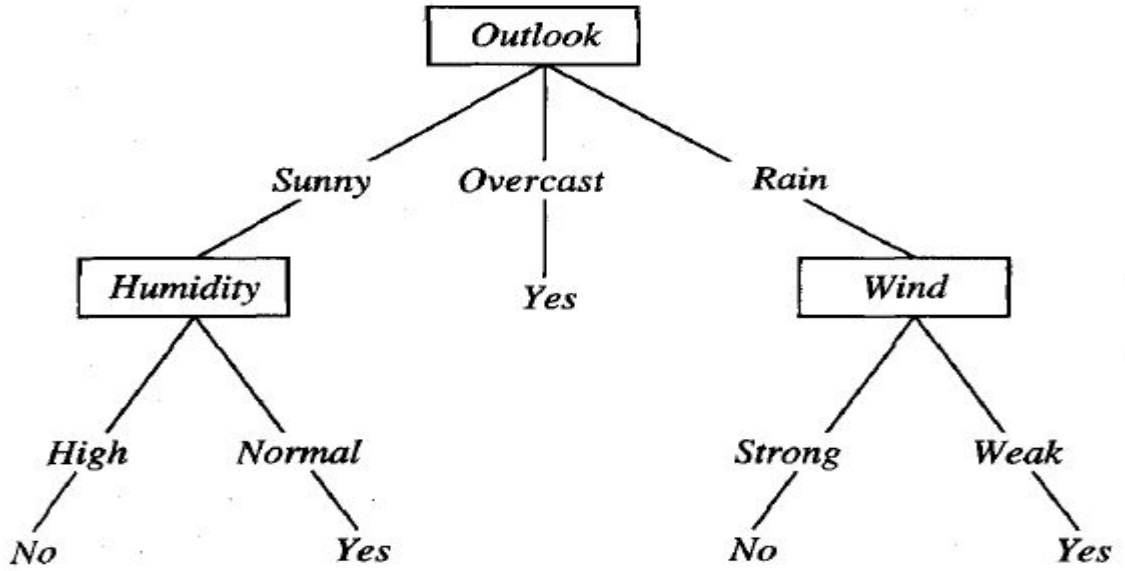
Decision Tree Learning

- Introduction
- Decision Tree Representation
- Appropriate Problems for Decision Tree Learning
- The Basic Decision Tree Learning Algorithm
- Hypothesis Space Search in Decision Tree Learning
- Inductive Bias in Decision Tree Learning
- Issues in Decision Tree Learning

Introduction

- Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.
- Learned trees can also be re-represented as sets of if-then rules to improve human readability.
- These methods have been successfully applied in many tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

Decision Tree Representation



- The above figure is a decision tree for the concept ***play tennis.***
- Now let us look at how this tree classifies the following instance (Outlook=sunny,Humidity=High,Wind=Strong)
- The expression for the tree in the previous slide is as follows: (Outlook=sunny ^ Humidity=normal)v(Outlook=overcast)v(Outlook=rain ^Wind=weak)

Appropriate Problems for Decision Tree Learning

- The following are the characteristics of problems suitable for decision tree learning
  - Instances are represented by attribute value pairs
  - The target function has discrete output values
  - Disjunctive descriptions may be required
  - The training data may contain errors
  - The training data may contain missing attribute values
- Decision tree learning has therefore been applied to problems such as
  - learning to classify medical patients by their disease,
  - equipment malfunctions by their cause,
  - and loan applicants by their likelihood of defaulting on payments.
- Such problems, in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as ***classification problems.***

The Basic Decision Tree Algorithm

- Which attribute is the best classifier?
- An illustrative Example

The Basic Decision Tree Algorithm

- The core ID3(Iterative Dichotomizer 3) algorithm used for learning decision trees employs a top-down, greedy search through the space of possible decision trees.
- The ID3 algorithm learns decision trees by constructing them top-down beginning with the question “Which attribute should be tested at the root of the tree?”.
- ID3 uses **information gain** measure to select among the candidate attributes at each step while growing the tree.

The Basic Decision Tree Algorithm

ID3(*Examples*, *Target_attribute*, *Attributes*)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
  - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
  - The decision attribute for *Root* $\leftarrow A$
  - For each possible value, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
    - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
    - If $Examples_{v_i}$ is empty
      - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
      - Else below this new branch add the subtree ID3($Examples_{v_i}$, *Target_attribute*, *Attributes* − {A}))
- End
- Return *Root*

---

* The best attribute is the one with highest *information gain*.

Which Attribute is the best Classifier

- Entropy measures homogeneity of examples
- Information gain measures the expected reduction in entropy

Entropy measures homogeneity of examples

- Entropy is a measure that characterizes the impurity of a collection of examples.
- In a collection S containing positive and negative examples of some target concept the entropy of S relative to the Boolean classification is:

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

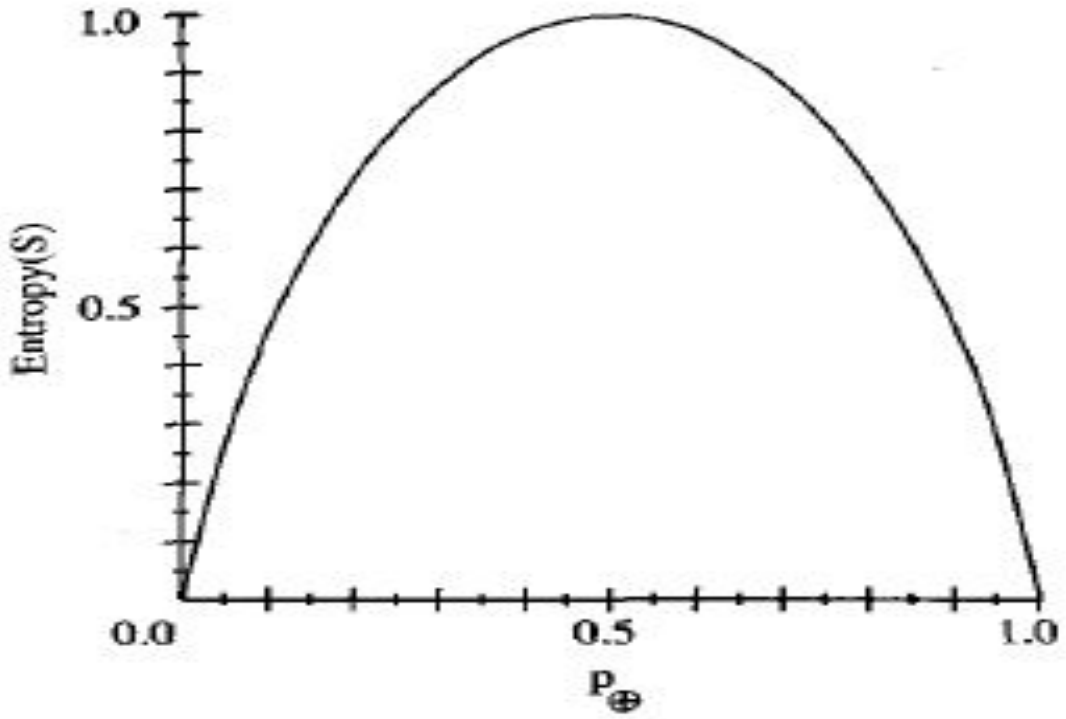
- where p+, is the proportion of positive examples in S and p-, is the proportion of negative examples in S. In all calculations involving entropy we define 0 log 0 to be 0.

Entropy measures homogeneity of examples

- Let us assume S is a collection of 14 examples of some Boolean concept, including 9 positive and 5 negative examples. The entropy of S relative to this Boolean classification is :
- Entropy([9+,5-])= $-(9/14)\log_2(9/14)-(5/14)\log_2(5/14)=0.940$
- Entropy will be 0 if all members belong to the same class and it will be 1 if there is an equal distribution of the examples.
- If the target concept can take c different values then the entropy measure is:

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

Entropy measures homogeneity of examples



Information Gain measures the expected reduction in entropy

A measure called Information gain simply measures the amount of reduction in the entropy caused by partitioning the examples according to a given attribute.

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Here S is the collection of examples, A is the given attribute, Values(A) is the set of all possible values for the attribute A and Sv is the subset of S for which attribute A has value v.

Information Gain measures the expected reduction in entropy

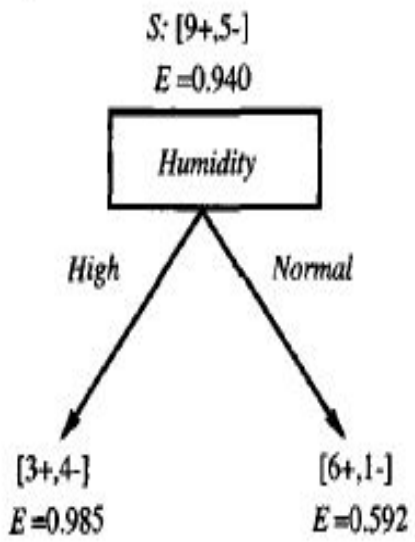
- Suppose S is a collection of training example days described by an attribute “wind” which can have values “weak” and “strong”. The information gain by partitioning the 14 examples by the attribute “wind” is calculated as:

$$
\begin{aligned}
Values(Wind) &= Weak, Strong \\
S &= [9+, 5-] \\
S_{Weak} &\leftarrow [6+, 2-] \\
S_{Strong} &\leftarrow [3+, 3-] \\
Gain(S, Wind) &= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v) \\
&= Entropy(S) - (8/14) Entropy(S_{Weak}) \\
&\quad - (6/14) Entropy(S_{Strong}) \\
&= 0.940 - (8/14) 0.811 - (6/14) 1.00 \\
&= 0.048
\end{aligned}
$$

Illustrative Example- Training Examples for the Target Concept Play Tennis

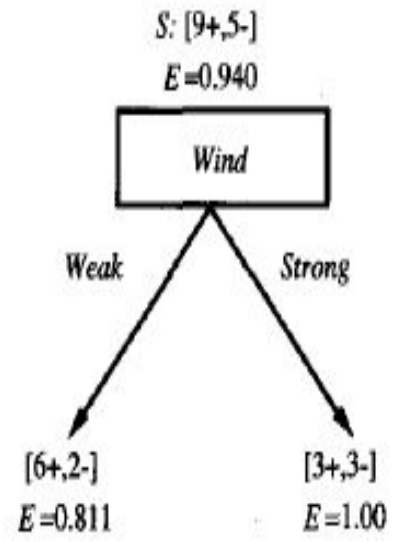
Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Illustrative Example-Information Gain for the given four attributes



Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592
= .151



Gain (S, Wind)

= .940 - (8/14).811 - (6/14)1.0
= .048

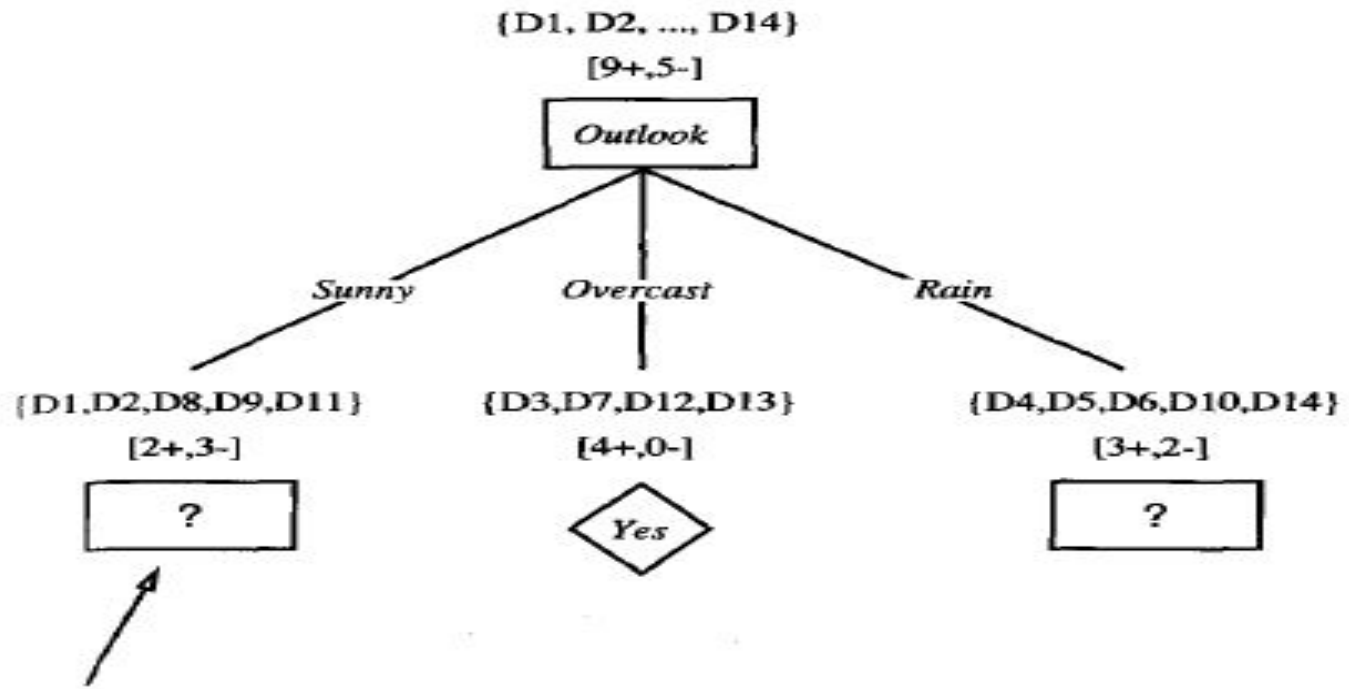
$$Gain(S, Outlook) = 0.246$$

$$Gain(S, Humidity) = 0.151$$

$$Gain(S, Wind) = 0.048$$

$$Gain(S, Temperature) = 0.029$$

Illustrative Example- The partially learned decision tree



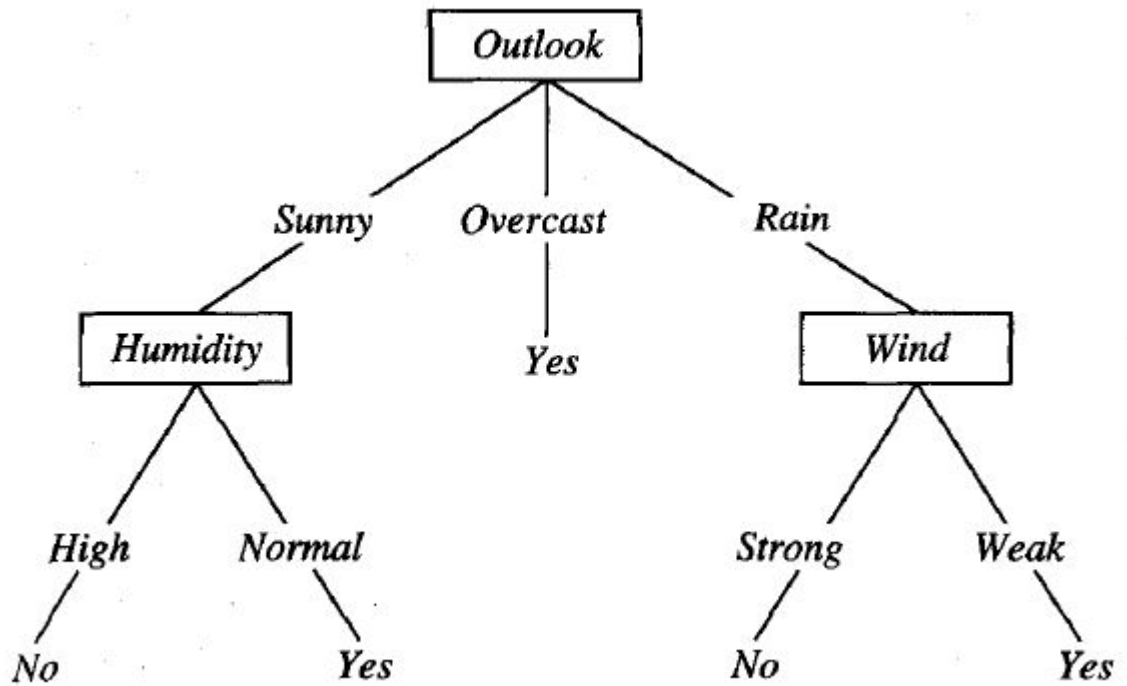
S_{sunny} = {D1,D2,D8,D9,D11}

Gain (S_{sunny}, *Humidity*) = .970 - (3/5) 0.0 - (2/5) 0.0 = .970

Gain (S_{sunny}, *Temperature*) = .970 - (2/5) 0.0 - (2/5) 1.0 - (1/5) 0.0 = .570

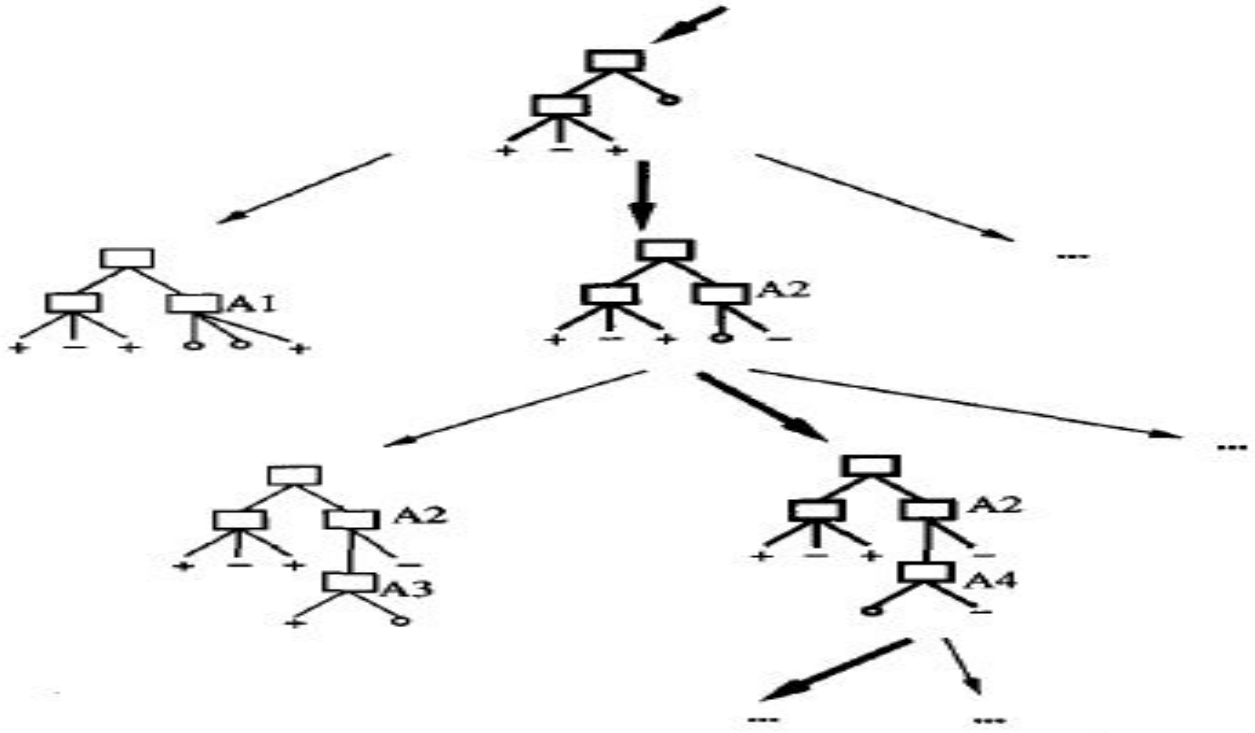
Gain (S_{sunny}, *Wind*) = .970 - (2/5) 1.0 - (3/5) .918 = .019

Illustrative Example- The Final Decision Tree



Hypothesis Space Search in Decision Tree Learning

- ID3 performs a simple to complex hill climbing search through the hypothesis space starting with the empty tree.
- The evaluation function used by ID3 is the information gain.



Hypothesis Space Search in Decision Tree Learning

- The following are the capabilities and limitations of ID3:
  - ID3's hypothesis space of all decision trees is a ***complete*** space of finite discrete-valued functions, relative to the available attributes.
  - ID3 maintains only a single current hypothesis as it searches through the space of decision trees unlike CANDIDATE-ELIMINATION.
  - ID3 in its pure form does not perform any backtracking and therefore might result in local optimization.
  - **ID3** uses all training examples at each step in the search to make statistically based (information gain) decisions regarding how to refine its current hypothesis in contrast to Find-S and CANDIDATE-ELIMINATION which use one example at a time.
  - **ID3** can be easily extended to handle noisy training data by modifying its termination criterion to accept hypotheses that imperfectly fit the training data.

Inductive Bias in Decision Tree Learning

- Restriction Bias and Preference Bias
- Why prefer short hypothesis

Inductive Bias in Decision Tree Learning

- What is the inductive bias in decision tree learning?
- ID3:
  - Selects in favor of shorter trees over longer ones
  - Selects trees that place the attributes with highest information gain closest to the root.

Restriction Bias and Preference Bias

- Let us look at the difference between the hypothesis space search of ID3 and CANDIDATE-ELIMINATION algorithm

ID3	CANDIDATE-ELIMINATION
Complete Hypothesis space is searched	Incomplete Hypothesis space is searched
It searches incompletely through this space, from simple to complex hypotheses, until its termination condition is met	It searches this space completely, finding every hypothesis consistent with the training data.
Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy.	Its inductive bias is solely a consequence of the expressive power of its hypothesis representation.
Its hypothesis space introduces no additional bias	Its search strategy introduces no additional bias.
The inductive bias of ID3 is thus a preference for certain hypotheses over others with no hard restriction on the hypotheses that can be eventually enumerated.	In contrast, the bias of the CANDIDATEELIMINATION algorithm is in the form of a categorical restriction on the set of hypotheses considered
This form of bias is typically called a **preference bias** (or, alternatively, a search bias).	This form of bias is typically called a **restriction bias** (or, alternatively, a language bias).

Why Prefer Shorter Hypothesis?

- **Occam's Razor**: Prefer the simplest hypothesis that fits the data.
- Why should one prefer simpler hypothesis?
- Let us consider decision tree hypothesis. There are many more 500 node decision trees than 5 node decision trees.
- We might therefore believe that the 5-node decision tree is less likely to be a statistical coincidence and prefer this hypothesis over the others.

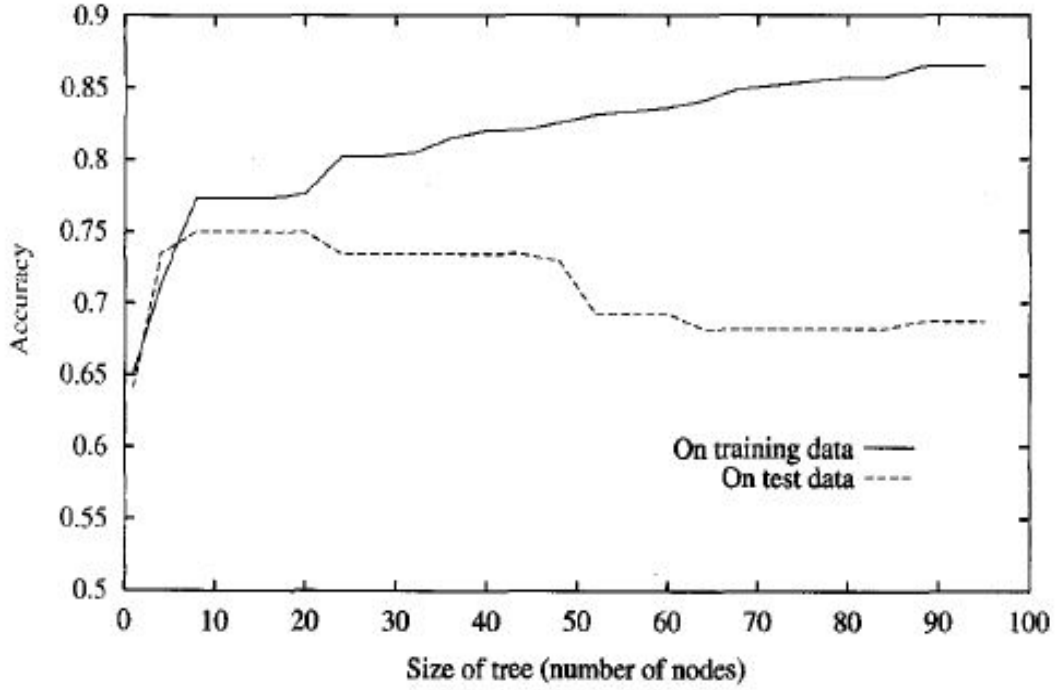
Issues in Decision Tree Learning

- Avoiding Overfitting the Data
  - Reduced Error Pruning
  - Rule Post-Pruning
- Incorporating Continuous Valued Attributes
- Alternative Measures for selecting Attributes
- Handling Training Examples with Missing Attribute Values
- Handling Attributes with different costs

Avoiding Overfitting of Data

Definition: Given a hypothesis space H, a hypothesis $h \in H$ is said to **overfit** the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

As ID3 adds new nodes to grow the decision tree, the accuracy of the tree measured over the training examples increases monotonically. However, when measured over a set of test examples independent of the training examples, accuracy first increases, then decreases.



Avoiding Overfitting of Data

- Let us look at an example to understand overfitting of the data.
- Let us add this following positive example incorrectly as negative example to the example dataset.

$$\langle Outlook = Sunny, Temperature = Hot, Humidity = Normal,$$
$$Wind = Strong, PlayTennis = No \rangle$$

- The result is going to be a more complex tree and the future instances will not be correctly categorized.

Avoiding Overfitting of Data

- There are several approaches to avoid overfitting in decision tree learning. They can be grouped into two classes:
- approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data,
- approaches that allow the tree to over fit the data, and then post-prune the tree.

Avoiding Overfitting of Data

- A very important question is what criterion is to be used to determine the correct size of the tree:
  - Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.(training and validation set approach).
  - Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set. (chi-square test)
  - Use an explicit measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. (Minimum Description Length principle)

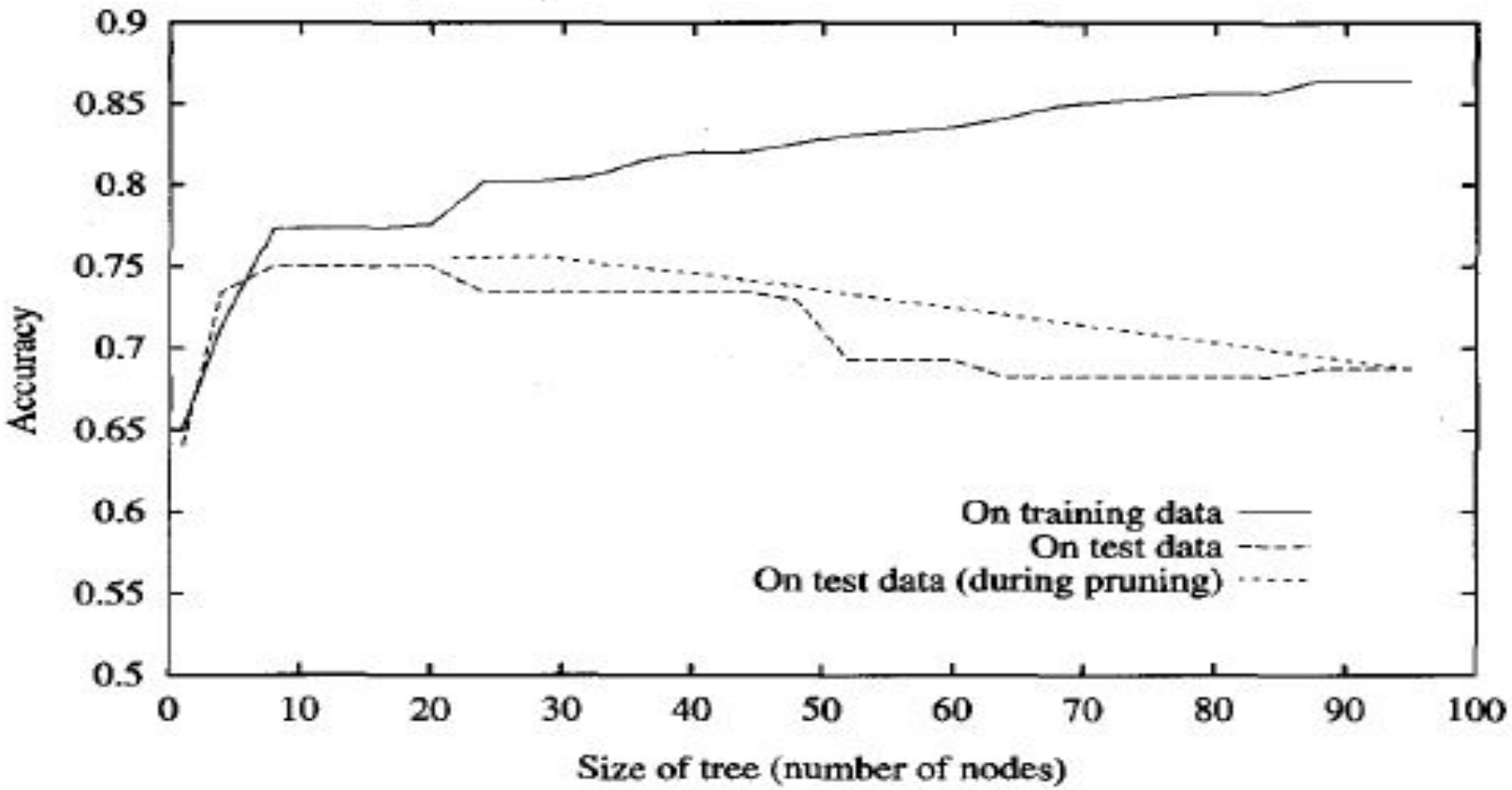
Avoiding Overfitting of Data

- In the training and validation set approach the data set is separated into two sets of examples namely:
  - Training set
  - Validation set
- There are two approaches to use a validation set to prevent overfitting. They are:
  - Reduced Error Pruning
  - Rule Post-Pruning

Reduced Error Pruning

- One approach to using a validation set to avoid overfitting of the data is called reduced error pruning.
- This technique considers each of the decision node as a candidate for pruning(removing the subtree rooted at the node and making it a leaf node).
- Pruning is done only if the pruned tree performs no worse than the original tree over the validation set.
- Pruning of nodes continues until further pruning is harmful.
- Using a separate validation dataset needs a large amount of data. This approach is not useful when the data is limited.

Reduced Error Pruning



Rule Post-Pruning

- One very successful method for finding high accuracy hypotheses is a technique we shall call rule post-pruning.
- Rule post-pruning involves the following steps:

  1. Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.

  2. Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.

  3**.** Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.

  4. Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

Rule Post-Pruning

- Each attribute test along the path from the root to the leaf becomes a rule antecedent (precondition) and the classification at the leaf node becomes the rule consequent (post-condition).
- Let us look at an example:

$$\text{IF} \quad (Outlook = Sunny) \wedge (Humidity = High)$$

$$\text{THEN} \quad PlayTennis = No$$

Rule Post-Pruning

- Why convert the decision tree to rules before pruning? There are three main advantages.
  - Converting to rules allows distinguishing among the different contexts in which a decision node is used. In contrast, if the tree itself were pruned, the only two choices would be to remove the decision node completely, or to retain it in its original form.
  - Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves.
  - Converting to rules improves readability. Rules are often easier for people to understand.

Incorporating Continuous Valued Attributes

- Our initial definition of ID3 is restricted to attributes which are discrete in nature.
- The target attribute must be discrete while other attributes can be continuous.
- For a continuous attribute A the algorithm can create a Boolean Ac that is true if A<c
- The most important question is how do we decide the threshold value c.

Incorporating Continuous Valued Attributes

- Let us look at an example. Let us assume the temperature in our playtennis data set is included as continuous valued.

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

- Here we want to pick a threshold c that produces the greatest information gain.
- First we sort the examples according to the attribute Temperature.
- The candidate thresholds are midway between the corresponding values of the attribute.
- We then identify the candidate threshold which differ in their target classification with their adjacent examples.

Alternative Measures for Selecting Attributes

- There is a natural bias in the information gain measure that favors attributes with many values over those with few values.
- One way to avoid this difficulty is to select decision attributes based on some measure other than information gain.
- One alternative measure that has been used successfully is the gain ratio.
- The gain ratio measure penalizes attributes with many values by incorporating a term, called split information. Split information is actually the entropy of S with respect to the values of attribute A.

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

Handling Training Examples with Missing Attribute values

- One strategy for dealing with missing attribute values is :
  - To assign the most common value among the training examples.
  - To assign the most common value among the training examples that have the same classification.
- A more complex procedure is to assign a probability to each of the possible values of A rather than simply assigning the most common value to A(x).

Handling Attributes with different costs

- In some learning tasks the instance attributes may have associated costs (in classifying medical diseases).
- In such tasks, we would prefer decision trees that use low-cost attributes where possible, relying on high-cost attributes only when needed to produce reliable classifications.
- A few approaches that include the cost of the attribute are:
- Tam and Schlimmer

$$\frac{Gain^2(S, A)}{Cost(A)}$$

- Nenez

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

Where $w \in [0,1]$