UNIT-1

Introduction: Well Posed Learning Problems, Designing a Learning system, Perspectives and Issues in Machine Learning

Concept Learning: Concept Learning as search, Find-S, Version Spaces and Candidate Elimination Algorithm, Inductive bias.

Decision Tree Learning: Decision Tree Representation, Decision Tree Learning Algorithm, Hypothesis Space Search.

INTRODUCTION

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

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

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

Some successful applications of machine learning

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

Why is Machine Learning Important?

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

difficult to continuously re-design systems "by hand".

1.1 WELL-POSED LEARNING PROBLEMS:

Definition: A computer program is said to learn from experience E with respect to some classof tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

To have a well-defined learning problem, three features needs to be identified:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience

Examples

1. ***Checkers game:*** 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.

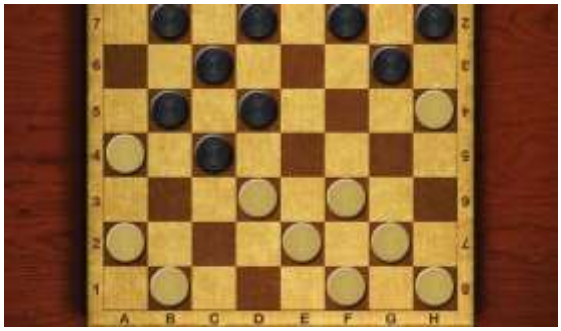


Fig: Checker game board

***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

2. ***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
3. ***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

1.2 DESIGNING A LEARNING SYSTEM:

The basic design issues and approaches to machine learning are illustrated by designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    1. Estimating training values
    2. Adjusting the weights
5. The Final Design

1. Choosing the Training Experience

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.

There are three attributes which impact on success or failure of the learner

1. Whether the training experience provides ***direct or indirect feedback*** regarding the choices made by the performance system.

    For example, in checkers game:
    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 training examples consisting of the ***move sequences*** and ***final outcomes*** of various games played. The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.

    Here the learner faces an additional problem of ***credit assignment***, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome. Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves.
    Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

2. The degree to which the ***learner controls the sequence of training examples***

   For example, in checkers game:
   The learner might depends 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 (indirect) training classifications, as it does when it learns by playing against itself with ***no teacher present***.

3. How well it represents the ***distribution of examples*** over which the final system performance P must be measured

   For example, in checkers game:
   In 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 a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.
   It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

2. Choosing the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

Let's consider 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. We must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let ***ChooseMove*** be the target function and the notation is

   $$ChooseMove : B \rightarrow M$$

   which indicate that 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.

ChooseMove is a choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system

2. An alternative target function is an ***evaluation function*** that assigns a ***numerical score*** to any given board state
   Let the target function V and the notation

$$V : B \rightarrow R$$

   which denote that V maps any legal board state from the set B to some real value. Intend for this target function V to assign higher scores to better board states. If thesystem can successfully learn such a target function V, then it can easily use it to selectthe best move from any current board position.

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

- If b is a final board state that is won, then V(b) = 100
- If b is a 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 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

3. Choosing a Representation for the Target Function

Let's choose a simple representation - for any given board state, the function ***c*** will be calculated as a linear combination of the following board features:

- x]: 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

Thus, learning program will represent 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,

- w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights w_1 through w_6 will determine the relative importance of the various board features in determining the value of the board
- The weight w_0 will provide an additive constant to the board value

4. 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 $V_{train}(b)$ for b.

Each training example is an ordered pair of the form $(b, V_{train}(b))$.

For instance, the following training example describes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore +100.

$$((x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0), +100)$$

Function Approximation Procedure

1. Derive training examples from the indirect training experience available to the learner
2. Adjusts the weights w_i to best fit these training examples

1. Estimating training values

   A simple approach for estimating training values for intermediate board states is to assign the training value of $V_{train}(b)$ for any intermediate board state b to be $\hat{V}$ (Successor(b))

   Where ,
   - $\hat{V}$ is the learner's current approximation to V
   - 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

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

2. Adjusting the weights

Specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{(b, V_{train}(b))\}$

A first step is to define what we mean by the bestfit to the training data.

One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.

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

Several algorithms are known for finding weights of a linear function that minimize E. 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 this training example

LMS weight update rule :- For each training example $(b, V_{train}(b))$

Use the current weights to calculate $\hat{V}(b)$

For each weight w_i, update it as

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

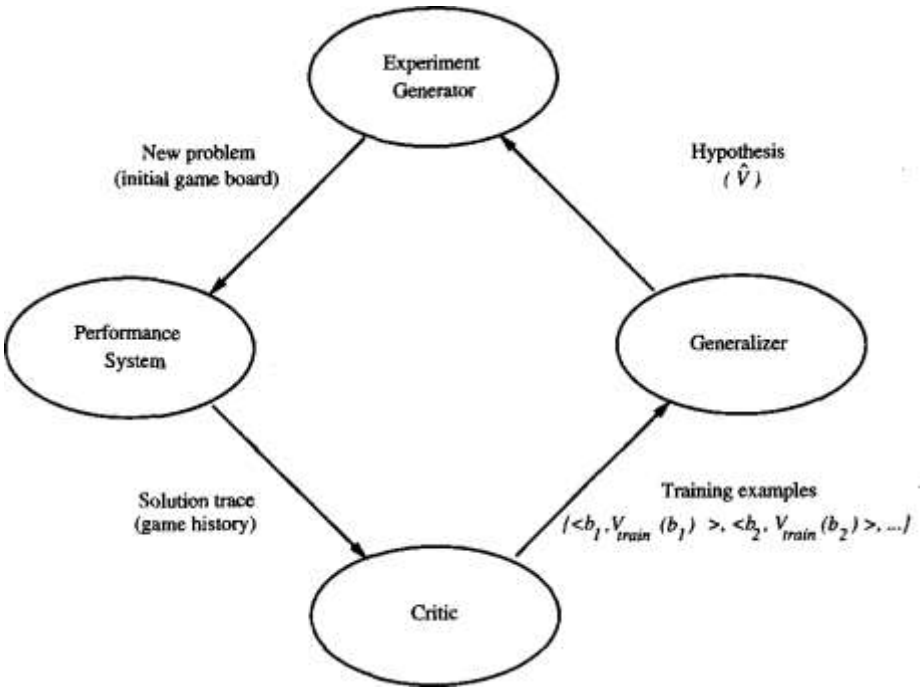
Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Working of weight update rule

- When the error $(V_{train}(b) - \hat{V}(b))$ is zero, no weights are changed.
- When $(V_{train}(b) - \hat{V}(b))$ is positive (i.e., when $\hat{V}(b)$ is too low), then each weightis increased in proportion to the value of its corresponding feature. This will raisethe value of $\hat{V}(b)$, reducing the error.
- If the value of some feature x_i is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

5. 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



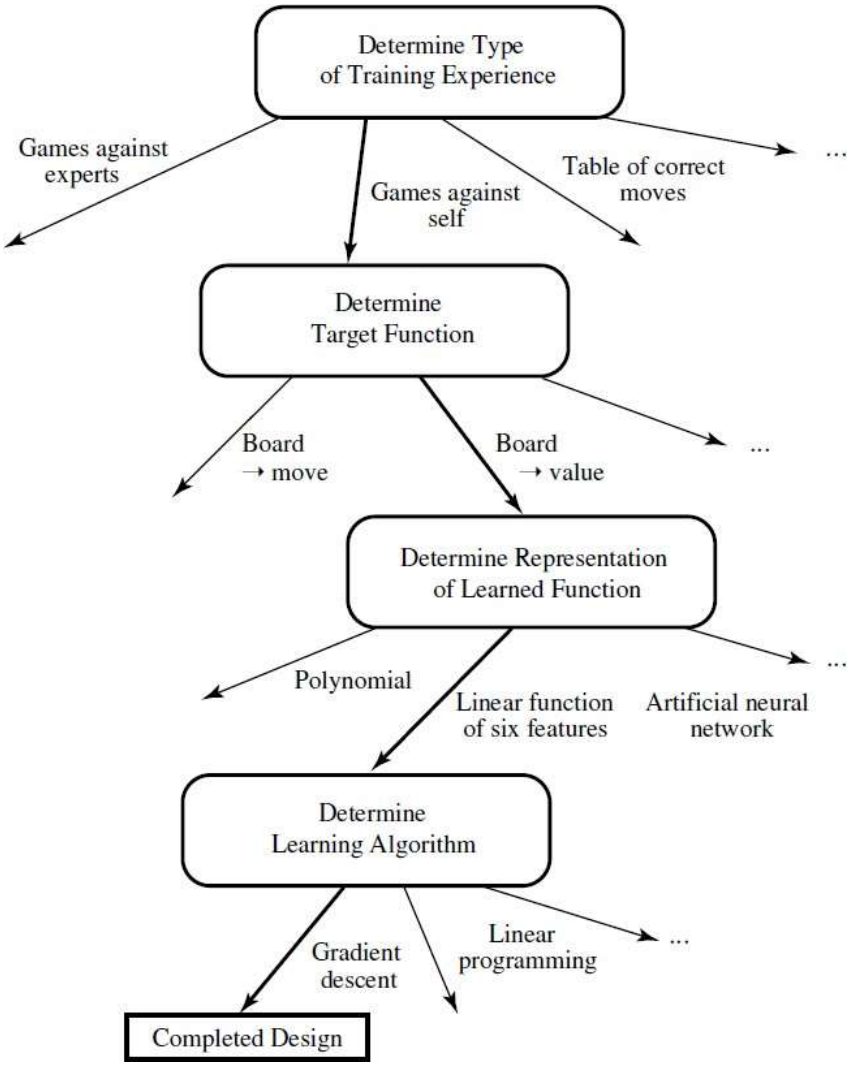
1. **The Performance System** is the module that must solve the given performance task by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

2. **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

3. **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.

4. **The Experiment Generator** takes as input the current hypothesis 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.

The sequence of design choices made for the checkers program is summarized in below figure



1.3 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 above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights ***wo*** 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

The field of machine learning, and much of this book, is concerned with answering questions such as the following

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, 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:

2.1 INTRODUCTION:

- Learning involves acquiring general concepts from specific training examples. Example: People continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc.
- Each such concept can be viewed as describing some subset of objects or events defined over a larger set
- Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set. (Example: A function defined over all animals, whose value is true for birds and false for other animals).

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

2.2 A CONCEPT LEARNING TASK:

To ground our discussion of concept learning, consider the example task of learning the target concept "days on which my friend Aldo enjoys his favorite water sport."

Table 2.1 describes a set of example days, each represented by a set of ***attributes.***

The attribute ***EnjoySport*** indicates whether or not Aldo enjoys his favorite 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.

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

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

Table 2.1 : Positive and negative training examples for the target concept ***EnjoySport.***

What hypothesis representation is provided to the learner?

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

For each attribute, the hypothesis will either

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

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

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

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

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

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

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

(Φ, Φ, Φ, Φ, Φ, Φ)

Notation

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

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

- The concept or function to be learned is called the ***target concept***, which is denoted by c.c can be any Boolean valued function defined over the instances X

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

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

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

its target ***concept value c(x).***

- ***D*** to denote the set of available training examples

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

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

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

- Given:
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values Sunny, Cloudy, and Rainy),
    - *AirTemp* (with values Warm and Cold),
    - *Humidity* (with values Normal and High),
    - *Wind* (with values Strong and Weak),
    - *Water* (with values Warm and Cool),
    - *Forecast* (with values Same and Change).

  - Hypotheses *H*: Each hypothesis is described by a conjunction of constraints on the attributes *Sky, AirTemp, Humidity, Wind, Water*, and *Forecast*. The constraints may be"?" (any value is acceptable), "Φ" (no value is acceptable), or a specific value.

  - Target concept *c*: ***EnjoySport*** : X → {0, l}
  - Training examples *D*: Positive and negative examples of the target function

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

Table 2.2 : The *Enjoy Sport* concept learning task.

The inductive learning hypothesis:

Notice that 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.

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 unobservedexamples.

2.3 CONCEPT LEARNING AS SEARCH:

- Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
- The goal of this search is to find the hypothesis that best fits the training examples.

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

3.2.2.2.2.2 = 96 distinct instances

5.4.4.4.4.4 = 5120 syntactically distinct hypotheses within H.

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

1 + (4.3.3.3.3.3) = 973. Semantically distinct hypotheses

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

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

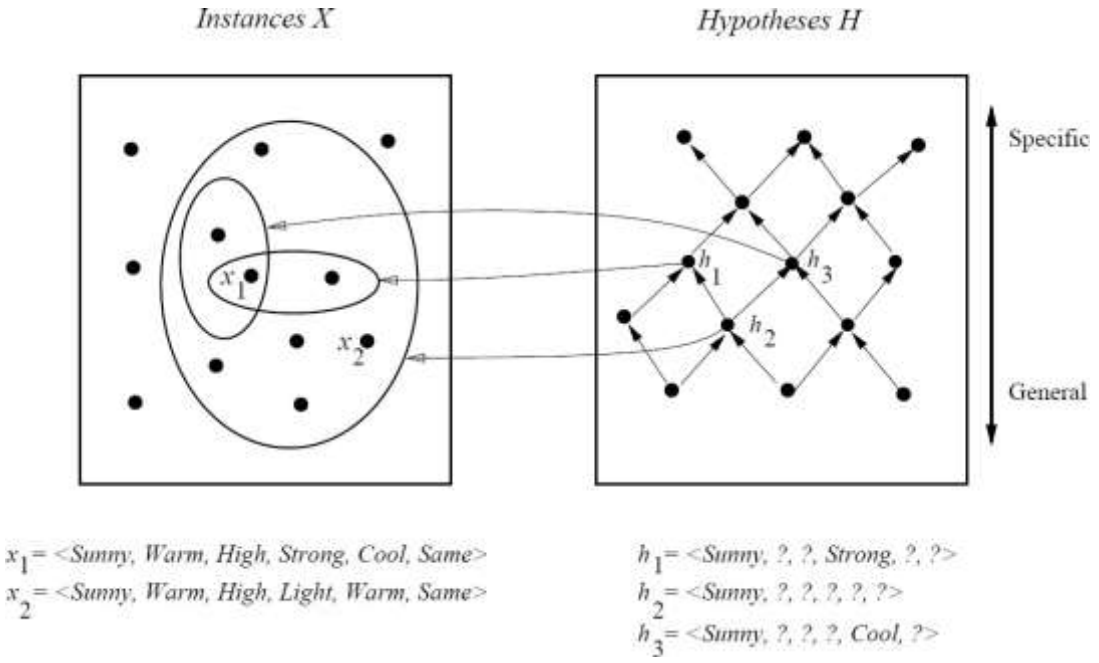
h_2 = (Sunny, ?, ?, ?, ?, ?)

- Consider the sets of instances that are classified positive by h_1 and by h_2.
- h_2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by h_1 will also be classified positive by h_2. Therefore, h_2 is more general than h_1.

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

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

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



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

2.4 FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS :

How can we use the *more-general-than* partial ordering to organize the search for a hypothesis consistent with the observed training examples?

One way is to begin with the most specific possible hypothesis in H, then generalize this hypothesis each time it fails to cover an observed positive training example.

(We say that a hypothesis "covers" a positive example if it correctly classifies the example as positive.)

To be more precise about how the partial ordering is used, consider the FIND-S algorithm defined in Table 2.3.

FIND-S Algorithm

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

FIND-S algorithm Table 2.3

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

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

- The first step of FIND-S is to initialize h to the most specific hypothesis in H

  h - (Ø, Ø, Ø, Ø, Ø, Ø)

- Consider the first training example

  x1 = <Sunny Warm Normal Strong Warm Same>, +

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

  h1 = <Sunny Warm Normal Strong Warm Same>

- Consider the second training example

  x2 = <Sunny, Warm, High, Strong, Warm, Same>, +

  The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example

  h2 = <Sunny Warm ? Strong Warm Same>

- Consider the third training example

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

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

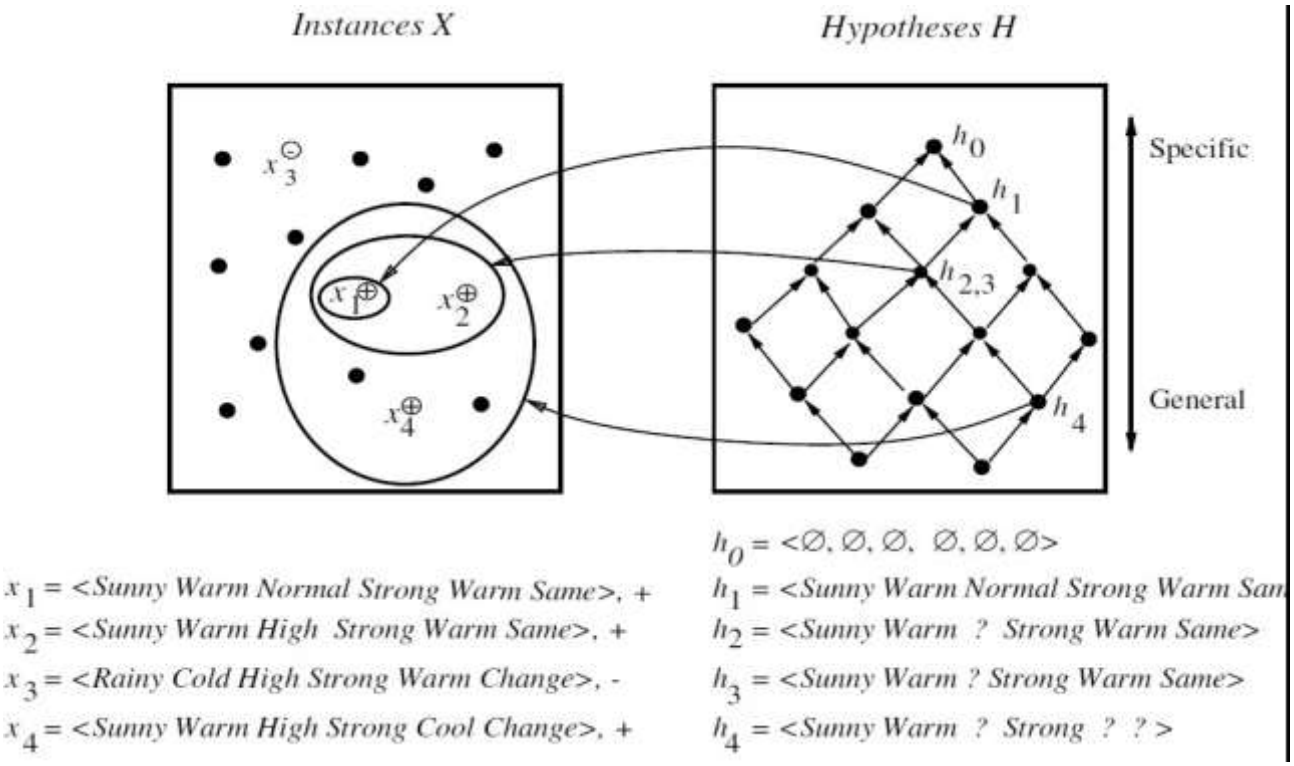
  h3 = < Sunny Warm ? Strong Warm Same>

- Consider the fourth training example

  x4 = <Sunny Warm High Strong Cool Change>, +

  The fourth example leads to a further generalization of h

  h4 = < Sunny Warm ? Strong ? ? >



The key property of the FIND-S algorithm

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

Unanswered by FIND-S

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

2.5 VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM :

- This section describes a second approach to concept learning, the CANDIDATEELIMINATION algorithm that addresses several of the limitations of FIND-S.
- Notice that 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.***
- Surprisingly, the CANDIDATE-ELIMINATION algorithm computes the description of this set without explicitly enumerating all of its members.
- This is accomplished by again using the ***more-general-than*** partial ordering, this time to maintain a compact representation of the set of consistent hypotheses and to incrementally refine this representation as each new training example is encountered.
- The CANDIDATE-ELIMINATION algorithm has been applied to problems such as learning regularities in chemical mass spectroscopy (Mitchell 1979) and learning control rules for heuristic search (Mitchell et al. 1983).
- Nevertheless, practical applications of the CANDIDATE-ELIMINATION, FNIN D-S algorithms are limited by the fact that they both perform poorly when given noisy training data.
- More importantly for our purposes here, the CANDIDATE-ELIMINATION algorithm provides a useful conceptual framework for introducing several fundamental issues in machine learning. In the remainder of this chapter we present the algorithm and discuss these issues. Beginning with the next chapter,
- we will examine learning algorithms that are used more frequently with noisy training data.
- The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all ***hypotheses consistent with the training examples***
- One limitation of the FIND-S algorithm is that it outputs just one hypothesis consistent with the training data – there might be many.
- To overcome this, introduce notion of version space and algorithms to compute it.

Representation

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

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

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

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

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

The LIST-THEN-ELIMINATION algorithm:

One obvious way to represent the version space is simply to list all of its members. This leads to a simple learning algorithm, which we might call the LIST-THEN - ELIMINATE algorithm, defined in Table 2.4.

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

---

1. ***VersionSpace c*** a list containing every hypothesis in H
2. For each training example, (x, c(x))
   remove from ***VersionSpace*** any hypothesis h for which h(x) ≠ c(x)
3. Output the list of hypotheses in ***VersionSpace***

Table 2.4 The LIST-THEN-ELIMINATE Algorithm

- The LIST-THEN-ELIMINATION 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 are 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.

- In principle, the LIST-THEN-ELIMINATE algorithm can be applied whenever the hypothesis space H is finite. It has many advantages, including the fact that it is guaranteed to output all hypotheses consistent with the training data. Unfortunately, it requires exhaustively enumerating all hypotheses in H-an unrealistic requirement for all but the most trivial hypothesis spaces.

A More Compact Representation for Version Spaces

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

A More Compact Representation for Version Spaces:

The CANDIDATE-ELIMINATE 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.

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.

To illustrate this representation for version spaces, consider again the *EnjoySport* concept learning problem described in Table 2.2. Recall that given the four training examples from Table 2.1, FIND-S outputs the hypothesis

$$h = \langle Sunny, Warm, ?, Strong, ?, ? \rangle$$

- In fact, this is just one of six different hypotheses from H that are consistent with these training examples. All six hypotheses are shown in Figure 2.3.
- They constitute the version space relative to this set of data and this hypothesis representation.

- The arrows among these six hypotheses in Figure 2.3 indicate instances of the *more-general~than* relation.
- The CANDIDATE-ELIMINATE algorithm represents the version space by storing only its most general members (labeled G in Figure 2.3) and its most specific (labeled *S* in the figure).

- 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. It is intuitively plausible that we can represent the version space in terms of its most specific and most general members.

- Below we define the boundary sets G and *S* precisely and prove that these sets do in fact represent the version space.

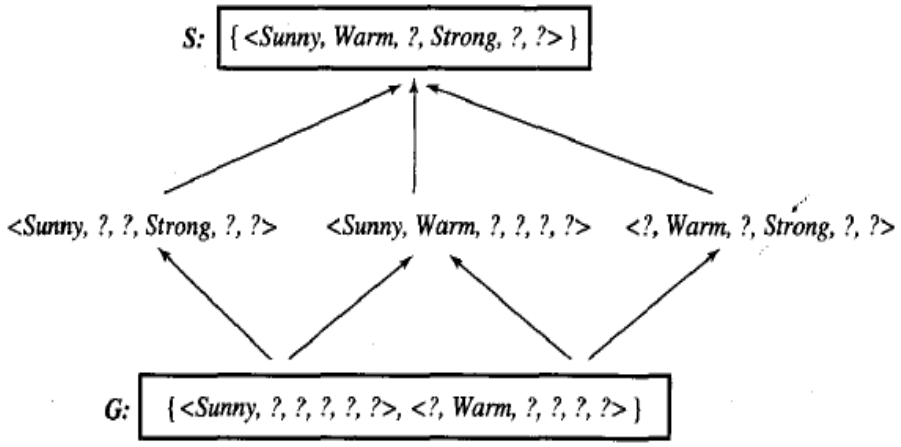


FIGURE 2.3
A version space with its general and specific boundary sets. The version space includes all six hypotheses shown here, but can be represented more simply by *S* and *G*. Arrows indicate instances of the *more_general_than* relation. This is the version space for the *EnjoySport* concept learning problem and training examples described in Table 2.1.

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.

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

Theorem: Version Space representation theorem

Theorem 2.1. 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 $c : X \rightarrow \{0, 1\}$ be an arbitrary target concept defined over X, and let D be an arbitrary set of training examples $\{\langle x, c(x)\rangle\}$. For all X, H, c, and D such that S and G are well defined,

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

Proof. To prove the theorem it suffices to show that (1) every h satisfying the right-hand side of the above expression is in $VS_{H,D}$ and (2) every member of $VS_{H,D}$ satisfies the right-hand side of the expression. To show (1) let g be an arbitrary member of G, s be an arbitrary member of S, and h be an arbitrary member of H, such that $g \geq_g h \geq_g s$. Then by the definition of S, s must be satisfied by all positive examples in D. Because $h \geq_g s$, h must also be satisfied by all positive examples in D. Similarly, by the definition of G, g cannot be satisfied by any negative example in D, and because $g \geq_g h$, h cannot be satisfied by any negative example in D. Because h is satisfied by all positive examples in D and by no negative examples in D, h is consistent with D, and therefore h is a member of $VS_{H,D}$. This proves step (1). The argument for (2) is a bit more complex. It can be proven by assuming some h in $VS_{H,D}$ that does not satisfy the right-hand side of the expression, then **showing that this leads to an inconsistency.**

CANDIDATE-ELIMINATION Learning Algorithm

The CANDIDATE-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

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

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

$$S_0 \leftarrow \{\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\rangle\}$$

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. This algorithm is summarize in Table 2.5.

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

Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

TABLE 2.5
CANDIDATE-ELIMINATION algorithm using version spaces. Notice the duality in how positive and negative examples influence S and G.

The detailed implementation of these operations will depend, of course, on the specific representations for instances and hypotheses. However, the algorithm itself can be applied to any concept learning task and hypothesis space for which these operations are well-defined.
In the following example trace of this algorithm, we see how such operations can be implemented for the representations used in the ***EnjoySport*** example problem.

An Illustrative Example

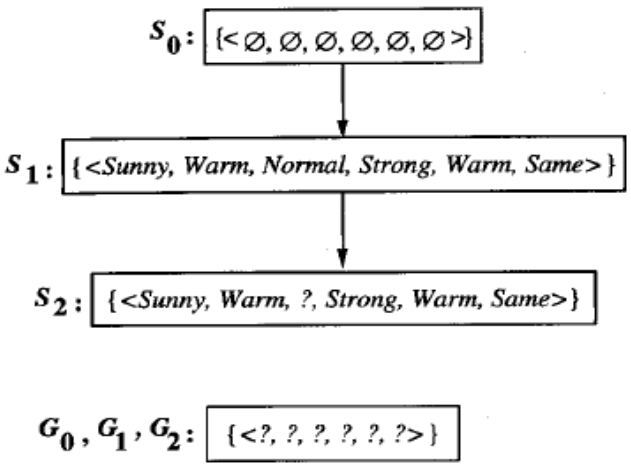
- Figure 2.4 traces the CANDIDATE-ELIMINATE algorithm applied to the first two training examples from Table 2.1. As described above, the boundary sets are first initialized to Go and So, the most general and most specific hypotheses in H, respectively.
- When the first training example is presented (a positive example in this case), the CANDIDATE ELIMINATE algorithm checks the S boundary and finds that it is overly specific-it fails to cover the positive example.
- The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example.

- This revised boundary is shown as S1 in Figure 2.4. No update of the G boundary is needed in response to this training example because Go correctly covers this example. When the second training example (also positive) is observed, it has a similar effect of generalizing ***S*** further to ***S2,*** leaving G again unchanged (i.e., G2 = **GI** = GO).

Notice the processing of these first two positive examples is very similar to the processing performed by the FIND-S algorithm.

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

Table 2.1



Training examples:

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

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

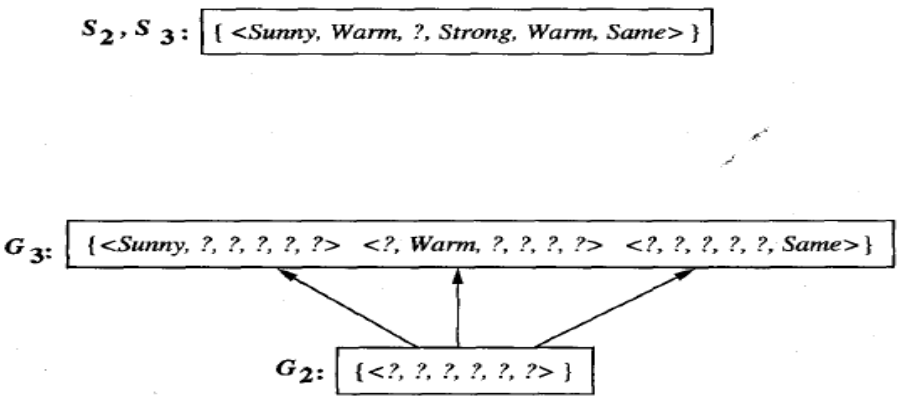
FIGURE 2.4
CANDIDATE-ELIMINATION Trace 1. S_0 and G_0 are the initial boundary sets corresponding to the most specific and most general hypotheses. Training examples 1 and 2 force the S boundary to become more general, as in the FIND-S algorithm. They have no effect on the G boundary.

- As illustrated by these first two steps, positive training examples may force the **S** boundary of the version space to become increasingly general. Negative training examples play the complimentary role of forcing the **G** boundary to become increasingly specific.

- Consider the third training example, shown in Figure 2.5. This negative example reveals that the **G** boundary of the version space is overly general; that is, the hypothesis in **G** incorrectly predicts that this new example is a positive example.

- The hypothesis in the **G** boundary must therefore be specialized until it correctly classifies this new negative example.

- As shown in Figure *2.5,* there are several alternative minimally more specific hypotheses. All of these become members of the new **G3** boundary set.



Training Example:

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

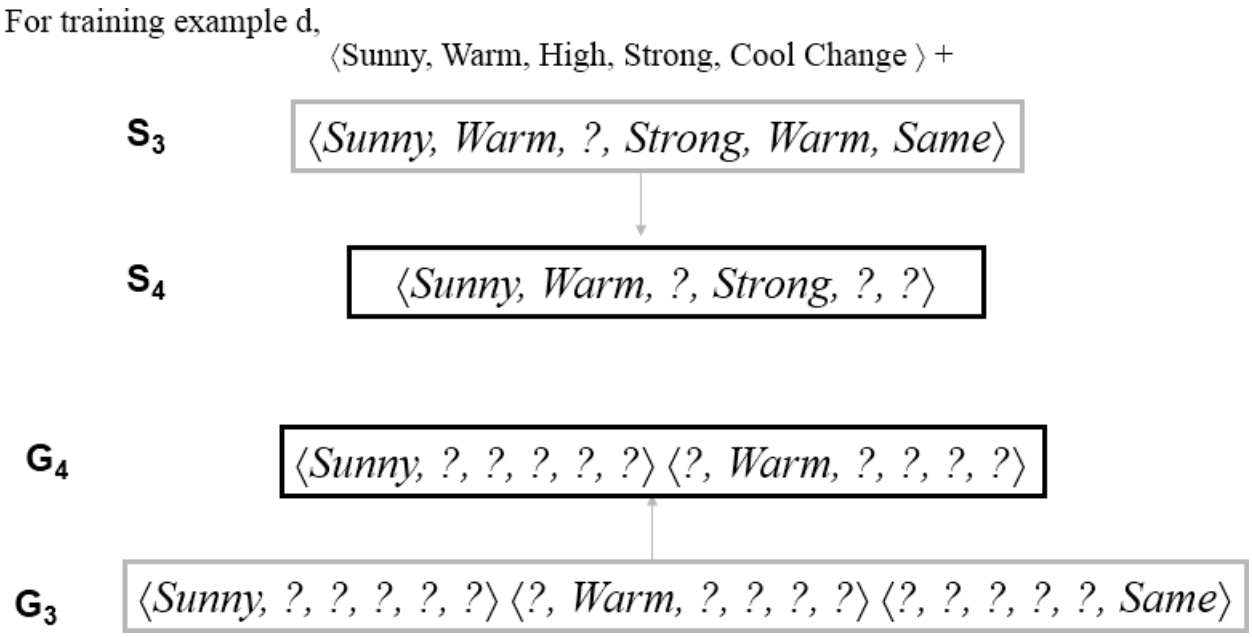
FIGURE 2.5
CANDIDATE-ELIMINATION Trace 2. Training example 3 is a negative example that forces the G_2 boundary to be specialized to G_3. Note several alternative maximally general hypotheses are included in G_3.

- When the second training example is observed, it has a similar effect of generalizing S further to S2, leaving G again unchanged i.e., G2 = G1 = G0
- Consider the third training example. This negative example reveals that the G boundaryof the version space is overly general, that is, the hypothesis in G incorrectly predicts that this new example is a positive example.
- The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example

Given that there are six attributes that could be specified to specialize G2, why are there only three new hypotheses in G3?

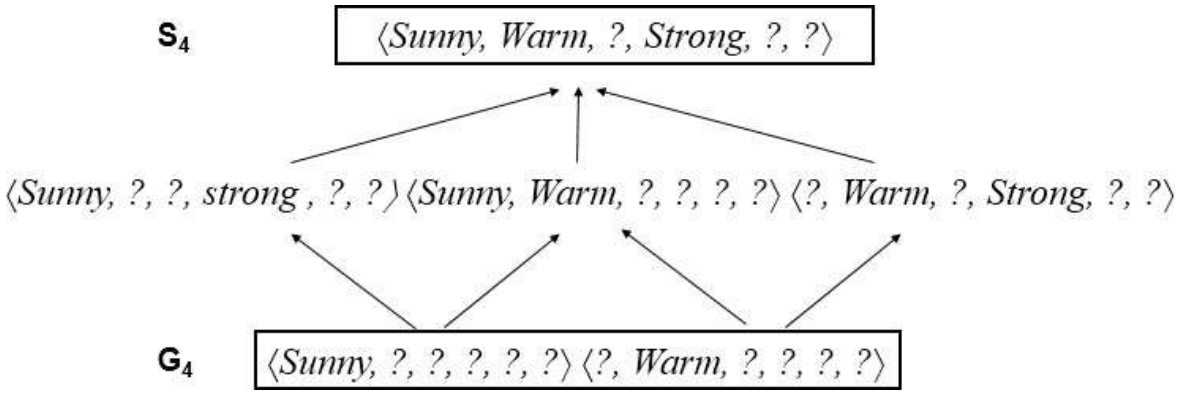
For example, the hypothesis h = (?, ?, Normal, ?, ?, ?) is a minimal specialization of G2 that correctly labels the new example as a negative example, but it is not included in G3. The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples

Consider the fourth training example.



- This positive example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example

After processing these four examples, the boundary sets S4 and G4 delimit the version spaceof all hypotheses consistent with the set of incrementally observed training examples.



The final version space

2.6 REMARKS ON VERSION SPACES AND CANDIDATE-ELIMINATION

Will the CANDIDATE-ELIMINATION Algorithm Converge to the Correct Hypothesis?

The version space learned by the CANDIDATE-ELIMINATION algorithm will converge toward the hypothesis that correctly describes the target concept, provided

1. There are no errors in the training examples
2. There is some hypothesis in H that correctly describes the target concept. In fact, as new training examples are observed, the version space can be monitored to determine the remaining ambiguity regarding the true concept and to determine when sufficient training examples have been observed to unambiguously identify the target concept.

The target concept is exactly learned when the S and G boundary sets converge to a single identical, hypothesis.

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

  In such cases algorithm may converge to empty version space
- If algorithm can request next training example (e.g. from teacher) can increase speed of convergence by requesting examples that split the version space
  - E.g. T5: $\langle Sunny, Warm, Normal, Light, Warm, Same \rangle$ satisfies 3 hypotheses in previous example
    - If T5 positive, S generalised, 3 hypotheses eliminated
    - If T5 negative, G specialised, 3 hypotheses eliminated
  - Optimal query strategy is to request examples that exactly split version space – converge in $\lceil log_2 |VS| \rceil$ steps. However, this is not always possible.

- When using (i.e **not** training) a classifier that has not completely converged, new examples may be
  1. classed as positive by all $h \in VS$
  2. classed as negative by all $h \in VS$
  3. classed as positive by some, and negative by other, $h \in VS$

  Cases 1 and 2 are unproblematic. In case 3. may want to consider proportion of positive vs. negative classifications (but then *a priori* probabilities of hypotheses are relevant)

What will happen if the training data contains errors?

- Suppose, for example, that the second training example above is incorrectly presented as anegative example instead of a positive example.
- Unfortunately, in this case the algorithm is certain to remove the correct target concept from the version space.
- Because it will remove every hypothesis that is inconsistent with each training example, it will eliminate the true target concept from the version space as soon as this false negative example is encountered.
- Of course, given sufficient additional training data, the learner will eventually detect an inconsistency by noticing that the S and G boundary sets eventually converge to an empty version space.
- Such an empty version space indicates that there is no hypothesis in H consistent with all observed training examples.
- A similar symptom will appear when the training examples are correct, but the target concept cannot be described in the hypothesis representation.
- For now, we consider only the case in which the training examples are correct and the true target concept is present in the hypothesis space.

INDUCTIVE BIAS

The fundamental questions for inductive inference

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

Inductive Bias

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

  What if no concept in H that describes the target concept?

- Consider the training data

Example	Sky	Temp	Humid	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Cloudy	Warm	Normal	Strong	Warm	Same	Yes
3	Rainy	Warm	Normal	Strong	Warm	Same	No

- No hypotheses consistent with 3 examples.

  Most specific hypothesis consistent with Ex 1 and 2 *and representable in H*:

  $$\langle ?, Warm, Normal, Strong, Warm, Same \rangle$$

  But this is inconsistent with Ex 3.

- Need more expressive hypothesis representation language.

  E.g. allow disjunctive or negative attribute values:

$$Sky = Sunny \vee Cloudy$$
$$Sky \neq Rainy$$

2.7.1 A Biased Hypothesis Space

Suppose we wish to assure that the hypothesis space contains the unknown target concept. The obvious solution is to enrich the hypothesis space to include *every possible* hypothesis. To illustrate, consider again the *EnjoySport* example in which 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*." In fact, given the following three training examples of this disjunctive hypothesis, our algorithm would find that there are zero hypotheses in the version 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 : ⟨?, *Warm*, *Normal*, *Strong*, *Cool*, *Change*⟩

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. In this case we require a more expressive hypothesis space.

An Unbiased Learner

- The solution to the problem of assuring that the target concept is in the hypothesis space H is to provide a hypothesis space capable of representing every teachable concept that is representing every possible subset of the instances X.
- The set of all subsets of a set X is called the power set of X

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

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

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

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

To see why, suppose we present three positive examples **(xl,x 2, x3)** and two negative examples **(x4,** x5) to the learner.

At this point, the ***S*** boundary of the version space will contain the hypothesis which is just the disjunction of the positive examples

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

because this is the most specific possible hypothesis that covers these three examples. Similarly, the G boundary will consist of the hypothesis that rules out only the observed negative examples

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

The problem here is that with this very expressive hypothesis representation, the S boundary will always be simply the disjunction of the observed positive examples, while the G boundary will always be the negated disjunction of the observed negative examples. Therefore, the only examples that will be unambiguously classified by S and G are the observed training examples themselves. In order to converge to a single, final target concept, we will have to present every single instance in X as a training example!

To see the reason, note that when H is the power set of X and **x** is some previously unobserved instance,

then for any hypothesis h in the version space that covers **x,** there will be another hypothesis *h'* in the power set that is identical to h except for its classification of x. And of course if *h* is in the version space, then h' will be as well, because it agrees with *h* on all the observed training examples.

The Futility of Bias-Free Learning

The Futility of Bias-Free Learning

- **Fundamental Property of Inductive Learning**
  - A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.
  - In EnjoySport, there are implicit assumption that the target concept could be represented by a conjunction of attribute values.
- **We constantly have recourse to inductive biases**
  - *Example:* we all know that the sun will rise tomorrow. Although we cannot *deduce* that it will do so based on the fact that it rose today, yesterday, the day before, etc., we do take this leap of faith or use this inductive bias, naturally!

The Futility of Bias-Free Learning

- **Notation:**
  - A |- B indicates that B follows deductively from A.
  - A $\succ$ B indicates that B follows inductively from A.
- **The inductive inference step:**

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

Because L is an inductive learning algorithm, the result $L(x_i, D_c)$ that it infers will not in general be provably correct; that is, the classification $L(x_i, D_c)$ need not follow deductively from the training data D_c and the description of the new instance x_i. However, it is interesting to ask what additional assumptions could be added to $D_c \wedge x_i$ so that $L(x_i, D_c)$ would follow deductively. We define the inductive bias of L as this set of additional assumptions. More precisely, we define the

inductive bias of L to be the set of assumptions B such that for all new instances x_i

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

where the notation $y \vdash z$ indicates that z follows deductively from y (i.e., that z is provable from y). Thus, we define the inductive bias of a learner as the set of additional assumptions B sufficient to justify its inductive inferences as deductive inferences. To summarize,

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

What, then, is the inductive bias of the CANDIDATE-ELIMINAION Algorithm ?

To answer this, let us specify ***L(xi, D,)*** exactly for this algorithm: given a set of data ***D,,*** the CANDIDATE-ELIMINATION Algorithm will first compute the version space ***VSH,D, ,t***hen classify the new instance ***xi*** by a vote among hypotheses in this version space. Here let us assume that it will output a classification for ***xi*** only if

this vote among version space hypotheses is unanimously positive or negative and that it will not output a classification otherwise. Given this definition of ***L(xi,*** D,) for the CANDIDATE-ELIMINATION Algorithm, what is its inductive bias? It is simply the assumption ***c*** E H. Given this

assumption, each inductive inference performed by the CANDIDATE-ELIMINATION Algorithm can bejustified deductively.

Inductive bias of CANDIDATE-ELIMINATION Algorithm The target concept c is contained in the given hypothesis space H. Figure 2.8 summarizes the situation schematically. The inductive CANDIDATE-ELIMINATION Algorithm at the top of the figure takes two inputs: the training examples and a new instance to be classified. At the bottom of the figure, a deductive theorem prover is given these same two inputs plus the assertion "H contains the target concept." These two systems will in principle produce identical outputs for every possible input set of training examples and every possible new instance in X.

One advantage of viewing inductive inference systems in terms of their inductive bias is that:

- It provides a nonprocedural means of characterizing their policy for generalizing beyond the observed data.
- A second advantage is that it allows comparison of different learners according to the strength of the inductive bias they employ. Consider, for example, the following three learning algorithms, which are listed from weakest to strongest bias.

1. ROTE-LEARNERL: Learning corresponds simply to storing each observed training example in memory. Subsequent instances are classified by looking them up in memory. If the instance is found in memory, the stored classification is returned. Otherwise, the system refuses to classify the new instance.

2. **CANDIDATE-ELIMINATION Algorithm** : New instances are classified only in the case where all members of the current version space agree on the classification. Otherwise, the system refuses to classify the new instance.

3. FIND-S: This algorithm, described earlier, finds the most specific hypothesis consistent with the training examples. It then uses this hypothesis to classify all subsequent instances.

DECISION TREE LEARNING:

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.

DECISION TREE REPRESENTATION

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

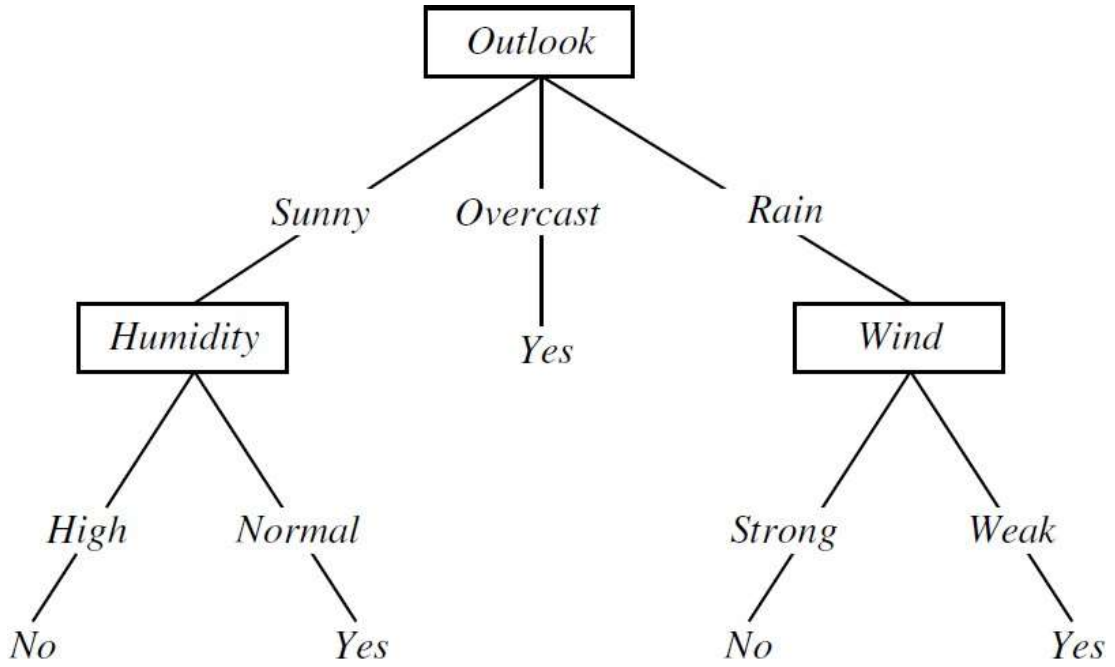


FIGURE: A decision tree for the concept ***PlayTennis***. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf

- Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions

For example, the decision tree shown in above figure corresponds to the expression

(Outlook = Sunny A Humidity = Normal)
V (Outlook = Overcast)
V (Outlook = Rain A Wind = Weak)

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

Decision tree learning is generally best suited to problems with the following characteristics:

1. ***Instances are represented by attribute-value pairs*** – Instances are described by afixed set of attributes and their values

2. ***The target function has discrete output values*** – The decision tree assigns a Boolean classification (e.g., yes or no) to each example. Decision tree methods easily extend to learning functions with more than two possible output values.

3. ***Disjunctive descriptions may be required***

4. ***The training data may contain errors*** – Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.

5. ***The training data may contain missing attribute values –*** Decision tree methods canbe used even when some training examples have unknown values

THE BASIC DECISION TREE LEARNING ALGORITHM

The basic algorithm is ID3 which learns decision trees by constructing them top-down

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*, as defined in Equation (3.4).

TABLE 3.1
Summary of the ID3 algorithm specialized to learning boolean-valued functions. ID3 is a greedy algorithm that grows the tree top-down, at each node selecting the attribute that best classifies the local training examples. This process continues until the tree perfectly classifies the training examples, or until all attributes have been used.

* The best attribute is the one with highest information gain

TABLE: Summary of the ID3 algorithm specialized to learning Boolean-valued functions. ID3is a greedy algorithm that grows the tree top-down, at each node selecting the attribute that best classifies the local training examples. This process continues until the tree perfectly classifies the training examples, or until all attributes have been used

Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- A statistical property called ***information gain*** that measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses ***information gain*** measure to select among the candidate attributes at each step while growing the tree.

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

To define information gain, we begin by defining a measure called entropy. *Entropy measures the impurity of a collection of examples.*

Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this 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

p_- is the proportion of negative examples in S.

Example:

Suppose S is a collection of 14 examples of some boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this boolean classification is

$$Entropy([9+, 5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$$
$$= 0.940$$

- The entropy is 0 if all members of S belong to the same class
- The entropy is 1 when the collection contains an equal number of positive and negative examples
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1

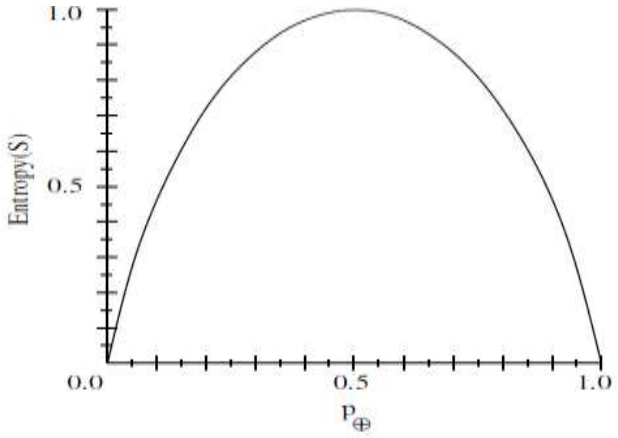


FIGURE The entropy function relative to a boolean classification, as the proportion, $p_{\oplus}$, of positive examples varies between 0 and 1.

INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY

- ***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examplesS, is defined as

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

For example, suppose S is a collection of training-example days described by attributes including *Wind*, which can have the values *Weak* or *Strong*. As before, assume S is a collection containing 14 examples, [9+, 5−]. Of these 14 examples, suppose 6 of the positive and 2 of the negative examples have $Wind = Weak$, and the remainder have $Wind = Strong$. The information gain due to sorting the original 14 examples by the attribute *Wind* may then be calculated as

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

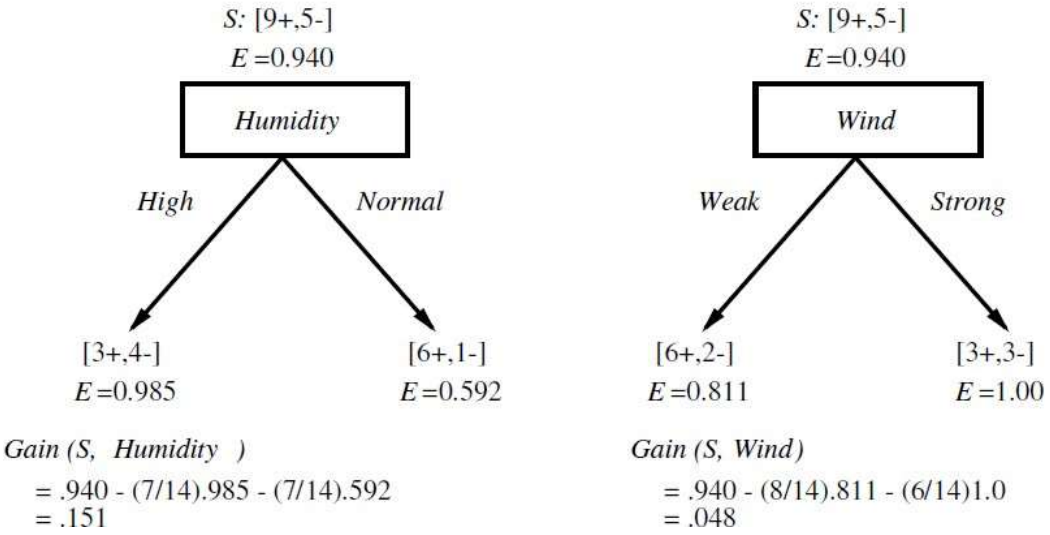
An Illustrative Example

- To illustrate the operation of ID3, consider the learning task represented by the training examples of below table.
- Here the target attribute ***PlayTennis***, which can have values ***yes*** or ***no*** for different days.
- Consider the first step through the algorithm, in which the topmost node of the decision tree is created.

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

- ID3 determines the information gain for each candidate attribute (i.e., Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain.

Which attribute is the best classifier?



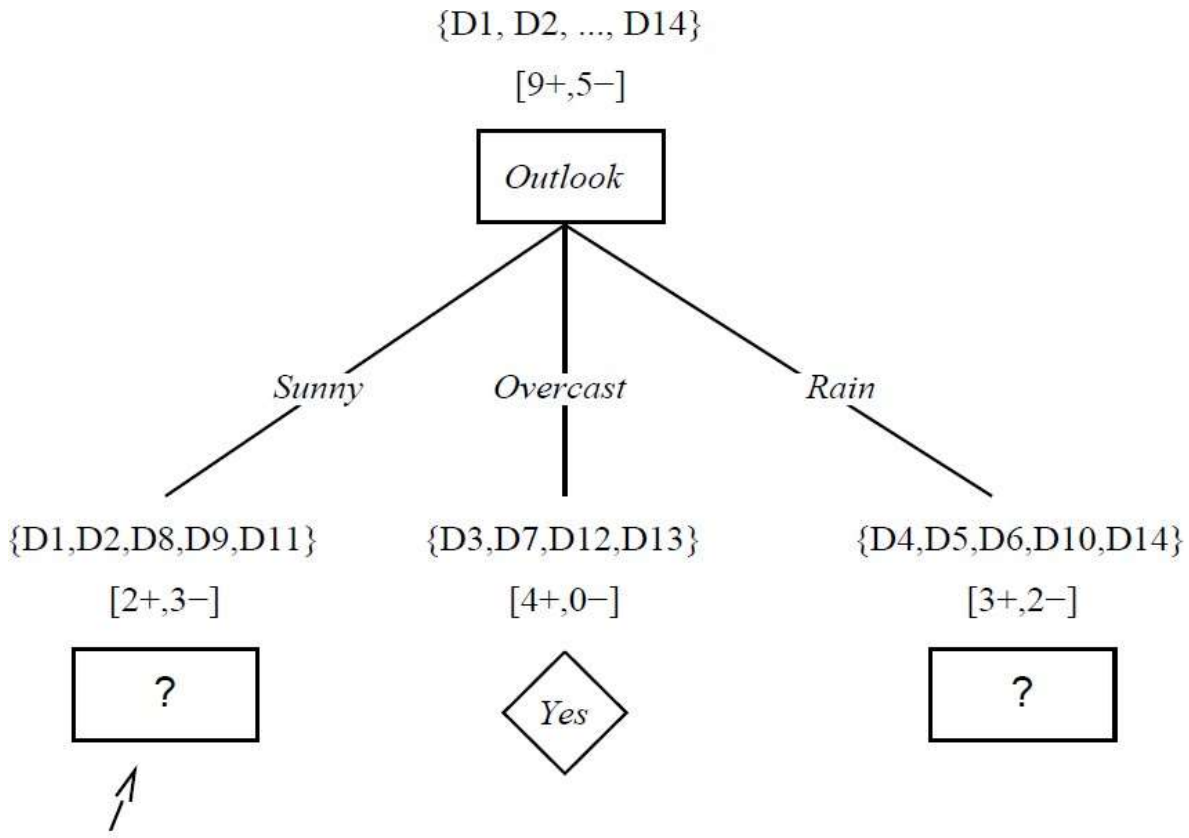
Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)
= .940 - (8/14).811 - (6/14)1.0
= .048

- The information gain values for all four attributes are

Gain(S, Outlook)	= 0.246
Gain(S, Humidity)	= 0.151
Gain(S, Wind)	= 0.048
Gain(S, Temperature)	= 0.029

- According to the information gain measure, the ***Outlook*** attribute provides the best prediction of the target attribute, ***PlayTennis***, over the training examples. Therefore, ***Outlook*** is selected as the decision attribute for the root node, and branches are created below the root for each of its possible values i.e., Sunny, Overcast, and Rain.



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

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

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

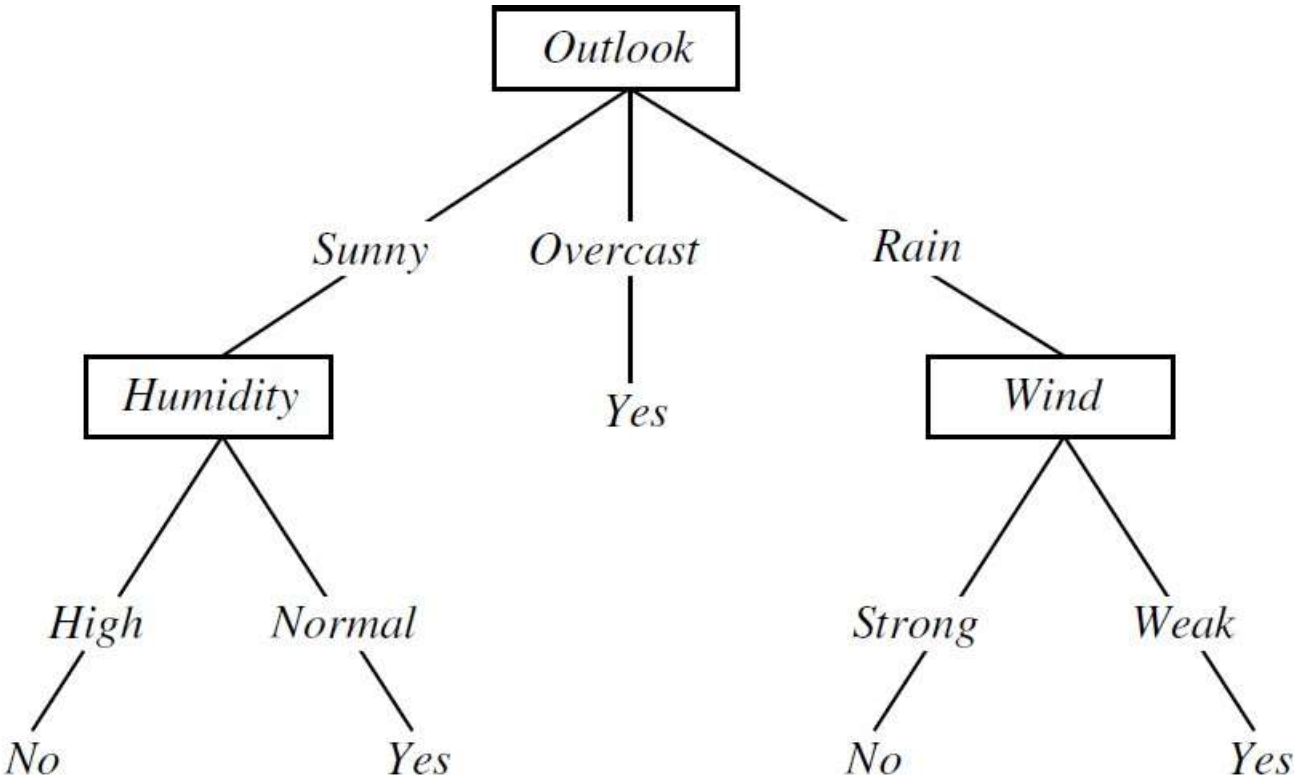
$Gain\ (S_{sunny}, Wind) = .970 - (2/5)\ 1.0 - (3/5)\ .918 = .019$

SRain = { D4, D5, D6, D10, D14}

Gain (SRain , Humidity) = 0.970 – (2/5)1.0 – (3/5)0.917 = 0.019

Gain (SRain , Temperature) =0.970 – (0/5)0.0 – (3/5)0.918 – (2/5)1.0 = 0.019

Gain (SRain , Wind) =0.970 – (3/5)0.0 – (2/5)0.0 = 0.970



HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING:

- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples.
- The hypothesis space searched by ID3 is the set of possible decision trees.
- ID3 performs a simple-to complex, hill-climbing search through this hypothesis space, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data

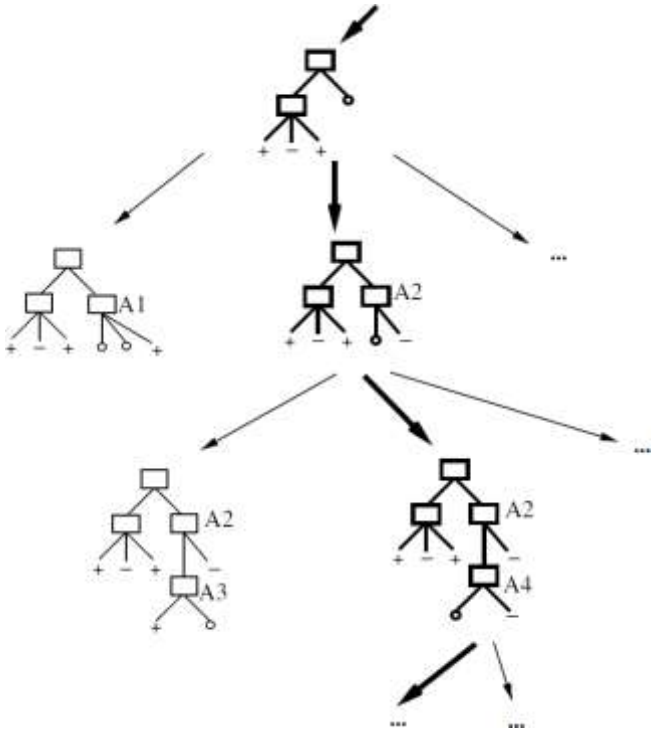


Figure: Hypothesis space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the information gain heuristic.

By viewing ID3 in terms of its search space and search strategy, there are some insight into its capabilities and limitations

1. *ID3's hypothesis space of all decision trees is a complete space of finite discrete-valued functions, relative to the available attributes. Because every finite discrete-valued function can be represented by some decision tree*
   ID3 avoids one of the major risks of methods that search incomplete hypothesis spaces: that the hypothesis space might not contain the target function.

2. *ID3 maintains only a single current hypothesis as it searches through the space of decision trees.*
   For example, with the earlier version space candidate elimination method, which maintains the set of all hypotheses consistent with the available training examples.

   By determining only a single hypothesis, ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses.
   For example, it does not have the ability to determine how many alternative decision trees are consistent with the available training data, or to pose new instance queries thatoptimally resolve among these competing hypotheses

3. *ID3 in its pure form performs no backtracking in its search. Once it selects an attributeto test at a particular level in the tree, it never backtracks to reconsider this choice*.
   In the case of ID3, a locally optimal solution corresponds to the decision tree it selects along the single search path it explores. However, this locally optimal solution may be less desirable than trees that would have been encountered along a different branch of the search.

4. *ID3 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis*.
   One advantage of using statistical properties of all the examples is that the resultingsearch is much less sensitive to errors in individual training examples.
   ID3 can be easily extended to handle noisy training data by modifying its termination criterion to accept hypotheses that imperfectly fit the training data.

Introduction to Machine Learning (Chapter 1)

1. Define Machine Learning. Discuss with examples why machine learning is important.
2. Discuss with examples some useful applications of machine learning.
3. Explain how some areas/disciplines that influenced the machine learning.
4. What do you mean by a well–posed learning problem? Explain the important features that are required towell–define a learning problem.
5. Define learning program for a given problem. Describe the following problems with respect to Tasks,
Performance and Experience: a. Checkers Learning Problems
   b. Handwritten Recognition Problem
   c. Robot Driving Learning Problem

6. Describe in detail all the steps involved in designing a learning system.
7. Discuss the perspective and issues in machine learning.

Concept Learning (Chapter 2)

8. Define Concept and Concept Learning. With example explain how the Concept Learning task determines theHypothesis for given target concept.

9. Discuss Concept learning as search with respect to General to specific ordering of hypothesis.
10. Describe Find S Algorithm. What are the properties and complaints of Find S.
11. Illustrate Find S Algorithm over *EnjoySport* concept. Training instances given below.

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

12. Define Consistent *Hypothesis* and *Version Space*. With example explain Version Space and Representation ofversion Space.
13. Describe List the Eliminate Algorithm.
14. Explain the candidate elimination algorithm.

15. Trace Candidate-Elimination algorithm on the following data.

a)

Origin	*Manufacturer*	*Color*	*Decade*	*Type*	*Example Type*
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive

b)

Origin	*Manufacturer*	*Color*	*Decade*	*Type*	*Example Type*
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

c)

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

16. Explain the inductive biased hypothesis space, unbiased learner and the futility of Bias Free Learning. Describethe three types of learner.
17. What is the role of a function approximation algorithm? How does learner system estimate training values andadjusts weights while learning?
18. Describe in brief: Version spaces and Candidate –Elimination Algorithm.
19. Define Inductive Learning Hypothesis.
20. Describe Inductive Systems and Equivalent Deductive Systems

Rank the following three types of leaners according to their biases:

a. Rote Learner
b. Candidate Elimination Learner
c. Find S Learner.

Unit-2

Evaluating Hypothesis:

- Motivation, Estimating hypothesis accuracy,
- Basics of sampling theorem,
- General approach for deriving confidence intervals,
- Difference in error of two hypothesis,
- Comparing learning algorithms.

INTRODUCTION:

- Evaluating the accuracy of hypotheses is fundamental to machine learning.
- This chapter presents an introduction to statistical methods for estimating hypothesis accuracy, focusing on three questions. First, given the observed accuracy of a hypothesis over a limited sample of data, how well does this estimate its accuracy over additional examples?
- Second, given that one hypothesis outperforms another over some sample of data, how probable is it that this hypothesis is more accurate in general?
- Third, when data is limited what is the best way to use this data to both learn a hypothesis and estimate its accuracy? Because limited samples of data might misrepresent the general distribution of data, estimating true accuracy from such samples can be misleading.
- Statistical methods, together with assumptions about the underlying distributions of data, allow one to bound the difference between observed accuracy over the sample of available data and the true accuracy over the entire distribution of data.

Motivation, Estimating hypothesis accuracy:

5.1 MOTIVATION

- In many cases it is important to evaluate the performance of learned hypotheses as precisely as possible. One reason is simply to understand whether to use the hypothesis.
- For instance, when learning from a limited-size database indicating the effectiveness of different medical treatments, it is important to understand as precisely as possible the accuracy of the learned hypotheses.
- A second reason is that evaluating hypotheses is an integral component of many learning methods. For example, in post-pruning decision trees to avoid overfitting, we must evaluate the impact of possible pruning steps on the accuracy of the resulting decision tree.
- Therefore it is important to understand the likely errors inherent in estimating the accuracy of the pruned and un pruned tree.
- Estimating the accuracy of a hypothesis is relatively straightforward when data is plentiful.

However, when we must learn a hypothesis and estimate its future accuracy given only a limited set of data, two key difficulties arise:

- ***Bias in the estimate.*** First, the observed accuracy of the learned hypothesis over the training examples is often a poor estimator of its accuracy over future examples. Because

the learned hypothesis was derived from these examples, they will typically provide an optimistically biased estimate of hypothesis accuracy over future examples. This is especially likely when the learner considers a very rich hypothesis space, enabling it to overfit the training examples. To obtain an unbiased estimate of future accuracy, we typically test the hypothesis on some set of test examples chosen independently of the training examples and the hypothesis.

- ***Variance in the estimate.*** Second, even if the hypothesis accuracy is measured over an unbiased set of test examples independent of the training examples, the measured accuracy can still vary from the true accuracy, depending on the makeup of the particular set of test examples. The smaller the set of test examples, the greater the expected variance.

5.2 ESTIMATING HYPOTHESIS ACCURACY:

- When evaluating a learned hypothesis we are most often interested in estimating the accuracy with which it will classify future instances. At the same time, we would like to know the probable error in this accuracy estimate (i.e., what error bars to associate with this estimate).

- Throughout this chapter we consider the following setting for the learning problem.

- There is some space of possible instances X (e.g., the set of all people) over which various target functions may be defined (e.g., people who plan to purchase new skis this year). We assume that different instances in X may be encountered with different frequencies.
- A convenient way to model this is to assume there is some unknown probability distribution ***D*** that defines the probability of encountering each instance in X (e-g., *23* might assign a higher probability to encountering 19-year-old people than 109-year-old people).
- Notice *23* says nothing about whether x is a positive or negative example; it only detennines the probability that ***x*** will be encountered.
- The learning task is to learn the target concept or target function f by considering a space H of possible hypotheses.
- Training examples of the target function f are provided to the learner by a trainer who draws each instance independently, according to the distribution D, and who then forwards the instance x along with its correct target value f ***(x)*** to the learner.

The target function f : X + **{0,1)** classifies each person according to whether or not they plan to purchase skis this year.

Within this general setting we are interested in the following two questions:

1. Given a hypothesis h and a data sample containing n examples drawn at random according to the distribution D, what is the best estimate of the accuracy of h over future instances drawn from the same distribution?
2. What is the probable error in this accuracy estimate?

Sample Error and True Error:

To answer these questions, we need to distinguish carefully between two notions of accuracy or, equivalently, error. One is the error rate of the hypothesis over the sample of data that is available. The other is the error rate of the hypothesis over the entire unknown distribution $\mathcal{D}$ of examples. We will call these the *sample error* and the *true error* respectively.

The *sample error* of a hypothesis with respect to some sample S of instances drawn from X is the fraction of S that it misclassifies:

Definition: The **sample error** (denoted $error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where n is the number of examples in S, and the quantity $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

The *true error* of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution $\mathcal{D}$.

Definition: The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target function f and distribution $\mathcal{D}$, is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

Here the notation $\Pr_{x \in \mathcal{D}}$ denotes that the probability is taken over the instance distribution $\mathcal{D}$.

What we usually wish to know is the true error $error_{\mathcal{D}}(h)$ of the hypothesis, because this is the error we can expect when applying the hypothesis to future examples. All we can measure, however, is the sample error $error_S(h)$ of the hypothesis for the data sample S that we happen to have in hand. The main question considered in this section is "How good an estimate of $error_{\mathcal{D}}(h)$ is provided by $error_S(h)$?"

Confidence Intervals for Discrete-Valued Hypotheses:

Here we give an answer to the question "How good an estimate of $error_{\mathcal{D}}(h)$ is provided by $error_S(h)$?" for the case in which h is a discrete-valued hypothesis. More specifically, suppose we wish to estimate the true error for some discrete-valued hypothesis h, based on its observed sample error over a sample S, where

- the sample S contains n examples drawn independent of one another, and independent of h, according to the probability distribution $\mathcal{D}$
- $n \geq 30$
- hypothesis h commits r errors over these n examples (i.e., $error_S(h) = r/n$).

Under these conditions, statistical theory allows us to make the following assertions:

1. Given no other information, the most probable value of $error_{\mathcal{D}}(h)$ is $error_S(h)$
2. With approximately 95% probability, the true error $error_{\mathcal{D}}(h)$ lies in the interval

$$error_S(h) \pm 1.96\sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

- To illustrate, suppose the data sample ***S*** contains ***n*** = ***40*** examples and that hypothesis ***h*** commits ***r*** = ***12*** errors over this data.
- In this case, the sample error ***errors(h)*** = ***12/40*** = ***.30.***
- Given no other information, the best estimate of the true error ***errorD(h)*** is the observed sample error ***.30.***
- However, we do not expect this to be a perfect estimate of the true error.
- If we were to collect a second sample S' containing ***40*** new randomly drawn examples, we might expect the sample error ***errors,(h)*** to vary slightly from the sample error ***errors(h).***

- We expect a difference due to the random differences in the makeup of S and S'.
- In fact, if we repeated this experiment over and over, each time drawing a new sample *S,* containing ***40*** new examples, we would find that for approximately ***95%*** of these experiments, the calculated interval would contain the true error. For this reason, we call this interval the ***95%*** confidence interval estimate for ***errorv(h).***

In the current example, where ***r*** = ***12*** and ***n*** = ***40,*** the ***95%*** confidence interval is, according to the above expression, ***0.30*** f ***(1.96*** - ***.07)*** = ***0.30*** f .14.

Confidence level $N\%$:	50%	68%	80%	90%	95%	98%	99%
Constant z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

TABLE 5.1
Values of z_N for two-sided $N\%$ confidence intervals.

The above expression for the 95% confidence interval can be generalized to any desired confidence level. The constant 1.96 is used in case we desire a 95% confidence interval. A different constant, z_N, is used to calculate the $N\%$ confidence interval. The general expression for approximate $N\%$ confidence intervals for $error_{\mathcal{D}}(h)$ is

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}} \qquad (5.1)$$

- where the constant *ZN* is chosen depending on the desired confidence level, using the values of *z~* given in Table ***5.1.***

- Thus, just as we could calculate the ***95%*** confidence interval for ***errorv(h)*** to be ***0.305 (1.96. .07)*** (when ***r*** = ***12, n*** = ***40),*** we can calculate the ***68%*** confidence interval in this case to be ***0.30 f (1.0*** - ***.07).***

- Note it makes intuitive sense that the ***68%*** confidence interval is smaller than the ***95%*** confidence interval, because we have reduced the probability with which we demand that ***errorv(h)*** fall into the interval.

- Equation ***(5.1)*** describes how to calculate the confidence intervals, or error bars, for estimates of ***errorv(h)*** that are based on ***errors(h).***

- In using this expression, it is important to keep in mind that this applies only to discrete-valued
- hypotheses, that it assumes the sample ***S*** is drawn at random using the same distribution from which future data will be drawn, and that it assumes the data is independent of the hypothesis being tested.

- We should also keep in mind that the expression provides only an approximate confidence interval, though the approximation is quite good when the sample contains at least ***30*** examples, and ***errors(h)*** is not too close to ***0*** or ***1. A*** more accurate rule of thumb is that the above approximation works well when

$$n\, error_S(h)(1 - error_S(h)) \geq 5$$

Above we summarized the procedure for calculating confidence intervals for discrete-valued hypotheses.

The following section presents the underlying statistical justification for this procedure.

5.3 BASICS OF SAMPLING THEORY

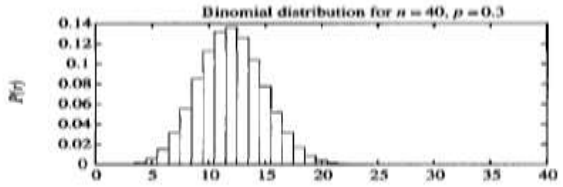
- This section introduces basic notions from statistics and sampling theory, including probability distributions, expected value, variance, Binomial and Normal distributions, and two-sided and one-sided intervals.
- A basic familiarity with these concepts is important to understanding how to evaluate hypotheses and learning algorithms.
- Even more important, these same notions provide an important conceptual framework for understanding machine learning issues such as overfitting and the relationship between successful generalization and the number of training examples considered.
- The key concepts introduced in this section are summarized in Table *5.2.*

- A *random variable* can be viewed as the name of an experiment with a probabilistic outcome. Its value is the outcome of the experiment.
- A *probability distribution* for a random variable Y specifies the probability $\Pr(Y = y_i)$ that Y will take on the value y_i, for each possible value y_i.
- The *expected value*, or *mean*, of a random variable Y is $E[Y] = \sum_i y_i \Pr(Y = y_i)$. The symbol μ_Y is commonly used to represent E[Y].
- The *variance* of a random variable is $Var(Y) = E[(Y - \mu_Y)^2]$. The variance characterizes the width or dispersion of the distribution about its mean.
- The *standard deviation* of Y is $\sqrt{Var(Y)}$. The symbol σ_Y is often used used to represent the standard deviation of Y.
- The *Binomial distribution* gives the probability of observing r heads in a series of n independent coin tosses, if the probability of heads in a single toss is p.
- The *Normal distribution* is a bell-shaped probability distribution that covers many natural phenomena.
- The *Central Limit Theorem* is a theorem stating that the sum of a large number of independent, identically distributed random variables approximately follows a Normal distribution.
- An *estimator* is a random variable Y used to estimate some parameter p of an underlying population.
- The *estimation bias* of Y as an estimator for p is the quantity $(E[Y] - p)$. An unbiased estimator is one for which the bias is zero.
- A *N% confidence interval* estimate for parameter p is an interval that includes p with probability *N%*.

TABLE 5.2 , Basic definitions and facts from statistics.

Error Estimation and Estimating Binomial Proportions :

- Collect a random sample S of *n* independently drawn instances from the distribution *D*, and then measure the sample error errors(h). Repeat this experiment many times, each time drawing a different random sample Si of size *n*, we would expect to observe different values for the various errorsi(h), depending on random differences in the makeup of the various Si. We say that errorsi(h), the outcome of the ith such experiment, is a ***random variable***.
- Imagine that we were to run k random experiments, measuring the random variables errors1(h), errors2(h) . . . errorssk(h) and plotted a histogram displaying the frequency with which each possible error value is observed.
- As ***k*** grows, the histogram would approach a particular probability distribution called the ***Binomial distribution*** which is shown in below figure



A *Binomial distribution* gives the probability of observing r heads in a sample of n independent coin tosses, when the probability of heads on a single coin toss is p. It is defined by the probability function

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

If the random variable X follows a Binomial distribution, then:

- The probability $\Pr(X = r)$ that X will take on the value r is given by $P(r)$
- The expected, or mean value of X, $E[X]$, is

$$E[X] = np$$

- The variance of X, $Var(X)$, is

$$Var(X) = np(1-p)$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sqrt{np(1-p)}$$

For sufficiently large values of ***n*** the Binomial distribution is closely approximated by a Normal distribution (see Table 5.4) with the same mean and variance. Most statisticians recommend using the Normal approximation only when ***np(1- p)*** 2 ***5.***

The Binomial Distribution:

- Consider the following problem for better understanding of Binomial Distribution
- Given a worn and bent coin and estimate the probability that the coin will turn up heads when tossed.
- Unknown probability of heads ***p***. Toss the coin ***n*** times and record the number of times ***r*** that it turns up heads. Estimate of ***p = r / n***
- If the experiment were *rerun*, generating a new set of ***n*** coin tosses, we might expect the number of heads ***r*** to vary somewhat from the value measured in the first experiment, yielding a somewhat different estimate for ***p***.
- The Binomial distribution describes for each possible value of ***r*** (i.e., from 0 to n), the probability of observing exactly ***r*** heads given a sample of ***n*** independent tosses of a coin whose true probability of heads is ***p***.

The general setting to which the Binomial distribution applies is:

1. There is a base experiment (e.g., toss of the coin) whose outcome can be described by a random variable 'Y'. The random variable Y can take on two possible values (e.g., Y = 1 if heads, Y = 0 if tails).

2. The probability that Y = 1 on any single trial of the base experiment is given by some constant p, independent of the outcome of any other experiment. The probability that Y = 0 is therefore (1 - p). Typically, p is not known in advance, and the problem is to estimate it.

3. A series of n independent trials of the underlying experiment is performed (e.g., n independent coin tosses), producing the sequence of independent, identically distributed random variables Y1, Y2, . . . , Yn. Let R denote the number of trials for which Yi = 1 in this series of n experiments.

$$R = \sum_{i=1}^{n} Y_i$$

4. The probability that the random variable R will take on a specific value r (e.g., the probability of observing exactly r heads) is given by the Binomial distribution.

$$\Pr(R = r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r} \qquad \text{equ (1)}$$

The Binomial distribution characterizes the probability of observing *r* heads from n coin flip experiments, as well as the probability of observing *r* errors in a data sample containing n randomly drawn instances.

Mean and Variance:

Definition: Consider a random variable Y that takes on the possible values $y_1, \ldots y_n$. The **expected value** of Y, $E[Y]$, is

$$E[Y] = \sum_{i=1}^{n} y_i \Pr(Y = y_i) \qquad (5.3)$$

For example, if Y takes on the value 1 with probability .7 and the value 2 with probability .3, then its expected value is $(1 \cdot 0.7 + 2 \cdot 0.3 = 1.3)$. In case the random variable Y is governed by a Binomial distribution, then it can be shown that

$$E[Y] = np \qquad (5.4)$$

where n and p are the parameters of the Binomial distribution defined in Equation (5.2).

A second property, the variance, captures the "width" or "spread" of the probability distribution; that is, it captures how far the random variable is expected to vary from its mean value.

Definition: The **variance** of a random variable Y, $Var[Y]$, is

$$Var[Y] = E[(Y - E[Y])^2] \qquad (5.5)$$

The variance describes the expected squared error in using a single observation of Y to estimate its mean $E[Y]$. The square root of the variance is called the *standard deviation* of Y, denoted σ_Y.

Definition: The **standard deviation** of a random variable Y, σ_Y, is

$$\sigma_Y = \sqrt{E[(Y - E[Y])^2]} \qquad (5.6)$$

In case the random variable Y is governed by a Binomial distribution, then the variance and standard deviation are given by

$$Var[Y] = np(1-p)$$
$$\sigma_Y = \sqrt{np(1-p)} \qquad (5.7)$$

Estimators, Bias, and Variance:

Now that we have shown that the random variable ***errors(h)*** obeys a Binomial distribution, we return to our primary question: What is the likely difference between ***errors(h)*** and the true error ***errorv(h)?*** Let us describe ***errors(h)*** and ***errorv(h)*** using the terms in Equation defining the Binomial distribution. We then have

$$error_S(h) = \frac{r}{n}$$
$$error_D(h) = p$$

- where ***n*** is the number of instances in the sample S,
- ***r*** is the number of instances from S misclassified by ***h,*** and
- p is the probability of misclassifying a single instance .
- Statisticians call ***errors(h)*** an ***estimator*** for the true error ***errorv(h).***
- In general, an estimator is any random variable used to estimate some parameter of the underlying population from which the sample is drawn.
- An obvious question to ask about any estimator is whether on average it gives the right estimate. We define the ***estimation bias*** to be the difference between the expected value of thee stimator and the true value of the parameter.

Definition: The **estimation bias** of **an** estimator Y for **an** arbitrary parameter p is

$$E[Y] - p$$

- If the estimation bias is zero, we say that Y is an ***unbiased estimator*** for ***p.*** Notice this will be the case if the average of many random values of Y generated by repeated random experiments (i.e., E[Y]) converges toward p.
- Is ***errors(h)*** an unbiased estimator for ***errorv(h)?*** Yes, because for a Binomial distribution the expected value of ***r*** is equal to ***np*** (Equation r5.41). It follows, given that ***n*** is a constant, that the expected value of ***r / n*** is p.

Two quick remarks are in order regarding the estimation bias.:

- First, when we mentioned at the beginning of this chapter that testing the hypothesis on the training examples provides an optimistically biased estimate of hypothesis error,it is exactly this notion of estimation bias to which we were referring.
- **A** second important property of any estimator is its variance. Given a choice among alternative unbiased estimators, it makes sense to choose the one with least variance. By our definition of variance, this choice will yield the smallest expected squared error between the estimate and the true value of the parameter.

- Example
  - 12 errors on a sample of 40 randomly drawn test examples

$$\hat{p} = error_S(h) = \frac{r}{n} = \frac{12}{40} = 0.3$$
$$\sigma_r^2 = np(1-p) \cong n\hat{p}(1-\hat{p}) = 40 \times 0.3 \times (1-0.3) = 8.4$$
$$\sigma_r = \sqrt{8.4} \cong 2.9$$
$$\sigma_{error_S(h)} = \sigma_{r/n} = \frac{\sigma_r}{n} = \frac{2.9}{40} = 0.07$$

In general, given *r* errors in a sample of *n* independently drawn test examples, the standard deviation for *errors(h)* is given by

$$\sigma_{error_S(h)} = \frac{\sigma_r}{n} = \sqrt{\frac{p(1-p)}{n}}$$

which can be approximated by substituting $r/n = error_S(h)$ for p

$$\sigma_{error_S(h)} \approx \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

5.3.5 Confidence Intervals

One common way to describe the uncertainty associated with an estimate is to give an interval within which the true value is expected to fall, along with the probability with which it is expected to fall into this interval. Such estimates are called *confidence interval* estimates.

Definition: An $N\%$ **confidence interval** for some parameter p is an interval that is expected with probability $N\%$ to contain p.

For example, if we observe $r = 12$ errors in a sample of $n = 40$ independently drawn examples, we can say with approximately 95% probability that the interval 0.30 ± 0.14 contains the true error $error_{\mathcal{D}}(h)$.

How can we derive confidence intervals for $error_{\mathcal{D}}(h)$? The answer lies in the fact that we know the Binomial probability distribution governing the estimator $error_S(h)$. The mean value of this distribution is $error_{\mathcal{D}}(h)$, and the standard deviation is given by Equation (5.9). Therefore, to derive a 95% confidence interval, we need only find the interval centered around the mean value $error_{\mathcal{D}}(h)$,

- Confidence intervals
  - N% confidence interval
    : An interval that is expected with probability N % to contain p
  - Confidence interval for μ and y : $y \pm z_N \sigma$, $\mu \pm z_N \sigma$

Obtaining Confidence Intervals for $error_D(h)$

1. $error_S(h)$ ~ **Binomial distribution where** $\mu = error_D(h)$, $\sigma = \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$
2. **For large n, this binomial distribution is approximated by a normal distribution**
3. **Find the N% confidence interval for estimating μ of a Normal distribution**

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1-error_S(h))}{n}} \qquad (n \geq 30 \text{ or } np(1-p) \geq 5)$$

  - Two approximations involved
    - $error_D(h)$ approximated by $error_S(h)$
    - Binomial distribution approximated by normal distribution

- Normal distribution
  - A bell shaped distribution specified by its mean μ and standard deviation σ
  - Central limit theorem (See Section 5.4.1)

"Binomial distribution can be approximated by normal distribution"

Normal distribution

- X : A random variable $X \in (-\infty, +\infty)$

Probability density function $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

Cumulative distribution $\Pr[a \leq X \leq b] = \int_a^b p(x)dx$

Expected value, variance, and standard deviation

$E[X] = \mu$ $\quad Var[X] = \sigma^2$ $\quad \sigma_X = \sigma$

- Normal distribution
  - Table about the Standard Normal distribution $(\mu = 0,\ \sigma = 1)$; Table 5.1
  - The size of the interval about the mean that contains N% of the probability

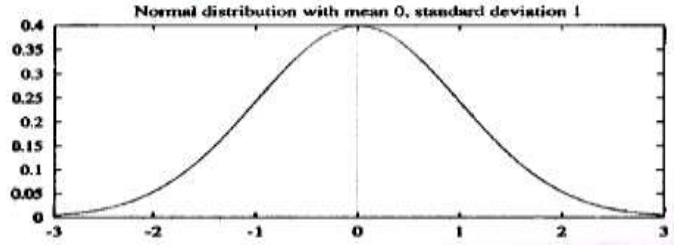


Table 5.1

Confidence Level N%	50%	68%	80%	90%	95%	98%	99%
Constant z_N	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Two-sided and One-sided Bounds:

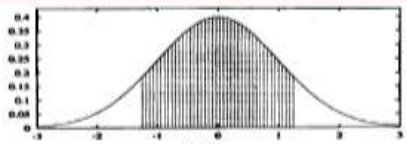
- Notice that the above confidence interval is a two-sided bound; that is, it bounds the estimated quantity from above and from below. In some cases, we will be interested only in a one-sided bound.
- For example, we might be interested in the question "What is the probability that errorz,(h) is at most U?' This kind of one sided question is natural when we are only interested in bounding the maximum error of h and do not mind if the true error is much smaller than estimated.

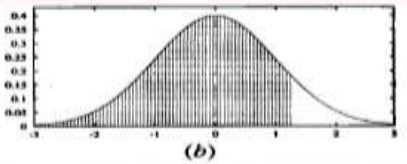
- There is an easy modification to the above procedure for finding such one sided error bounds. Because of this fact, any two-sided confidence interval based on a Normal distribution can be converted to a corresponding one-sided interval with twice the confidence.
- That is, a 100(1- a)% confidence interval with lower bound L and upper bound ***U*** implies a 100(1- a/2)% confidence interval with lower bound L and no upper bound.
- It also implies a 100(1 -a/2)% confidence interval with upper bound ***U*** and no lower bound.
- Here a corresponds to the probability that the correct value lies outside the stated interval.

- Two-sided and one-sided bounds
  - Two-sided bound specifies both lower and upper bound
  - One-sided bound specifies either of them

"What is the probability that $error_D(h)$ is at most U ?" → One-sided bound

$100(1-\alpha)\%$ Confidence Interval $100(1-\alpha/2)\%$ Confidence Interval	α : The probability that the correct value lies outside the interval





- Example
  - 12 errors on a sample of 40 randomly drawn test examples

$error_S(h) = 0.3$

$\sigma_{error_S(h)} = 0.07$

(Two-sided) 95% confidence interval $(\alpha = 0.05)$

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1-error_S(h))}{n}} = 0.3 \pm 1.96 \times 0.07 = 0.3 \pm 0.14$$

(One-sided) 97.5% confidence interval $(\alpha = 0.05)$

$error_D(h)$ is at most $0.3+0.14=0.44$

No assertion about the lower bound!

5.4 A GENERAL APPROACH FOR DERIVING CONFIDENCE INTERVALS:

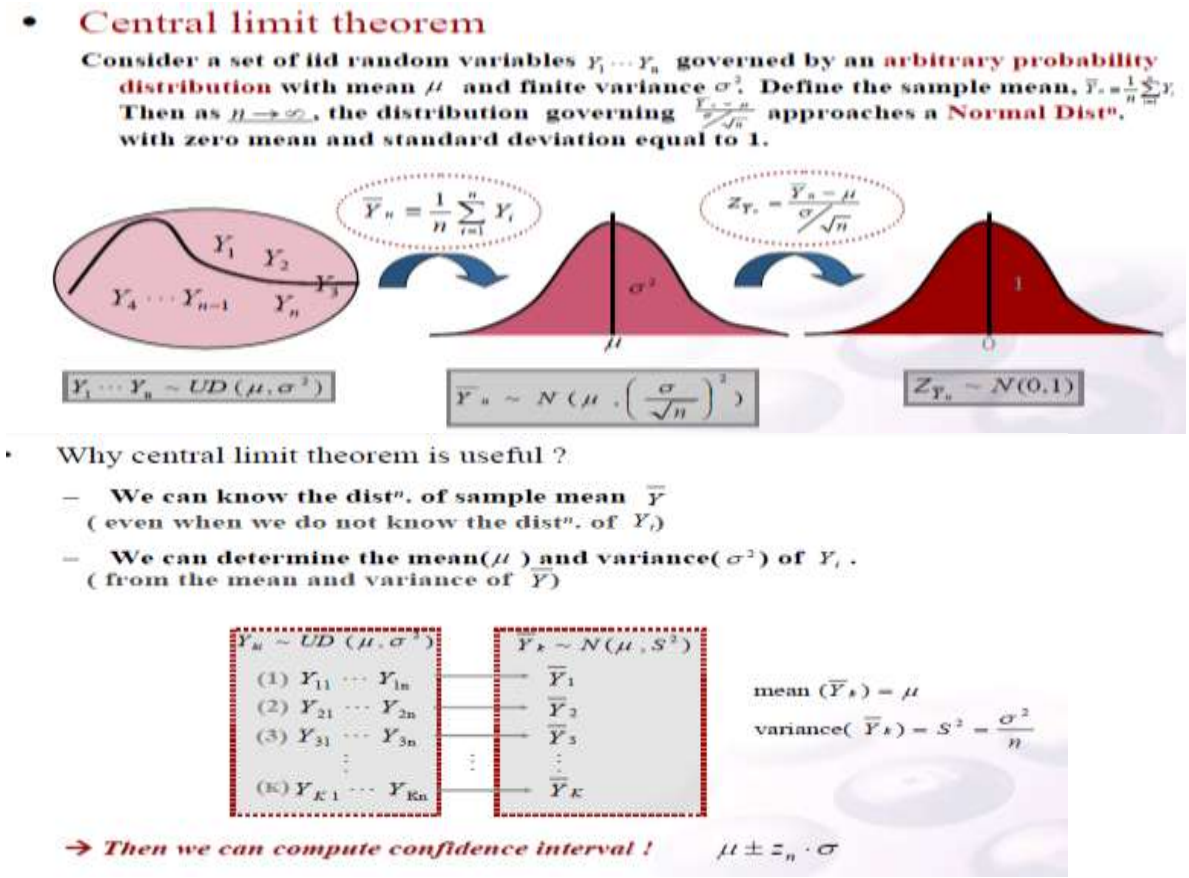
- The previous section described in detail how to derive confidence interval estimates for one particular case: estimating ***errorv(h)*** for a discrete-valued hypothesis ***h,*** based on a sample of ***n*** independently drawn instances.
- The approach described there illustrates a general approach followed in many estimation problems.

In particular, we can see this as a problem of estimating the mean (expected value) of a population based on the mean of a randomly drawn sample of size ***n.***

The general process includes the following steps:

1. Identify the underlying population parameter p to be estimated, for example, ***errorv(h).***
2. Define the estimator Y (e.g., ***errors(h)).*** It is desirable to choose a minimumvariance, unbiased estimator.
3. Determine the probability distribution ***Vy*** that governs the estimator Y, including its mean and variance.
4. Determine the ***N%*** confidence interval by finding thresholds L and ***U*** such that ***N%*** of the mass in the probability distribution ***Vy*** falls between L and U.

Central Limit Theorem:



- Why central limit theorem is useful ?
  - **We can know the dist^n. of sample mean $\bar{Y}$**
    (even when we do not know the dist^n. of Y_i)
  - **We can determine the mean(μ) and variance(σ^2) of Y_i .**
    (from the mean and variance of $\bar{Y}$)

→ ***Then we can compute confidence interval !*** $\mu \pm z_n \cdot \sigma$

5.5 DIFFERENCE IN ERROR OF TWO HYPOTHESES:

Consider the case where we have two hypotheses h_1 and h_2 for some discrete-valued target function. Hypothesis h_1 has been tested on a sample S_1 containing n_1 randomly drawn examples, and h_2 has been tested on an independent sample S_2 containing n_2 examples drawn from the same distribution. Suppose we wish to estimate the difference d between the true errors of these two hypotheses.

- Parameter to be estimated

  : The difference between the true error of 2 hypotheses, h_1 & h_2.

  : Parameter $d \equiv error_D(h_1) - error_D(h_2)$

- CASE 1 : Tested on independent test samples

  – Hypothesis h_1 : sample S_1 containing n_1 examples

  – Hypothesis h_2 : sample S_2 containing n_2 examples

  : Estimator $\hat{d} \equiv error_{S_1}(h_1) - error_{S_2}(h_2)$

  – $\hat{d}$ gives an unbiased estimate of d : $E(\hat{d}) = d$

$$
\begin{aligned}
E(\hat{d}) - d &= E\{error_{S1}(h_1) - error_{S2}(h_2)\} - \{error_D(h_1) - error_D(h_2)\} \\
&= E\{error_{S1}(h_1)\} - E\{error_{S2}(h_2)\} - \{error_D(h_1) - error_D(h_2)\} \\
&= [E\{error_{S1}(h_1)\} - error_D(h_1)] + [-E\{error_{S2}(h_2)\} + error_D(h_2)] \\
&\cong [error_D(h_1) - error_D(h_1)] + [-error_D(h_2) + error_D(h_2)] \\
&= 0
\end{aligned}
$$

- CASE 1 : Tested on independent test samples (continued)

  – **For large n_1, n_2 (>= 30), dist^n. of $\hat{d}$ is approximately Normal dist^n.**

  $\because$ $error_{S1}(h_1) \sim N(\mu_1, \sigma_1)$, $error_{S2}(h_2) \sim N(\mu_2, \sigma_2)$

  Difference of 2 normal distributions is also a normal distribution

  – **Mean of $\hat{d}$**

$$E(\hat{d}) = E\{error_{S1}(h_1) - error_{S2}(h_2)\} \cong \mu_1 - \mu_2$$

  recall : $E(aX - bY) = aE(X) - bE(Y)$ (if X and Y are independent R.V.)

  – **Variance of $\hat{d}$**

$$\sigma^2_{\hat{d}} \equiv \frac{error_{S1}(h_1)(1 - error_{S1}(h_1))}{n_1} + \frac{error_{S2}(h_2)(1 - error_{S2}(h_2))}{n_2}$$

  recall : $Var(aX - bY) = a^2 Var(X) + b^2 Var(Y)$ (if X and Y are independent R.V.)

  – **Confidence Interval of $\hat{d}$ (when n_1, n_2 are large enough).**

$$\hat{d} \pm z_N \cdot \sqrt{\frac{error_{S1}(h_1)(1 - error_{S1}(h_1))}{n_1} + \frac{error_{S2}(h_2)(1 - error_{S2}(h_2))}{n_2}}$$

- CASE 2 : Tested on a single test sample

  : Hypothesis h_1 & Hypothesis h_2 are tested on a single test sample S.

  : Estimator $\hat{d} \equiv error_S(h_1) - error_S(h_2)$

  – **Confidence interval of $\hat{d}$.**

$$\hat{d} \pm z_N \cdot \sqrt{\frac{error_S(h_1)(1 - error_S(h_1)) + error_S(h_2)(1 - error_S(h_2))}{n}}$$

  – **Smaller variance comparing with CASE1.**

  : Single sample S eliminates the variance due to random differences in the S_1 and S_2.

Hypothesis Testing:

: Testing for some specific conjecture (rather than in confidence intervals for some parameter)

– **Situation**
- Independent sample S_1 & S_2 ($|S_1| = |S_2| = 100$)
- $error_{S1}(h_1) = 0.30$
- $error_{S2}(h_2) = 0.20$
- $\hat{d} = 0.10$

"What is the probability the $error_D(h_1) > error_D(h_2)$ given $\hat{d} = 0.10$?"
"What is the probability that d>0 given $\hat{d} = 0.10$?"

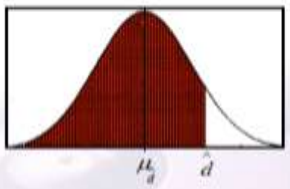
- $\hat{d}$ falls into the one-sided interval $\hat{d} < d + 0.10 \rightarrow \hat{d} < \mu_{\hat{d}} + 0.10$

$$\hat{d} < \mu_{\hat{d}} + Z_N \cdot \sigma_{\hat{d}}$$

$$Z_N \cdot \sigma_{\hat{d}} = 0.10 \quad , \quad \sigma_{\hat{d}} = \sqrt{\frac{0.3(1-0.3)+0.2(1-0.2)}{100}} \approx 0.061$$

$$Z_N = 1.64$$

Two-sided constant for 90% confidence interval



– **Test result**

Therefore, the probability the $error_D(h_1) > error_D(h_2)$ is approximately 95% .
- Accept H_0 with 95% confidence
- Reject H_0 with 5% significant level

5.6 COMPARING LEARNING ALGORITHMS:

- Often we are interested in comparing the performance of two learning algorithms ***LA*** and ***LB,*** rather than two specific hypotheses.
- What is an appropriate test for comparing learning algorithms, and how can we determine whether an observed difference between the algorithms is statistically significant?
- Although there is active debate within the machine-learning research community regarding the best method for comparison, we present here one reasonable approach.
- A discussion of alternative methods is given by Dietterich (1996).

Which of L_A and L_B is the better learning method on average for learning some particular target function f?

- Comparing the performance of two algorithms (L_A, L_B)

: Expected value of the difference in errors between L_A and L_B where $L_A(S)$ is the hypothesis output by learning method, L_A, on the sample, S, of training data.

$$\underset{S \subset D}{E}[error_D(L_A(S)) - error_D(L_B(S))]$$

(S : Training Data sampled from underlying distribution D)

- Practical ways of algorithm comparison given limited sample, D_0, of data

(1) Partitioning data set into training set & test set

: A limited sample D_0 is divided into a training set S_0 and Test Set T_0

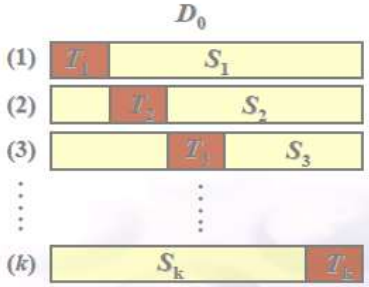
$$error_{T_0}(L_A(S_0)) - error_{T_0}(L_B(S_0))$$

(2) Repeated partitioning and averaging : k-fold method

: D_0 is divided into disjoint training and test sets repeatedly and then the mean of the test set errors for these different experiment is calculated. $\underset{S \subset D_0}{E}[error_{D}(L_A(S)) - error_{D}(L_B(S))]$

1. Partition the available data D_0 into k disjoint subsets $T_1, T_2, \ldots, T_k$ of equal size, where this size is at least 30.
2. For i from 1 to k, do
   use T_i for the test set, and the remaining data for training set S_i
   - $S_i \leftarrow \{D_0 - T_i\}$
   - $h_A \leftarrow L_A(S_i)$
   - $h_B \leftarrow L_B(S_i)$
   - $\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$
3. Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k}\sum_{i=1}^{k}\delta_i \qquad \text{(T5.1)}$$



$\bar{\delta}$ returned from the above is the estimate of

$$\underset{S \subset D}{E}[error_D(L_A(S)) - error_D(L_B(S))] \qquad |S_k| = \frac{k-1}{k}|D_0|, \qquad |T_k| \geq 30$$

which is again the approximation of $\underset{S \subset D_0}{E}[error_D(L_A(S)) - error_D(L_B(S))]$

- The approximate $N\%$ confidence interval

$$\bar{\delta} \pm t_{N,k-1} \cdot s_{\bar{\delta}} \quad \text{where} \quad s_{\bar{\delta}} \equiv \sqrt{\frac{1}{k(k-1)}\sum_{i=1}^{k}(\delta_i - \bar{\delta})^2}$$

	Confidence level N			
	90%	95%	98%	99%
$\nu = 2$	2.92	4.30	6.96	9.92
$\nu = 5$	2.02	2.57	3.36	4.03
$\nu = 10$	1.81	2.23	2.76	3.17
$\nu = 20$	1.72	2.09	2.53	2.84
$\nu = 30$	1.70	2.04	2.46	2.75
$\nu = 120$	1.66	1.98	2.36	2.62
$\nu = \infty$	1.64	1.96	2.33	2.58

- N : Confidence level ,
- k-1 : Degrees of freedom ν, number of independent random events producing the values for random variable $\bar{\delta}$
- If $k \rightarrow \infty$ $t_{N,k-1}$ approaches the constant z_N.

Paired test : *Tests where the hypotheses are evaluated over identical samples.*
Paired Test generate tighter confidence interval than Test on Separate Data samples (Due to eliminate the difference of sample makeup)

- **Paired t-test**

: Statistical justification of the previous comparing algorithm procedure

— **Estimation procedure**

(1) Given i.i.d. random variables : $Y_1, \ldots, Y_k$

(2) Estimate the mean μ of distribution governing Y_i from estimator

(3) Estimator : $\bar{Y} = \frac{1}{k}\sum_{i=1}^{k} Y_i$

- t-test, which is applicable to the special case of the estimator procedure where each Y_i follows a Normal distribution, provides

$$\bar{Y} - t_{N,k-1} \cdot s_{\bar{Y}} \leq \mu = E(Y_i) \leq \bar{Y} + t_{N,k-1} \cdot s_{\bar{Y}} \quad \text{, where} \quad s_{\bar{Y}} = \sqrt{\frac{1}{k(k-1)}\sum_{i=1}^{k}(Y_i - \bar{Y})^2}$$

where $t_{N,k-1}$ is a constant characterizing t distribution as z_n characterizes a Normal distribution.

- In the previous comparing learning algorithm, if on each iteration a new random training set S_i and new random test set T_i are drawn from the underlying instance distribution instead of the fixed sample D_0, then each $\delta_i = error_{T_i}(h_A) - error_{T_i}(h_B)$ with $|T_i| \geq 30$ follows a normal distribution and thus from t-test result,

$$\mu = E(\delta_i) = \underset{S \subset D}{E}[error_D(L_A(S)) - error_D(L_B(S))] = \bar{\delta} \pm t_{N,K-1} \cdot s_{\bar{\delta}}$$

- Practical considerations

Paired t-test does not strictly justify the confidence interval previously discussed because it is evaluated on a limited data D_0 and partitioned method. Nevertheless, this confidence interval provides good basis for experimental comparisons of learning methods.

- When data is limited…

(1) k-fold method

- k is limited.
- Test set are drawn independently (examples are tested exactly once)

(2) Randomized method

: Randomly choose a test set at least 30 examples from D_0 and use remaining examples for training.

- Procedure can be repeated infinitely (k can be infinite number → narrower confidence interval)
- Test sets are not independent.

UNIT-2 QUESTIONS:

1. Explain the two key difficulties that arise while estimating the Accuracy of Hypothesis.
2. Define the following terms

a. Sample error b. True error c. Random Variable

d. Expected value e. Variance f. standard Deviation

3. Explain Binomial Distribution with an example.
4. Explain Normal or Gaussian distribution with an example.
5. Explain the Central Limit Theorem with an example.
6. Explain the Sampling Theorem
7. Discus general approach for deriving confidence intervals
8. Discuss the method of comparing two algorithms. Justify with paired to tests method.
9. Write the Procedure for estimating the difference in error between two learning methods. Approximate confidence intervals for this estimate

UNIT-3

Bayesian Learning:

- **Bayes Theorem and Concept Learning,**
- **Maximum Likelihood,**
- **Minimum Description Length Principle,**
- **Bayes Optimal Classifier,**
- **Gibbs Algorithm,**
- **Naïve Bayes Classifier,**
- **Bayesian Belief Network,**
- **EM Algorithm.**

Bayesian Learning:

Bayesian reasoning provides a probabilistic approach to inference. It is based on the assumption that the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about these probabilities together with observed data.

6.1 INTRODUCTION

Bayesian learning methods are relevant to study of machine learning for two different reasons.

1. First, Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems
2. The second reason is that they provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities.

Features of Bayesian Learning Methods

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example
- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian learning, prior knowledge is provided by asserting (1) a prior probability for each candidate hypothesis, and (2) a probability distribution over observed data for each possible hypothesis.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
- Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

Practical difficulty in applying Bayesian methods

1. One practical difficulty in applying Bayesian methods is that they typically require initial knowledge of many probabilities. When these probabilities are not known in advance they are often estimated based on background knowledge, previously availabledata, and assumptions about the form of the underlying distributions.
2. A second practical difficulty is the significant computational cost required to determine the Bayes optimal hypothesis in the general case. In certain specialized situations, this computational cost can be significantly reduced.

6.2 BAYES THEOREM

Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

Notations

- P(h) prior probability of h, reflects any background knowledge about the chance that h is correct
- P(D) prior probability of D, probability that D will be observed
- P(D|h) probability of observing D given a world in which h holds
- P(h|D) posterior probability of h, reflects confidence that h holds after D has been observed

Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability P(h|D), from the prior probability P(h), together with P(D) and P(D|h).

Bayes Theorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- P(h|D) increases with P(h) and with P(D|h) according to Bayes theorem.
- P(h|D) decreases as P(D) increases, because the more probable it is that D will be observed independent of h, the less evidence D provides in support of h.

Maximum a Posteriori (MAP) Hypothesis

- In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis.
- Bayes theorem to calculate the posterior probability of each candidate hypothesis is hMAP is a MAP hypothesis provided

P(D) can be dropped, because it is a constant independent of h

$$h_{MAP} = \underset{h \in H}{argmax}\ P(h|D)$$
$$= \underset{h \in H}{argmax}\ \frac{P(D|h)P(h)}{P(D)}$$
$$= \underset{h \in H}{argmax}\ P(D|h)P(h)$$

Maximum Likelihood (ML) Hypothesis

- In some cases, it is assumed that every hypothesis in H is equally probable a priori ($P(h_i) = P(h_j)$ for all h_i and h_j in H).
- In this case the below equation can be simplified and need only consider the term P(D|h) to find the most probable hypothesis.

$$h_{MAP} = \underset{h \in H}{argmax}\ P(D|h)P(h)$$

the equation can be simplified

$$h_{ML} = \underset{h \in H}{argmax}\ P(D|h)$$

P(D|h) is often called the likelihood of the data D given h, and any hypothesis that maximizesP(D|h) is called a maximum likelihood (ML) hypothesis

Example

- Consider a medical diagnosis problem in which there are two alternative hypotheses: (1) that the patient has particular form of cancer, and (2) that the patient does not. The available data is from a particular laboratory test with two possible outcomes: + (positive) and - (negative).
- We have prior knowledge that over the entire population of people only .008 have this disease. Furthermore, the lab test is only an imperfect indicator of the disease.
- The test returns a correct positive result in only 98% of the cases in which the disease is actually present and a correct negative result in only 97% of the cases in which the disease is not present. In other cases, the test returns the opposite result.
- The above situation can be summarized by the following probabilities:

$$P(cancer) = .008 \qquad P(\neg cancer) = 0.992$$
$$P(\oplus|cancer) = .98 \qquad P(\ominus|cancer) = .02$$
$$P(\oplus|\neg cancer) = .03 \qquad P(\ominus|\neg cancer) = .97$$

Suppose a new patient is observed for whom the lab test returns a positive (+) result.
Should we diagnose the patient as having cancer or not?

$$P(\oplus|cancer)P(cancer) = (.98).008 = .0078$$
$$P(\oplus|\neg cancer)P(\neg cancer) = (.03).992 = .0298$$
$$\Rightarrow h_{MAP} = \neg cancer$$

The exact posterior probabilities can also be determined by normalizing the above quantities so that they sum to 1

$$P(cancer|\oplus) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$

$$P(\neg cancer|\oplus) = \frac{0.0298}{0.0078 + 0.0298} = 0.79$$

Basic formulas for calculating probabilities are summarized in Table

---

- *Product rule*: probability $P(A \wedge B)$ of a conjunction of two events A and B

$$P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$$

- *Sum rule*: probability of a disjunction of two events A and B

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

- *Bayes theorem*: the posterior probability $P(h|D)$ of h given D

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- *Theorem of total probability*: if events $A_1, \ldots, A_n$ are mutually exclusive with $\sum_{i=1}^{n} P(A_i) = 1$, then

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

6.3 BAYES THEOREM AND CONCEPT LEARNING:

What is the relationship between Bayes theorem and the problem of concept learning?

Since Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data, and can use it as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs the most probable.

Brute-Force Bayes Concept Learning

Consider the concept learning problem

- Assume the learner considers some finite hypothesis space H defined over the instance space X, in which the task is to learn some target concept $c : X \rightarrow \{0,1\}$.
- Learner is given some sequence of training examples $((x1, d1) \ldots (xm, dm))$ where x_i is some instance from X and where d_i is the target value of x_i (i.e., $d_i = c(x_i)$).
- The sequence of target values are written as $D = (d1 \ldots dm)$.

We can design a straightforward concept learning algorithm to output the maximum a posteriori hypothesis, based on Bayes theorem, as follows:

BRUTE-FORCE MAP LEARNING algorithm:

1. For each hypothesis h in H, calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

$$h_{MAP} = \underset{h \in H}{argmax}\ P(h|D)$$

In order specify a learning problem for the BRUTE-FORCE MAP LEARNING algorithm wemust specify what values are to be used for P(h) and for P(D|h) ?

Let's choose P(h) and for P(D|h) to be consistent with the following assumptions:

1. The training data D is noise free (i.e., $d_i = c(x_i)$)
2. The target concept c is contained in the hypothesis space H
3. Do not have a priori reason to believe that any hypothesis is more probable than any other.

What values should we specify for P(h)?

- Given no prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis h in H.
- Assume the target concept is contained in H and require that these prior probabilities sum to 1.

$$P(h) = \frac{1}{|H|} \text{ for all } h \in H$$

What choice shall we make for P(D|h)?

- P(D|h) is the probability of observing the target values $D = (d_1 \ldots d_m)$ for the fixed set of instances $(x_1 \ldots x_m)$, given a world in which hypothesis h holds
- Since we assume noise-free training data, the probability of observing classification d_i given h is just 1 if $d_i = h(x_i)$ and 0 if $d_i \neq h(x_i)$. Therefore,

$$P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \in D \\ 0 & \text{otherwise} \end{cases}$$

Given these choices for P(h) and for P(D|h) we now have a fully-defined problem for the above BRUTE-FORCE MAP LEARNING algorithm.

Recalling Bayes theorem, we have

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Consider the case where h is inconsistent with the training data D

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$$

The posterior probability of a hypothesis inconsistent with D is zero

Consider the case where h is consistent with D

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)} = \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} = \frac{1}{|VS_{H,D}|}$$

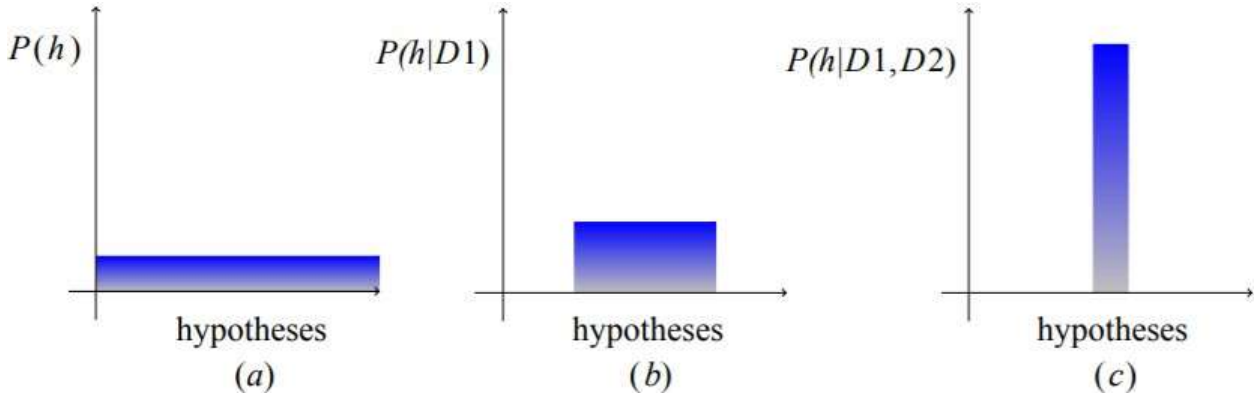
Where, $VS_{H,D}$ is the subset of hypotheses from H that are consistent with D

To summarize, Bayes theorem implies that the posterior probability P(h|D) under our assumed P(h) and P(D|h) is

$$P(D|h) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if h is consistent with D} \\ 0 & \text{otherwise} \end{cases}$$

The Evolution of Probabilities Associated with Hypotheses

- Figure (a) all hypotheses have the same probability.
- Figures (b) and (c), As training data accumulates, the posterior probability for inconsistent hypotheses becomes zero while the total probability summing to 1 is shared equally among the remaining consistent hypotheses.



MAP Hypotheses and Consistent Learners

- A learning algorithm is a consistent learner if it outputs a hypothesis that commits zero errors over the training examples.
- Every consistent learner outputs a MAP hypothesis, if we assume a uniform prior probability distribution over H ($P(h_i) = P(h_j)$ for all i, j), and deterministic, noise free training data (P(D|h) =1 if D and h are consistent, and 0 otherwise).

Example:

- FIND-S outputs a consistent hypothesis, it will output a MAP hypothesis under the probability distributions P(h) and P(D|h) defined above.
- Are there other probability distributions for P(h) and P(D|h) under which FIND-S outputs MAP hypotheses? Yes.
- Because FIND-S outputs a maximally specific hypothesis from the version space, its output hypothesis will be a MAP hypothesis relative to any prior probability distribution that favours more specific hypotheses.

Note

- Bayesian framework is a way to characterize the behaviour of learning algorithms
- By identifying probability distributions P(h) and P(D|h) under which the output is a optimal hypothesis, implicit assumptions of the algorithm can be characterized (Inductive Bias)
- Inductive inference is modelled by an equivalent probabilistic reasoning system based on Bayes theorem

6.4 MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES

Consider the problem of learning a *continuous-valued target function* such as neural network learning, linear regression, and polynomial curve fitting

A straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a *maximum likelihood (ML) hypothesis*

- Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X, i.e., $(\forall\ h \in H)[\ h : X \rightarrow R]$ and training examples of the form $<x_i, d_i>$

- The problem faced by L is to learn an unknown target function $f : X \rightarrow R$

- A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution with zero mean ($d_i = f(x_i) + e_i$)

- Each training example is a pair of the form (x_i, d_i) where $d_i = f(x_i) + e_i$.
  - Here $f(x_i)$ is the noise-free value of the target function and e_i is a random variable representing the noise.
  - It is assumed that the values of the e_i are drawn independently and that they are distributed according to a Normal distribution with zero mean.

- The task of the learner is to *output a maximum likelihood hypothesis* or a *MAP hypothesis assuming all hypotheses are equally probable a priori*.

- A simple example of such a problem is learning a linear function, though our analysis applies to learning arbitrary real-valued functions.
- igure 6.2 illustrates a linear target function f depicted by the solid line, and a set of noisy training
- examples of this target function. The dashed line corresponds to the hypothesis ***hML*** with least-squared training error, hence the maximum likelihood hypothesis. Notice that the maximum likelihood hypothesis is not necessarily identical to the correct hypothesis, f, because it is inferred from only a limited sample of noisy training data.

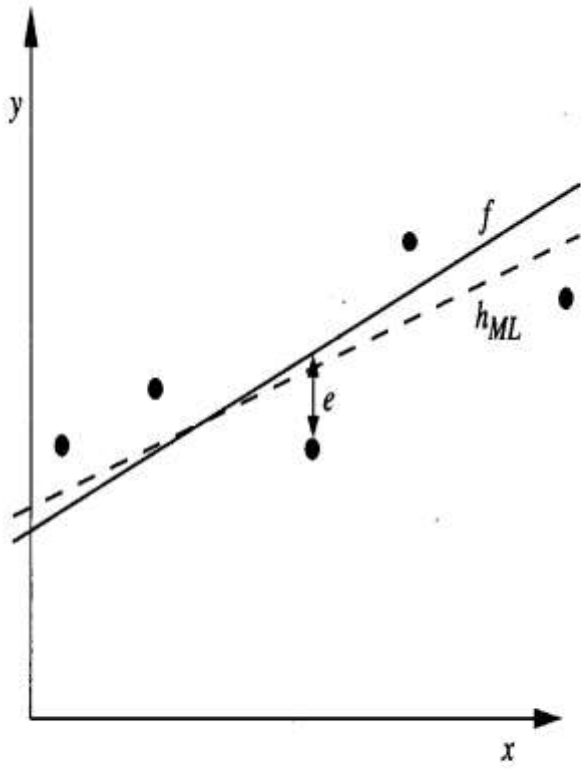


FIGURE 6.2
Learning a real-valued function. The target function f corresponds to the solid line. The training examples (x_i, d_i) are assumed to have Normally distributed noise e_i with zero mean added to the true target value $f(x_i)$. The dashed line corresponds to the linear function that minimizes the sum of squared errors. Therefore, it is the maximum likelihood hypothesis h_{ML}, given these five training examples.

Before showing why a hypothesis that minimizes the sum of squared errors in this setting is also a maximum likelihood hypothesis, let us quickly review two basic concepts from probability theory: probability densities and Normal distributions. First, in order to discuss probabilities over continuous variables such as e, we must introduce probability *densities*. The reason, roughly, is that we wish for the total probability over all possible values of the random variable to sum to one. In the case of continuous variables we cannot achieve this by assigning a finite probability to each of the infinite set of possible values for the random variable. Instead, we speak of a probability *density* for continuous variables such as e and require that the integral of this probability density over all possible values be one. In general we will use lower case p to refer to the probability density function, to distinguish it from a finite probability P (which we will sometimes refer to as a probability *mass*). The probability density $p(x_0)$ is the limit as ϵ goes to zero, of $\frac{1}{\epsilon}$ times the probability that x will take on a value in the interval $[x_0, x_0 + \epsilon)$.

Probability density function:

$$p(x_0) \equiv \lim_{\epsilon \to 0} \frac{1}{\epsilon} P(x_0 \leq x < x_0 + \epsilon)$$

Second, we stated that the random noise variable e is generated by a Normal probability distribution. A Normal distribution is a smooth, bell-shaped distribution that can be completely characterized by its mean μ and its standard deviation σ. See Table 5.4 for a precise definition.

Using the definition of hML we have

$$h_{ML} = \underset{h \in H}{argmax}\ p(D|h)$$

Assuming training examples are mutually independent given h, we can write P(D|h) as the product of the various (di|h)

$$h_{ML} = \underset{h \in H}{argmax} \prod_{i=1}^{m} p(d_i|h)$$

Given the noise ei obeys a Normal distribution with zero mean and unknown variance σ^2, eachdi must also obey a Normal distribution around the true target value f(xi). Because we are writing the expression for P(D|h), we assume h is the correct description of f.
Hence, $\mu = f(x_i) = h(x_i)$

$$h_{ML} = \underset{h \in H}{\text{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2}$$

$$h_{ML} = \underset{h \in H}{argmax} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2}$$

Maximize the less complicated logarithm, which is justified because of the monotonicity of function p

$$h_{ML} = \underset{h \in H}{argmax} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of h, and can therefore be discarded, yielding

$$h_{ML} = \underset{h \in H}{argmax} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity

$$h_{ML} = \underset{h \in H}{argmin} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally, discard constants that are independent of h.

$$h_{ML} = \underset{h \in H}{argmin} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

Thus, above equation shows that the maximum likelihood hypothesis h_{ML} is the one that minimizes the sum of the squared errors between the observed training values d_i and the hypothesis predictions $h(x_i)$

Note:

Why is it reasonable to choose the Normal distribution to characterize noise?

- Good approximation of many types of noise in physical systems
- Central Limit Theorem shows that the sum of a sufficiently large number of independent, identically distributed random variables itself obeys a Normal distribution

Only noise in the target value is considered, not in the attributes describing the instances themselves

6.5 MAXIMUM LIKELIHOOD HYPOTHESES FOR PREDICTING PROBABILITIES

- Consider the setting in which we wish to learn a nondeterministic (probabilistic) function f : X → {0, 1}, which has two discrete output values.
- We want a function approximator whose output is the probability that f(x) = 1. In other words, learn the target function f ` : X → [0, 1] such that f ` (x) = P(f(x) = 1)

How can we learn f ` using a neural network?

- Use of brute force way would be to first collect the observed frequencies of 1's and 0's for each possible value of x and to then train the neural network to output the target frequency for each x.

What criterion should we optimize in order to find a maximum likelihood hypothesis for f' inthis setting?

- First obtain an expression for P(D|h)
- Assume the training data D is of the form D = {(x1, d1) . . . (xm, dm)}, where d_i is the observed 0 or 1 value for f (x_i).
- Both x_i and d_i as random variables, and assuming that each training example is drawn independently, we can write P(D|h) as

$$P(D \mid h) = \prod_{i=1}^{m} P(x_i, d_i \mid h) \qquad \text{equ (1)}$$

Applying the product rule

$$P(D \mid h) = \prod_{i=1}^{m} P(d_i \mid h, x_i) P(x_i) \qquad \text{equ (2)}$$

The probability P(d_i|h, x_i)

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases} \qquad \text{equ (3)}$$

Re-express it in a more mathematically manipulable form, as

$$P(d_i|h, x_i) = h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} \qquad \text{equ (4)}$$

Equation (4) to substitute for P(di |h, xi) in Equation (5) to obtain

$$P(D|h) = \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} P(x_i) \qquad \text{equ (5)}$$

We write an expression for the maximum likelihood hypothesis

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} P(x_i)$$

The last term is a constant independent of h, so it can be dropped

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \qquad \text{equ (6)}$$

It easier to work with the log of the likelihood, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)) \qquad \text{equ (7)}$$

Equation (7) describes the quantity that must be maximized in order to obtain the maximum likelihood hypothesis in our current problem setting

Gradient Search to Maximize Likelihood in a Neural Net:

- Derive a weight-training rule for neural network learning that seeks to maximize G(h,D)using gradient ascent
- The gradient of G(h,D) is given by the vector of partial derivatives of G(h,D) with respect to the various network weights that define the hypothesis h represented by the learned network
- In this case, the partial derivative of G(h, D) with respect to weight w_{jk} from input k to unit j is

$$\begin{aligned} \frac{\partial G(h,D)}{\partial w_{jk}} &= \sum_{i=1}^{m} \frac{\partial G(h, D)}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\ &= \sum_{i=1}^{m} \frac{\partial (d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)))}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\ &= \sum_{i=1}^{m} \frac{d_i - h(x_i)}{h(x_i)(1 - h(x_i))} \frac{\partial h(x_i)}{\partial w_{jk}} \qquad \text{equ (1)} \end{aligned}$$

- Suppose our neural network is constructed from a single layer of sigmoid units. Then,

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \sigma'(x_i) x_{ijk} = h(x_i)(1 - h(x_i)) x_{ijk}$$

  where x_{ijk} is the k^{th} input to unit j for the i^{th} training example, and d(x) is the derivativeof the sigmoid squashing function.

- Finally, substituting this expression into Equation (1), we obtain a simple expression forthe derivatives that constitute the gradient

$$\frac{\partial G(h, D)}{\partial w_{jk}} = \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk}$$

Because we seek to maximize rather than minimize P(D|h), we perform gradient ascent rather than gradient descent search. On each iteration of the search the weight vector is adjusted in the direction of the gradient, using the weight update rule

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

Where,

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk} \qquad \text{equ (2)}$$

Where, η is a small positive constant that determines the step size of the i gradient ascent search

6.6 MINIMUM DESCRIPTION LENGTH PRINCIPLE

- A Bayesian perspective on Occam's razor
- Motivated by interpreting the definition of h_{MAP} in the light of basic concepts from information theory.

$$h_{MAP} = \underset{h \in H}{argmax}\ P(D|h)P(h)$$

which can be equivalently expressed in terms of maximizing the $\log_2$

$$h_{MAP} = \underset{h \in H}{argmax}\ \log_2 P(D|h) + \log_2 P(h)$$

or alternatively, minimizing the negative of this quantity

$$h_{MAP} = \underset{h \in H}{argmin}\ -\log_2 P(D|h) - \log_2 P(h) \qquad \text{equ (1)}$$

This equation (1) can be interpreted as a statement that short hypotheses are preferred,assuming a particular representation scheme for encoding hypotheses and data

- $-\log_2 P(h)$: the description length of h under the optimal encoding for the hypothesis space H, $L_{C_H}(h) = -\log_2 P(h)$, where C_H is the optimal code for hypothesis space H.
- $-\log_2 P(D \mid h)$: the description length of the training data D given hypothesis h, under theoptimal encoding from the hypothesis space H: $L_{C_H}(D|h) = -\log_2 P(D|\ h)$, where $C_{D|h}$ is the optimal code for describing data D assuming that both the sender and receiver know the hypothesis h.
- Rewrite Equation (1) to show that h_{MAP} is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data giventhe hypothesis.

$$h_{MAP} = \underset{h \in H}{argmin}\ L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Where, C_H and $C_{D|h}$ are the optimal encodings for H and for D given h

The Minimum Description Length (MDL) principle recommends choosing the hypothesis thatminimizes the sum of these two description lengths of equ.

$$h_{MAP} = \underset{h \in H}{argmin}\, L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Minimum Description Length principle:

$$h_{MDL} = \underset{h \in H}{\text{argmin}}\, L_{C_1}(h) + L_{C_2}(D \mid h)$$

Where, codes $C1$ and $C2$ to represent the hypothesis and the data given the hypothesis

The above analysis shows that if we choose $C1$ to be the optimal encoding of hypotheses C_H,and if we choose $C2$ to be the optimal encoding $C_{D|h}$, then $h_{MDL} = h_{MAP}$

Application to Decision Tree Learning

Apply the MDL principle to the problem of learning decision trees from some training data. *What should we choose for the representations C1 and C2 of hypotheses and data?*

- For $C1$: $C1$ might be some obvious encoding, in which the description length grows withthe number of nodes and with the number of edges
- For $C2$: Suppose that the sequence of instances $(x1 \ldots xm)$ is already known to both the transmitter and receiver, so that we need only transmit the classifications $(f(x1) \ldots f(xm))$.
- Now if the training classifications $(f(x1) \ldots f(xm))$ are identical to the predictions of the hypothesis, then there is no need to transmit any information about these examples. Thedescription length of the classifications given the hypothesis ZERO
- If examples are misclassified by h, then for each misclassification we need to transmit a message that identifies which example is misclassified as well as its correct classification
- The hypothesis h_{MDL} under the encoding $C1$ and $C2$ is just the one that minimizes the sum of these description lengths.

6.7 BAYES OPTIMAL CLASSIFIER

So far we have considered the question "what is the most probable *hypothesis* given the training data?" In fact, the question that is often of most significance is the closely related question "what is the most probable *classification* of the new instance given the training data?" Although it may seem that this second question can be answered by simply applying the MAP hypothesis to the new instance, in fact it is possible to do better.

To develop some intuitions consider a hypothesis space containing three hypotheses, h_1, h_2, and h_3. Suppose that the posterior probabilities of these hypotheses given the training data are .4, .3, and .3 respectively. Thus, h_1 is the MAP hypothesis. Suppose a new instance x is encountered, which is classified positive by h_1, but negative by h_2 and h_3. Taking all hypotheses into account, the probability that x is positive is .4 (the probability associated with h_1), and the probability that it is negative is therefore .6. The most probable classification (negative) in this case is different from the classification generated by the MAP hypothesis.

In general, the most probable classification of the new instance is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities. If the possible classification of the new example can take on any value v_j from some set V, then the probability $P(v_j|D)$ that the correct classification for the new instance is v_j, is just

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

The optimal classification of the new instance is the value v_j, for which $P(v_j|D)$ is maximum.

Bayes optimal classification:

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \tag{6.18}$$

To illustrate in terms of the above example, the set of possible classifications of the new instance is $V = \{\oplus, \ominus\}$, and

$$P(h_1|D) = .4, \quad P(\ominus|h_1) = 0, \quad P(\oplus|h_1) = 1$$
$$P(h_2|D) = .3, \quad P(\ominus|h_2) = 1, \quad P(\oplus|h_2) = 0$$
$$P(h_3|D) = .3, \quad P(\ominus|h_3) = 1, \quad P(\oplus|h_3) = 0$$

therefore

$$\sum_{h_i \in H} P(\oplus|h_i)P(h_i|D) = .4$$
$$\sum_{h_i \in H} P(\ominus|h_i)P(h_i|D) = .6$$

and

$$\underset{v_j \in \{\oplus, \ominus\}}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = \ominus$$

Any system that classifies new instances according to Equation (6.18) is called a *Bayes optimal classifier*, or Bayes optimal learner. No other classification method using the same hypothesis space and same prior knowledge can outperform this method on average. This method maximizes the probability that the new instance is classified correctly, given the available data, hypothesis space, and prior probabilities over the hypotheses.

For example, in learning boolean concepts using version spaces as in the earlier section, the Bayes optimal classification of a new instance is obtained by taking a weighted vote among all members of the version space, with each candidate hypothesis weighted by its posterior probability.

Note one curious property of the Bayes optimal classifier is that the predictions it makes can correspond to a hypothesis not contained in H! Imagine using Equation (6.18) to classify every instance in X. The labeling of instances defined in this way need not correspond to the instance labeling of any single hypothesis h from H. One way to view this situation is to think of the Bayes optimal classifier as effectively considering a hypothesis space H' different from the space of hypotheses H to which Bayes theorem is being applied. In particular, H' effectively includes hypotheses that perform comparisons between linear combinations of predictions from multiple hypotheses in H.

6.8 GIBBS ALGORITHM:

Although the Bayes optimal classifier obtains the best performance that can be achieved from the given training data, it can be quite costly to apply.

The expense is due to the fact that it computes the posterior probability for every hypothesis in H and then combines the predictions of each hypothesis to classify each new instance.

An alternative, less optimal method is the Gibbs algorithm (see Opper and Haussler 1991), defined as follows:

1. Choose a hypothesis ***h*** from H at random, according to the posterior probability distribution over H.

2. Use ***h*** to predict the classification of the next instance ***x.***

- Given a new instance to classify, the Gibbs algorithm simply applies a hypothesis drawn at random according to the current posterior probability distribution. Surprisingly, it can be shown that under certain conditions the expected

- Mis classification error for the Gibbs algorithm is at most twice the expected error of the Bayes optimal classifier

- More precisely, the expected value is taken over target concepts drawn at random according to the prior probability distribution assumed by the learner.

- Under this condition, the expected value of the error of the Gibbs algorithm is at worst twice the expected value of the error of the Bayes optimal classifier.

- In particular, it implies that if the learner assumes a uniform prior over H, and if target concepts are in fact drawn from such a distribution when presented to the learner, **then classifying the next instance according to a hypothesis drawn at random from the current version space (according to a uniform distribution), will have expected error at most twice that of the Bayes optimal classifiers.**

- Again, we have an example where a Bayesian analysis of a non-Bayesian algorithm yields insight into the performance of that algorithm.

6.9 NAIVE BAYES CLASSIFIER

- The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V.
- A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (al, a2.. .am).
- The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable targetvalue, VMAP, given the attribute values (al, a2.. .am) that describe the instance

$$v_{MAP} = \underset{v_j \in V}{\text{argmax}}\, P(v_j | a_1, a_2 \ldots a_n)$$

Use Bayes theorem to rewrite this expression as

$$v_{MAP} = \underset{v_j \in V}{\text{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\text{argmax}}\, P(a_1, a_2 \ldots a_n | v_j) P(v_j) \qquad \textbf{equ (1)}$$

- The naive Bayes classifier is based on the assumption that the attribute values are conditionally independent given the target value. Means, the assumption is that given the target value of the instance, the probability of observing the conjunction (al, a2.. .am),is just the product of the probabilities for the individual attributes:

$$P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$$

Substituting this into Equation (1),

Naive Bayes classifier:

$$\mathbf{V_{NB} = \underset{v_j \in V}{\text{argmax}}\, P(v_j) \prod_i P(a_i | v_j)} \qquad \text{equ (2)}$$

Where, VNB denotes the target value output by the naive Bayes classifier

An Illustrative Example

- Let us apply the naive Bayes classifier to a concept learning problem i.e., classifying days according to whether someone will play tennis.
- The below table provides a set of 14 training examples of the target concept ***PlayTennis***, where each day is described by the attributes Outlook, Temperature, Humidity, and Wind

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

- Use the naive Bayes classifier and the training data from this table to classify thefollowing novel instance:

  < Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong >

- Our task is to predict the target value ***(yes or no)*** of the target concept ***PlayTennis*** forthis new instance

$$V_{NB} = \underset{v_j \in \{yes, no\}}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

$$V_{NB} = \underset{v_j \in \{yes, no\}}{\operatorname{argmax}} P(v_j)\, P(Outlook=sunny|v_j)\, P(Temperature=cool|v_j)\, P(Humidity=high|v_j)\, P(Wind=strong|v_j)$$

The probabilities of the different target values can easily be estimated based on theirfrequencies over the 14 training examples

- P(P1ayTennis = yes) = 9/14 = 0.64
- P(P1ayTennis = no) = 5/14 = 0.36

Similarly, estimate the conditional probabilities. For example, those for Wind = strong

- P(Wind = strong | PlayTennis = yes) = 3/9 = 0.33
- P(Wind = strong | PlayTennis = no) = 3/5 = 0.60

Calculate VNB according to Equation (1)

$$P(yes)\ P(sunny|yes)\ P(cool|yes)\ P(high|yes)\ P(strong|yes) = .0053$$
$$P(no)\ P(sunny|no)\ P(cool|no)\ P(high|no)\ P(strong|no) = .0206$$

Thus, the naive Bayes classifier assigns the target value ***PlayTennis = no*** to this newinstance, based on the probability estimates learned from the training data.

By normalizing the above quantities to sum to one, calculate the conditional probability thatthe target value is ***no***, given the observed attribute values

$$\frac{.0206}{(.0206 + .0053)} = .795$$

Estimating Probabilities

- We have estimated probabilities by the fraction of times the event is observed to occurover the total number of opportunities.
- For example, in the above case we estimated P(Wind = strong | Play Tennis = no) by the fraction n_c /n where, n = 5 is the total number of training examples for which PlayTennis = no, and n_c = 3 is the number of these for which Wind = strong.
- When n_c = 0, then n_c /n will be zero and this probability term will dominate the quantity calculated in Equation (2) requires multiplying all the other probability terms by this zero value
- To avoid this difficulty we can adopt a Bayesian approach to estimating the probability,using the ***m-estimate*** defined as follows

***m*-estimate of probability:**

$$\frac{n_c + mp}{n + m}$$

- p is our prior estimate of the probability we wish to determine, and m is a constant called the equivalent sample size, which determines how heavily to weight p relativeto the observed data
- Method for choosing p in the absence of other information is to assume uniformpriors; that is, if an attribute has k possible values we set p = 1 /k.

6.11 BAYESIAN BELIEF NETWORKS

- The naive Bayes classifier makes significant use of the assumption that the values of theattributes $a_1 \ldots a_n$ are conditionally independent given the target value v.
- This assumption dramatically reduces the complexity of learning the target function

A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities
Bayesian belief networks allow stating conditional independence assumptions that apply to subsets of the variables

Notation

- Consider an arbitrary set of random variables $Y_1 \ldots Y_n$, where each variable Yi cantake on the set of possible values $V(Y_i)$.
- The joint space of the set of variables Y to be the cross product $V(Y_1)$ x $V(Y_2)$ x. . . $V(Y_n)$.
- In other words, each item in the joint space corresponds to one of the possible assignments of values to the tuple of variables ($Y_1 \ldots Y_n$). The probability distributionover this joint' space is called the joint probability distribution.
- The joint probability distribution specifies the probability for each of the possible variable bindings for the tuple ($Y_1 \ldots Y_n$).
- A Bayesian belief network describes the joint probability distribution for a set of variables.

Conditional Independence

Let X, Y, and Z be three discrete-valued random variables. X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given a value for Z, that is, if

$$(\forall x_i, y_j, z_k)\ P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

Where,

$$x_i \in V(X),\ y_j \in V(Y),\ \text{and}\ z_k \in V(Z).$$

The above expression is written in abbreviated form as

$$P(X \mid Y, Z) = P(X \mid Z)$$

Conditional independence can be extended to sets of variables. The set of variables $X1 \ldots X_l$ is conditionally independent of the set of variables $Y1 \ldots Y_m$ given the set of variables $Z1 \ldots Z_n$ if

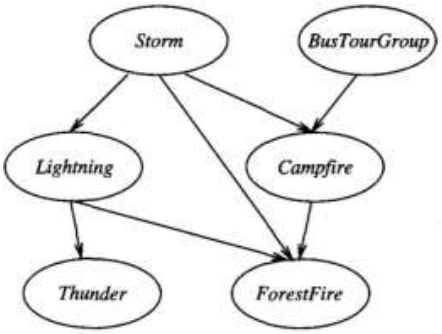
$$P(X_1 \ldots X_l | Y_1 \ldots Y_m, Z_1 \ldots Z_n) = P(X_1 \ldots X_l | Z_1 \ldots Z_n)$$

The naive Bayes classifier assumes that the instance attribute A1 is conditionally independent of instance attribute A2 given the target value V. This allows the naive Bayes classifier to calculate P(Al, A2 | V) as follows,

$$\begin{aligned} P(A_1, A_2|V) &= P(A_1|A_2, V)P(A_2|V) \\ &= P(A_1|V)P(A_2|V) \end{aligned}$$

Representation

A Bayesian belief network represents the joint probability distribution for a set of variables. Bayesian networks (BN) are represented by directed acyclic graphs.



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

Campfire

The Bayesian network in above figure represents the joint probability distribution over the boolean variables *Storm, Lightning, Thunder, ForestFire, Campfire,* and *BusTourGroup*

A Bayesian network (BN) represents the joint probability distribution by specifying a set of *conditional independence assumptions*

- BN represented by a directed acyclic graph, together with sets of local conditionalprobabilities
- Each variable in the joint space is represented by a node in the Bayesian network
- The network arcs represent the assertion that the variable is conditionally independentof its non-descendants in the network given its immediate predecessors in the network.
- A ***conditional probability table*** **(CPT)** is given for each variable, describing the probability distribution for that variable given the values of its immediate predecessors

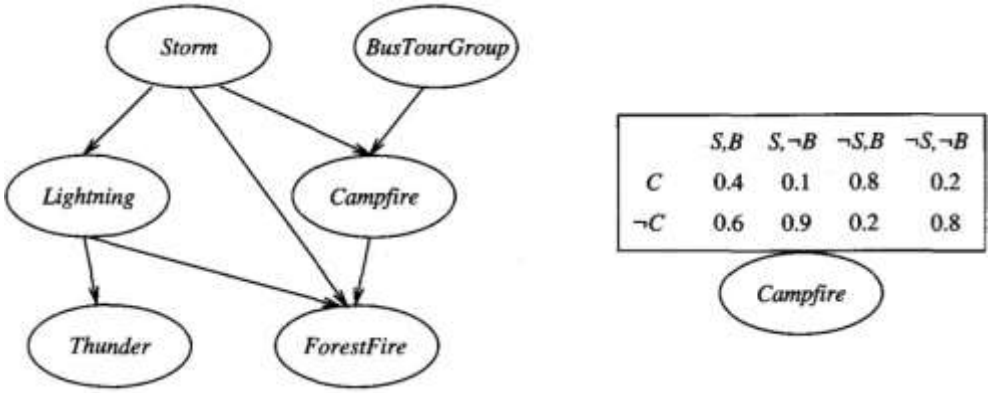
The joint probability for any desired assignment of values (y1, . . . , yn) to the tuple of networkvariables (Y1 . . . Ym) can be computed by the formula

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

Where, Parents(Y_i) denotes the set of immediate predecessors of Y_i in the network.

Example:

Consider the node ***Campfire***. The network nodes and arcs represent the assertion that ***Campfire*** is conditionally independent of its non-descendants ***Lightning*** and ***Thunder***, given its immediate parents Storm and ***BusTourGroup***.



This means that once we know the value of the variables ***Storm*** and ***BusTourGroup***, the variables ***Lightning*** and ***Thunder*** provide no additional information about ***Campfire***

The conditional probability table associated with the variable ***Campfire.*** The assertion

isP(Campfire = True | Storm = True, BusTourGroup = True) = 0.4

Inference

- Use a Bayesian network to infer the value of some target variable (e.g., ForestFire) giventhe observed values of the other variables.
- Inference can be straightforward if values for all of the other variables in the network are known exactly.
- A Bayesian network can be used to compute the probability distribution for any subset of network variables given the values or distributions for any subset of the remaining variables.
- An arbitrary Bayesian network is known to be NP-hard

Learning Bayesian Belief Networks

Affective algorithms can be considered for learning Bayesian belief networks from training data by considering several different settings for learning problem

- First, the network structure might be given in advance, or it might have to be inferred fromthe training data.
- Second, all the network variables might be directly observable in each training example, or some might be unobservable.
  - In the case where the network structure is given in advance and the variables are fullyobservable in the training examples, learning the conditional probability tables is straightforward and estimate the conditional probability table entries
  - In the case where the network structure is given but only some of the variable values are observable in the training data, the learning problem is more difficult. The learningproblem can be compared to learning weights for an ANN.

Gradient Ascent Training of Bayesian Network

The gradient ascent rule which maximizes P(D|h) by following the gradient of ***ln P(D|h)*** withrespect to the parameters that define the conditional probability tables of the Bayesian network.

Let w_{ijk} denote a single entry in one of the conditional probability tables. In particular w_{ijk} denote the conditional probability that the network variable Y_i will take on the value y_i, given that its immediate parents U_i take on the values given by u_{ik}.

The gradient of ***ln P(D|h)*** is given by the derivatives $\frac{\partial \ln P(D|h)}{\partial w_{ijk}}$ for each of the w_{ijk}.As shown below, each of these derivatives can be calculated as

$$\frac{\partial \ln P(D|h)}{\partial w_{ij}} = \sum_{d \in D} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|d)}{w_{ijk}} \qquad \textbf{equ(1)}$$

Derive the gradient defined by the set of derivatives $\frac{\partial P_h(D)}{\partial w_{ijk}}$ for all ***i, j,*** and ***k***. Assuming thetraining examples ***d*** in the data set D are drawn independently, we

$$\begin{aligned}\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \frac{\partial}{\partial w_{ijk}} \ln \prod_{d \in D} P_h(d) \\ &= \sum_{d \in D} \frac{\partial \ln P_h(d)}{\partial w_{ijk}} \\ &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial P_h(d)}{\partial w_{ijk}}\end{aligned}$$

We write the abbreviation $P_h(D)$ to represent $P(D|h)$.

This last step makes use of the general equality $\frac{\partial \ln f(x)}{\partial x} = \frac{1}{f(x)} \frac{\partial f(x)}{\partial x}$. We can now introduce the values of the variables Y_i and $U_i = Parents(Y_i)$, by summing over their possible values $y_{ij'}$ and $u_{ik'}$.

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}, u_{ik'})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}|u_{ik'}) P_h(u_{ik'})$$

This last step follows from the product rule of probability . Now consider the rightmost sum in the final expression above. Given that $w_{ijk} \equiv P_h(y_{ij}|u_{ik})$, the only term in this sum for which $\frac{\partial}{\partial w_{ijk}}$ is nonzero is the term for which $j' = j$ and $i' = i$. Therefore

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) P_h(y_{ij}|u_{ik}) P_h(u_{ik})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) w_{ijk} P_h(u_{ik})$$

$$= \sum_{d \in D} \frac{1}{P_h(d)} P_h(d|y_{ij}, u_{ik}) P_h(u_{ik})$$

Applying Bayes theorem to rewrite $P_h(d|y_{ij}, u_{ik})$, we have

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{P_h(y_{ij}, u_{ik}|d) P_h(d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{P_h(y_{ij}|u_{ik})}$$

$$= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}} \qquad \textbf{equ (2)}$$

Thus, we have derived the gradient given in Equation (1). There is one more item that must be considered before we can state the gradient ascent training procedure. In particular, we require that as the weights w_{ijk} are updated they must remain valid probabilities in the interval [0,1]. We also require that the sum $\sum_j w_{ijk}$ remains 1 for all i, k. These constraints can be satisfied by updating weights in a two-step process. First we update each w_{ijk} by gradient ascent

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

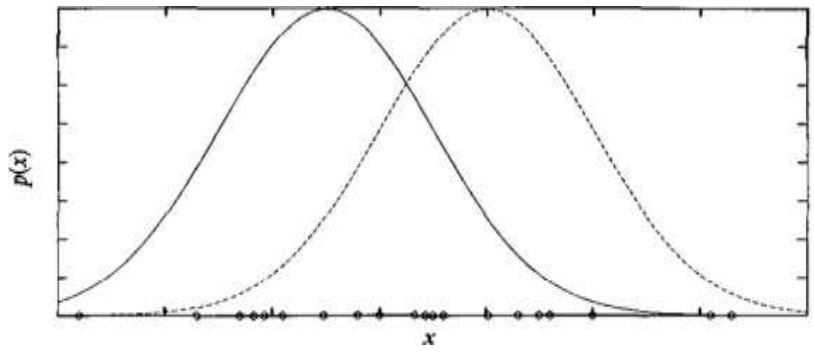
where η is a small constant called the learning rate. Second, we renormalize the weights w_{ijk} to assure that the above constraints are satisfied.
this process will converge to a locally maximum likelihood hypothesis for the conditional probabilities in the Bayesian network.

THE EM ALGORITHM

The EM algorithm can be used even for variables whose value is never directly observed,provided the general form of the probability distribution governing these variables is known.

Estimating Means of k Gaussians

- Consider a problem in which the data D is a set of instances generated by a probabilitydistribution that is a mixture of k distinct Normal distributions.



- This problem setting is illustrated in Figure for the case where k = 2 and where theinstances are the points shown along the x axis.
- Each instance is generated using a two-step process.
  - First, one of the k Normal distributions is selected at random.
  - Second, a single random instance x_i is generated according to this selecteddistribution.
- This process is repeated to generate a set of data points as shown in the figure.

- To simplify, consider the special case
  - The selection of the single Normal distribution at each step is based on choosingeach with uniform probability
  - Each of the k Normal distributions has the same variance σ^2, known value.
- The learning task is to output a hypothesis h = ($\mu 1$, . . . ,μk) that describes the means ofeach of the k distributions.
- We would like to find a maximum likelihood hypothesis for these means; that is, ahypothesis ***h*** that maximizes p(D |h).

$$\mu_{ML} = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{m} (x_i - \mu)^2 \qquad (1)$$

In this case, the sum of squared errors is minimized by the sample mean

$$\mu_{ML} = \frac{1}{m} \sum_{i=1}^{m} x_i \qquad (2)$$

- Our problem here, however, involves a mixture of k different Normal distributions, andwe cannot observe which instances were generated by which distribution.
- Consider full description of each instance as the triple (x_i, z_{i1}, z_{i2}),
  - where x_i is the observed value of the ith instance and
  - where z_{i1} and z_{i2} indicate which of the two Normal distributions was used togenerate the value x_i
- In particular, z_{ij} has the value 1 if x_i was created by the j^{th} Normal distribution and 0otherwise.
- Here x_i is the observed variable in the description of the instance, and z_{il} and z_{i2} arehidden variables.
- If the values of z_{il} and z_{i2} were observed, we could use following Equation to solve forthe means p1 and p2
- Because they are not, we will instead use the EM algorithm

EM algorithm

Step 1: Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

Step 2: Calculate a new maximum likelihood hypothesis $h' = \langle \mu_1', \mu_2' \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in Step 1. Then replace the hypothesis $h = \langle \mu_1, \mu_2 \rangle$ by the new hypothesis $h' = \langle \mu_1', \mu_2' \rangle$ and iterate.

Let us examine how both of these steps can be implemented in practice. Step 1 must calculate the expected value of each z_{ij}. This $E[z_{ij}]$ is just the probability that instance x_i was generated by the jth Normal distribution

$$E[z_{ij}] = \frac{p(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i | \mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$$

Thus the first step is implemented by substituting the current values $\langle \mu_1, \mu_2 \rangle$ and the observed x_i into the above expression.

In the second step we use the $E[z_{ij}]$ calculated during Step 1 to derive a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$.
maximum likelihood hypothesis in this case is given by

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}] \; x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

UNIT WISE QUESTIONS:

1. Define (i) Prior Probability (ii) Conditional Probability (iii) Posterior Probability

2. Define Bayesian theorem? What is the relevance and features of Bayesian theorem? Explain the practicaldifficulties of Bayesian theorem.

3. Consider a medical diagnosis problem in which there are two alternative hypotheses: 1. That the patient has a particular form of cancer (+) and 2. That the patient does not (-). A patient takes a lab test and the result comes back positive. The test returns a correct positive result in only 98% of the cases in which the disease is actually present, anda correct negative result in only 97% of the cases in which the disease is not present. Furthermore, .008 of the entire population have this cancer. Determine whether the patient has Cancer or not using MAP hypothesis.

4. Explain Brute force Bayes Concept Learning

5. Define MAP hypothesis. Derive the relation for h_{MAP} using Bayesian theorem.

6. What are Consistent Learners?

7. Discuss Maximum Likelihood and Least Square Error Hypothesis.

8. Describe Maximum Likelihood Hypothesis for predicting probabilities.

9. Describe the concept of MDL. Obtain the equation for h_{MDL}

10. What is conditional Independence?

11. Explain Naïve Bayes Classifier with an Example.
12. Explain the Gradient Search to Maximize Likelihood in a neural Net.
13. What are Bayesian Belief nets? Where are they used?
14. Explain Bayesian belief network and conditional independence with example.
15. Explain the concept of EM Algorithm. Discuss what are Gaussian Mixtures.

UNIT-4

INSTANCE BASED LEARNING

Instance Based Learning:

- K-Nearest Neighbor Learning,
- Locally Weighted Regression,
- Radial Basis Functions,
- Case Based Reasoning,
- Lazy and Eager learning.

8.1 INTRODUCTION

- Instance-based learning methods such as nearest neighbor and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions.
- Learning in these algorithms consists of simply storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance
- Instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified

Advantages of Instance-based learning

1. Training is very fast
2. Learn complex target function
3. Don't lose information

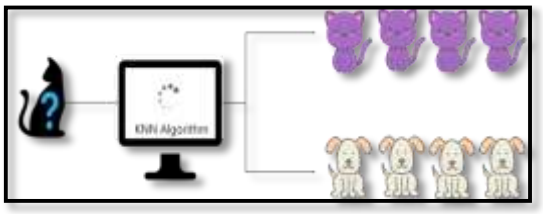
Disadvantages of Instance-based learning

- The cost of classifying new instances can be high. This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered.
- In many instance-based approaches, especially nearest-neighbor approaches, is that they typically consider all attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be alarge distance apart.

8.2 k-EAREST NEIGHBOR LEARNING:

KNN which stand for K Nearest Neighbor is a Supervised Machine Learning algorithm that classifies a new data point into the target class, depending on the features of its neighboring data points.

Example: Let's say we want a machine to distinguish between images of cats & dogs. To do this we must input a dataset of cat and dog images and we have to train our model to detect the animals based on certain features. For example, features such as pointy ears can be used to identify cats and similarly we can identify dogs based on their long ears.



After studying the dataset during the training phase, when a new image is given to the model, the KNN algorithm will classify it into either cats or dogs depending on the similarity in their features. So if the new image has pointy ears, it will classify that image as a cat because it is similar to the cat images.

Features of KNN Algorithm

- KNN is a Supervised Learning algorithm that uses labeled input data set to predict the output of the data points.
- It is one of the most simple Machine learning algorithms and it can be easily implemented for a varied set of problems.
- It is mainly based on feature similarity. KNN checks how similar a data point is to its neighbor and classifies the data point into the class it is most similar to.
- Unlike most algorithms, KNN is a non-parametric model which means that it does not make any assumptions about the data set. This makes the algorithm more effective since it can handle realistic data.
- KNN is a lazy algorithm, this means that it memorizes the training data set instead of learning a discriminative function from the training data.
- KNN can be used for solving both classification and regression problems.

The most basic instance-based method is the K- Nearest Neighbor Learning.
This algorithm assumes all instances correspond to points in the n-dimensional space R^n.

- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- Let an arbitrary instance x be described by the feature vector

  $$((a1(x), a2(x), \ldots\ldots, an(x))$$

  Where, $a_r(x)$ denotes the value of the r^{th} attribute of instance x.

- Then the distance between two instances x_i and x_j is defined to be $d(x_i , x_j)$
  Where,

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

- In nearest-neighbor learning the target function may be either discrete-valued or real-valued.

Let us first consider learning ***discrete-valued target functions*** of the

$$f : \Re^n \rightarrow V.$$

formWhere, V is the finite set $\{v1, \ldots vs\}$

The k- Nearest Neighbor algorithm for approximation a **discrete-valued target function** is given below:

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

- The value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of f among the k training examples nearest to x_q.
- If k = 1, then the 1- Nearest Neighbor algorithm assigns to $\hat{f}(x_q)$ the value $f(x_i)$. Where x_i is the training instance nearest to x_q.
- For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

- Below figure illustrates the operation of the k-Nearest Neighbor algorithm for the case where the instances are points in a two-dimensional space and where the target function is Boolean valued.

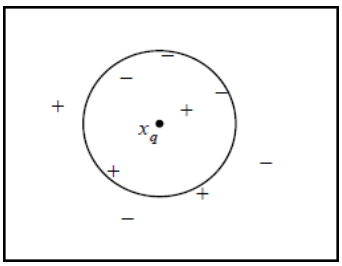
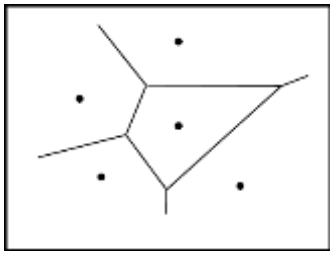


Figure: k-NEAREST NEIGHBOR A set of positive and negative training examples is shown on the left, along with a query instance x, to be classified.

- The positive and negative training examples are shown by "+" and "-" respectively. A query point x_q is shown as well.
- The 1-Nearest Neighbor algorithm classifies x_q as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.

- Below figure shows the shape of this **decision surface** induced by 1- Nearest Neighbor over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples.



For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagramis often called the ***Voronoi diagram*** of the set of training example

The K- Nearest Neighbor algorithm for approximation a **real-valued target function** is given below $f : \Re^n \rightarrow \Re$

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

8.2.1 Distance-Weighted Nearest Neighbor Algorithm

- The refinement to the k-NEAREST NEIGHBOR Algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point xq,giving greater weight to closer neighbors.
- For example, in the k-Nearest Neighbor algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from xq

Distance-Weighted Nearest Neighbor Algorithm for approximation a discrete-valued target functions

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Distance-Weighted Nearest Neighbor Algorithm for approximation a Real-valued target functions

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

8.2.2 Remarks on k-NEAREST NEIGHBOR Algorithm:

- The distance-weighted k-NEARESNTE IGHBOR algorithm is a highly effective inductive inference method for many practical problems.
- It is robust to noisy training data and quite effective when it is provided a sufficiently large set of training data.
- Note that by taking the weighted average of the k neighbors nearest to the query point, it can smooth out the impact of isolated noisy training examples.

What is the inductive bias of k-NEAREST NEIGHBOR?

The basis for classifying new query points is easily understood based on the diagrams in Figure 8.1. The inductive bias corresponds to an assumption that the classification of an instance ***x,*** will be most similar to the classification of other instances that are nearby in Euclidean distance.

One practical issue in applying k-NEAREST NEIGHBOR algorithms is that the distance between instances is calculated based on *all* attributes of the instance (i.e., on all axes in the Euclidean space containing the instances).

To see the effect of this policy, consider applying k-NEARESNTE IGHBE to a problem in which each instance is described by 20 attributes, but where only 2 of these attributes are relevant to determining the classification for the particular target function. In this case, instances that have identical values for the 2 relevant attributes may nevertheless be distant from one another in the 20-dimensional instance space.

As a result, the similarity metric used by k-NEARESNT EIGHBOR—depending on all 20 attributes- will be misleading. The distance between neighbors will be dominated by the large number of irrelevant attributes. This difficulty, which arises when many irrelevant attributes are present, is sometimes referred to as the ***curse of dimensionality.*** Nearest-neighbor approaches are especially sensitive to this problem.

One interesting approach to overcoming this problem is to weight each attribute differently when calculating the distance between two instances. This corresponds to stretching the axes in the Euclidean space, shortening the axes that correspond to less relevant attributes, and lengthening the axes that correspond to more relevant attributes.

One additional practical issue in applying k-NEARESNTE IGHBOR is efficient memory indexing. Because this algorithm delays all processing until a new query is received, significant computation can be required to process each new query. Various methods have been developed for indexing the stored training examples so that the nearest neighbors can be identified more efficiently at some additional cost in memory. One such indexing method is the kd-tree in which instances are stored at the leaves of a tree, with nearby instances stored at the same or nearby nodes. The internal nodes of the tree sort the new query ***x,*** to the relevant leaf by testing selected attributes of ***x,.***

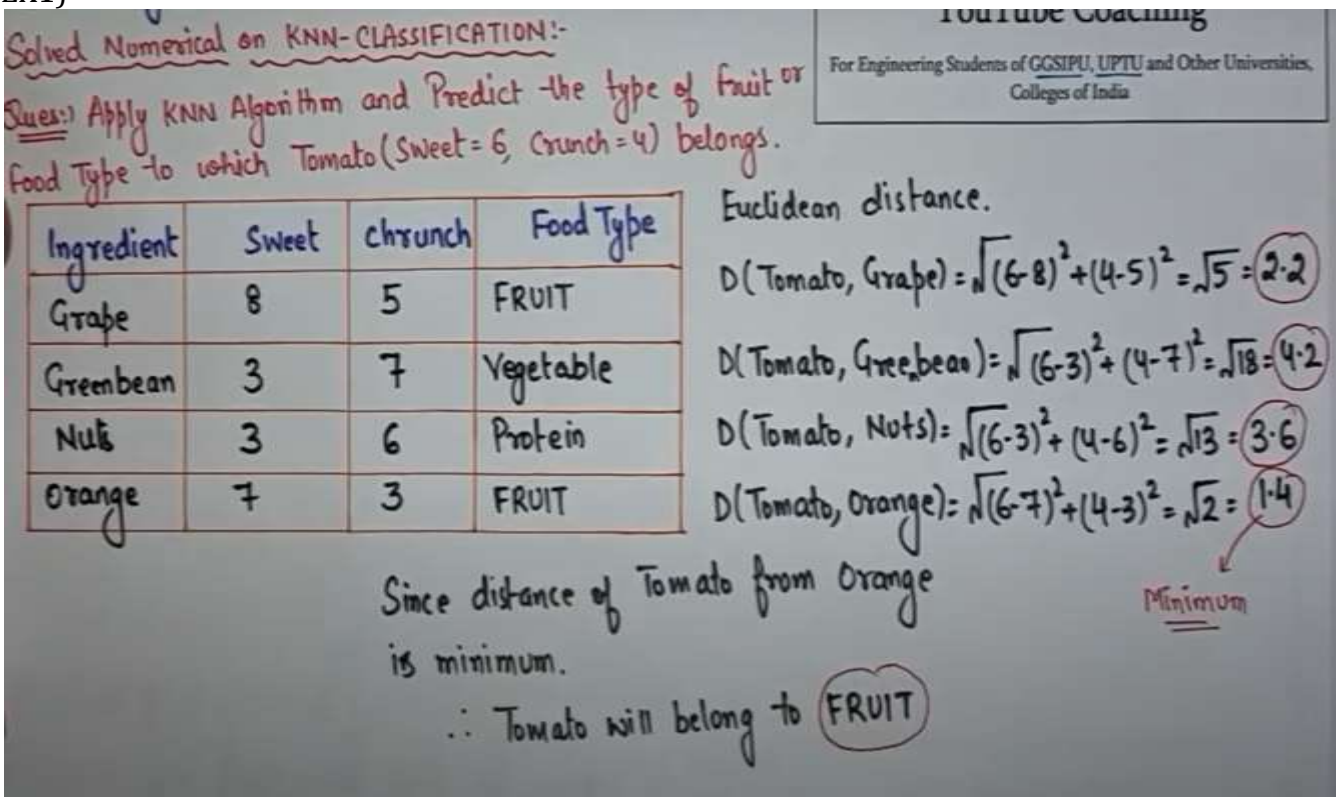
8.2.3 Terminology

- **Regression** means approximating a real-valued target function.
- **Residual** is the error $\hat{f}(x) - f(x)$ in approximating the target function.
- **Kernel function** is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$

KNN EXAMPLES:

Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

EX1)



EX2)

Consider a dataset that contains two variables: height (cm) & weight (kg). Each point is classified as normal or underweight.

Weight(x2)	Height(y2)	Class
51	167	Underweight
62	182	Normal
69	176	Normal
64	173	Normal
65	172	Normal
56	174	Underweight
58	169	Normal
57	173	Normal
55	170	Normal

Based on the above data, you need to classify the following set as normal or underweight using the KNN algorithm.

Now, we have a new data point (x1, y1), and we need to determine its class.

57 kg	170 cm	?

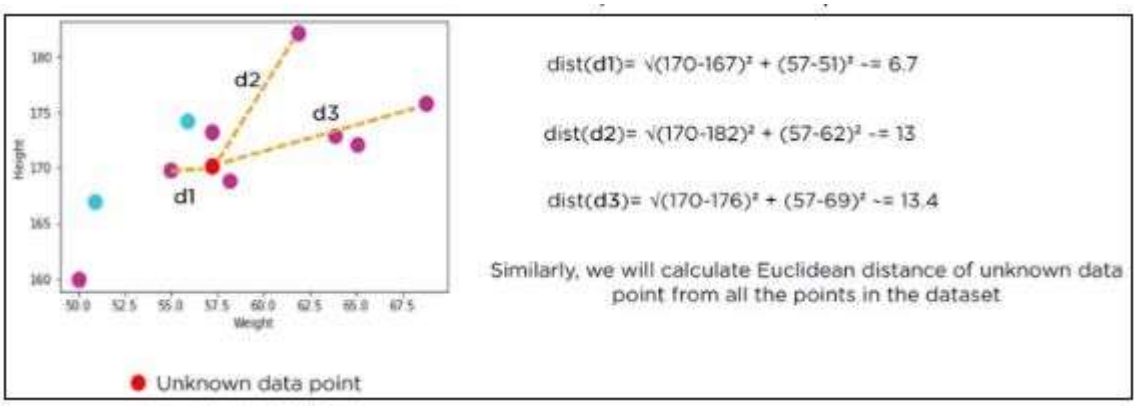
ANS)

Looking at the new data, we can consider the last three rows from the table—K=3.

To find the nearest neighbors, we will calculate the Euclidean distance.

The Euclidean distance between two points in the plane with coordinates (x,y) and (a,b) is given by:

Let us calculate the Euclidean distance with the help of unknown data points.



The following table shows the calculated Euclidean distance of unknown data points from all points.

Weight(x2)	Height(y2)	Class	Euclidean Distance
51	167	Underweight	6.7
62	182	Normal	13
69	176	Normal	13.4
64	173	Normal	7.6
65	172	Normal	8.2
56	174	Underweight	4.1
58	169	Normal	1.4
57	173	Normal	3
55	170	Normal	2

Now, we have a new data point (x1, y1), and we need to determine its class.

57 kg	170 cm	?

Looking at the new data, we can consider the last three rows from the table—K=3.

Weight(x2)	Height(y2)	Class	Euclidean Distance
51	167	Underweight	6.7
62	182	Normal	13
69	176	Normal	13.4
64	173	Normal	7.6
65	172	Normal	8.2
56	174	Underweight	4.1
58	169	Normal	1.4
57	173	Normal	3
55	170	Normal	2

Since the majority of neighbors are classified as normal as per the KNN algorithm, the data point (57, 170) should be normal.

8.3 LOCALLY WEIGHTED REGRESSION:

- The nearest-neighbor approaches described in the previous section can be thought of as approximating the target function f ***(x)*** at the single query point ***x*** = ***x,.*** Locally weighted regression is a generalization of this approach. It constructs an explicit approximation to f over a local region surrounding ***x,.***
- Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to f. For example, we might approximate the target function in the neighborhood surrounding ***x,*** using a linear function, a quadratic function, a multilayer neural network, or some other functional form.

- The phrase "locally weighted regression" is called local because the function is approximated based only on data near the query point, weighted because the contribution of each training example is weighted by its distance from the query point, and regression because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

- Given a new query instance xq, the general approach in locally weighted regression is to construct an approximation f that fits the training examples in the neighborhood surrounding xq. This approximation is then used to calculate the value f(xq), which is output as the estimated target value for the query instance.

8.3.1 Locally Weighted Linear Regression

- Consider locally weighted regression in which the target function ***f*** is approximated near xq using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

Where, ai(x) denotes the value of the i^{th} attribute of the instance x

- Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

Which led us to the gradient descent train**ing rule**

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

Where, η is a constant learning rate

- Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \qquad \text{equ(1)}$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from xq :

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x)) \qquad \text{equ(2)}$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x)) \qquad \text{equ(3)}$$

If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) \; (f(x) - \hat{f}(x)) \; a_j(x)$$

The differences between this new rule and the rule given by Equation (3) are that the contribution of instance x to the weight update is now multiplied by the distance penalty **K(d(xq, x)),** and that the error is summed over only the k nearest training examples.

8.3.2 Remarks on Locally Weighted Regression

Above we considered using a linear function to approximate f in the neighborhood of the query instance ***x,.*** The literature on locally weighted regression contains a broad range of alternative methods for distance weighting the training examples, and a range of methods for locally approximating the target function.
In most cases, the target function is approximated by a constant, linear, or quadratic function.
More complex functional forms are not often found because
(1) the cost of fitting more complex functions for each query instance is prohibitively high, and
(2) these simple approximations model the target function quite well over a sufficiently small sub region of the instance space.

8.4 RADIAL BASIS FUNCTIONS

- One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions
- In this approach, the learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x)) \qquad \textbf{equ (1)}$$

- Where, each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance d(xu, x) increases.
- Here k is a user provided constant that specifies the number of kernel functions to be included.
- f is a global approximation to f (x), the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u.

Choose each function $K_u(d(x_u, x))$ to be a Gaussian function centred at the point x_u with some variance σ_u^2

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

- The functional form of equ(1) can approximate any function with arbitrarily small error, provided a sufficiently large number k of such Gaussian kernels and provided the width σ^2 of each kernel can be separately specified
- The function given by equ(1) can be viewed as describing a two layer network where the first layer of units computes the values of the various $K_u(d(x_u, x))$ and where the second layer computes a linear combination of these first-layer unit values

Example: Radial basis function (RBF) network

Given a set of training examples of the target function, RBF networks are typically trained ina two-stage process.

1. First, the number k of hidden units is determined and each hidden unit u is defined by choosing the values of x_u and σ_u^2 that define its kernel function $K_u(d(x_u, x))$
2. Second, the weights w, are trained to maximize the fit of the network to the training data, using the global error criterion given by

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

Because the kernel functions are held fixed during this second stage, the linear weight values w, can be trained very efficiently

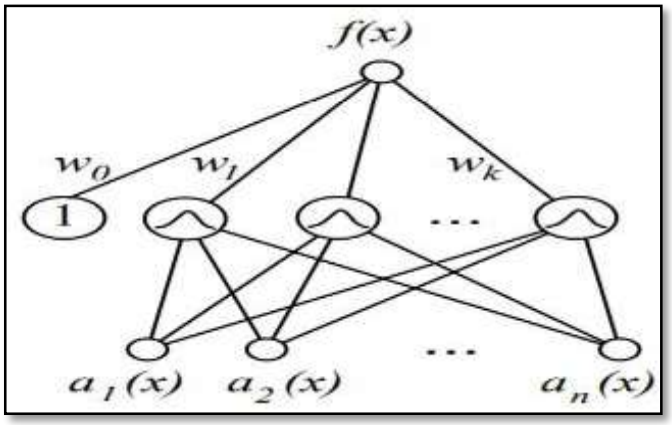


FIGURE 8.2

A radial basis function network. Each hidden unit produces an activation determined by a Gaussian function centered at some instance *xu.* Therefore, its activation will be close to zero unless the input x is near *xu.* The output unit produces a linear combination of the hidden unit activations. Although the network shown here has just one output, multiple output units can also be included.

Several alternative methods have been proposed for choosing an appropriate number of hiddenunits or, equivalently, kernel functions.

- One approach is to allocate a Gaussian kernel function for each training example (xi,f (xi)), centring this Gaussian at the point xi.
  Each of these kernels may be assigned the same width σ^2. Given this approach, the RBF network learns a global approximation to the target function in which each training example (xi, f (xi)) can influence the value of ***f*** only in the neigh bourhood of xi.

- A second approach is to choose a set of kernel functions that is smaller than the number of training examples. This approach can be much more efficient than the first approach, especially when the number of training examples is large.

- The set of kernel functions may be distributed with centers spaced uniformly throughout the instance space X. Alternatively, we may wish to distribute the centers non uniformly, especially if the instances themselves are found to be distributed non uniformly over X. In this later case, we can pick kernel function centers by randomly selecting a subset of the training instances, thereby sampling the underlying distribution of instances.
- Alternatively, we may identify prototypical clusters of instances, then add a kernel function centered at each cluster. The placement of the kernel functions in this fashion can be accomplished using unsupervised clustering algorithms that fit the training instances (but not their target values) to a mixture of Gaussians. In the case of the EM algorithm, the means are chosen to maximize the probability of observing the instances xi, given the k estimated means. Note the target function value f (xi) of the instance does not enter into the calculation of kernel centers by unsupervised clustering methods. The only role of the target values f (xi) in this case is to determine the output layer weights w,.

Summary

- Radial basis function networks provide a global approximation to the target function, represented by a linear combination of many local kernel functions.
- The value for any given kernel function is non-negligible only when the input x falls into the region defined by its particular centre and width. Thus, the network can be viewed as a smooth linear combination of many local approximations to the target function.
- One key advantage to RBF networks is that they can be trained much more efficiently than feedforward networks trained with BACKPROPAGATION.

8.5 CASE-BASED REASONING:

Instance-based methods such as k-NEARESTN EIGHBOR and locally weighted regression share three key properties:

- First, they are lazy learning methods in that they defer the decision of how to generalize beyond the training data until a new query instance is observed.
- Second, they classify new query instances by analyzing similar instances while ignoring instances that are very different from the query.
- Third, they represent instances **as** real-valued points in an n-dimensional Euclidean space.

Case-based reasoning (CBR) is a learning paradigm based on lazy learning methods andthey classify new query instances by analysing similar instances while ignoring instances that are very different from the query.

- In CBR represent instances are not represented as real-valued points, but instead, they use a ***rich symbolic*** representation.
- CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs, reasoning about new legal cases based on previous rulings, and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems

A prototypical example of a case-based reasoning

- The CADET system employs case-based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets.
- It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems.
-
- Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function.
- New design problems are then presented by specifying the desired function and requesting the corresponding structure.

The problem setting is illustrated in below figure

This problem setting is illustrated in Figure 8.3.

The top half of the figure shows the description of a typical stored case called a T-junction pipe. Its function is represented in terms of the qualitative relationships among the waterflow levels and temperatures at its inputs and outputs.
In the functional description at its right, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. For example, the output water flow ***Q3*** increases with increasing input waterflow **Ql.** Similarly, a "-" label indicates that the variable at the head decreases with the variable at the tail.

The bottom half of this figure depicts a new design problem described by its desired function. This particular function describes the required behavior of one type of water faucet.
Here Q, refers to the flow of cold water into the faucet,

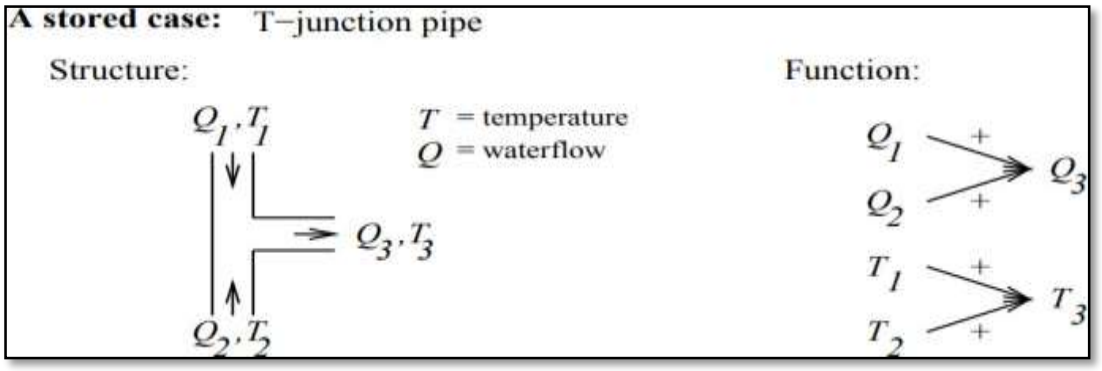
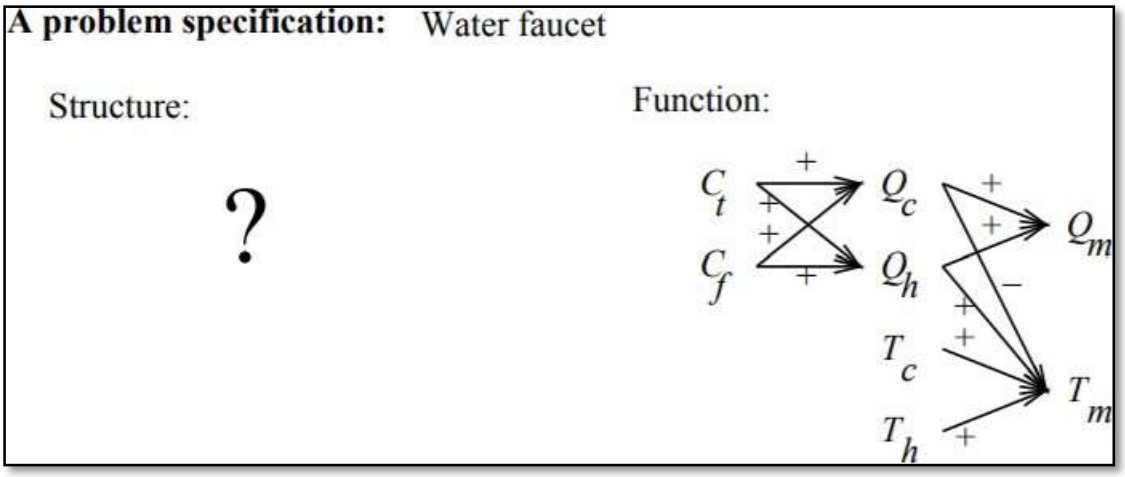


FIGURE 8.3

A stored case and a new problem.

The top half of the figure describes a typical design fragment in the case library of CADET. The function is represented by the graph of qualitative dependencies among the T-junction variables (described in the text). The bottom half of the figure shows a typical design problem.

- The function is represented in terms of the qualitative relationships among the water-flow levels and temperatures at its inputs and outputs.
- In the functional description, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. A "-" label indicates that the variable at the head decreases with the variable at the tail.
- Here Q_c refers to the flow of cold water into the faucet, Q_h to the input flow of hot water, and Q_m to the single mixed flow out of the faucet.
- T_c, T_h, and T_m refer to the temperatures of the cold water, hot water, and mixed water respectively.
- The variable C_t denotes the control signal for temperature that is input to the faucet, and C_f denotes the control signal for waterflow.
- The controls C_t and C_f are to influence the water flows Q_c and Q_h, thereby indirectly influencing the faucet output flow Q_m and temperature T_m.



Given this functional specification for the new design problem **CADET** searches its library for stored cases whose functional descriptions match the design problem.

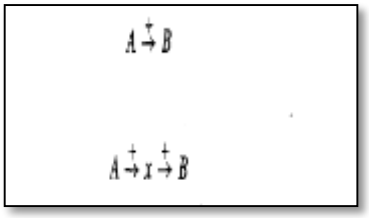
If an exact match is found, indicating that some stored case implements exactly the desired function, then this case can be returned as a suggested solution to the design problem.

If no exact match occurs, **CADET** may find cases that match various sub graphs of the desired functional specification.

In Figure 8.3, for example, the T-junction function matches a subgraph of the water faucet function graph.

More generally, **CADET** searches for subgraph isomorphisms between the two function graphs, so that parts of a case can be found to match parts of the design specification. Furthermore, the system may elaborate the original function specification graph in order to create functionally equivalent graphs that may match still more cases. It uses general knowledge about physical influences to create these elaborated function graphs.

For example, it uses a rewrite rule that allows it to rewrite the influence



This rewrite rule can be interpreted as stating that if B must increase with A, then it is sufficient to find some other quantity ***x*** such that B increases with ***x,*** and ***x*** increases with A. Here x is a universally quantified variable whose value is bound when matching the function graph against the case library. In fact, the function graph for the faucet shown in Figure 8.3 is an elaboration of the original - functional specification produced by applying such rewrite rules.
It is instructive to examine the correspondence between the problem setting of CADET and the general setting for instance-based methods such as k-NEAREST NEIGHBOR.

In CADET each stored training example describes a function graph along with the structure that implements it.

- New queries correspond to new function graphs.

- Thus, we can map the CADET problem into our standard notation by defining the space of instances X to be the space of all function graphs.
- The target function f maps function graphs to the structures that implement them.
- Each stored training example (x, f (x)) is a pair that describes some function graph x and the structure f ***(x)*** that implements x.
- The system must learn from the training example cases to output the structure f (x,) that successfully implements the input function graph query x,.

The above sketch of the CADET system illustrates several generic properties of case-based reasoning systems that distinguish them from approaches such as k-NEARESNTE IGHBOR.

- Instances or cases may be represented by rich symbolic descriptions, such as the function graphs used in CADET. This may require a similarity metric different from Euclidean distance, such as the size of the largest shared subgraph between two function graphs.

- Multiple retrieved cases may be combined to form the solution to the new problem. This is similar to the k NEARESNTE IGHBOaRp proach, in that multiple similar cases are used to construct a response for the new query.
- However, the process for combining these multiple retrieved cases can be very different, relying on knowledge-based reasoning rather than statistical methods.

- There may be a tight coupling between case retrieval, knowledge-based reasoning, and problem solving. One simple example of this is found in CADET, which uses generic knowledge about influences to rewrite function graphs during its attempt to find matching cases. Other systems have been developed that more fully integrate case-based reasoning into general search based problem-solving systems.

To summarize, case-based reasoning is an instance-based learning method in which instances (cases) may be rich relational descriptions and in which the retrieval and combination of cases to solve the current query may rely on knowledge based reasoning and search-intensive problem-solving methods.

- One current research issue in case-based reasoning is to develop improved methods for indexing cases. The central issue here is that syntactic similarity measures (e.g., subgraph isomorphism between function graphs) provide only an approximate indication of the relevance of a particular case to a particular problem.
- When the CBR system attempts to reuse the retrieved cases it may uncover difficulties that were not captured by this syntactic similarity measure. For example, in CADET the multiple retrieved design fragments may turn out to be incompatible with one another, making it impossible to combine them into a consistent final design. When this occurs in general, the CBR system may backtrack and search for additional cases, adapt the existing cases, or resort to other problem-solving methods.
- Importantly,when such difficulties are detected they also provide training data for improving the similarity metric or, equivalently, the indexing structure for the case library.
- In particular, if a case is retrieved based on the similarity metric, but found to be irrelevant based on further analysis, then the similarity metric should be refined to reject this case for similar subsequent queries.

8.6 REMARKS ON LAZY AND EAGER LEARNING:

Eager vs. Lazy Learning:

- Eager learning methods construct general, explicit description of the target function based on the provided training examples.
- Lazy learning methods simply store the data and generalizing beyond these data is postponed until an explicit request is made.
- Lazy learning methods can construct a different approximation to the target function for each encountered query instance.
- Eager learning methods use the same approximation to the target function, which must be learned based on training examples and before input queries are observed.

In this chapter we considered three lazy learning methods:
The k-NEAREST NEIGHBOR algorithm, locally weighted regression, and case-based reasoning.
We call these methods lazy because they defer the decision of how to generalize beyond the training data until each new query instance is encountered.
We also discussed one ***eager*** learning method: the method for learning radial basis function networks.
We call this method eager because it generalizes beyond the training data before observing the new query, committing at training time to the network structure and weights that define its approximation to the target function.

Let us distinguish between two kinds of differences:

- Differences in computation time and differences in the classifications produced for new queries.
- There are obviously differences in computation time between eager and lazy methods.
- For example, lazy methods will generally require less computation during training, but more computation when they must predict the target value for a new query.

The more fundamental question is whether there are essential differences in the inductive bias that can be achieved by lazy versus eager methods. The key difference between lazy and eager methods in this regard is

- Lazy methods may consider the query instance ***x,*** when deciding how to generalize beyond the training data D.

- Eager methods cannot. By the time they observe the query instance ***x,*** they have already chosen their (global) approximation to the target function.

Does this distinction affect the generalization accuracy of the learner?

- It does if we require that the lazy and eager learner employ the same hypothesis space H. To illustrate, consider the hypothesis space consisting of linear functions.
- The locally weighted linear regression algorithm discussed earlier is a lazy learning method based on this hypothesis space. For each new query ***x,*** it generalizes from the training data by choosing a new hypothesis based on the training examples near ***x,.***
- In contrast, an eager learner that uses the same hypothesis space of linear functions must choose its approximation before the queries are observed. The eager learner must therefore commit to a single linear function hypothesis that covers the entire instance space and all future queries.

- The lazy method effectively uses a richer hypothesis space because it uses many different local linear functions to form its implicit global approximation to the target function. Note this same situation holds for other learners and hypothesis spaces as well.

- The key point in the above paragraph is that a lazy learner has the option of (implicitly) representing the target function by a combination of many local approximations, whereas an eager learner must commit at training time to a single global approximation. The distinction between eager and lazy learning is thus related to the distinction between global and local approximations to the target function.

Can we create eager methods that use multiple local approximations to achieve the same effects as lazy local methods?

- Radial basis function networks can be seen as one attempt to achieve this. The RBF learning methods we discussed are eager methods that commit to a global approximation to the target function at training time.
- However, an RBF network represents this global function as a linear combination of multiple local kernel functions. Nevertheless, because RBF learning methods must commit to the hypothesis before the query point is known, the local approximations they create are not specifically targeted to the query point to the same degree as in a lazy learning method. Instead, RBF networks are built eagerly from local approximations centered around the training examples, or around clusters of training examples, but not around the unknown future query points.

Eager vs. Lazy Learning:

Lazy	**Eager**
Defer the decision of how to generalize beyond the training data until each new query instance is encountered	Generalizes beyond the training data before observing the new query, committing at training time to the network structure and weights that define its approximation to the target function.
Local Approximation to the target function	Global Approximation to the target function
Inductive Bias: Lazy methods may consider the query instance Xq, when deciding how to generalize beyond the training data D.	**Inductive Bias:** Eager methods will have already built the global approximation before observing the query instance Xq.
Variation in Generalization Accuracy: A lazy learner has the option of (implicitly) representing the target function bya combination of many local approximations. Better accuracy for unseen instances because they are observed before approximation.	**Variation in Generalization Accuracy:** An eager learner must commit at training time to a single global approximation.

To summarize,

- Lazy methods have the option of selecting a different hypothesis or local approximation to the target function for each query instance.
- Eager methods using the same hypothesis space are more restricted because they must commit to a single hypothesis that covers the entire instance space. Eager methods can, of course, employ hypothesis spaces that combine multiple local approximations, as in RBF networks.
- However, even these combined local approximations do not give eager methods the full ability of lazy methods to customize to unknown future query instances.

UNIT-WISE QUESTIONS:

1. What are instance based learning? Explain key features and disadvantages of these methods.
2. Explain the K – nearest neighbor algorithm with example. Explain remarks on k- nearest neighbor algorithm.
3. Explain the K – nearest neighbour algorithm for approximating a discrete – valued function and continues valued target function
4. Describe the concept of Locally Weighted Linear Regression and obtain the rule for gradient descent.
5. Explain the concept of Locally Weighted Linear Regression descent. Explain remarks on Locally Weighted Regression
6. Explain CADET System using Case based reasoning.
7. Discuss radial basis functions.
8. Differentiate lazy and eager learners

   (OR)

   Discuss Remarks on lazy and eager learning

UNIT-5
Learning Set of Rules

- **Sequential covering algorithms,**
- **Learning Rule Sets,**
- **Learning First Order Rules,**
- **Learning Sets of First Order Rules,**
- **Induction as Inverted Deduction,**
- **Inverting Resolution.**

INTRODUCTION:

One of the most expressive and human readable representations for learned hypotheses is sets of if-then rules. This chapter explores several algorithms for learning such sets of rules. One important special case involves learning sets of rules containing variables, called first-order Horn clauses. Because sets of first-order Horn clauses can be interpreted as programs in the logic programming language PROLOG learning them is often called inductive logic programming (ILP).

As an example of first-order rule sets, consider the following two rules that jointly describe the target concept ***Ancestor.***
Here we use the predicate

Parent(x, y) to indicate that y is the mother or father of ***x,*** and the predicate
Ancestor(x, y) to indicate that ***y*** is an ancestor of ***x*** related by an arbitrary number of family generations.
IF Parent (x, y) THEN Ancestor(x,y)
IF ***Parent(x,*** z) A ***Ancestor(z, y) THEN Ancestor(x, y)***

10.2 SEQUENTIAL COVERING ALGORITHMS:

Here we consider a family of algorithms for learning rule sets based on the strategy of learning one rule, removing the data it covers, then iterating this process. Such algorithms are called ***sequential covering*** algorithms.
To elaborate, imagine we have a subroutine **LEARN-ONE-RULE** tha t accepts a set of positive and negative training examples as input, then outputs a single rule that covers many of the positive examples and few of the negative examples.
We require that this output rule have high accuracy, but not necessarily high coverage. By high accuracy, we mean the predictions it makes should be correct. By accepting low coverage, we mean it need not make predictions for every training example.

Given this LEARN-ONE-RULE subroutine for learning a single rule, one obvious approach to learning a set of rules is to invoke LEARN-ONE-RULE on all the available training examples, remove any positive examples covered by the rule it learns, then invoke it again to learn a second rule based on the remaining training examples. This procedure can be iterated as many times as desired to learn a disjunctive set of rules that together cover any desired fraction of the positive examples. This is called a ***sequential covering*** algorithm because it sequentially learns a set of rules that together cover the full set of positive examples. The final set of rules can then be sorted so that more accurate rules will be considered first when a new instance must be classified.

A prototypical sequential covering algorithm is described in Table 10.1.

SEQUENTIAL-COVERING(*Target_attribute*, *Attributes*, *Examples*, *Threshold*)

- *Learned_rules* ← {}
- *Rule* ← LEARN-ONE-RULE(*Target_attribute*, *Attributes*, *Examples*)
- while PERFORMANCE(*Rule*, *Examples*) > *Threshold*, do
  - *Learned_rules* ← *Learned_rules* + *Rule*
  - *Examples* ← *Examples* − {examples correctly classified by *Rule*}
  - *Rule* ← LEARN-ONE-RULE(*Target_attribute*, *Attributes*, *Examples*)
- *Learned_rules* ← sort *Learned_rules* accord to PERFORMANCE over *Examples*
- return *Learned_rules*

TABLE 10.1
The sequential covering algorithm for learning a disjunctive set of rules. LEARN-ONE-RULE must return a single rule that covers at least some of the *Examples*. PERFORMANCE is a user-provided subroutine to evaluate rule quality. This covering algorithm learns rules until it can no longer learn a rule whose performance is above the given *Threshold*.

This sequential covering algorithm is one of the most widespread approaches to learning disjunctive sets of rules. It reduces the problem of learning a disjunctive set of rules to a sequence of simpler problems, each requiring that a single conjunctive rule be learned. Because it performs a greedy search, formulating a sequence of rules without backtracking, it is not guaranteed to find the smallest or best set of rules that cover the training examples.

10.2.1 General to Specific Beam Search:

One effective approach to implementing LEARN-ONE-RULE is to organize the hypothesis space search in the same general fashion as the ID3 algorithm, but to follow only the most promising branch in the tree at each step.

As illustrated in the search tree of Figure 10.1, the search begins by considering the most general rule precondition possible (the empty test that matches every instance), then greedily adding the attribute test that most improves rule performance measured over the training examples. Once this test has been added, the process is repeated by greedily adding a second attribute test, and so on. Like ID3, this process grows the hypothesis by greedily adding new attribute tests until the hypothesis reaches an acceptable level of performance.

Unlike ID3, this implementation of LEARN-ONE RULE follows only a single descendant at each search step-the attribute-value pair yielding the best performance-rather than growing a subtree that covers all possible values of the selected attribute.

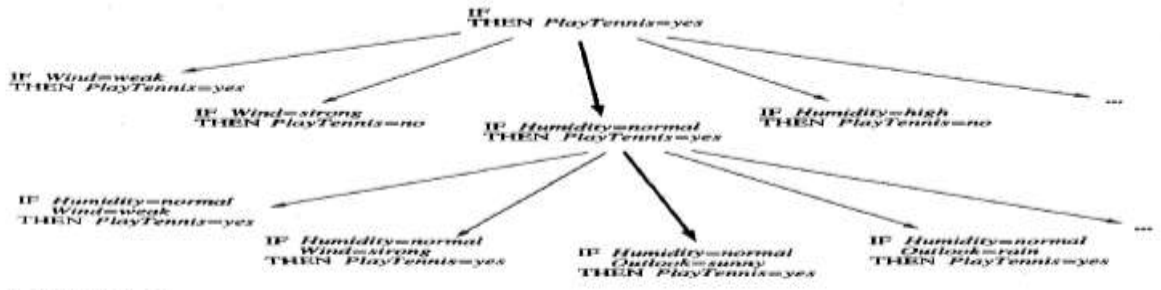


FIGURE 10.1
The search for rule preconditions as LEARN-ONE-RULE proceeds from general to specific. At each step, the preconditions of the best rule are specialized in all possible ways. Rule postconditions are determined by the examples found to satisfy the preconditions. This figure illustrates a beam search of width 1.

This approach to implementing LEARN-ONE-RULE forms a general-to specific search through the space of possible rules in search of a rule with high accuracy, though perhaps incomplete coverage of the data.
As in decision tree learning, there are many ways to define a measure to select the "best" descendant. The general-to-specific search suggested above for the LEARN-ONE-RULE algorithm is a greedy depth-first search with no backtracking. As with any greedy search, there is a danger that a suboptimal choice will be made at any step. To reduce this risk, we can extend the algorithm to perform a ***beam search;*** that is, a search in which the algorithm maintains a list of the k best candidates at each step, rather than a single best candidate.
On each search step, descendants (specializations) are generated for each of these k best candidates, and the resulting set is again reduced to the k most promising members. Beam search keeps track of the most promising alternatives to the current top-rated hypothesis, so that all of their successors can be considered at each search step. This general to specific beam search algorithm is used by the CN2 program described by Clark and Niblett (1989).

The algorithm is described in Table 10.2.

LEARN-ONE-RULE(*Target_attribute, Attributes, Examples, k*)

Returns a single rule that covers some of the Examples. Conducts a general_to_specific greedy beam search for the best rule, guided by the PERFORMANCE *metric.*

- Initialize *Best_hypothesis* to the most general hypothesis ∅
- Initialize *Candidate_hypotheses* to the set {*Best_hypothesis*}
- While *Candidate_hypotheses* is not empty, Do
  1. *Generate the next more specific candidate_hypotheses*
     - *All_constraints* ← the set of all constraints of the form $(a = v)$, where a is a member of *Attributes*, and v is a value of a that occurs in the current set of *Examples*
     - *New_candidate_hypotheses* ←
       for each h in *Candidate_hypotheses*,
       for each c in *All_constraints*,
       - create a specialization of h by adding the constraint c
     - Remove from *New_candidate_hypotheses* any hypotheses that are duplicates, inconsistent, or not maximally specific
  2. *Update Best_hypothesis*
     - For all h in *New_candidate_hypotheses* do
       - If (PERFORMANCE(h, *Examples*, *Target_attribute*)
         > PERFORMANCE(*Best_hypothesis*, *Examples*, *Target_attribute*))
         Then *Best_hypothesis* ← h
  3. *Update Candidate_hypotheses*
     - *Candidate_hypotheses* ← the k best members of *New_candidate_hypotheses*, according to the PERFORMANCE measure.
- Return a rule of the form
  "IF *Best_hypothesis* THEN *prediction*"
  where *prediction* is the most frequent value of *Target_attribute* among those *Examples* that match *Best_hypothesis*.

PERFORMANCE(h, *Examples*, *Target_attribute*)

- *h_examples* ← the subset of *Examples* that match h
- return $-Entropy(h\_examples)$, where entropy is with respect to *Target_attribute*

TABLE 10.2
One implementation for LEARN-ONE-RULE is a general-to-specific beam search. The frontier of current hypotheses is represented by the variable *Candidate_hypotheses*. This algorithm is similar to that used by the CN2 program, described by Clark and Niblett (1989).

10.2.2 Variations :

The SEQUENTIAL-COVERING algorithm together with the LEARN-ONE-RULE algorithm learns a set of if-then rules that covers the training examples. Many variations on this approach have been explored. For example, in some cases it might be desirable to have the program learn only rules that cover positive examples and to include a "default" that assigns a negative classification to instances not covered by any rule.
This approach might be desirable, say, if one is attempting to learn a target concept such as "pregnant women who are likely to have twins." In this case, the fraction of positive examples in the entire population is small, so the rule set will be more compact and intelligible to humans if it identifies only classes of positive examples, with the default classification of all other examples as negative.
This approach also corresponds to the "negation-as-failure" strategy of PROLOG in, which any expression that cannot be proven to be true is by default assumed to be false.

Another variation is provided by a family of algorithms called AQ (Michalski 1969, Michalski et al. 1986), that predate the CN2 algorithm on which the above discussion is based. Like CN2, AQ learns a disjunctive set of rules that together cover the target function. However, AQ differs in several ways from the algorithms given here.
First, the covering algorithm of AQ differs from the SEQUENTIAL-COVERalIgNoGri thm because it explicitly seeks rules that cover a particular target value, learning a disjunctive set of rules for each target value in turn.
Second, AQ's algorithm for learning a single rule differs from LEARN-ONERULE. While it conducts a general-to-specific beam search for each rule, it uses a single positive example to focus this search.

10.3 LEARNING RULE SETS: SUMMARY:

This section considers several key dimensions in the design space of such rule learning algorithms.

First, *sequential covering* algorithms learn one rule at a time, removing the covered examples and repeating the process on the remaining examples.
In contrast, decision tree algorithms such as ID3 learn the entire set of disjuncts simultaneously as part of the single search for an acceptable decision tree.
We might, therefore, call algorithms such as ID3 *simultaneous covering* algorithms, in cntrast to sequential covering algorithms such as CN2.
Which should we prefer?
The key difference occurs in the choice made at the most primitive step in the search.
At each search step ID3 chooses among alternative *attributes* by comparing the *partitions* of the data they generate.
In contrast, CN2 chooses among alternative *attribute-value* pairs, by comparing the *subsets* of data they cover. One way to see the significance of this difference is to compare the number of distinct choices made by the two algorithms in order to learn the same set of rules.
To learn a set of *n* rules, each containing k attribute-value tests in their preconditions, sequential covering algorithms will perform *n* . k primitive search steps, making an independent decision to select each precondition of each rule.
In contrast, simultaneous covering algorithms will make many fewer independent choices, because each choice of a decision node in the decision tree corresponds to choosing the precondition for the multiple rules

associated with that node. In other words, if the decision node tests an attribute that has m possible values, the choice of the decision node corresponds to choosing a precondition for each of the *m* corresponding rules (see Exercise 10.1).

Thus, sequential covering algorithms such as CN2 make a larger number of independent choices than simultaneous covering algorithms such as ID3.

Still, the question remains, which should we prefer?

The answer may depend on how much training data is available. If data is plentiful, then it may support the larger number of independent decisions required by the sequential covering algorithm, whereas if data is scarce, the "sharing" of decisions regarding preconditions of different rules may be more effective.

The second dimension along which approaches vary is the direction of the search in LEARN-ONE-RULE in the algorithm described above, the search is from ***general to specijic*** hypotheses.

Other algorithms we have discussed (e.g., FIND-S from Chapter 2) search from ***specijic to general.*** One advantage of general to specific search in this case is that there is a single maximally general hypothesis from which to begin the search, whereas there are very many specific hypotheses in most hypothesis spaces (i.e., one for each possible instance).

Given many maximally specific hypotheses, it is unclear which to select as the starting point of the search.

One program that conducts a specific-to-general search, called GOLEM (Muggleton and Feng 1990), addresses this issue by choosing several positive examples at random to initialize and to guide the search. The best hypothesis obtained through multiple random choices is then selected.

Third dimension is whether the LEARN-ONE-RULE search is a ***generate then test*** search through the syntactically legal hypotheses, as it is in our suggested implementation, or whether it is ***example-driven*** so that individual training examples constrain the generation of hypotheses. Prototypical example-driven search algorithms include the FIND-S and CANDIDATE-ELIMINA TE algorithm , AQ algorithm, and the CIGOL algorithm.

One important advantage of the generate and test approach is that each choice in the search is based on the hypothesis performance over ***many*** examples, so that the impact of noisy data is minimized. In contrast, example-driven algorithms that refine the hypothesis based on individual examples are more easily misled by a single noisy training example and are therefore less robust to errors in the training data.

A fourth dimension is whether and how rules are post-pruned. As in decision tree learning, it is possible for LEARN-ONE-RULE to formulate rules that perform very well on the training data, but less well on subsequent data.

As in decision tree learning, one way to address this issue is to post-prune each rule after it is learned from the training data. In particular, preconditions can be removed from the rule whenever this leads to improved performance over a set of pruning examples distinct from the training examples.

In general pruning is a process of removal of selected part of plant such as bud, branches and roots . In Decision Tree pruning does the same task it removes the branches of decision tree to overcome the overfitting condition of decision tree.

A final dimension is the particular definition of rule **PERFORMANCE** used to guide the search in **LEARN-ONE-RUlE Various** evaluation functions have been used.

Some common evaluation functions include:

- *Relative frequency.* Let n denote the number of examples the rule matches and let n_c denote the number of these that it classifies correctly. The relative frequency estimate of rule performance is

$$\frac{n_c}{n}$$

Relative frequency is used to evaluate rules in the AQ program.

- *m-estimate of accuracy.* This accuracy estimate is biased toward the default accuracy expected of the rule. It is often preferred when data is scarce and the rule must be evaluated based on few examples. As above, let n and n_c denote the number of examples matched and correctly predicted by the rule. Let p be the prior probability that a randomly drawn example from the entire data set will have the classification assigned by the rule (e.g., if 12 out of 100 examples have the value predicted by the rule, then $p = .12$). Finally, let m be the weight, or equivalent number of examples for weighting this prior p. The m-estimate of rule accuracy is

$$\frac{n_c + mp}{n + m}$$

Note if m is set to zero, then the m-estimate becomes the above relative frequency estimate. As m is increased, a larger number of examples is needed to override the prior assumed accuracy p. The m-estimate measure is advocated by Cestnik and Bratko (1991) and has been used in some versions of the CN2 algorithm. It is also used in the naive Bayes classifier discussed in Section 6.9.1.

- *Entropy.* This is the measure used by the PERFORMANCE subroutine in the algorithm of Table 10.2. Let S be the set of examples that match the rule preconditions. Entropy measures the uniformity of the target function values for this set of examples. We take the negative of the entropy so that better rules will have higher scores.

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

where c is the number of distinct values the target function may take on, and where p_i is the proportion of examples from S for which the target function takes on the ith value. This entropy measure, combined with a test for statistical significance, is used in the CN2 algorithm of Clark and Niblett (1989). It is also the basis for the information gain measure used by many decision tree learning algorithms.

10.4 LEARNING FIRST-ORDER RULES:

In the previous sections we discussed algorithms for learning sets of propositional (i.e., variable-free) rules. In this section, we consider learning rules that contain variables-in particular, learning first-order Horn theories.

Our motivation for considering such rules is that they are much more expressive than propositional rules. Inductive learning of first-order rules or theories is often referred to as ***inductive logic programming*** (or LP for short), because this process can be viewed as automatically inferring **PROLOG** programs from examples.

PROLOG is a general purpose, Turing-equivalent programming language in which programs are expressed as collections of Horn clauses.

10.4.1 First-Order Horn Clauses :

To see the advantages of first-order representations over propositional (variablefree) representations, consider the task of learning the simple target concept ***Daughter (x, y),*** defined over pairs of people ***x*** and ***y.*** The value of ***Daughter(x, y)*** is ***True*** when ***x*** is the daughter of ***y,*** and ***False*** otherwise.

Suppose each person in the data is described by the attributes ***Name, Mother, Father, Male, Female.*** Hence, each training example will consist of the description of two people in terms of these attributes, along with the value of the target attribute ***Daughter.***

For example, the following is a positive example in which Sharon is the daughter of Bob:

$$\begin{array}{lll} (Name_1 = Sharon, & Mother_1 = Louise, & Father_1 = Bob, \\ Male_1 = False, & Female_1 = True, & \\ Name_2 = Bob, & Mother_2 = Nora, & Father_2 = Victor, \\ Male_2 = True, & Female_2 = False, & Daughter_{1,2} = True) \end{array}$$

where the subscript on each attribute name indicates which of the two persons is being described. Now if we were to collect a number of such training examples for the target concept $Daughter_{1,2}$ and provide them to a propositional rule learner such as CN2 or C4.5, the result would be a collection of very specific rules such as

$$\begin{array}{ll} \text{IF} & (Father_1 = Bob) \wedge (Name_2 = Bob) \wedge (Female_1 = True) \\ \text{THEN} & Daughter_{1,2} = True \end{array}$$

Although it is correct, this rule is so specific that it will rarely, if ever, be useful in classifying future pairs of people. The problem is that propositional representations offer no general way to describe the essential *relations* among the values of the attributes. In contrast, a program using first-order representations could learn the following general rule:

$$\text{IF} \quad Father(y, x) \wedge Female(y), \quad \text{THEN} \quad Daughter(x, y)$$

where x and y are variables that can be bound to any person.

First-order Horn clauses may also refer to variables in the preconditions that do not occur in the postconditions. For example, one rule for *GrandDaughter* might be

$$\begin{array}{ll} \text{IF} & Father(y, z) \wedge Mother(z, x) \wedge Female(y) \\ \text{THEN} & GrandDaughter(x, y) \end{array}$$

Note the variable ***z*** in this rule, which refers to the father of ***y,*** is not present in the rule post conditions. Whenever such a variable occurs only in the preconditions, it is assumed to be existentially quantified; that is, the rule preconditions are satisfied as long as there exists at least one binding of the variable that satisfies the corresponding literal.

10.4.2 TERMINOLOGY:

Before moving on to algorithms for learning sets of Horn clauses, let us introduce some basic terminology from formal logic.

All expressions are composed of ***constants*** (e.g., ***Bob, Louise), variables*** (e.g., ***x, y), predicate*** symbols (e.g.,***Married, Greater-Than),*** and ***function*** symbols (e.g., ***age).***

The difference between predicates and functions is that predicates take on values of ***True*** or ***False,*** whereas functions may take on any constant as their value. We will use lowercase symbols for variables and capitalized symbols for constants.

Also, we will use lowercase for functions and capitalized symbols for predicates.

From these symbols, we build up expressions as follows: A *term* is any constant, any variable, or any function applied to any term (e.g., Bob, x, $age(Bob)$). A *literal* is any predicate or its negation applied to any term (e.g., $Married(Bob, Louise)$, $\neg Greater\_Than(age(Sue), 20)$). If a literal contains a negation ($\neg$) symbol, we call it a *negative literal*, otherwise a *positive literal*.

A *clause* is any disjunction of literals, where all variables are assumed to be universally quantified. A *Horn clause* is a clause containing at most one positive literal, such as

$$H \vee \neg L_1 \vee \ldots \neg L_n$$

where H is the positive literal, and $\neg L_1 \ldots \neg L_n$ are negative literals. Because of the equalities $(B \vee \neg A) = (B \leftarrow A)$ and $\neg(A \wedge B) = (\neg A \vee \neg B)$, the above Horn clause can alternatively be written in the form

$$H \leftarrow (L_1 \wedge \ldots \wedge L_n)$$

which is equivalent to the following, using our earlier rule notation

$$\text{IF } L_1 \wedge \ldots \wedge L_n, \text{ THEN } H$$

Whatever the notation, the Horn clause preconditions $L_1 \wedge \ldots \wedge L_n$ are called the clause *body* or, alternatively, the clause *antecedents*. The literal H that forms the postcondition is called the clause *head* or, alternatively, the clause *consequent*. For easy reference, these definitions are summarized in Table 10.3, along with other definitions introduced later in this chapter.

- Every well-formed expression is composed of *constants* (e.g., $Mary$, 23, or Joe), *variables* (e.g., x), *predicates* (e.g., $Female$, as in $Female(Mary)$), and *functions* (e.g., age, as in $age(Mary)$).
- A *term* is any constant, any variable, or any function applied to any term. Examples include $Mary$, x, $age(Mary)$, $age(x)$.
- A *literal* is any predicate (or its negation) applied to any set of terms. Examples include $Female(Mary)$, $\neg Female(x)$, $Greater\_than(age(Mary), 20)$.
- A *ground literal* is a literal that does not contain any variables (e.g., $\neg Female(Joe)$).
- A *negative literal* is a literal containing a negated predicate (e.g., $\neg Female(Joe)$).
- A *positive literal* is a literal with no negation sign (e.g., $Female(Mary)$).
- A *clause* is any disjunction of literals $M_1 \vee \ldots M_n$ whose variables are universally quantified.
- A *Horn clause* is an expression of the form

  $$H \leftarrow (L_1 \wedge \ldots \wedge L_n)$$

  where $H, L_1 \ldots L_n$ are positive literals. H is called the *head* or *consequent* of the Horn clause. The conjunction of literals $L_1 \wedge L_2 \wedge \ldots \wedge L_n$ is called the *body* or *antecedents* of the Horn clause.
- For any literals A and B, the expression $(A \leftarrow B)$ is equivalent to $(A \vee \neg B)$, and the expression $\neg(A \wedge B)$ is equivalent to $(\neg A \vee \neg B)$. Therefore, a Horn clause can equivalently be written as the disjunction

  $$H \vee \neg L_1 \vee \ldots \vee \neg L_n$$

- A *substitution* is any function that replaces variables by terms. For example, the substitution $\{x/3, y/z\}$ replaces the variable x by the term 3 and replaces the variable y by the term z. Given a substitution θ and a literal L we write $L\theta$ to denote the result of applying substitution θ to L.
- A *unifying substitution* for two literals L_1 and L_2 is any substitution θ such that $L_1\theta = L_2\theta$.

TABLE 10.3
Basic definitions from first-order logic.

10.5 LEARNING SETS OF FIRST-ORDER RULES: FOIL

A variety of algorithms has been proposed for learning first-order rules, or Horn clauses.
In this section we consider a program called FOIL (Quinlan 1990) that employs an approach very similar to the **SEQUENTIAL-COVER**ING and L**EARN-ONE RULE** algorithms of the previous section. In fact, the FOIL program is the natural extension of these earlier algorithms to first-order representations.
Formally, the hypotheses learned by FOIL are sets of first-order rules, where each rule is similar to a Horn clause with two exceptions.
First, the rules learned by FOIL are more restricted than general Horn clauses, because the literals are not permitted to contain function symbols (this reduces the complexity of the hypothesis space search).
Second, FOIL rules are more expressive than Horn clauses, because the literals appearing in the body of the rule may be negated. FOIL has been applied to a variety of problem domains.
For example, it has been demonstrated to learn a recursive definition of the QUICKSORT algorithm and to learn to discriminate legal from illegal chess positions.
The FOIL algorithm is summarized in Table 10.4:

FOIL(*Target_predicate*, *Predicates*, *Examples*)

- *Pos* ← those *Examples* for which the *Target_predicate* is *True*
- *Neg* ← those *Examples* for which the *Target_predicate* is *False*
- *Learned_rules* ← {}
- while *Pos*, do
  - *Learn a NewRule*
    - *NewRule* ← the rule that predicts *Target_predicate* with no preconditions
    - *NewRuleNeg* ← *Neg*
    - while *NewRuleNeg*, do
      - *Add a new literal to specialize NewRule*
        - *Candidate_literals* ← generate candidate new literals for *NewRule*, based on *Predicates*
        - *Best_literal* ← $\underset{L \in Candidate\_literals}{\text{argmax}}$ *Foil_Gain*(*L*, *NewRule*)
        - add *Best_literal* to preconditions of *NewRule*
        - *NewRuleNeg* ← subset of *NewRuleNeg* that satisfies *NewRule* preconditions
    - *Learned_rules* ← *Learned_rules* + *NewRule*
    - *Pos* ← *Pos* − {members of *Pos* covered by *NewRule*}
- Return *Learned_rules*

TABLE 10.4
The basic **FOIL** algorithm. The specific method for generating *Candidate_literals* and the definition of *Foil_Gain* are given in the text. This basic algorithm can be modified slightly to better accommodate noisy data, as described in the text.

The **outer loop** at each iteration adds a new rule to an overall disjunctive hypothesis (i.e.*rule*1 ∨*rule*2 ∨...)

This loop may be viewed as a specific-to-general search
– starting with the empty disjunctive hypothesis which covers no positive instances
– stopping when the hypothesis is general enough to cover all positive examples
The **inner loop** works out the detail of each specific rule, adding conjunctive constraints to the rule precondition on each iteration.

This loop may be viewed as a general-to-specific search

– starting with the most general precondition (empty)
– stopping when the hypothesis is specific enough to exclude all negative examples

The two most substantial differences between FOIL and our earlier SEQUENTIAL-COVERING AND LEARN-ONE-RULE algo rithm follow from the requirement that it accommodate first-order rules.

These differences are:

1. In its general-to-specific search to 'learn each new rule, FOIL employs different detailed steps to generate candidate specializations of the rule. This difference follows from the need to accommodate variables in the rule preconditions.
2. FOIL employs a PERFORMANCE measure, Foil-Gain, that differs from the entropy measure shown for LEARN-ONE-RULE in the table 10.2. This difference follows from the need to distinguish between different bindings of the rule variables and from the fact that FOIL seeks only rules that cover positive examples.

10.5.1 Generating Candidate Specializations in FOIL

- To generate candidate specializations of the current rule, FOIL generates a variety of new literals, each of which may be individually added to the rule preconditions. Suppose we are learning a rule of the form:

$$P(x_1,x_2,...,x_k) \leftarrow L_1 ... L_n$$

- Then candidate specializations add a new literal of the form:
  - $Q(v_1,...,v_r)$, where
    - Q is any predicate in the rule or training data;
    - at least one of the v_i in the created literal must already exist as a variable in the rule
  - $Equal(x_j,x_k)$, where x_j and x_k are variables already present in the rule; or
  - The negation of either of the above forms of literals

Example:

Suppose we are trying to learn $granddaughter(X,Y)$, given instances described by the predicates $father$ and $female$:

- FOIL starts with the most general rule:
  $granddaughter(X,Y) \leftarrow$
- The above procedure generates candidate additional literals:
  $equal(X,Y)$, $female(X)$, $female(Y)$, $father(X,Y)$, $father(Y,X)$, $father(X,Z)$, $father(Z,X)$, $father(Y,Z)$, $father(Z,Y)$ plus the negations of each of these
- Suppose FOIL now chooses $father(Y,Z)$ as most promising:
  $granddaughter(X,Y) \leftarrow father(Y,Z)$

New candidate literals to specialize this rule include those from above, plus $equal(Z,X)$,$equal(Z,Y)$, $female(Z)$, $female(Y)$, $father(Z,W)$, $father(W,Z)$ plus their negations

- If FOIL next selects $father(Z,X)$, then $female(Y)$ the rule $granddaughter(X,Y) \leftarrow father(Y,Z) \wedge father(Z,X) \wedge female(Y)$ is generated.
- Assuming this rule covers only positive examples, no more specializations for the current rule are sought. If more positive examples remain, search for another rule begins.

10.5.2 Guiding the Search in FOIL

- How do we decide which is the best literal to add when specialising a rule?
- To do this FOIL considers each possible binding of variables in the candidate rule specialisation to constants in the training examples.
- For example, suppose we have the training data: *granddaughter*(*Victor,Sharon*) *father*(*Sharon,Bob*) *father*(*tom, Bob*) *female*(*Sharon*) *father*(*Bob, Victor*) and we also assume ("closed world assumption") that any literals
  - involving predicates *granddaughter, father,* and *female*
  - involving constants *Victor, Sharon, Bob* and *tom* not in the training data are false. E.g. ¬*granddaughter*(*Tom, Bob*) is also true.
- Given the initial rule: $granddaughter(X,Y) \leftarrow$

FOIL considers all possible bindings of X and Y to the constants *Victor, Sharon, Bob* and *tom*. Note that only {X /*Victor*,Y / *Sharon*} is a positive binding (i.e. corresponds to a positive training example). The other 15 bindings of constants to X and Y are negative.

- At each stage of rule specialization, candidate specializations are preferred according towhether they possess more positive and fewer negative bindings. The precise evaluation measure used by FOIL is:

The precise evaluation measure used by FOIL is:

$$Foil\_Gain(L,R) \equiv t\left(\log_2 \frac{p_1}{p_1+n_1} - \log_2 \frac{p_0}{p_0+n_0}\right)$$

Where

- L is the candidate literal to add to rule R
- p_0 = number of positive bindings of R
- n_0 = number of negative bindings of R
- p_1 = number of positive bindings of $R+L$

- n_1 = number of negative bindings of $R+L$
- t is the number of positive bindings of R also covered by $R+L$

Note, in information-theoretic terms:

- $-\log_2 \frac{p_0}{p_0+n_0}$ is minimum number of bits to encode the classification of a positive binding covered by R
- $-\log_2 \frac{p_1}{p_1+n_1}$ is minimum number of bits to encode the classification of a positive binding covered by $R+L$
- t is number of positive bindings covered by R that remain covered by $R+L$
- So, $Foil\_Gain(R, L)$ is reduction due to L in total number of bits required to encode the classification of all positive bindings of R

10.5.4 Summary of FOIL

FOIL extends the SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms for propositional rule learning to first order rule learning
FOIL learns in two phases:

- an outer loop which acquires a disjunction of Horn clause-like rules which together coverthe positive examples

an inner loop which constructs individual rules by progressive specialisation of a rule through adding new literals selected according to the FOIL-gain measure until no negativeexamples are covered.
The literals introduced in the rule preconditions are drawn from the attributes used indescribing the training examples

- Variables used in new literals may be those occurring already in the rule pre-/postconditions, or they may be new
- If the new literal is allowed to use the target predicate then recursive rules may be learned In this case special care must be taken to avoid learning rule sets that produce infinite recursion

FOIL can add literals until no negative examples are covered only in the case of noise-freedata – otherwise some other strategy must be adopted
Quinlan proposed use of a minimum description length measure to stop rule extension when the description length of rules exceeds that of the training data they explain.

10.6 Induction as Inverted Deduction

A second approach to inductive logic programming is based on the observation that induction is the inverse of deduction

- More formally, induction is finding a hypothesis h such that

$$(\forall \langle x_i, f(x_i) \rangle \in D)\ B \wedge h \wedge x_i \vdash f(x_i)$$

where
  - D is training data
  - x_i is ith training instance
  - $f(x_i)$ is the target function value for x_i
  - B is other background knowledge

"For each training data instance, the instance's target classification is logically entailed by the background knowledge, together with hypothesis and the instance itself"

- Suggests we design an inductive algorithm by inverting operators for automated deduction . . .

- Ex: Concept to be learned: "pairs of people, $\langle u,v \rangle$ such that child of u is v"
  - Concept to be learned: "pairs of people, $\langle u, v \rangle$ such that child of u is v"
  - Suppose we are given
    - one training example
      - describing two people Tim and $Sharon$ in terms of their gender and their relation in terms of the predicate $father$
      - giving the target predicate $Child(Tim, Sharon)$ for these two people
    - background knowledge asserting that "if u is the father of v then u is a parent of v"

$$\begin{aligned} f(x_i): &\quad Child(Tim, Sharon) \\ x_i: &\quad Male(Tim), Female(Sharon), Father(Sharon, Tim) \\ B: &\quad Parent(u,v) \leftarrow Father(u,v) \end{aligned}$$

  - What h satisfies $(\forall \langle x_i, f(x_i) \rangle) \in D)\ B \wedge h \wedge x_i \vdash f(x_i)$?

$$\begin{aligned} h_1: &\quad Child(u,v) \leftarrow Father(v,u) \\ h_2: &\quad Child(u,v) \leftarrow Parent(v,u) \end{aligned}$$

$h_1 \wedge Father(Sharon, Tim) \vdash Child(Tim, Sharon)$

$Parent(u,v) \leftarrow Father(u,v) \wedge h_2 \wedge Father(Sharon, Tim) \vdash Child(Tim, Sharon)$

- There are pluses and minuses to casting the learning task as a search for hypotheses h such that

$$(\forall \langle x_i, f(x_i) \rangle \in D)\ (B \wedge h \wedge x_i) \vdash f(x_i)$$

- Positives:
  - Subsumes earlier idea of finding h that "fits" training data
  - Domain theory B helps define meaning of "fit" the data

$$B \wedge h \wedge x_i \vdash f(x_i)$$

  - Suggests algorithms that search H guided by B
- Negatives:
  - Doesn't naturally allow for noisy data – noise can result in inconsistent constraints in h and most logical frameworks break down when given inconsistent sets of assertions
  - First order logic gives a *huge* hypothesis space H
    - overfitting...
    - intractability of calculating all acceptable h's

    While using background knowledge B should help constrain hypothesis search, for manyILP systems hypothesis space search *increases* as B is increased.
- We are seeking mechanical *inductive* operators O such that
  $O(B,D) = h$ where $(\forall \langle x_i, f(x_i) \rangle \in D)\ (B \wedge h \wedge x_i) \vdash f(x_i)$

10.7 Inverting Resolution

A general method for automated deduction is the resolution rule.

- We are seeking mechanical *inductive* operators O such that

$$O(B, D) = h \text{ where } (\forall \langle x_i, f(x_i) \rangle \in D)\ (B \wedge h \wedge x_i) \vdash f(x_i)$$

- We already have mechanical *deductive* operators

$$F(A, B) = C, \text{ where } A \wedge B \vdash C$$

Can we invert these?

- The best known mechanical deductive operator is **resolution**

$$\begin{array}{lccc} C_1: & P & \vee & L \\ C_2: & \neg L & \vee & R \\ \hline \text{Resolvent:} & P & \vee & R \end{array}$$

Treating clauses as **sets** of literals (i.e. implicit disjunction) resolution is defined as follows:

1. Given initial clauses C_1 and C_2, find a literal L from clause C_1 such that $\neg L$ occurs in C_2
2. Form the resolvent C by including all literals from C_1 and C_2, except for L and $\neg L$. More precisely, the set of literals occurring in the conclusion C is

$$C = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})$$

where $\cup$ denotes set union, and "$-$" denotes set difference.

- Consider:

$$\begin{array}{ll} C_2 & Study \rightarrow KnowMaterial \\ C_1 & KnowMaterial \rightarrow PassExam \\ \hline C & Study \rightarrow PassExam \end{array} \equiv \begin{array}{l} \neg Study \vee KnowMaterial \\ \neg KnowMaterial \vee PassExam \\ \hline \neg Study \vee PassExam \end{array}$$

- So, to invert we need to go from C and C_1 to C_2. A general operation for doing this is (**inverted resolution**):
  1. Given initial clauses C_1 and C, find a literal L that occurs in clause C_1, but not in clause C.
  2. Form the second clause C_2 by including the following literals

$$C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}$$

10.7.1 First – Order Resolution

The resolution rule extends easily to first –order expressions. The process is based on the notion of unifying substitutions.

Like propositional resolution, first order resolution takes two clauses C_1 and C_2 as input an yields a third C as output.

Unlike propositional resolution the C_1 and C_2 must be related not by sharing a literal and its negation, but by sharing a literal and negated literal that can be matched by a **unifying substitution**.

Formally, first order resolution is defined as follows:

1. Find a literal L_1 from clause C_1, literal L_2 from clause C_2, and substitution θ such that $L_1\theta = \neg L_2\theta$
2. Form the resolvent C by including all literals from $C_1\theta$ and $C_2\theta$, except for $L_1\theta$ and $\neg L_2$ More precisely, the set of literals occurring in the conclusion C is

$$C = (C_1 - (L_1))\theta \cup (C_2 - (L_2))\theta$$

- Example:

$$\begin{array}{lll} C_1 & Swan(X) \rightarrow White(X) & \neg Swan(X) \vee White(X) \\ C_2 & Swan(fred) & = \quad Swan(fred) \\ \hline C & White(fred) & White(fred) \end{array}$$

Setting $\theta = \{X/fred\}$ we have

$$\begin{aligned} C &= (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta \\ &= (\{\neg Swan(X), White(X)\} - \{\neg Swan(X)\})\{X/fred\} \cup \\ &\quad (\{Swan(fred)\} - \{Swan(fred)\})\{X/fred\} \\ &= White(fred) \end{aligned}$$

10.7.2 Inverting Resolution: First –Order Case

- Can derive an inverse first order resolution operator by algebraic manipulation of the equation expressing the definition of the first order resolvent:

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$$

- First note the substitution θ can be factored into two substitutions θ_1 and θ_2 such that:
  - $\theta = \theta_1\theta_2$
  - θ_1 contains all and only variable bindings involving variables in C_1
  - θ_2 contains all and only variable bindings involving variables in C_2

  Since C_1 and C_2 are universally quantified they can be rewritten, if necessary, to contain no variables in common. Hence the above definition can be rewritten as:

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2$$

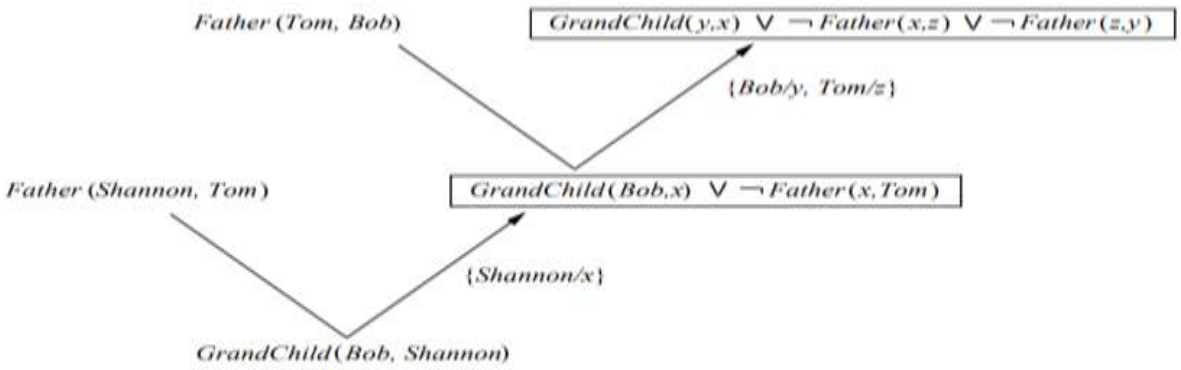
- If we restrict inverse resolution to infer clauses C_2 that contain **no** literals in common with C_1 (i.e. we prefer the shortest C_2's) then re-write above as:

$$C - (C_1 - \{L_1\})\theta_1 = (C_2 - \{L_2\})\theta_2$$

- Finally, noting $L_2 = \neg L_1\theta_1\theta_2^{-1}$ (by definition of the resolution rule) we get:

$$\textbf{Inverse Resolution: } C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\}$$

Example:



- Suppose we wish to learn rules for target predicate $GrandChild(y,x)$ given
  - training data $D = GrandChild(Bob, Shannon)$
  - background information $B = \{Father(Shannon, Tom), Father(Tom, Bob)\}$

- Proceed as follows:
  1. Set $C = GrandChild(Bob, Shannon)$
  2. Select $C_1 = Father(Shannon, Tom)$ from B
  3. For inverse resolution L_1 must be $Father(Shannon, Tom)$. Selecting $\theta_2^{-1} = \{Shannon/x\}$,

$$\begin{aligned} C_2 &= (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1 \theta_1 \theta_2^{-1}\} \\ &= \{GrandChild(Bob, x)\} \cup \{\neg Father(x, Tom)\} \end{aligned}$$

  4. Repeating, with appropriate choices for L_1 and θ_2^{-1} yields:

$$GrandChild(y, x) \leftarrow Father(x, z) \wedge Father(z, y)$$

10.7.3 Summary of Inverse Resolution

- The inverse resolution operation is non-deterministicIn general, for a given target predicate C
  - there are many ways to pick C_1 and L_1
  - many ways to pick the unifying substitutions θ_1 and θ_2
- Rule learning algorithms based on inverse resolution have been developed
  For example CIGOL uses sequential covering to iteratively learn a set of Horn clauses that covers positive examples
  - on each iteration a training example $(x_i, f(x_i))$not yet covered by rules is selected
  - inverse resolution is used to generate a candidate hypothesis h that satisfies

$$B \wedge h \wedge x_i \vdash f(x_i)$$

    where B is background knowledge plus clauses learned already
  - note that this is an example-driven search, though if multiple hypotheses cover the example, then the one with highest accuracy over further examples can be preferred
  - this contrasts with FOIL which uses a generate-then-test approach
- Another approach to using inverse resolution is PROGOL.

10.7.4 Generalization, θ-Subsumption and Entailment

- interesting to consider the relationship between the $more\_general\_than$ relation and inverse entailment
- $more\_general\_than$: $h_j \geq_g h_k$ iff $(\forall x \in X)[h_k(x) \rightarrow h_j(x)]$. A hypothesis can also be expressed as $c(x) \leftarrow h(x)$.
- $\theta - subsumption$: Consider two clauses C_j and C_k, both of the form $H \vee L_1 \vee \ldots \vee L_n$, where H is a positive literal and the L_i are arbitrary literals. Clause C_j is said to $\theta - subsume$ clause C_k iff $(\exists \theta)[C_j \theta \subseteq C_k]$.
- $Entailment$: Consider two clauses C_j and C_k. Clause C_j is said to $entail$ clause C_k (written $C_j \vdash C_k$) iff C_k follows deductively from C_j.

Relationship among these three definitions:

- if $h_1 \geq_g h_2$ then $C_1 : c(x) \leftarrow h_1(x)$ θ-subsumes $C_2 : c(x) \leftarrow h_2(x)$
- furthermore, θ-subsumption can hold even when the clauses have different heads

  $A : Mother(x, y) \leftarrow Father(x, z) \wedge Spouse(z, y)$

  $B : Mother(x, L) \leftarrow Father(x, B) \wedge Spouse(B, y) \wedge Female(x)$

  $A\theta \subseteq B$ if we choose $\theta = \{y/L, z/B\}$
- θ-subsumption is a special case of entailment

  $A : Elephant(father\_of(x)) \leftarrow Elephant(x)$

  $B : Elephant(father\_of(father\_of(y))) \leftarrow Elephant(y)$

  $A \vdash B$, but $\neg\exists\theta[A\theta \subseteq B]$

Generalization is a special case of θ-Subsumption
θ-Subsumption is a special case of entailment
In its most general form, inverse entailment produces intractable searches
θ-Subsumption provides a convenient notion that lies midway between generalization and entailment!

10.7.5 PROGOL

- Inverse resolution is one way to invert deduction to derive inductive generalisations.
  But, can easily lead to combinatorial explosion of candidate hypotheses due to multiplechoices for:
  - input clauses/literals for inverse resolution
  - unifying substitutions for inverse resolution
- PROGOL reduces this combinatorial explosion by using an alternative approach, called *modedirected inverse entailment (MDIE)*
  - Use inverse entailment to generate most specific h that together with backgroundinformation entails observed data
  - Then perform general-to-specific search through a hypothesis space H bounded by themost specific hypothesis and constrained by user-specified predicates
- More fully:
  1. User specifies H by stating predicates, functions, and forms of arguments allowed for each
  2. PROGOL uses sequential covering algorithm: For each $\langle x_i, f(x_i)\rangle$not covered by learned rules
     - Find most specific hypothesis h_i s.t. $B \wedge h_i \wedge x_i \vdash f(x_i)$
       * actually, considers only k-step entailment
     - Conduct general-to-specific search of H bounded by specific hypothesis h_i, choosinghypothesis with minimum description length
     - remove positive examples covered by new hypothesis

UNIT-WISE QUESTIONS:

1. Explain Sequential Covering Algorithm is suitable for learning a disjunctive set of rules.
2. Differentiate Simultaneous versus sequential algorithms
3. Discuss Learning Rule sets
4. Explain Learning First order rules

   (OR)
5. Explain First- order Horn Clauses
6. Discuss Learning sets of First - Order Rules (FOIL)
7. Discuss the concept of Induction as Inverted deduction
8. Explain Inverting Resolution

   (OR)
9. First Order Resolution

   (OR)
10. Inverting Resolution : First Order Case