

MODULE 1

INTRODUCTION TO MACHINE LEARNING

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”.

WELL-POSED LEARNING PROBLEMS

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

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.

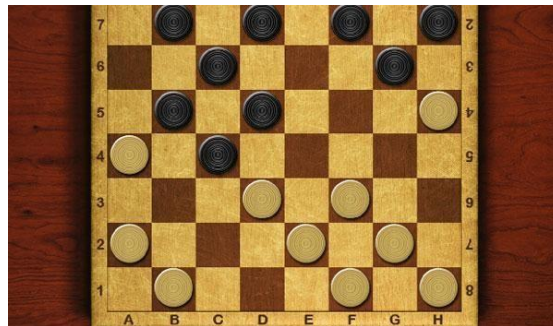


Fig: Checker game board

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

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

$$\text{ChooseMove} : B \rightarrow M$$

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 the system can successfully learn such a target function V , then it can easily use it to select the best move from any current board position.

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_1 : the number of black pieces on the board
- x_2 : the number of red pieces on the board
- x_3 : the number of black kings on the board
- x_4 : the number of red kings on the board
- x_5 : the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- x_6 : the number of red pieces threatened by black

Thus, learning program will represent as a linear function of the form

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

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 $V_{\text{train}}(b)$ for b .

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

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 $V_{\text{train}}(b)$ 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 $V_{\text{train}}(b)$ for any intermediate board state b to be $\hat{V}(\text{Successor}(b))$

Where ,

- $\hat{V}$ is the learner's current approximation to V
- $\text{Successor}(b)$ denotes the next board state following b for which it is again the program's turn to move

Rule for estimating training values

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

2. Adjusting the weights

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

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.

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

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 $(b, V_{\text{train}}(b))$

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

For each weight w_i , update it as

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

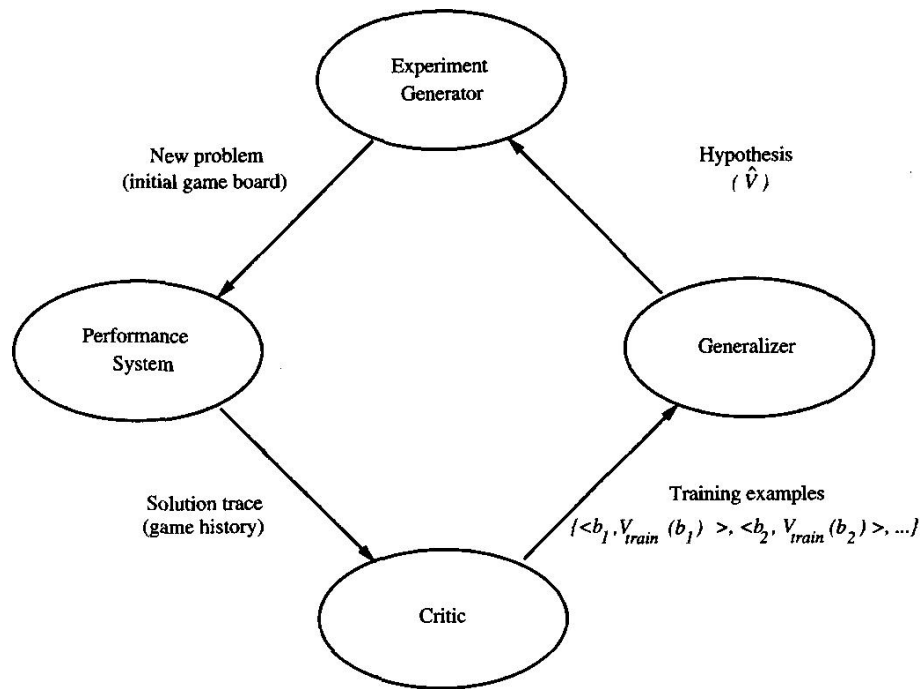
Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Working of weight update rule

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

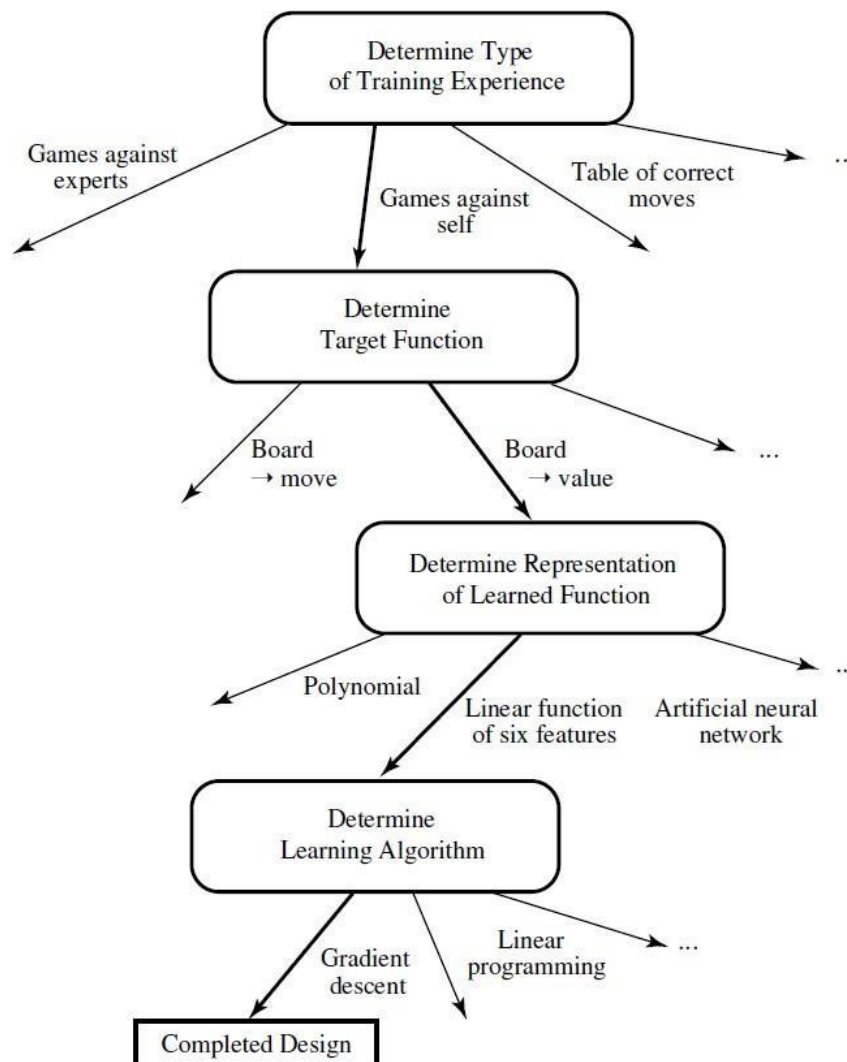
5. The Final Design

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



PERSPECTIVES AND ISSUES IN MACHINE LEARNING

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

- 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 of its input and output

A CONCEPT LEARNING TASK

Consider the example task of learning the target concept "Days on which *Aldo* enjoys his 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: Positive and negative training examples for the target concept *EnjoySport*.

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

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 humidity is represented by the expression

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

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

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

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

(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** : $X \rightarrow \{0, 1\}$
 - Training examples D : Positive and negative examples of the target function
 - Determine:
 - A hypothesis h in H such that $h(x) = c(x)$ for all x in X .
-

Table: The **EnjoySport** concept learning task.

The inductive learning hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

CONCEPT LEARNING AS SEARCH

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

Example:

Consider the instances X and hypotheses H in the *EnjoySport* 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

$$3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96 \text{ distinct instances}$$

$$5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120 \text{ syntactically distinct hypotheses within } H.$$

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

$$1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973. \text{ Semantically distinct hypotheses}$$

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

$$h_1 = (\text{Sunny}, ?, ?, \text{Strong}, ?, ?)$$

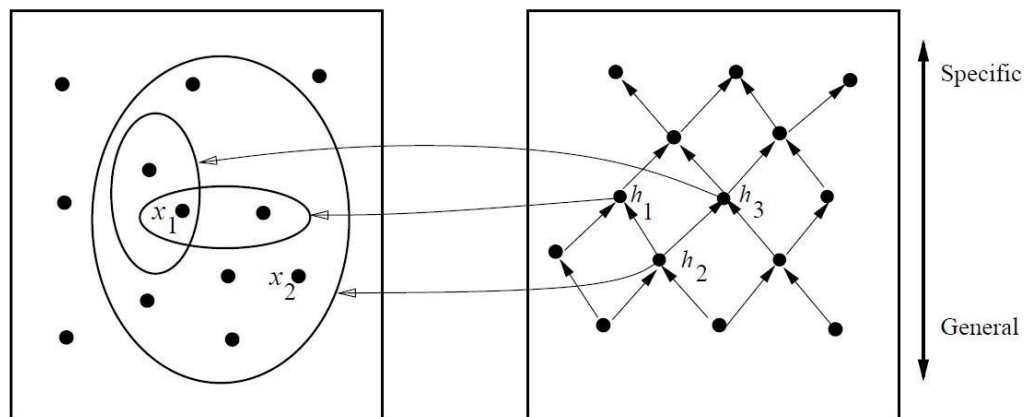
$$h_2 = (\text{Sunny}, ?, ?, ?, ?, ?)$$

- 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 to h_k if and only if any instance that satisfies h_k also satisfies h_j

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

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



$x_1 = \langle \text{Sunny, Warm, High, Strong, Cool, Same} \rangle$
 $x_2 = \langle \text{Sunny, Warm, High, Light, Warm, Same} \rangle$

$h_1 = \langle \text{Sunny, ?, ?, Strong, ?, ?} \rangle$
 $h_2 = \langle \text{Sunny, ?, ?, ?, ?, ?} \rangle$
 $h_3 = \langle \text{Sunny, ?, ?, ?, Cool, ?} \rangle$

- 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

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

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

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
 $h = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$

- Consider the first training example

$$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$$

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

$$h_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle$$

- Consider the second training example

$$x_2 = \langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle, +$$

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

$$h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$$

- Consider the third training example

$$x_3 = \langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle, -$$

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

$$h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$$

- Consider the fourth training example

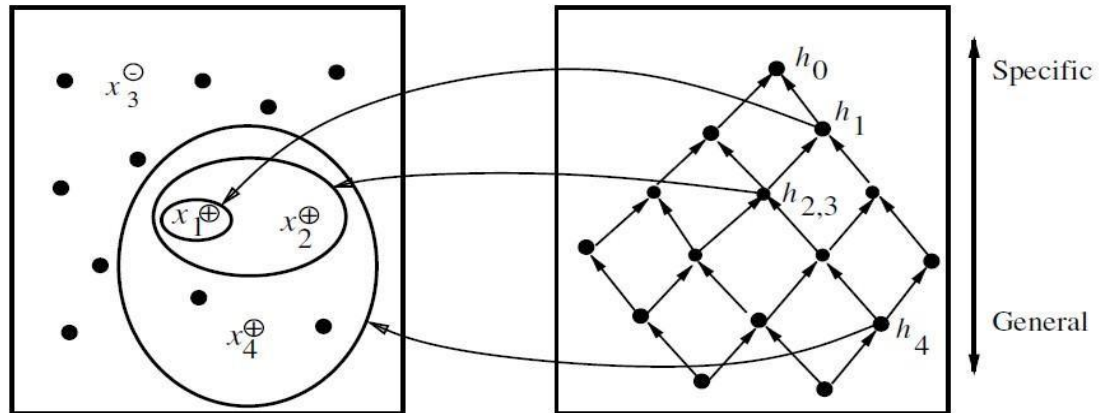
$$x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$$

The fourth example leads to a further generalization of h

$$h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$$

Instances X

Hypotheses H



$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$
 $x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle, +$
 $x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle, -$
 $x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$
 $h_1 = \langle \text{Sunny Warm Normal Strong Warm Sam} \rangle$
 $h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$
 $h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$
 $h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$

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?

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all *hypotheses consistent with the training examples*

Representation

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

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

Note difference between definitions of *consistent* and *satisfies*

- An example x is said to **satisfy** hypothesis h when $h(x) = 1$, regardless of whether x is a positive or negative example of the target concept.
- An example x is said to **consistent** with hypothesis h iff $h(x) = c(x)$

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

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

The LIST-THEN-ELIMINATION algorithm

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 $h(x) \neq c(x)$
 3. Output the list of hypotheses in **VersionSpace**
-

The LIST-THEN-ELIMINATE Algorithm

- 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 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.

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

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

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 .

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

Theorem: Version Space representation theorem

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 $\{(x, c(x))\}$. For all X, H, c , and D such that S and G are well defined,

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

To Prove:

1. Every h satisfying the right hand side of the above expression is in $VS_{H,D}$
2. Every member of $VS_{H,D}$ satisfies the right-hand side of the expression

Sketch of proof:

1. let g, h, s be arbitrary members of G, H, S respectively with $g \geq_g h \geq_g s$
 - 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 .
 - 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 $VS_{H,D}$.
2. 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.

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

CANDIDATE- ELIMINATION algorithm using version spaces

An Illustrative Example

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

CANDIDATE-ELIMINATION algorithm begins by initializing the version space to the set of all hypotheses in H ;

Initializing the G boundary set to contain the most general hypothesis in H

$$G_0 \langle ?, ?, ?, ?, ?, ? \rangle$$

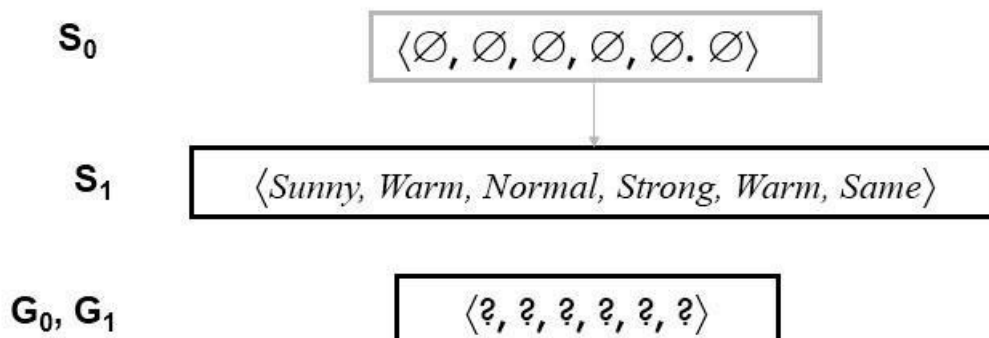
Initializing the S boundary set to contain the most specific (least general) hypothesis

$$S_0 \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

- When the first training example is presented, the CANDIDATE-ELIMINATION algorithm checks the S boundary and finds that it is overly specific and 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
- No update of the G boundary is needed in response to this training example because G_0 correctly covers this example

For training example d ,

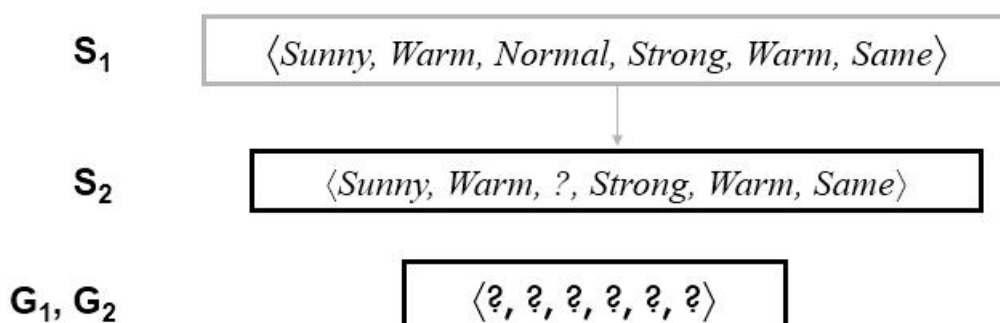
$$\langle \text{Sunny, Warm, Normal, Strong, Warm, Same} \rangle +$$



- When the second training example is observed, it has a similar effect of generalizing S further to S_2 , leaving G again unchanged i.e., $G_2 = G_1 = G_0$

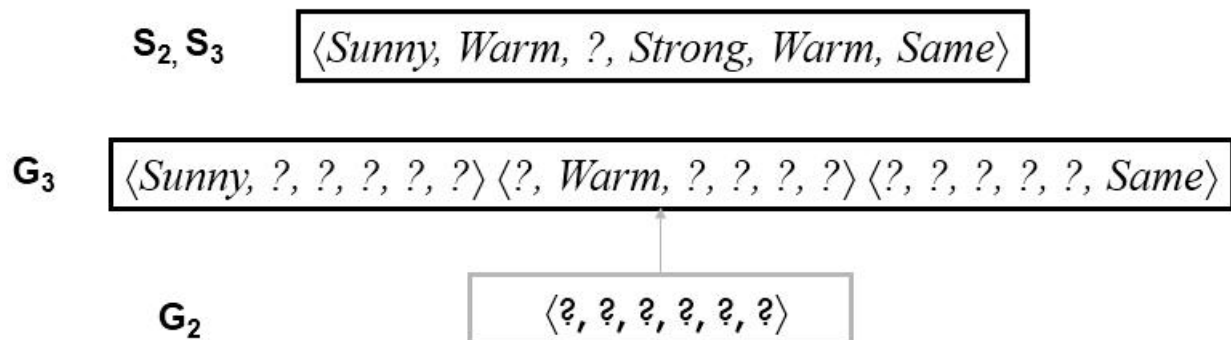
For training example d ,

$$\langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle +$$



- Consider the third training example. 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

For training example d, $\langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle -$

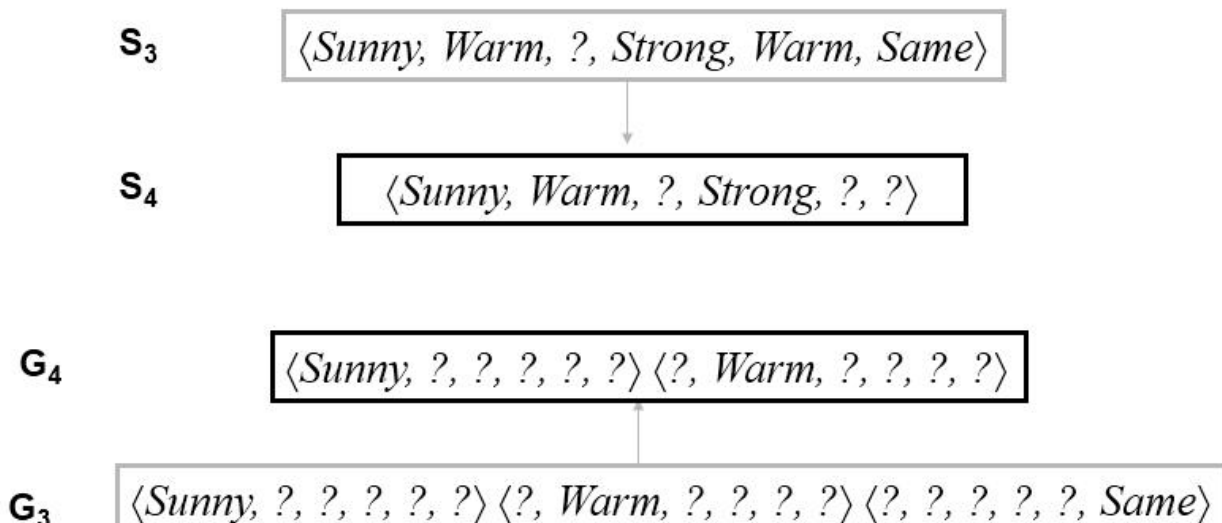


Given that there are six attributes that could be specified to specialize G_2 , why are there only three new hypotheses in G_3 ?

For example, the hypothesis $h = (\text{?, ?, Normal, ?, ?, ?})$ is a minimal specialization of G_2 that correctly labels the new example as a negative example, but it is not included in G_3 . The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples

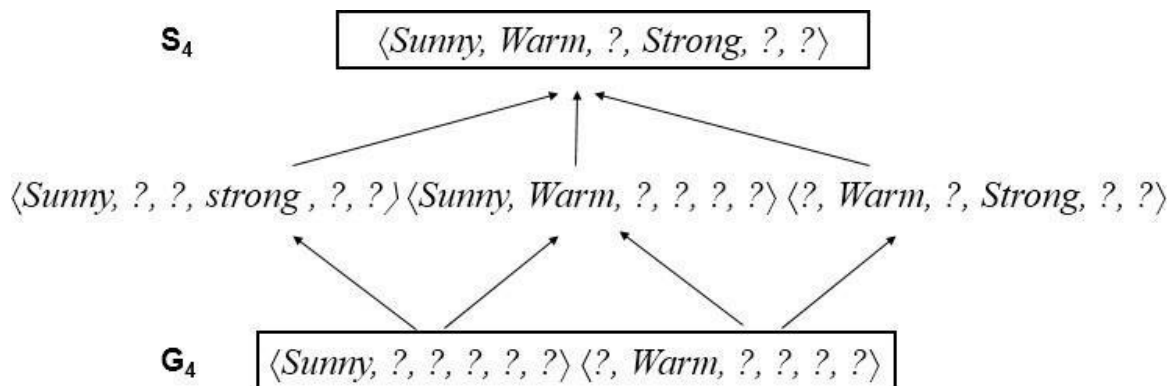
- Consider the fourth training example.

For training example d, $\langle \text{Sunny, Warm, High, Strong, Cool Change} \rangle +$



- 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 S_4 and G_4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



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?

These fundamental questions are examined in the context of the CANDIDATE-ELIMINATION algorithm

A Biased Hypothesis Space

- Suppose the target concept is not contained in the hypothesis space H , then obvious solution is to enrich the hypothesis space to include every possible hypothesis.
- Consider the *EnjoySport* example in which the hypothesis space is restricted 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."
- The following three training examples of disjunctive hypothesis, the algorithm would find that there are zero hypotheses in the version space

⟨Sunny Warm Normal Strong Cool Change⟩	Y
⟨Cloudy Warm Normal Strong Cool Change⟩	Y
⟨Rainy Warm Normal Strong Cool Change⟩	N

- If Candidate Elimination algorithm is applied, then it end up with empty Version Space. After first two training example
 $S = \langle ? \text{ Warm Normal Strong Cool Change} \rangle$
- This new hypothesis is overly general and it incorrectly covers the third negative training example! So H does not include the appropriate c .
- In this case, a more expressive hypothesis space is required.

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

$$(\text{Sunny}, ?, ?, ?, ?, ?) \vee (\text{Cloudy}, ?, ?, ?, ?, ?)$$

The Futility of Bias-Free Learning

Inductive learning requires some form of prior assumptions, or inductive bias

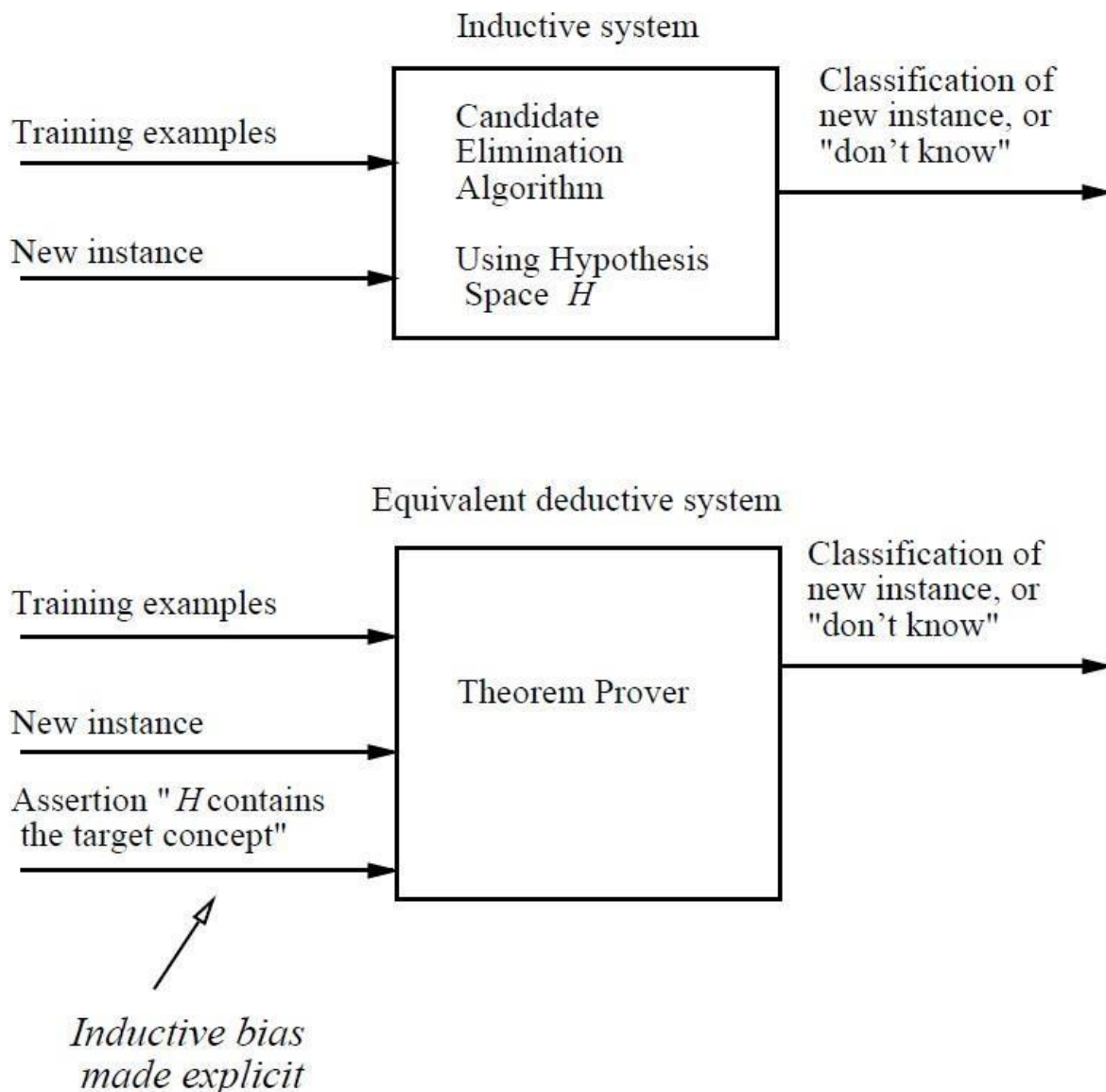
Definition:

Consider a concept learning algorithm L for the set of instances X .

- Let c be an arbitrary concept defined over X
- Let $D_c = \{(x, c(x))\}$ 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
- $(\forall \langle x_i \in X \rangle) [(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$

The below figure explains

- Modelling inductive systems by equivalent deductive systems.
- The input-output behavior of the CANDIDATE-ELIMINATION algorithm using a hypothesis space H is identical to that of a deductive theorem prover utilizing the assertion " H contains the target concept." This assertion is therefore called the inductive bias of the CANDIDATE-ELIMINATION algorithm.
- Characterizing inductive systems by their inductive bias allows modelling them by their equivalent deductive systems. This provides a way to compare inductive systems according to their policies for generalizing beyond the observed training data.



MODULE 2

Learning Sets of Rules

1. Introduction

Many machine learning problems require **learning a set of classification rules** instead of constructing a decision tree. Rule-based learning generates a collection of **IF–THEN rules** that are easy to understand and interpret.

One of the most popular methods for learning rule sets is the **Sequential Covering Algorithm**, also called the **Separate-and-Conquer Algorithm**.

Unlike decision tree algorithms that divide the entire dataset recursively (**divide-and-conquer**), Sequential Covering learns **one rule at a time**, removes the examples covered by that rule, and then repeats the process until all positive examples are covered.

2. What is Sequential Covering?

Definition

Sequential Covering is a rule-learning algorithm that repeatedly learns one rule that correctly classifies a subset of positive training examples, removes those examples from the training set, and continues until no uncovered positive examples remain.

The final classifier consists of a **set of rules** rather than a tree.

3. Why Sequential Covering?

Decision trees sometimes become very large and difficult to interpret.

Sequential Covering offers several advantages:

- Produces compact rule sets
- Handles overlapping classes
- Easy to modify individual rules
- More interpretable than complex trees
- Suitable for expert systems

4. Rule Representation

A rule has two parts.



General Form

```
IF <conditions>  
THEN <class>
```

Example

```
IF Outlook = Sunny  
AND Humidity = High  
  
THEN Play = No
```

Here,

Antecedent

```
Outlook = Sunny  
Humidity = High
```

Consequent

```
Play = No
```

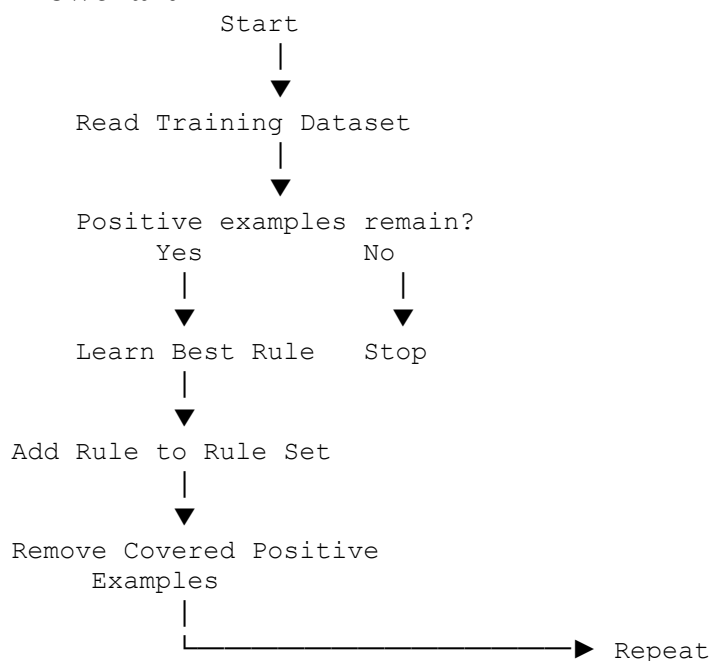
5. Working Principle

Sequential Covering follows a **greedy approach**.

Instead of learning all rules simultaneously, it learns **one rule**, removes covered positive examples, and repeats the process.

The procedure continues until every positive example has been covered.

Flowchart



6. Sequential Covering Algorithm

Algorithm

Input

Training examples

Output

Set of IF–THEN rules

Steps

1. Initialize an empty rule set.
2. Learn one rule that covers many positive examples and few negative examples.
3. Add the learned rule to the rule set.
4. Remove all positive examples covered by that rule.
5. Repeat until all positive examples are covered.
6. Return the complete rule set.

Pseudocode

SequentialCovering (Data)

RuleSet $\leftarrow \emptyset$

while PositiveExamplesExist (Data) do

 Rule $\leftarrow$ LearnOneRule (Data)

 RuleSet $\leftarrow$ RuleSet $\cup$ Rule

 Remove positive examples covered by Rule

end while

return RuleSet

7. Learn-One-Rule Procedure

The success of Sequential Covering depends on how a single rule is learned.

Initially,

IF TRUE

THEN Positive

This rule covers **all examples**, including many negatives.

Conditions are gradually added until the rule becomes sufficiently accurate.



Example

Initially

IF TRUE

THEN Play = Yes

↓

Add condition

IF Outlook = Sunny

THEN Play = Yes

↓

Add another condition

IF Outlook = Sunny

AND Humidity = Normal

THEN Play = Yes

The algorithm stops when the rule has acceptable accuracy.

8. Rule Evaluation

A rule is evaluated using several measures.

Coverage

Coverage indicates how many positive examples are covered.

Coverage = $\frac{\text{Positive examples covered}}{\text{Total positive examples}}$

Higher coverage is desirable.

Accuracy

Accuracy measures correctness.

Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$

where



TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall

$$\text{Recall} = \frac{TP}{TP + FN}$$

Information Gain

Many rule learners select the next condition using Information Gain.

$$\text{IG}(S) = \text{Entropy}(S) - \text{Entropy}(S | A)$$

Higher Information Gain indicates a better attribute.

9. Worked Example

Consider the following dataset.

Outlook	Humidity	Wind	Play
Sunny	High	Weak	No
Sunny	Normal	Weak	Yes
Overcast	High	Weak	Yes
Rain	Normal	Strong	No
Rain	Normal	Weak	Yes
Sunny	Normal	Strong	Yes

Positive Class = Play = Yes

Step 1

Initial Rule

IF TRUE

THEN Play = Yes



This covers every example.

Not acceptable because many negatives are included.

Step 2

Add best condition

IF Humidity = Normal

THEN Play = Yes

Coverage improves.

Still covers one negative example.

Step 3

Add another condition

IF Humidity = Normal

AND Wind = Weak

THEN Play = Yes

Now only positive examples remain.

Rule accepted.

Remove Covered Examples

Remove

Sunny Normal Weak

Rain Normal Weak

Remaining positive examples

Overcast High Weak

Sunny Normal Strong

Learn Second Rule

IF Outlook = Overcast

THEN Play = Yes

Remove covered example.



Learn Third Rule

IF Outlook = Sunny

AND Humidity = Normal

AND Wind = Strong

THEN Play = Yes

Final Rule Set

Rule 1

IF Humidity = Normal

AND Wind = Weak

THEN Play = Yes

Rule 2

IF Outlook = Overcast

THEN Play = Yes

Rule 3

IF Outlook = Sunny

AND Humidity = Normal

AND Wind = Strong

THEN Play = Yes

10. Computational Complexity

Suppose

- n = number of examples
- m = number of attributes

Learning one rule typically requires evaluating many candidate conditions, giving an approximate complexity of $O(nm)O(nm)O(nm)$. Since multiple rules are learned, the overall complexity is roughly proportional to the number of rules generated.

11. Advantages

- Easy to understand and interpret.
- Produces modular IF–THEN rules.
- Handles overlapping concepts.
- Easy to update by adding or removing rules.
- Suitable for expert systems and knowledge-based systems.
- Often generates smaller models than decision trees.

12. Disadvantages

- Greedy search may miss the optimal rule set.

- Rule order can affect classification.
- Sensitive to noisy training data.
- Can generate redundant rules.
- Learning may become expensive for very large datasets.

13. Applications

- Medical diagnosis
- Credit risk assessment
- Fraud detection
- Customer segmentation
- Intrusion detection systems
- Fault diagnosis
- Expert systems
- Weather prediction
- Loan approval
- Disease prediction

14. Sequential Covering vs Decision Tree

Feature	Sequential Covering	Decision Tree
Output	Rule Set	Tree
Strategy	Separate and Conquer	Divide and Conquer
Learning	One rule at a time	Entire tree
Interpretability	Very High	High
Rule Modification	Easy	Difficult
Handles Overlapping Classes	Better	Limited
Model Structure	Independent rules	Hierarchical tree

MODULE 3

Decision by Committee

1. Introduction

In machine learning, **a single classifier may not always provide the best prediction** because it may suffer from high variance, high bias, or overfitting. To overcome these limitations, multiple classifiers are combined to make a final decision. This approach is known as **Decision by Committee** or **Ensemble Learning**.

The fundamental idea is that **a group of learners usually performs better than a single learner**. Each learner contributes its prediction, and the final decision is obtained by combining these predictions.

This concept is similar to a committee of experts making a decision: rather than relying on one opinion, the committee reaches a more reliable conclusion through collective judgment.

2. What is Decision by Committee?

Definition

Decision by Committee is a machine learning technique in which **multiple models (committee members)** are trained, and their predictions are combined to produce a final decision.

Each model is called a **committee member** or **base learner**.

The final prediction is usually obtained by:

- Majority voting
- Weighted voting
- Averaging
- Probability estimation

3. Why Use Decision by Committee?

A single model may:

- Overfit the training data.

- Miss important patterns.
- Be sensitive to noise.
- Produce unstable predictions.

A committee of models can:

- Improve prediction accuracy.
- Reduce overfitting.
- Increase robustness.
- Improve generalization.

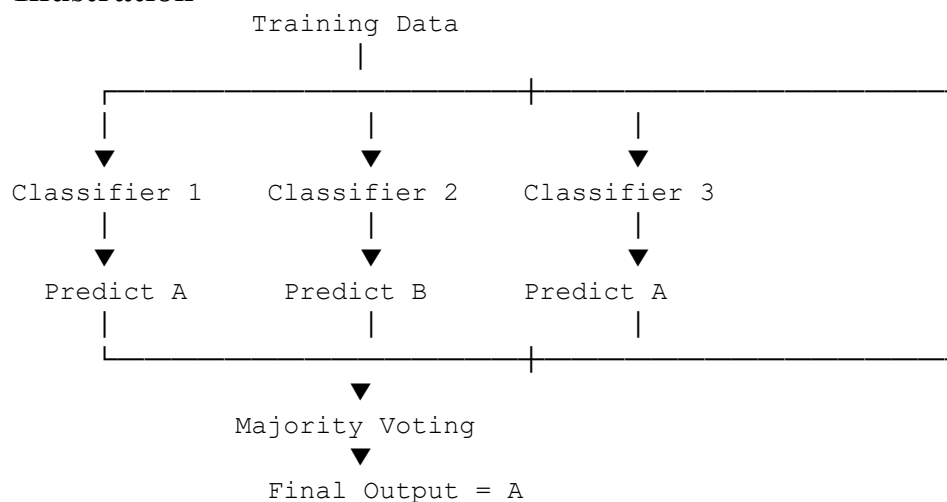
4. Basic Concept

Instead of training **one classifier**, train **many classifiers**.

Each classifier makes its own prediction.

The committee combines all predictions to obtain the final result.

Illustration



5. Ensemble Learning

Decision by Committee is also known as **Ensemble Learning**.

An **ensemble** is a collection of machine learning models whose predictions are combined.

Instead of relying on one model,

Prediction = $f(x)$

ensemble learning combines multiple models

Prediction = $f_1(x) + f_2(x) + \dots + f_n(x)$
 $f_1(x) + f_2(x) + \dots + f_n(x)$

The combination improves prediction quality.

6. Types of Ensemble Methods

There are three major committee methods.

1. Bagging

(Bootstrap Aggregating)

- Models are trained independently.
- Different random samples are used.
- Final prediction uses majority voting.

Example

Random Forest

2. Boosting

Models are trained sequentially.

Each new model learns from previous mistakes.

Examples

- AdaBoost
- Gradient Boosting
- XGBoost

3. Stacking

Different learning algorithms are trained.

Another model combines their outputs.

7. Committee Generation

Committee members can be generated by

Different Training Samples

Different subsets of data



Example

Bootstrap Sampling

Different Algorithms

Decision Tree

Support Vector Machine

Naïve Bayes

Neural Network

Different Parameters

Same algorithm

Different parameters

Example

Decision Tree

Depth = 5

Depth = 10

Depth = 15

Different Features

Each learner receives different features.

Used in Random Forest.

8. Majority Voting

The simplest committee method.

Each classifier votes.

Prediction receiving maximum votes wins.

Example



Suppose

Classifier 1 $\rightarrow$ Yes

Classifier 2 $\rightarrow$ No

Classifier 3 $\rightarrow$ Yes

Classifier 4 $\rightarrow$ Yes

Classifier 5 $\rightarrow$ No

Votes

Yes = 3

No = 2

Final Prediction

Yes

Mathematical Representation

Suppose

$h_1(x), h_2(x), h_3(x), \dots, h_n(x)$
 $h_1(x), h_2(x), h_3(x), \dots, h_n(x)$

are committee members.

Final prediction

$H(x) = \text{Majority}(h_1, h_2, \dots, h_n)$
 $H(x) = \text{Majority}(h_1, h_2, \dots, h_n)$

9. Weighted Voting

Sometimes

Some classifiers perform better.

They receive higher weights.

Example

Classifier	Weight
Tree	0.50



Classifier	Weight
SVM	0.30
Naïve Bayes	0.20

Prediction

Tree → Yes

SVM → Yes

Naïve Bayes → No

Weighted votes

Yes

$0.50 + 0.30$

=

0.80

No

0.20

Final Prediction

Yes

10. Averaging Method

Used in regression.

Instead of voting,

Average prediction is computed.

Example

House price prediction

Model 1

₹49 lakh

Model 2



₹52 lakh

Model 3

₹50 lakh

Prediction

$$49+52+50=50.33 \quad \frac{49+52+50}{3} = 50.33 \quad 49+52+50 = 50.33$$

11. Decision by Committee Algorithm

Input

Training data

Output

Committee classifier

Algorithm

Train multiple classifiers

For every test sample

Obtain prediction from every classifier

Combine predictions

Return final prediction

Pseudocode

Input : Training Dataset

Train Classifier1

Train Classifier2

Train Classifier3

For every Test Example

Predict using all classifiers

Final Prediction $\leftarrow$ Majority Vote

Return Final Prediction

12. Worked Example

Suppose

Five classifiers classify an email.



Model	Prediction
Tree	Spam
SVM	Spam
Naïve Bayes	Not Spam
Random Forest	Spam
Neural Network	Spam

Votes

Spam

4

Not Spam

1

Final Decision

Spam

13. Advantages

- ✓ Higher prediction accuracy
- ✓ Reduces overfitting
- ✓ Better generalization
- ✓ Robust to noisy data
- ✓ Improves stability
- ✓ Works well on large datasets
- ✓ Combines strengths of multiple learners

14. Disadvantages

- ✗ Computationally expensive
- ✗ Requires more memory
- ✗ Training time increases

- ✖ Difficult to interpret
- ✖ Model management becomes complex

15. Applications

Decision by Committee is widely used in

- Medical diagnosis
- Credit scoring
- Stock market prediction
- Image classification
- Speech recognition
- Fraud detection
- Weather forecasting
- Disease prediction
- Face recognition
- Intrusion detection

16. Real-Life Example

Imagine a college selecting the **Best Student**.

Instead of asking one teacher,

five teachers evaluate the student.

If four teachers recommend the student,

the student is selected.

Similarly,

multiple machine learning models decide together.

17. Comparison: Single Learner vs Decision by Committee

Feature	Single Learner	Decision by Committee
Number of Models	One	Multiple
Accuracy	Moderate	Higher
Overfitting	Higher	Lower
Robustness	Lower	Higher
Training Time	Low	High
Prediction Stability	Moderate	High

Feature	Single Learner	Decision by Committee
Complexity	Simple	Complex
Generalization	Moderate	Better

18. Summary

Decision by Committee, also known as Ensemble Learning, combines the predictions of multiple machine learning models to obtain a more accurate and reliable final decision. By aggregating diverse models through methods such as majority voting, weighted voting, or averaging, it reduces overfitting, improves robustness, and enhances generalization. Popular ensemble techniques include Bagging, Boosting, Random Forests, and Stacking, making Decision by Committee one of the most effective approaches for solving real-world classification and regression problems.

MODULE 4

Unsupervised Learning

1. Introduction

Machine Learning enables computers to learn patterns from data without being explicitly programmed. Based on the availability of labeled data, machine learning is broadly classified into:

1. **Supervised Learning**
2. **Unsupervised Learning**
3. **Semi-Supervised Learning**
4. **Reinforcement Learning**

In **Unsupervised Learning**, the training data does **not contain predefined output labels**. The objective is to discover hidden structures, relationships, or patterns in the data.

Unlike supervised learning, the algorithm is not told what the correct output is. Instead, it independently analyzes the input data and groups similar data points or discovers meaningful relationships.

2. Definition

Unsupervised Learning is a machine learning technique in which the algorithm learns from **unlabeled data** by identifying hidden structures, clusters, patterns, or relationships without any prior knowledge of the output classes.

Mathematical Representation

Given a dataset:

$$D = \{x_1, x_2, x_3, \dots, x_n\}$$

where each x_i is an input feature vector, the objective is to discover an unknown structure:

$$f(x) \rightarrow \text{Hidden Pattern}$$

No target variable y is provided.

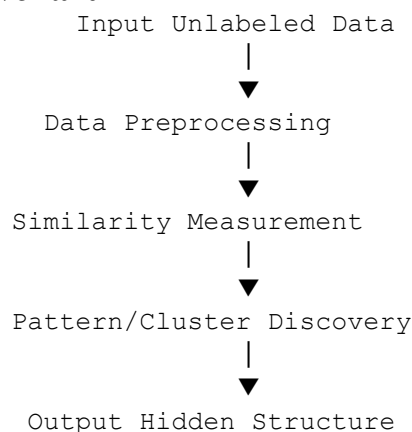
3. Characteristics of Unsupervised Learning

- Works with **unlabeled datasets**
- No predefined output classes
- Automatically identifies patterns
- Discovers hidden relationships
- Useful for exploratory data analysis
- Suitable for large datasets

4. Working Principle

The algorithm examines similarities and differences among data points and groups them based on common characteristics.

Flowchart



5. Types of Unsupervised Learning

Unsupervised learning is broadly classified into:

1. **Clustering**
2. **Association Rule Learning**
3. **Dimensionality Reduction**

6. Clustering

Definition

Clustering is the process of grouping similar data points into clusters such that:

- Data points within the same cluster are highly similar.
- Data points belonging to different clusters are significantly different.

Example

Suppose customer data contains:

- Age
- Income
- Spending Score

The algorithm may automatically identify:

- High-income customers
- Budget-conscious customers
- Frequent shoppers

without being told these categories in advance.

Types of Clustering

1. Partition-Based Clustering

Example:

- K-Means
- K-Medoids

2. Hierarchical Clustering

Creates a hierarchy of clusters.

Two approaches:

- Agglomerative (Bottom-Up)
- Divisive (Top-Down)

3. Density-Based Clustering

Groups densely connected points.

Example:

- DBSCAN

4. Model-Based Clustering

Assumes data follows a probability distribution.

Example:

- Gaussian Mixture Models (GMM)

7. K-Means Clustering

Definition

K-Means is one of the simplest and most widely used clustering algorithms.

It partitions the dataset into **K clusters**, where each cluster is represented by its centroid.

Algorithm

1. Select the number of clusters K .
2. Initialize K centroids randomly.
3. Assign each data point to the nearest centroid.
4. Recalculate the centroid of each cluster.
5. Repeat Steps 3–4 until centroids no longer change or a maximum number of iterations is reached.

Distance Formula

Euclidean Distance:

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Advantages

- Simple to implement
- Fast for large datasets
- Easy to interpret

Limitations

- Requires predefined K
- Sensitive to outliers
- May converge to a local optimum

8. Hierarchical Clustering

Definition

Hierarchical clustering builds a tree-like structure called a **dendrogram**.

Types

Agglomerative (Bottom-Up)

- Start with each data point as a separate cluster.
- Merge the closest clusters iteratively.

Divisive (Top-Down)

- Start with one large cluster.
- Split it recursively into smaller clusters.

Advantages

- No need to specify K
- Produces a hierarchy of clusters

Limitations

- Computationally expensive for large datasets

9. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Definition

DBSCAN groups points that are closely packed together and identifies outliers as noise.

Key Parameters

- **Epsilon (ϵ):** Neighborhood radius.
- **MinPts:** Minimum number of points required to form a dense region.

Advantages

- Detects clusters of arbitrary shape.
- Handles noise effectively.
- No need to specify the number of clusters.

Limitations

- Sensitive to parameter selection.
- Less effective when cluster densities vary significantly.

10. Association Rule Learning

Definition

Association rule learning discovers interesting relationships among variables in large datasets.

Example

In market basket analysis:



Bread → Butter

indicates that customers who buy bread often buy butter.

Evaluation Measures

Support

Support($A \rightarrow B$) = $\frac{\text{Transactions containing both A and B}}{\text{Total transactions}}$
 $\text{Support}(A \rightarrow B) = \frac{\text{Transactions containing both A and B}}{\text{Total transactions}}$

Confidence

Confidence($A \rightarrow B$) = $\frac{\text{Transactions containing both A and B}}{\text{Transactions containing A}}$
 $\text{Confidence}(A \rightarrow B) = \frac{\text{Transactions containing both A and B}}{\text{Transactions containing A}}$

Lift

Lift($A \rightarrow B$) = $\frac{\text{Confidence}(A \rightarrow B)}{\text{Support}(B)}$
 $\text{Lift}(A \rightarrow B) = \frac{\text{Confidence}(A \rightarrow B)}{\text{Support}(B)}$

Applications

- Market basket analysis
- Recommendation systems
- Retail analytics

11. Dimensionality Reduction

Definition

Dimensionality reduction reduces the number of input features while retaining most of the useful information.

Benefits

- Reduces computational cost
- Removes redundant features
- Improves visualization
- Reduces overfitting

Popular Techniques

- Principal Component Analysis (PCA)
- Singular Value Decomposition (SVD)
- t-SNE
- UMAP

12. Principal Component Analysis (PCA)

Definition

PCA transforms correlated variables into a smaller number of uncorrelated variables called **principal components**.

Steps

1. Standardize the data.
2. Compute the covariance matrix.
3. Calculate eigenvalues and eigenvectors.
4. Select the principal components with the largest eigenvalues.
5. Transform the original data.

Advantages

- Reduces dimensionality
- Removes redundancy
- Speeds up learning algorithms

Limitations

- Reduced interpretability
- Assumes linear relationships

13. Applications of Unsupervised Learning

- Customer segmentation
- Recommendation systems
- Image segmentation
- Social network analysis
- Fraud detection
- Medical diagnosis
- Document clustering
- Topic modeling
- Gene sequence analysis
- Anomaly detection

14. Advantages

- Does not require labeled data
- Identifies hidden structures
- Useful for exploratory analysis
- Discovers unknown patterns
- Can process large datasets

15. Limitations

- Difficult to evaluate results
- Sensitive to algorithm parameters
- Interpretation of clusters may be subjective
- Some algorithms are computationally expensive

16. Comparison: Supervised vs Unsupervised Learning

Feature	Supervised Learning	Unsupervised Learning
Data	Labeled	Unlabeled
Objective	Predict output	Discover patterns
Output	Known classes	Hidden structures
Examples	Classification, Regression	Clustering, Association, PCA
Evaluation	Accuracy, Precision, Recall	Silhouette Score, Davies - Bouldin Index, Cluster Validity

17. Summary

Unsupervised learning is a powerful branch of machine learning that analyzes **unlabeled data** to uncover hidden patterns, structures, and relationships. It is widely used for clustering, association rule mining, and dimensionality reduction. Popular algorithms such as K-Means, Hierarchical Clustering, DBSCAN, and PCA help solve real-world problems in customer segmentation, recommendation systems, anomaly detection, healthcare, and data visualization.

Sampling

1. Introduction

Machine learning algorithms are trained using data. In real-world applications, datasets may contain **millions of records**, making it computationally expensive and time-consuming to process the entire dataset.

Instead of using the complete dataset, a **representative subset**, called a **sample**, is selected. The process of selecting this subset is known as **sampling**.

A well-chosen sample preserves the characteristics of the original dataset, enabling machine learning models to learn effectively while reducing computational cost.

2. What is Sampling?

Definition

Sampling is the process of selecting a subset of observations from a larger dataset (population) so that the sample accurately represents the population for training and evaluating machine learning models.

Mathematical Representation

Let:

- $P = \{x_1, x_2, x_3, \dots, x_N\}$ $P = \{x_1, x_2, x_3, \dots, x_N\}$ be the population of size N .
- $S = \{x_1, x_2, \dots, x_n\}$ $S = \{x_1, x_2, \dots, x_n\}$ be a sample of size n , where $n < N$.

The objective of sampling is to ensure that:

$$S \approx P \quad \text{or} \quad P \approx S$$

meaning the sample reflects the important characteristics of the population.

3. Population vs Sample

Population	Sample
Complete collection of data	Subset of the population
Usually very large	Smaller and manageable
More expensive to process	Less expensive to process
Represents the entire dataset	Represents the population

Example

A university wants to study the academic performance of **10,000 students**.

- Population = All 10,000 students
- Sample = 1,000 randomly selected students

4. Why is Sampling Important?

Sampling is essential because:

- Large datasets require high computational resources.
- It reduces memory usage.
- It decreases training time.
- It allows faster experimentation.
- It improves efficiency in model development.
- It is useful when collecting the entire dataset is impractical.

5. Characteristics of a Good Sample

A good sample should:

- Represent the population accurately.
- Be free from bias.
- Include sufficient observations.
- Capture the variability of the population.



- Produce reliable results.

6. Types of Sampling

Sampling techniques are broadly classified into:

A. Probability Sampling

Each observation has a known probability of being selected.

1. Simple Random Sampling
2. Systematic Sampling
3. Stratified Sampling
4. Cluster Sampling

B. Non-Probability Sampling

The probability of selection is unknown.

1. Convenience Sampling
2. Judgment Sampling
3. Snowball Sampling
4. Quota Sampling

Machine learning primarily relies on **probability sampling** because it reduces bias.

7. Simple Random Sampling

Definition

Every observation has an equal probability of being selected.

Procedure

1. Assign a unique number to each observation.
2. Randomly select the required number of observations.
3. Use the selected observations as the sample.

Example

Suppose a dataset contains **1,000 customer records**.

To select a sample of **100 customers**, every customer has an equal chance of being selected.

Advantages

- Easy to implement
- Reduces selection bias



- Produces representative samples (when the population is homogeneous)

Limitations

- May not represent minority classes well
- Requires access to the complete dataset

8. Systematic Sampling

Definition

Systematic sampling selects every **k-th observation** after a random starting point.

Formula

$$k = \frac{N}{n}$$

where:

- N = Population size
- n = Sample size

Example

Population = 1,000

Sample size = 100

$$k = \frac{1000}{100} = 10$$

If the random start is 4, the selected observations are:

4, 14, 24, 34, ...

Advantages

- Simple
- Faster than random sampling

Limitations

- Can introduce bias if the data has periodic patterns

9. Stratified Sampling

Definition

The population is divided into **homogeneous groups (strata)**, and samples are selected from each group.

Example



A dataset contains:

- 60% Male
- 40% Female

A sample of 100 should ideally contain:

- 60 Males
- 40 Females

Advantages

- Preserves class distribution
- Improves model performance
- Reduces sampling bias

Applications

- Medical diagnosis
- Credit scoring
- Customer segmentation

10. Cluster Sampling

Definition

The population is divided into clusters, and entire clusters are selected randomly.

Example

A city contains 100 schools.

Instead of sampling students from every school, select 10 schools randomly and include all students from those schools.

Advantages

- Cost-effective
- Suitable for geographically distributed data

Limitations

- Selected clusters may not represent the entire population

11. Bootstrap Sampling

Definition

Bootstrap sampling selects observations **with replacement**, meaning the same observation may be selected multiple times.



Procedure

1. Select an observation randomly.
2. Return it to the dataset.
3. Repeat until the desired sample size is obtained.

Example

Original dataset:

A B C D

Bootstrap sample:

A C A D B C

Advantages

- Useful for estimating model uncertainty
- Forms the basis of **Bagging** and **Random Forest**

Applications

- Ensemble learning
- Model evaluation
- Confidence interval estimation

12. Sampling in Machine Learning Workflow

The dataset is usually divided into:

1. Training Set
2. Validation Set
3. Test Set

Typical Split

Dataset	Percentage
Training	70%
Validation	15%
Test	15%

Alternative splits:

- 80% / 20%
- 60% / 20% / 20%

Purpose

- **Training Set:** Used to train the model.
- **Validation Set:** Used to tune hyperparameters.
- **Test Set:** Used to evaluate the final model.

13. Cross-Validation

Cross-validation provides a more reliable estimate of model performance.

K-Fold Cross-Validation

1. Divide the dataset into **K equal folds**.
2. Train on $K-1$ folds.
3. Test on the remaining fold.
4. Repeat until every fold has been used as the test set.
5. Average the results.

Example

For **5-Fold Cross-Validation**:

- Fold 1 → Test
- Fold 2 → Test
- Fold 3 → Test
- Fold 4 → Test
- Fold 5 → Test

The final accuracy is the average of the five accuracies.

Advantages

- Better use of limited data
- Reduces variance in performance estimates

14. Sampling Bias

Definition

Sampling bias occurs when the sample does not accurately represent the population.

Causes

- Non-random selection
- Underrepresentation of certain groups
- Small sample size

Effects

- Poor model performance
- Reduced generalization

- Incorrect conclusions

15. Applications of Sampling

- Big data analytics
- Medical diagnosis
- Financial forecasting
- Image processing
- Customer behavior analysis
- Survey analysis
- Recommendation systems
- Fraud detection
- Quality control

16. Advantages of Sampling

- Reduces computational cost
- Saves memory
- Faster model training
- Enables efficient experimentation
- Makes large datasets manageable
- Reduces data collection costs

17. Limitations of Sampling

- Poor sampling may introduce bias.
- Small samples may not represent the population.
- Important information may be omitted.
- Incorrect sampling affects model accuracy.

18. Comparison of Sampling Techniques

Technique	Selection Method	Advantages	Limitations
Simple Random	Random selection	Easy, unbiased	May miss minority classes
Systematic	Every k-th observation	Fast	Sensitive to periodicity
Stratified	Sample from each stratum	Preserves class balance	Requires knowledge of strata
Cluster	Select entire clusters	Cost-effective	Clusters may not be representative
Bootstrap	Sampling with replacement	Supports ensemble methods	Duplicate observations may occur

19. Real-World Example

A bank has **1 million customer records** and wants to build a loan approval model.

Instead of training on all records, it:

1. Uses **stratified sampling** to maintain the ratio of approved and rejected loans.
2. Splits the sampled data into training, validation, and test sets.
3. Trains the model using the training set.
4. Tunes parameters using the validation set.
5. Evaluates performance using the test set.

This approach reduces computation while maintaining reliable performance.

MODULE 5

Graphical Models

Introduction

Many real-world problems involve **uncertainty**.

Examples:

- Disease diagnosis
- Weather prediction
- Robot navigation
- Speech recognition
- Spam filtering

Traditional machine learning models often assume complete information. However, real-world data is noisy, incomplete, or uncertain.

Probabilistic graphical models address this uncertainty by representing variables and their probabilistic relationships.

Two major probabilistic graphical models are:

- Bayesian Networks
- Markov Random Fields

2. Bayesian Networks

Definition

A **Bayesian Network (BN)** is a **Directed Acyclic Graph (DAG)** in which:

- Nodes represent random variables.
- Directed edges represent conditional dependencies.
- Each node has an associated Conditional Probability Table (CPT).

Mathematical Definition

Let

$$X = \{X_1, X_2, \dots, X_n\} \quad X = \{X_1, X_2, \dots, X_n\}$$

be a set of random variables.

The joint probability distribution is

$$P(X) = \prod_{i=1}^n P(X_i \mid \text{Parents}(X_i)) \quad P(X) = \prod_{i=1}^n P(X_i \mid \text{Parents}(X_i))$$

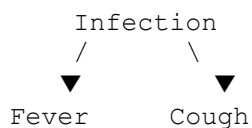
This factorization is the key advantage of Bayesian Networks.

Example

Disease Diagnosis

Variables:

- Fever
- Infection
- Cough



Interpretation

- Infection causes Fever.
- Infection causes Cough.
- Fever and Cough become conditionally independent once Infection is known.

Conditional Probability Table (CPT)

Example

Infection	Fever	Probability
Yes	Yes	0.9
Yes	No	0.1

Infection	Fever	Probability
No	Yes	0.2
No	No	0.8

Each node maintains a probability table.

Advantages

- Compact representation
- Handles uncertainty
- Supports probabilistic reasoning
- Efficient inference

Applications

- Medical diagnosis
- Bioinformatics
- Fault diagnosis
- Recommendation systems
- Decision support systems

Approximate Inference

Why Approximate Inference?

Exact inference in Bayesian Networks becomes computationally expensive when:

- Many variables exist
- Large CPTs
- Complex network structures

Exact inference is often NP-hard.

Approximate inference estimates probabilities instead of computing exact values.

Definition

Approximate inference refers to methods that estimate posterior probabilities using sampling or optimization techniques.

Types

Sampling Methods

- Monte Carlo Sampling
- Importance Sampling



- Likelihood Weighting
- Gibbs Sampling

Variational Methods

- Mean Field Approximation
- Variational Bayes

Monte Carlo Sampling

Random samples are generated.

Probability is estimated as

$$P(A) \approx \frac{\text{Number of samples satisfying } A}{\text{Total samples}}$$

Gibbs Sampling

One variable is updated at a time while keeping others fixed.

Algorithm

1. Initialize variables.
2. Update one variable.
3. Repeat.
4. Estimate probabilities.

Advantages

- Simple
- Good for large Bayesian Networks

Likelihood Weighting

Evidence variables remain fixed.

Remaining variables are sampled.

Each sample receives a weight.

Higher weights imply better estimates.

Comparison

Method	Speed	Accuracy
Exact	Slow	High

Method	Speed	Accuracy
Monte Carlo	Medium	Medium
Gibbs	Fast	Good
Variational	Very Fast	Approximate

Making Bayesian Networks

Steps

Step 1

Identify variables.

Example

Student Performance

Variables

- Attendance
- Study Hours
- Internal Marks
- Final Grade

Step 2

Determine dependencies.

Study Hours → Internal Marks

Attendance → Internal Marks

Internal Marks → Final Grade

Step 3

Draw DAG.

Step 4

Construct CPT.

Step 5

Validate Network.



Example

Weather Prediction

Variables

- Rain
- Cloud
- Wet Grass



Learning Bayesian Networks

Two approaches:

Structure Learning

Learn graph.

Methods

- Score-based
- Constraint-based

Parameter Learning

Learn probabilities.

Methods

- Maximum Likelihood
- Bayesian Estimation

Chapter 4: Markov Random Fields (MRFs)

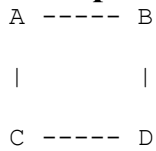
Definition

A **Markov Random Field** is an **undirected probabilistic graphical model**.

Unlike Bayesian Networks,

MRFs use **undirected edges**.

Graph



Markov Property

Each node depends only on its neighbors.

Joint Probability

$$P(X) = \frac{1}{Z} \prod \phi(C) \quad P(X) = \frac{1}{Z} \prod \phi(C)$$

where:

- Z is the partition function.
- $\phi(C)$ is the potential function over a clique C .

Potential Function

Potential functions measure compatibility among neighboring variables.

Applications

- Image segmentation
- Computer vision
- Protein structure prediction
- Object recognition

Advantages

- Represents symmetric relationships
- Suitable for spatial data
- Flexible modeling

Disadvantages

- Complex inference
- Partition function difficult to compute

Bayesian Network vs Markov Random Field

Bayesian Network	Markov Random Field
Directed Graph	Undirected Graph
DAG	Cyclic allowed
CPT	Potential Function
Causal relationships	Symmetric relationships
Easier interpretation	Better for image processing

Hidden Markov Models (HMMs)

Introduction

Many real-world problems involve sequences.

Examples

- Speech recognition
- Handwriting recognition
- DNA analysis
- Weather forecasting

Hidden Markov Models are designed for sequential data.

Definition

A Hidden Markov Model is a probabilistic model in which:

- States are hidden.
- Observations are visible.
- Hidden states generate observations.

Components

Hidden States

Unknown variables.

Example

Weather

Sunny

Rainy

Cloudy



Observations

Visible outputs.

Example

Umbrella

No Umbrella

Transition Probability

Probability of moving from one state to another.

$$A = P(S_j | S_i) \quad A = P(S_{j+1} | S_i) \quad A = P(S_{j+1} | S_i)$$

Emission Probability

Probability of observation.

$$B = P(O | S) \quad B = P(O | S) \quad B = P(O | S)$$

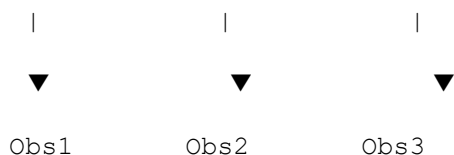
Initial Probability

Probability of starting state.

$$\pi = P(S_1) \quad \pi = P(S_1) \quad \pi = P(S_1)$$

HMM Diagram

State1 ---> State2 ---> State3



Three Fundamental Problems

1. Evaluation

Find probability of observation sequence.

Algorithm

Forward Algorithm.

2. Decoding

Find hidden states.



Algorithm

Viterbi Algorithm.

3. Learning

Estimate parameters.

Algorithm

Baum-Welch Algorithm.

Forward Algorithm

Computes

$$P(O|\lambda)P(O|\lambda)P(O|\lambda)$$

where

O = observation sequence

Viterbi Algorithm

Finds

Most likely hidden state sequence.

Dynamic programming algorithm.

Baum-Welch Algorithm

Expectation-Maximization algorithm.

Learns

- Transition probabilities
- Emission probabilities

Applications

- Speech recognition
- Machine translation
- POS tagging
- Gene prediction
- Gesture recognition

- Financial prediction
- Robot localization

Advantages

- Excellent for sequence modeling
- Efficient algorithms
- Handles uncertainty

DisadvantagesFirst-order Markov assumption

- Limited long-range dependency modeling
- Computationally intensive for large state spaces

Comparison of Bayesian Networks, MRFs, and HMMs

Feature	Bayesian Network	Markov Random Field	Hidden Markov Model
Graph Type	Directed Acyclic Graph	Undirected Graph	Directed Chain
Dependencies	Conditional	Neighborhood	Sequential
Probability Representation	Conditional Probability Tables	Potential Functions	Transition & Emission Probabilities
Best Suited For	Causal reasoning	Spatial relationships	Time-series and sequential data
Examples	Medical diagnosis	Image segmentation	Speech recognition

Summary

- **Bayesian Networks** represent probabilistic dependencies using **directed acyclic graphs** and conditional probability tables.
- **Approximate Inference** uses sampling or optimization methods such as Monte Carlo, Gibbs Sampling, and Variational Inference to estimate probabilities when exact inference is computationally expensive.
- **Making Bayesian Networks** involves identifying variables, determining dependencies, constructing a DAG, defining conditional probability tables, and learning parameters.
- **Markov Random Fields** model symmetric relationships with undirected graphs and potential functions, making them useful for image analysis and spatial data.



- **Hidden Markov Models** are specialized probabilistic models for sequential data, using hidden states and observable outputs, with key algorithms including the Forward, Viterbi, and Baum–Welch algorithms.