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.

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.

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”.

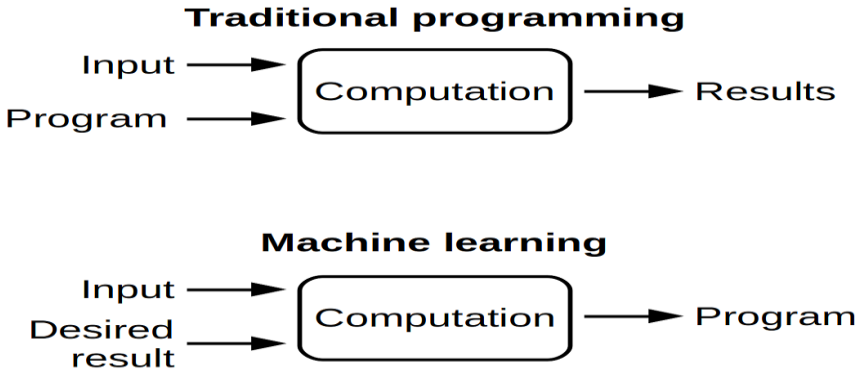
Areas of Influence for Machine Learning

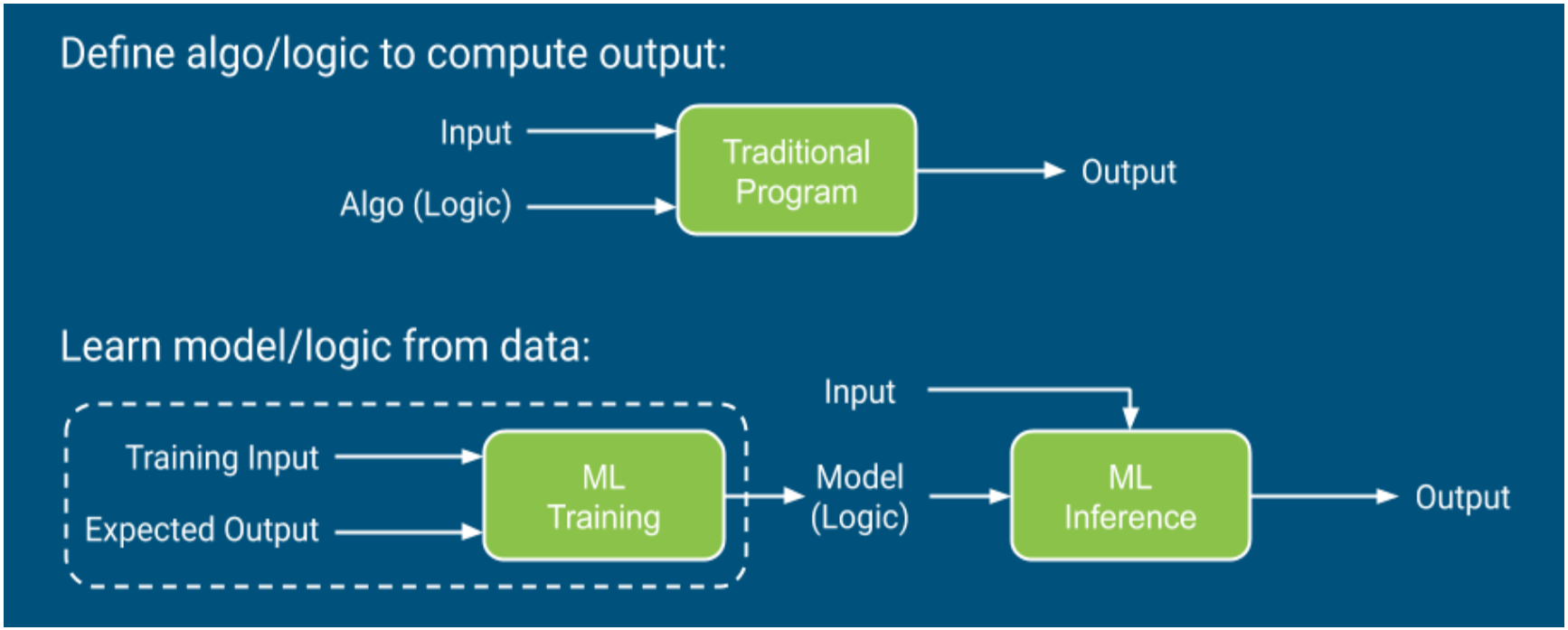
- **Statistics:** How best to use samples drawn from unknown probability distributions to help decide from which distribution some new sample is drawn?
- **Brain Models:** Non-linear elements with weighted inputs (Artificial Neural Networks) have been suggested as simple models of biological neurons.
- **Adaptive Control Theory:** How to deal with controlling a process having unknown parameters that must be estimated during operation?
- **Psychology:** How to model human performance on various learning tasks?
- **Artificial Intelligence:** How to write algorithms to acquire the knowledge humans are able to acquire, at least, as well as humans?
- **Evolutionary Models:** How to model certain aspects of biological evolution to improve the performance of computer programs?

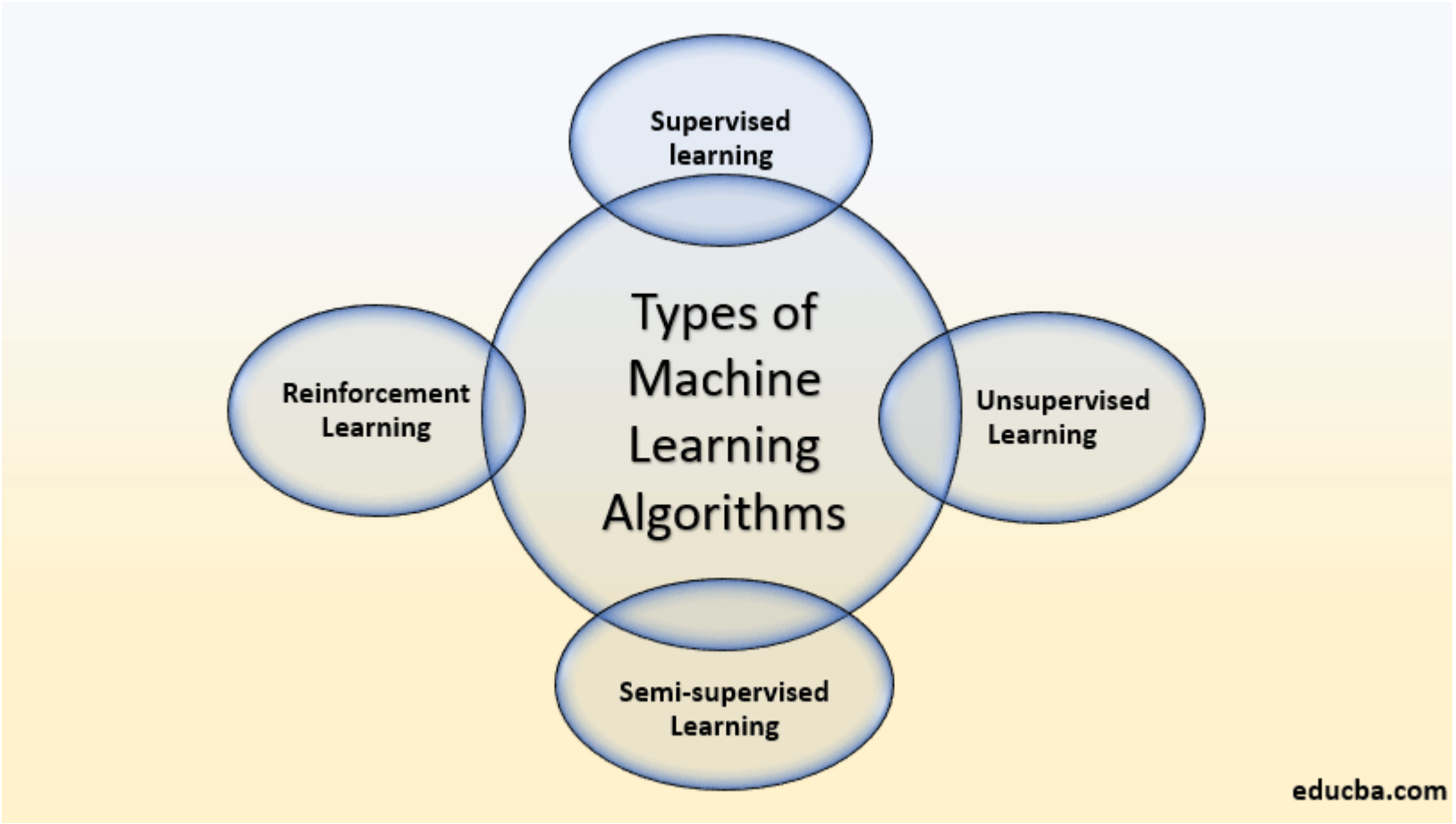
Machine Learning was coined in 1959 by Arthur Samuel, an American IBMer and pioneer in the field of computer gaming and artificial intelligence.

“Machine learning is the science of getting computers to act without being explicitly programmed.” — Stanford University
It’s a subset of AI which uses statistical methods to enable machines to improve with experience.

Machine learning involves computers discovering how they can perform tasks without being explicitly programmed to do so. It involves computers learning from data provided so that they carry out certain tasks.



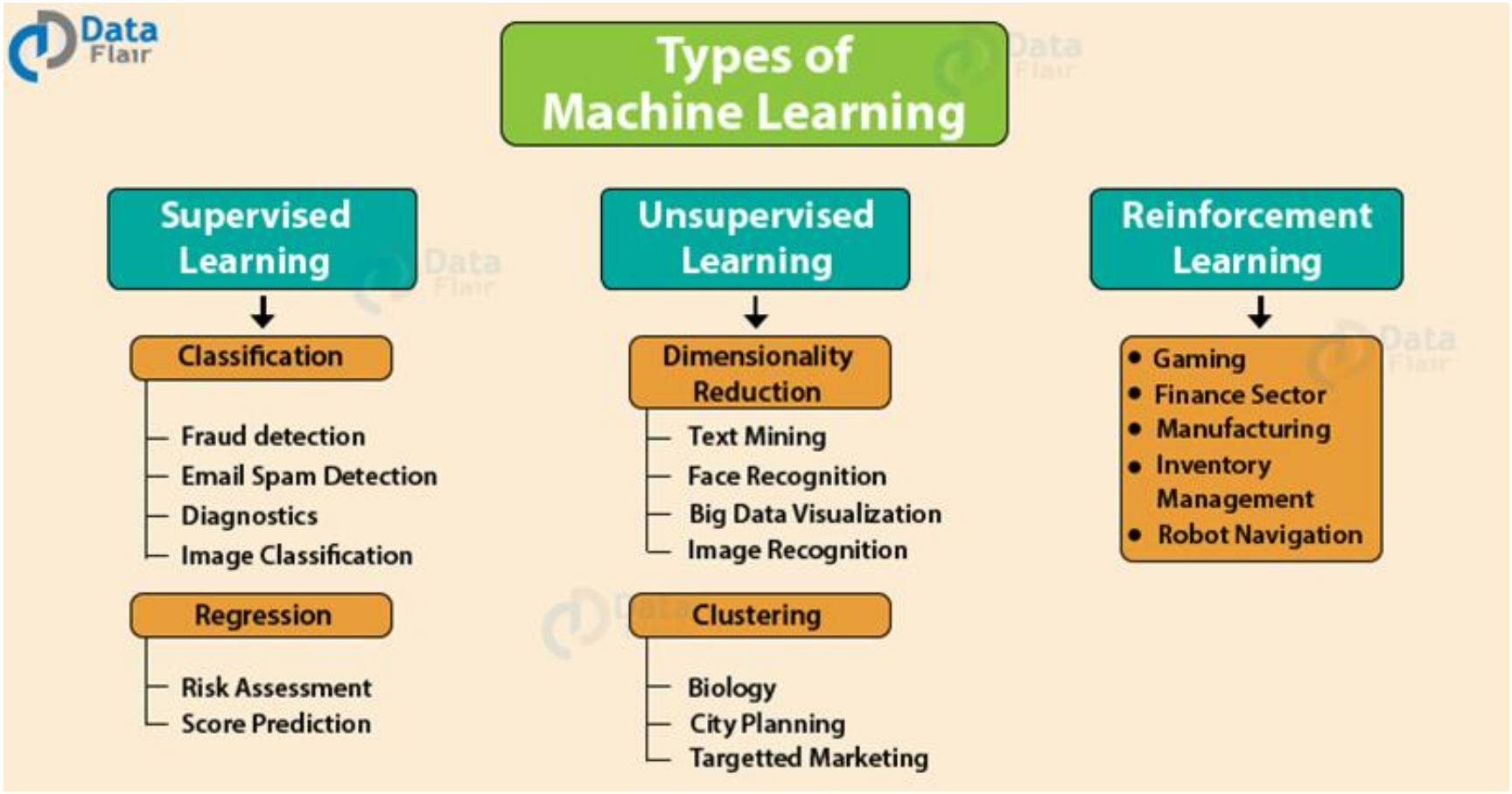

Define algo/logic to compute output:
Input
Algo (Logic)
Traditional Program
Output
Learn model/logic from data:
Training Input
Expected Output
ML Training
Model (Logic)
Input
ML Inference
Output

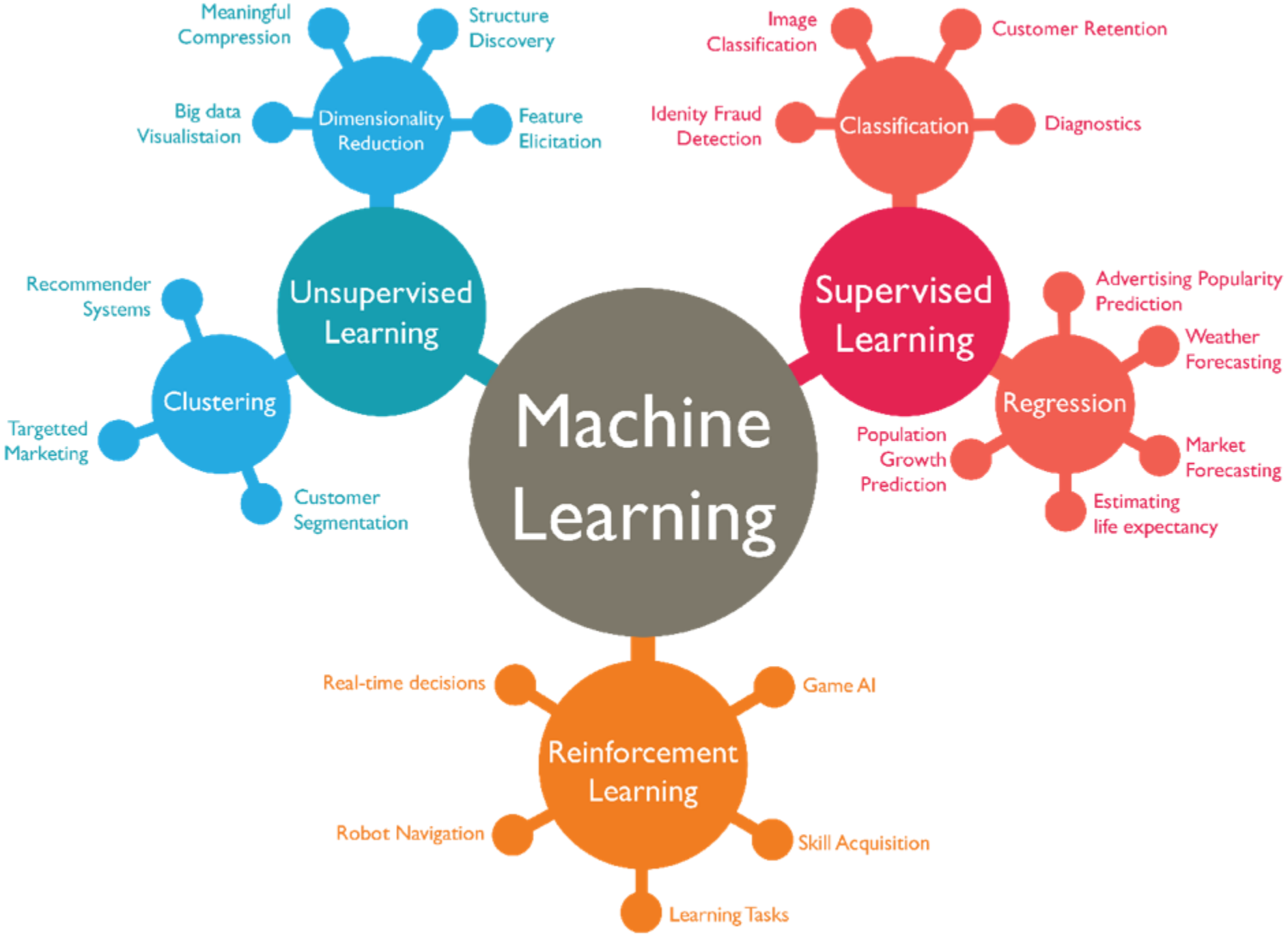

Supervised learning
Types of Machine Learning Algorithms
Reinforcement Learning
Unsupervised Learning
Semi-supervised Learning
educba.com

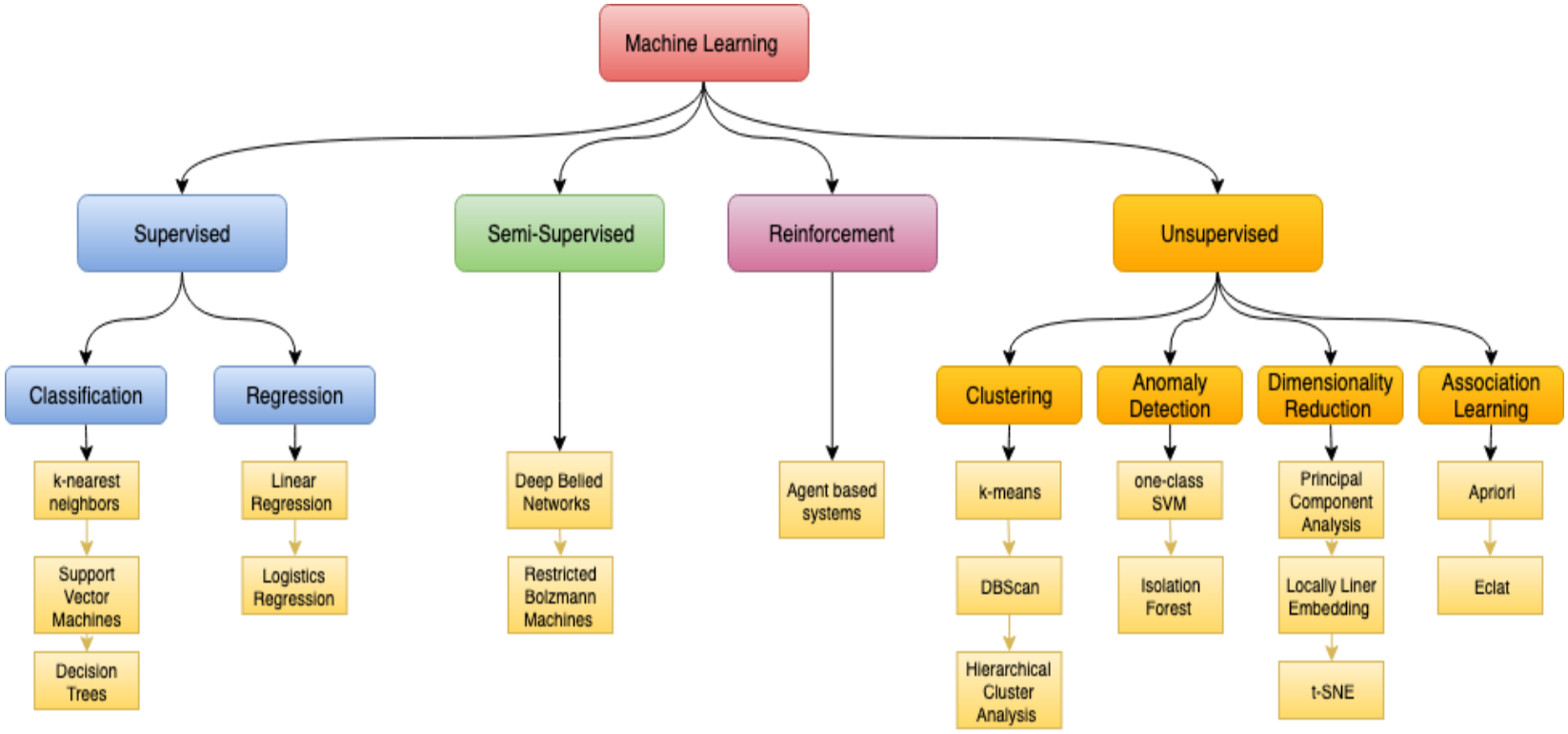
Styles of Learning

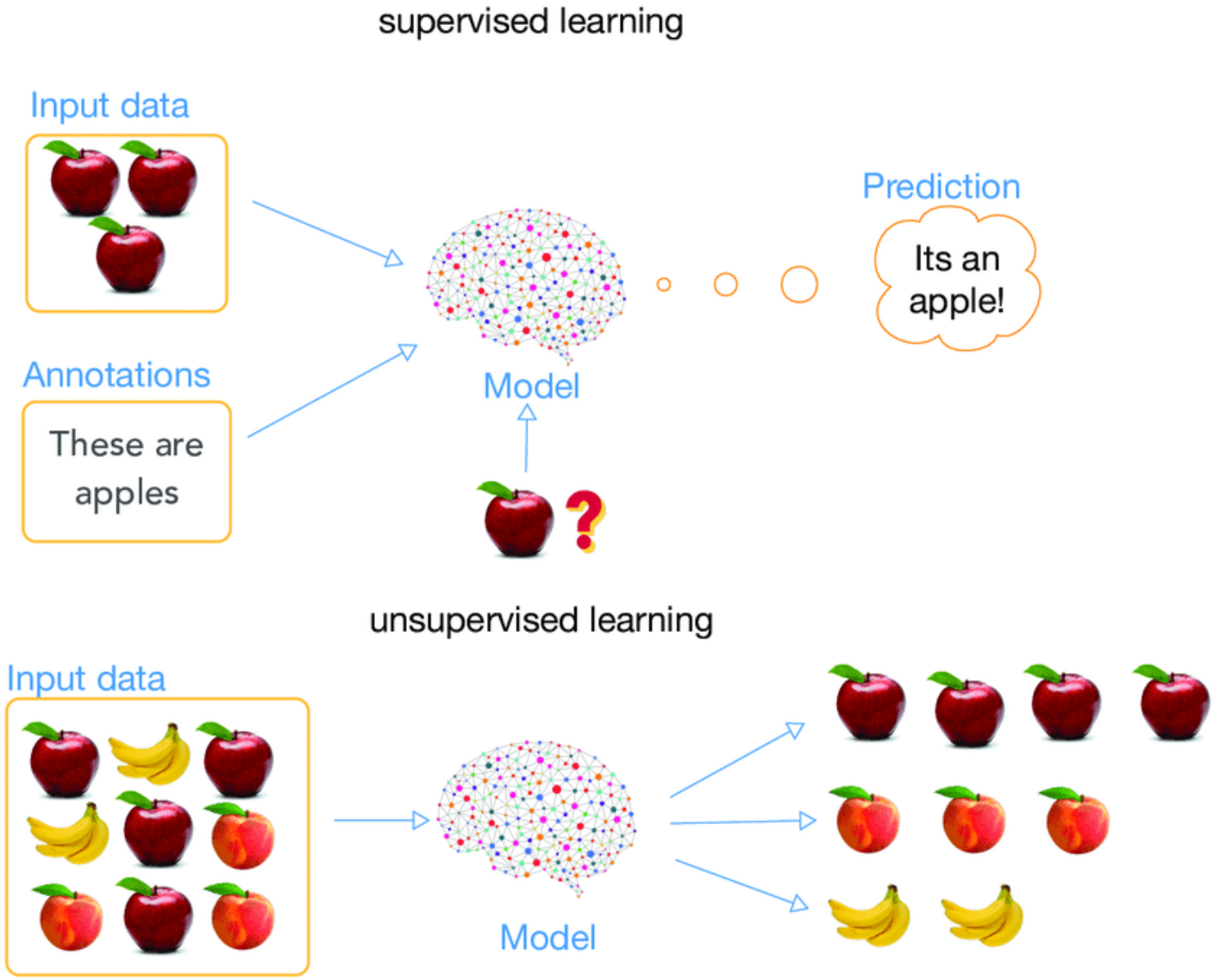
Supervised	Unsupervised	Semi-Supervised	Reinforcement
• Data has **known labels** or output	• Labels or output unknown • Focus on **finding patterns and gaining insight** from the data	• Labels or output known for **a subset of** data • A blend of supervised and unsupervised learning	• Focus on **making decisions** based on previous experience • Policy-making with feedback
• Insurance underwriting • Fraud detection	• Customer clustering • Association rule mining	• Medical predictions (where tests and expert diagnoses are expensive, and only part of the population receives them)	• Game AI • Complex decision problems • Reward systems

sas


Data Flair
Types of Machine Learning
Supervised Learning
Classification
Fraud detection
Email Spam Detection
Diagnostics
Image Classification
Regression
Risk Assessment
Score Prediction
Unsupervised Learning
Dimensionality Reduction
Text Mining
Face Recognition
Big Data Visualization
Image Recognition
Clustering
Biology
City Planning
Targetted Marketing
Reinforcement Learning
Gaming
Finance Sector
Manufacturing
Inventory Management
Robot Navigation


Machine Learning
Unsupervised Learning
Dimensionality Reduction
Meaningful Compression
Structure Discovery
Big data Visualistaion
Feature Elicitation
Clustering
Recommender Systems
Targetted Marketing
Customer Segmentation
Supervised Learning
Classification
Image Classification
Customer Retention
Idenity Fraud Detection
Diagnostics
Regression
Advertising Popularity Prediction
Weather Forecasting
Population Growth Prediction
Market Forecasting
Estimating life expectancy
Reinforcement Learning
Real-time decisions
Game AI
Robot Navigation
Skill Acquisition
Learning Tasks


Machine Learning
Supervised
Semi-Supervised
Reinforcement
Unsupervised
Classification
Regression
k-nearest neighbors
Support Vector Machines
Decision Trees
Linear Regression
Logistics Regression
Deep Belied Networks
Restricted Bolzmann Machines
Agent based systems
Clustering
Anomaly Detection
Dimensionality Reduction
Association Learning
k-means
DBScan
Hierarchical Cluster Analysis
one-class SVM
Isolation Forest
Principal Component Analysis
Locally Liner Embedding
t-SNE
Apriori
Eclat


supervised learning
Input data
Prediction
Its an apple!
Annotations
These are apples
Model
unsupervised learning
Input data
Model

Machine Learning : A Definition:

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

Learning is used when:

- Human expertise does not exist (navigating on Mars)
- Humans are unable to explain their expertise (speech recognition)
- Solution changes with time (routing on a computer network)
- Solution needs to be adapted to particular cases (user biometrics)

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, we must identity these three features:

1. the class of tasks

2. the measure of performance to be improved

3. the source of experience.

WELL-POSED LEARNING PROBLEMS

A checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

A handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified
- Training experience E: a database of handwritten words with given classifications

A robot driving learning problem:

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance travelled before an error (as judged by human overseer)
- Training experience E: a sequence of images and steering commands recorded while observing a human driver

Checkers game

About game

•Classic Checkers is played by two players. Each player begins the game with 12 colored discs. (Typically, one set of pieces is black and the other white.)

•

• The board consists of 64 squares, alternating between 32 dark and 32 light squares. It is positioned so that each player has a light square on the right side corner closest to him or her.

•A player wins the game when the opponent cannot make a move. In most cases, this is because all of the opponent's pieces have been captured, but it could also be because all of his pieces are blocked in.

Rules of the game

•Each player places his or her pieces on the 12 dark squares closest to him or her.

• White moves first. Players then alternate moves.

• Moves are allowed only on the dark squares, so pieces always move diagonally. Single pieces are always limited to forward moves (toward the opponent).

• A piece making a non-capturing move (not involving a jump) may move only one square.

• A piece making a capturing move (a jump) leaps over one of the opponent's pieces, landing in a straight diagonal line on the other side. Only one piece may be captured in a single jump; however, multiple jumps are allowed on a single turn.

Rules of the game cont.

• When a piece is captured, it is removed from the board.

• If a player is able to make a capture, there is no option - the jump must be made. If more than one capture is available, the player is free to choose whichever he or she prefers.

• When a piece reaches the furthest row from the player who controls that piece, it is crowned and becomes a king.

• Kings are limited to moving diagonally, but may move both forward and backward. (Remember that single pieces, i.e. non-kings, are always limited to forward moves.)

DESIGNING A LEARNING SYSTEM

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

To illustrate some of the basic design issues and approaches to machine learning, let us consider designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament.

1 **Choosing the Training Experience**

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

The type(key attribute) of training experience available can have a significant 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.
2. the degree to which the learner controls the sequence of training examples.
3. how well it represents the distribution of examples over which the final system performance P must be measured.

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

For example, 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.
- Alternatively, it might have available only *indirect information consisting of the move sequences and final outcomes* of various games played.
- In this later case, 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. Training experience is the degree to which the learner controls the sequence of training examples.

- the learner might rely on the teacher to select informative board states and to provide the correct move for each.
- Alternatively, the learner might itself propose board states 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.
- Notice in this last case the learner may choose between experimenting with novel board states that it has not yet considered, or honing its skill by playing minor variations of lines of play it currently finds most promising.

3. Training experience is how well it represents the distribution of examples over which the final system performance P must be measured.
learning is most reliable when the training examples follow a distribution similar to that of future test examples.

- In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.
- If its training experience E consists only of games played against itself, there is an obvious danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested. For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion.
- it is often necessary to learn from a distribution of examples that is somewhat different from those on which the final system will be evaluated (e.g., the world checkers champion might not be interested in teaching the program!). Such situations are problematic because mastery of one distribution of examples will not necessary lead to strong performance over some other distribution.

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 us begin with a checkers-playing program that can generate the *legal moves from any board* state.

•The program needs only to learn how to choose the *best move from among* these legal moves. This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known.

Given this setting where 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.

Let us call this function *ChooseMove and use the notation*

ChooseMove : B → M

to 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.*

Throughout our discussion of machine learning we will find it useful to reduce the problem of improving performance P at task T to the problem of learning some particular *targetfunction such as ChooseMove*

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

An alternative target function and one that will turn out to be easier to learn in this setting-is an evaluation function that assigns a numerical score to any given board state.

Let the target function ***V and the notation V : B → R to denote that V maps*** any legal board state from the set B to some real value (we use *8 to denote the set* of real numbers). We 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 therefore define the target value ***V(b) for an arbitrary board state b in B, as follows:***

1. if *b is a final board state that is won, then V(b) = 100*

2. if b is a final board state that is lost, then V(b) = -100

3. if b is a final board state that is drawn, then V(b) = 0

4. if b is a not a final state in the game, then V(b) = V(b^{l}), where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game (assuming the opponent plays optimally, as well).

3. Choosing a Representation for the Target Function

let us choose a simple representation:

for any given board state, the function c will be calculated as a linear combination of the following board features:

xl: 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

learning program will represent c(b) as a linear function of the

Form

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

where w_0 *through W6 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.

Partial design of a checkers learning program:

Task T: playing checkers

Performance measure *P: percent of games won in the world tournament*

Training experience E: games played against itself

Target function: V: Board $\rightarrow \Re$

Target function representation

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

The first three items above correspond to the specification of the learning task, whereas the final two items constitute design choices for the implementation of the learning program

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(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 $V_{train}(b)$s therefore **+100.**

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

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

$V_{train}(b) \leftarrow \hat{V}$ (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 best fit 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.

we require an algorithm that will incrementally refine the weights as new training examples become available and that will be robust to errors in these estimated training values.

One such algorithm is called 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

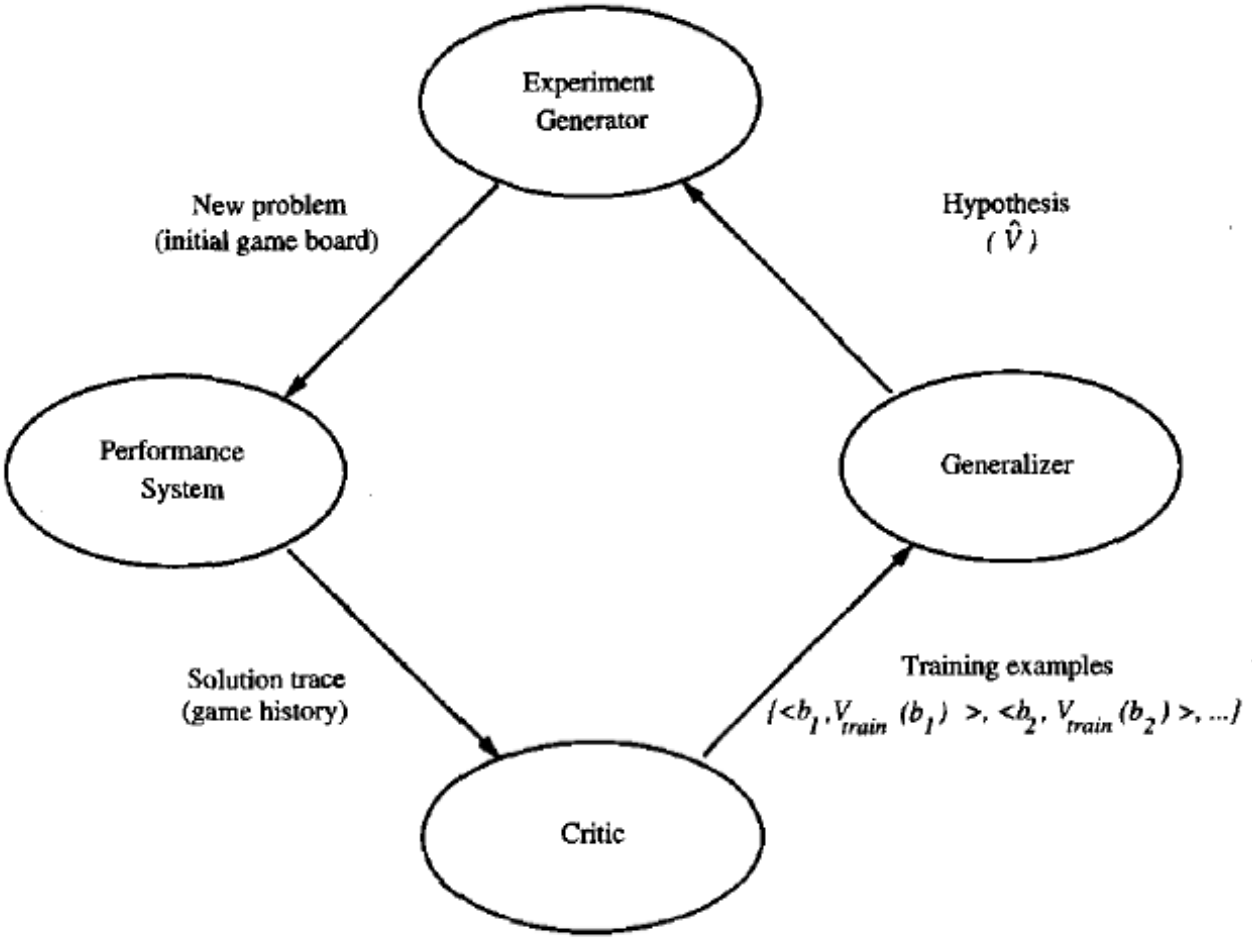
• When the error ($V_{train}(b)$- $\hat{V}(b)$) is zero, no weights are changed.

•When ($V_{train}(b)$ - $\hat{V}(b)$) is positive (i.e., when $\hat{V}(b)$ is too low), then each 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



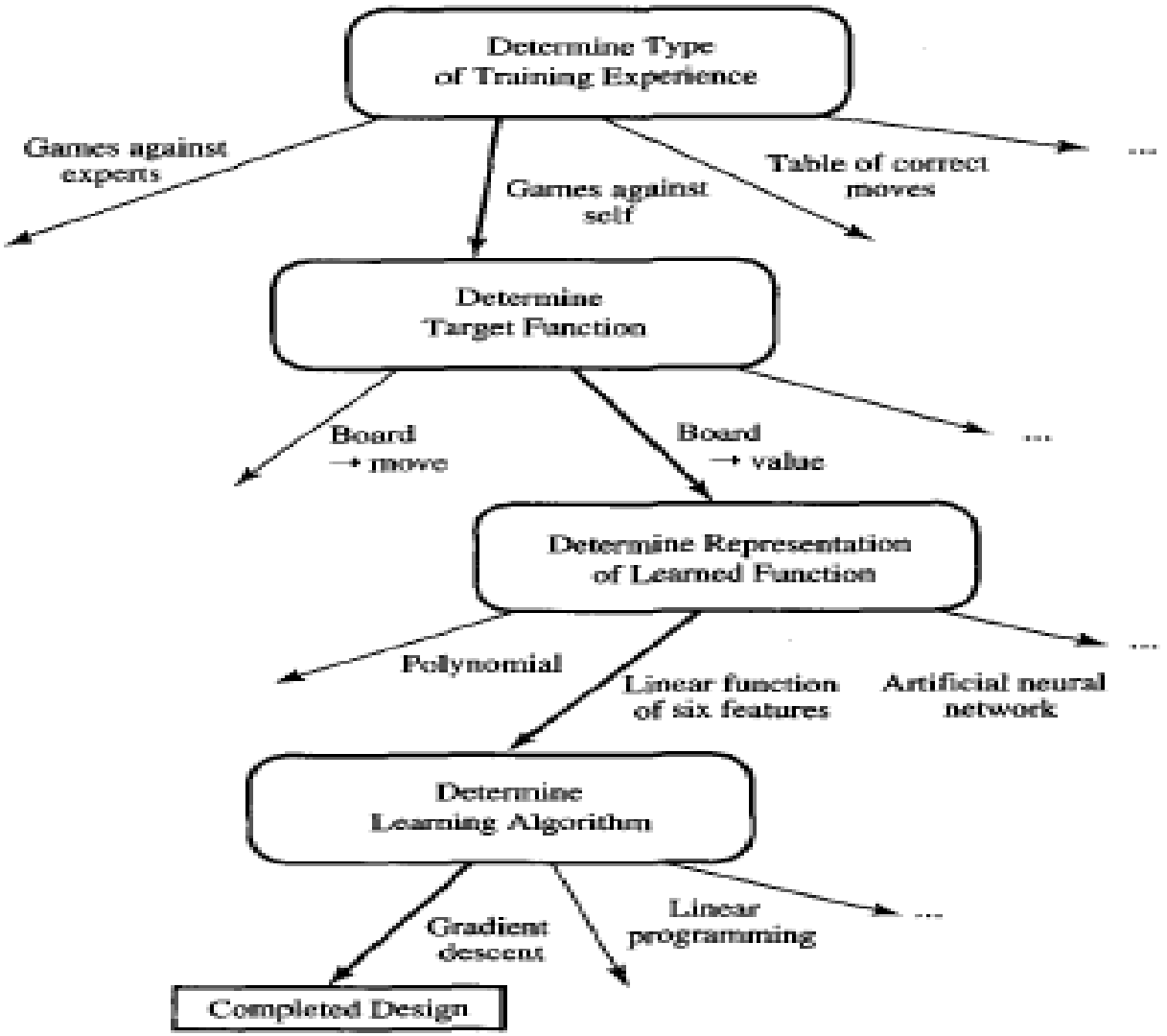
1. **The Performance System** is the module that must solve the given performance task by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

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 Generat**or 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?

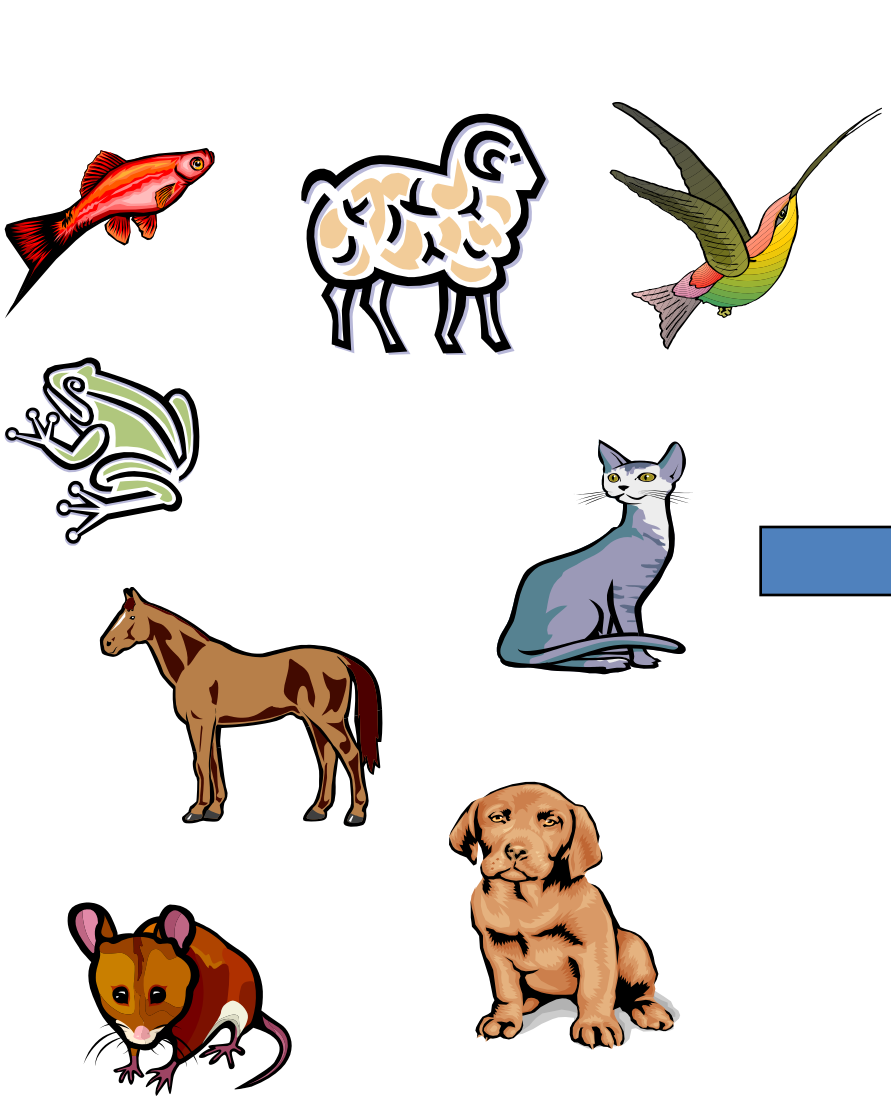
Concept Learning

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

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

What is a Concept ?

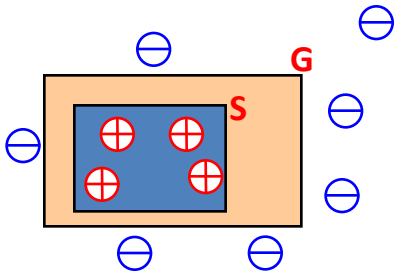


A concept describes a subset of objects or events defined over a larger set

Concept Learning

learning based on symbolic representations

- Acquire/Infer the definition of a general concept or category given a (labeled) sample of positive and negative training examples of the category
  - Each concept can be thought of as a Boolean-valued (true/false or yes/no) function
    - Approximate a Boolean-valued function from examples
  - Concept learning can be formulated as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples
  - Take advantage of a naturally occurring structure over the hypothesis space
    - General-to-specific ordering of hypotheses



Training Examples for *EnjoySport*

- Concept to be learned
  - "Days on which Aldo enjoys his favorite water sport"

Attributes

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Days

Concept to be learned

- Days (examples/instances) are represented by a set of attributes

- What is the general concept ?
  - The task is to learn to predict the value of *EnjoySport* for an arbitrary day based on the values of other attributes
  - Learn a (a set of) hypothesis representation(s) for the concept

Representing Hypotheses

- Many possible representations for hypotheses *h*
- Here *h* is conjunction of constraints on attributes
- Each constraint can be
  - A specific value (e.g., "*Water=Warm*")
  - Don't care (e.g., "*Water=?*")
  - No value acceptable (e.g., "*Water=Ø* ")

A hypothesis is a vector of constraints

- For example

Sky	AirTemp	Humid	Wind	Water	Forecast
< *Sunny*	?	?	*Strong*	?	*Same* >

  - Most general hypothesis
    < ? ? ? ? ? ? >
  - Most specific hypothesis
    < Ø Ø Ø Ø Ø Ø >

All are positive examples

All are negative examples

Definition of Concept Learning Task

- Given
  - Instances X: possible days, each described by six attributes
    Sky, *AirTemp*, *Humidity*, *Wind*, *Water*, *Forecast*
    (*Sunny, Cloudy, Rainy*) (*Warm, Cold*) (*Normal, High*) (*Strong, Week*) (*Warm, Cool*) (*Same, Change*)
  - Target concept/function c : *EnjoySport* X → {0, 1}
  - Hypotheses H : Conjunctions of Literals. E.g., "No" "Yes"
    <?,*Cold*, *High*, ?, ?, ? >
  - Training examples D : Positive and negative examples (members/nonmembers) of the target function (concept)
- Determine $\langle x_1, c(x_1)\rangle, \langle x_2, c(x_2)\rangle, \ldots, \langle x_m, c(x_m)\rangle$
  - A hypothesis h in H (an approximate target function) such that $h(x)=c(x)$ for all x in D
    target concept value

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

  – Assumption of Inductive Learning
    - The best hypothesis regarding the unseen instances is the hypothesis that best fits the observed training data

Viewing Learning As a Search Problem

- Concept learning can be viewed as the task of searching through a large space of hypotheses

Instance space *X*

Sky (*Sunny/Cloudy/Rainy*)
AirTemp (*Warm/Cold*)
Humidity (*Normal/High*)
Wind (*Strong/Weak*)
Water (*Warm/Cool*)
Forecast (*Same/Change*)

=> 3*2*2*2*2*2=96 instances

Hypothesis space *H*

Ø

5*4*4*4*4*4=5120 syntactically distinct hypotheses

1+4*3*3*3*3*3=973 semantically distinct hypotheses

Each hypothesis is represented as a conjunction of constraints

E.g.,
< Ø Warm Normal Strong Cool Same >
< Sunny Ø Normal Strong Cool Change>

Viewing Learning As a Search Problem

- Study of learning algorithms that examine different strategies for searching the hypothesis space, e.g.,
  - *Find-S* Algorithm
  - *List-Then-Eliminate* Algorithm
  - *Candidate Elimination* Algorithm

- How to exploit the naturally occurring structure in the hypothesis apace ?
  - Relations among hypotheses , e.g.,
    - General-to-Specific-Ordering

General-to-Specific-Ordering of Hypothesis

- Many concept learning algorithms organize the search through the hypothesis space by taking advantage of a naturally occurring structure over it
  - *"general-to-specific ordering"*

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

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

Suppose that h_1 and h_2 classify positive examples

- *h_2 is more general than h_1*
  - h_2 imposes fewer constraints on instances
  - h_2 classify more positive instances than h_1 does

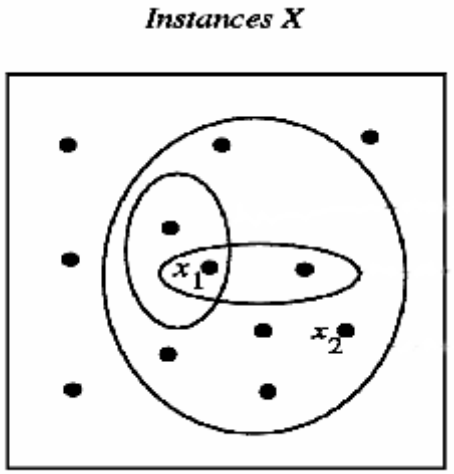
  - A useful structure over the hypothesis space

More-General-Than Partial Ordering

- Definition
  - Let h_j and h_k be Boolean-valued functions defined over X. Then h_j is *more general than* h_k ($h_j >_g h_k$) if and only if

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

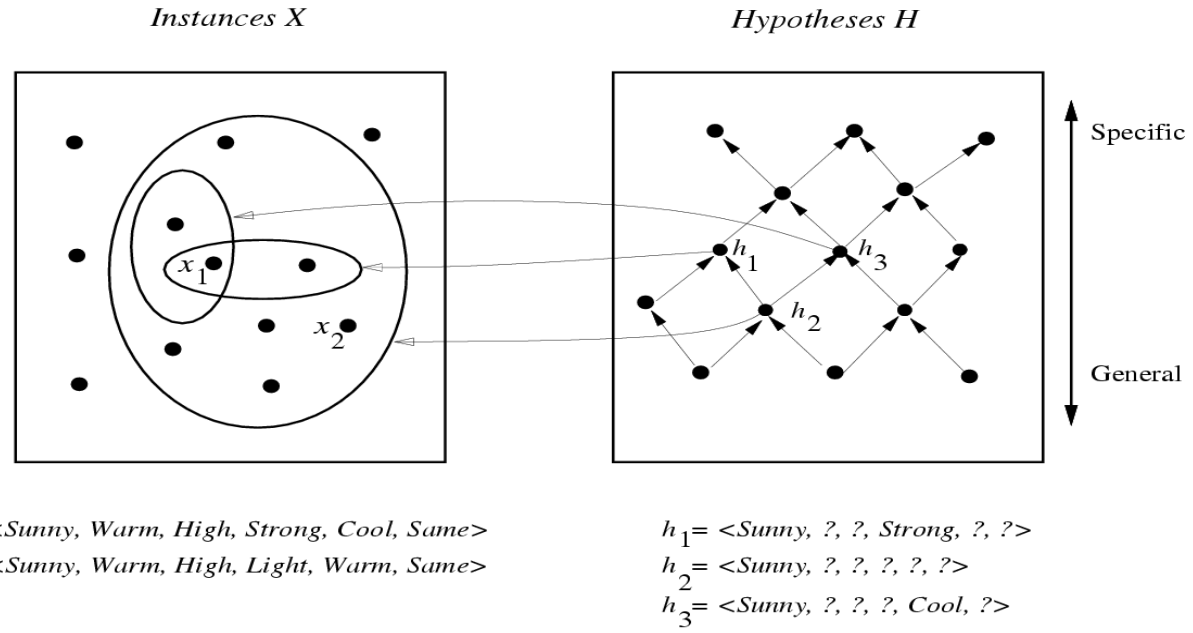
x satisfies h_k



- We also can define the *more-specific-than* ordering

General-to-Specific Ordering of Hypotheses

- An illustrative example



- Suppose instances are classified positive by h_1, h_2, h_3
  - h_2 (imposing fewer constraints) are *more general than* h_1 and h_3
  - $h_1 \overset{?}{\longleftrightarrow} h_3$

partial order relation
- antisymmetric, transitive

$$h_a \geq_g h_b,\ h_b \geq_g h_c \Rightarrow h_a \geq_g h_c$$

Find-S Algorithm

- Find a maximally specific hypothesis by using the *more-general-than* partial ordering to organize the search for a hypothesis consistent with the observed training examples

1. Initialize h to the most specific hypothesis in H $h \leftarrow \langle \phi, \phi, \phi, \phi, \phi, \phi \rangle$
2. For each positive training instance x
   - For each attribute constraint a_i in h

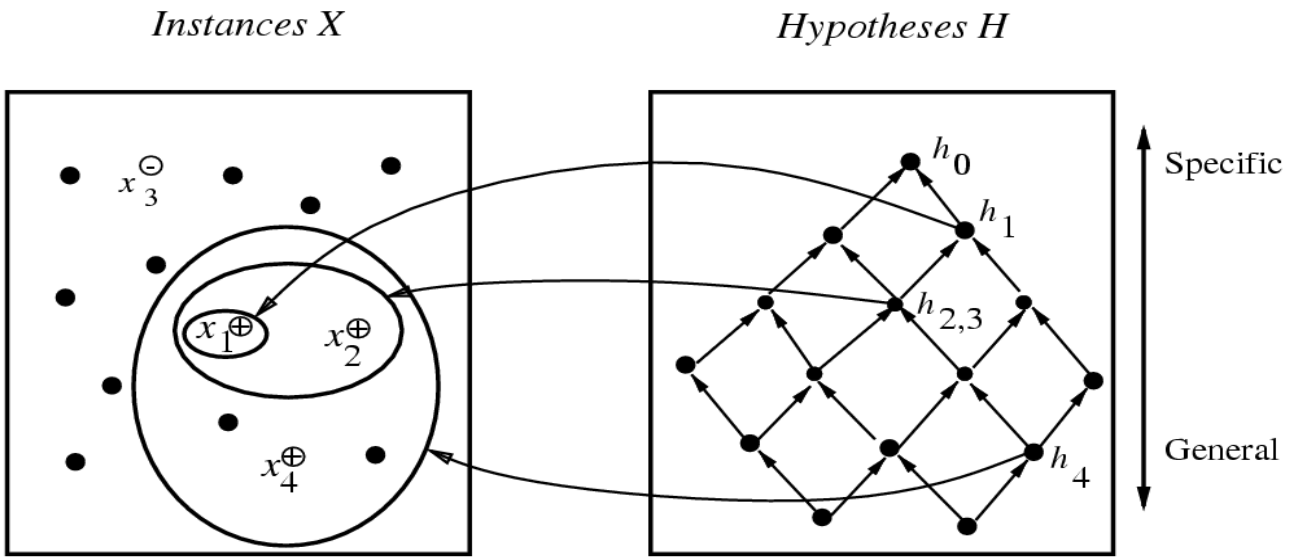
     If the constraint a_i in h is satisfied by x

     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

- Hypothesis Space Search by ***Find-S***



x_1 = <*Sunny Warm Normal Strong Warm Same*>, +
x_2 = <*Sunny Warm High Strong Warm Same*>, +
x_3 = <*Rainy Cold High Strong Warm Change*>, -
x_4 = <*Sunny Warm High Strong Cool Change*>, +

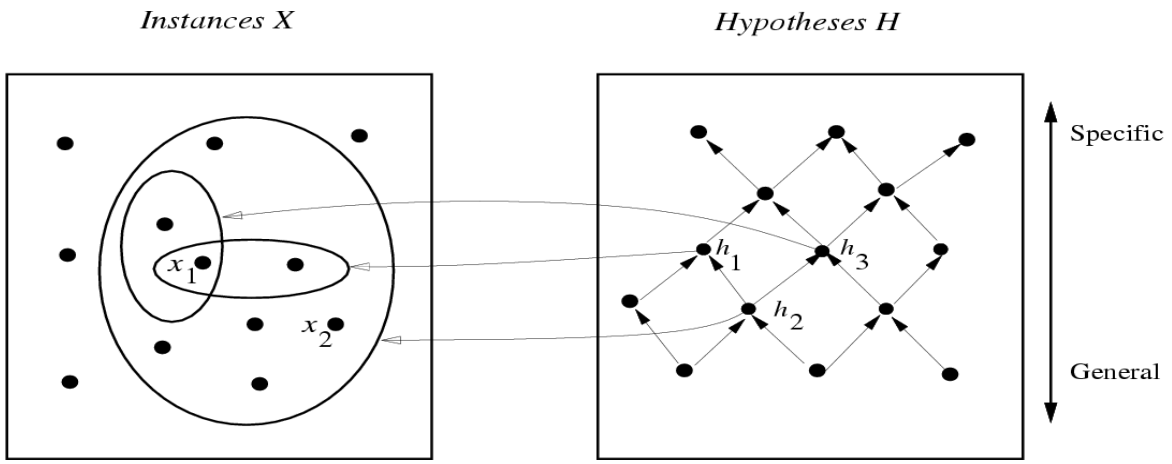
h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <*Sunny Warm Normal Strong Warm Same*>
h_2 = <*Sunny Warm ? Strong Warm Same*>
h_3 = <*Sunny Warm ? Strong Warm Same*> no change!
h_4 = <*Sunny Warm ? Strong ? ?* >

Substitute a "?" in place of any attribute value in *h* that is not satisfied by the new example

Find-S Algorithm

- Why *F-S* never check a negative example ?
  - The hypothesis *h* found by it is the most specific one in *H*
  - Assume the target concept *c* is also in *H* which will cover both the training and unseen positive examples
    - *c* is more general than *h*
  - Because the target concept will not cover the negative examples, thus neither will the hypothesis *h*

can be represented as a conjunction of attributes



Complaints about *Find-S*

- Can not tell whether it has learned concept
  (Output only one. Many other consistent hypotheses may exist!)
- Picks a maximally specific *h* (why?)
  (Find a most specific hypothesis consistent with the training data)
- Can not tell when training data inconsistent
  - What if there are noises or errors contained in training examples
- Depending on *H,* there might be several !

Consistence of Hypotheses

- 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 <*x*, *c*(*x*)> in *D*

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

- But *satisfaction* has another meaning
  - An example *x* is said to satisfy a hypothesis *h* when *h*(*x*)=1, regardless of whether *x* is positive or negative example of the target concept

Version Space

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

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

  - A subspace of hypotheses
  - Contain all plausible versions of the target concepts

List-Then-Eliminate Algorithm

1. *VersionSpace* ← a list containing all hypotheses in *H*
2. For each training example, <*x*, *c*(*x*)>

   remove from *VersionSpace* any hypothesis *h* for which *h*(*x*)≠*c*(*x*)
   - i.e., eliminate hypotheses inconsistent with any training examples
   - The *VersionSpace* shrinks as more examples are observed
3. Output the list of hypotheses in *VersionSpace*

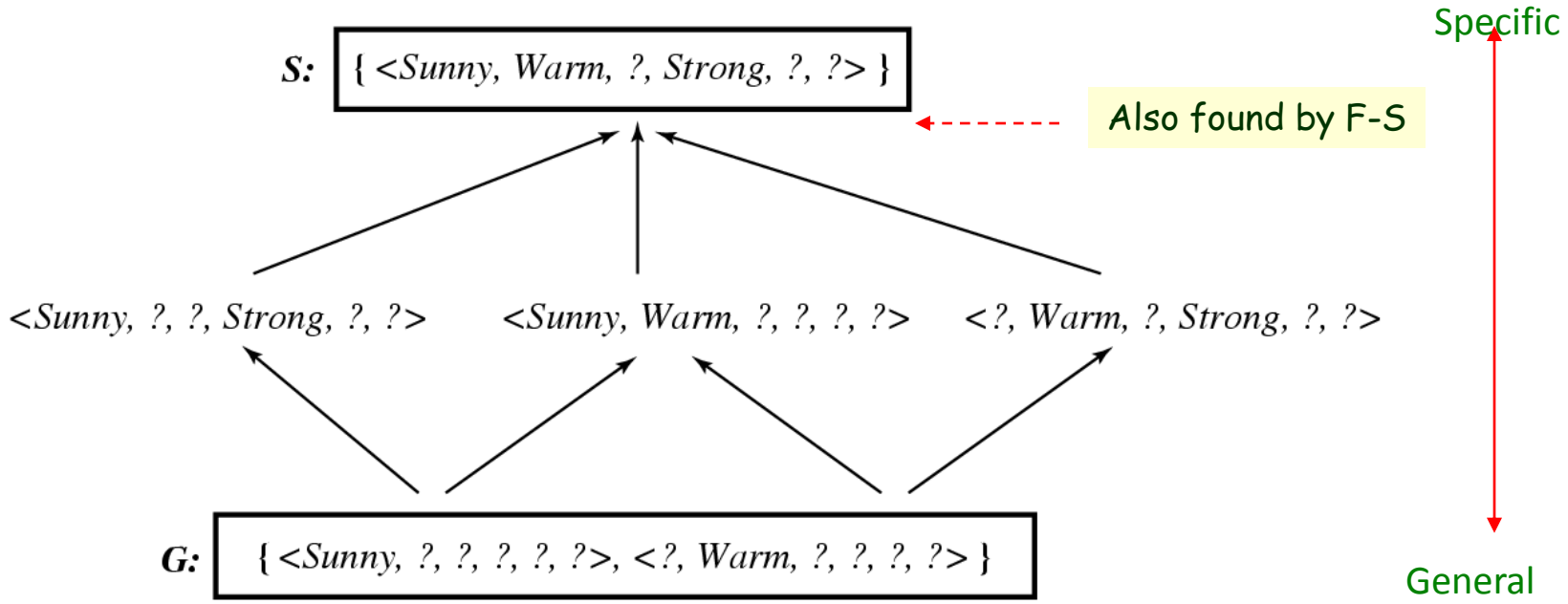
Drawbacks of *List-Then-Eliminate*

- The algorithm requires exhaustively enumerating all hypotheses in *H*
  - An unrealistic approach ! (full search)

- If insufficient (training) data is available, the algorithm will output a huge set of hypotheses consistent with the observed data

Example Version Space

- Employ a much more compact representation of the version space in terms of its most general and least general (most specific) members



Arrows mean more-general-than relations

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Representing Version Space

- The **General boundary** *G,* of version space $VS_{H,D}$ is the set of its maximally general members

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

- The **Specific boundary** *S,* of version space $VS_{H,D}$ is the set of its maximally specific members

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

- Every member of the version space lies between these boundaries
  - Version Space Representation Theorem

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

Candidate Elimination Algorithm

- 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*
        (i.e., partial-ordering relations exist)

positive training examples force the S boundary become increasing general

Candidate Elimination Algorithm

- 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*

negative training examples force the G boundary become increasing specific

Candidate Elimination Algorithm

- $G \leftarrow$ maximally general hypotheses in H

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

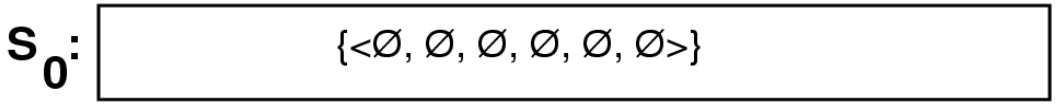
Should be specialized

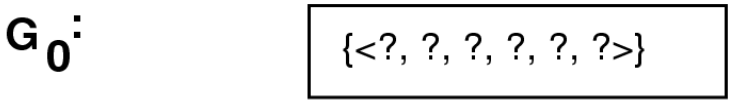
- $S \leftarrow$ maximally specific hypotheses in H

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

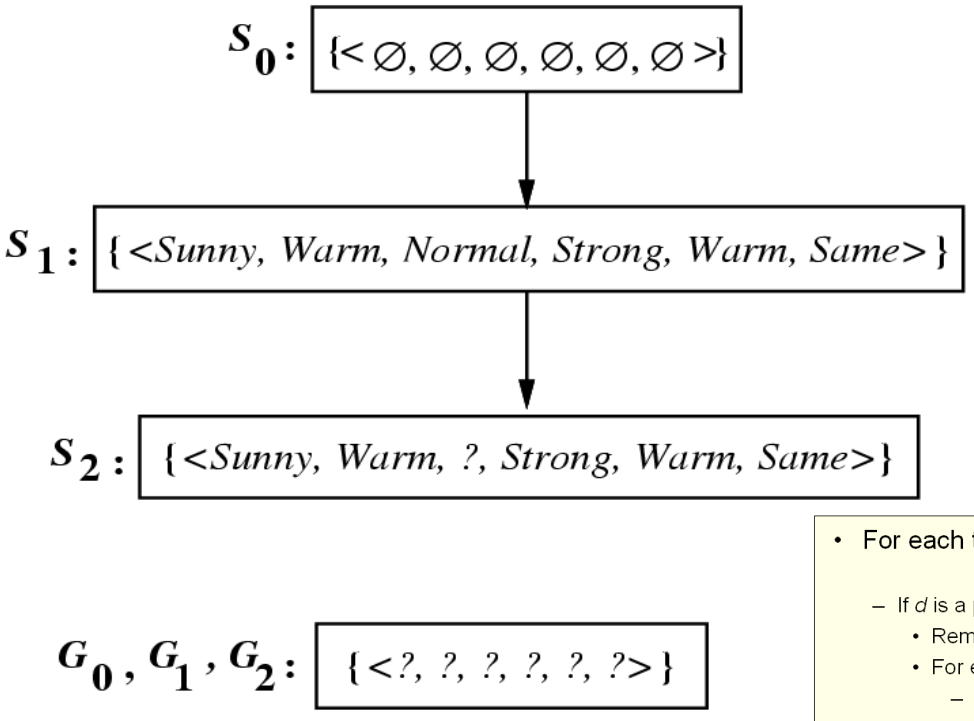
Should be generalized

Example Trace





Example Trace



- 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* (i.e., partial-ordering relations exist)

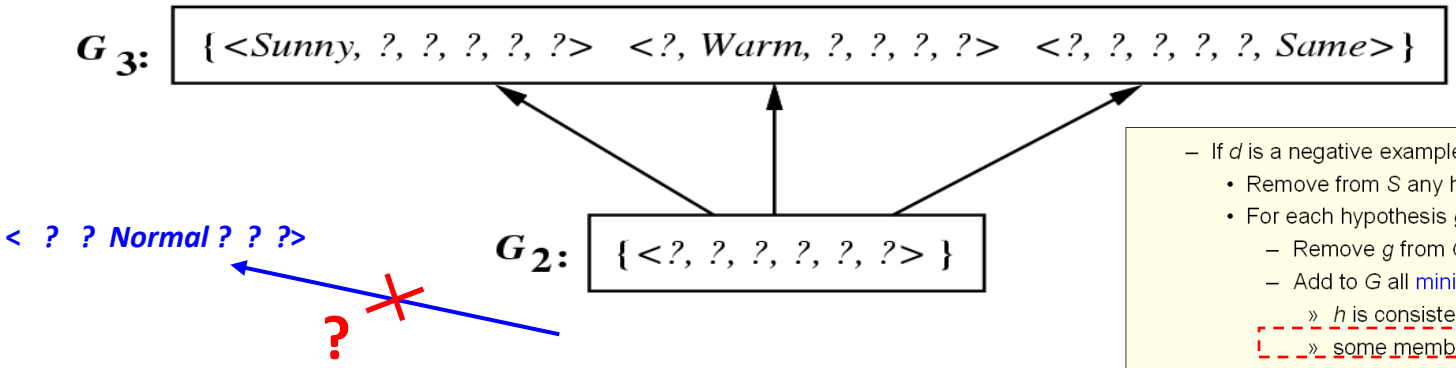
Training examples:

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

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

Example Trace

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



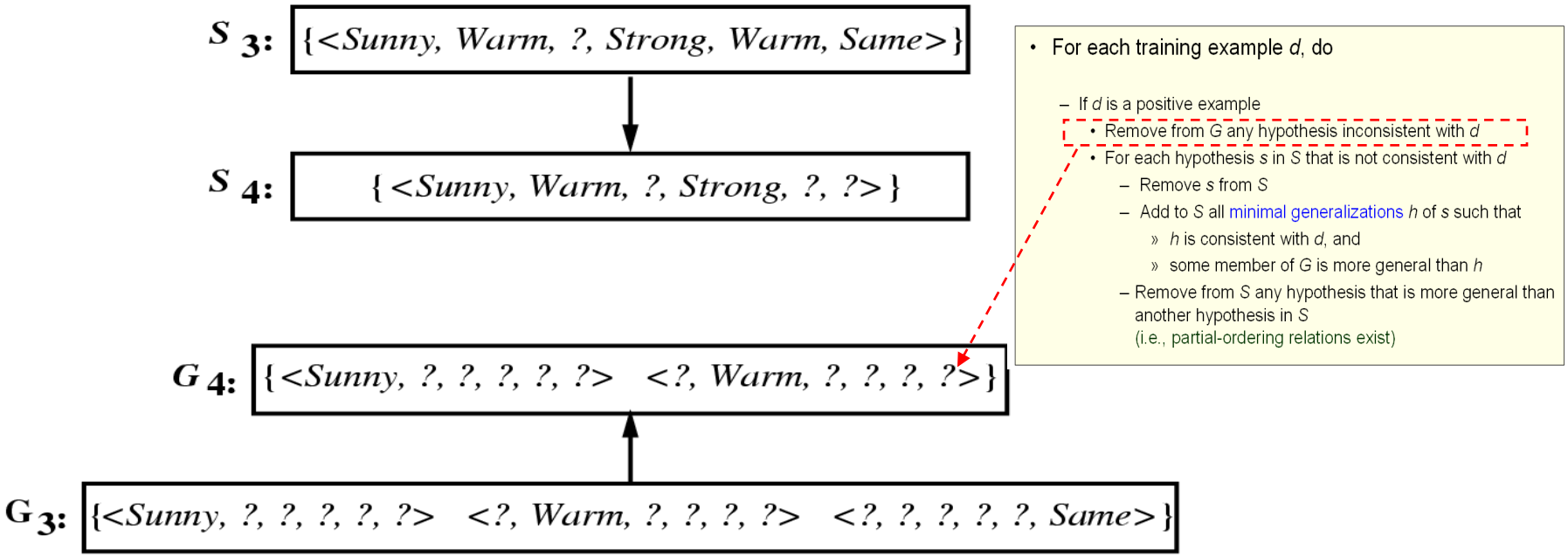
- If *d* is a negative example
  - Remove from *S* any hypothesis inconsistent with *d*
  - For each hypothesis *g* in *G* that is not consistent with *d*
    - Remove *g* from *G*
    - Add to *G* all minimal specializations *h* of *g* such that
      - » *h* is consistent with *d*, and
      - » some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

Training Example:

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

– Why <?,? ,Normal,?,?,? > etc. do not exist in G_3 ?

Example Trace

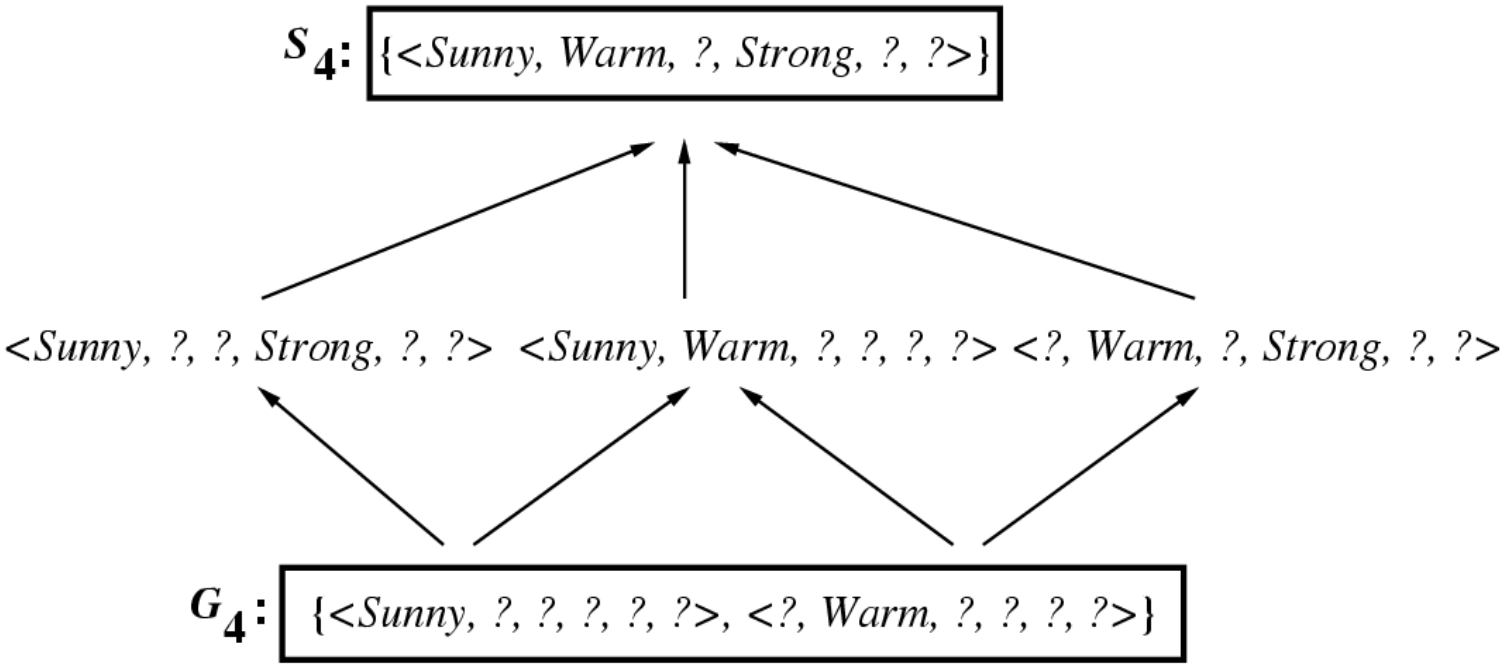


Training Example:

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

- Notice that,
  - *S* is a summary of the previously positive examples
  - *G* is a summary of the previously negative examples

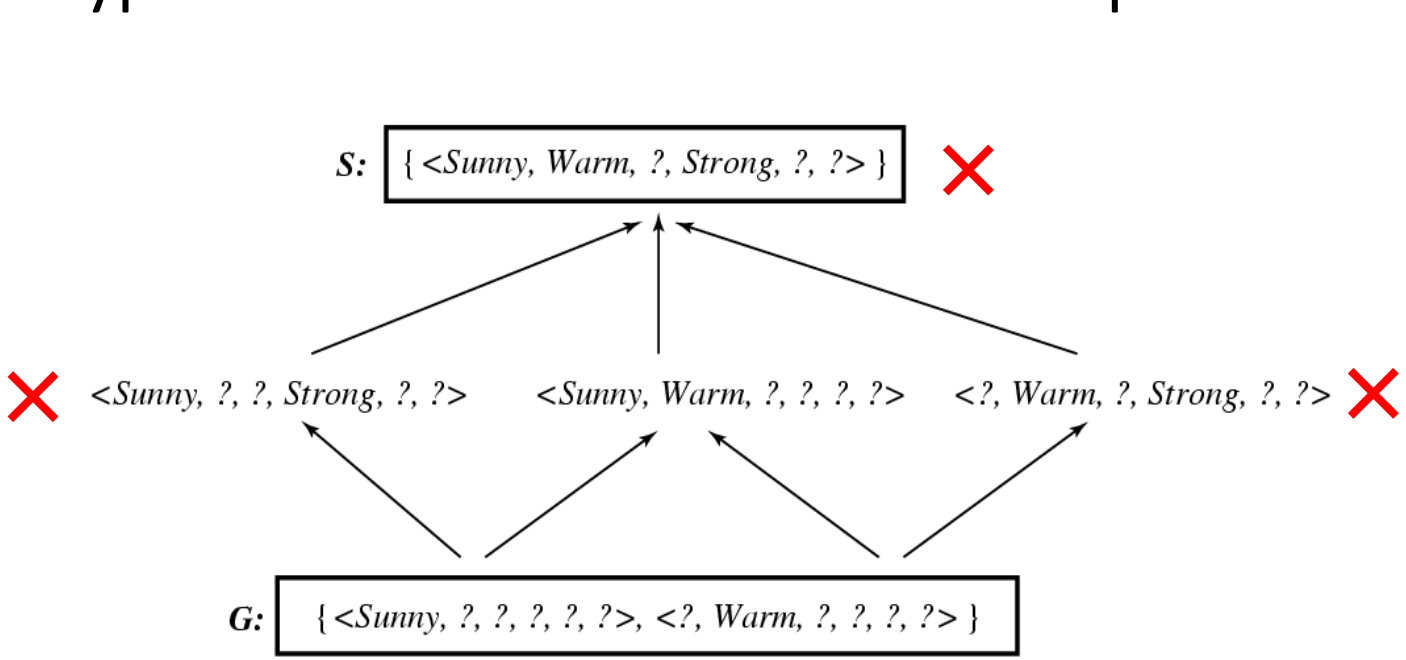
Example Trace



- *S* and *G* boundaries move monotonically closer to each other, delimiting a smaller and smaller version space

What Next Training Example

- Learner can generate useful queries
  - Discriminate among the alternatives competing hypotheses in the current version space



Specific

General

If a positive hypothesis is posed:
<Sunny, Warm, Normal, Light, Warm, Same >
What if it is a negative one ?

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-ELIMINTION 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= <? Warm Normal Strong Cool Change>

•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

(Sunny, ?, ?, ?, ?, ?) v (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 (xi , Dc) denote the classification assigned to the instance xi by L after training on the data Dc.
- The inductive bias of L is any minimal set of assertions B such that for any target concept c and corresponding training examples Dc

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

The below figure explains

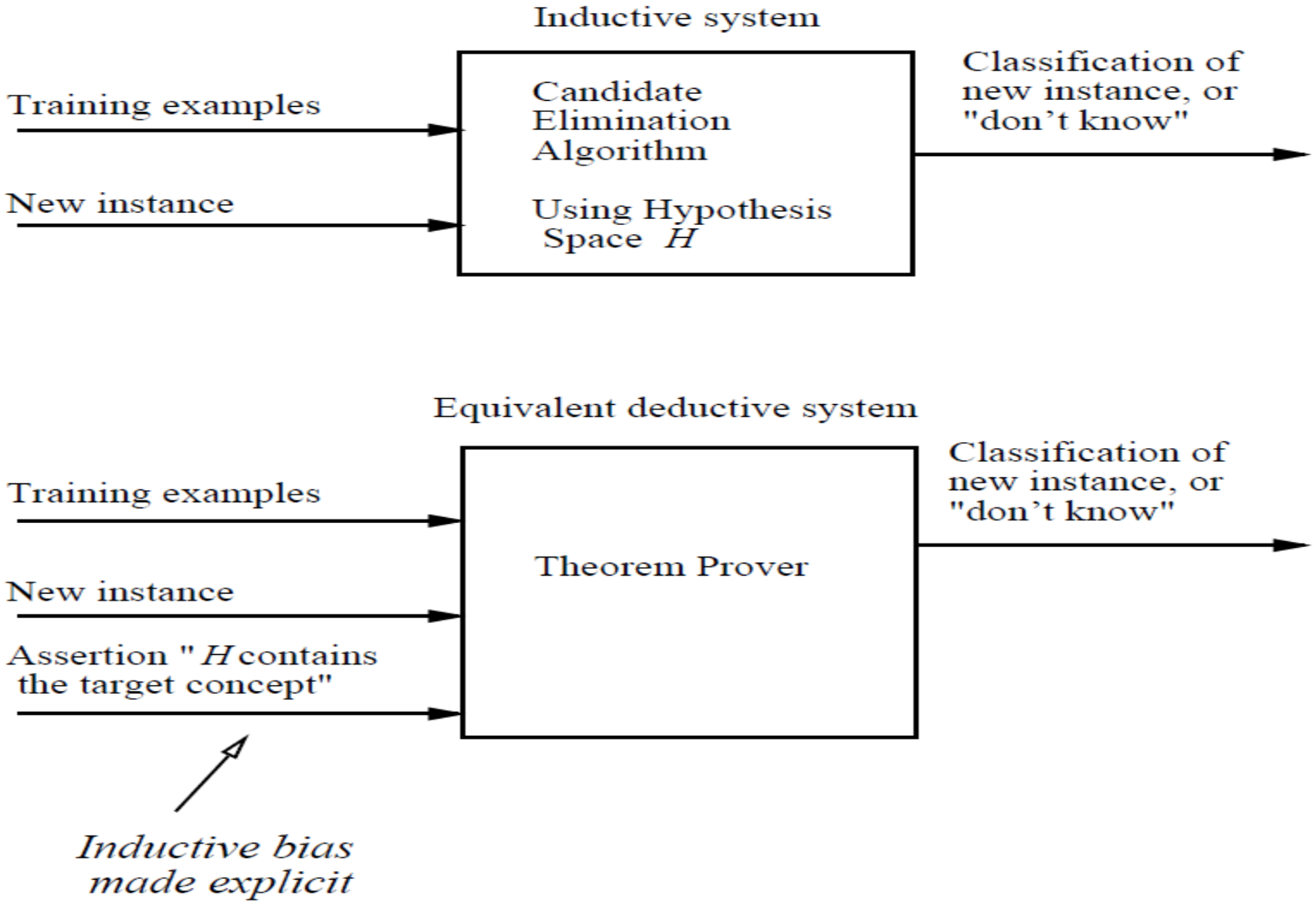
•Modelling inductive systems by equivalent deductive systems.

•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.

DEDUCTIVE AND INDUCTIVE METHOD COMPARED

Deductive	Inductive
General laws are first stated and particular examples to prove them.	First particular cases are dealt with and then laws are derived from them.
It does not lead to new knowledge.	It leads to new knowledge.
It is a method of verification and explanation.	It is method of discovery.


Inductive system
Training examples
New instance
Candidate Elimination Algorithm
Using Hypothesis Space H
Classification of new instance, or "don't know"
Equivalent deductive system
Training examples
New instance
Assertion "H contains the target concept"
Theorem Prover
Classification of new instance, or "don't know"
Inductive bias made explicit

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.

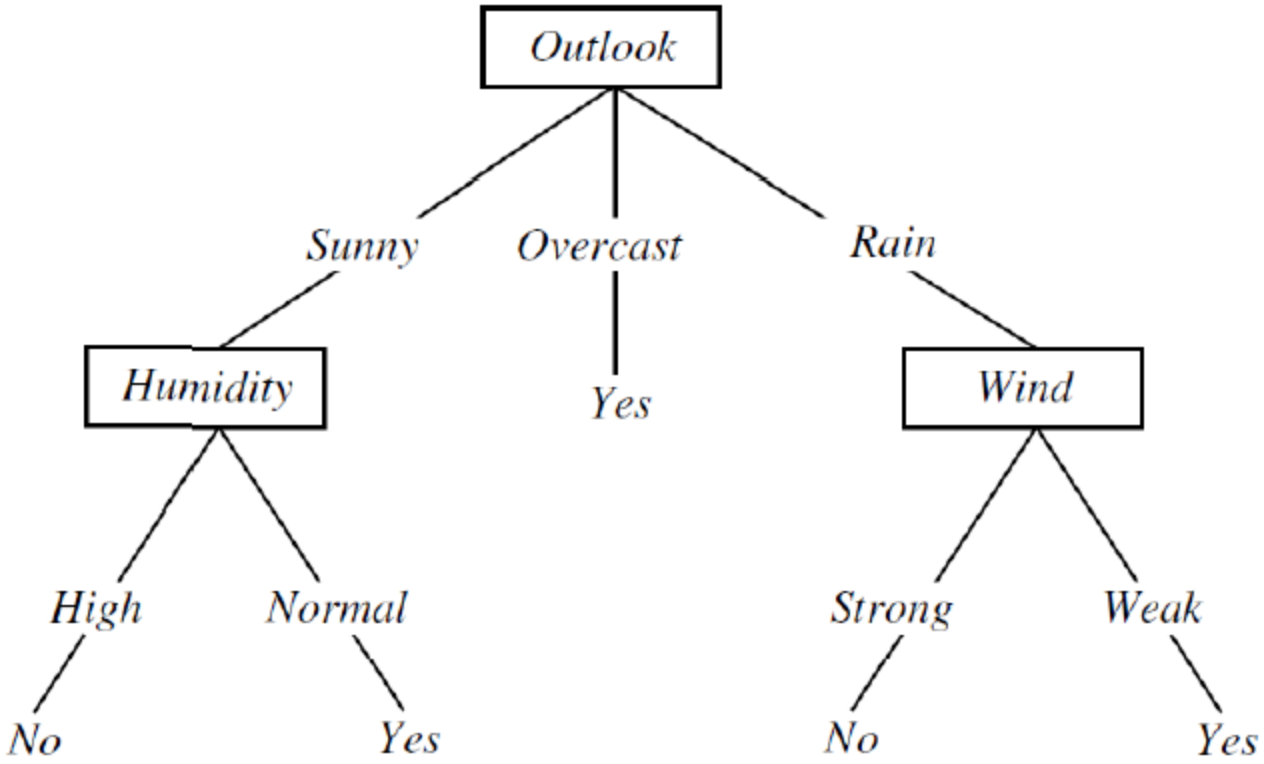


FIGURE: A decision tree for the concept ***PlayTennis***. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf

•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 ∧ Humidity = Normal)

∨ (Outlook = Overcast)

∨ (Outlook = Rain ∧ 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 a fixed 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 can be 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 ← the attribute from Attributes that best* classifies Examples
  - The decision attribute for Root ← 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* $_{vi}$, *be the subset of Examples that have value* v_i *for A*
    - If *Examples* $_{vi}$ *, 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* $_{vi}$, *Targe_tattribute, Attributes – {A})*
  - End
  - Return Root

* The best attribute is the one with highest information gain

TABLE: 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

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

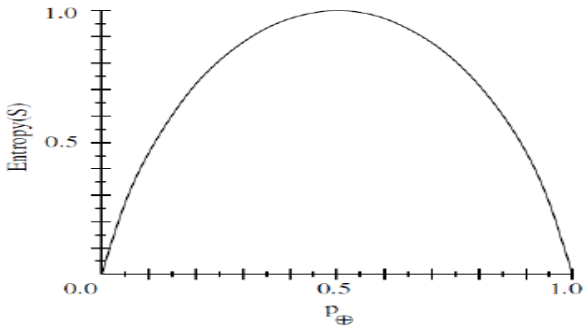


FIGURE The entropy function relative to a boolean classification, as the proportion, $p_\oplus$, of positive examples varies between 0 and 1.

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 examples S, is defined as

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

Example: Information gain

Let, *Values(Wind) = {Weak, Strong}*

S = [9+ , 5−]

S_{weak} = [6+ , 2−]

S_{strong} = [3+ , 3−]

Information gain of attribute *Wind:*

Gain(S, Wind) = Entropy(S) − 8/14 Entropy (S_{Weak}) − 6/14 Entropy (S_{Strong})

= 0.94 – (8/14)* 0.811 – (6/14) *1.00

= 0.048

- Calculate **Entropy** (Amount of uncertainity in dataset):

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

- Calculate **Average Information**:

$$I(Attribute) = \sum \frac{p_i + n_i}{p+n} Entropy(A)$$

- Calculate **Information Gain**: (Difference in Entropy before and after splitting dataset on attribute A)

$$Gain = Entropy(S) - I(Attribute)$$

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

S. No.	Outlook	Temperature	Humidity	Windy	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rainy	Mild	High	Weak	Yes
5	Rainy	Cool	Normal	Weak	Yes
6	Rainy	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rainy	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rainy	Mild	High	Strong	No

P = 9

N = 5

Total = 14

- For each Attribute: (let say **Outlook**)
  - Calculate Entropy for each Values, i.e for 'Sunny', 'Rainy','Overcast'

Outlook	PlayTennis
Sunny	No
Sunny	No
Sunny	No
Sunny	Yes
Sunny	Yes

Outlook	PlayTennis
Rainy	Yes
Rainy	Yes
Rainy	No
Rainy	Yes
Rainy	No

Outlook	PlayTennis
Overcast	Yes
Overcast	Yes
Overcast	Yes
Overcast	Yes

Outlook	p	n	Entropy
Sunny	2	3	0.971
Rainy	3	2	0.971
Overcast	4	0	0

- Calculate **Average Information Entropy**:

$$I(Outlook) = \frac{p_{sunny} + n_{sunny}}{p+n} Entropy(Outlook = Sunny) +$$

$$\frac{p_{rainy} + n_{rainy}}{p+n} Entropy(Outlook = Rainy) +$$

$$\frac{p_{Overcast} + n_{Overcast}}{p+n} Entropy(Outlook = Overcast)$$

$$I(Outlook) = \frac{3+2}{9+5} * 0.971 + \frac{2+3}{9+5} * 0.971 + \frac{4+0}{9+5} * 0 \quad = 0.693$$

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

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

Values (outlook) :- Sunny, Overcast, Rain

$$Gain(S, outlook) = Entropy(S) - \left[\frac{5}{14} Entropy(S_{sunny}) \right] + \left(\frac{4}{14} Entropy(S_{overcast}) \right) + \left(\frac{5}{14} Entropy(S_{Rain}) \right)$$

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

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

$$Entropy(S_{sunny}) = -\left(\frac{2}{5} \log_2 \frac{2}{5}\right) - \left(\frac{3}{5} \log_2 \frac{3}{5}\right)$$

$$= 0.970$$

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

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

Entropy ($S_{overcast}$) = 0

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

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

$$Entropy(S_{Rain}) = -\left(\frac{3}{5} \log_2 \frac{3}{5}\right) - \left(\frac{2}{5} \log_2 \frac{2}{5}\right)$$

$$= 0.970$$

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

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

$$= 0.940 - \left[\left\{ \left(\frac{5}{14}\right) \times 0.9709 + \quad 0 \quad + \left(\frac{5}{14}\right) 0.9709 \right.\right.$$

$$= \quad 0.246$$

- For each Attribute: (let say **Temperature**)
  - Calculate Entropy for each Temp, i.e for 'Hot', 'Mild' and 'Cool'

Temperature	PlayTennis
Hot	No
Hot	No
Hot	Yes
Hot	Yes

Temperature	PlayTennis
Mild	Yes
Mild	No
Mild	Yes
Mild	Yes
Mild	Yes
Mild	No

Temperature	PlayTennis
Cool	Yes
Cool	No
Cool	Yes
Cool	Yes

Temperature	p	n	Entropy
Hot	2	2	1
Mild	4	2	0.918
Cool	3	1	0.811

- Calculate **Average Information Entropy**:

$$I(Temperature) = \frac{p_{hot} + n_{hot}}{p+n} Entropy(Temperature = Hot) +$$

$$\frac{p_{mild} + n_{mild}}{p+n} Entropy(Temperature = Mild) +$$

$$\frac{p_{Cool} + n_{Cool}}{p+n} Entropy(Temperature = Cool)$$

$$I(Temperature) = \frac{2+2}{9+5} * 1 + \frac{4+2}{9+5} * 0.918 + \frac{3+1}{9+5} * 0.811 => 0.911$$

- Calculate **Gain**: attribute is Temperature

$$Gain = Entropy(S) - I(Attribute)$$

$$Entropy(S) = 0.940$$

$$Gain(Temperature) = 0.940 - 0.911 = 0.029$$

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

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

V (Temperature) => Hot, mild, cool

$$G(S, Temperature) = Entropy(S) - \left\{ \left\{ \frac{4}{14} Entropy(S_{Hot}) \right\} + \left\{ \left(\frac{6}{14}, Entropy(S_{mild}) \right\} + \left\{ \frac{4}{14} Entropy(S_{cool}) \right\} \right\} = 0.029$$

- For each Attribute: (let say **Humidity**)
  - Calculate Entropy for each Humidity, i.e for 'High', 'Normal'

Humidity	PlayTennis
Normal	Yes
Normal	No
Normal	Yes
Normal	Yes
Normal	Yes
Normal	Yes
Normal	Yes

Humidity	PlayTennis
High	No
High	No
High	Yes
High	Yes
High	No
High	Yes
High	No

Humidity	p	n	Entropy
High	3	4	0.985
Normal	6	1	0.591

- Calculate **Average Information Entropy**:

$$I(Humidity) = \frac{p_{High} + n_{High}}{p+n} Entropy(Humidity = High) +$$

$$\frac{p_{Normal} + n_{Normal}}{p+n} Entropy(Humidity = Normal)$$

$$I(Humidity) = \frac{3+4}{9+5} * 0.985 + \frac{6+1}{9+5} * 0.591 => 0.788$$

- Calculate **Gain**: attribute is Humidity

$$Gain = Entropy(S) - I(Attribute)$$

$$Entropy(S) = 0.940$$

$$Gain(Humidity) = 0.940 - 0.788 = 0.152$$

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

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

$Gain(S, Humidity) = Entropy(S) -$

$\left[\left\{ \left(\frac{7}{14} Entropy(Humidity_{High}) \right) \right\} + \left\{ \frac{7}{14} Entropy(Humidity_{Normal}) \right\} \right]$

$= 0.151$

- For each Attribute: (let say **Windy**)
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

Windy	PlayTennis
Weak	No
Weak	Yes
Weak	Yes
Weak	Yes
Weak	No
Weak	Yes
Weak	Yes
Weak	Yes

Windy	PlayTennis
Strong	No
Strong	No
Strong	Yes
Strong	Yes
Strong	Yes
Strong	No

Windy	p	n	Entropy
Strong	3	3	1
Weak	6	2	0.811

- Calculate **Average Information Entropy**:

$$I(Windy) = \frac{p_{Strong} + n_{Strong}}{p+n} Entropy(Windy = Strong) + \frac{p_{Weak} + n_{Weak}}{p+n} Entropy(Windy = Weak)$$

$$I(Windy) = \frac{3+3}{9+5} * 1 + \frac{6+2}{9+5} * 0.811 => 0.892$$

- Calculate **Gain**: attribute is Windy

$$Gain = Entropy(S) - I(Attribute)$$

$$Entropy(S) = 0.940$$

$$Gain(Windy) = 0.940 - 0.892 = 0.048$$

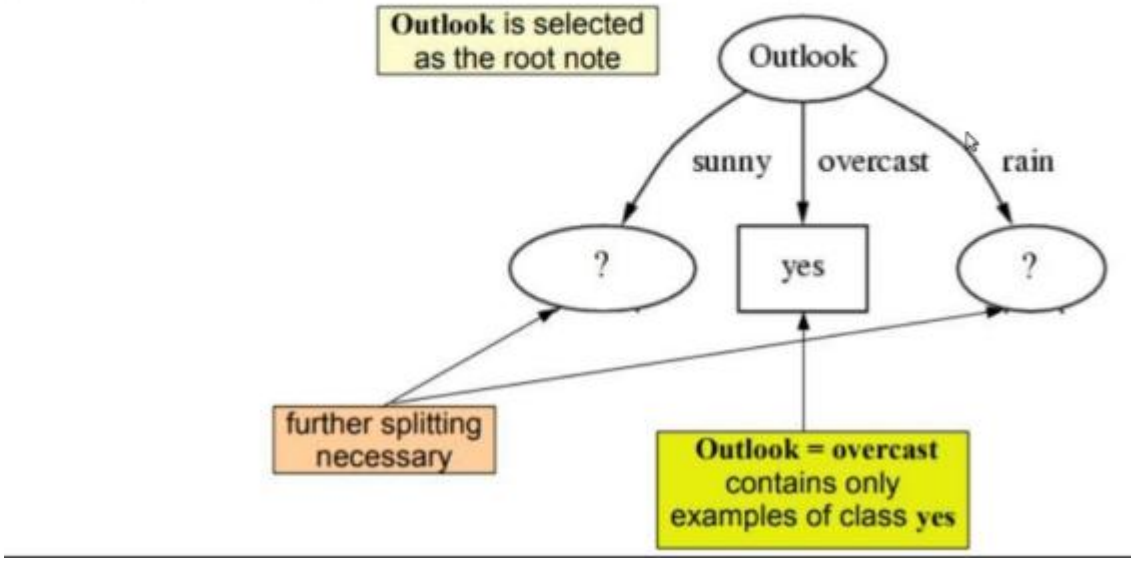
- PICK THE HIGHEST GAIN ATTRIBUTE.

Attributes	Gain
Outlook	0.247
Temperature	0.029
Humidity	0.152
Windy	0.048

ROOT NODE:

OUTLOOK

Outlook	Temperature	Humidity	Windy	PlayTennis
Overcast	Hot	High	Weak	Yes
Overcast	Cool	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes



- Repeat the same thing for sub-trees till we get the tree.

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

Outlook = "Sunny"

Outlook	Temperature	Humidity	Windy	PlayTennis
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Rainy	Mild	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

Outlook = "Rainy"

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

P= 2 N= 3 Total= 5

- ENTROPY:

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

$$Entropy(S_{sunny}) = \frac{-2}{2+3} log_2(\frac{2}{2+3}) - \frac{3}{2+3} log_2(\frac{3}{2+3})$$

=>0.971

- For each Attribute: (let say **Humidity**):
  - Calculate Entropy for each Humidity, i.e for 'High' and 'Normal'

Outlook	Humidity	PlayTennis
Sunny	High	No
Sunny	High	No
Sunny	High	No
Sunny	Normal	Yes
Sunny	Normal	Yes

Humidity	p	n	Entropy
high	0	3	0
normal	2	0	0

- Calculate **Average Information Entropy**: I(Humidity) = 0
- Calculate **Gain:** Gain = 0.971

- For each Attribute: (let say **Windy**):
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

Outlook	Windy	PlayTennis
Sunny	Strong	No
Sunny	Strong	Yes
Sunny	Weak	No
Sunny	Weak	No
Sunny	Weak	Yes

Windy	p	n	Entropy
Strong	1	1	1
Weak	1	2	0.918

- Calculate **Average Information Entropy**: I(Windy) = 0.951
- Calculate **Gain:** Gain = 0.020

- For each Attribute: (let say **Temperature**):
  - Calculate Entropy for each Windy, i.e for 'Cool', 'Hot' and 'Mild'

Outlook	Temperature	PlayTennis
Sunny	Cool	Yes
Sunny	Hot	No
Sunny	Hot	No
Sunny	Mild	No
Sunny	Mild	Yes

Temperature	p	n	Entropy
Cool	1	0	0
Hot	0	2	0
Mild	1	1	1

- Calculate **Average Information Entropy**: I(Temp) = 0.4
- Calculate **Gain:** Gain = 0.571

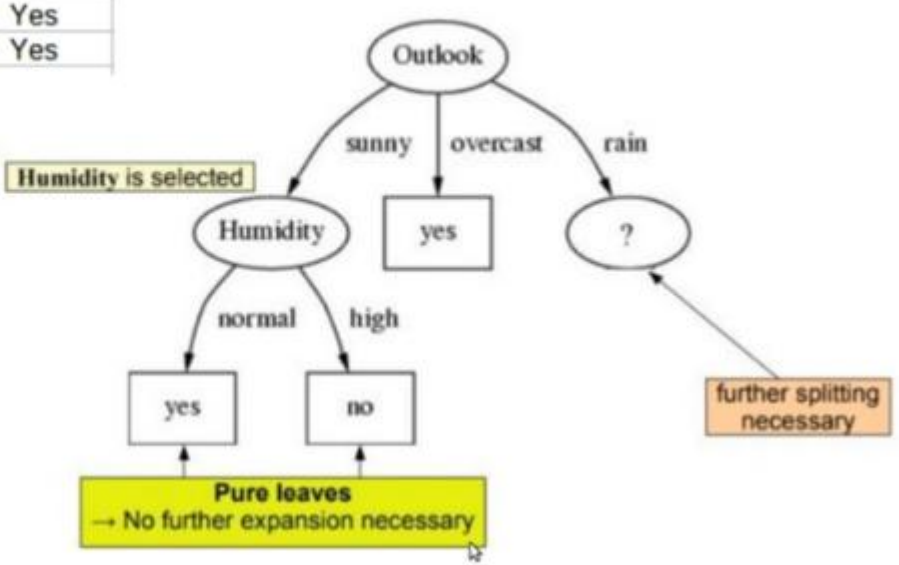
- PICK THE HIGHEST GAIN ATTRIBUTE.

Attributes	Gain
Temperature	0.571
Humidity	0.971
Windy	0.02

NEXT NODE IN SUNNY:

HUMIDITY

Outlook	Humidity	PlayTennis
Sunny	High	No
Sunny	High	No
Sunny	High	No
Sunny	Normal	Yes
Sunny	Normal	Yes



Outlook	Temperature	Humidity	Windy	PlayTennis
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Rainy	Mild	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

P= 3 N= 2

Total = 5

- ENTROPY:

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

$$Entropy(S_{Rainy}) = \frac{-3}{3+2} log_2(\frac{3}{3+2}) - \frac{2}{3+2} log_2(\frac{2}{2+3})$$

=>0.971

- For each Attribute: (let say **Humidity**):
  - Calculate Entropy for each Humidity, i.e for 'High' and 'Normal'

Outlook	Humidity	PlayTennis
Rainy	High	Yes
Rainy	High	No
Rainy	Normal	Yes
Rainy	Normal	No
Rainy	Normal	Yes

Attribute	p	n	Entropy
High	1	1	1
Normal	2	1	0.918

- Calculate **Average Information Entropy**: I(Humidity) = 0.951
- Calculate **Gain:** Gain = 0.020

- For each Attribute: (let say **Windy**):
  - Calculate Entropy for each Windy, i.e for 'Strong' and 'Weak'

Outlook	Windy	PlayTennis
Rainy	Strong	No
Rainy	Strong	No
Rainy	Weak	Yes
Rainy	Weak	Yes
Rainy	Weak	Yes

Attribute	p	n	Entropy
Strong	0	2	0
Weak	3	0	0

- Calculate **Average Information Entropy**: I(Windy) = 0
- Calculate **Gain:** Gain = 0.971

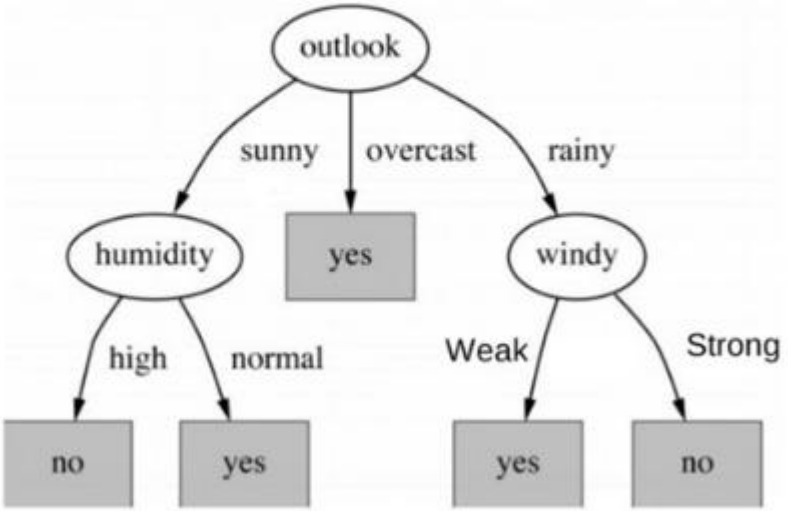
- PICK THE HIGHEST GAIN ATTRIBUTE.

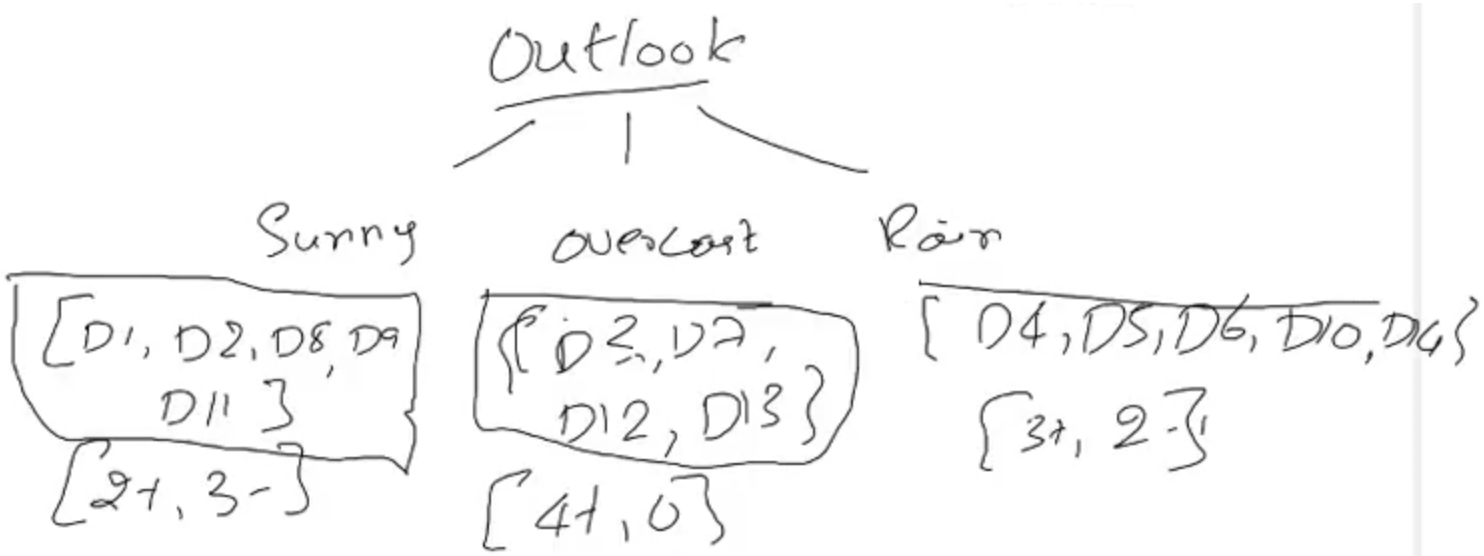
Attributes	Gain
Humidity	0.02
Windy	0.971
Temperature	0.02

NEXT NODE IN RAINY:

WINDY

Final decision tree




Outlook
Sunny
Overcast
Rain
[D1, D2, D8, D9, D11]
[2+, 3-]
[D3, D7, D12, D13]
[4+, 0]
[D4, D5, D6, D10, D14]
[3+, 2-]

$S_{Sunny} = \{D1, D2, D8, D9, D11\}$

$Gain(S_{Sunny}, Humidity) =$

$Gain(S_{Sunny}, Temperature) =$

$Gain(S_{Sunny}, Wind) =$

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

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

$$Gain = Entropy(S_{sunny}) - \left[\left(\frac{3}{5} \right) Entropy(Humidity_{high}) \right]$$

$$+ \left[\left(\frac{2}{5} \right) Entropy(Humidity_{normal}) \right]$$

$$Entropy(S_{sunny}) = -\left\{ \frac{2}{5} \log_2 \left(\frac{2}{5} \right) \right\} - \left\{ \frac{3}{5} \log_2 \frac{3}{5} \right\}$$

$$= 0.970$$

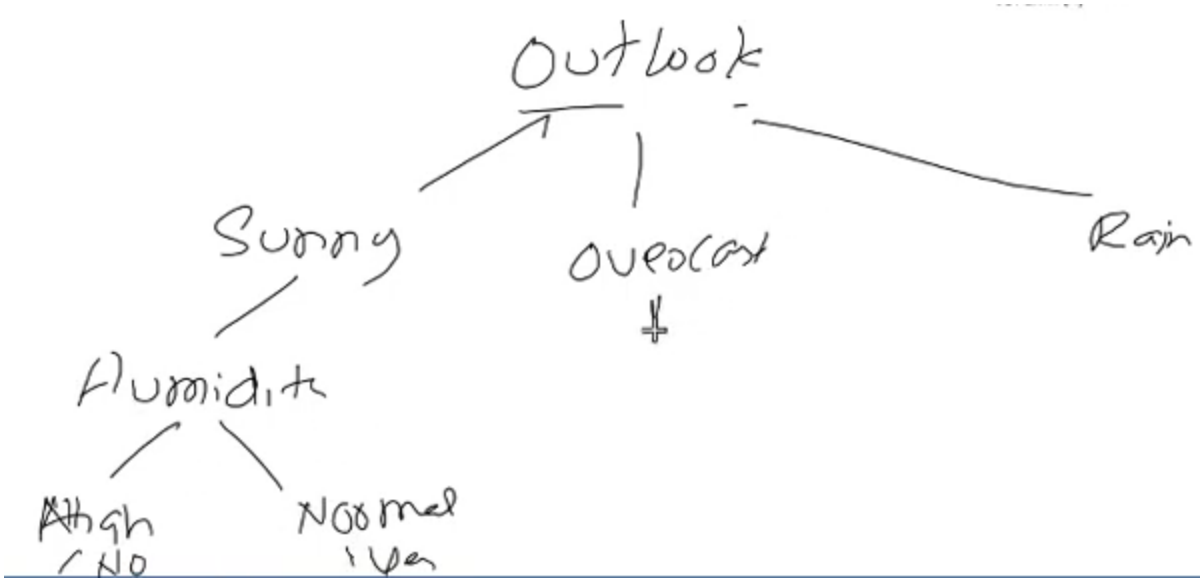
$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

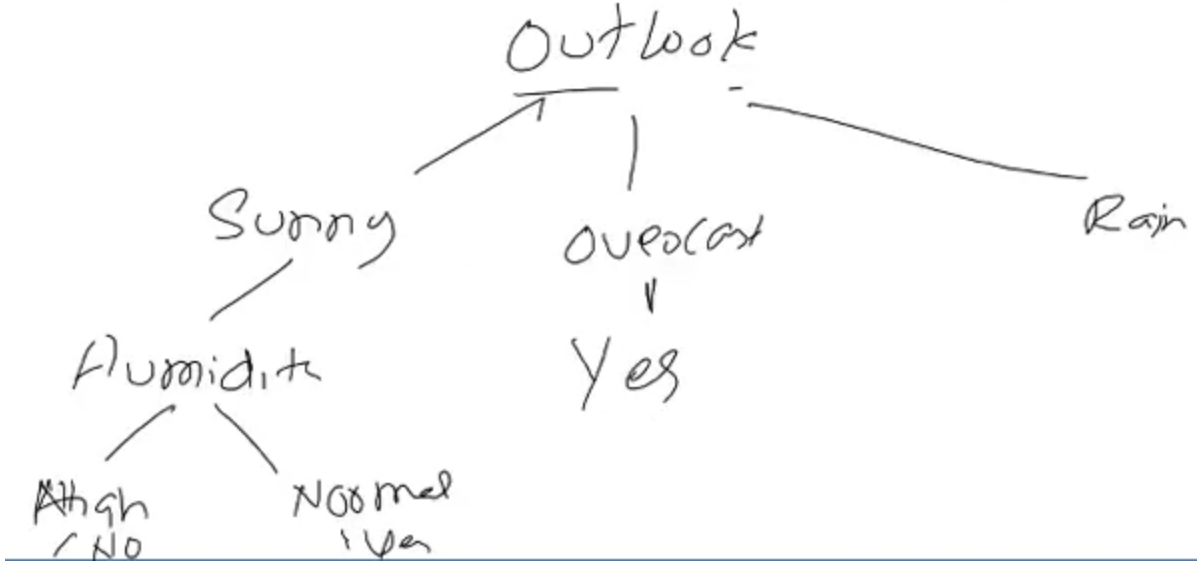
$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Gain (outlook: Sunn, Humidity) = 0.970

Gain (outlook: Sunny; Temperature) = 0.570

Gain (outlook = Sunn, Wind) = 0.0192




Outlook
Sunny
Overcast
Rain
Humidity
Yes
High
No
Normal
Yes

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

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

Gain: Entropy(O=R) $-\left[\left(\frac{T_H}{5}\right) Entropy\left(T_H\right) + \left(\frac{T_M}{5}\right.\right.$ Entropy $\left(T_M\right) +$

$\left.\left.\left(\frac{T_{Cool}}{5}\right) Entropy\left(T_{Cool}\right)\right)\right]$

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

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

$E(O=R, T=+1) = 0$

$E(O=R, T=M) = -\left(\frac{2}{3} \log_2 \frac{2}{3}\right) - \left(\frac{1}{3} \log_2 \frac{1}{3}\right)$

$= 0.918$

$E(O=R, T=C) = 1$

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

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

$$Gain(O:R,T) = 0.970 - \left\{ \frac{0}{5} \times 0 + \frac{3}{5} \times 0.918 + \left(\frac{2}{5}\right) \times 1 \right\}$$

$$= 0.0198$$

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

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

$$Gain(O:R, Wind) = 0.970 - \left\{ \frac{3}{5} \times Ent.(W:Weak) + \frac{2}{5} \times E(W:Strong) \right\}$$

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

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

$E(O=R, Wind=Weak) = 0$

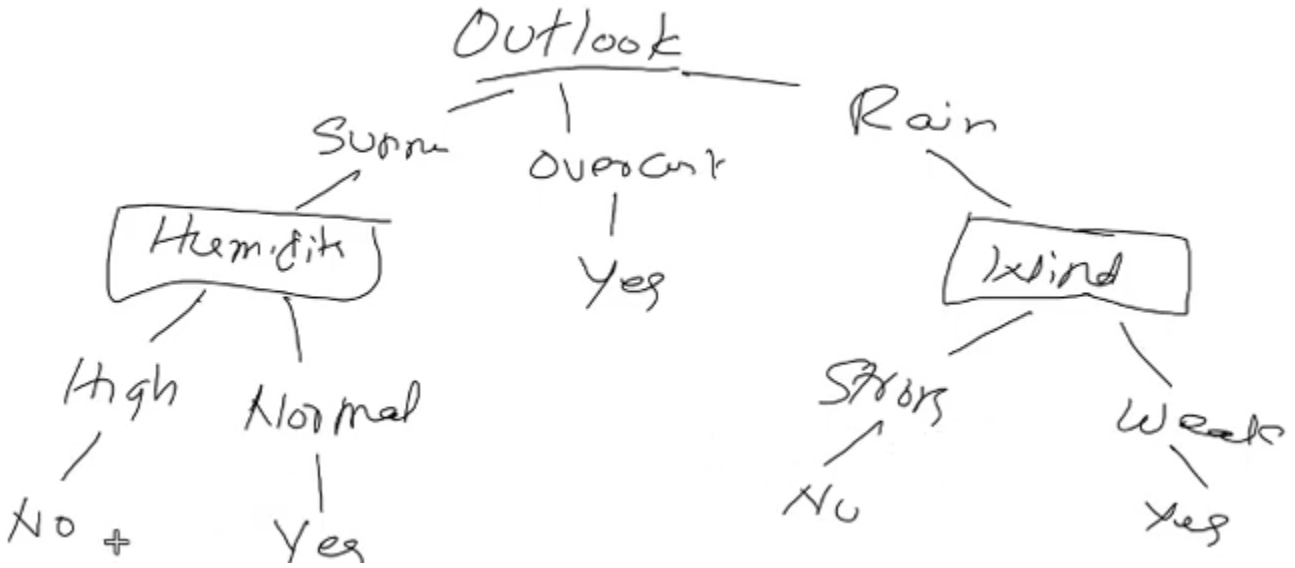
$E(O=R, Wind=Strong) = 0$

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

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

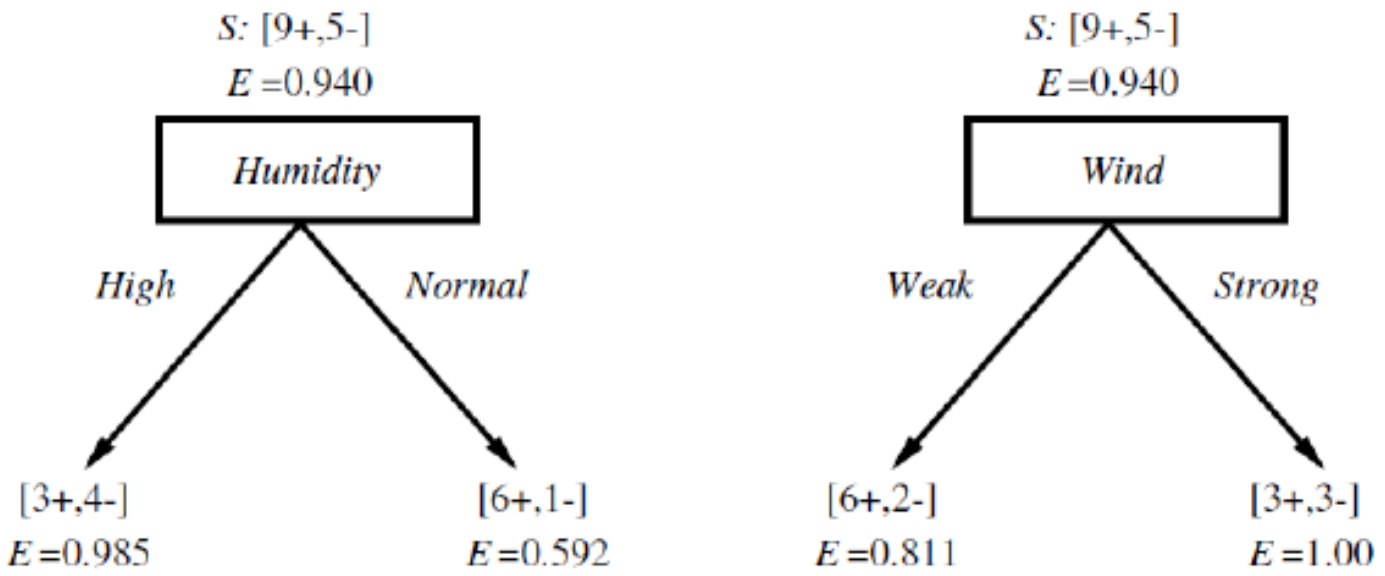
$$Gain(O=R, Wind) = 0.970 - \left[\frac{3}{5} \times 0 + \frac{2}{5} \times 0 \right]$$

$$= 0.970$$


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

•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

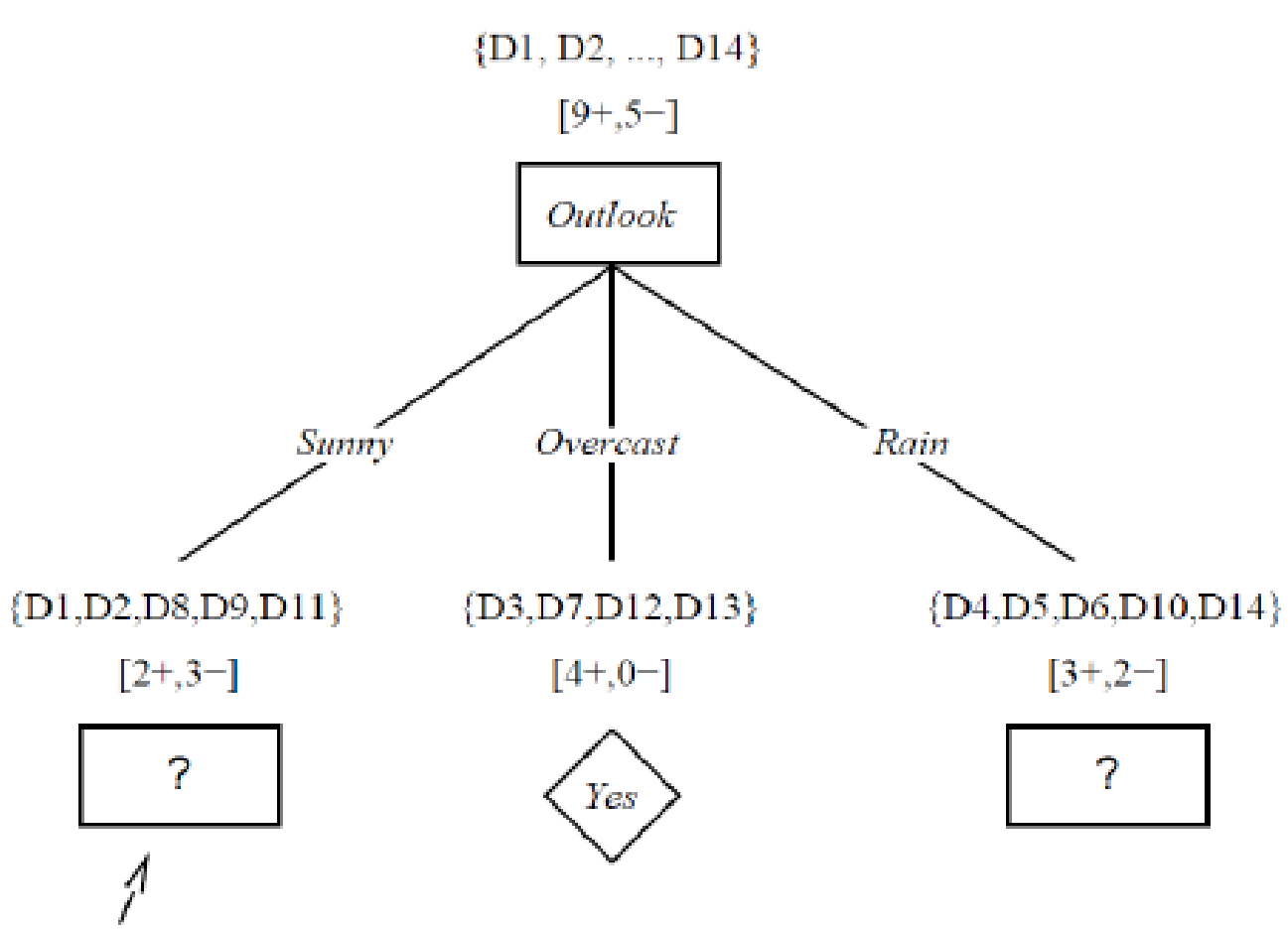
Gain(S, Outlook) = 0.246

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.*



Which attribute should be tested here?

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

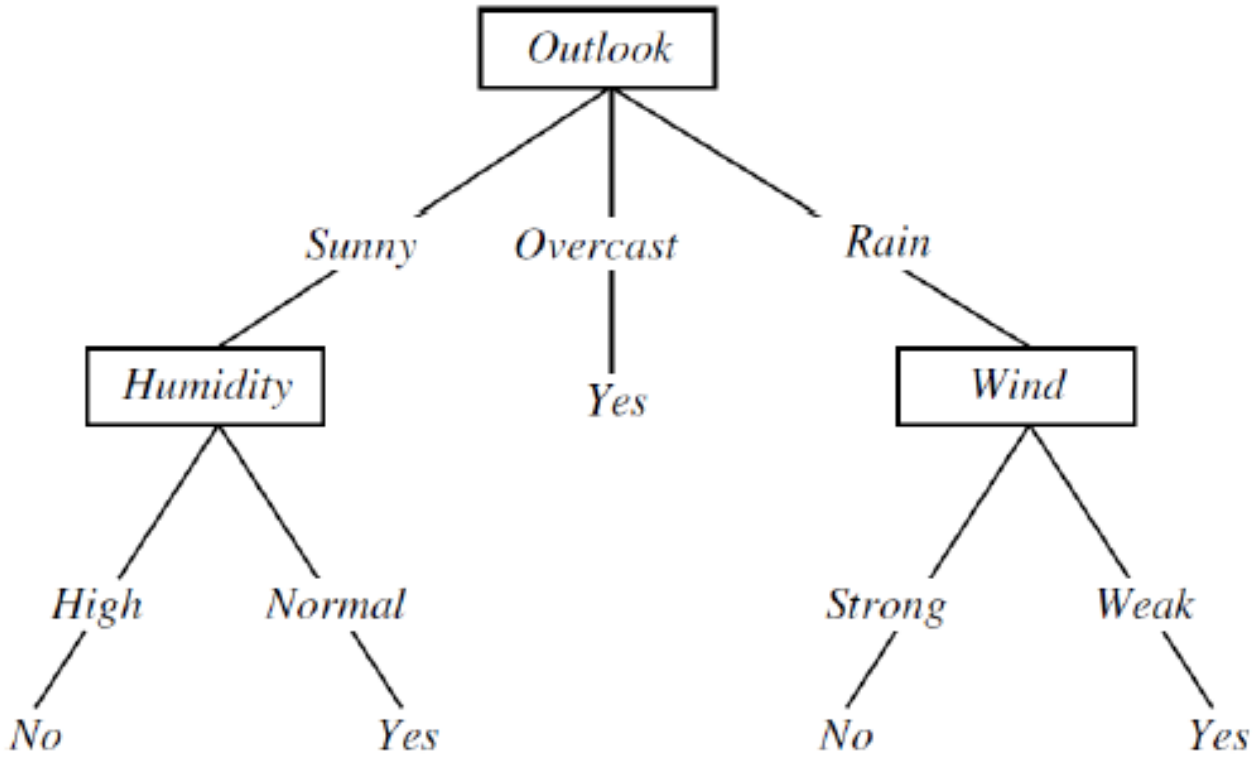
Gain (S_{sunny}, *Wind*) = .970 − (2/5) 1.0 − (3/5) .918 = .019

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

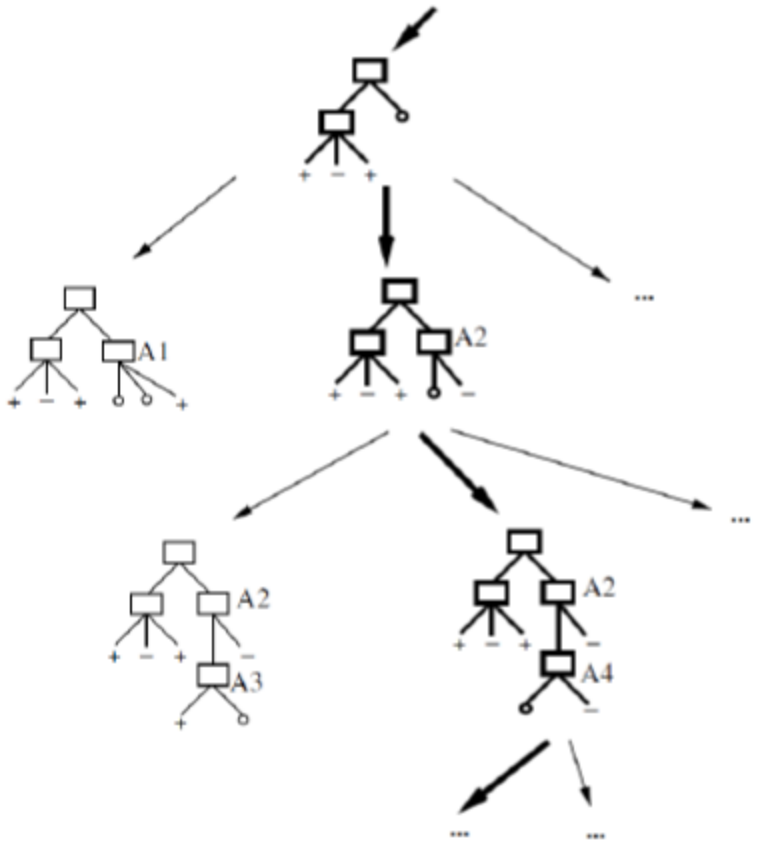


Figure: Hypothesis space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the information gain heuristic.

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 that optimally resolve among these competing hypotheses

3. *ID3 in its pure form performs no backtracking in its search. Once it selects an attribute to 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 resulting search 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.

References

•Machine Learning, M M. Mitchell, McGraw Hill
•Internet(WWW)