



Learning System

Artificial Intelligence

Learning systems, topics : Definition, learning agents, components of learning system, paradigms of machine learning. Rote Learning : learning by memorization, learning something by repeating. Learning from example : Induction, Winston's learning, Version spaces - learning algorithm (generalization and specialization tree), Decision trees - ID3 algorithm. Explanation Based Learning (EBL) : general approach, EBL architecture, EBL system, generalization problem, explanation structure. Discovery : theory driven – AM system, data driven - BACON system. Clustering : distance functions, K-mean clustering algorithm. Learning by analogy; Neural Net – Perceptron; Genetic Algorithm. Reinforcement Learning : RL Problem, agent - environment interaction, RL tasks, Markov system, Markov decision processes, agent's learning task, policy, reward function, maximize reward, value functions.

Learning System

Artificial Intelligence

Topics

(Lectures 31, 32, 33, 34 4 hours)

Slides

1. What is Learning

03-09

Definition, learning agents, components of learning system; Paradigms of machine learning.

2. Rote Learning

10

Learning by memorization, Learning something by repeating.

3. Learning from Example : Induction

11-38

Winston's learning, Version spaces - learning algorithm (generalization and specialization tree), Decision trees - ID3 algorithm.

4. Explanation Based Learning (EBL)

39-43

General approach, EBL architecture, EBL system, Generalization problem, Explanation structure.

5. Discovery

44-52

Theory driven - AM system, Data driven - BACON system

6. Clustering

53-62

Distance functions, K-mean clustering - algorithm.

7. Analogy

63

8. Neural net and Genetic Learning

64-67

Neural Net - Perceptron; Genetic learning - Genetic Algorithm.

9. Reinforcement Learning

68-80

RL Problem : Agent - environment interaction, key Features; RL tasks, Markov system, Markov decision processes, Agent's learning task, Policy, Reward function, Maximize reward, Value functions.

10 References

81

Learning

Machine Learning

What is learning ? Some Quotations

- **Herbert Simon, 1983**

Learning denotes changes in a system that enable a system to do the same task more efficiently the next time.

- **Marvin Minsky, 1986**

Learning is making useful changes in the workings of our minds.

- **Ryszard Michalski, 1986**

Learning is constructing or modifying representations of what is being experienced.

- **Mitchell, 1997**

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**.

1. What is Learning

Learning denotes changes in a system that enable the system to do the same task more efficiently next time.

Learning is an important feature of "Intelligence".

1.1 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**. (Mitchell 1997)

This means :

Given : A task **T**

A performance measure **P**

Some experience **E** with the task

Goal : Generalize the experience in a way that allows to improve your performance on the task.

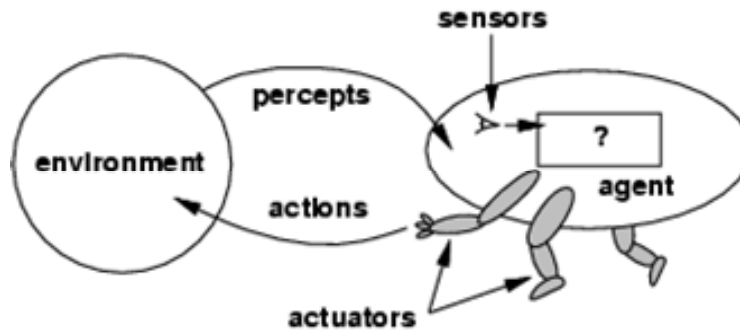
Why do you require Machine Learning ?

- Understand and improve efficiency of human learning.
- Discover new things or structure that is unknown to humans.
- Fill in skeletal or incomplete specifications about a domain.

2 Learning Agents

An **agent** is an entity that is capable of **perceiving** and do **action**.

An agent can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**.



Environment / Agent	Sensors	Actuators
Human agent	Eyes, ears, etc	Leg, hands, mouth
Robotic agent	Cameras, IR range finders	motors
Software agent	Key stroke, File contents	Displays to screen, write files

In computer science an **agent is a software agent** that assists users and acts in performing computer-related tasks.

● **Intelligent Agent (Learning Agent)**

Agent is an entity that is capable of **perceiving** and do action.

In computer science an agent is a software agent.

In artificial intelligence, the term used for agent is an **intelligent agent**.

Learning is an important feature of "Intelligence".

Percept : agent's perceptual inputs

Percept sequence : history of everything the agent has perceived

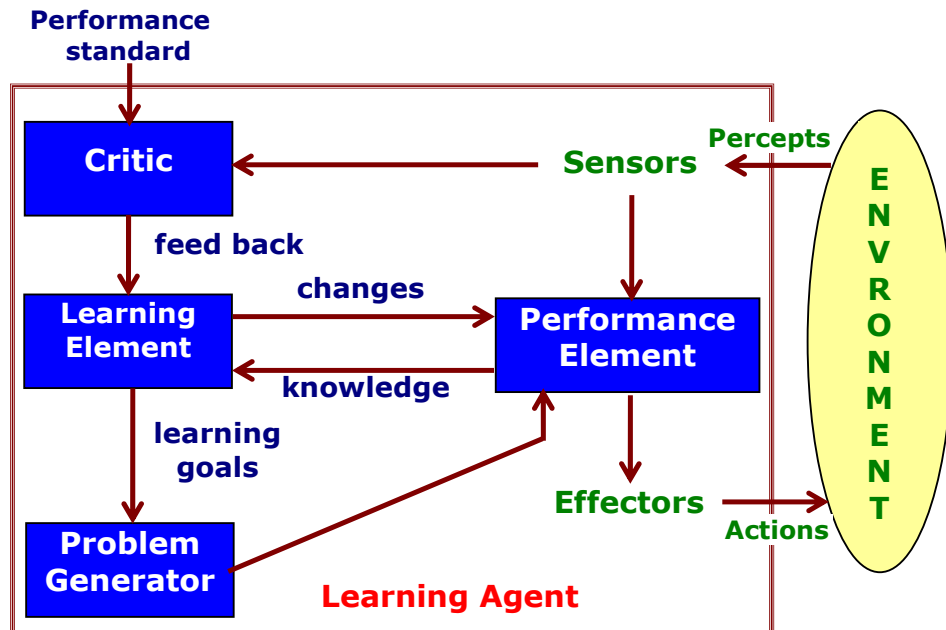
Agent function : describes agent's behavior

Agent program : Implements agent's function

Learning Agent consist of four main components :

- ◇ Learning element,
- ◇ Performance element,
- ◇ Critic, and
- ◇ Problem generator.

Components of a Learning System



The components are described in the next slide.

Components of a Learning System

- **Performance Element:** The Performance Element is the agent itself that acts in the world. It takes in percepts and decides on external actions.
- **Learning Element:** It responsible for making improvements, takes knowledge about performance element and some feedback, determines how to modify performance element.
- **Critic:** Tells the Learning Element how agent is doing (success or failure) by comparing with a fixed standard of performance.
- **Problem Generator:** Suggests problems or actions that will generate new examples or experiences that will aid in training the system further.

Example : Automated Taxi on city roads

- **Performance Element:** Consists of knowledge and procedures for driving actions.
e.g., turning, accelerating, braking are performance element on roads.
- **Learning Element:** Formulates goals.
e.g., learn rules for braking, accelerating, learn geography of the city.
- **Critic:** Observes world and passes information to learning element.
e.g., quick right turn across three lanes of traffic, observe reaction of other drivers.
- **Problem Generator:** Try south city road.

2 Paradigms of Machine Learning

- **Rote Learning:** Learning by **memorization**; One-to-one mapping from inputs to stored representation; Association-based storage and retrieval.
- **Induction:** Learning from **examples**; A form of supervised learning, uses specific examples to reach general conclusions; Concepts are learned from sets of labeled instances.
- **Clustering:** Discovering similar **group**; Unsupervised, Inductive learning in which natural classes are found for data instances, as well as ways of classifying them.
- **Analogy:** Determine **correspondence** between two different representations that come from Inductive learning in which a system transfers knowledge from one database into another database of a different domain.

- **Discovery:** Learning without the help from a **teacher**; Learning is both inductive and deductive. It is **deductive** if it proves theorems and discovers concepts about those theorems. It is **inductive** when it raises conjectures (guess). It is unsupervised, specific goal not given.
- **Genetic Algorithms:** Inspired by natural **evolution**; In the natural world, the organisms that are poorly suited for an environment die off, while those well-suited for it prosper. Genetic algorithms search the space of individuals for good candidates. The "goodness" of an individual is measured by some fitness function. Search takes place in parallel, with many individuals in each generation.
- **Reinforcement:** Learning from **feedback** (+ve or -ve reward) given at end of a sequence of steps. Unlike supervised learning, the reinforcement learning takes place in an environment where the agent cannot directly compare the results of its action to a desired result. Instead, it is given some reward or punishment that relates to its actions. It may win or lose a game, or be told it has made a good move or a poor one. The job of reinforcement learning is to find a successful function using these rewards.

2. Rote Learning

Rote learning technique avoids understanding the inner complexities but focuses on memorizing the material so that it can be recalled by the learner exactly the way it was read or heard.

- **Learning by Memorization** which avoids understanding the inner complexities the subject that is being learned; Rote learning instead focuses on memorizing the material so that it can be recalled by the learner exactly the way it was read or heard.
- **Learning something by Repeating** over and over and over again; saying the same thing and trying to remember how to say it; it does not help us to understand; it helps us to remember, like we learn a poem, or a song, or something like that by rote learning.

3. Learning from Example : Induction

A process of learning by **example**. The system tries to induce a general rule from a set of observed instances. The learning methods extract rules and patterns out of massive data sets.

The learning processes belong to **supervised learning**, does classification and constructs class definitions, called induction or concept learning.

The techniques used for constructing class definitions (or concept leaning) are :

- Winston's Learning program
- Version Spaces
- Decision Trees

Note : The Decision tree technique has been explained in all details while other two techniques are briefly described in the next few slides.

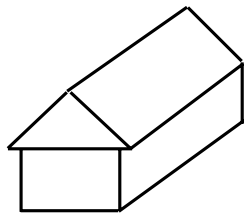
1 Winston's Learning

Winston (1975) described a **Blocks World Learning program**. This program operated in a simple blocks domain. The goal is to construct representation of the definition of concepts in the blocks domain.

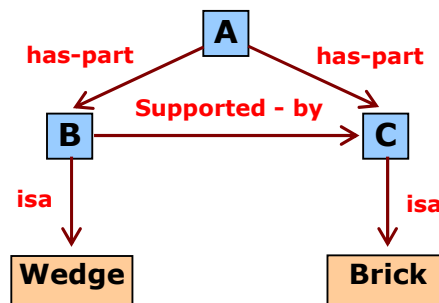
Example : Concepts such a "house".

- Start with input, a line drawing of a blocks world structure.
It learned Concepts **House, Tent, Arch** as :
brick (rectangular block) with a **wedge** (triangular block) suitably placed on top of it, **tent** – as 2 wedges touching side by side, or an **arch** – as 2 non-touching bricks supporting a third wedge or brick.
- The program for Each concept is learned through near miss. A near miss is an object that is not an instance of the concept but a very similar to such instances.
- The program uses procedures to analyze the drawing and construct a semantic net representation.
- An example of such an structural for the house is shown below.

Object - house



Semantic net



- Node **A** represents entire structure, which is composed of two parts : node **B**, a Wedge, and node **C**, a Brick.
Links in network include **supported-by**, **has-part**, and **isa**.

Winston's Program

- Winston's program followed 3 basic steps in concept formulation:
 1. Select one known instance of the concept.
Call this the **concept definition**.
 2. Examine definitions of other known instance of the concept.
Generalize the definition to include them.
 3. Examine descriptions of **near misses**.
Restrict the definition to **exclude** these.
- Both steps 2 and 3 of this procedure rely heavily on comparison process by which similarities and differences between structures can be detected.
- Winston's program can be similarly applied to learn other concepts such as "ARCH".

2 Version Spaces

A Version Space is a hierarchical representation of knowledge that keeps track of all the useful information supplied by a sequence of learning examples without remembering any of the examples. The version space method is a concept learning process. It is another approach to learning by Mitchell (1977).

● Version Space Characteristics

- Represents all the alternative plausible descriptions of a heuristic. A plausible description is one that is applicable to all known +ve examples and no known -ve example.
- A version space description consists of two complementary trees:
 - ‡ one, contains nodes connected to overly general models, and
 - ‡ other, contains nodes connected to overly specific models.
- Node values/attributes are discrete.

● Fundamental Assumptions

- The data is correct ie no erroneous instances.
- A correct description is a conjunction of some attributes with values.

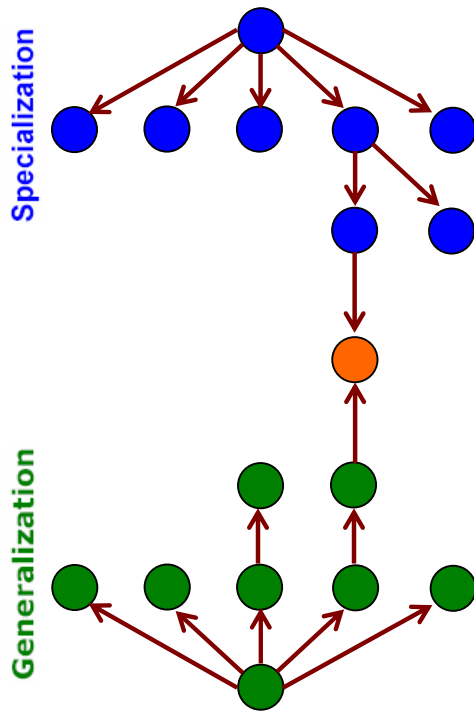
Version Space Methods

A version space is a hierarchical representation of knowledge.

Version space method is a concept learning process accomplished by managing multiple models within a version space.

It is specialization of general models and generalization of specific models.

Version Space diagram described below consists : a **Specialization** tree (colored Blue) and a **Generalization** tree (colored green) .



Version Space diagram

Top 1st row : the top of the tree, we have the most general hypotheses.

Top 2nd row : This row is an expanded version of the first. This row of hypotheses is slightly more specific than the root node.

Top 3rd row : As training data (positive examples) is processed, the inconsistent nodes are removed from the general specification.

Finally, we converge to a solution.

Bottom 3rd row : Any hypothesis that is inconsistent with the training data (negative instances) is removed from the tree.

Bottom 2nd row : The specific hypothesis is expanded to form more nodes that are slightly more general.

Bottom 1st row : This is the most specific hypothesis.

Tree nodes :

- ‡ each node is connected to a model.
- ‡ nodes in the generalization tree are connected to a model that matches everything in its sub-tree.
- ‡ nodes in the specialization tree are connected to a model that matches only one thing in its sub-tree.

Links between nodes :

- ‡ denotes generalization relations in a generalization tree, and
- ‡ denotes specialization relations in a specialization tree.

● **Version Space Convergence**

Generalization and Specialization leads to Version Space convergence.

The key idea in version space learning is that specialization of the general models and generalization of the specific models may ultimately lead to just one correct model that matches all observed positive examples and does not match any negative examples. This means :

- Each time a **+ve** example is used to specialize the general models, then those specific models that match the **-ve** example are eliminated.
- Each time a **-ve** example is used to generalize the specific models, those general models that fail to match the **+ve** example are eliminated.
- Finally, the positive and negative examples may be such that only one general model and one identical specific model survive.

● **Version Space Learning Algorithm:** Candidate – Elimination

The Candidate – Elimination algorithm finds all describable hypotheses that are consistent with the observed training examples.

The version space method handles +ve and -ve examples symmetrically.

- **Given** : A representation language and a set of positive and negative examples expressed in that language.
- **Compute** : A concept description that is consistent with all the positive examples and none of the negative examples.

Note : The methodology / algorithm is explained in the next slide.

[Continued from previous slide - version space]

- **Version Space Search Algorithm**

- ‡ Let **G** be the set of maximally general hypotheses.

- Initialize G**, to contain most general pattern, a null description.

- ‡ Let **S** be the set of maximally specific hypotheses.

- Initialize S**, to contain one element : the first positive example.

- ‡ **Accept a new training example.**

- ◇ For each new **positive training** example **p** do :

- 1. Delete all members of **G** that fail to match **p** ;

- 2. For every **s** in **S** do

- If **s** does not match **p**, then replace **s** with its maximally

- specific generalizations** that match **p**;

- 3. Delete from **S** any hypothesis that is subsumed, a more comprehensive, by other hypothesis in **S**;

- 4. Delete from **S** any hypothesis more general than some hypothesis in **G**.

- ◇ For each new **negative training** example **n** do :

- 1. Delete all members of **S** that match **n**;

- 2. For each **g** in **G** do

- If **g** match **n**, then replace **g** with its maximally

- general specializations** that do not match **n**;

- 3. Delete from **G** any hypothesis more specific than some other hypothesis in **G**;

- 4. Delete from **G** any hypothesis that is not more general than some hypothesis in **S**;

- ◇ If **S** and **G** are both singleton sets, then

- 1. If they are identical, output their value and halt.

- 2. If they are different, the training cases were inconsistent; output this result and halt.

- 3. Else continue accepting new training examples.

- ‡ The algorithm stops when it runs out of data;

- 1. If 0 , no consistent description for the data in the language.

- 2. If 1 , answer (version space converges).

- 3. If 2 , all descriptions in the language are implicitly included.

Version Space – Problem 1

Learning the concept of "Japanese Economy Car"

[Ref: E. Rich, K. Knight, "Artificial Intelligence", McGraw Hill, Second Edition]

Examples 1 to 5 of features

Country of Origin, Manufacturer, Color, Decade, Type

Example	Origin	Manufacturer	Color	Decade	Type	Example Type
1.	Japan	Honda	Blue	1980	Economy	Positive
2.	Japan	Toyota	Green	1970	Sports	Negative
3.	Japan	Toyota	Blue	1990	Economy	Positive
4.	USA	Chrysler	Red	1980	Economy	Negative
5.	Japan	Honda	White	1980	Economy	Positive

Step by step solution : Symbol for Prune ✂

1. Initialize : to maximally general and maximally specific hypotheses

‡ Initialize **G** to a null description, means all features are variables.

$$G = { (?, ?, ?, ?, ?) }$$


‡ Initialize **S** to a set of maximally specific hypotheses;

Apply a **positive example**;

Apply Example 1 (Japan, Honda, Blue, 1980, Economy)

$$S = { (Japan, Honda, Blue, 1980, Economy) }$$

‡ Resulting Specialization (blue) and Generalization (green) tree nodes

$$G = (?, ?, ?, ?, ?)$$




$$S = { (Japan, Honda, Blue, 1980, Economy) }$$

These two models represent the most general and the most specific heuristics one might learn.

The actual heuristic to be learned, "Japanese economy car", probably lies between them somewhere within the version space.

2. Apply Example 2 (Japan, Toyota, Green, 1970, Sports)

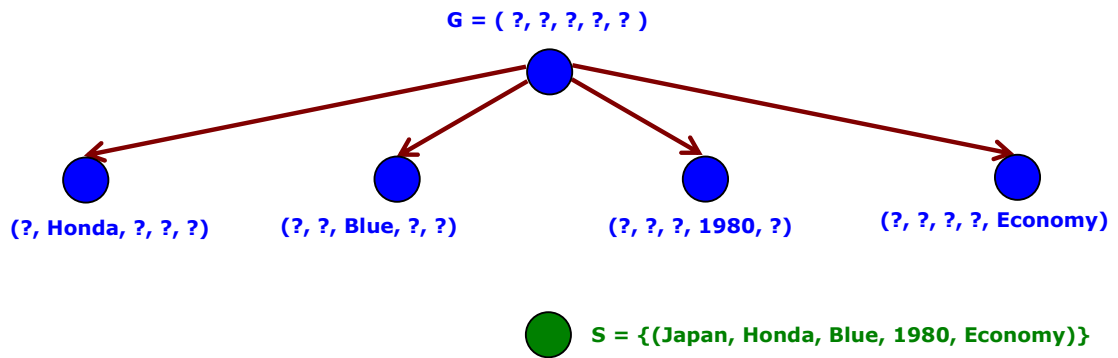
It is a negative example.

- ⊕ Specialize **G**, to exclude the negative example;
i.e., **G** must be specialized in such a way that the negative example is no longer in the version space.

The available specializations are :

$$\mathbf{G} = \{ (? , \mathbf{Honda} , ? , ? , ?) , (? , ? , \mathbf{Blue} , ? , ?) , (? , ? , ? , \mathbf{1980} , ?) , (? , ? , ? , ? , \mathbf{Economy}) \}$$

- ⊕ Resulting Specialization (blue) and Generalization (green) tree nodes and links



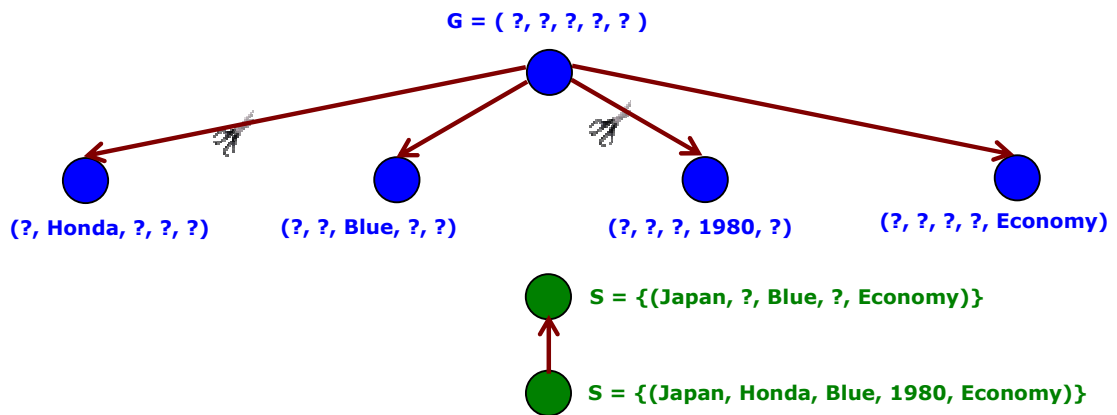
3. Apply Example 3 (Japan, Toyota, Blue, 1990, Economy)

It is a **positive example**.

≠ Prune **G**, to exclude descriptions inconsistent with the positive example; i.e., remove from the **G** set any descriptions that are inconsistent with the positive example. The new **G** set is **G = { (?, ?, Blue, ?, ?), (?, ?, ?, ?, Economy) }**

≠ Generalize **S**, to include the new example. The new **S** set is **S = { (Japan, ?, Blue, ?, Economy) }**

≠ Resulting Specialization (blue) and Generalization (green) tree nodes and links :



≠ At this point, the new **G** and **S** sets specify a version space that can be translated roughly in to English as : The concept may be as specific as "Japanese, blue economy car".

4. Apply Example 4 (USA, Chrysler, Red, 1980, Economy)

It is another negative example.

- ⊕ Specialize **G** to exclude the negative example; (but stay consistent with **S**) i.e.; **G** set must avoid covering this new example.

The new **G** set is

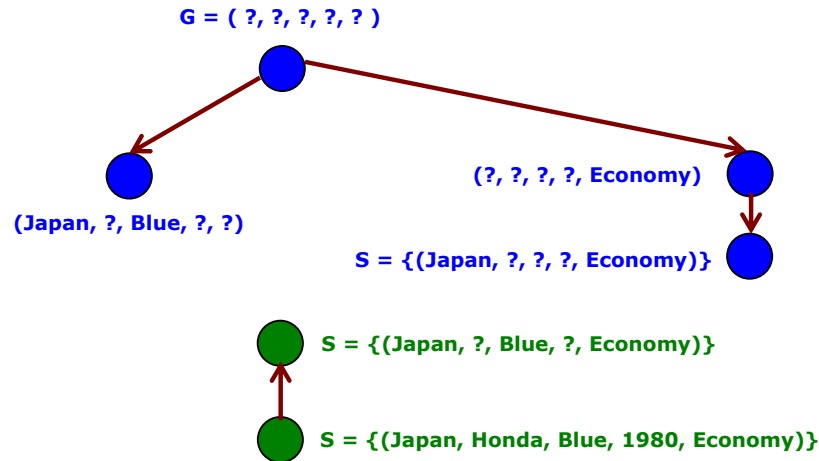
$$\mathbf{G} = \{ (? , ? , \mathbf{Blue} , ? , ?) , (\mathbf{Japan} , ? , ? , ? , \mathbf{Economy}) \}$$

- ⊕ Prune away all the specific models that match the negative example.

No match found therefore no change in **S**

$$\mathbf{S} = \{ (\mathbf{Japan} , ? , \mathbf{Blue} , ? , \mathbf{Economy}) \}$$

- ⊕ Resulting Specialization (blue) and Generalization (green) tree nodes and links :



- ⊕ It is now clear that the car must be Japanese, because all description in the version space contain Japan as origin.

5. Apply Example 5 (Japan, Honda, White, 1980, Economy)

It is positive example.

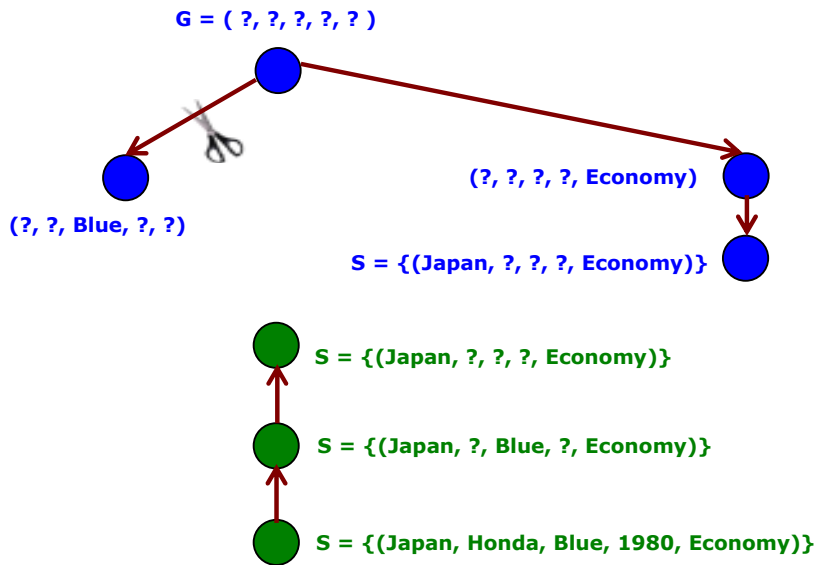
≠ Prune **G** to exclude descriptions inconsistent with positive example.

$$G = \{ (Japan, ?, ?, ?, Economy) \}$$

≠ Generalize **S** to include this positive example.

$$S = \{ (Japan, ?, ?, ?, Economy) \}$$

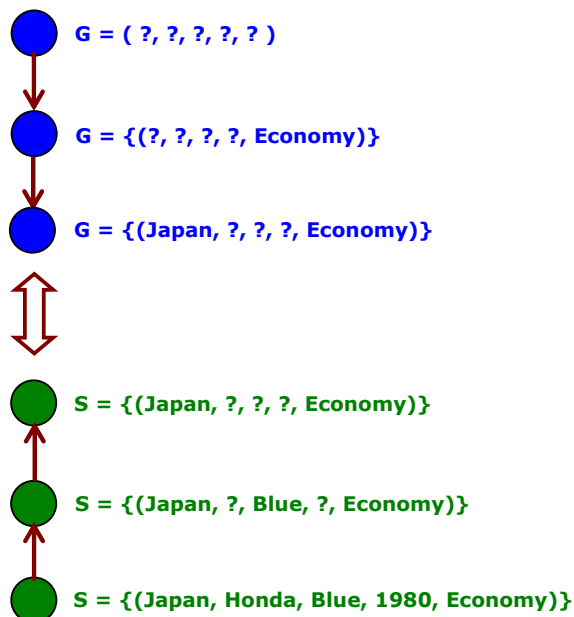
≠ Resulting Specialization (blue) and Generalization (green) tree nodes and links :



≠ Thus, finally **G** and **S** are singleton sets and **S = G**.

The algorithm has converged. No more examples needed.

Algorithm stops.



Decision Trees

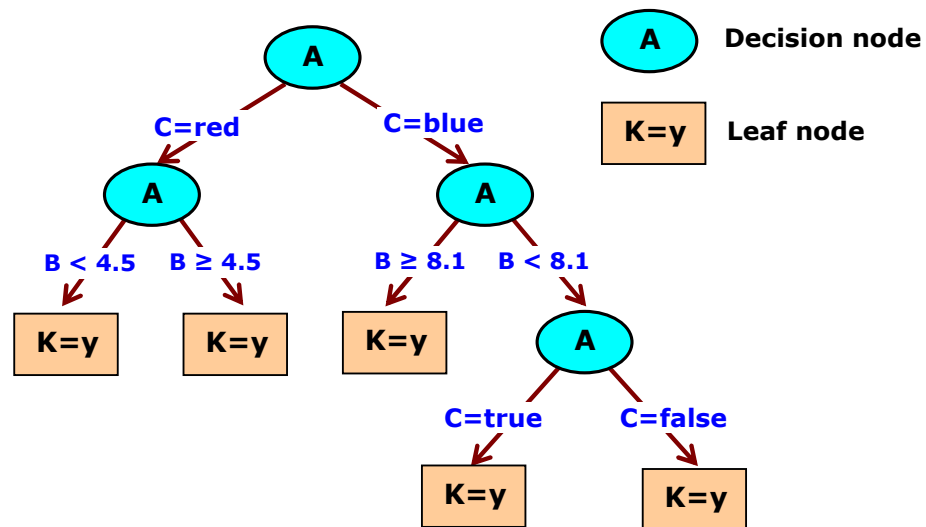
Decision trees are powerful tools for classification and prediction.

Decision trees represent rules. Rules are easily expressed so that humans can understand them or even directly use in a database access language like SQL so that records falling into a particular category may be retrieved.

● Description

- Decision tree is a classifier in the form of a tree structure where each node is either a leaf or decision node.
 - ‡ **leaf node** - indicates the target attribute (class) values of examples.
 - ‡ **decision node** - specify test to be carried on an attribute-value.
- Decision tree is a typical inductive approach to learn knowledge on classification. The conditions are :
 - ‡ **Attribute-value description:** Object or case must be expressible as a fixed collection of properties or attributes having discrete values.
 - ‡ **Predefined classes :** Categories to which examples are to be assigned must already be defined (ie supervised data).
 - ‡ Discrete classes: Classes must be sharply delineated; continuous classes broken up into vague categories as "hard", "flexible", "soft".
 - ‡ Sufficient data: Enough training cases to distinguish valid patterns.

● Example : A simple decision tree



● **Algorithm to generate a Decision Tree**

A decision tree program constructs a decision tree **T** from a set of training cases. The advantage of a decision tree learning is that a program, rather than a knowledge engineer, elicits knowledge from an expert.

ID3 Algorithm (Iterative Dichotomiser 3)

ID3 is based on the Concept Learning System (CLS) algorithm, developed J. Ross Quinlan, at the University of Sydney in 1975.

■ **Description**

- ‡ ID3 builds a decision tree from a set of "examples".
The "examples" have several attributes.
They belong to a class (yes or no).
- ‡ Leaf nodes contain the class name.
- ‡ Decision node is an attribute test .

■ **Attribute Selection**

How does ID3 decide which attribute is the best?

There are different criteria to select which attribute will become a test attribute in a given branch of a tree. A well known criteria is information gain. It uses a log function with a base of 2, to determine the number of bits necessary to represent a piece of information.

‡ **Entropy** measures information content in an attribute.

Shannon defines information entropy in terms of a discrete random variable X , with possible states (or outcomes) $x_1 \dots x_n$ as:

$$H(X) = \sum_{i=1}^n p(x_i) \bullet \log_2 (1/p(x_i)) = - \sum_{i=1}^n p(x_i) \bullet \log_2 p(x_i)$$

where $p(x_i) = P(X = x_i)$ is the probability of the i^{th} outcome of X .

‡ **Information gain** of an attribute X with respect to class attribute Y .

Let Y and X are discrete variables that take values in $\{y_1 \dots y_i \dots y_k\}$ and $\{x_1 \dots x_j \dots x_l\}$.

The information gain of a given attribute X with respect to the class attribute Y is the **reduction in uncertainty** about the value of Y when we know the value of X , is called **$I(Y ; X)$** .

Uncertainty

Uncertainty about the value of Y is measured by its **entropy**, **$H(Y)$** . uncertainty about the value of Y when we know the value of X is given by the **conditional entropy** of Y given X , i.e., **$H(Y |X)$** .

Information Gain of a given attribute X with respect to the class attribute Y is given by : **$I(Y ; X) = H(Y) - H(Y |X)$** . where

$H(Y)$ = $-\sum_{y_i=1}^k p(y_i) \bullet \log_2 p(y_i)$ is the initial entropy in Y ,

$H(Y |X)$ = $\sum_{x_j=1}^l p(x_j) \bullet H(Y |x_j)$ is conditional entropy of Y

given X where **$H(Y |x_j) = - \sum_{y_i=1}^k p(y_i | x_j) \bullet \log_2 p(y_i | x_j)$** is the entropy in Y given a particular answer x_j .

■ **Example**

Decide if the weather is amenable to play Golf or Baseball.

‡ **Training Data** Collected are Over 2 weeks

Day	Outlook	Temp	Humidity	wind	Play
1	sunny	85	85	week	no
2	sunny	80	90	strong	no
3	cloudy	83	78	week	yes
4	rainy	70	96	week	yes
5	rainy	68	80	week	yes
6	rainy	65	70	strong	no
7	cloudy	64	65	strong	yes
8	sunny	72	95	week	no
9	sunny	69	70	week	yes
10	rainy	75	80	week	yes
11	sunny	75	70	strong	yes
12	cloudy	72	90	strong	yes
13	cloudy	81	75	week	yes
14	rainy	71	85	strong	no

‡ **Learning set**

In the above example, two attributes, the Temperature and Humidity have continuous ranges. ID3 requires them to be discrete like hot, medium, cold, high, normal. Table below indicates the acceptable values.

Attribute	Possible Values		
Outlook	Sunny	Cloudy	Rainy
Temperature	Hot	Medium	Cold
Humidity	High	Normal	
Wind	Strong	Week	
Class	play	no play	
Decision	n (negative)	p (positive)	

‡ Assign discrete values to the Attributes

Partition the continuous attribute values to make them discrete, following the key mentioned below.

Temperature : Hot (H) 80 to 85 Medium (M) 70 to 75 Cold (C) 64 to 69

Humidity : High (H) 81 to 96 Normal (N) 65 to 80

Class : Yes (Y) play No (N) no play

Day	Outlook	Temp	Humidity	Wind	Class (play)
1	Sunny	85 Hot	85 High	week	no
2	Sunny	80 Hot	90 High	strong	no
3	Cloudy	83 Hot	78 High	week	yes
4	Rainy	70 Medium	96 High	week	yes
5	Rainy	68 Cold	80 Normal	week	yes
6	Rainy	65 Cold	70 Normal	strong	no
7	Cloudy	64 Cold	65 Normal	strong	yes
8	Sunny	72 Medium	95 High	week	no
9	Sunny	69 Cold	70 Normal	week	yes
10	Rainy	75 Medium	80 Normal	week	yes
11	Sunny	75 Medium	70 Normal	strong	yes
12	Cloudy	72 Medium	90 High	strong	yes
13	Cloudy	81 Hot	75 Normal	week	yes
14	Rainy	71 Medium	85 High	strong	no

‡ Attribute Selection

By applying definitions of Entropy and Information gain

- ◇ **Entropy** : From the definition, When **Y** is a discrete variables that take values in $\{y_1 \dots y_i \dots y_k\}$ then the entropy of **Y** is given by **H(Y)**

$$H(Y) = \sum_{i=1}^k p(y_i) \bullet \log_2 (1/p(y_i)) = - \sum_{i=1}^k p(y_i) \bullet \log_2 p(y_i)$$

where **p(y_i)** is the probability of the **ith** outcome of **Y**.

Apply his definition to the example stated before :

Given, **S** is a collection of 14 examples with 9 YES and 5 NO

then **Entropy(S) = - (9/14) Log₂ (9/14) - (5/14) Log₂ (5/14)**

$$= 0.940$$

Note that **S** is not an attribute but the entire sample set.

Entropy range from 0 ("perfectly classified") to 1 ("totally random").

[Continued in next slide]

[Continued from previous slide – Attribute election]

◇ **Information Gain** : From the definition, a given attribute **X** with respect to the class attribute **Y** is given by :

$$I(Y ; X) = H(Y) - H(Y | X) \text{ where}$$

$$H(Y) = - \sum_{y_i=1}^k p(y_i) \bullet \log_2 p(y_i) \text{ is the initial entropy in } Y,$$

$$H(Y | X) = \sum_{x_j=1}^l p(x_j) \bullet H(Y | x_j) \text{ is conditional entropy of } Y$$

$$\text{given } X \text{ where } H(Y | x_j) = - \sum_{y_i=1}^k p(y_i | x_j) \bullet \log_2 p(y_i | x_j) \text{ is}$$

entropy in **Y** given a particular answer **x_j** .

Apply this definition to the example stated above :

Information **Gain (S, A)** of the example set **S** on attribute **A** is

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum ((|S_v| / |S|) * \text{Entropy}(S_v)) \text{ where}$$

\sum sum over each value **v** of all possible values of attribute **A**

S_v is subset of **S** for which attribute **A** has value **v**

|S_v| is number of elements in **S_v**

|S| is number of elements in **S**

Given, **S** is a set of 14 examples, in which one of the attributes is wind speed. The values of Wind can be Weak or Strong. The classification of these 14 examples are 9 YES and 5 NO. Given, for attribute **wind** :

8 occurrences of **wind = weak** and

6 occurrences of **wind = strong**.

For **wind = weak**, **6** of the examples are **YES** and **2** are **NO**.

For **wind = strong**, **3** of the examples are **YES** and **3** are **NO**.

$$\text{Gain}(S, \text{wind}) = \text{Entropy}(S) - (8/14) \bullet \text{Entropy}(S_{\text{weak}})$$

$$- (6/14) \bullet \text{Entropy}(S_{\text{strong}})$$

$$= 0.940 - (8/14) \bullet 0.811 - (6/14) \bullet 1.00 = 0.048$$

$$\text{Entropy}(S_{\text{weak}}) = - (6/8) \bullet \log_2(6/8) - (2/8) \bullet \log_2(2/8) = 0.811$$

$$\text{Entropy}(S_{\text{strong}}) = - (3/6) \bullet \log_2(3/6) - (3/6) \bullet \log_2(3/6) = 1.00$$

For each attribute, the gain is calculated and the highest gain is used in the decision node.

Step-by-Step Calculations : Ref. Training data stated before

◇ Step 01 : "example" set S

The set S of 14 examples is with 9 'yes' and 5 'no' then

$$\text{Entropy}(S) = - (9/14) \log_2 (9/14) - (5/14) \log_2 (5/14) = 0.940$$

◇ Step 02 : Attribute Outlook

Outlook value can be sunny, cloudy or rainy.

Outlook = sunny is of occurrences 5

Outlook = cloudy is of occurrences 4

Outlook = rainy is of occurrences 5

Outlook = sunny, 2 of the examples are 'yes' and 3 are 'no'

Outlook = cloudy, 4 of the examples are 'yes' and 0 are 'no'

Outlook = rainy, 3 of the examples are 'yes' and 2 are 'no'

$$\text{Entropy}(S_{\text{sunny}}) = - (2/5) \times \log_2(2/5) - (3/5) \times \log_2(3/5) = 0.970950$$

$$\text{Entropy}(S_{\text{cloudy}}) = - (4/4) \times \log_2(4/4) - (0/4) \times \log_2(0/4) = 0$$

$$\text{Entropy}(S_{\text{rainy}}) = - (3/5) \times \log_2(3/5) - (2/5) \times \log_2(2/5) = 0.970950$$

$$\begin{aligned} \text{Gain}(S, \text{Outlook}) &= \text{Entropy}(S) - (5/14) \times \text{Entropy}(\text{Sunny}) \\ &\quad - (4/14) \times \text{Entropy}(\text{ScLOUD}) \\ &\quad - (5/14) \times \text{Entropy}(\text{Srainy}) \\ &= 0.940 - (5/14) \times 0.97095059 - (4/14) \times 0 \\ &\quad - (5/14) \times 0.97095059 \\ &= 0.940 - 0.34676 - 0 - 0.34676 \\ &= 0.246 \end{aligned}$$

◇ **Step 03 : Attribute Temperature**

Temp value can be hot, medium or cold.

Temp = hot is of occurrences 4

Temp = medium is of occurrences 6

Temp = cold is of occurrences 4

Temp = hot, 2 of the examples are 'yes' and 2 are 'no'

Temp = medium, 4 of the examples are 'yes' and 2 are 'no'

Temp = cold, 3 of the examples are 'yes' and 1 are 'no'

$$\text{Entropy(Shot)} = - (2/4) \times \log_2(2/4) - (2/4) \times \log_2(2/4) = -0.99999999$$

$$\text{Entropy(Smedium)} = - (4/6) \times \log_2(4/6) - (2/6) \times \log_2(2/6) = -0.91829583$$

$$\text{Entropy(Scold)} = - (3/4) \times \log_2(3/4) - (1/4) \times \log_2(1/4) = -0.81127812$$

$$\begin{aligned} \text{Gain}(S, \text{Temp}) &= \text{Entropy}(S) - (4/14) \times \text{Entropy}(\text{Shot}) \\ &\quad - (6/14) \times \text{Entropy}(\text{Smedium}) \\ &\quad - (4/14) \times \text{Entropy}(\text{Scold}) \\ &= 0.940 - (4/14) \times 0.99999 - (6/14) \times 0.91829583 \\ &\quad - (4/14) \times 0.81127812 \\ &= 0.940 - 0.2857142 - 0.393555 - 0.2317937 \\ &= 0.0289366072 \end{aligned}$$

◇ **Step 04 : Attribute Humidity**

Humidity value can be high, normal.

Humidity = high is of occurrences 7

Humidity = normal is of occurrences 7

Humidity = high, 3 of the examples are 'yes' and 4 are 'no'

Humidity = normal, 6 of the examples are 'yes' and 1 are 'no'

$$\text{Entropy}(\text{Shigh}) = - (3/7) \times \log_2(3/7) - (4/7) \times \log_2(4/7) = -0.9852281$$

$$\text{Entropy}(\text{Snormal}) = - (6/7) \times \log_2(6/7) - (1/7) \times \log_2(1/7) = -0.5916727$$

$$\begin{aligned} \text{Gain}(S, \text{Humidity}) &= \text{Entropy}(S) - (7/14) \times \text{Entropy}(\text{Shigh}) \\ &\quad - (7/14) \times \text{Entropy}(\text{Snormal}) \\ &= 0.940 - (7/14) \times 0.9852281 - (7/14) \times 0.5916727 \\ &= 0.940 - 0.49261405 - 0.29583635 \\ &= 0.1515496 \end{aligned}$$

◇ **Step 05 : Attribute wind**

Wind value can be weak or strong.

Wind = weak is of occurrences 8

Wind = strong is of occurrences 6

Wind = weak, 6 of the examples are 'yes' and 2 are 'no'

Wind = strong, 3 of the examples are 'yes' and 3 are 'no'

$$\text{Entropy}(S_{\text{weak}}) = - (6/8) \times \log_2(6/8) - (2/8) \times \log_2(2/8) = 0.811$$

$$\text{Entropy}(S_{\text{strong}}) = - (3/6) \times \log_2(3/6) - (3/6) \times \log_2(3/6) = 1.00$$

$$\begin{aligned} \text{Gain}(S, \text{Wind}) &= \text{Entropy}(S) - (8/14) \times \text{Entropy}(S_{\text{weak}}) - \\ &\quad (6/14) \times \text{Entropy}(S_{\text{strong}}) \\ &= 0.940 - (8/14) \times 0.811 - (6/14) \times 1.00 \\ &= 0.048 \end{aligned}$$

AI –Learning from example : Induction

◇ **Step 06 : Summary of results are**

- Entropy(S) = 0.940
- Gain(S, Outlook) = **0.246**
- Gain(S, Temp) = 0.0289366072
- Gain(S, Humidity) = 0.1515496
- Gain(S, Wind) = 0.048

◇ **Step 07 : Find which attribute is the root node.**

Gain(S, Outlook) = 0.246 is highest.

Therefore "Outlook" attribute is the decision attribute in the root node.

"Outlook" as root node has three possible values - **sunny, cloudy, rain.**

S.No	Day	Outlook	Temp	Humidity	Wind	Play
1	11	sunny	75 Medium	70 Normal	strong	yes
2	2	sunny	80 Hot	90 High	strong	no
3	1	sunny	85 Hot	85 High	week	no
4	8	sunny	72 Medium	95 High	week	no
5	9	sunny	69 Cold	70 Normal	week	yes
6	12	cloudy	72 Medium	90 High	strong	yes
7	3	cloudy	83 Hot	78 High	week	yes
8	7	cloudy	64 Cold	65 Normal	strong	yes
9	13	cloudy	81 Hot	75 Normal	week	yes
10	14	rainy	71 Medium	85 High	strong	no
11	6	rainy	65 Cold	70 Normal	strong	no
12	10	rainy	75 Medium	80 Normal	week	yes
13	5	rainy	68 Cold	80 Normal	week	yes
14	4	rainy	70 Medium	96 High	week	yes

AI –Learning from example : Induction

◇ **Step 08 : Find which attribute is next decision node.**

Outlook has three possible values,

so root node has three branches (sunny, cloudy, rain).

Ssunny = {D1, D2, D8, D9, D11} = 5 "examples". "D" represent "Days".

With outlook = sunny

Gain(Ssunny, Humidity) = **0.970**

Gain(Ssunny, Temp) = 0.570

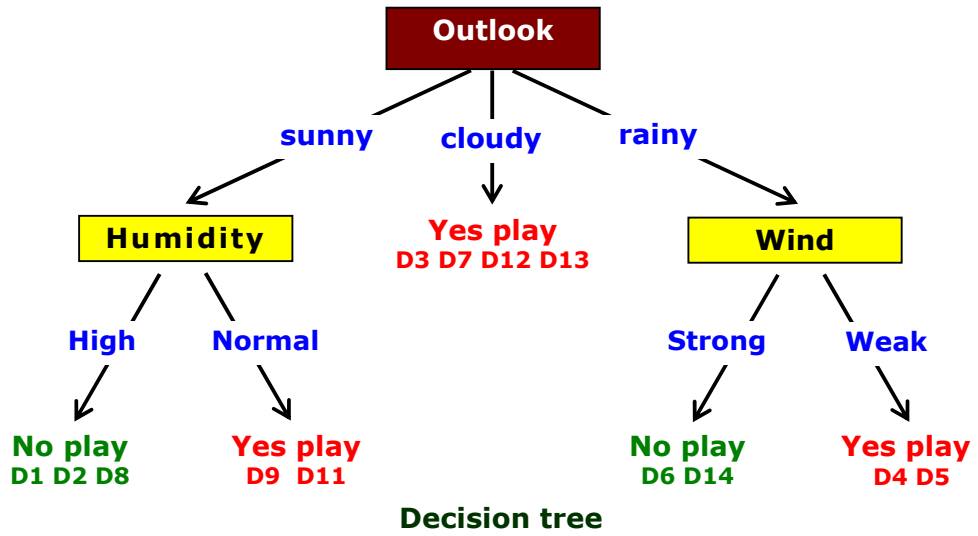
Gain(Ssunny, Wind) = 0.019

Humidity has the highest gain 0.970 is next decision node.

◇ **Step 09 : This process goes on until all days data are classified perfectly or run out of attributes.**

AI - Learning from example : Induction

- Thus, the classification tree built using ID3 algorithm is shown below. It tells if the weather was amenable to play ?



■ **Applications**

- ‡ ID3 is easy to use.
- ‡ Its primary use is replacing the expert who would normally build a classification tree by hand.
- ‡ ID3 has been incorporated in a number of commercial rule-induction packages.
- ‡ Some specific applications include
 - ◇ medical diagnosis,
 - ◇ credit risk assessment of loan applications,
 - ◇ equipment malfunctions by their cause,
 - ◇ classification of plant diseases, and
 - ◇ web search classification.

4. Explanation Based Learning (EBL)

Humans appear to learn quite a lot from one example.

Human learning is accomplished by examining particular situations and relating them to the background knowledge in the form of known general principles.

This kind of learning is called "Explanation Based Learning (EBL)".

4.1 General Approach

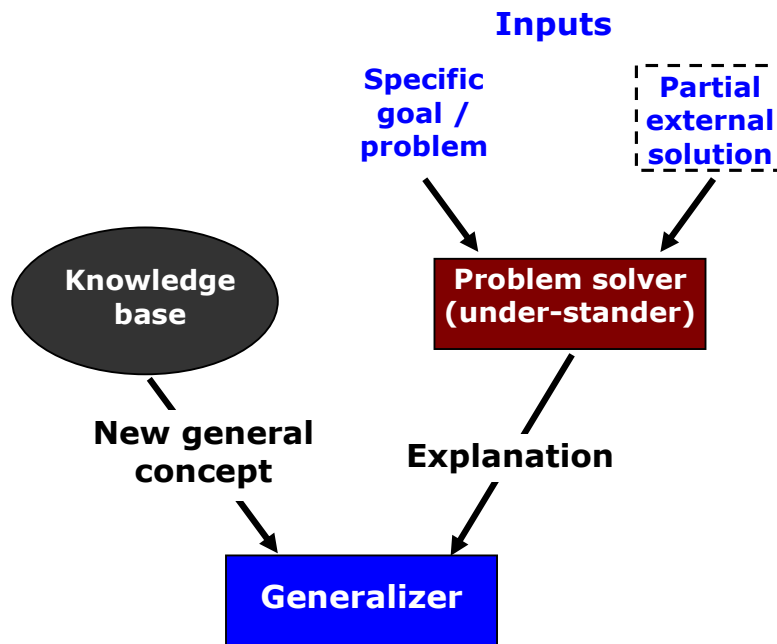
EBL is abstracting a general concept from a particular training example.

EBL is a technique to formulate general concepts on the basis of a specific training example. EBL analyses the specific training example in terms of domain knowledge and the goal concept. The result of EBL is an explanation structure, that explains why the training example is an instance of the goal concept. The explanation-structure is then used as the basis for formulating the general concept.

Thus, EBL provides a way of generalizing a machine-generated explanation of a situation into rules that apply not only to the current situation but to similar ones as well.

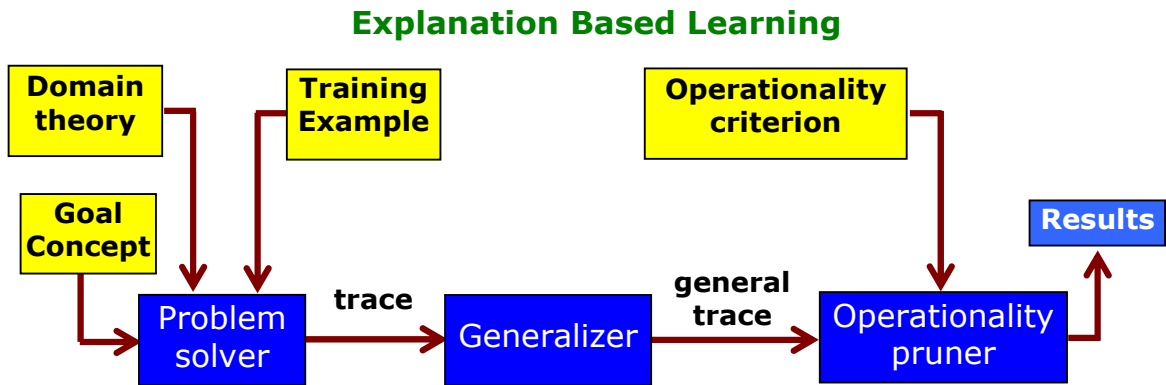
4.2 EBL Architecture

The overall architecture of the EBL learning method.



EBL System Schematic

The schematic below shows explanation based learning.



- **Input to EBL :**

The blocks color yellow are external to EBL.

They are the 4 different kinds of input that EBL algorithm accepts.

1. **Training example** - Situation description, facts .

An example is a set of facts describing an instance of the goal concept.

2. **Goal concept** - Some predicate calculus statements .

A high level description of concept that the program is supposed to learn; e.g., the definition of an object "**Cup**" in function-structure involves functional properties (**lift-able, stable, open-vessel . . .**) rather than structural features (light, part of, pointing . . .).

3. **Domain theory** - Inference rules .

It represents facts and rules that constitute what the learner knows. The facts describe an instance of the goal concept and the rules describe relationships between objects and actions in a domain; e.g., the cup domain includes **facts: concavities, bases, and lugs**, as well as **rules: about lift ability, stability and what makes an open vessel.**

4. **Operational criterion** - description of concepts .

A description in an appropriate form of the final concept.

A predicate from domain theory over concept definition, specifying the form in which the learned concept definition must be expressed.

■ **Description of EBL algorithm**

Given the four inputs just described, the task is to construct an explanation in terms of the domain theory and then determine a set of sufficient conditions under which the explanation holds.

Thus, EBL algorithm consisting of two stages:

1. **Explain:** Construct an explanation in terms of the domain theory that shows how the training example satisfies the goal concept definition. This explanation must be constructed so that each branch of the explanation structure terminates in an expression that satisfies the operability criterion.
2. **Generalize:** Determine a set of sufficient conditions under which the explanation holds, stated in terms that satisfy the operability criterion. This is accomplished by regressing (back propagating) the goal concept through the explanation structure. The conjunction of the resulting expressions constitutes the desired concept definition.

Algorithm : The step involved are

1. Given an example, an "explanation" is first constructed by applying a problem solver to relate the example to some general domain theory. The result of this operation is a trace or "proof" of the example with respect to the theory.
2. Next, generalizes the explanation using the method of goal regression. This involves traversing the tree from top to bottom, replacing constants by variables, but just those constants that are not embedded in the rules or facts used to create the proof. The result of this operation is the generalized proof tree.
3. The tree is then pruned by removing irrelevant leaf nodes until no operational predicates appear at internal nodes. The result of this operation is that explanation structure satisfies the operability criterion that applies to other examples too.
4. Finally, the operational rules are extracted from the general explanation. The definition of the target concept is conjunction of the remaining leaf nodes.

4 Example : "CUP" Generalization problem

The EBL methodology described before is applied to concept "CUP".

The training example, domain theory, goal concept, operability criterion, and a proof tree generalization using a goal regression technique are presented. The logical operators used are

^ and; → implies; if .. then; ↔ if and only if; iff

Goal Concept

lift-able(x) ^ stable(x) ^ open-vessel(x) ↔ cup(x)

Training example

colour(Obj23, Blue) ^ has-part(Obj23, Handle16) ^ has-part(Obj23, Bottom19) ^
 owner(Obj23, Ralph) ^ has-part(Obj23, Concavity12) ^ is(Obj23, Light) ^
 is(Ralph, Male) ^ isa(Handle16, Handle) ^ isa(Bottom19, Bottom) ^
 is(Bottom19, Flat) ^ isa(Concavity12, Concavity) ^
 is(Concavity12, Upward-Pointing)

Domain theory

has-part(x,y) ^ isa(y,Concavity) ^ is(y, Upward-Pointing) → open-vessel(x)
 is(x, Light) ^ has-part(x,y) ^ isa(y,Handle) → liftable(x)
 has-part(x,y) ^ isa(y, Bottom) ^ is(y,Flat) → stable(x)

Operability Criterion

The concept definition is expressed in terms of structural features.

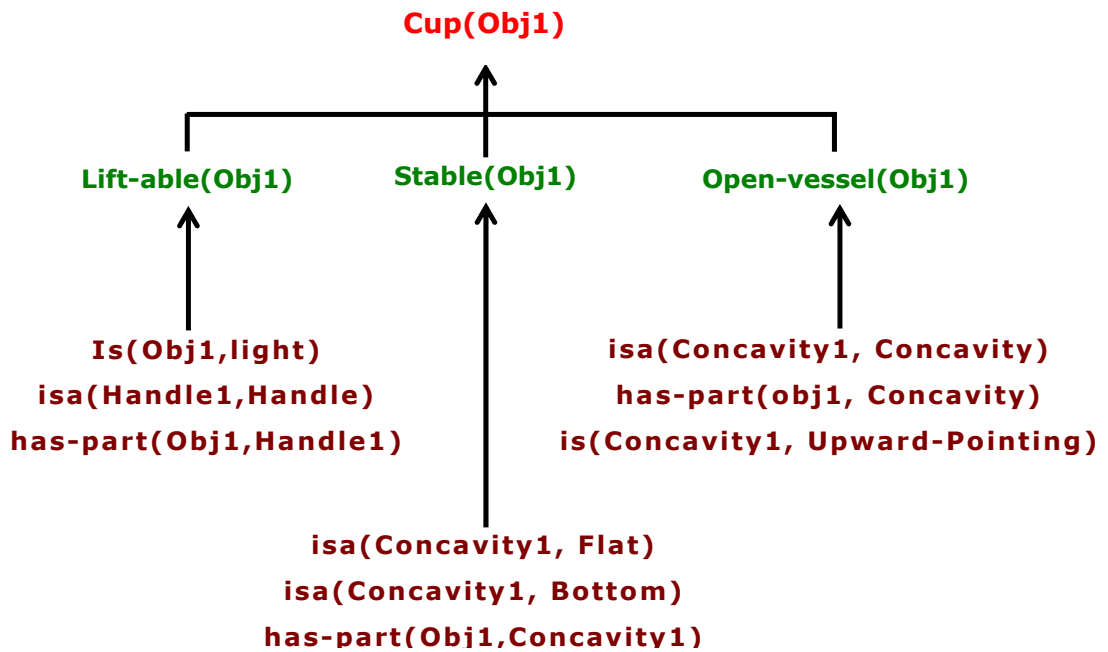
The first stage of the EBL process is to show why the training example is an example of a cup. This represents a proof.

This proof is expressed only in terms of the operability criterion and irrelevant details discarded relating to the Owner and Color.

Explanation structure of the cup.

Explanation Structure of the cup

The structure below represents the first stage of the EBL process. It proves why the training example is an example of a cup.



Proof tree generalization using a goal regression technique.

The above proof is now generalized by using a goal regression technique. In this example replacing the constants with variables gives the required generalization of the cup.

has-part(x, y) U isa(y, Concavity) U is(y, Upward-Pointing) U
 has-part(x, z) U isa(z, Bottom) U is(z, Flat) U has-part(x,w) U
 isa(w, Handle) U is(x, Light)

5. Discovery

Simon (1966) first proposed the idea that we might explain scientific discovery in computational terms and automate the processes involved on a computer. Project **DENDRAL** (Feigenbaum 1971) demonstrated this by **inferring structures of organic molecules** from mass spectra, a problem previously solved only by experienced chemists.

Later, a knowledge based program called **AM** the Automated Mathematician (Lenat 1977) **discovered many mathematical concepts**.

After this, an equation discovery systems called **BACON** (Langley, 1981) **discovered a wide variety of empirical laws** such as the ideal gas law. The research continued during the 1980s and 1990s but reduced because the computational biology, bioinformatics and scientific data mining have convinced many researchers to focus on domain-specific methods. But need for research on general principles for scientific reasoning and discovery very much exists.

Discovery system AM relied strongly on **theory-driven** methods of discovery. BACON employed **data-driven** heuristics to direct its search for empirical laws.

These two discovery programs are illustrated in the next few slides.

5.1 Theory Driven Discovery

The Simon's theory driven science, means AI-modeling for theory building. It starts with an existing theory represented in some or all aspects in form of a symbolic model and one tries to transform the theory to a runnable program. One important reason for modeling a theory is scientific discovery in the theory driven approach, this means the discovery of new theoretical conclusions, gaps, or inconsistencies.

Many computational systems have been developed for modeling different types of discoveries. The **Logic Theorist** (1956) was designed to prove theorems in logic when AI did not exist. Among the more recent systems, the Automated Mathematician **AM** (Lenat, 1979) is a good example in modeling mathematical discovery.

- **AM** (Automated Mathematician)

AM is a heuristic driven program that discovers concepts in elementary mathematics and set theory. AM has 2 inputs:

- (a) description of some concepts of set theory: e.g. union, intersection;
- (b) information on how to perform mathematics. e.g. functions.

AM have successively rediscovered concepts such as :

- (a) Integers , Natural numbers, Prime Numbers;
- (b) Addition, Multiplication, Factorization theorem ;
- (c) Maximally divisible numbers, e.g. 12 has six divisors 1, 2, 3, 4, 6, 12.

[AM is described in the next slide.]

● **How does AM work ?**

AM employs many general-purpose AI techniques.

The system has around **115 basic elements** such as sets, lists, elementary relations. The mathematical concepts are represented as frames. Around **250 heuristic rules** are attached to slots in the concepts. The rules present hints as how to employ functions, create new concepts, generalization etc. about activities that might lead to interesting discoveries.

The system operates from an agenda of tasks. It selects the most interesting task as determined by a set of over **50 heuristics**. It then performs all heuristics it can find which should help in executing it. The heuristics represented as operators are used to generalize, to specialize or to combine the elementary concepts or relations to make more complex ones. Heuristics can fill in concept slots, check the content of the slots, create new concepts, modify task agenda, interestingness levels, etc. Because it selects the most interesting task to perform at all times, AM is performing the **best-first search** in a space of mathematical concepts. However, its numerous heuristics (over 200) guide its search very effectively, limiting the number of concepts it creates and improving their mathematical quality.

5.2 Data Driven Discovery

Data driven science, in contrast to theory driven, starts with empirical data or the input-output behavior of the real system without an explicitly given theory. The modeler tries to write a computer program which generates the empirical data or input-output behavior of the system. Typically, models are produced in a **generate-and-test-procedure**. Generate-and-test means writing program code which tries to model the i-o-behavior of the real system first approximately and then improve as long as the i-o-behavior does not correspond to the real system. A family of such discovery models are known as **BACON programs**.

- **BACON System**

Equation discovery is the area of machine learning that develops methods for automated discovery of quantitative laws, expressed in the form of equations, in collections of measured data.

BACON is pioneer among equation discovery systems. BACON is a family of algorithms for discovering scientific laws from data.

BACON.1 discovers simple numeric laws.

BACON.3 is a knowledge based system, has discovered simple empirical laws like physicists and shown its generality by rediscovering the **Ideal gas law, Kepler's third law, Ohm's law** and more.

The next few slides shows how BACON1 rediscovers Kepler's third Law and BACON3 rediscovers Ideal Gas Law.

BACON.1 : Discovers simple numeric laws.

Given a set of observed values about two variables **X** and **Y**, BACON.1 finds a function **Y = f(X)** using four heuristics:

- Heuristic 1 : If **Y** has value **V** in several observed cases, make the hypothesis that **Y = V** in all cases.
- Heuristic 2 : If **X** and **Y** are linearly related with slope **S** and intercept **I** in several observed cases, then make the hypothesis that **Y = S · X + I** is in all cases.
- Heuristic 3 : If **X** increases as **Y** decreases, and **X** and **Y** are not linearly related define a new term **T** as the product of **X** and **Y** ie., **T = X · Y**
- Heuristic 4 : If **X** increases as **Y** increases, and **X** and **Y** are not linearly related, define a new term **T** as the division of **X** by **Y** ie., **T = X / Y**

Note : BACON1 iteratively applies these 4 heuristics until a scientific law is discovered. Heuristics 1 and 2 detect linear relationships. Heuristics 3 and 4 detect simple non-linear relationships. Heuristics 1 and 2 produce scientific laws. Heuristics 3 and 4 are intermediate steps.

Example : Rediscovering Kepler’s third Law

Kepler’s third Law is stated below. Assume the law is not discovered or known.

"The square of the orbital period **T** is proportional to the cube of the mean distance **a** from the Sun." ie., **T² = k a³**, **k** is constant number, is same for all planets. If we measure **T** in years and all distances in "astronomical units AUs" with 1 AU the mean distance between the Earth and the Sun, then if **a = 1 AU**, **T** is one year, and **k** with these units just equals 1, i.e. **T² = a³**.

- **Input** : Planets, Distance from Sun (**D**), orbit time Period (**P**)

Planet	D	P
Mercury	0.382	0.241
Venus	0.724	0.616
Earth	1.0	1.0
Mars	1.524	1.881
Jupiter	5.199	11.855
Saturn	9.539	29.459

[continued in next slide]

■ **Apply heuristics 1 to 4 :** (Iteration 1)

Try heuristic 1:	not applicable,	neither D nor P is constant
Try heuristic 2:	not applicable,	no linear relationship
Try heuristic 3:	not applicable,	D increasing P not decreasing
Try heuristic 4:	applicable,	D increases as P increase, so add new variable D/P to the data set.

Adding new variable **D/P** to the data set:

Planet	D	P	D/P
Mercury	0.382	0.241	1.607
Venus	0.724	0.616	1.175
Earth	1.0	1.0	1.0
Mars	1.524	1.881	0.810
Jupiter	5.199	11.855	0.439
Saturn	9.539	29.459	0.324

■ **Apply heuristics 1 to 4 :** (Iteration 2)

Try heuristic 1:	not applicable,	D/P is not constant
Try heuristic 2:	not applicable,	no linear relationship between D/P and D or P
Try heuristic 3:	applicable,	D/P decreases as D increases and as P increases, so the system could add two new variables: $D \times (D/P) = D^2/P$ or $P \times (D/P) = D$ but D already exists, so add new variable D²/P

Adding new variable **D²/P** to the data set:

Planet	D	P	D/P	D ² /P
Mercury	0.382	0.241	1.607	0.622
Venus	0.724	0.616	1.175	0.851
Earth	1.0	1.0	1.0	1.0
Mars	1.524	1.881	0.810	1.234
Jupiter	5.199	11.855	0.439	2.280
Saturn	9.539	29.459	0.324	3.088

■ **Apply heuristics 1 to 4 :** (Iteration 3)

Try heuristic 1: not applicable, D^2/P is not constant
 Try heuristic 2: not applicable, D^2/P is not linearly related with any other variable
 Try heuristic 3: **applicable**, D^2/P decreases as D/P increases, so add the new variable: $(D^2/P) \times (D/P) = D^3/P^2$

Adding new variable D^3/P^2 to the data set:

Planet	D	P	D/P	D^2/P	D^3/P^2
Mercury	0.382	0.241	1.607	0.622	1.0
Venus	0.724	0.616	1.175	0.851	1.0
Earth	1.0	1.0	1.0	1.0	1.0
Mars	1.524	1.881	0.810	1.234	1.0
Jupiter	5.199	11.855	0.439	2.280	1.0
Saturn	9.539	29.459	0.324	3.088	1.0

■ **Apply heuristics 1 to 4 :** (Iteration 4)

Try heuristic 1: **applicable**, D^3/P^2 is constant

■ **Conclusion :** D^3/P^2

This is Kepler's third law. (took about 20 years to discover it !)

■ **A limitation of BACON.1**

It works only for target equation relating at most two variable.

BACON.3 :

BACON.3 is a knowledge based system production system that discovers empirical laws. The main heuristics detect constancies and trends in data, and lead to the formulation of hypotheses and the definition of theoretical terms. The program represents information at varying levels of description. The lowest levels correspond to direct observations, while the highest correspond to hypotheses that explain everything so far observed. BACON.3 is built on top of BACON.1.

- It starts with a set of variables for a problem. For example, to derive the ideal gas law, it started with four variables, **p, V, n, T**.
 - ‡ p - gas pressure,
 - ‡ V - gas volume,
 - ‡ T - gas temperature,
 - ‡ n - is the number of moles.
- Values from experimental data are inputted.
- BACON holds some constant and try to notice trends in the data.
- Finally draws inferences. Recall **$pV/nT = k$** where **k** is a constant.
- BACON has also been applied to Kepler's 3rd law, Ohm's law, conservation of momentum and Joule's law.

Example :

Rediscovering the ideal gas law $pV/nT = 8.32$, where p is the pressure on a gas, n is the number of moles, T is the temperature and V the volume of the gas. [The step-by-step complete algorithm is not given like previous example, but the procedure is explained below]

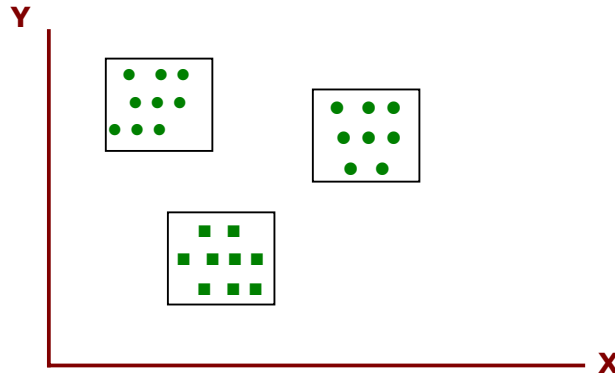
- At the first level of description we hold $n = 1$ and $T = 300$ and vary p and V . Choose V to be the dependent variable.
- At this level, BACON discovers the law $pV = 2496.0$.
- Now the program examines this phenomenon :
when $n = 1$ and $T = 310$ then $pV = 2579.2$. Similarly,
when $n = 1$ and $T = 320$ then $pV = 2662.4$.
- At this point, BACON has enough information to relate the values of pV and the temperature T . These terms are linearly related with an intercept of 0 , making the ratio pV/T equal to 8.32 .
- Now the discovery system can vary its third independent term.
while $n = 2$, the pV/T is found to be 16.64 ,
while $n = 3$, the pV/T is found to be 24.96 .
- When it compares the values of n and pV/T , BACON finds another linear relation with a zero intercept. The resulting equation, $pV/nT = 8.32$, is equivalent to the ideal gas law.

6. Clustering

Clustering is a way to form **natural groupings** or clusters of patterns.

Clustering is often called an **unsupervised learning**.

Example : Three natural groups of data points, i.e., 3 natural clusters.



- Clustering is one of the most utilized data mining techniques.
- The data have no target attribute.
The data set is explored to find some intrinsic structures in them.
- Unlike Classification where the task is to learn to assign instances to predefined classes, in Clustering no predefined classification is required.
The task is to learn a classification from the data.

1 Distance Functions

- **Euclidean geometry** is study of relationships between angles and distances in space. Mathematical spaces may be extended to any dimension, called an n-dimensional Euclidean space or an n-space.

Let \mathbf{R} denote the field of real numbers. The space of all **n-tuples** of real numbers forms an n-dimensional vector space over \mathbf{R} , denoted by \mathbf{R}^n .

An element of \mathbf{R}^n is written as $\mathbf{X} = (x_1, x_2, \dots, x_i, \dots, x_n)$, where x_i is a real number. Similarly the other element $\mathbf{Y} = (y_1, y_2, \dots, y_i, \dots, y_n)$.

The vector space operations on \mathbf{R}^n are defined by

$$\mathbf{X} + \mathbf{Y} = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n) \quad \text{and}$$

$$a\mathbf{X} = (ax_1, ax_2, \dots, ax_n)$$

The standard inner product (ie dot product) on \mathbf{R}^n , given by

$$\mathbf{X} \bullet \mathbf{Y} = \sum_{i=1}^n (x_i y_i + x_2 y_2 + \dots + x_n y_n) \quad \text{is a real number.}$$

This product defines a **distance function** (or metric) on \mathbf{R}^n by

$$d(\mathbf{X}, \mathbf{Y}) = \|\mathbf{X} - \mathbf{Y}\| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

The (interior) angle θ between \mathbf{x} and \mathbf{y} is then given by

$$\theta = \cos^{-1} \left(\frac{\mathbf{x} \bullet \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \right)$$

The inner product of X with itself is always nonnegative.

$$\|\mathbf{X}\|^2 = \sum_{i=1}^n (x_i - y_i)^2$$

- Euclidean distance** or Euclidean metric is the "ordinary" distance between two points that one would measure with a ruler. The Euclidean distance between two points $P = (p_1, p_2, \dots, p_n)$ and $Q = (q_1, q_2, \dots, q_n)$ in Euclidean n -space, is defined as :

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

Example : 3-dimensional distance

For two 3D points, $P = (p_x, p_y, \dots, p_z)$ and $Q = (q_x, q_y, \dots, q_z)$

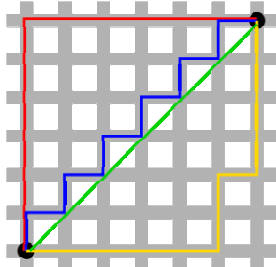
The Euclidean 3-space, is computed as :

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$$

- Manhattan distance** also known as **city block distance** between two points in an Euclidean space with fixed cartesian coordinate system is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes.

Example : In a plane, the Manhattan distance between the point P_1 with coordinates (x_1, y_1) and the point P_2 with coordinates (x_2, y_2) is

$$|x_1 - x_2| + |y_1 - y_2|$$



Manhattan versus Euclidean distance:

The red, blue, and yellow lines represent Manhattan distance. They all are of same length as **12**.

The green line represent Euclidian distance of length **$6 \times \sqrt{2} \approx 8.48$** .

■ Minkowski Metric

Let X_i and X_j are data points (vectors) then Minkowski distance between these two data points is :

$$\text{dist}(X_i, X_j) = ((x_{i1} - x_{j1})^h + (x_{i2} - x_{j2})^h + \dots + (x_{ir} - x_{jr})^h)^{1/h}$$

where h is a positive integer.

‡ Euclidean distance IF $h = 2$

$$\begin{aligned} \text{dist}(X_i, X_j) &= ((x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ir} - x_{jr})^2)^{1/2} \\ &= \sqrt{ (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ir} - x_{jr})^2 } \end{aligned}$$

Weighted Euclidean distance

$$\text{dist}(X_i, X_j) = \sqrt{ (w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \dots + w_r(x_{ir} - x_{jr})^2) }$$

‡ Manhattan distance IF $h = 1$

$$\text{dist}(X_i, X_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ir} - x_{jr}|$$

‡ **Squared Euclidean distance:** to place progressively greater weight on data points that are further apart.

$$\text{dist}(X_i, X_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ir} - x_{jr})^2$$

‡ **Chebychev distance:** to define two data points as "different" if they are different on any one of the attributes.

$$\text{dist}(X_i, X_j) = \max (|x_{i1} - x_{j1}| , |x_{i2} - x_{j2}| , \dots , |x_{ir} - x_{jr}|)$$

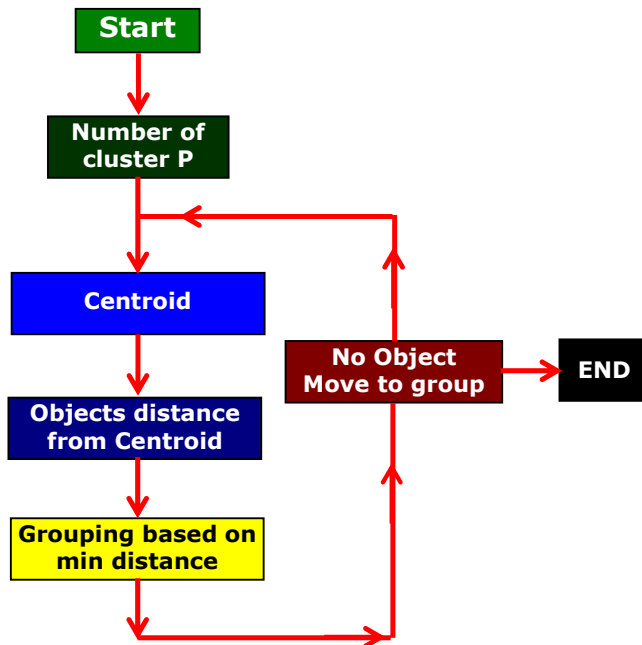
K-Mean Clustering

k-means clustering is an algorithm to classify or to group objects based on attributes/features into **K** number of group.

K is positive integer number.

The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid.

K-Mean Clustering algorithm



The algorithm consists of 3 steps. First take any random object as initial centroid, then the three steps mentioned below are iterated until converge.

Step-1 Determine centroid coordinate

Step-2 Determine distance of each object to the centroids

Step-3 Group the object based on minimum distance

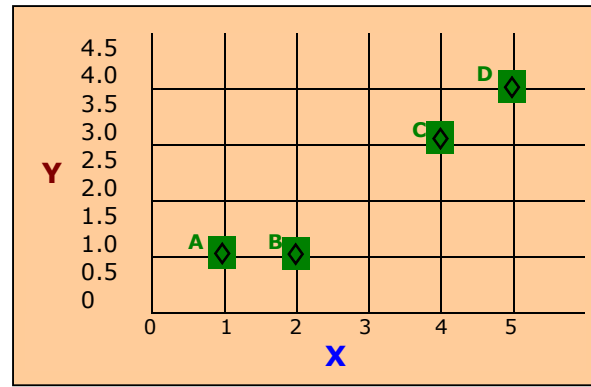
The next slide explains step by step the complete algorithm

Example : K-Mean Clustering

Objects : 4 medicines as A, B, C, D.

Attributes : 2 as X is weight & Y is PH

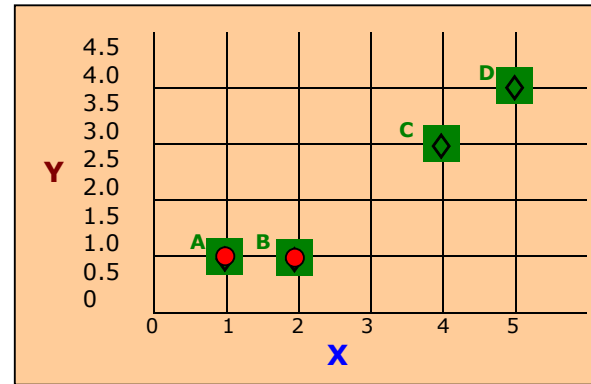
Objects	Attributes	
	X	Y
A	1	1
B	2	1
C	4	3
D	5	4



Initial value of centroids:

Suppose medicine A and medicine B be first centroids. If P_1 and P_2 denote coordinates of the centroids, then $P_1 = (1, 1)$ and $P_2 = (2, 1)$.

Let centroid P_1 be for cluster group-1
 Let centroid P_2 be for cluster group-2.



1. Iteraion 0

(a) **Objects Clusters centers stated before :** **Objects** : **A , B, C, D**

Group-1 has center **P₁ = (1, 1)** ;

Group-2 has center **P₂ = (2, 1)** ;

Attributes : **X** and **Y**

	A	B	C	D
X	1	2	4	5
Y	1	1	3	4

(b) **Calculate distances between cluster center to each object**

- 1st, calculate the Euclidean distances from cetroid **P₁** to each point **A, B, C, D**. It is the 1st row of the distance matrix.
- 2nd, calculate the Euclidean distances from cetroid **P₂** to each point **A, B, C, D**. It is the 2nd row of the distance matrix.

The ways to calculate just two distance matrix elements **D₁₃** and **D₂₃** are :

$$D_{13} = \sqrt{(C_x - P_{1x})^2 + (C_y - P_{1y})^2} = \sqrt{(4 - 1)^2 + (3 - 1)^2} = 3.61$$

$$D_{23} = \sqrt{(C_x - P_{2x})^2 + (C_y - P_{2y})^2} = \sqrt{(4 - 2)^2 + (3 - 1)^2} = 2.83$$

Similarly calculate other elements **D₁₁ , D₁₂ , D₁₄ , D₂₁ , D₂₂ , D₂₄**

(c) **Distance matrix becomes**

$$D^0 = \left\{ \begin{array}{cccc} 0 & 1 & 3.61 & 5 \\ 1 & 0 & 2.83 & 4 \\ \text{A} & \text{B} & \text{C} & \text{D} \end{array} \right\} \begin{array}{l} \text{1st row indicates group-1 cluster} \\ \text{2nd row indicates group-2 cluster} \end{array}$$

(d) **Objects clustering into groups:**

Assign group to each object based on the minimum distance. Thus,

medicine **A** is assigned to **group 1**;

medicine **B** is assigned to **group 2**,

medicine **C** is assigned to **group 2** , and

medicine **D** is assigned to **group 2**.

Group Matrix : matrix element is **1** if the object is assigned to that group

$$G^0 = \left\{ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ \text{A} & \text{B} & \text{C} & \text{D} \end{array} \right\} \begin{array}{l} \text{1st row as group 1} \\ \text{2nd row as group 2} \end{array}$$

2. Iteration 1 :

The cluster groups have new members. Compute the new centroids of each group. **Repeat the process of iteration indicated below.**

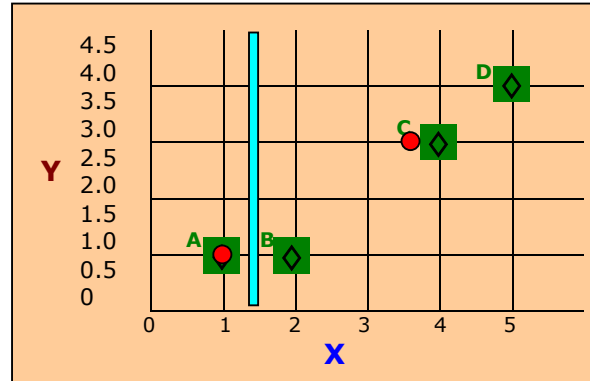
Group-1 has one member **A**, the centroid remains as $P_1 = (1, 1)$.

Group-2 now has 3 members **B, C, D**, so centroid is the average of their coordinates:

$$P_2 = \left(\frac{2+4+5}{3}, \frac{1+3+5}{3} \right)$$

$$= (11/3, 9/3)$$

$$= (3.66, 3)$$



(a) **Objects Clusters centers stated above :** **Objects** : **A, B, C, D**

Group-1 has center $P_1 = (1, 1)$;

Attributes : **X** and **Y**

Group-2 has center $P_2 = (3.66, 3)$;

	A	B	C	D
X	1	2	4	5
Y	1	1	3	4

(b) **Calculate distances between cluster center to each object**

[The method is same as in Iteration 0 (b)]

- 1st, calculate the Euclidean distances from centroid P_1 to each point **A, B, C, D**. You get $D_{11}, D_{12}, D_{13}, D_{14}$ as the 1st row of the distance matrix.
- 2nd, calculate the Euclidean distances from centroid P_2 to each point **A, B, C, D**. You get $D_{21}, D_{22}, D_{23}, D_{24}$ as the 2nd row of the distance matrix.

(c) **Distance matrix becomes**

$$D^1 = \begin{matrix} & \mathbf{0} & \mathbf{1} & \mathbf{3.61} & \mathbf{5} \\ \mathbf{0} & & & & \\ \mathbf{3.14} & \mathbf{2.36} & \mathbf{0.47} & \mathbf{1.89} & \end{matrix}$$

1st row indicates **group-1** cluster
2nd row indicates **group-2** cluster

(d) **Objects clustering into groups:**

Assign group to each object based on the minimum distance.

medicine **A** no change, remains in **group 1**;

medicine **B** moves to **group 1**,

medicine **C** no change, remains in **group 2**,

medicine **D** no change, remains in **group 2**.

Group Matrix : matrix element is **1** if the object is assigned to that group

$$G^1 = \left\{ \begin{matrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{matrix} \right\}$$

1st row indicates **group-1** cluster
2nd row indicates **group-2** cluster

A **B** **C** **D**

3. Iteration 2 :

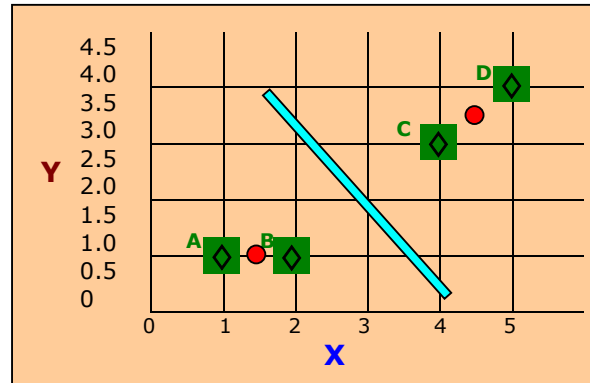
The cluster groups have new members. Compute the new centroids of each group. **Repeat the process of iteration indicated below.**

Group-1 has 2 members, so the new centroid is the average coordinate among A and B :

$$P_1 = \left(\frac{1+2}{2}, \frac{1+1}{2} \right) = (1.5, 1)$$

Group-2 has 2 members, so the new centroid is the average coordinate among C and D :

$$P_2 = \left(\frac{4+5}{2}, \frac{3+4}{2} \right) = (4.5, 3.5)$$



(a) **Objects Clusters centers stated above :** **Objects** : A , B , C , D

Group-1 has center $P_1 = (1.5, 1)$;

Attributes : X and Y

Group-2 has center $P_2 = (4.5, 3.5)$;

	A	B	C	D
X	1	2	4	5
Y	1	1	3	4

(b) **Calculate distances between cluster center to each object**

[The method is same as in Iteration 0 (b) or Iteration 1 (b)]

- 1st, calculate the Euclidean distances from centroid P_1 to each point A, B, C, D. You get $D_{11}, D_{12}, D_{13}, D_{14}$ as the 1st row of the distance matrix.
- 2nd, calculate the Euclidean distances from centroid P_2 to each point A, B, C, D. You get $D_{21}, D_{22}, D_{23}, D_{24}$ as the 2nd row of the distance matrix.

(c) **Distance matrix becomes**

$$D^2 = \begin{Bmatrix} 0.5 & 0.5 & 3.2 & 4.61 \\ 4.3 & 3.54 & 0.71 & 0.71 \end{Bmatrix}$$

1st row indicates **group-1** cluster
2nd row indicates **group-2** cluster

(d) **Objects clustering into groups:**

Assign group to each object based on the minimum distance.

- medicine A no change, remains in **group 1**;
- medicine B no change, remains in **group 1**;
- medicine C no change, remains in **group 2** ,
- medicine D no change, remains in **group 2**.

Group Matrix : matrix element is 1 if the object is assigned to that group

$$G^2 = \begin{Bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{Bmatrix}$$

1st row indicates **group-1** cluster
2nd row indicates **group-2** cluster

4. Finally, the result state that Group Matrix $G^2 = G^1$.

This means the grouping of objects in this last iteration and the one before does not change anymore. Thus, the computation of the k-mean clustering has reached its stability and no more iteration is needed.

Results of final grouping are :

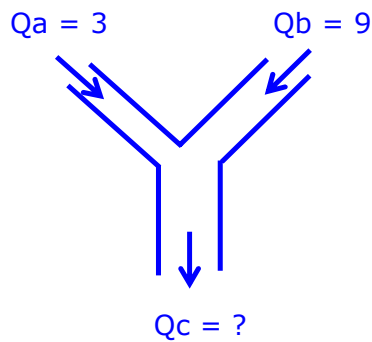
Objects	Feature 1 (X) Weight index	Feature 2 (Y) pH	Cluster Group
Medicine A	1	1	1
Medicine B	2	1	1
Medicine C	4	3	2
Medicine A	3	4	2

7. Analogy

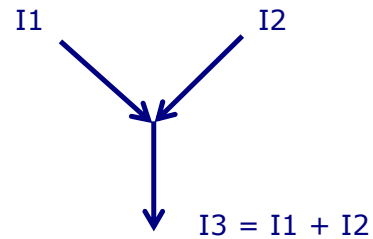
Learning by analogy means acquiring new knowledge about an input entity by transferring it from a known similar entity.

This technique transforms the solutions of problems in one domain to the solutions of the problems in another domain by **discovering analogous states** and operators in the two domains.

Example: Infer by analogy the hydraulics laws that are similar to Kirchoff's laws.



Hydraulic Problem



Kirchoff's First Law

The other similar examples are :

- Pressure Drop is like Voltage Drop
- Hydrogen Atom is like our Solar System :

The Sun has a greater mass than the Earth and attracts it, causing the Earth to revolve around the Sun. The nucleus also has a greater mass than the electron and attracts it. Therefore it is plausible that the electron also revolves around the nucleus.

8. Neural net and Genetic Learning

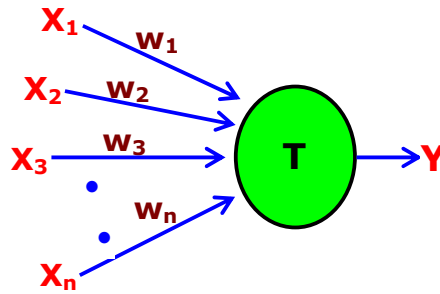
The Neural net, the Genetic learning and the Reinforcement learning are the Biology-inspired AI techniques. In this section the Neural net and Genetic learning are briefly described.

8.1 Neural Net (NN)

A neural net is an artificial representation of the human brain that tries to simulate its learning process. An artificial neural network (ANN) is often just called a "neural network" (NN).

- Neural Networks **model a brain** learning by example.
- Neural networks are structures "trained" to recognize input patterns.
- Neural networks typically take a vector of input values and produce a vector of output values; inside, they train weights of "neurons".
- A **Perceptron** is a model of a single 'trainable' neuron.

- **Perceptron** is a model of a single 'trainable' neuron.



- ⊕ Inputs are x_1, x_2, \dots, x_n as real numbers or boolean values depending on the problem.
- ⊕ w_1, w_2, \dots, w_n are weights of the edges and are real valued.
- ⊕ T is the threshold and is real valued.
- ⊕ Y is the output and is boolean.
- If the net input which is $(w_1 x_1 + w_2 x_2 + \dots + w_n x_n)$ is greater than the threshold T then output Y is **1** else **0**.
- Neural network uses supervised learning, in which inputs and outputs are known and the goal is to build a representation of a function that will approximate the input to output mapping.

2 Genetic Learning

Genetic algorithms (GAs) are part of evolutionary computing.

GA is a rapidly growing area of AI.

- Genetic algorithms are implemented as a computer simulation, where techniques are inspired by evolutionary biology.
- Mechanics of biological evolution:
 - ‡ Every **organism** has a set of **rules**, describing how that organism is built, and encoded in the **genes** of an organism.
 - ‡ The genes are connected together into long strings called **chromosomes**.
 - ‡ Each gene represents a specific trait (feature) of the organism and has several different settings, e.g. setting for a hair color gene may be black or brown.
 - ‡ The genes and their settings are referred as an organism's **genotype**.
 - ‡ When two organisms mate they share their genes. The resultant offspring may end up having half the genes from one parent and half from the other. This process is called **cross over**.
 - ‡ A gene may be **mutated** and expressed in the organism as a completely new trait.
- Thus, Genetic Algorithms are a way of solving problems by mimicking processes the nature uses ie **Selection, Crosses over, Mutation** and **Accepting** to evolve a solution to a problem.

■ Genetic Algorithm steps

- (1) **[Start]** Generate random population of n chromosomes (Encode suitable solutions for the problem)
- (2) **[Fitness]** Evaluate the fitness $f(x)$ of each chromosome x in the population.
- (3) **[New population]** Create a new population by repeating following steps until the new population is complete.
 - (a) **[Selection]** Select two parent chromosomes from a population according to their fitness.
 - (b) **[Crossover]** With a crossover probability, cross over the parents to form new offspring (children). If no crossover is performed, the offspring would be the exact copy of parents.
 - (c) **[Mutation]** With a mutation probability, mutate the new offspring at each locus (position in chromosome).
 - (d) **[Accepting]** Place new offspring in the new population.
- (4) **[Replace]** Use new generated population for a further run of the algorithm.
- (5) **[Test]** If the end condition is satisfied, stop, and return the best solution in the current population.
- (6) **[Loop]** Go to step 2.

- Genetic Algorithms does **unsupervised learning** - the right answer is not known beforehand.

9. Reinforcement Learning

Reinforcement learning refers to a class of problems in machine learning which postulate an **agent exploring an environment**.

- The agent perceives its current state and takes actions.
- The environment, in return, provides a reward positive or negative.
- The algorithms attempt to find a policy for maximizing cumulative reward for the agent over the course of the problem.

In other words, the **definition** of Reinforcement learning is :

" A computational approach to learning whereby an agent tries to maximize the total amount of reward it receives when interacting with a complex, uncertain environment."

Reinforcement learning is "a way of programming agents by reward and punishment without needing to specify how the task is to be achieved".

[Kaelbling, Littman, & Moore, 96]

The RL Problem and the tasks are illustrated in the next few slides.

2.1 Reinforcement Learning (RL) Problem

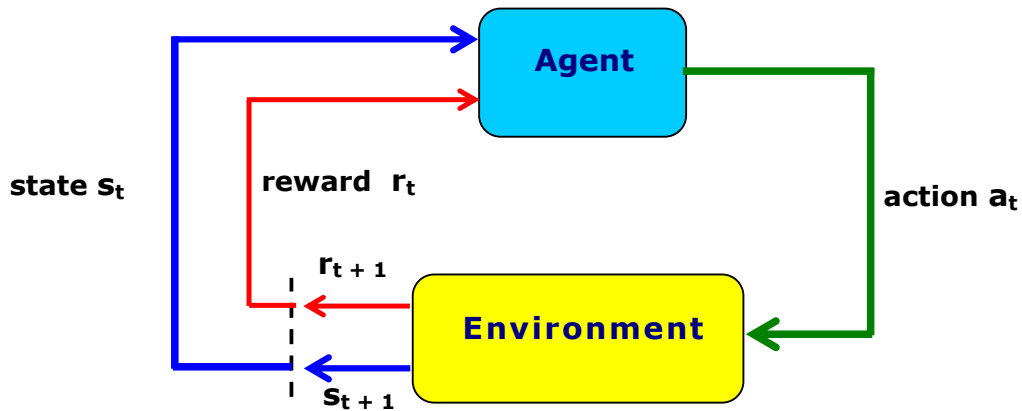
Reinforcement Learning is a computational approach to understanding and automating goal-directed learning and decision-making.

- **Agent - Environment interaction**

RL emphasizes learning from interaction with its environment.

- RL uses a formal framework defining the interaction between a learning agent and its environment in terms of states, actions, and rewards.
- The state-action pairs form value functions that does efficient search in the space of policies guided by evaluations of all policies.

Agent - Environment interaction in Reinforcement learning



- ‡ The agent and environment interact in a sequence of discrete time steps, $t = 0, 1, 2, 3, 4, \dots$
- ‡ At each discrete time t , the agent (learning system) observes state $s_t \in \mathbf{S}$ and chooses action $a_t \in \mathbf{A}$
- ‡ Then agent receives an immediate numerical reward $r_{t+1} \in \mathfrak{R}$ and the state changes to s_{t+1}
- ‡ At each time step, the agent implements a mapping from states to probabilities of selecting each possible action.
- ‡ This mapping is called the agent's policy π_t , where $\pi_t(s, a)$ is the probability at $a_t = a$ if $s_t \in \mathbf{S}$.
- ‡ RL methods specify how the agent changes its policy as a result of its experience.
- ‡ The agent's goal, is to maximize the total amount of reward it receives over the long run.

● Key Features of RL

- The learner is not told what actions to take, instead it finds out what to do by trial-and-error search.
- The environment is stochastic; ie., the behavior is non-deterministic means a "state" does not fully determine its next "state".
- The reward may be delayed, so the learner may need to sacrifice short-term gains for greater long-term gains.
- The learner has to balance between the need to explore its environment and the need to exploit its current knowledge.

9.2 Reinforcement Learning tasks

Most RL research is conducted within the mathematical framework of Markov decision processes (**MDPs**).

- The MDPs involve a decision-making **agent** interacting with its **environment** so as to maximize the cumulative **reward** it receives over time.
- The agent perceives aspects of the environment's **state** and selects **actions**.
- The agent may estimate a **value function** and use it to construct better and better decision-making **policies** over time.

Thus the elements of Reinforcement Learning are

- ‡ Markov Decision Process
- ‡ Policy
- ‡ Agent's Learning Task
- ‡ Reward
- ‡ Value Functions

Each of these elements are explained in the next few slides. However, first the Stochastic process, Markov chain, and few notations are explained.

● **Recall** : Stochastic processes, Markov chain , and the notations.

■ **Stochastic processes**

A physical stochastic processes is a sequences of events or path governed by probabilistic laws.

A stochastic process $X = \{ X(t), t \in T \}$ is a collection of random variables. For each t in the index set T , the $X(t)$ is a random variable. The t is often interpreted as time and $X(t)$ the state of the process at time t .

If the index set T is a countable set,

then we have a **discrete-time stochastic process**;

Else if T is a non-countable continuous set,

then we have a **continuous-time stochastic process**.

Any realization of X is named a sample path, which can be discrete or continuous.

■ **Markov chain**

A Markov chain, is a stochastic process with the **Markov property**.

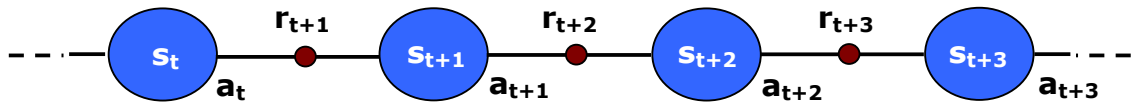
Markov property means, given the present state, the **future states are independent of the past states**; means the future states depends only on its present state, and not on any past states.

In other words, the description of the present state fully captures all the information that could influence the future evolution of the process. Future states will be reached through a probabilistic process instead of a deterministic one.

At each step the system may change its state from the current state to another state, or remain in the same state, according to a certain probability distribution. The changes in state are called **transitions**, and the probabilities associated with various state-changes are called **transition probabilities**.

■ Notations followed

- t** discrete time step
- T** final time step of an episode
- s_t** state perceived by agent at time **t**
- a_t** action selected by agent at time **t**
- r_{t+1}** reward received by agent at time **t**
- s_{t+1}** resulting next state perceived by agent at time **t**



- R_t** return (cumulative discounted reward) following **t**
- R_t⁽ⁿ⁾** **n** - step return
- R_t^λ** **λ** - return
- π** policy, decision-making rule
- π(s)** action taken in state **s** under deterministic policy **π**
- π(s, a)** probability of taking action **a** in state **s** under stochastic policy **π**
- S** set of all non-terminal states
- S⁺** set of all states, including the terminal state
- A(s)** set of actions possible in state **s**
- p^a_{ss'}** probability of transition from state **s** to state **s'** under action **a**
- r^a_{ss'}** expected immediate reward on transition from state **s** to state **s'** under action **a**

Markov System

A system that can be in one of several specified states with a specified time step, and at each step the system will randomly change states or remain where it is according to some fixed probability. The probability of going from State i to State j is a fixed number p_{ij} , called the transition probability.

Example : Laundry Detergent Switching

A market analyst for Detergents is interested to know whether consumers prefer powdered detergents or liquid detergents.

Two market surveys taken 1 year apart revealed that :

- 20% of all powdered detergent users had switched to liquid 1 year later, while the rest were using powder.
- 10% of liquid detergent users had switched to powder 1 year later, with the rest using liquid.

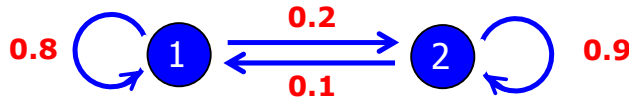
Analysis of the survey

1. Every year a consumer may be in one of two possible states:
 - State 1, he is a powdered detergent user;
 - State 2, he is a liquid detergent user.
2. There is a basic time step of 1 year;
3. If a consumer during a given year is in State 1, a powdered detergent user, then the probability that he will be in State 2, a liquid detergent user, in the next year is 20% = 0.2 denoted by $p_{12} = 0.2$.
Therefore probability for him to remain in State 1 is thus $(1 - 0.2 = 0.8)$, denoted by $p_{11} = 0.8$.
4. If a consumer during a given year is in State 2, a liquid detergent user, then the probability that he will be in State 1, a powdered detergent user, in the next year is 10% = 0.1 denoted by $p_{21} = 0.1$.
Therefore probability for him to remain in State 2 is thus $(1 - 0.1 = 0.9)$, denoted by $p_{22} = 0.9$.

The state transition diagram is shown in the next slide.

[Continued from previous slide - Example : Laundry Detergent Switching]

We can draw a state transition diagram or use a matrix to represent these probabilities called transition probabilities.



State transition diagram

$$P = \begin{Bmatrix} 0.8 & 0.2 \\ 0.9 & 0.1 \end{Bmatrix}$$

State transition matrix

The Figure 1 and 2 : the numbers are transition probabilities, from State i to State j , a number p_{ij} , and the arrows show the switching directions.

■ **Markov State Transition**

State transition probabilities mentioned before are explained here.

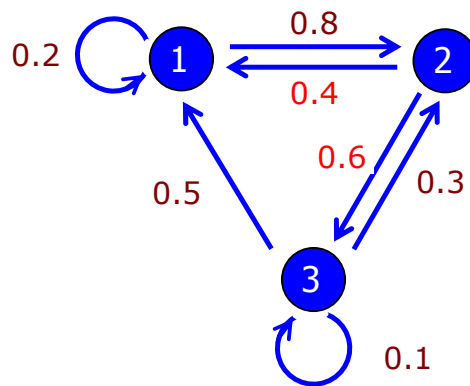
Transition probability :

If a Markov system is in state i , there is a fixed probability p_{ij} of it going into state j in the next time step. The probability p_{ij} is called a transition probability.

Transition diagram :

A Markov system is illustrated by a state transition diagram, that shows all the states and transition probabilities.

Example : Transition Diagram (Missing arrows indicate zero probability.)

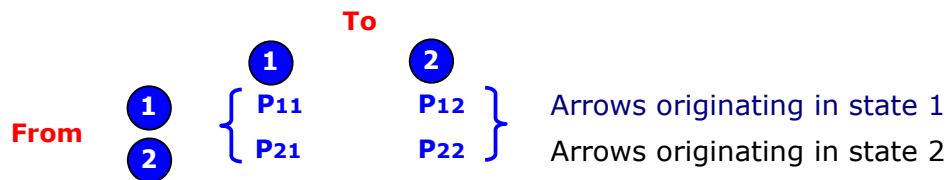


■ **Transition matrix :**

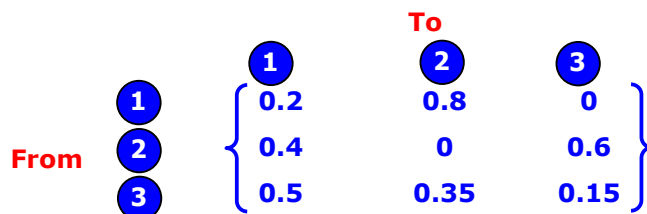
The matrix P whose ij^{th} entry is p_{ij} is called transition matrix associated with the system. The entries in each row add up to 1 .

Example :

A 2×2 transition matrix P



The 3×3 Matrix corresponding to Transition Diagram

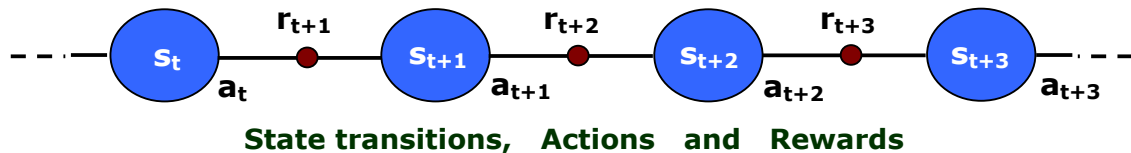


Markov Decision Processes (MDPs)

Markov decision processes (MDPs), provide a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of the decision maker.

Description :

- MDP is a discrete time stochastic control process characterized by a set of states;
- In each state there are several actions from which the decision maker must choose. For a state **s** and an action **a**, a state transition function **Pa(s)** determines the transition probabilities to next state.
- The decision maker earns a reward **r** for each state transition. The state transitions possess the Markov property, ie., given the state of the MDP at time **t**, the transition probabilities to the state at time **t+1** are independent of all previous states or actions.
- MDPs are an extension of Markov chains. The differences are the addition of actions (allowing choice) and the rewards (giving motivation). If there are only one action, or if the action to take were fixed for each state, a Markov decision process would reduce to a Markov chain.



- ‡ Set of states **S**
- ‡ Set of actions **A(s)** available in each state **S**
- ‡ Markov assumption: **s_{t+1}** and **r_{t+1}** depend only on **s_t**, **a_t** and not on anything that happened before **t**
- ‡ Rewards: on transition from state **S** to state **S'** under action **a**

$$r_{ss'}^a = E \left\{ r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s' \right\}$$
- ‡ Transition probabilities : from state **S** to state **S'** under action **a**

$$p_{ss'}^a = P (s_{t+1} = s' \mid s_t = s, a_t = a)$$

- **Agent's Learning Task**

Agent execute actions in the environment.

Observe results, and learn policy which is a strategy (a way of behaving).

$$\pi : \mathbf{S} \times \mathbf{A} \rightarrow [0, 1] \text{ with } \pi(\mathbf{s}, \mathbf{a}) = \mathbf{P}(\mathbf{a}_t = \mathbf{a} | \mathbf{s}_t = \mathbf{s})$$

If the policy is deterministic, then write more simply as

$$\pi : \mathbf{S} \rightarrow \mathbf{A} \text{ with } \pi(\mathbf{s}) = \mathbf{a}$$

giving the action chosen in the state **S**.

- **Policy**

It defines the learning agent's way of behaving at a given time.

A mapping from states to actions (deterministic policy), or the distributions over actions (stochastic policy).

- It is a mapping from perceived states of the environment to actions to be taken in those states.
- It in some cases may be a simple function or lookup table; it may involve extensive computation like a search process.
- It is the core of a reinforcement learning agent; it alone is sufficient to determine behavior.
- In general it may be stochastic (random).

● **Reward Function**

The function defines the goal in a reinforcement learning problem.

- Maps each perceived state (or state-action pair) of the environment to a single number, indicating intrinsic desirability of that state.
- Indicates what is good in an immediate sense.
- Tells what are the good and bad events for the agent;
Example : a biological system identify rewards high with pleasure and low with pain
- RL agent's sole objective is to maximize the total reward it receives in the long run.
- Rewards servers a basis for altering the policy;
Example : an action selected by the policy if followed by low reward, then the policy may be changed to select some other action in that situation.
- In general, the reward function may be stochastic.

● **Maximize Reward**

Agent's goal is to maximize reward it receives in long run.

"Agent-Environment" interaction is generally a sequence of episodes.

If the interaction result into a sequence of separate episodes, it is known as **episodic tasks**. If the interaction does not break into identifiable episodes, but continue without limit is called continuing tasks.

- If Episodic task then the simplest return R_t is sum of rewards

$$R_t = r_{t+1} + r_{t+2} + \dots + r_T,$$

where T is the time when a terminal state is reached.

- If Continuing tasks then the discounted return R_t

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=1}^{\infty} \gamma^{t+k-1} r_{t+k},$$

where parameter γ is discount factor, $0 \leq \gamma \leq 1$.

- If Average-reward tasks then return R_t

$$R_t = \lim_{T \rightarrow \infty} (1/T) (r_{t+1} + r_{t+2} + \dots + r_T)$$

● **Value Functions**

The value of state s under policy π is the expected return when starting from s and choosing actions according to π :

$$V_{K=1}^{\pi'}(s) = E_{\pi} \{ R_t | s_t = s \} = E_{\pi} \left\{ \sum_{k=1}^{\infty} \gamma^{t+k-1} r_{t+k} | s_t = s \right\}$$

10. References : Textbooks

1. "Artificial Intelligence", by Elaine Rich and Kevin Knight, (2006), McGraw Hill companies Inc., Chapter 17, page 447-484.
2. "Artificial Intelligence: A Modern Approach" by Stuart Russell and Peter Norvig, (2002), Prentice Hall, Chapter 18-21, page 649-788.
3. "Computational Intelligence: A Logical Approach", by David Poole, Alan Mackworth, and Randy Goebel, (1998), Oxford University Press, Chapter 11, page 397-438.
4. "Artificial Intelligence: Structures and Strategies for Complex Problem Solving", by George F. Luger, (2002), Addison-Wesley, Chapter 10- 13, page 385-570.
5. "AI: A New Synthesis", by Nils J. Nilsson, (1998), Morgan Kaufmann Inc., Chapter 10, Page 163-178.
6. "Artificial Intelligence: Theory and Practice", by Thomas Dean, (1994), Addison-Wesley, Chapter 5, Page 179-254
7. Related documents from open source, mainly internet. An exhaustive list is being prepared for inclusion at a later date.