Artificial Intelligence

Module 6 Planning and Learning

Rajesh Kumar,
VIT Chennai

Contents

Planning

- *Definition : Planning is arranging a sequence of actions to achieve a goal.*
- Uses core areas of AI like searching and reasoning

Planning

- Planning starts with general facts about the world
  - Facts about the particular situation and a statement of a goal.
  - Facts about the effects of actions,
  - Planning programs generate a strategy for achieving the goal.
- Generally, the strategy is just a sequence of actions.

Planning

- A planning agent will construct plans to achieve its goals, and then execute them.
- Analyse a situation in which it finds itself and develop a strategy for achieving the agent's goal.
- Achieving a goal requires finding a sequence of actions that can be expected to have the desired outcome.

Need

- **Domain description**
- **Action specification**
- **Goal description**

Plan Representation

- Representation of actions
  - actions generate successor states
- Representation of states
  - representations are complete
- Representation of goals
  - goal test and heuristic function
- Representation of plans
  - unbroken sequence of actions leading from initial to goal state

Classical planning

- Each action is indivisible
- No concurrent actions are allowed
- Deterministic actions
- Complete knowledge of the state

Level of Planning

- Forward State Space Planning (FSSP)
  - **Disadvantage:** Large branching factor
  - **Advantage:** Algorithm is Sound
- Backward State Space Planning (BSSP)
  - **Disadvantage:** Not a sound algorithm (sometimes inconsistency can be found)
  - **Advantage:** Small branching factor (very small compared to FSSP)

Two kinds of planning

- *Projection* into the future
  - The planner searches through the possible combination of actions to find the *plan* that will work
- *Memory based planning*
  - looking into the past
  - The agent can retrieve a plan from its memory

Components of Planning

- Choose the best rule to apply next - based on the best available heuristic information.
- Apply the choosen rule to compute the new probem state that arises from its application
- Detect when a solution has been found
- Detect dead ends so that they can be abandoned and the system's effort directed in more fruitful directions.
- Detect when an atmost corret solution has been found and employ special techniques to make it totally correct.

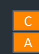
Blocks World Problem

- Compare the variety of methods of planning,
  - we should find it useful to look at all of them in a single domain
  - Complex enough that the need for each of the mechanisms is apparent
  - Yet simple enough that easy-to-follow examples can be found.

Blocks World Problem

- There is a flat surface on which blocks can be placed.
- There are a number of square blocks, all the same size.
- They can be stacked one upon the other.
- There is robot arm that can manipulate the blocks

Plan for the block world problem



- For the given problem, Start → Goal can be achieved by the following sequences
- Actions of the robot arm
  - Unstack(C,A) - Pick up block C from its current position on block A
  - Putdown(C) - Put block C down on the table
  - Pickup(B) - Pick up block B from the table and hold it.
  - Stack(B,C) - Place block B on block C

States by predicates

- **Predicates**
  In order to specify both the conditions under which an operation may be performed and the results of performing it, we need the following predicates:
  1. ON(A, B): Block A is on Block B.
  2. ONTABLES(A): Block A is on the table.
  3. CLEAR(A): There is nothing on the top of Block A.
  4. HOLDING(A): The arm is holding Block A.
  5. ARMEMPTY: The arm is holding nothing.

Blocks World Problem

- [CLEAR(x,s) ∧ ON(x,y,s)] → [HOLDING(x, Do (UNSTACK(x,y),s) ∧
  CLEAR(y, Do(UNSTACK(x,y),s)]
- DO (function) - for a given state and given action the new state that results from the execution of the action.
- if CLEAR(x) and ON(x,y) both hold in state s
  - then HOLDING(x) and CLEAR(y) will hold in the state S1
  - that results from DOing an UNSTACK(x,y) starting in state s.

Blocks World Problem

- Frame axioms - The components of the state that are not affected by each operator.
  - ONTABLE(z,s) --> ONTABLE(z,(DO(UNSTACK(x,y),s))
  - ONTABLE relation is never affected by the UNSTACK operator
- ON relation is only affected by the UNSTACK operator if
  - the blocks involved is ON relation are the same ones involved in the UNSTACK operation.
  - [ON(m,n,,s) ∧ ~ EQUALON(m,x)] → [ON(m,n, Do (UNSTACK(m,y),s)]
    - Advantage – single mechanism to perform all the operation for state description
    - More Axioms.
  - COLOR(x,c,s) → COLOR(x,c, Do (UNSTACK(y,z),s)

STRIPS

- Stanford Research Institute Problem Solver (1970s)
- Knowledge Representation : First Order Logic.
- Algorithm : Forward chaining on rules.
- Any search procedure : Finds a path from *start* to *goal*.
  - Forward Chaining : Data-driven inferencing
  - Backward Chaining : Goal-driven

STRIPS ESSENSE

A state in STRIPS consists of a set of formulae describing the current world state.

Planning is search through the space of world states, using actions as operators to generate the search space.

Actions connect 'before' and 'after' world states

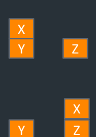
Before and after states are described using only ground literals (conjunction thereof) and some general axioms.

STRIPS ESSENSE ...

- Actions are described by preconditions and effects (conjunctions of literals).
- Effects are split into an ADD-list and a DELETE-list:
  - the ADD-list contains every new formula to be added to the current state as result of the action
  - the DELETE-list contains all formulae to be deleted from the current state as result of the action

Note: ADD- and DELETE-lists explicitly specify what becomes true and what becomes false after an action. Thus, the state is carried along, just with necessary changes made. This circumvents the Frame Problem.

STRIPS ESSENSE ...



Actions in STRIPS are described as schemata with variables which are instantiated during planning (→ unification).

Example:

move (x, y, z)

precondition:	on (x, y) ∧ clear (x) ∧ clear (z)
delete-list:	clear (z), on (x, y)
add-list:	on (x, z), clear (y), clear (Table)
	in case z=Table

Example : Blocks World

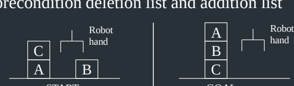
Fundamental Problem :
The *frame problem* in AI is concerned with the question of what piece of knowledge is relevant to the situation.

Fundamental Assumption : Closed world assumption
If something is not asserted in the knowledge base, it is assumed to be false.

(Also called "Negation by failure")

Example : Blocks World

•STRIPS : A planning system – Has rules with precondition deletion list and addition list



START	GOAL
on(B, table)	on(C, table)
on(A, table)	on(B, C)
on(C, A)	on(A, B)
hand empty	hand empty
clear(C)	clear(A)
clear(B)	

Rules

•*R1 : pickup(x)*
Precondition: clear(x) Λ on(x,table) Λ ARMEMPTY
Deletion List : on(x,table) Λ ARMEMPTY
Add List : holding(x)

•*R2 : putdown(x)*
Precondition : holding(x)
Deletion List : holding(x)
Add List : ARMEMPTY, on(x,table), clear(x)

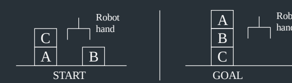
Rules

•*R3 : stack(x,y)*
Precondition: holding(x) Λ clear(y)
Deletion List :holding(x) Λ clear(y)
Add List : ARMEMPTY Λ on(x,y)

•*R4 : unstack(x,y)*
Precondition : on(x,y)Λ clear(x) Λ ARMEMPTY
Deletion List : on(x,y) Λ ARMEMPTY
Add List : holding(x), clear(y)

Example : Blocks World

•STRIPS : A planning system – Has rules with precondition deletion list and addition list



Sequence of actions :

1. Grab C
2. Pickup C
3. Place on table C
4. Grab B
5. Pickup B
6. Stack B on C
7. Grab A
8. Pickup A
9. Stack A on B

ADL - Action Description Language

Can be seen as extension of the STRIPS language
Contains type of parameters
Allows explicit expression of negation
Allows equality of terms in precondition formulas

ADL

- Consider the problem of air freight transport,
  - certain goods must be transported from an airport to another airport by plane
  - airplanes need to be loaded and unloaded.
- The necessary actions would be
  - *loading*, *unloading* and *flying*;
  - over the descriptors one could express
    - In(c, p) - freight *c* is in an airplane *p*
    - At(x, A) - an object *x* is at an airport *A*

ADL

Action (
Load (c: Freight, p: Airplane, A: Airport)
Precondition: At(c, A) ^ At(p, A)
Effect: ¬At(c, A) ^ In(c, p)
)

Action (
Unload (c: Freight, p: Airplane, A: Airport)
Precondition: In(c, p) ^ At(p, A)
Effect: At(c, A) ^ ¬In(c, p)
)

Action (
Fly (p: Airplane, from: Airport, to: Airport)
Precondition: At(p, from)
Effect: ¬At(p, from) ^ At(p, to)
)

Example 1

- A car is punctured. The flat wheel from the axle needs to be changed with the spare wheel from the trunk

Example

- Flat tire problem
- Initial state :
  - Flat tire – axle
  - Spare tire – trunk
- Goal state :
  - Spare tire – Axle
  - Flat tire-trunk

Actions

1) Remove spare
2) Remove flat
3) Put spare -> axle
4) Put flat -> trunk

Remove spare

- Action (Remove (spare, trunk))
  - Precondition : at(spare, trunk)
  - Effect : ~ at (spare, trunk)

Remove flat

- Action (Remove (flat, axle))
  - Precondition : at(flat, axle)
  - Effect : ~ at (Flat, axle)

Put spare -> axle

- Action (put (spare, axle))
  - Precondition : ~at(spare, trunk)
  - Effect : at (spare, axle)

Put flat -> trunk

- Action (put (flat, trunk))
  - Precondition : ~at(flat, axle)
  - Effect : at (flat, trunk)

Example 2

Put right and left shoes

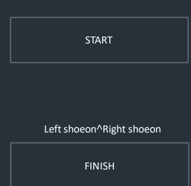
Initial :

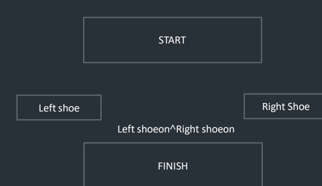
S1 : Action:Start

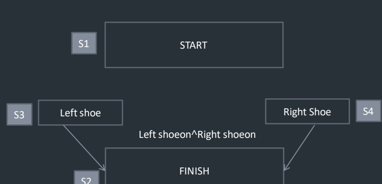
S2 : Action: Finish

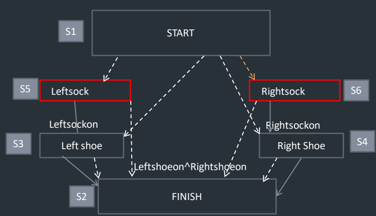
Precond:RightShoeon^LeftshoeOn

- Actions:
- Op(Action: Rightshoeon,
  Precond: Rightsockon
  Effect : Rightshoeon)
- Op(Action : rightsockon,
  Effect : Rightsockon)
- Op(Action :Leftshoeon,
  Precond: Leftsockon
  Effect : Leftshoeon)
- Op(Action: Leftsockon,
  effect:Leftsockon)









Planning Problems

1. You need to transfer a cargo package from the aiport1 to another airport2
2. You are asked to go to a supermarket and get eggs and than go to a book store and get a book for your mother.

Draw Planning for the above two scenarios. Include conditional constraints and ordering constraints if any

Air Cargo Transport

- Init(At(C_1, CLE) ∧ At(C_2, LAS) ∧ At(P_1, CLE) ∧ At(P_2, LAS) ∧ Cargo(C_1) ∧ Cargo(C_2) ∧ Plane(P_1) ∧ Plane(P_2) ∧ Airport(CLE) ∧ Airport(LAS))
- Goal(At(C_1, LAS) At(C_2, CLE))

Air Cargo Transport

- *Action(Load(c, p, a),*
  Precond: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
  Effect: ¬ At(c, a) ∧ In(c, p))
- *Action(Unload(c, p, a),*
  Precond: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧Plane(p) ∧ Airport(a)
  Effect: At(c, a) ∧ ¬ In(c, p))
- *Action(Fly(p, from, to),*
  Precond: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
  Effect: ¬ At(p, from) ∧ At(p, to))

Partial Order Planning

Partial Order Planning

- **Partial-order planning** is an approach to automated planning
- It maintains a partial ordering between actions
  - It commits ordering between actions when forced to
    - Ordering of actions is partial.
    - This planning doesn't specify which action will come out first when two actions are processed.
  - It Specifies all actions that need to be taken, but specifies an ordering between actions only where necessary.
  - Principle of Least Commitment
- **Total-order planning** maintains a total ordering between all actions at every stage of planning.

Prof. Rajesh Kumar VIT Chennai

A plan for baking a cake starts

- A partial plan
  - Go to the store
  - Get eggs; get flour; get milk
  - Pay for all goods
  - Go to the kitchen
- There is no order specified
  - The IA roams store until the list is complete.

Example

Planning problem

Initial state : no chips and no books at home

Goal state : chips and books at home

POP

- Initial state:
- Op(Action : Start,

  Effect: At(home)^ sells(BS, Book) ^sells(VS, chips))

POP

- Initial state:
- Op(Action : Start,

  Effect: At(home)^ sells(BS, Book) ^sells(VS, chips))
- Goal state

Op(Action : Finish,

Precond: At(home)^ have (Book) ^ have (chips))

POP

Actions

Op(Action : Go(there)

precond: at(here)

Effect: At(there)^~at(here))

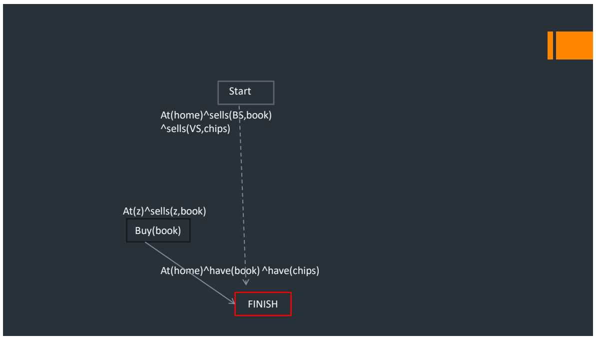
Actions

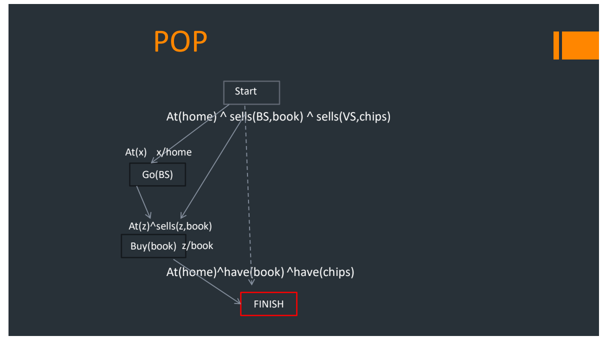
Op(Action : Buy(x)

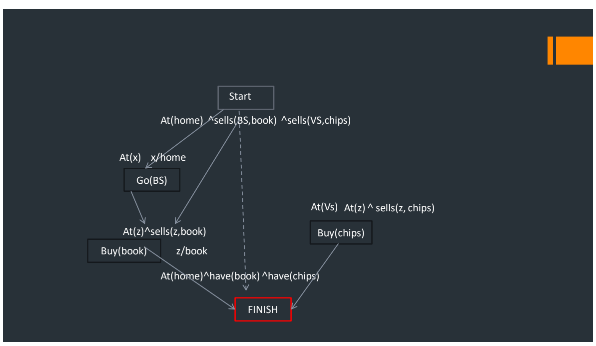
precond: at(store) ^ Sells(store,x)

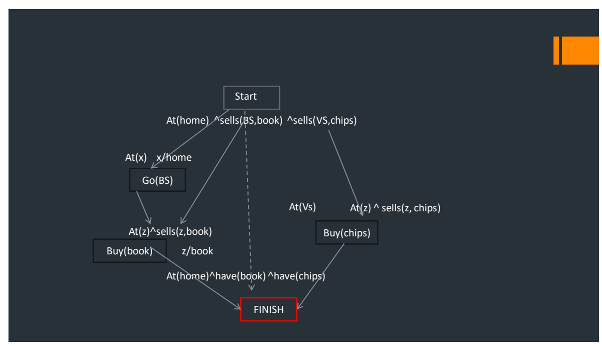
Effect: have(x))

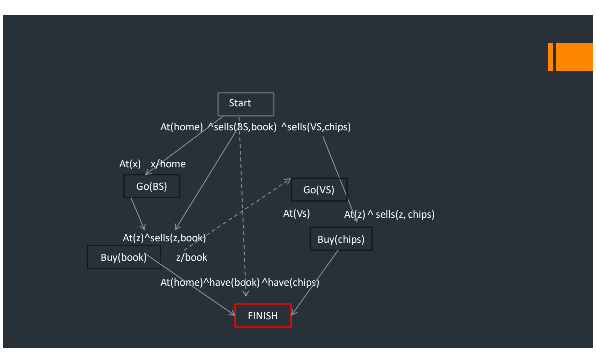
POP

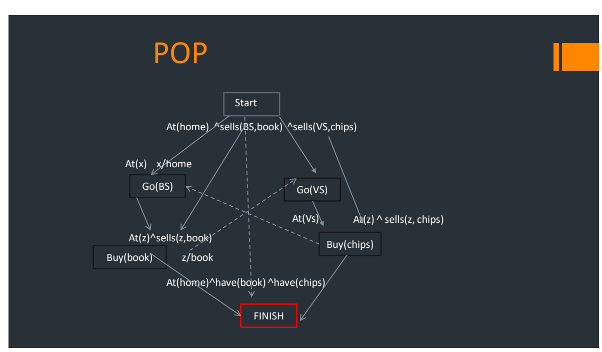

Start
At(home)^sells(BS,book)
^sells(VS,chips)
At(z)^sells(z,book)
Buy(book)
At(home)^have(book) ^have(chips)
FINISH


POP
Start
At(home) ^ sells(BS,book) ^ sells(VS,chips)
At(x) x/home
Go(BS)
At(z)^sells(z,book)
Buy(book) z/book
At(home)^have(book) ^have(chips)
FINISH


Start
At(home) ^sells(BS,book) ^sells(VS,chips)
At(x) x/home
Go(BS)
At(Vs) At(z) ^ sells(z, chips)
Buy(chips)
At(z)^sells(z,book)
Buy(book) z/book
At(home)^have(book) ^have(chips)
FINISH


Start
At(home) ^sells(BS,book) ^sells(VS,chips)
At(x) x/home
Go(BS)
At(Vs) At(z) ^ sells(z, chips)
Buy(chips)
At(z)^sells(z,book)
Buy(book) z/book
At(home)^have(book) ^have(chips)
FINISH


Start
At(home) ^sells(BS,book) ^sells(VS,chips)
At(x) x/home
Go(BS)
Go(VS)
At(Vs) At(z) ^ sells(z, chips)
Buy(chips)
At(z)^sells(z,book)
Buy(book) z/book
At(home)^have(book) ^have(chips)
FINISH


POP
Start
At(home) ^sells(BS,book) ^sells(VS,chips)
At(x) x/home
Go(BS)
Go(VS)
At(Vs) At(z) ^ sells(z, chips)
Buy(chips)
At(z)^sells(z,book)
Buy(book) z/book
At(home)^have(book) ^have(chips)
FINISH

To Do

You are planning to bake a cake
Actions needed:
- go to the store
- get eggs;
- get flour;
- get sugar
- get milk
- go to the kitchen

POP

- Initial state:
- Op(Action : Start,
  Effect: At(kitchen)^ sells(ES, Eggs) ^sells(SM, Flour) ^sells(SM, sugar) ^ sells (MS, milk))
- Goal state

Op(Action : Finish,
  Precond: At(Kitchen)^ have (Eggs) ^ have (Flour) ^have (sugar) ^have(milk))

POP

Actions
Op(Action : Go(there)
  precond: at(here)
  Effect: At(there)^~at(here))
Actions
Op(Action : Buy(x)
  precond: at(store) ^ Sells(store,x)
  Effect: have(x))

- Partial-order planning is more adept at finding the quickest path, and is therefore the more efficient of these two main types of planning.
  - POP
    - it is faster and thus more efficient
  - TOP
  - partial-order planning performs better because it produces more trivial serializability than total-order planning.
    - A planner's ability to perform quickly when dealing with goals that contain subgoals. Planners perform more slowly when dealing

Prof. Rajesh Kumar VIT Chennai

Forms of Learning

- **Inductive learning**
  - Learning by generalization from a set of examples
    - Learning by examples – given by
      - Teacher, Knowledge Base, Learner
    - General concept is inferred from examples
  - Learning by experiments
    - Stimulus – Response learning
  - Learner infers general rules and regulation which explains the observation, discovery
  - It does not need a teacher
  - A general function or rule from specific input–output pairs

Forms of Learning

- **Analytical or Deductive learning**
  - Going from a known general rule to a new rule
    - That is logically entailed
    - Useful because it allows more efficient processing
  - Knowledge reformulation
    - Conclusion
    - Compilation
  - Transforming the knowledge in usable form
    - Preserve the information content of original data

Learning by types of feedback

- **Supervised learning**
  - The agent observes some example input–output pairs and learns a function that maps from input to output.
    - The inputs are percepts and the output are provided by a teacher who says "Brake!" or "Turn left"
    - The inputs are camera images and the outputs again come from a teacher who says "that's a bus."
    - The theory of braking is a function from states and
      - braking actions to stopping distance in feet.
      - Agents gets the value from percept
      - Environment is the teacher

Learning by types of feedback

- **Unsupervised learning**
  - The agent learns patterns in the input even though no explicit feedback is supplied
    - Clustering – Detecting useful clusters of input examples
      - Good traffic days, Bad traffic days
      - Marketing groups based on so many factors to improvements of sales

Learning by types of feedback

- **Reinforcement learning**
  - the agent learns from a series of reinforcements
    - rewards or punishments.
    - The lack of a tip at the end of the journey gives the taxi agent an indication that it did something wrong.
    - The two points for a win at the end of a chess game tells the agent it did something right.
  - It is up to the agent to decide
    - which of the actions prior to the reinforcement were most responsible for it.

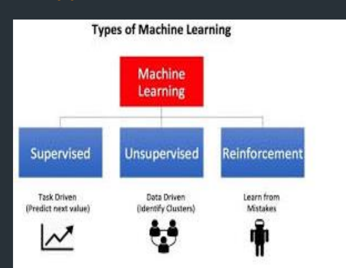
Learning by types of feedback

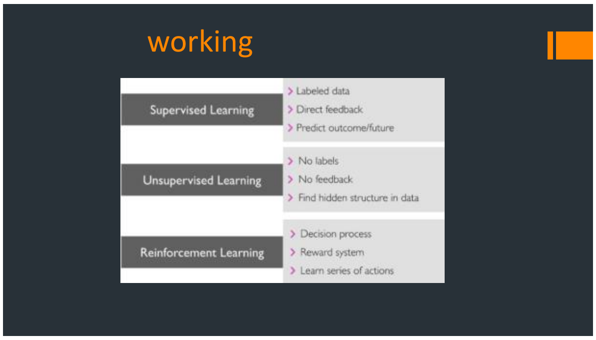
- **Semi-supervised learning**
  - we are given a few labeled examples and
    - must make out what we can of
      - a large collection of unlabeled examples.
  - The labels themselves may not be the oracular truths as per hope
    - Imagine that you are trying to build a system to guess a person's age from a photo.
    - You gather some labeled examples by snapping pictures of people and asking their age. (Supervised learning)

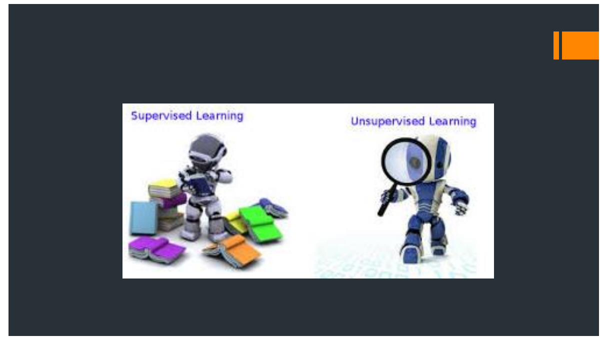
Learning by types of feedback

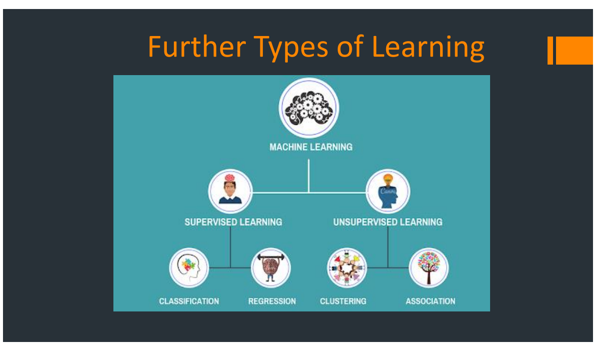
- **Semi-supervised learning**
  - But in reality some of the people lied about their age.
    - It's not just that there is random noise in the data
    - The inaccuracies are systematic
    - To uncover them is an unsupervised learning problem
      - Involving images,
        - self-reported ages, and
        - true (unknown) ages.
    - Both noise and lack of labels create a continuum between supervised and unsupervised learning.

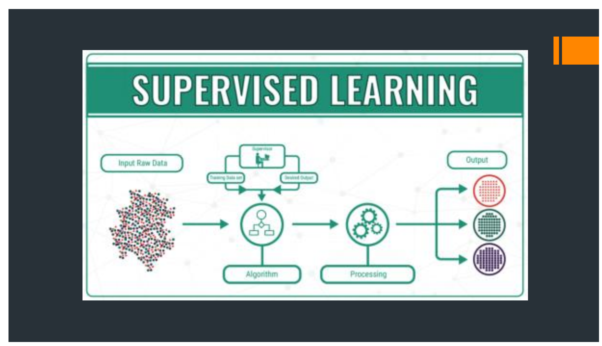
Types of ML

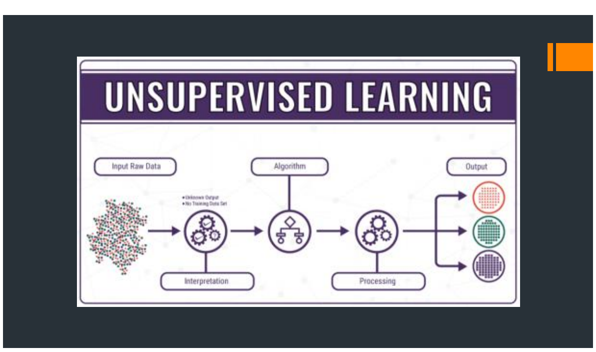


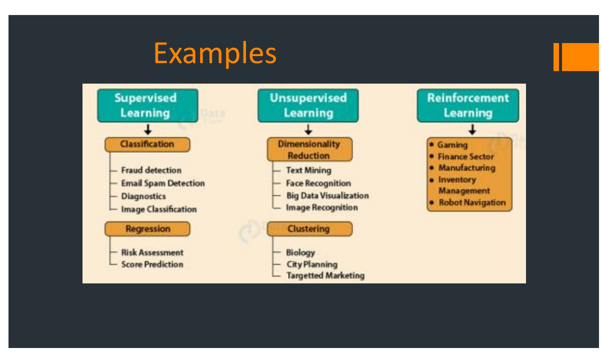

working
Supervised Learning
Labeled data
Direct feedback
Predict outcome/future
Unsupervised Learning
No labels
No feedback
Find hidden structure in data
Reinforcement Learning
Decision process
Reward system
Learn series of actions

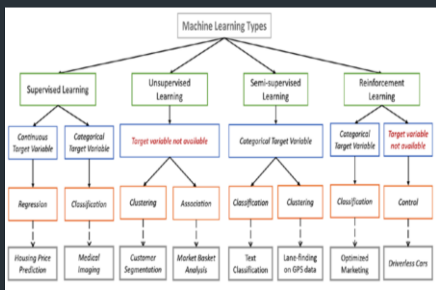

Supervised Learning
Unsupervised Learning

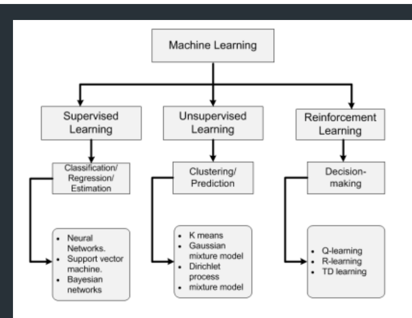

Further Types of Learning
MACHINE LEARNING
SUPERVISED LEARNING
UNSUPERVISED LEARNING
CLASSIFICATION
REGRESSION
CLUSTERING
ASSOCIATION


SUPERVISED LEARNING
Input Raw Data
Output
Algorithm
Processing


UNSUPERVISED LEARNING
Input Raw Data
Algorithm
Output
Interpretation
Processing


Examples
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





Hypothesis & Supervised learning

- A **factored representation**
  - Inputs - a vector of attribute values
  - Outputs that can be either a continuous numerical value or a discrete value.
- Given a **training set** of
  - N example input–output pairs
    - (x1, y1), (x2, y2), . . . (xN, yN) each yj was generated by
      - Number, Vectors or any value
    - An unknown function y = f(x)
    - Discover a function h (Hypothesis) that approximates the true function f

Supervised learning

- Learning is a search through the space of possible hypotheses
  - one that will perform well, even on new examples beyond the training set.
  - How much is the accuracy of a hypothesis
    - Generalization - correctly predicts the value of y for novel examples.
    - Test Set of examples
  - Function f may be stochastic – Conditional Probability Distribution => P(Y|x)

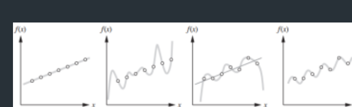
Supervised learning

- Classification
  - Y is one of a finite set of values
    - *sunny, cloudy* or *rainy*
  - Binary classification
- Regression
  - y is a number - tomorrow's temperature, price of a house
    - The probability that we have found *exactly* the right real-valued number for y is 0
    - Finding a conditional expectation or average value of y

Supervised learning

- Consistent hypothesis
  - Agrees with all the input data
- Hypothesis H space
  - $f(x) = x^5+3x^2+2$
  - $f(x) = 0.4x + 3$

$f(x)= ax + b + c \sin(x)$



Supervised learning

- *How do we choose from among multiple consistent hypotheses?*
  - **Ockham's razor**
  - Prefer the *simplest* hypothesis consistent with the data
  - Defining simplicity is not easy,
    - It seems clear that a degree-1 polynomial is simpler than a degree-7 polynomial

Hypothesis space

- All programs – Java, C
- All Turing machine
  - All computable function can be represented by some Turing machine
  - Computational complexity
- Simpler hypothesis h is usable after learning
  - Computing h(x) when h is a linear function is guaranteed to be fast
  - An arbitrary Turing machine program is not guaranteed to terminate.

Supervised learning

- *There is a tradeoff between*
  - *Complex hypotheses that fit the training data well*
  - *Simpler hypotheses that may generalize better.*
- **Realizable** learning problem
  - The hypothesis space contains the true function.
  - We cannot always tell whether a given learning problem is realizable.
    - The true function is not known

Supervised learning

- An analyst looking at a problem without data
  - Can make more fine-grained distinctions about the hypothesis space
    - Hypothesis is possible or impossible
    - How probable is the hypothesis.
    - $h^* = \underset{h \in H}{\text{argmax}}\ P(h|data)$
    - $h^* = \underset{h \in H}{\text{argmax}}\ P(data|h)\ P(h)$
      - P(h) high for low degree of polynomial, low for high degree of polynomial
      - Low probability for unusual looking function

Expressiveness–complexity tradeoff

- An expressive language makes it possible for a *simple* hypothesis to fit the data
- Restricting the expressiveness of the language
  - means that any consistent hypothesis (??) must be very complex.
  - FOL vs. Propositional Logic for Chess

- Training an algorithm with data in ML vs AI techniques?
- Purpose of Captcha with images and the process followed
  - to authenticate human vs. computer.

Reference : LET'S LEARN **ARTIFICIAL INTELLIGENCE** , Niti Aayog

Gathering data

- **GIGO**
- **Determining the Information to be collected**
  - Quality of data, Source, Quantity
- Time frame for data collection
  - Early plan, Discontinuous/Continuous collection
- Identify data source – Primary , Secondary, Onlione, Offline, , Authenticity of data
- Collect data and verify if it meets the requirement

Data collection

- **Data Discovery** - Search for new datasets
- **Data augmentation** - Collect data is insufficient for coverage
  - OpenCV - scaling, cropping, rotation, filters, blur, translation
  - **Scikit-image** – color, resize, erosion + above.
- **Data generation** – Generate/Synthesis more data for robust model, crowdsource.
  - Pydbgen, Mockaroo
- Data labelling - labeling the data with
  - Features, properties, characteristics, or classifications
  - To analyze patterns
  - http://labelme.csail.mit.edu/Release3.0/
  - https://labelbox.com/

Data coverage

- Distribution of data
  - Across target classes or values.
  - Over other variables that contribute to variation
  - Overcomes the variation embedded in what we are trying to measure
    - Normal vs Error
- Iterative data collection
  - Budget and time is limited
  - collecting too much may be very expensive and time-consuming

Data Preparation

- **Collected data is not suitable for analysis**
  - **Duplicate entries (few/ Too Many)**
  - **Typographical error -**
    - **Remove irrelevant data,**
    - **structural error – Cardinality**
    - **Outliers – Careful consideration (abnormality)**
  - **Missing data – Missing data treatment**
  - **Different format - Transformation**
- **Cleaning and Labelling of data is required**

Missing data

- **Missing at Random(MAR)**
  - The missing values in any feature or column are dependent on the values of other features or columns.
- Missing Not At Random (MNAR)
  - The value that missing is related to the reason it's missing.
    - **Depression column by Depressed person. (Issue)**
- **Missing** Completely At Random (MCAR)
  - the missing values in any features are not dependent on any other features values.
  - It is the highest level of randomness.

DATA TRANSFORMATION

- Data transforms or consolidates data into a form appropriate for machine learning
  - Normalization – Different scales in to 1 (KM, M, millimeter)
    - 0-1, 1-10
  - Decomposition - finding patterns in data with complex features
    - Making more features from one features.
    - Monthly demand from quarterly demand.
  - Aggregation – Combine several features in to one
    - Reduces dataset without loss of information

SPLITTING DATASET

- Training Set - to train a model and define its optimal parameters
  - Parameters are to be learnt from data
- Test Set – Evaluation of trained model
  - Generalization – unseen data
    - No overfitting of model
- Validation sets – Tweak the hyper parameters of the models
  - Can not be directly learnt from datasets
  - Find the pattern in data – Complexity of model

Classification/Regression

- "What will the temperature be in Mumbai tomorrow?"
- "Is this email spam or not spam?"
- "How many copies of this book will sell?"
- "Will the customer buy this product?"
- "Is this comment written by a human or a robot?"
- "What price will this car sell for?"
- "Is this product a book, movie, or clothing?
- "Which category of products is most interesting to this customer?"
- "Is this movie a romantic comedy, documentary, or thriller?"