MACHINE INTELLIGENCE

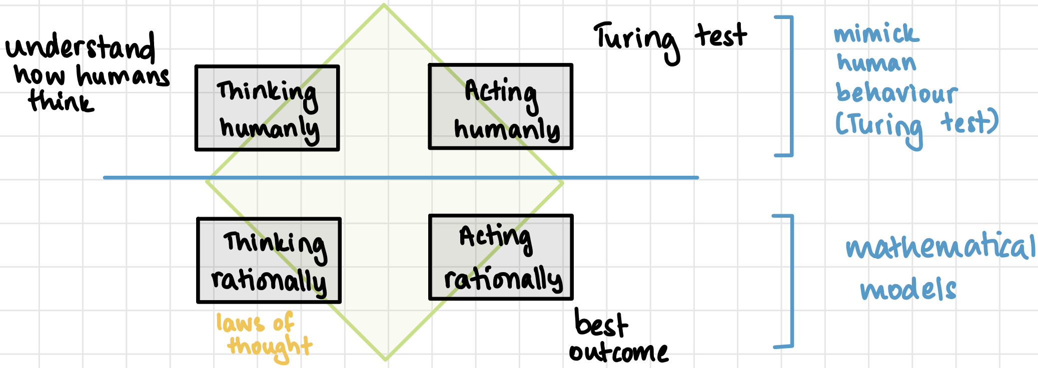
UNIT-1

Introduction, Search Algorithms, Classification with Decision Trees and Performance Metrics

feedback/corrections: vibha@pesu.pes.edu

- VIBHA MASTI

FOUR CATEGORIES VIEW OF AI



Levels of AI

(i) Narrow AI : performs specific task where machine can perform better than humans

(ii) General AI: AI at a general state where it can perform any intellectual task with the same level of accuracy as humans

(iii) Super AI: AI that always beats humans in tasks

AI vs Human Intelligence

COMPARISON FACTORS	ARTIFICIAL INTELLIGENCE	HUMAN INTELLIGENCE
NATURE	Aims to build machines that can mimic human behavior and perform human-like actions.	Aims to adapt to new environments by utilizing a combination of different cognitive processes,
FUNCTIONING	AI-powered machines rely on data and specific instructions fed into the system.	Use the brain's computing power, memory, and ability to think,
LEARNING POWER	Learn from data and through continuous training, but they can never achieve the thought process unique to humans	It is all about learning from various incidents and past experiences. It is about learning from mistakes made via trial-and-error approach throughout one's life.

PTE ML Model

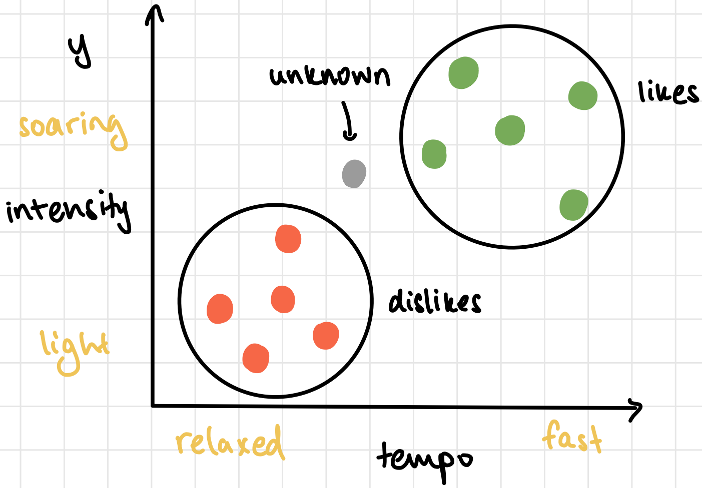
- <P, T, E> three tuple

- Task T
- Performance measure P
- Training experience E

- Example 1 — checkers learning program
  - T: playing checkers
  - P: percent of games won
  - E: playing practice games against itself

- Example 2 — handwriting recognition
  - T: recognising handwritten characters
  - P: percent of characters recognised correctly
  - E: identifying characters from a large database

Predict songs that a user will like



Q: Identify Task (T), Training Experience (E) and Performance Measure (P) for the following.

1. Learning to play checkers

T: Learning to play checkers
E: No. of games played against itself (practice)
P: No. of games won

2. Handwriting Recognition Learning Problem

T: Convert a handwriting image to text
E: Images of characters studied (database)
P: Accuraccy (% correct words)

3. Self-driven cars

T: To drive a car using only sensors
E: Sequence of images and driving / steering instructions
P: Avg. distance travelled before error is made

4. Text categorisation

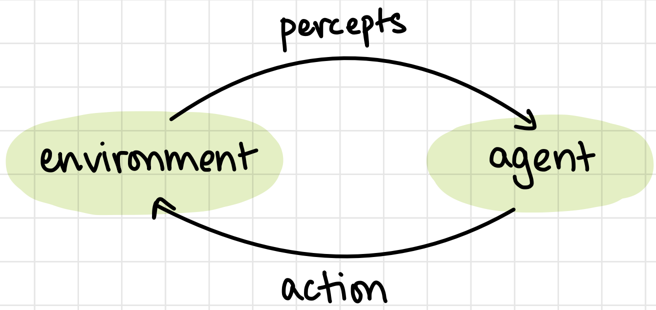
T: Assign document to a given category
E: Database of pre-classified document
P: Fraction of documents correctly tagged

AGENTS in AI

- Agent perceives the environment's state and takes actions based on the state

- Perceives through sensors

- Makes decisions using AI, using percept history and past actions



- Intelligent agents learn from environment and act upon it
  - Eg: AI assistants, chess bots

- Rational agents perform the tasks in the most optimal manner and has a clear preference (deterministic)
  - Eg: temperature sensors

PEAS Model for AI Agents

- <P, E, A, S> four tuple

- Performance measure P : unit of success
  Environment E : surroundings of agent
  Actuator A : delivers agent's output
  Sensor S : takes in input

- Textbook uses PAGE (goal) instead

- Example 1 — self-driving car
  - P: safety, speed, violations
  - E: roads, obstacles, signs
  - A: mechanical parts of car
  - S: IR sensors, camera

TYPES of ENVIRONMENTS

1. Observability
2. Determinism
3. Episodicity
4. Dynamism
5. Continuity

1. Observability

(i) Fully observable: agent's sensors give it access to the complete state of the environment
- Eg: crosswords, sudoku, 8-puzzle problem

(ii) Partially observable: entire state of environment not visible through sensors
- Eg: autonomous driving

2. Determinism

(i) Deterministic: current state of agent & action can determine next state
- Eg: state of chess board after making a move

(ii) Stochastic: random environment ; not deterministic
- Eg: poker

3. Episodicity

(i) Episodic: agent's experience divided into atomic episodes where actions in each episode depend only on the episode
- Eg: spam filter where each mail is an episode

errors in diff stages (episodes of assembly line) —defects

(ii) Sequential: next state dependent on current action
- Eg: chess moves in a single game

4. Dynamism

(i) Static: idle environment with no change in state between actions
- Eg: snakes and ladders

(ii) Dynamic: environment can change when the agent is deliberating
- Eg: driving, tennis

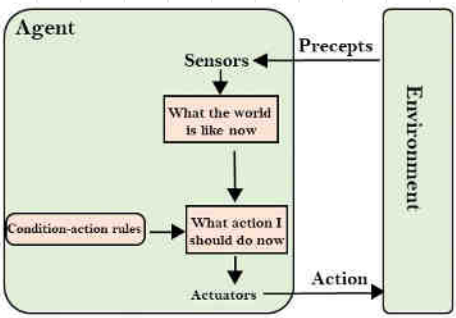
5. Continuity

(i) Discrete: environment has finite number of actions
- Eg: chess, snakes and ladders

(ii) Continuous: infinite number of actions
- Eg: tennis, driving

Classes of Intelligent Agents

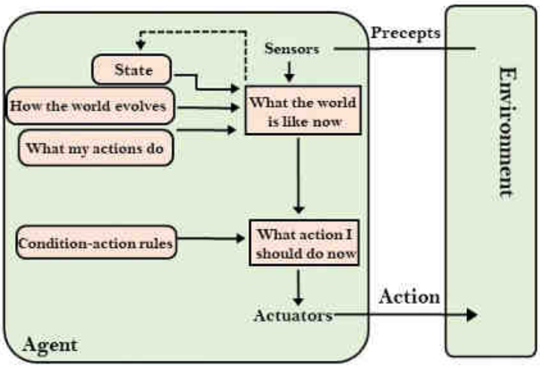
1. Simple reflex agents
- uses simple conditional statements to make decisions
- fully observable agents
- eg: temperature sensor, light sensor, metal detector



```python
def simple_reflex_agent(percept):
    state = get_state_from_percept(percept)
    rule = match_rule(state, rules)
    action = rule.Action
    return action
```

2. Model-based agents

- perceives and takes action based on experience (history)
- can work in partially observable environment
- eg: self-driving car



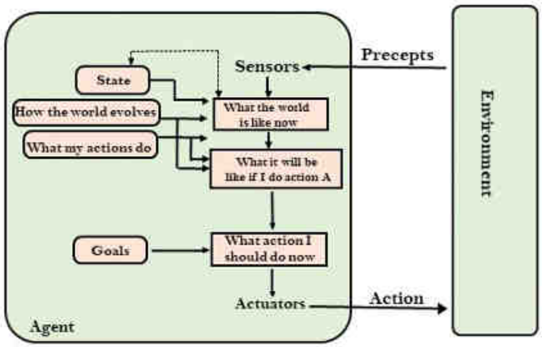

```python
def model_based_reflex_agent(percept):
    state = update_state(state, action, percept)
    rule = match_rule(state, rules)
    action = rule.Action
    return action
```

Simple Reflex Agent vs Model-Based Reflex Agent

- A **simple-reflex agent** selects actions based on the agent's current perception of the world and not based on past perceptions
- A **model-based-reflex agent** is made to deal with partial accessibility; they do this by keeping track of the part of the world it can see now. It does this by keeping an internal state that depends on what it has seen before so it holds information on the unobserved aspects of the current state.
- The former only base its analysis on current states while the latter takes account of **past events**

3. Goal-based agents

- experience & goal fed into agent
- eg: max shopping with min cost, Google's Waymo driverless

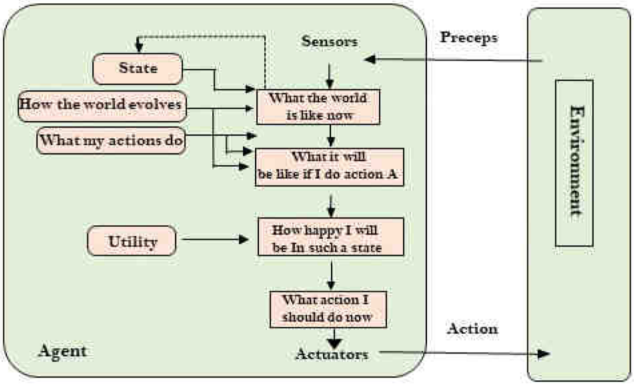


Pseudo code

```
function MODEL-GOAL-BASED-AGENT(percept) returns an action

    state: what the current agent sees as the world state
    model: a description detailing how the next state is a result of the
    current state and action.
    goals: a set of goals the agent needs to accomplish
    action: the action that most recently occurred and is initially null

    state = UPDATE-STATE(state, action, percept, model)
    action = BEST-ACTION(goals, state)
    return action
```

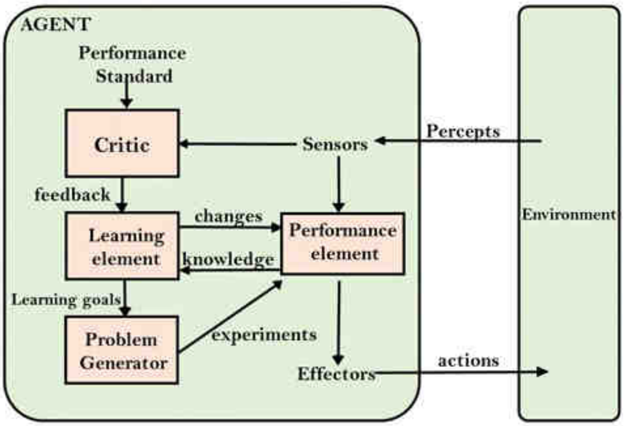
4. Utility agents

- if the actions taken to reach the goal make the user happy
- eg: route recommendation system that changes dynamically (if problems occur)



5. Learning agents

- critics give feedback to learning agents
- effectors instead of actuators
- problem generator suggests actions
- performance elements responsible for selecting external action
- eg: Google Assistant, computer vision, search engines



Applications of Learning Agents

- Agents in uncertain environments
- Humans are learing agents
- Search engines
- Computer vision
- Recognition of gestures

Search PROBLEMS

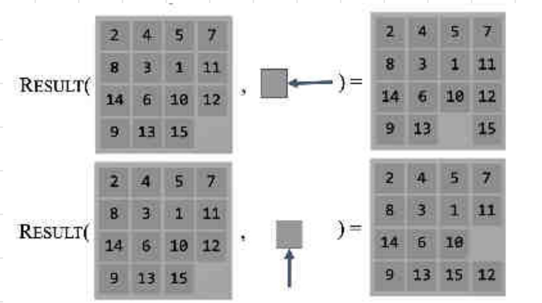
- Agent is given an initial state and a goal state
- Returns solution of how to get from initial state to goal state

1. Agent
    - entity that perceives environment and acts upon the environment
    - can be function (abstract mathematical description) or a program (concrete implementation)
2. State
    - configuration of the agent and its environment
3. Initial state
    - initial configuration of environment
4. Actions
    - choices that can be made in a state

- action(s) returns set of actions that can be executed in state s

5. Transition model

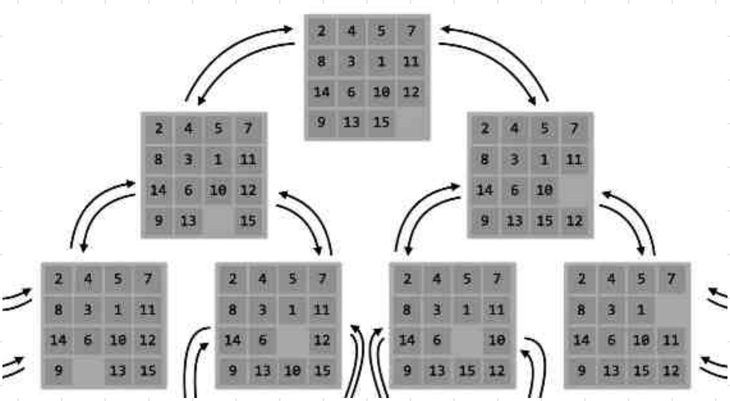
- $T(s,a) \rightarrow s'$



6. State Space

- set of all states reachable from initial state by any sequence of actions

- tree or graph



7. Goal test

- determine if given state is goal state

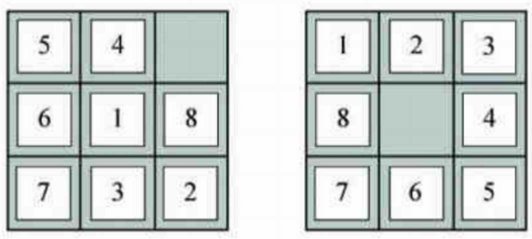
8. Path cost
- numerical cost associated with a given path

Formalising a Search Problem
1. Initial state
2. Action
3. Transition model
4. Goal test
5. Path cost

Frontiers
- Data structure that supports the task
- Stack or queue

Analyse

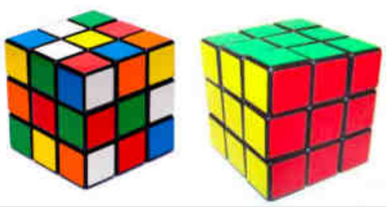
1. 8 Puzzle Problem (optimal solution – NP hard)



Start State Goal State

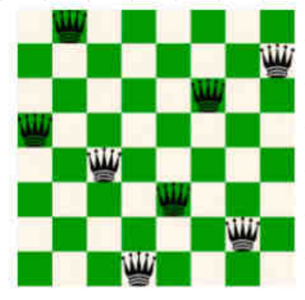
state: tile locations
initial state: specific tile config
actions: move blank tile left, right, up, down
goal test: tiles are in goal config
path cost: 1 per move

2. Rubik's Cube



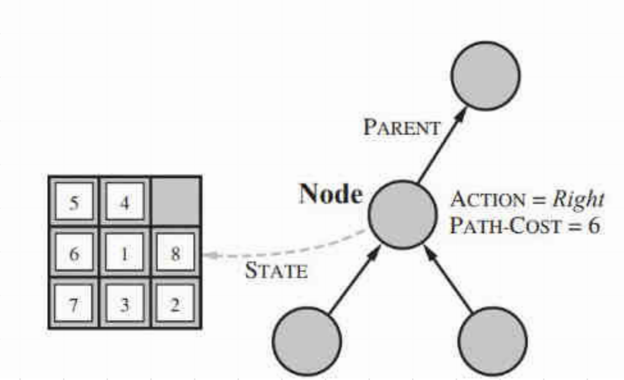
state: colours on each face
initial state: specific cube config
actions: rotate a column or a face
goal test: same colour on face
path cost: 1 per move

3. 8 Queens Problem



state: configuration of queens
initial state: empty board
actions: add a queen to the board
goal test: solution to 8-queens problem (no attacks)
path cost: time taken to solve

NODE in SEARCH TREE



- Each node: n

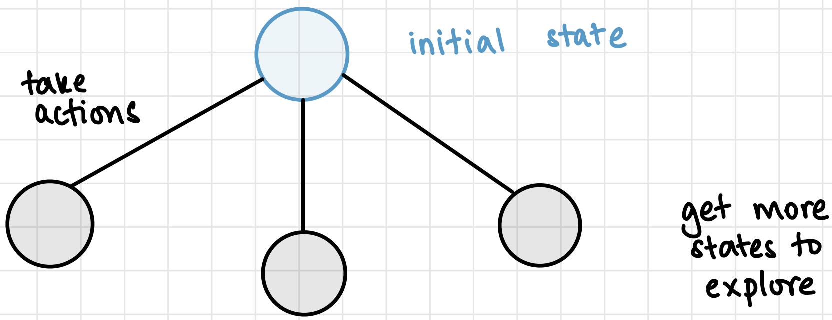
1. n. STATE
   - state in the state space

2. n. PARENT
   - node that generated n

3. n.ACTION
- action applied by parent to generate n

4. n.PATH-COST
- g(n)
- cost from initial state to the node

Search Strategies



Frontier

- Data structure to store states to be explored
- Could be stack, queue, priority queue
- Frontier initially stores only the initial state

Tree-Search (general algorithm)

```
function TREE-SEARCH(problem):
    frontier.add(problem.INITIAL_STATE)
    repeat:
        if the frontier is empty:
            return no solution
        else:
            node = frontier.remove_node()
            if node is a goal state:
                return the solution
            else:
                expand node, add neighbour nodes to the frontier
```

Problems with Tree Search

- No visited array / storage of history
- Solution: use explored set (closed set) that remembers every expanded node

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     add the node to the explored set
     expand the chosen node, adding the resulting nodes to the frontier
       only if not in the frontier or explored set
```

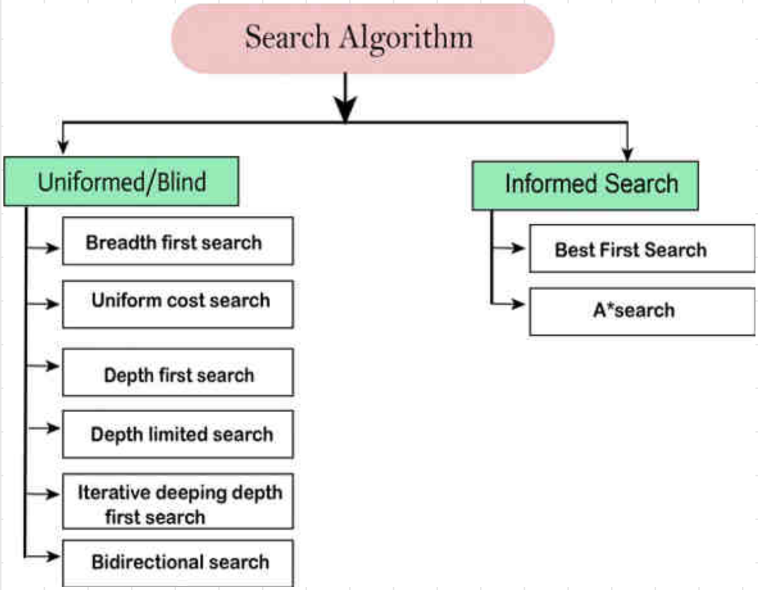
Parameters to Define a Good Strategy

1. Completeness — does it always find solution if it exists?
2. Time complexity — number of nodes generated
3. Space complexity — max no. of nodes in memory
4. Optimality — does it always find least cost solution?

* time & space complexity

max no. of children at each node

- b: maximum branching factor of search tree
- d: depth of least-cost solution
- m: maximum depth of state space (could be ∞)



1. Uninformed search

- blind search
- only use information available in the problem definition
- generate successors and distinguish goal state from non-goal state

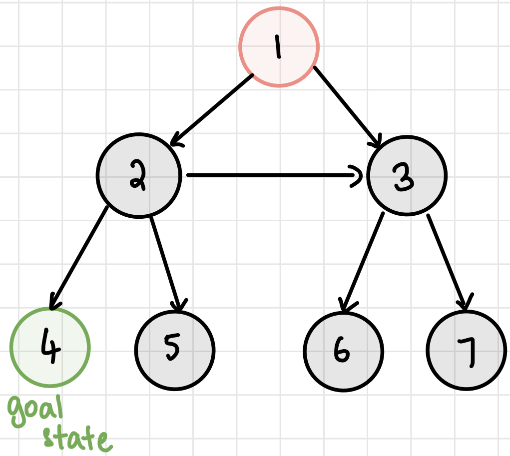
2. Informed search

- heuristic search
- know whether one goal state is better than another
- greedy best search first

Uninformed Search Strategy

1. Breadth First Search

- Frontier: queue
- Applications: P2P Networks, web crawlers, navigation systems, network broadcasting

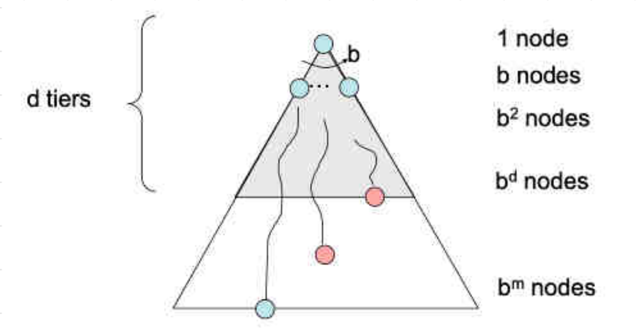


1. Completeness: yes (finite graphs)
2. Time Complexity: $O(b^d)$
3. Space Complexity: $O(b^d)$ and $O(b^{d-1})$
4. Optimality: no

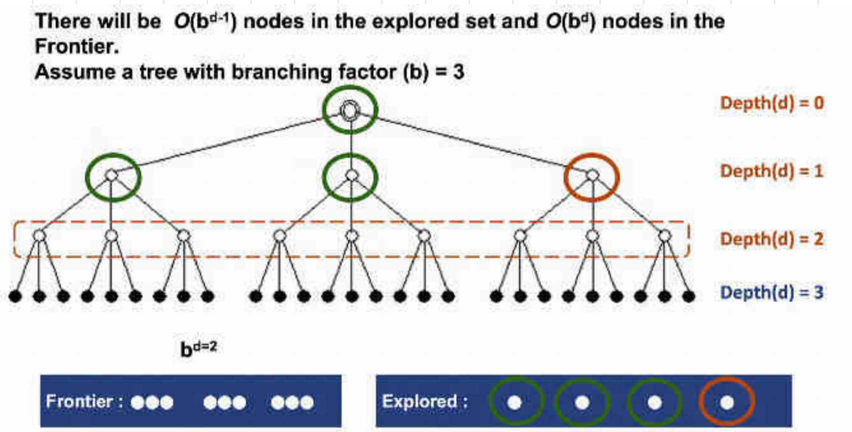
queue explored

```
function BFS(problem)
        node = a node with STATE = problem.INITIAL_STATE, PATH_COST = 0

        if node.STATE == problem.GOAL_STATE then
                return SOLUTION(node)
        end if

        frontier = FIFO queue with node as the only element
        explored = empty set

        while frontier is not empty:
                node = POP(frontier)
                add node.STATE to explored

                for all edges from node.STATE to neighbour in ADJ_EDGES(node.STATE) do
                        if neighbour.STATE not in explored and not in frontier then
                                if neighbour.STATE == problem.GOAL_STATE then
                                        return SOLUTION(neighbour)
                                end if
                                frontier.insert(neighbour)
                        end if
                end for
        end while
end function
```

Time Complexity



Space Complexity



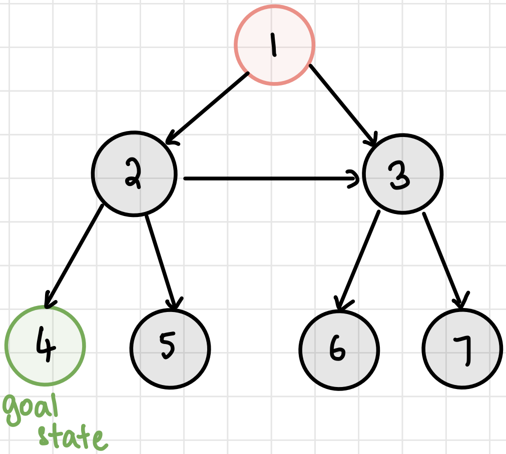
Useful

- Infinite paths
- Space available
- Bad when heuristic knowledge present

- P2P networks
- web crawlers
- nav systems
- net broadcasting

2. Depth First Search

- Frontier: Stack



Tree search (no explored set)

1. Completeness: no
2. Time Complexity: space state size
3. Space Complexity: O(bm)
4. Optimality: no

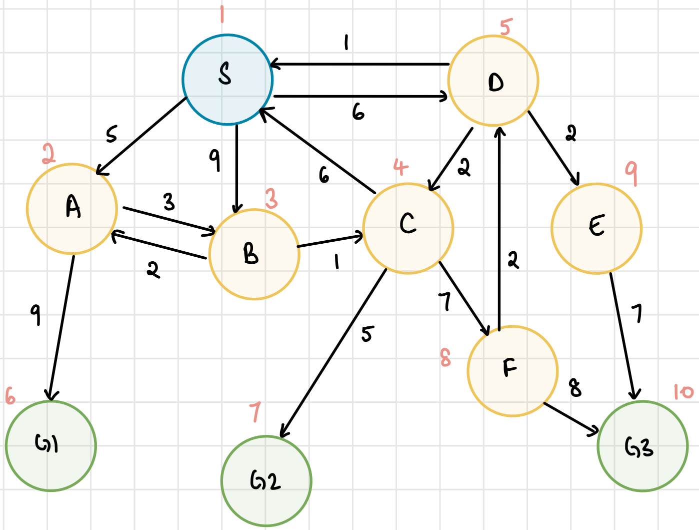
Graph search (explored set)

1. Completeness: yes
2. Time Complexity: $O(b^m)$
3. Space Complexity: $O(b^m)$
4. Optimality: no

```
function DFS(problem)
        node = a node with STATE = problem.INITIAL_STATE, PATH_COST = 0

        if node.STATE == problem.GOAL_STATE then
                return SOLUTION(node)
        end if

        frontier = stack with node as the only element
        explored = empty set

        while frontier is not empty:
                node = POP(frontier)
                add node.STATE to explored

                if node.STATE == problem.GOAL_STATE then
                        return SOLUTION(node)
                end if

                for all edges from node.STATE to neighbour in ADJ_EDGES(node.STATE) do
                        if neighbour.STATE not in explored and not in frontier then
                                frontier.insert(neighbour)
                        end if
                end for
        end while
end function
```

3. Uniform Cost Search

- Extension of BFS when graph is weighted
- Expands node n with lowest path cost g(n)
- Frontier: priority queue with key = g(n)
- Path to goal node with lowest cumulative cost
- Dijkstra's algorithm

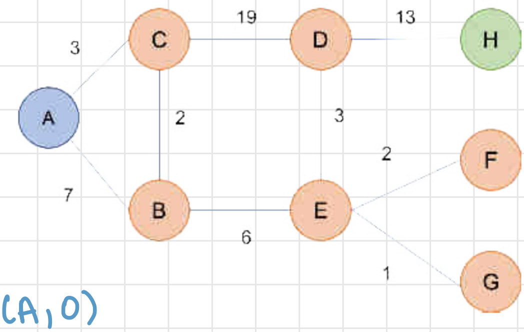


1. Completeness: yes
2. Time Complexity: $O(b^{1+[C^*/\varepsilon]})$
3. Space Complexity: $O(b^{1+[C^*/\varepsilon]})$
4. Optimality: yes

C: cost of optimal sol
ε: cost of each step that gets you closer to goal

```
function UNIFORM_COST_SEARCH(problem)
  node = a node with STATE = problem.INITIAL_STATE, PATH_COST = 0

  if node.STATE == problem.GOAL_STATE then
    return SOLUTION(node)
  end if

  frontier = priority queue with key as PATH_COST and node is the only element
  explored = empty set

  while frontier is not empty:
    node = POP(frontier)
    add node.STATE to explored

    if node.STATE == problem.GOAL_STATE then
      return SOLUTION(node)
    end if

    for all edges from node.STATE to neighbour in ADJ_EDGES(node.STATE) do
      if neighbour.STATE not in explored and not in frontier then
        frontier.insert(neighbour)
      else if (
        neighbour.STATE in frontier with PATH_COST higher than node.PATH_COST +
        problem.COST(node, neighbour)
      ) then
        frontier.replace_priority(neighbour, node.PATH_COST + problem.COST(node, neighbour))
      end if
    end for
  end while

  return FAILURE

end function
```

Q: Find optimal path cost to goal node H



1. visited:
queue: (A, 0)
distances: (A, 0), (B, ∞), (C, ∞), (D, ∞), (E, ∞), (F, ∞), (G, ∞), (H, ∞)
previous: (A, –), (B, –), (C, –), (D, –), (E, –), (F, –), (G, –), (H, –)

2. visited: A
queue: (C, 3), (B, 7)
distances: (A, 0), (B, 7), (C, 3), (D, ∞), (E, ∞), (F, ∞), (G, ∞), (H, ∞)
previous: (A, –), (B, A), (C, A), (D, –), (E, –), (F, –), (G, –), (H, –)

3. visited: A, C
queue: (B, 5), (D, 22)
distances: (A, 0), (B, 5), (C, 3), (D, 22), (E, ∞), (F, ∞), (G, ∞), (H, ∞)
previous: (A, –), (B, C), (C, A), (D, C), (E, –), (F, –), (G, –), (H, –)

4. visited: A, C, B
queue: (E, 11), (D, 22)
distances: (A, 0), (B, 5), (C, 3), (D, 22), (E, 11), (F, ∞), (G, ∞), (H, ∞)
previous: (A, –), (B, C), (C, A), (D, C), (E, B), (F, –), (G, –), (H, –)

5. visited: A, C, B, E
queue: (G, 12), (F 13), (D, 14)
distances: (A, 0), (B, 5), (C, 3), (D, 14), (E, 11)
(F, 13), (G, 12), (H, ∞)
previous: (A, –), (B, C), (C, A), (D, E), (E, B),
(F, E), (G, E), (H, –)

6. visited: A, C, B, E, G
queue: (F 13), (D, 14)
distances: (A, 0), (B, 5), (C, 3), (D, 14), (E, 11)
(F, 13), (G, 12), (H, ∞)
previous: (A, –), (B, C), (C, A), (D, E), (E, B),
(F, E), (G, E), (H, –)

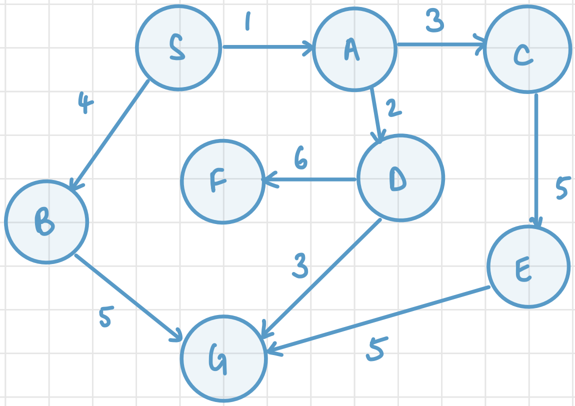
7. visited: A, C, B, E, G, F
queue: (D, 14)
distances: (A, 0), (B, 5), (C, 3), (D, 14), (E, 11)
(F, 13), (G, 12), (H, ∞)
previous: (A, –), (B, C), (C, A), (D, E), (E, B),
(F, E), (G, E), (H, –)

8. visited: A, C, B, E, G, F, D
queue: (H, 27)
distances: (A, 0), (B, 5), (C, 3), (D, 14), (E, 11)
(F, 13), (G, 12), (H, 27)
previous: (A, –), (B, C), (C, A), (D, E), (E, B),
(F, E), (G, E), (H, D)

Optimal path: A → C → B → E → D → H

Optimal cost: 27

Q: Find optimal path cost to goal G from S



prev: (S, None), (A, S), (B, S)
dist: (S, 0), (A, 1), (B, 4)
PQ: (A, 1), (B, 4)
vis: S

prev: (S, None), (A, S), (B, S), (D, A), (C, A)
dist: (S, 0), (A, 1), (B, 4), (D, 3), (C, 4)
PQ: (D, 3), (B, 4), (C, 4)
vis: S, A

prev: (S, None), (A, S), (B, S), (D, A), (C, A),
(F, D), (G, D)
dist: (S, 0), (A, 1), (B, 4), (D, 3,), (C, 4),
(F, 9), (G, 6)
PQ: (B, 4), (C, 4), (G, 6), (F, 9)
vis: S, A, D

prev: (S, None), (A, S), (B, S), (D, A), (C, A),
(F, D), (G, D),
dist: (S, 0), (A, 1), (B, 4), (D, 3), (C, 4),
(F, 9), (G, 6)
PQ: (C, 4), (G, 6), (F, 9)
vis: S, A, D, B

prev: (S, None), (A, S), (B, S), (D, A), (C, A),
(F, D), (G, D),
dist: (S, 0), (A, 1), (B, 4), (D, 3), (C, 4),
(F, 9), (G, 6), (E, 9)
PQ: (G, 6), (F, 9), (E, 9)
vis: S, A, D, B, C

prev: (S, None), (A, S), (B, S), (D, A), (C, A),
(F, D), (G, D),
dist: (S, 0), (A, 1), (B, 4), (D, 3), (C, 4),
(F, 9), (G, 6), (E, 9)
PQ: (F, 9), (E, 9)
vis: S, A, D, B, C, G

Cost to goal G = 6

Path : S → A → D → G

Informed Search Strategy

- Search algorithm with information on the goal state that helps in efficient searching

- Information in function $f(n)$ — estimates how close the node is to a goal state — gives out a positive number

HEURISTIC FUNCTION $h(n)$

- $h(n)$ is an estimated cost of the cheapest path from the node n to a goal state

- If $h^*(n)$ is the actual cheapest cost of the path from node n to a goal state, then

$$h(n) \leq h^*(n)$$

- In other words, $h(n)$ can never overestimate the cost to the goal nodes

- If n is a goal node, $h(n) = 0$

- Heuristic function must be designed/ chosen smartly; eg: Euclidean distance

- Two algorithms for us to study

1. Best First Search

- Improved version of UCS

- Greedy strategy (no backtracking; irrevocable)

- Greedy fails in giving us the optimal solution

Arad → Bucharest

(T1)

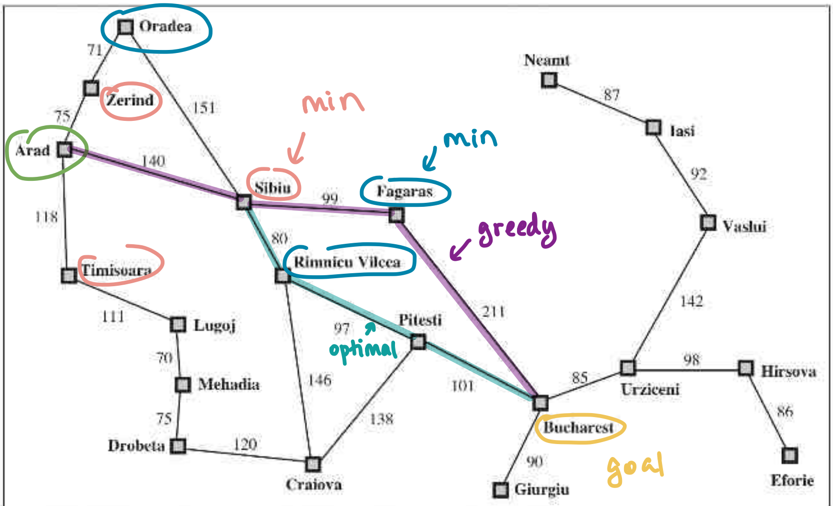


Figure 3.2 A simplified road map of part of Romania.

start goal min min

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

- Path chosen is not optimal and can be incomplete if there is no check for infinite loops

- $O(b^m)$ space & time

Iasi → Fagaras (tree search - no visited)

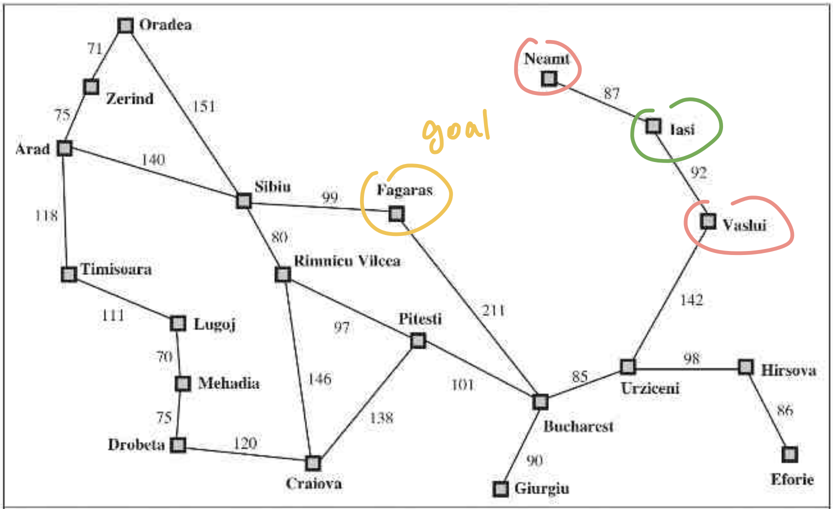


Figure 3.2 A simplified road map of part of Romania.

- Stuck in a loop (Iasi → Neamt → Iasi ···)
- Incomplete in finite space

Iasi → Fagaras

- Complete in finite state space
- Incomplete in infinite

1. Completeness: no
2. Time Complexity: $O(b^m)$ — improvement with good h(n)
3. Space Complexity: $O(b^m)$ — all nodes in memory for h(n)
4. Optimality: no

2. A* Search

- Heuristic function gives estimate of minimum cost between the node and a goal state

- A* algorithm combines the Heuristic function $h(n)$ with the actual cost from start node to the node n, called $g(n)$ to select the next node to travel to — $f(n)$

$$f(n) = g(n) + h(n)$$

- Enhancement to Best First Search to attain optimal solution

Conditions for Optimality

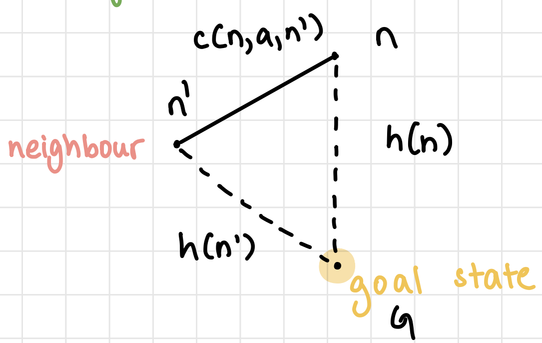
1. Admissibility

- $h(n)$ should be an admissible heuristic; it never overestimates the path cost

- $\therefore$ $f(n)$ never overestimates path cost

- eg: h_{SLD} - straight line distance

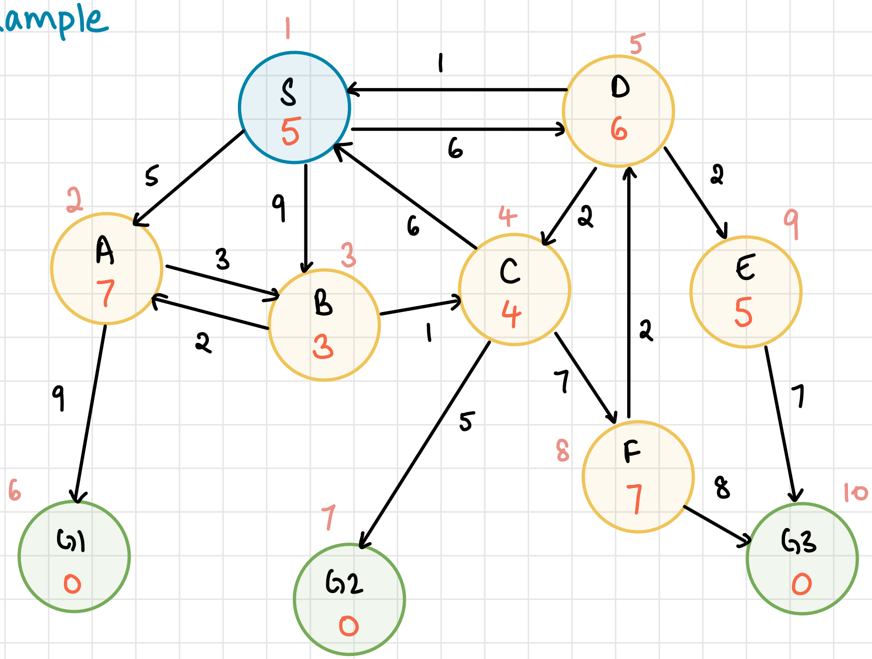
2. Consistency / Monotone



(triangle inequality)

- $h(n) \leq c(n, a, n') + h(n')$

Example



1. Priority queue :

S 5	

Prevs :

S	A	B	C	D	E	F	G1	G2	G3

2. Priority queue :

A 12	B 12	D 12	

Visited:

S

Prevs :

S	A	B	C	D	E	F	G1	G2	G3
	S	S		S					

3. Priority queue :

B	D	G1	
11	12	14	

visited:

S, A

Prevs :

S	A	B	C	D	E	F	G1	G2	G3
	S	A		S			A		

4. Priority queue :

D	C	G1	
12	13	14	

visited:

S, A, B

Prevs :

S	A	B	C	D	E	F	G1	G2	G3
	S	A	B	S			A		

5. Priority queue :

C	E	G1	
12	13	14	

visited:

S, A, B, D

Prevs :

S	A	B	C	D	E	F	G1	G2	G3
	S	A	D	S	D		A		

6. Priority queue :

E	G2	G1	F	
13	13	14	22	

visited:

S, A, B, D, C

Prevs :

S	A	B	C	D	E	F	G1	G2	G3
	S	A	D	S	D	C	A	C	

7. Priority queue :

G2	G1	G3	F	
13	14	15	22	

visited:

S, A, B, D, C, E

Prevs :

S	A	B	C	D	E	F	G1	G2	G3
	S	A	D	S	D	C	A	C	E

8. Priority queue :

G1	G3	F		
14	15	22		

visited:

S, A, B, D, C, E, G2

Prevs :

S	A	B	C	D	E	F	G1	G2	G3
	S	A	D	S	D	C	A	C	

Optimal path: S → D → C → G2

Heuristic Function Design

- A good Heuristic function is crucial in determining efficiency of A* algorithm
- 8-puzzle: a bad Heuristic is no. of displaced tiles (tiles not on final tile spot) ; maxes at 8
- Manhattan distance: sum of distances from each tile to its final state
  - Eg: 8- puzzle

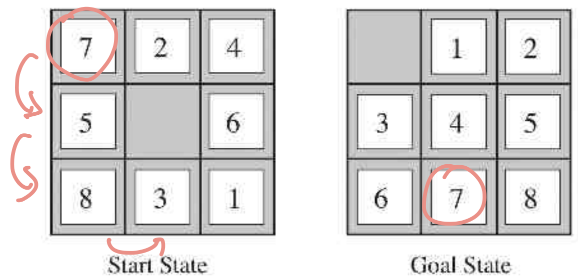


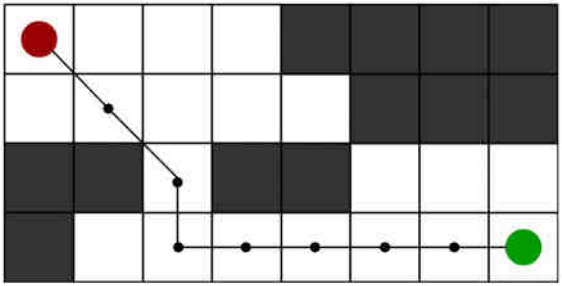
Figure 3.28 A typical instance of the 8-puzzle. The solution is 26 steps long.

$h(1) = 3$
$h(2) = 1$
$h(3) = 2$
$h(4) = 2$
$h(5) = 2$
$h(6) = 3$
$h(7) = 3$
$h(8) = 2$

$h(S) = \sum_i h(i) = 18$

- Must take into account effort involved in calculating h(n)

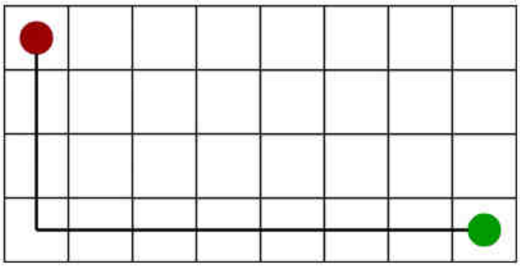
- Eg: find path from start state to goal state in the given maze



source: geeks for geeks

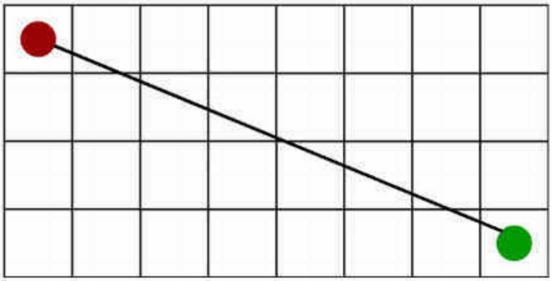
Manhattan Distance

```
h = abs (current_cell.x – goal.x) + abs (current_cell.y – goal.y)
```

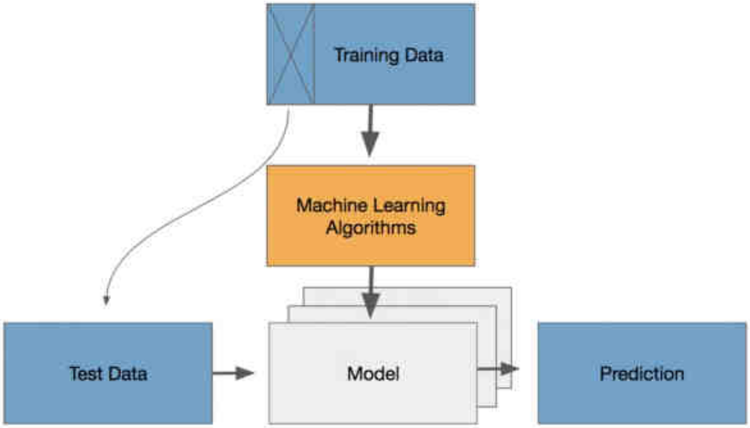


Euclidean Distance

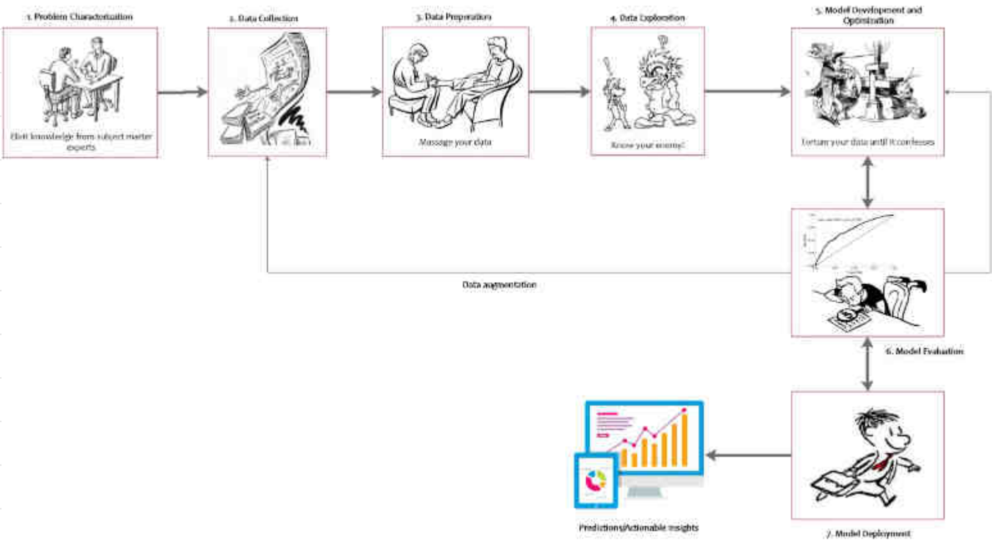
```
h = sqrt((current_cell.x - goal.x)^2 + (current_cell.y - goal.y)^2)
```



Machine Learning

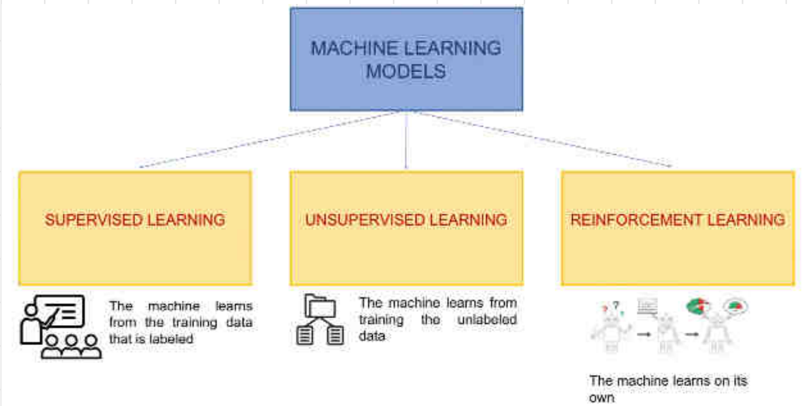


7-Step Procedure of Machine Learning



source: towards data science

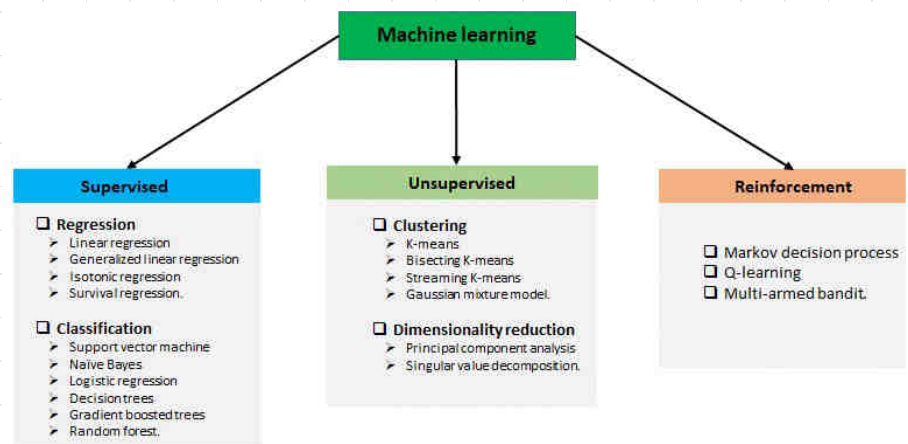
Machine Learning Models



1. Supervised Learning
   - Classification algorithms (kNN is supervised)
   - Regression

2. Unsupervised Learning
   - Clustering algorithms

3. Reinforcement Learning
   - Policy-based or value-based
   - Agent-environment interface



COMPARISON TABLE

https://www.aitude.com/supervised-vs-unsupervised-vs-reinforcement/

Comparison Table

Criteria	Supervised ML	Unsupervised ML	Reinforcement ML
Definition	Learns by using labelled data	Trained using unlabelled data without any guidance.	Works on interacting with the environment
Type of data	Labelled data	Unlabelled data	No – predefined data
Type of problems	Regression and classification	Association and Clustering	Exploitation or Exploration
Supervision	Extra supervision	No supervision	No supervision
Algorithms	Linear Regression, Logistic Regression, SVM, KNN etc.	K – Means, C – Means, Apriori	Q – Learning, SARSA
Aim	Calculate outcomes	Discover underlying patterns	Learn a series of action
Application	Risk Evaluation, Forecast Sales	Recommendation System, Anomaly Detection	Self Driving Cars, Gaming, Healthcare

Concept learning

- Attributes describe a concept
- Use data to teach a machine to solve binary classification problem

Attribute A_1 — circle: 1, oval: 0

Attribute A_2 — dark: 1, light: 0

Concept C – yes/no

Possible instances $= 2 \times 2 = 4 = |A_1| \times |A_2|$

Attribute A_1	Attribute A_2
0	0
0	1
1	0
1	1

instance space = 4

concept

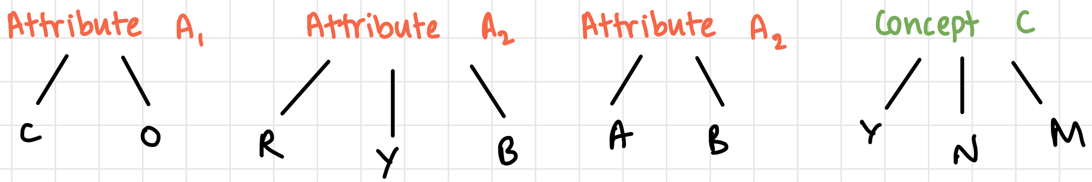
Concept Space – Power Set

00 — yes / no
01 ⋮
10 ⋮
11 ⋮

$2^{2\times 2} = 16$

$= C^{|A_1| \times |A_2|}$

More Attributes & Concepts



Instance space $= 2 \times 3 \times 2 = 12$

concept space $= 3^{12} = 531441 \approx 500{,}000$

- Grows exponentially
- Reduce the part of concept space to define and train the model
- Use logical operators to club attributes together and eliminate concepts
- Inductive bias: assumptions made to reduce concept space (conjunctive concepts)
- Hypothesis: can be represented syntactically or semantically

INDUCTIVE BIAS

- Fundamental set of assumptions that the learner makes about the target function
- Allows learner to generalise beyond training data

Find S Algorithm

- Finds most specific hypothesis that fits all positive samples
- Considers only positive training examples
- starts with most specific hypothesis and generalises this hypothesis each time it fails to classify positive training data (using logical AND)
- Assumes binary attributes
- Don't cares (?) introduced when two instances with opposing attributes found (accept all)
- Assume initial hypothesis is $<\phi \wedge \phi \wedge \ldots \wedge \phi>$ for all attributes (most specific)
- The most general hypothesis is $<? \wedge ? \wedge \ldots \wedge ?>$
- ϕ indicates no value is acceptable (reject all)

Example: Find hypothesis using Find S Algorithm

Example	Citations	Size	in Library	Price	Editions	Buy
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expensive	few	no
4	many	medium	no	expensive	many	yes
5	many	small	no	affordable	many	yes

(i) Initial instance space $= 2 \times 3 \times 2 \times 2 \times 3 = 72$

(ii) Initial concept space $= 2^{72} = 4.7 \times 10^{21}$

1. Start with specific hypothesis $H = \langle \phi, \phi, \phi, \phi, \phi \rangle$
2. Ignore eg #1
3. $H = $ <many, big, no, expensive, one>
4. Ignore eg #3
5. $H = $ <many, ?, no, expensive, ?>
6. $H = $ <many, ?, no, ?, ?>

(iii) New concept space

- 3 possible values for every attribute (two binary and one don't care – ?) + 1 initial reject-all hypothesis $\langle \phi, \phi \ldots \phi \rangle$
- Concept space $= 3^5 + 1 = 243 + 1 = 244$ (semantic)
- Concept space reduced from 4.7×10^{21} to 243

HYPOTHESIS SPACE

- Conjuctive concept space / shrunken concept space is called hypothesis space
- Syntactically distinct HS: add 2 wild card possibilities for each attribute — ?, ϕ
  - however, if one attribute is ϕ, the whole hyp is ϕ
  - semantically distinct HS needed
- Semantically distinct HS: add 1 wild card — ? — and one separate empty set $\langle \phi, \ldots, \phi \rangle$

VERSION SPACE

- Hypothesis is said to be consistent with respect to the training dataset if it correctly classifies all training examples

- For a hypothesis h and a point x_i in D_{train} with a true classification of $c(x_i)$,

$$h(x_i) = c(x_i) \quad \forall \quad x_i \in D_{train}$$

- Version space VS is a subset of Hypothesis space H such that VS contains all the hypotheses consistent with D_{train}

$$VS = \{h : h \in H \text{ and } h \text{ is consistent with } D_{train}\}$$

Limitations of Find S

- No way to determine if hypothesis is consisent throughout training examples

- Once ? introduced, all further info lost on that attribute

- No negative examples are taken into account

Candidate Elimination - Not in Syllabus

- Consider both + and - samples

- Two hypotheses: General (G) and Specific (S)

- $G = ? \wedge ? \wedge \cdots \wedge ?$
- $S = \phi \wedge \phi \wedge \ldots \wedge \phi$
- For every + training example, modify the specific hypothesis S (just like in Find S) by making it more general
- For every − training example, modify the general hypothesis G to make it more specific

---

Algorithm 1: Candidate Elimination Algorithm

Data: D: a dataset of objects labeled as positive or negative
Result: V: the version space of hypotheses consistent with D
Initialize G to the set containing the most general hypothesis
Initialize S to the set containing the most specific hypothesis
for *each object* $x \in D$ **do**
  if *x is a positive object* **then**
    Remove from G any hypothesis inconsistent with x
    for *each hypothesis $s \in S$ that is inconsistent with x* **do**
      Remove s from S
      Add to S all the minimal generalizations h of s such that h is consistent with x and for some member g of G it holds that $g \geq h$
      Remove from S any hypothesis that's more general than another hypothesis in S
    end
  end
  else
    Remove from S any hypothesis inconsistent with x
    for *each hypothesis $g \in G$ that is inconsistent with x* **do**
      Remove g from G
      Add to G all the minimal specializations h of g such that h is consistent with x and for some member s of S it holds that $h \geq s$
      Remove from G any hypothesis that's less general than another hypothesis in G
    end
  end
end
return V as $V(G, S)$

Performance metrics

Confusion Matrix

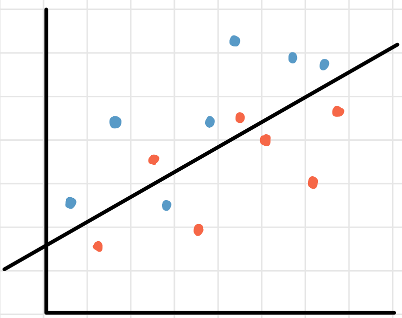
- Binary classification model : 2x2 matrix

Predictions

Actuals	+ve	-ve
+ve	true +ve	false -ve
-ve	false +ve	true -ve

Example

Find # of TP, TN, FP, FN cases where blue is +ve and red is -ve



Prediction

Data	+ve	-ve
+ve	TP 6	FN 1
-ve	FP 2	TN 5

- FP: type 1 error
- FN: type 2 error

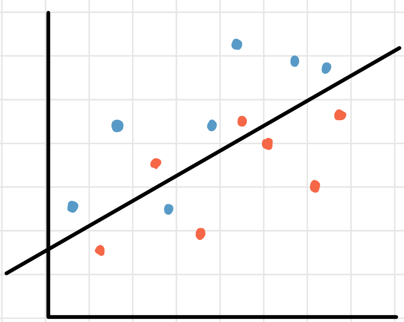
Accuracy

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Accuracy is a good measure when the target variable classes are neatly balanced
- If samples are majorly leaning towards one side (eg: 99% of emails received are spam), the model might always predict one outcome while retaining a high accuracy

Example

Find accuracy where blue is +ve and red is -ve



Prediction

Data	+ve	-ve
+ve	TP 6	FN 1
-ve	FP 2	TN 5

$$\text{accuracy} = \frac{6+5}{6+5+2+1} = \frac{11}{14}$$

Precision

- Correct positive cases out of predicted positive cases

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

Recall

- How many +ve cases caught (sensitivity) ; not missed
- True positive rate

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

- Model that always predicts +ve has a recall of 100% even though it is not a good model

Specificity

- How many -ve cases caught ; not missed

$$\text{specificity} = \frac{TN}{TN + FP} = 1 - FPR$$

F1 Score

- Harmonic mean of precision and recall

$$\text{F1 score} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$$

- Higher score → better (0 is worst, 1 is best)

- Only if precision and recall are 100%, F1 = 1

- Harmonic mean

$$\mu = \frac{2xy}{x+y}$$

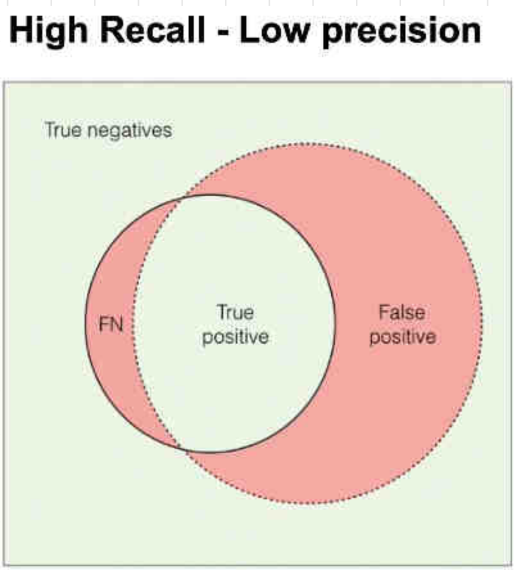
$$\frac{1}{\mu} = \frac{x+y}{2xy} = \frac{1}{2}\left(\frac{1}{x} + \frac{1}{y}\right)$$

- Geometric mean

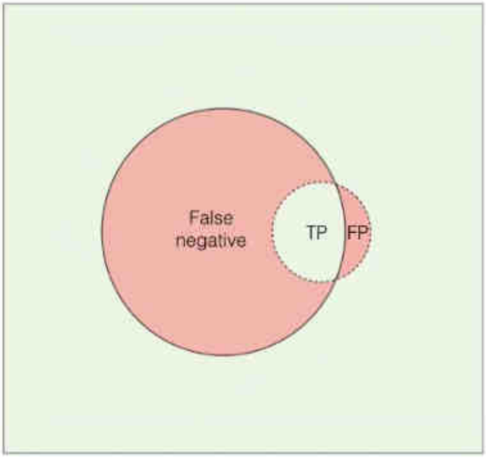
$$\mu = \sqrt{xy}$$

$$\mu^2 = xy$$

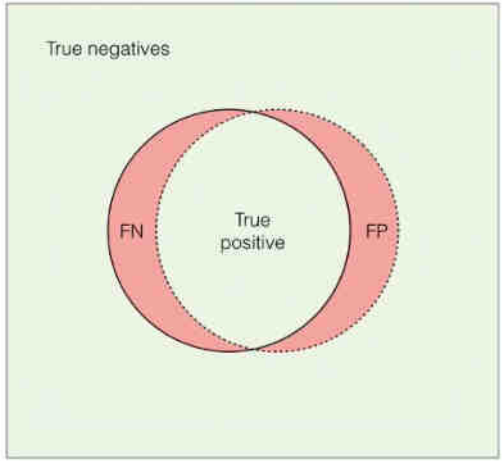
Recall & Precision



Low Recall - High precision



High Recall - High precision



Q: Find precision, recall, F1 score

		Prediction	
		C(+)	NC(-)
Actual	C(+)	3	97
	NC(-)	0	0

$$P = \frac{TP}{TP+FP} = \frac{3}{3} = 100\%$$

$$R = \frac{TP}{TP+FN} = \frac{3}{3+97} = 3\%$$

$$F1 \text{ score} = \frac{2 \times 1 \times 0.03}{1.03} = 5.83\%$$

Q: Find recall : how many patients diagnosed as sick correctly?

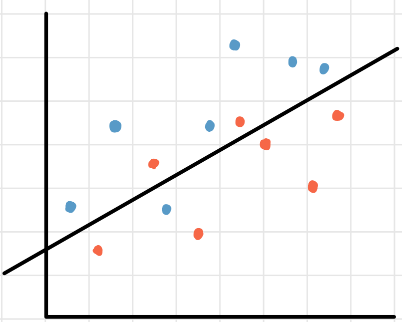
		Diagnosis	
		Sick	Healthy
Patients	Sick	1000	200
	Healthy	800	8000

TP = 1000 FP = 800

TN = 8000 FN = 200

$$\text{Recall} = \text{true +ve rate} = \frac{TP}{TP+FN} = 83.33\%$$

Q: Find recall where blue is +ve and red is -ve



		Prediction	
		+ve	-ve
Data	+ve	TP 6	FN 1
	-ve	FP 2	TN 5

$$\text{recall} = \frac{6}{7} = 85.71\%$$

Confusion Matrix

		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

Multi-Class Confusion Matrix

- Diagonal: true +ve for each of the classes

		Predicted: A	B	C	
True labels	A	2	2	0	4
	B	1	2	0	3
	C	0	0	3	3
		3	4	3	Total

Example

- For fish : true +ve = 24
  false +ve = 2+2+0 = 4
  false -ve = 2+1+2 = 5

Actual \ Predicted	Elephant	Monkey	Fish	Lion
Elephant	25	3	0	2
Monkey	3	53	2	3
Fish	2	1	24	2
Lion	1	0	2	71

Example

		Predicted		
		Husky	Labrador	Bulldog
Actual	Husky	P(H,H)	P(L,H)	P(B,H)
	Labrador	P(H,L)	P(L,L)	P(B,L)
	Bulldog	P(H,B)	P(L,B)	P(B,B)

Accuracy for Multi-Class

correct predictions

$$\text{Accuracy} = \frac{P(H,H) + P(L,L) + P(B,B)}{(P(H,H) + P(L,H) + P(B,H) + P(H, L) + P(L,L) + P(B,L) + P(H, B) + P(L,B) + P(B,B))}$$

all predictions

Precision for Multi-class

$$P = \frac{TP}{TP + FP}$$

Recall for Multi-Class

$$R = \frac{TP}{TP + FN}$$

Q: Find all metrics for the confusion matrix (D is +ve)

Actual values →

Predicted values ↓

	A	B	C	D
A	100	0	0	0
B	80	9	1	1
C	10	0	8	0
D	10	1	1	9

$$\text{Accuracy} = \frac{126}{230} = 54.78\%$$

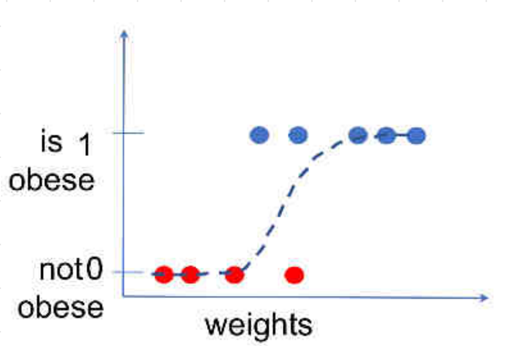
$$P = \frac{TP}{TP+FP} = \frac{9}{9+10+1+1} = 42.86\%$$

$$R = \frac{TP}{TP+FN} = \frac{9}{9+1} = 90\%$$

$$F1\ \text{score} = \frac{2 \times 0.9 \times 0.4286}{0.9 + 0.4286} = 58.07\%$$

ROC - Receiver Operating Characteristics

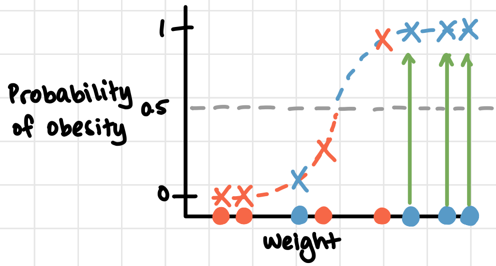
- Fit logistic regression curve to data (sigmoid)
- Blue: obese
  Red: not obese



- For different threshold values, values of TP, FP, FN, TN will vary

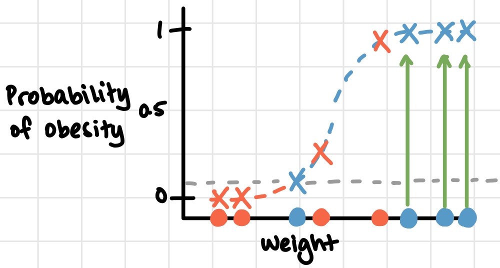
- Plot the values against threshold

(i) Threshold = 0.5



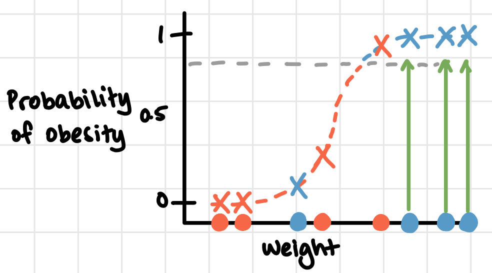
		Actual: obese	Actual: not obese
Predicted	obese	3	1
	not obese	1	3

(ii) Threshold = 0.1



		Actual: obese	Actual: not obese
Predicted	obese	4	2
	not obese	0	2

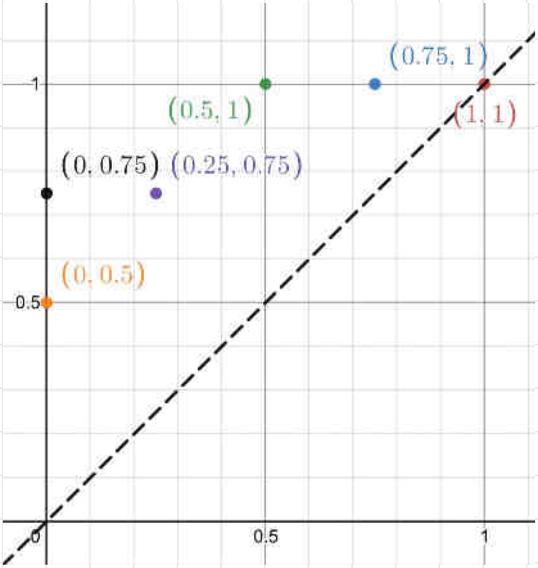
(iii) Threshold = 0.9



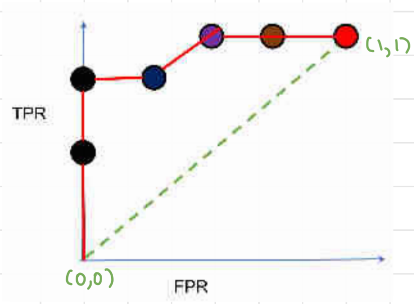
		Actual: obese	Actual: not obese
Predicted	obese	3	0
	not obese	1	4

TPR and FPR

Threshold	FPR	TPR
0 (all sample classified obese)	1	1
0.3	0.75	1
0.4	0.5	1
0.6	0.25	0.75
0.7	0	0.75
0.9	0	0.5



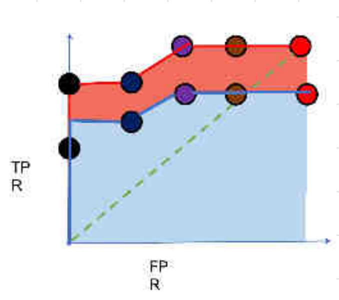
- $x=y$ line: worst performance of fully random sample
- If curve falls below green line, more false positives than random and such a classifier will not work



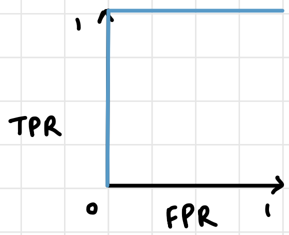
- Lowering the threshold classifies more items as positive, increasing both FP and TP
- Points should be closer to (1,0) → TPR = 1 and FPR = 0 (no false positives)
- Evaluating logistic regression model with different thresholds to compute points in ROC curve is inefficient

AUC- Area Under the ROC Curve

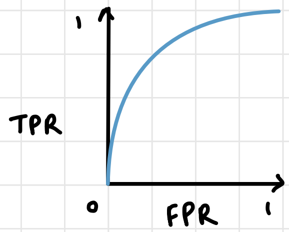
- If AUC higher, curve is better
- Here, red area > blue area ⇒ red curve is better



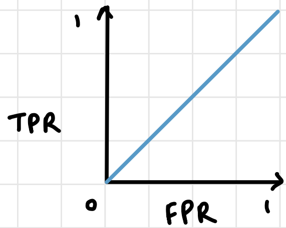
(i) Perfect (AUC=1)



(ii) Acceptable (0.5 < AUC < 1)



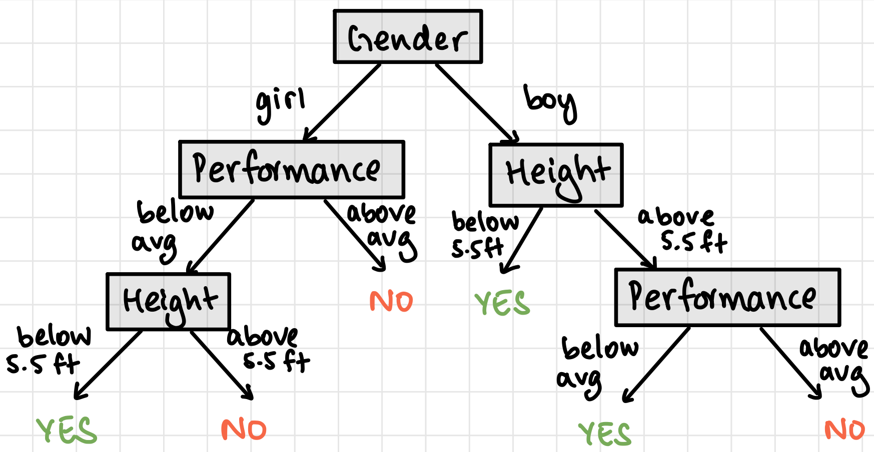
(iii) Predicting Random Classes



Decision Trees

Example:

- Class of 40 students, the following trend is observed
- Girls of height < 5.5 ft, whose performance in class tests is above average play cricket
- Girls of height > 5.5 ft, whose performance in class tests is above average do not play cricket
- Girls < average do not play cricket
- Boys < 5.5 ft play cricket
- Boys > 5.5 ft, whose performance is below average play cricket
- Boys > 5.5 ft, whose performance is above average do not play cricket



is this optimal?

how to find the shortest decision tree?

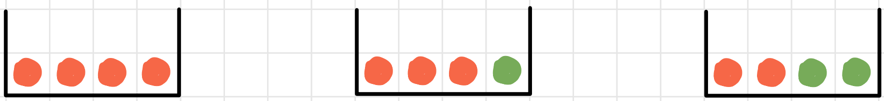
Real-Life Applications

1. Churn Analysis
2. Sentiment Analysis
3. Classification / Regression Models

Decision Tree

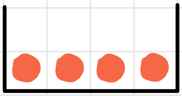
- Learning method for approximating discrete-valued target function in which the learnt function is represented by a decision tree
- Each internal node and root node tests for an attribute
- Disjunction of conjunctions (or of ands)
- Unsupervised learning technique

ENTROPY

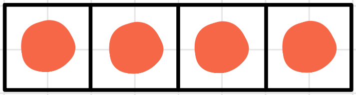


less entropy	medium entropy	high entropy
100% certain that red ball is picked	75 % certain that red ball is picked	25 % certain that red ball is picked

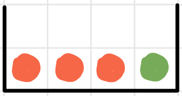
Game: Pick out balls in a particular sequence one-by-one with replacing



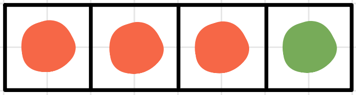
BUCKET 1



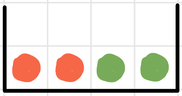
$1 \times 1 \times 1 \times 1 = 1$



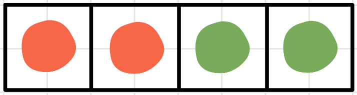
BUCKET 2



$0.75 \times 0.75 \times 0.75 \times 0.25 = 0.105$



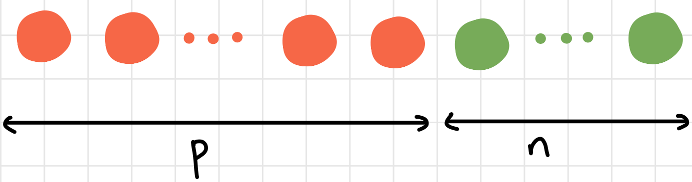
BUCKET 3



$0.5 \times 0.5 \times 0.5 \times 0.5 = 0.0625$

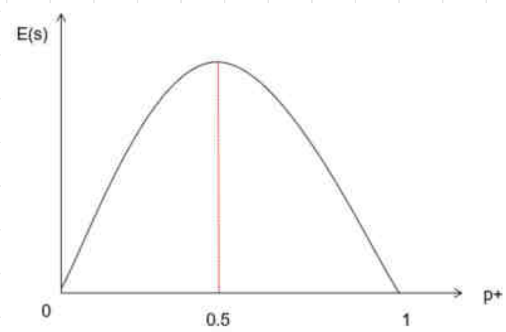
Turning Products to Sum

$$\log(ab) = \log a + \log b$$



$$\text{Entropy} = E(S) = \frac{-p}{p+n} \log_2\left(\frac{p}{p+n}\right) + \frac{-n}{p+n} \log_2\left(\frac{n}{p+n}\right)$$

For Binary classification



ID3 Algorithm

Cubics, CART- Gini index
$C_{4.5}$ (gain ratio), candidate elimination

- Construct decision trees top-down (invented by Ross Quinlan)
- Iterative Dichotomiser 3 (1984)
- Decide which attribute should be tested for at the tree's root
- Descendant of root node created for every possible value of root attribute and entire process repeated
- Greedy search for acceptable decision tree (no backtracking)
- Define statistical property information gain that measures how well a given attribute separates training examples according to their target classification
- Information gain calculates the reduction in the entropy and measures how well a given feature classifies the target class
- The feature with the highest information gain is selected as the best one for that level

ENTROPY QUANTIFIED

- For binary classification (target column has only 2 classes), entropy is 0 when all the values in the target column are homogeneous and 1 when there are equal number of values for both classes

- Entropy of dataset S

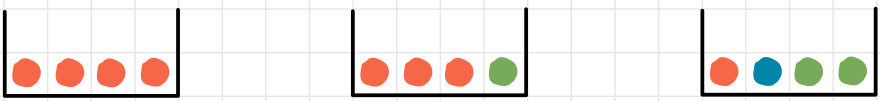
$$\text{Entropy}(S) = -\sum_{i=1}^{c} p_i * \log_2(p_i) = H(S)$$

- c: total number of classes in target column
- p_i: probability of class i — ratio of number of rows with class i in the target column

$$\text{Entropy}(S) = \frac{-p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log_2\left(\frac{n}{p+n}\right)$$

- p: positive examples, n: negative examples

Information Gain



0 questions   1 question   2 questions

- Information gain G(S,A) of an attribute A relative to collection of samples S is defined by

$$G(S,A) = \text{Entropy}(S) - I(A)$$

- Effectiveness of an attribute classifying training data

Average Information Entropy

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

Q: Calculate entropy of the dataset S

Outlook	Temp	Humidity	Windy	Play tennis
Sunny	High	High	Weak	No
Sunny	High	High	Strong	No
Overcast	High	High	Weak	Yes
Rainy	Medium	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Medium	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rainy	Medium	Normal	Weak	Yes
Sunny	Medium	Normal	Strong	Yes
Overcast	Medium	High	Strong	Yes
Overcast	High	Normal	Weak	Yes
Rainy	Medium	High	Strong	No

p = no. of yes = 9 $\quad$ n = no. of no = 5

$$\text{Entropy}(S) = \frac{-9}{9+5}\log_2\left(\frac{9}{9+5}\right) - \frac{5}{9+5}\log_2\left(\frac{5}{9+5}\right)$$

$$= 0.94$$

Q: Calculate IG(S, Outlook)

Outlook	Temp	Humidity	Windy	Play tennis
Sunny	High	High	Weak	No
Sunny	High	High	Strong	No
Overcast	High	High	Weak	Yes
Rainy	Medium	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Medium	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rainy	Medium	Normal	Weak	Yes
Sunny	Medium	Normal	Strong	Yes
Overcast	Medium	High	Strong	Yes
Overcast	High	Normal	Weak	Yes
Rainy	Medium	High	Strong	No

$$IG(S, Outlook) = Entropy(S) - \sum_{i}^{Outlook} \frac{p_i + n_i}{p+n} Entropy(S_{outlook=i})$$

$$= 0.94 - \Bigg(\underbrace{\frac{5}{14} \times \left[-\frac{2}{5} \log_2\left(\frac{2}{5}\right) - \frac{3}{5} \log_2\left(\frac{3}{5}\right) \right]}_{Sunny}$$

$$+ \frac{4}{14} \times \underbrace{\left[-\frac{4}{4} \cancelto{0}{\log_2\left(\frac{4}{4}\right)} - \frac{0}{4} \right]}_{Overcast} + \frac{5}{14} \times \underbrace{\left[-\frac{3}{5} \log_2\left(\frac{3}{5}\right) - \frac{2}{5} \log_2\left(\frac{2}{5}\right) \right]}_{Rainy} \Bigg)$$

$$= 0.94 - \left(\frac{5}{14} \times 0.971 + \frac{5}{14} \times 0.971 \right) = 0.94 - 0.69 = 0.247$$

$$\boxed{IG(S, Outlook) = 0.247}$$

Q: Construct ID3 Decision Tree for the table shown

a_1	a_2	O/P
Y	B	Y
Y	B	Y
Y	W	N
R	W	Y
R	B	N
R	B	N

$$\text{Entropy}(S) = -\frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) = 1$$

Information gain on a_1

$$G(S, a_1) = \text{Entropy} - I(a_1)$$

$$= 1 - \left(\frac{3}{6}\left(-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}\right) + \frac{3}{6}\left(-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}\right)\right)$$

$$= 1 - 0.918$$

$$= 0.081$$

Information gain on a_2

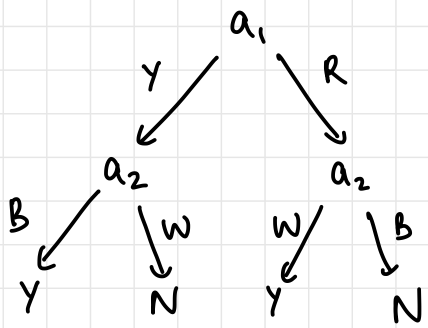
$$G(S, a_2) = \text{Entropy} - I(a_2)$$

$$= 1 - \left(\frac{4}{6}\left(-\frac{2}{4}\log_2\left(\frac{2}{4}\right) - \frac{2}{4}\log_2\left(\frac{2}{4}\right)\right) + \frac{2}{6}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)\right)$$

$$= 1 - \left(\frac{4}{6} + \frac{2}{6}\right) = 1 - 1 = 0$$

$\therefore$ First root $= a_1$

$\Rightarrow$ Second root $= a_2$



Note: if direct attribute (entropy = 0), decision tree ends

Need to repeat for all levels of tree

Q: Construct ID3 Tree for the following

M	N	O	Y
A	C	X	T
A	C	Z	T
A	D	X	T
A	D	Z	T
B	C	X	T
B	C	Z	F
B	D	X	F
B	D	Z	F

$$\text{Entropy}(S) = -\frac{5}{8}\log_2\left(\frac{5}{8}\right) - \frac{3}{8}\log_2\left(\frac{3}{8}\right) = 0.9544$$

$$IG(S,M) = 0.9544 - \left(\frac{4}{8}\times 0 + \frac{4}{8}\times\left(-\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}\right)\right)$$

$$= 0.9544 - 0.4056$$

$$= 0.548$$

$$IG(S,N) = 0.9544 - \left(\frac{4}{8}\times\overbrace{\left(-\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}\right)}^{0.811} + \frac{4}{8}\underbrace{\left(-\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}\right)}_{1}\right)$$

$$= 0.9544 - (0.4056 + 0.5) = 0.9544 - 0.9056$$

$$= 0.0487$$

$$IG(S,O) = 0.9544 - \left(\frac{4}{8}\times 0.811 + \frac{4}{8}\times 1\right) = 0.9544 - 0.9056$$

$$= 0.0487$$

$\therefore$ First node: M

For M = A

$$\text{Entropy}(S_{M=A}) = 0$$

Decision Tree ends here

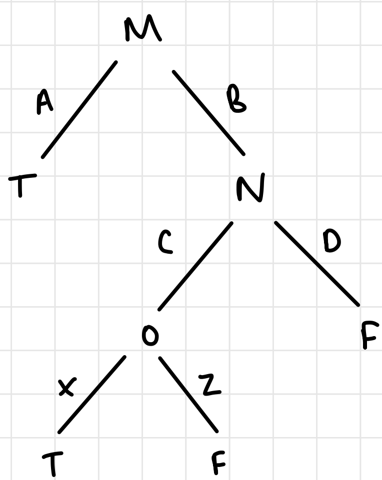
For $M = B$

$$\text{Entropy}(S_{M=B}) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}$$
$$= 0.811$$

$$IG(S_{M=B}, N) = 0.811 - \left(\frac{1}{2}\times 1 + \frac{1}{2}\times 0\right)$$
$$= 0.311$$

$$IG(S_{M=B}, O) = 0.811 - \left(\frac{1}{2}\times 1 + \frac{1}{2}\times 0\right)$$
$$= 0.311$$

$\therefore$ Second node can be either one



ID3 and Outliers

- ID3 handles outliers
- IF statements to derive branches
- Eg: missing salary for one employee
- Ignores missing data
- Better than Find S and Candidate Elimination

Hypothesis Space Search

- ID3 searches a space of hypotheses for one that fits the training examples
- Hypothesis space searched: set of possible decision trees
- Simple-to-complex, hill-climbing search through the hypothesis space
- Disjunction of conjuctions
- Starts with empty tree and considers more elaborate hypotheses with each step of selecting an attribute
- Evaluation function: information gain
- Find time complexity of constructing decision tree
- ID3 is greedy and performs no backtracking
- ID3 uses all training examples to make statistically based decisions on how to refine the current hypothesis (unlike Find S and Candidate Eliminations)

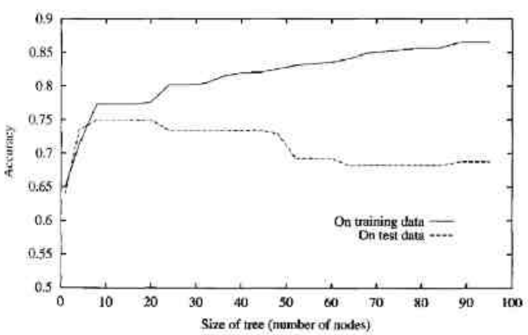
Inductive Bias

- Basis by which it chooses a consistent hypothesis over others
- Attribute that gives highest information gain is closest to root (best)
- Inductive bias of ID3 follows from its search strategy and not the definition of its search space (like Find S, Candidate Elim)

Issues

(a) Overfitting

- Perfectly classifies training data (more attributes → overfitting)
- As depth of tree grows, more overfitting
- Hypothesis $h \in H$ is said to overfit a dataset if there exists another hypothesis $h' \in H$ such that the error of h is less than the error of h' on the training set, but the error of h' is less than the error of h on the test set



Source : medium

- Shallow trees are less likely to overfit
- To make shallow, must prune (loss in accuracy)

Note: check paper on drive link (Bottom Up) (BRIDT)

PRUNING

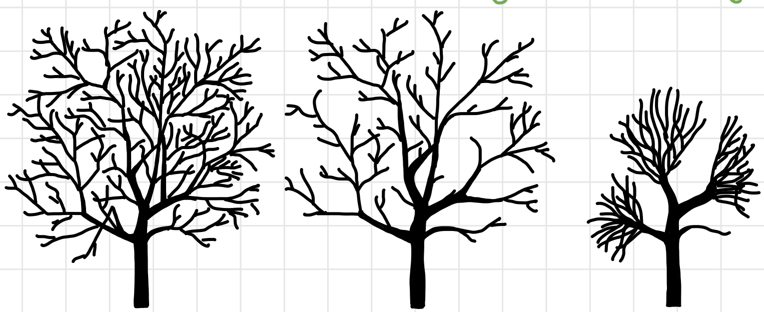
(i) Post-pruning

- Fully grow the tree and then selectively chop leaves and aggregate them into a parent with the most common value as predictor

(ii) Pre-pruning

- Halt tree growth when goodness of a split falls below a certain threshold (eg: min IG)
- Data loss ; not used as often

Post Pruning Pre Pruning



Approaches

- Use a separate set of examples (not training) to evaluate post-pruning nodes

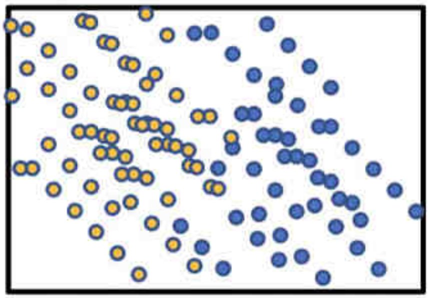
- Statistical test to estimate whether expanding or pruning a node is likely to produce an improvement beyond the training set

- Explicit measure of the complexity for encoding training examples and the decision tree ; halt growth when encoding size minimised (Minimum Description Length)

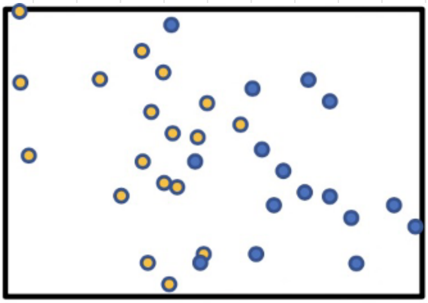
Post-Pruning: Reduced Error Pruning

replace with most common class

- Prune as long as error decreases
- Once error increases, undo pruning step
- Post-pruning (more commonly used)

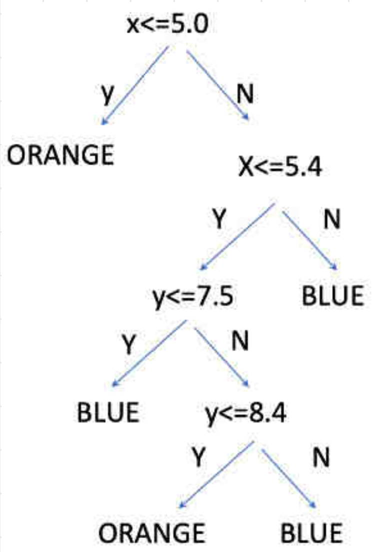


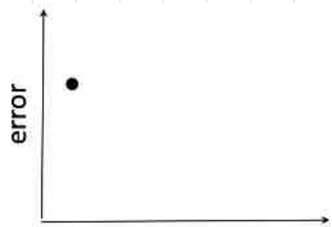
train set



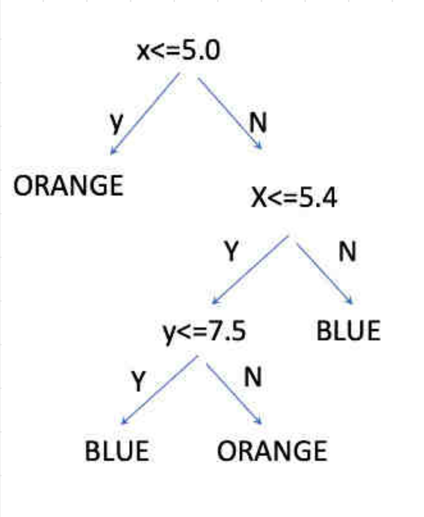
validation set

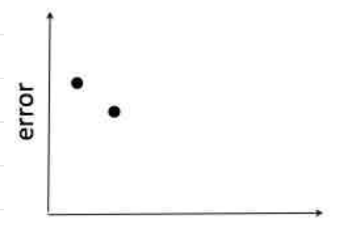
1)



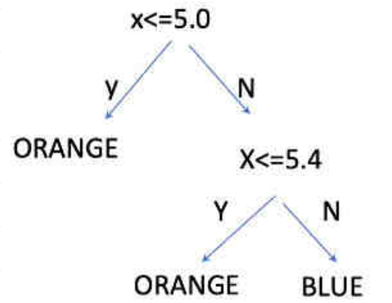


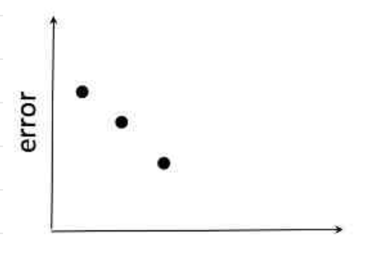
2)


x<=5.0
y
N
ORANGE
X<=5.4
Y
N
y<=7.5
BLUE
Y
N
BLUE
ORANGE

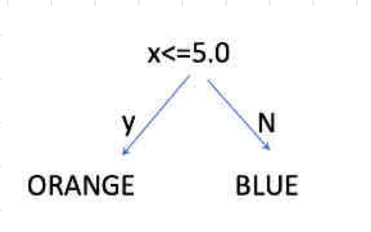

error

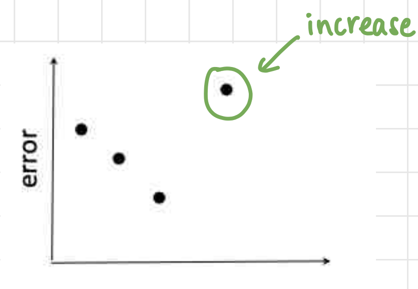
3)


x<=5.0
y
N
ORANGE
X<=5.4
Y
N
ORANGE
BLUE

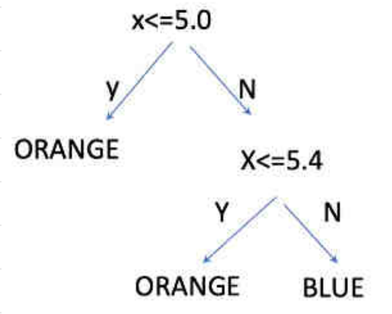

error

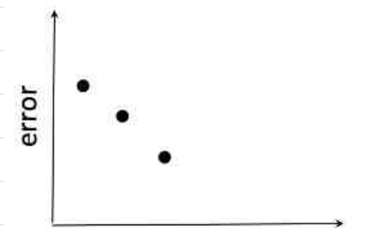
4)


x<=5.0
y
N
ORANGE
BLUE


increase
error

5)





Final tree

Post Pruning - Rule Post Pruning - check prof Preet's slides (L9)

- Uses conditionals (IF-THEN-ELSE)
- $C_{4.5}$ uses

(6) Handling continuous attributes

- Define intervals
- https://medium.com/@pralhad2481/chapter-3-decision-tree-learning-part-2-issues-in-decision-tree-learning-babdfdf15ec3
- How to define intervals?

TEMP	40	48	60	72	80	90
Play Sport	no	no	yes	yes	yes	no

interval boundary 1 = $\frac{60+48}{2}$ = 54

interval boundary 2 =

(c) handling attributes with missing data

- Can estimate based on other instances
- Eg: could use most common value - read Prof Preet's slides for more

(d) does not guarantee convergence