POMDP Basics

Lecture 8

MDP vs. POMDPs

- **MDP:** Agent's percept in any given state identify the state that it is in, e.g., state (4,3) vs (3,3)
  - Given observations, uniquely determine the state
  - Hence, we will not explicitly consider observations, only states

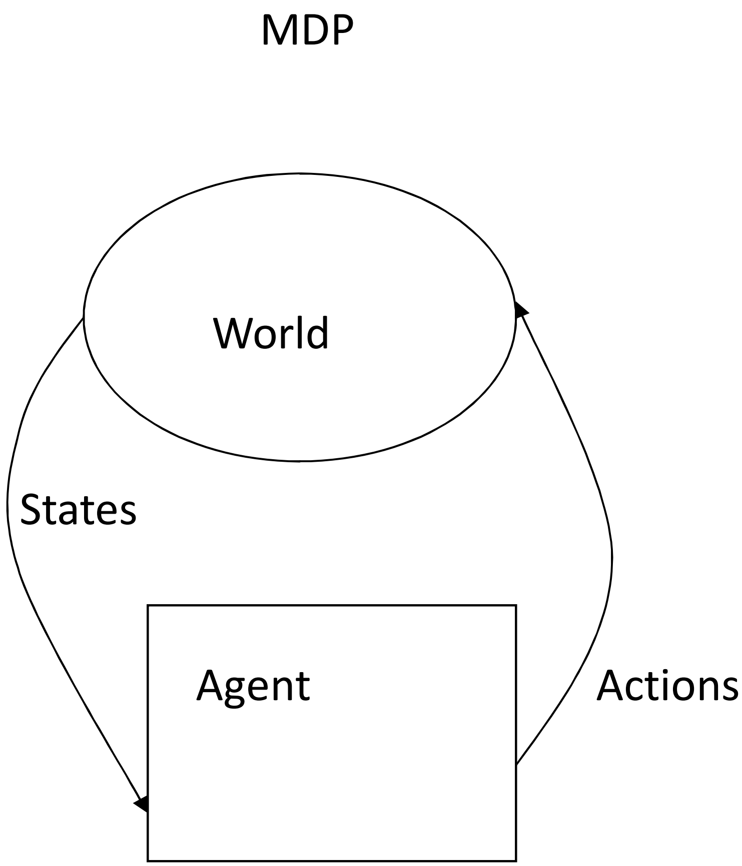
- **POMDP:** Agent's percepts in any given state **DO NOT** identify the state that it is in, e.g., may be (4,3) or (3,3)
  - Given observations, not uniquely determine the state
  - POMDP: Partially observable MDP for inaccessible environments

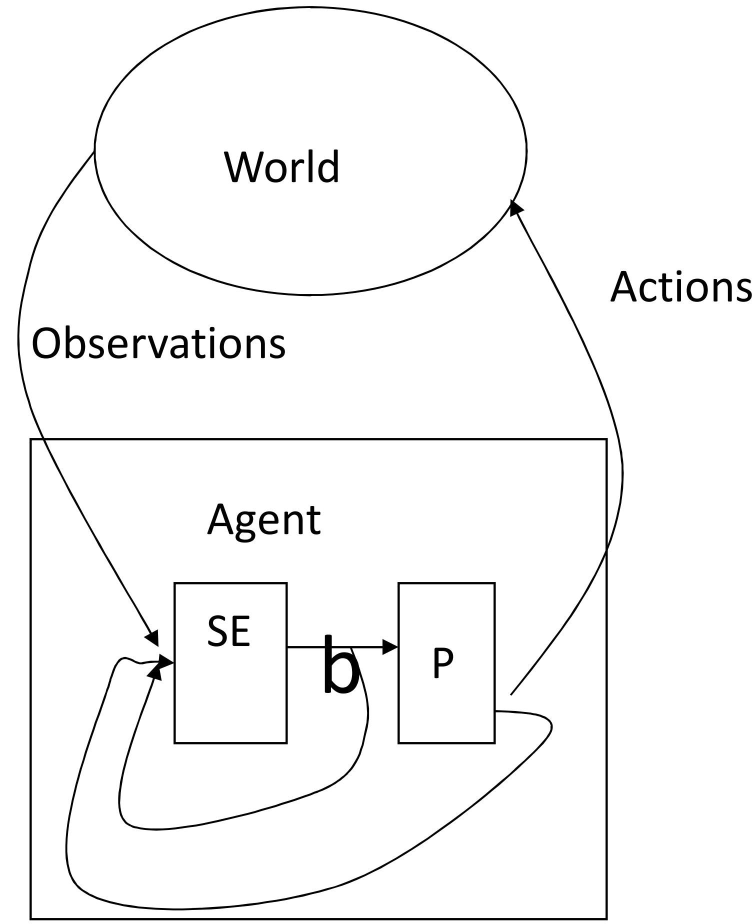
POMDP: Partially Observable Markov Decision Process

- Set of states, **S**
- Set of actions, **A**
- **P** is the table of transition probabilities
- **R(s,a)** reward received for taking action “a” in state “s”
- **Policy** π maps a **state “s”** to an **action “a”**

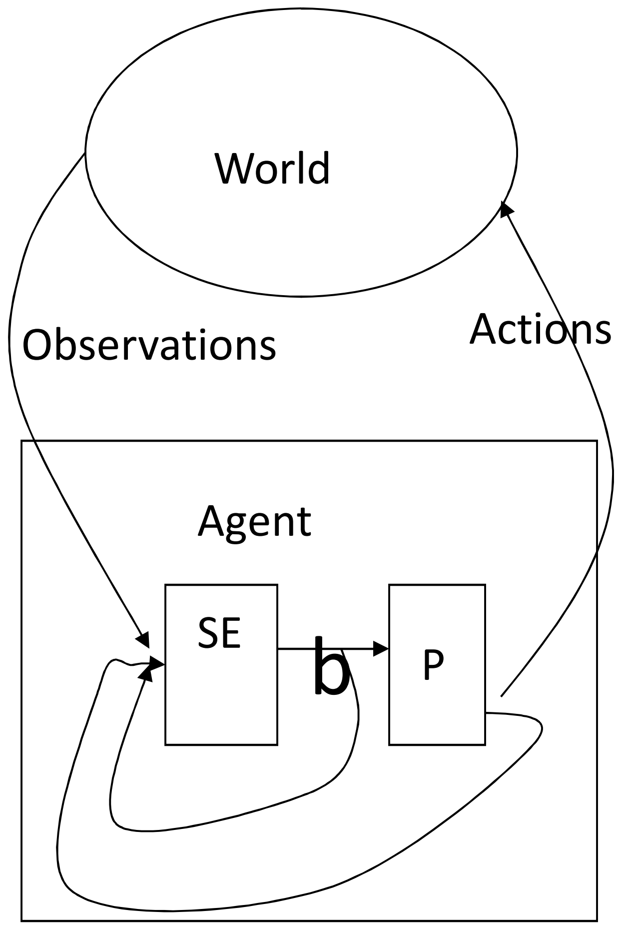
- **PLUS**
  - Finite set Ω of observations
  - Table **O** of observation probabilities where **O(o|a,s’)** is the probability that “o” is observed given that action “a” taken leads to state s’
  - **Policy** maps **histories of observations** to **actions**

MDP vs POMDP


MDP
World
States
Agent
Actions


World
Observations
Actions
Agent
SE
b
P

POMDP



SE: State estimator

b: Belief state

SE updates the beliefs
based on last observation,
previous belief state
and previous action

P: Policy is no longer a function of the state,
But of the agent's belief state

POMDP:**<S, A, P, R, Ω, O>**

- **S,** Set of states
- **A,** finite set of actions
- **P** is the table of transition probabilities
- **R(s,a)** reward received for taking action “a” in state “s”
- Finite set Ω of observations, e.g., *{red, green} in example below*
  - Observations hint at state, e.g., *Observe Red room, but not S3*
- Table **O** of observation probabilities
  - ***O(o|a,s’)*** *prob “o” observed given action “a” leads to state s’*
  - *P(red | LEFT, S3) = 0.4*

POMDP: Partially Observable Markov Decision Process

- Agent has initial beliefs
- Agent takes an action
- Gets an observation
- Interprets the observation
- Updates beliefs
- Decides on an action
- Repeats

Agent takes optimal action considering world/other agents

Elements: {States, Actions, Transitions, Rewards, Observations }

POMDP: Partially Observable Markov Decision Process

- Underlying dynamics are still **Markovian**: World has **NOT** changed its characteristics, agents sensors have changed

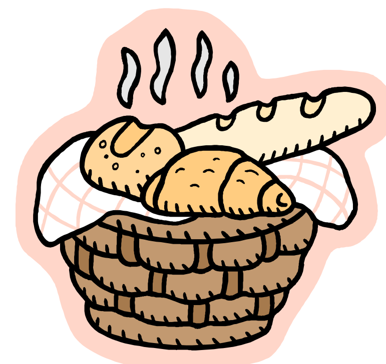
- Observations only hint at what state we are in, but not exactly identify state

- So, somehow agent may need to remember what it observed in the past and what action it took:
  - *If I observed feature "green" in the past, then took action "left" and then observed "red", it must mean that I am either in state S3 (probability of 0.9) or S2 (Prob 0.1) now*

- *Need to maintain beliefs*

Tiger Problem





- Standing in front of two closed doors
- World is in one of two states: tiger is behind left door or right door
- Three actions: Open left door, open right door, listen
  - *Listening is not free, and not accurate (may get wrong info)*

- Reward: Open the wrong door and get eaten by the tiger (large –ve)
  Open the right door and get a prize (small +ve)

Tiger Problem: POMDP Formulation

- Two states: SL and SR
- Three actions: LEFT, RIGHT, LISTEN
- Transition probabilities:

Listen	SL	SR
SL	1.0	0.0
SR	0.0	1.0

Left	SL	SR
SL	0.5	0.5
SR	0.5	0.5

Right	SL	SR
SL	0.5	0.5
SR	0.5	0.5

Tiger Problem: POMDP formulation

- Observations: TL (tiger left) or TR (tiger right)
- Observation probabilities:

Listen	TL	TR
SL	0.85	0.15
SR	0.15	0.85

Left	TL	TR
SL	0.5	0.5
SR	0.5	0.5

Right	TL	TR
SL	0.5	0.5
SR	0.5	0.5

- Rewards:
  - *R(SL, Listen) = R(SR, Listen) = -1*
  - *R(SL, Left) = R(SR, Right) = -100*
  - *R(SL, Right) = R(SR, Left) = +10*

How to Find the Optimal Policy?

- Now lets find an optimal policy for this problem

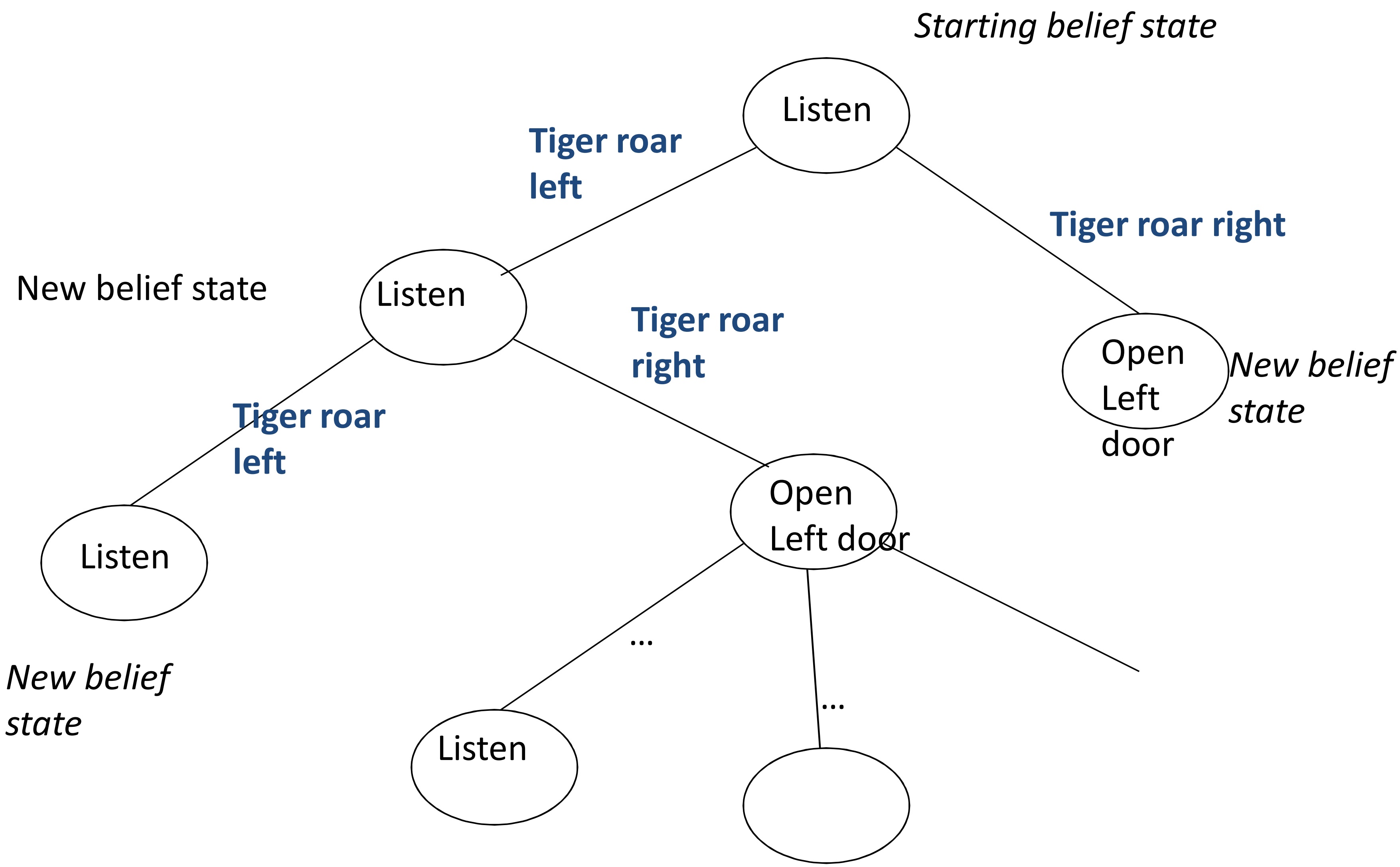
- Why not use value iteration directly?
  - $U_{t+1}(I) = R(I) + \max_{A} \sum_{J} P(J|I,A) * U_{t}(J)$

- Could we compute the utilities in this manner?

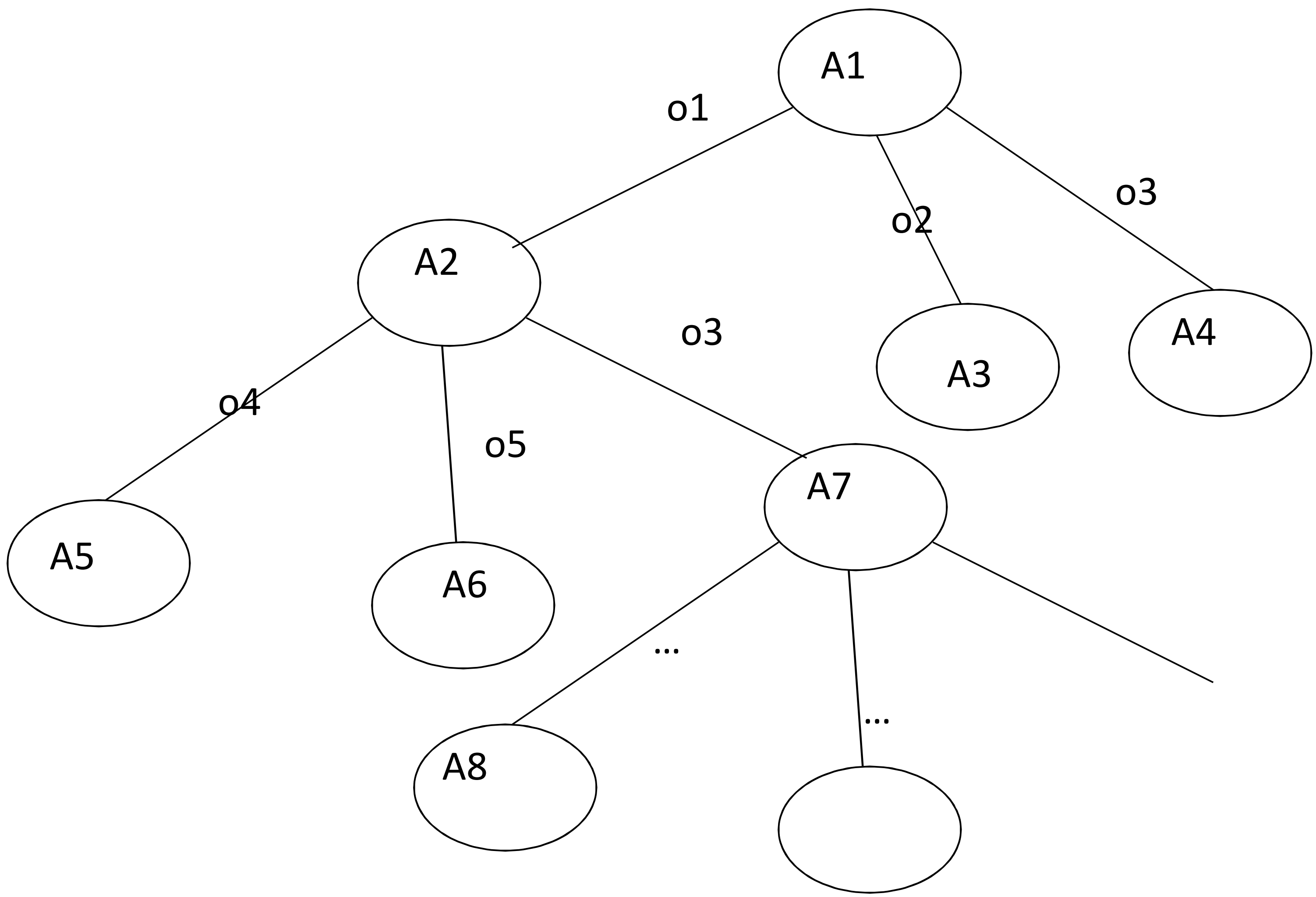
- Could such utilities be actually used?

- Need mapping from belief states to actions!

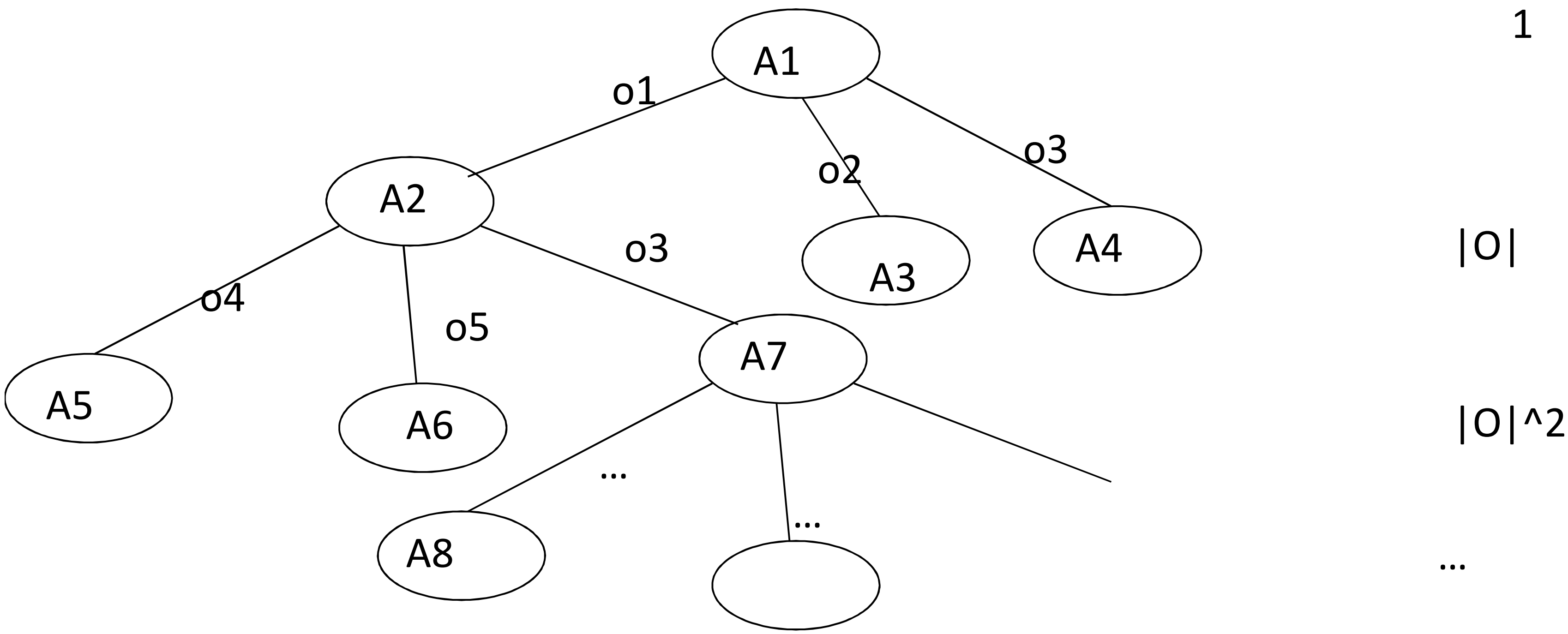
Sample POMDP Policy Tree



POMDP Policy Tree



How Many POMDP policies possible



How many policy trees, if |A| actions, |O| observations, T horizon:

- How many nodes in a tree:

$$N = \sum_{i=0}^{T-1} |O|^i = (|O|^T - 1) / (|O| - 1)$$

How many trees:

$$|A|^N$$

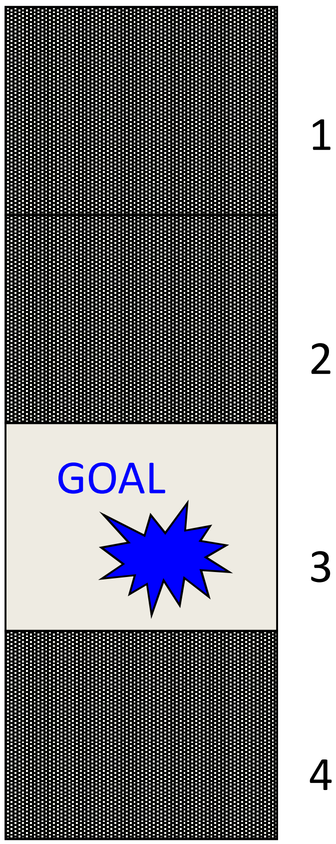
POMDP Belief State

- Computing belief state important, since policy maps belief state to action
  - *Not just the most probable state of the world*

- Probability distributions over the states of the world
  - *Sufficient statistic* for the past history and initial belief state:
    - *No additional data about past actions & observations supplies any further information*

  - *That is, process over belief states is a markov process (why?)*
  - *As if maintained a complete history of actions & observations*

Evolution of Belief State: 1



- Set of states, S1, S2 S3, S4
- For each $s \in S$, A_s set of actions: Down or Up
- Transition Prob T: 0.9 (direction of move), 0.1 opp
- R(s,a) reward received

–Finite set Ω of observations

– O observation probabilities:

  – $Pr(o1|s1) = Pr(o1|s2) = Pr(o1|s4) = 1$

  – $Pr(o2|s3) = 1$

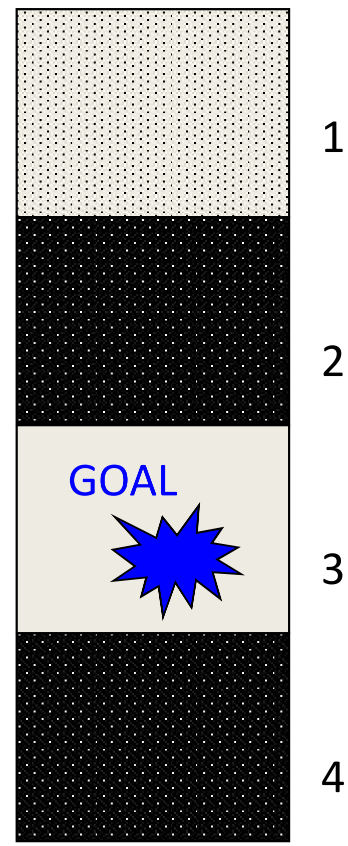
Initial belief state: [0.333, 0.333, 0, 0.333]

S1 S2 S3 S4

Evolution of Belief State: 2

Suppose agent moves down and observes o1:

- What should the agent believe about its state? Does it now know more about where it is more likely to be?



- [0.100, 0.450, 0, 0.450]

Belief State

- b $\rightarrow$ probability distribution over our set of states, e.g., over s1, s2, s3, s4

- b(s) denotes the probability assigned to world state s by belief state b
- *In [0.333, 0.333, 0, 0.333], what is b(s1)?*

- 0 <= b(s) <= 1

$$\sum_{s \in S} b(s) = 1$$

Computing Belief States

- s = old state
- b = old belief state, and b(s) probability of s given belief state b
- a = action
- b’ = new belief state
- b’(s’) = probability of s’ given b’
- o = observation

Computing Belief States

$$b'(s') = \Pr(s' \mid o, a, b) = \Pr(s' \wedge o \wedge a \wedge b) / \Pr(o \wedge a \wedge b)$$

$$= \frac{\Pr(o \mid s', a, b)\, \Pr(s' \mid a, b) * \Pr(a \wedge b)}{\Pr(o \mid a, b) * \Pr(a \wedge b)}$$

$$= \frac{\Pr(o \mid s', a)\, \Pr(s' \mid a, b)}{\Pr(o \mid a, b)}$$

Will not repeat $\Pr(o \mid a, b)$ in the next slide, but it is there!

– *Treated as a normalizing factor, so that b' sums to 1*

Computing Belief States: Numerator

$= Pr(o \mid s' \; a) \; Pr\,(s' \mid a, b) \;\; = O(s', a, o) \; Pr\,(s' \mid a, b)$

$= O(s', a, o) \sum Pr\,(s' \mid a, b, s) \; Pr\,(s \mid a, b)$

$= O(s', a, o) \sum Pr\,(s' \mid a, b, s) \; b(s) \quad ; Pr\,(s \mid a, b) = Pr\,(s \mid b) = b(s)$

$= O(s', a, o) \sum T(s, a, s') \; b(s)$

(Please work out some of the details later)

Belief State

Overall formula

$$= \frac{O(s', a, o) \sum T(s, a, s')\, b(s)}{Pr\,(o \mid a, b)}$$

Example

Moves down and does not observe s3

- b → b'
- i.e., [0.333, 0.333, 0, 0.333] → [0.1, 0.45, 0, 0.45]

b'(s1) = probability of s1 in our new belief state b'

Numerator = Pr (o1 | s1, down)) *

[Pr(s1 | s1, down) * b(s1) + Pr (s1 | s2, down) * b(s2) +

Pr (s1 | s3, down) * b(s3) + Pr (s1 | s4, down) * b(s4)]

= 1 * [0.1 * 0.333 + 0.1 * 0.333 + 0 + 0] = 0.0666

Why is this not 0.1?

Example

Moves down and observes o1 (I.e., not observe s3)

- b → b'
- i.e., [0.333, 0.333, 0, 0.333] → [0.1, 0.45, 0, 0.45]

In b'(s1), *Numerator* = 0.0666

The above is unnormalized probability, hence not 0.1!

Denote unnormalized b'(s1) as Ub'(s1)

- Similarly calculate unnormalized Ub'(s2), Ub'(s3), Ub'(s4)
- [Ub'(s1) + Ub'(s2) + Ub'(s3) + Ub'(s4)]/denominator = 1
- Denominator = 0.666 (please check at home)
- b(s1) = 0.0666/0.666 = 0.1

Policy: Map Belief State to Action

Convert POMDP → "Belief MDP"

- *Recall, process over belief states is markov*
- B, the set of belief states, is the set of MDP states
- A, the set of actions, is the same
- R'(b,a) is the reward function on the belief states:

$$R'(b, a) = \sum_{s \in S} b(s)\, R(s, a)$$

- Transition function:

$$T(b, a, b') = Pr(b' \mid a, b) = \sum_{o \in \Omega} Pr(b' \mid a, b, o) * Pr(o \mid a, b)$$

$$\textit{Where } Pr(b' \mid b, a, o) = \begin{cases} 1 \textit{ if } SE(b, a, o) = b' \\ 0 \textit{ otherwise} \end{cases}$$

Transition Function

<u>Note: observe-s3 = o2, not(observe-s3) = o1</u>

E.g., T([0.330, 0.330, 0, 0.330], down, [0.1, 0.45, 0, 0.45])
= *Pr (b' | down, b, observe-s3) * pr (observe-s3 | down, b)*
*+ Pr (b' | down, b, Not(observe-s3)) * pr (Not (observe-s3)| down, b)*

= 0 * pr (observe-s3 | down, b) + 1 * pr (Not(observe-s3) | down, b)

= Pr (*Not(observe-s3)* | down, b)   = 0.666

T(b, a, b') = Pr (o | a, b) where when action "a" taken in belief-state "b" and we observed "o", we ended up in belief-state b'

Try Value Iteration

- Given belief MDP, if we can generate an optimal policy, it will give rise to optimal behavior for the original POMDP
- How about trying value iteration in this belief MDP?

$$V'(b) = \max [\, r(b,a) + \Sigma P(b' \mid b,a)\, V(b')]$$
$$= \max [\, r(b,a) + \Sigma P(o \mid b,a)\, V(b^a{}_o)]$$

- *Where $r(b, a) = \Sigma r(s,a)b(s)$ is the expected immediate reward for taking action a in belief state b*
- *o is the observation, $P(o|b, a)$ implies the probability of observing o given action a in belief state b*
- *$V(b^a{}_o)$ denotes the value for belief state at the next point in time given that action a was taken in belief state b, with observation o*

Problem in Value Iteration

- Infinite possible belief states:
  - Assume: we don't have a fixed start belief state
  - *MDP has a continuous state space*
  - *No longer a table of states where we can maintain a value per state*
- Also, how to back up values of future belief states --- there are too many (infinite) future belief states as well

Learning from examples
Chapter 18 from the book by Russell and Norvig
Decision Tree Learning

Lecture 9

What is a Decision Tree?

- Input: A vector of attribute values
- Output: Decision (e.g., Yes/No, Win/Tie/Loss)
  - Each leaf node provides an output
- Decision tree reaches its decision by performing a sequence of tests
  - No chance nodes
  - Instead, test attributes and provide a response
- DT Learning: One of the simplest, yet most successful forms of ML
  - Each internal node represents test of attribute
  - Branches represent possible values of the attribute
- Natural representation for humans e.g. many “How To” manuals written as single DT over 100s of pages

Example Problem

- Predicting the outcome of a RoboCup Soccer game
  - After first few minutes of play, predict the outcome
  - Assume no goals scored yet…

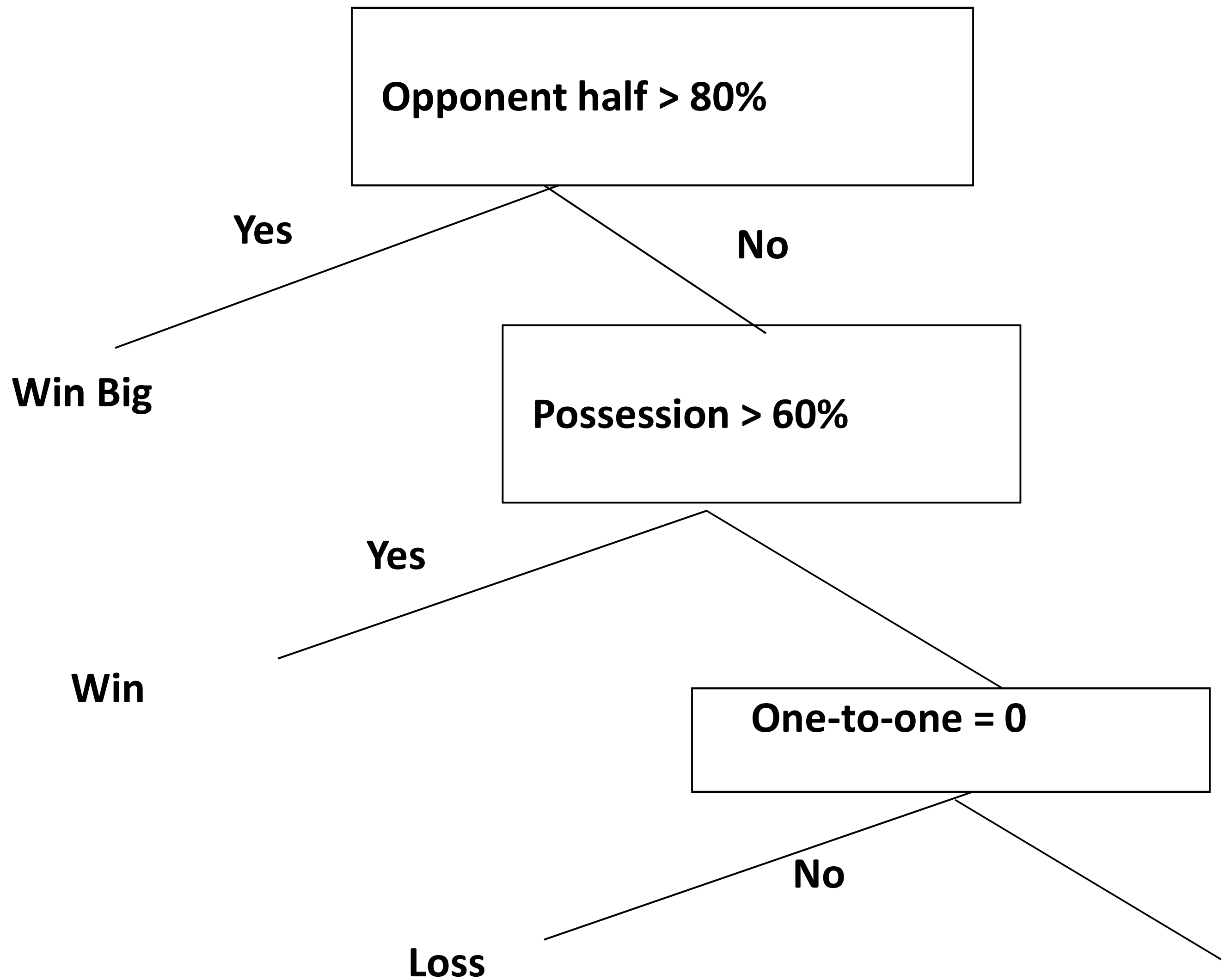
- Aim: Learn a function that will tell us if a team will:
  - Win Big: > 5 goals difference with the opponent
  - Win: 1-5 goal difference
  - Tie: No goal difference
  - Lose: Opponent outscores by 1 to 5 goals
  - Lose big: Opponent outscores by more than 5 goals

Relevant Attributes

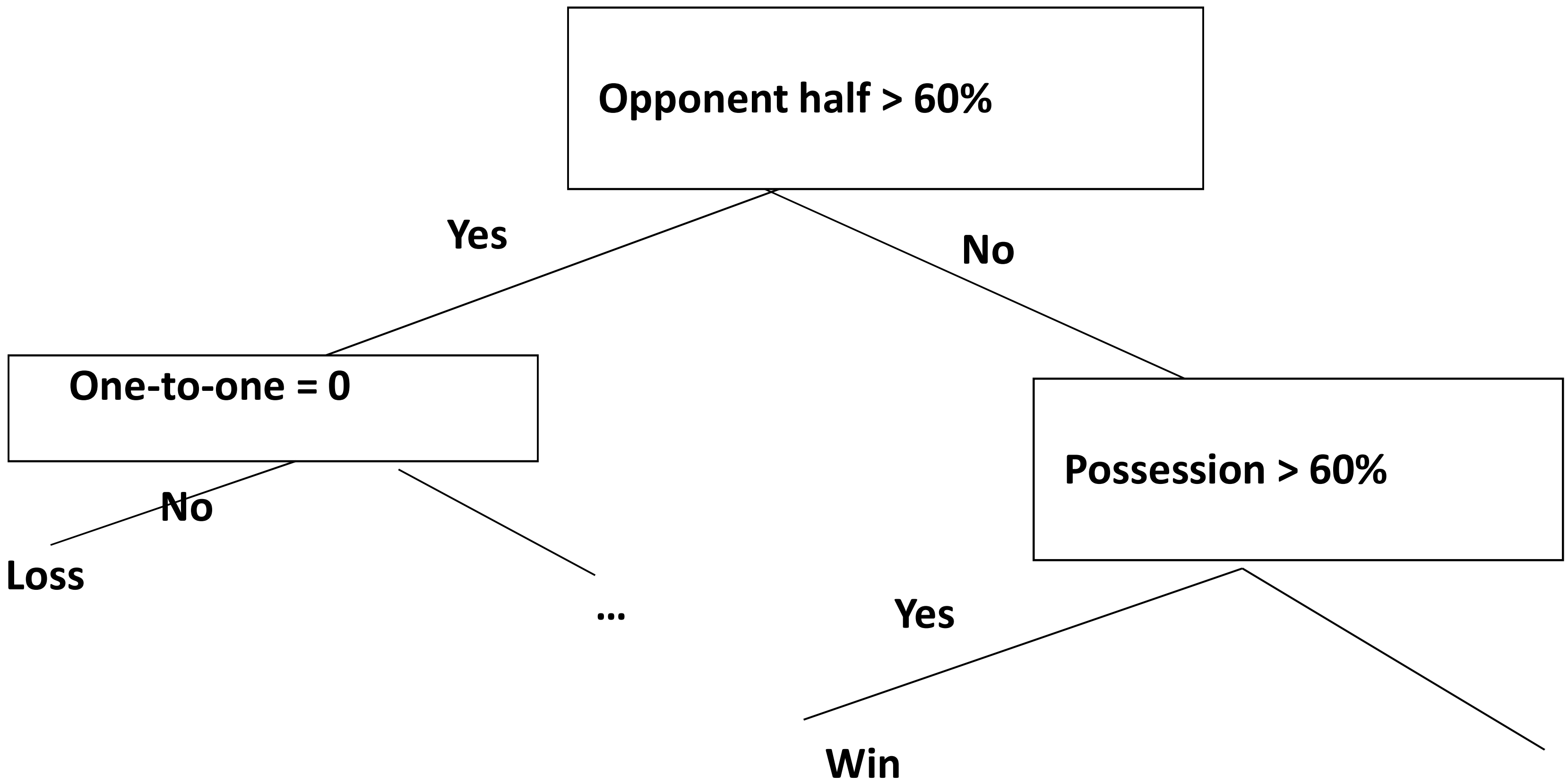
- What attributes would you consider relevant for this prediction?
  - Possession time: What percent of time ball with team?
  - % time ball in opponent half
  - Placement of opponent’s defenders
  - Opponent playing along sidelines or the center
  - One-on-one with opposition goal keeper
  - …

Possible Decision Tree

- Attribute tests expressed as rules
- Tests from root to leaf
- If OH > 80%, then Win Big



Possible Decision Tree



Tree could be of different shapes

EXAMPLE INPUT FOR DECISION TREE LEARNING

Number	Outlook	Temp	Humid	Windy	*Class*
1	Sunny	hot	high	false	***N***
2	Sunny	hot	high	true	***N***
3	overcast	hot	high	false	***P***
4	rain	mild	high	false	***P***
5	rain	cool	normal	false	***P***
6	rain	cool	normal	true	***N***
7	overcast	cool	normal	true	***P***
8	sunny	mild	high	false	***N***
9	sunny	cool	normal	false	***P***
10	rain	mild	normal	false	***P***
11	sunny	mild	normal	true	***P***
12	overcast	mild	high	true	***P***
13	overcast	hot	normal	false	***P***
14	rain	mild	high	true	***N***

Selecting Order of Tests

- Intuition is that if we could cleanly separate the sets into positive and negative that would be the best
- What if we get two sets where 100% positive in one set and 90% negative in 2nd set? What if 50% negative in 2^{nd} set?
- How do you formalize this idea?
  - Need a function that differentiates between cleanly separated sets (P & N examples are separated out completely) and mixed sets (where 50% P & 50% N)
  - Based on “information gain” idea
  - Information gain in turn based on information necessary to classify an example

Expected Information

- Expected information needed to classify an example instance is given by the following formula:

$$I(p,n) = -\frac{p}{(p+n)} \operatorname{Log}_2 \frac{p}{(p+n)} - \frac{n}{(p+n)} \operatorname{Log}_2 \frac{n}{(p+n)}$$

Can be generalized to more than 2 classes: p, n
For instance, if we had k such classes…

How does this work

- Out of 14 objects in our example, 7 are P and 7 are N
- Information required for classification is therefore:

$$I(p,n) = -\frac{7}{14}\,\mathrm{Log}_2\,\frac{7}{14} - \frac{7}{14}\,\mathrm{Log}_2\,\frac{7}{14} = 1 \text{ bit}$$

- *What if out of 14 objects, 14 are P? What if 14 are N?*
- *Does I(p,n) have the properties we might be looking for?*
- *when is it highest? When is it lowest?*

Another Example

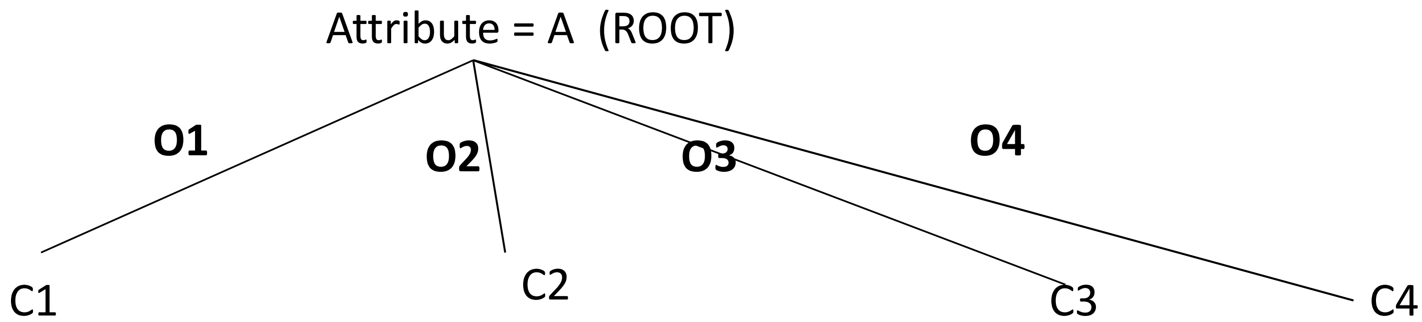
- Out of 14 objects in our example, 9 are P and 5 are N
- Information required for classification is therefore:

$$I(p,n) = -\frac{9}{14}\mathrm{Log}_2\frac{9}{14} - \frac{5}{14}\mathrm{Log}_2\frac{5}{14}$$

= ~0.4096 + ~0.5292

~0.940 bits

The overall situation…



Proportion
In each branch

$$\frac{(p1 + n1)}{(p + n)} \qquad \frac{(p2 + n2)}{(p + n)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots$$

Information requirement for each branch

I(p1, n1) I(p2, n2) ……….

Several Attribute Tests Possible

- Lets look through these tests case by case
- outlook attribute:
  - Divides the set with values {sunny, overcast, rain}
  - Division is {5 (sunny), 4 (overcast), 5 (rain)}
  - Further we have the following:
    - 5 (Sunny) → 2 P, 3 N → I(p1,n1) = 0.971
    - 4 (overcast) → 4 P, 0 N → I(p2, n2) = 0
    - 5 (rain) → 3 P, 2 N → I(p3, n3) = 0.971

- E(outlook) = 5/14 * I(p1,n1) + 4/14 * I(p2,n2) + 5/14 * I(p3, n3)
  = 0.346 + 0 + 0.346 = 0.694 bits

- Gain (outlook) = 0.940 – E(outlook) = 0.246 bits
- As computed earlier - I(p,n) = .940 with 9 positive and 5 negative

Next Attribute Test

- Temperature attribute:
  - Divides the set with values {hot, mild, cool}
  - Division is {4 (hot), 6 (mild), 4 (cool)}
  - Further we have the following:
    - 4 (hot) → 2 P, 2 N → I(p1,n1) = 1
    - 6 (mild) → 4 P, 2 N → I(p2, n2) = …
    - 4 (cool) → 3 P, 1 N → I(p3, n3) = .8075

- E(temperature) = 4/14*I(p1,n1) + 6/14 * I(p2,n2) + 4/14 * I(p3, n3)
  = 0.285 + 0.396 + 0.230 = 0.911 bits

- Gain (temp) = 0.940 – E(temp) = 0.029 bits

Which Test First?

- Gain(outlook) = 0.246 bits
- Gain (temperature) = 0.029 bits
- Gain (humidity) = 0.151 bits
- Gain (windy) = 0.048 bits

- So choose outlook as the attribute for root of decision tree
- Objects divided into subsets according to values of outlook attribute
- Decision tree for each subset induced in a similar fashion
- Smaller decision tree generated by this procedure