Nov 9 Networks and Information

1) What if the chain were longer? 10 states? 100 states?

2) What if this 0.04 was 0.4?

Consider a state space of 5 locations on a line, with rewards given below:

State:	a	b	c	d	e
Reward:	10	-0.04	-0.04	-0.04	1

Our agent can either move left L or right R. *If* it chooses to move a direction, there is an 80% chance it actually does so and a 20% chance that it remains in place. The left and right endpoints are terminal states.

What do we do?

Announcements and To-Dos

Due Nov 16.

1. Homework to be posted tomorrow (EVIU, Bayes Net).
2. Practicum "due" today, but can turn it in up to Friday.

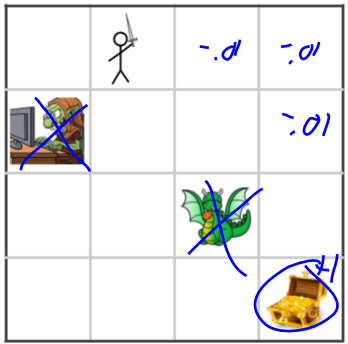
Last time we learned: Midterm is posted

1. Finished up with HMM!

MDPs

Consider an agent-based game. We win if we reach the treasure. We lose if we run into the internet troll or the dragon.

Goal: describe the appropriate set of moves from our current location to the treasure.



A **Markov Decision Process** answers this problem by specifying:

1. Sets of states s, from which we can take actions a.
2. Transition probabilities of what the next state s' is *given* the action we're taking from state s.
3. Rewards from each movement, depending on where we go.
4. A possible discount factor, devaluing later rewards compared to current ones.

time

MDP Value Iteration:

Along the way, we populate the **utility** of each state: the expected rewards if we move optimally from that state.

Value Iteration Algorithm:

1. Start with *candidate* utilities for each state. $U(s) = 0 \quad \forall s$
2. Do the following *many* times:
   For each state s: (consider (3,1))
   - 2.1 Collect the set of valid actions $a \in A$
   - 2.2 For each a, consider the successor of that action and their associate probabilities: $P(s'|s,a)$, **then** calculate the expected utility of action a.
   - 2.3 Assign $U(s)$ to the max of the discounted expected utilities, plus any immediate reward for s.

$U^{\uparrow} = .7$ $U^{\rightarrow} = .7$ $U^{\leftarrow} = -.1$ $U^{\downarrow} = .1$

set $U(3,1) = U^{\rightarrow} = .7$

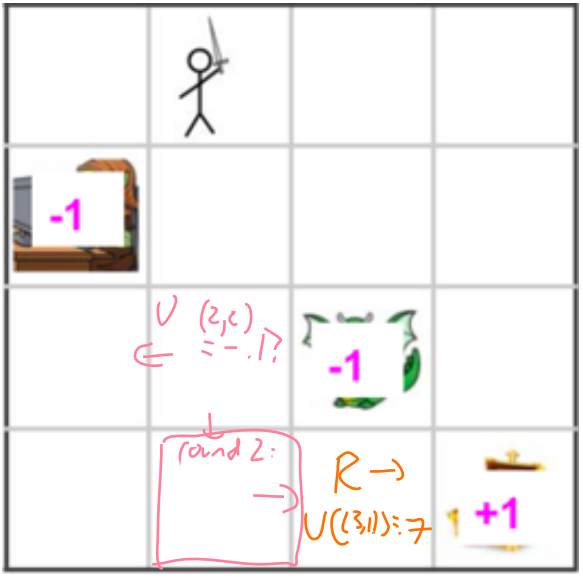
MDP Value Iteration:

for each state...
for each action
find U^a
Choose best U^a, a.

Value Iteration Algorithm:

1. Start with *candidate* utilities for each state.
2. Do the following *many* times:
   For each state s: $U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$
3. Stop when the utilities are no longer updating by more than specified tolerance.

Example: Find U_0 and U_1 for each state.



Value Iteration Pseudocode

```
def value_iteration(mdp, tolerance):

    # initilize utility for all states

    # iterate:

        # make a copy of current utility, to be modified

        # initialize maximum change to 0

        # for each state s:

            # for each available action, what next states
            # are possible, and their probabilities?

            # calculate the maximum expected utility

            # new utility of s = reward(s) +
            #                    discounted max expected utility

            # update maximum change in utilities, if needed

        # if maximum change in utility from one iteration to the
        # next is less than some tolerance, break!

    return # final utility
```

for each state and / action compute $\hat{U}(s)$

Policy Iteration

An alternative to the value iteration to solve an MDP is the policy iteration.

Policy Iteration Algorithm:

for each state for each action: check best.

A two-step iterative algorithm that alternates between **policy evaluation** and **policy improvement**

Initialize/randomize to

Break if policies don't change

1. **Policy Evaluation**: *Given* a policy π_i, compute $U_i = U_i^{\pi}$, the utility of each state if that policy is followed. This is only *one* action considered per state.

$$U_i(s) = R(S) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

↳ option of Q-table exact utilities of actions.

2. **Policy Improvement**: After updating the utility calculations in step 1, calculate a new policy π_{i+1} using π_i and U_i. Compare the utility from π_i to alternatives $a \in A$. If

$$\max_{a \in A} \sum_{s'} P(s'|s,a) U_i(s') > \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

a vs prior policy.

max a∈A: Util of a. Util of $\pi_i(s)$.

then set $\pi_{i+1}(s) =$ that action.

Policy Iteration Pseudocode

```
def policy_iteration(mdp):

    # initilize utility for all states
    # initialize a policy for each state, being a random action
    # iterate:

        # update utility, using policy evaluation and
        # current estimates of utility and policy

        # initialize unchanged = True

        # for each state s:

            # among the possible actions, which yields
            # the maximum expected utility?

            # if the best action choice is not currently
            # the policy for s, update it

        # if no policy values are changed, break!

    return # final policy (and/or utility)

def policy_evaluation(policy, utility, mdp, n_iter):
    # do a handful of value iteration updates of
    # the input utility, under the given policy
    return # updated utility
```

U(1)=0

random choice → ← ↓ ↑

U((3,2)) ← utility of policy for (3,2)

find new policy given these utilities

Policy Iteration Example

Consider a state space of 5 locations on a line, with rewards given below:

State:	a	b	c	d	e
Reward:	10	-0.04	-0.04	-0.04	1

Our agent can either move left L or right R. *If* it chooses to move a direction, there is an 80% chance it actually does so and a 20% chance that it remains in place. The left and right endpoints are terminal states.

What do we do?

Suppose we initialize the states as $U(s) = 0$ and intialize our policy as $\pi(s) =$ "R" for all states.

Policy Iteration Example

State:	a	b	c	d	e
Reward:	10	-0.04	-0.04	-0.04	1
Utility$_0$:	0	0	0	0	0

Step 1: Policy Evaluation: Compute the value of our policy at each step, given by

$$U_i(s) = R(S) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

we are going →, ∀ states

$U^{\rightarrow}(a) = 10 + \text{exit}$ (−.04, +.80)

$U^{\rightarrow}(b) = R(b) + .8(U(c)) + .2(U(b)) \simeq -.04$

80% works; 20% stay in place

Policy Iteration Example

State:	a	b	c	d	e
Reward:	10	-0.04	-0.04	-0.04	1
Utility$_0$:	0	0	0	0	0

Step 1: Policy Evaluation: Compute the value of our policy at each step, given by

$$U_i(s) = R(S) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

$U_R(a) = R(a) + \text{exit} = 10 + \text{exit}$
$U_R(b) = R(b) + .8(U(c)) + .2(U(b)) = -.04 + .8(0) + .2(0) = -.04$
$U_R(c) = R(c) + .8(U(d)) + .2(U(c)) = -.04 + .8(0) + .2(0) = -.04$
$U_R(d) = R(d) + .8(U(e)) + .2(U(d)) = -.04 + .8(0) + .2(0) = -.04$
$U_R(e) = R(e) + \text{exit} = 1 + \text{exit}$

Policy Iteration Example

State:	a	b	c	d	e
Reward:	10	-0.04	-0.04	-0.04	1
Utility$_1$:	10	-0.04	-0.04	-0.04	1
Policy$_0$:	E	R	R	R	E

Step 2: Policy Improvement: Choose the best policy at each step.

$$U_i(s) = R(S) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

all:

80% where we try to go

+ 20% where we are

+ R(state we are in.

$U^{\rightarrow}(b)$ vs $U^{\leftarrow}(b)$

AND

$U^{\rightarrow}(c)$ vs $U^{\leftarrow}(c)$

$U^{\rightarrow}(d)$ vs $U^{\leftarrow}(d)$

Policy Iteration Example

State:	a	b	c	d	e
Reward:	10	-0.04	-0.04	-0.04	1
Utility$_1$:	10	-0.04	-0.04	-0.04	1
Policy$_0$:	E	R	R	R	E

Step 2: Policy Improvement: Choose the best policy at each step.

$$U_i(s) = R(S) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

2 options $\in A(b)$ — 80% — 10% 10%

$\pi(b) = 'L'$

$$U_R(b) = R(b) + .8(U(c)) + .2(U(b)) = -.04 + .8(-0.04) + .2(-0.04) = -.08$$
$$U_L(b) = R(b) + .8(U(a)) + .2(U(b)) = -.04 + .8(10) + .2(-0.04) = 7.95 \text{ choose L}$$

$\pi(c) = 'R'$

$$U_R(c) = R(c) + .8(U(d)) + .2(U(c)) = -.04 + .8(-0.04) + .2(-0.04) = -.08$$
$$U_R(c) = R(c) + .8(U(b)) + .2(U(c)) = -.04 + .8(-0.04) + .2(-0.04) = -.08$$

$\pi(d) = 'R'$

$$U_R(d) = R(d) + .8(U(e)) + .2(U(d)) = -.04 + .8(1) + .2(-0.04) = .75$$
$$U_L(d) = R(d) + .8(U(c)) + .2(U(d)) = -.04 + .8(-0.04) + .2(-0.04) = -.04 \text{ choose R}$$

Policy Iteration Example

	b	c	d
Policy$_1$	L	R	R
Policy$_2$	L	L	R
Policy 3	L	L	L
Policy 4	L	L	L

State:	a	b	c	d	e
Reward:	10	-0.04	-0.04	-0.04	1
Utility$_1$:	10	-0.04	-0.04	-0.04	1
Policy$_0$:	E	R	R	R	E
Utility$_2$:	10	7.95	-0.08	.75	1
Policy$_1$:	E	L	R	R	E

And then we repeat... eventually node d will "see" BREAK the utility make its way over from $R(a)$, and we'll choose to go left!

After this point, we'd run the two steps *again*, and break the algorithm once the policy didn't update for *any* of the 3 interior states.

Decision-Making Networks

Our next discussion on networks tries to bridge the concepts we discussed over EVUI and EVPI with a network.
\vspace.5cm

As usual, we often couch optimal decision making in terms of *expectation*.

Example: Suppose a lottery has a payout of $\$1,000,000$ and the odds of winning are 1 in $500,000$. Tickets cost $\$1$. Should you buy a lottery ticket?
\vspace.5cm

,2¿**Solution:** *YES!* The reward for winning is 10^6, and the probability is 1 in $5 \cdot 10^5$. Then the *expected rewards* for each ticket is $10^6 \times \frac{1}{5 \cdot 10^5} = 2$, a gain of \$1. So buying a ticket is the decision that maximizes our expected utility, when the other decision was "don't buy a ticket.")

Decision-Making Networks

Definition: A *Decision-Making Network* (or influence diagram) is an extension of a Bayesian network in which nodes are classified as chance, decision, or utility.

1. 

A circular *chance node* is a probabilistic node. All nodes were chance nodes in our prior treatment of Bayesian networks.

2. 

A rectangular *decision node* is a node where an agent can and must select from a set of actions.

3. 

A diamond *utility node* is an outcome node. These reflect the costs and benefits of both our choices and the actions that precede them.

Another Weather Decision

Example: Do we bring our Umbrella?

Solution:

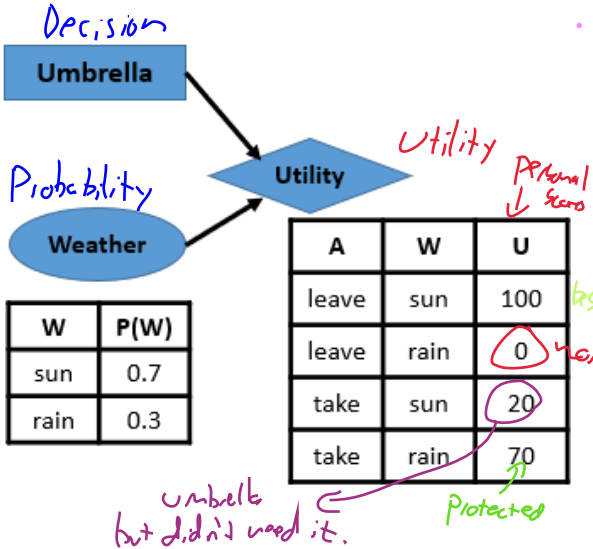
decisions:

E[Utility] of U = Take

E[Utility] of U = leave

Utility of U = Take:

$$\sum_{\text{outcomes}} P(\text{outcome}) \cdot \text{Utility}(\text{outcome})$$



Another Weather Decision

Example: Do we bring our Umbrella?

Solution: $U(take) =$

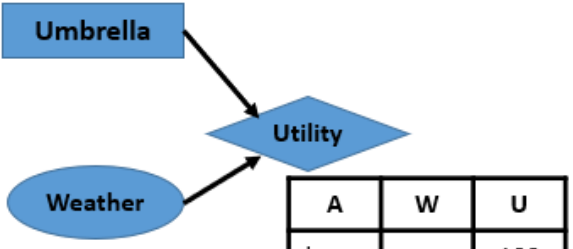
$U(take \,\&\, rain)P(rain) + U(take \,\&\, sun)P(sun)$

$= 70(.3) + 20(.7) = 35.$

$U(leave) =$

$U(leave \,\&\, rain)P(rain) + U(leave \,\&\, sun)P(sun)$

$= 0(.3) + 100(.7) = 70.$



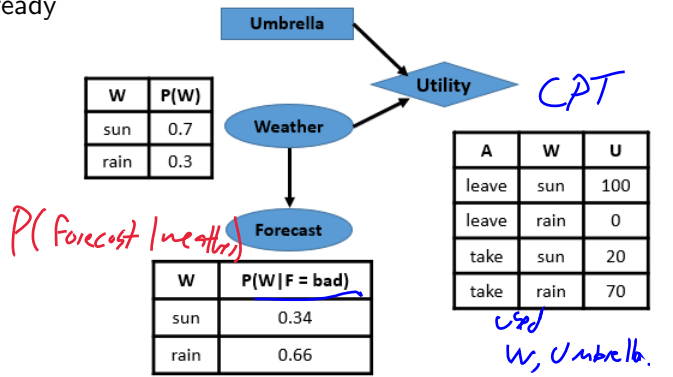
W	P(W)
sun	0.7
rain	0.3

A	W	U
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

Compare Utility(decison), Choose decison w/ max utility

Example: Now the weather is forecast to be bad. Do we bring our Umbrella?

Typically we would model something like $P(forecast \mid weather)$. Suppose we've already reversed that via Bayes' to $P(weather \mid forecast)$.
Now we have:



Example: Now the weather is forecast to be bad. Do we bring our Umbrella?

Typically we would model something like $P(forecast \mid weather)$. Suppose we've already reversed that via Bayes' to $P(weather \mid forecast)$.
Now we have: $U(take) =$

$$U(take \,\&\, rain)P(rain) + U(take \,\&\, sun)P(sun)$$

$$= 70(.66) + 20(.34) = 53.3.$$

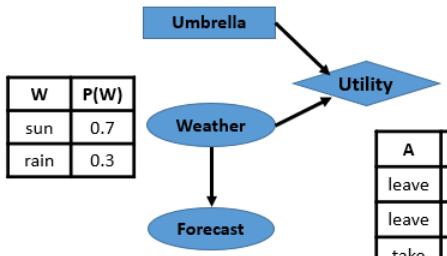
$U(take)$

$U(leave) =$

$$U(leave \,\&\, rain)P(rain) + U(leave \,\&\, sun)P(sun)$$

$$= 0(.66) + 100(.34) = 34.$$

$U(leave)$

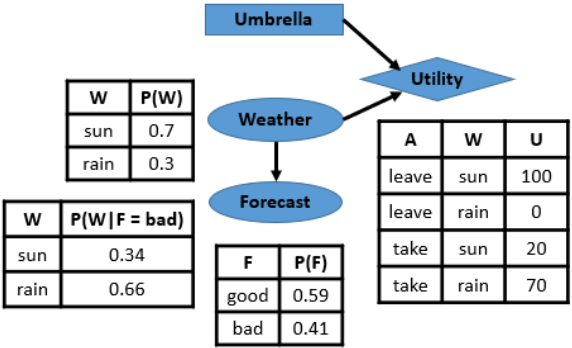


W	P(W)
sun	0.7
rain	0.3

A	W	U
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

W	P(W | F = bad)
sun	0.34
rain	0.66

Decision: Utility of 53.3 gained 19.3 points!



W	P(W)
sun	0.7
rain	0.3

A	W	U
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

W	P(W|F = bad)
sun	0.34
rain	0.66

F	P(F)
good	0.59
bad	0.41

We could even add in the accuracy of the Forecast if we wish! We're just backing up the chain the same as we did with Bayes Nets.

We might think of the decision without observing the forecast as a decision *with uncertainty*. It has an expected value which may differ from the informed decision... which also may differ from the *perfect information* decision that actually knows **weather.**

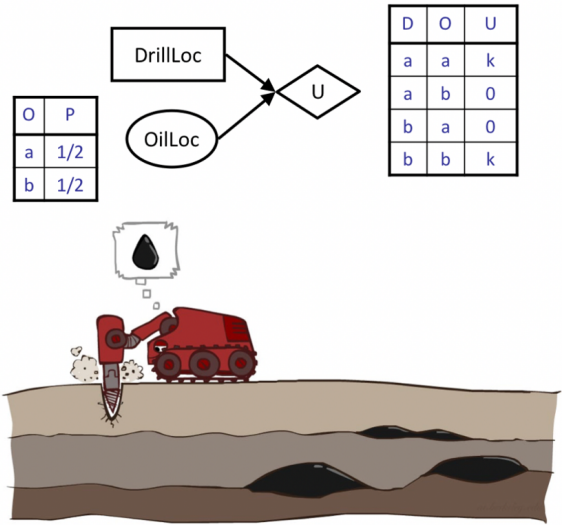
Value of Information

Definition: The *value of information* of a chance node is the increase in expected utility if that node is revealed to us.

1. We can choose different actions with more information
2. Many actions we can take might even include *gathering* information
3. We gather information in hopes that we make better decisions!

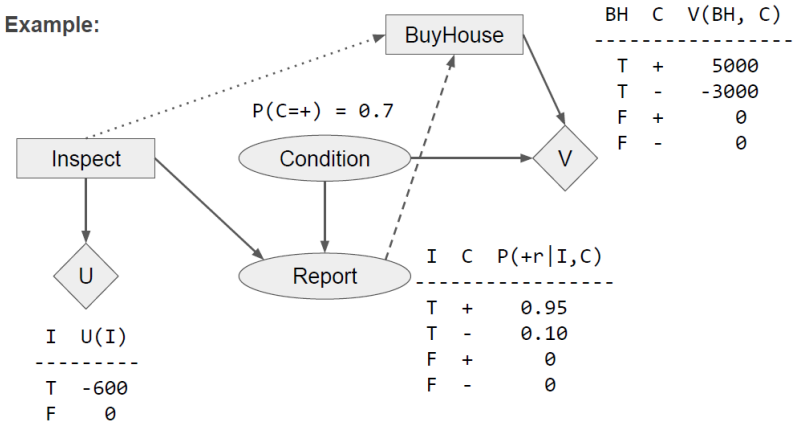
Information itself has (expected) utility!

Example: If we drill in location A, what's the utility? What's the value of the information of where the oil O is?



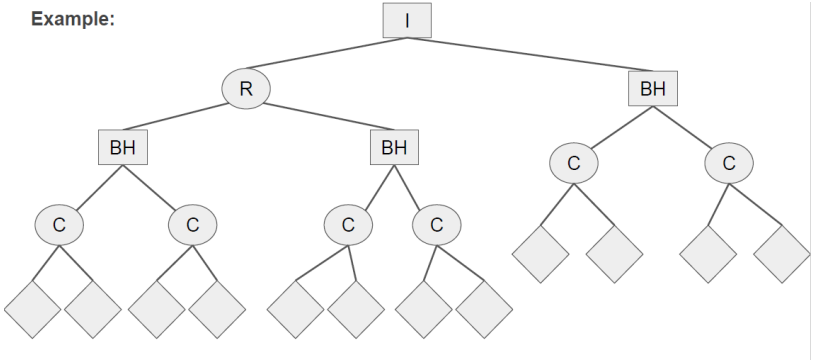
Quick Examples

- The soup of the day is either clam chowder or split pea, but you wouldn't order either one. What's the value of knowing which it is?
- There are two kinds of plastic forks at a picnic. One kind is slightly sturdier. What's the value of knowing which?
- You're playing the lottery. The prize will be $0 or $100. You can play any number between 1 and 100 (chance of winning is 1%). What is the value of knowing the winning number?



Example: Suppose we need to decide *both* whether to buy a house and whether or not to get a house inspection, prior to that.

Generalizing to Sequential Decisions



Example: We might try to represent this as a tree: The I node is a decision either way, but the R node only provides information when $I = True$.
The utility nodes end up as leaves. Here each leaf is *cumulative:* the utility/cost from the I decision as well as the utility from the BH decision.

Next time intro:

Suppose there is a mouse trying to figure out whether it should run out of its hole and eat cheese (E) or do nothing (N).

If the mouse hides, nothing happens but it stays hungry. If the mouse runs out to eat the cheese and the cat attacks, the mouse dies (which has a low utility). Otherwise, if the mouse tries to eat the cheese and the cat doesn't attack, it gets to eat the cheese (high utility).

Whether the cat will attack (A) depends on whether the cat is hungry (H) and whether the cat is sleepy (S). The mouse can observe two things, whether the cat is sleepy (S) and whether the cat has a collar (C). The cat is more often sleepy (S) when it's either full (f) or starved (v) than when it is peckish (p), and the collar (C) tends to indicate that the cat is not starved.

Draw the decision diagram according to this decision-making setup.

Moving Forward

- Coming up:
  1. Reinforcement learning begins!