Computational Models for describing Social Cognition

Written by Aaron Baker

Background information - Special thanks to the UCB CS188 course notes

Bayesian Inference (at its most boiled down) - Etz & Vandekerckhove 2018

- We have some set of assumptions we are working from $\{A_0, \ldots, A_k\}$, let's just say we pick one of them: A
- We also collect some evidence E that this assumption might explain (out of a sample space of possible evidence $\{E_0, \ldots, E_k\}$)
- We want to know **how likely our assumption is given the evidence we've seen**

Posterior Probability	Prior Probability	Likelihood Function	Prior Predictive Prob.
The probability our assumption is true given the observed evidence	The probability our assumption is true a priori	The probability we would observe that evidence if our assumption was true	The probability of seeing that evidence in general

Bayes Rule:

$$P(A|E) = \frac{P(A)\,P(E|A)}{P(E)}$$

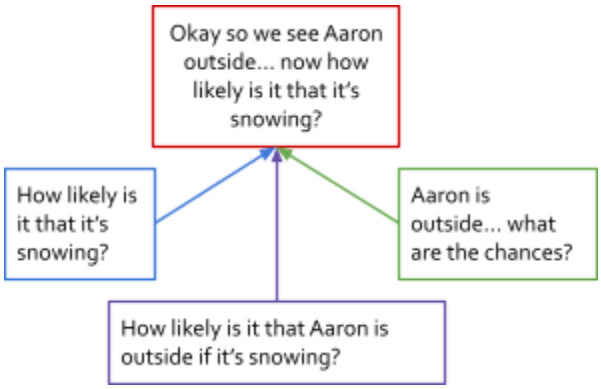
This is the primary equation that is used to derive interesting information, FOR EXAMPLE **how much the evidence E affected our belief**

- We can see how the evidence affected the change prom the prior probability to the posterior probability

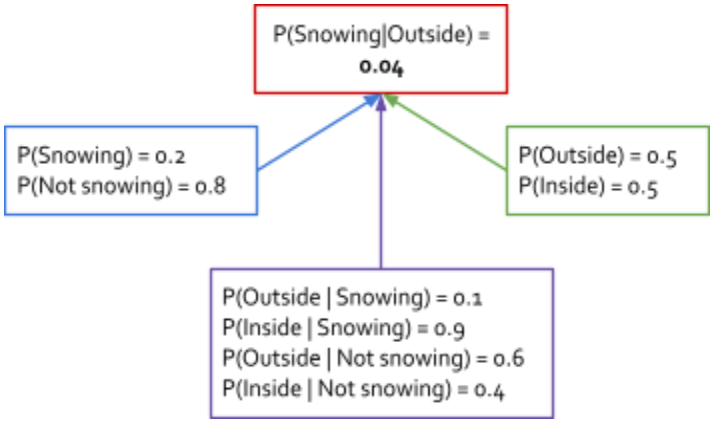
Side note: the stronger the posterior probability, the harder it is for new evidence to sway it

Bayesian inference at its core is using certain probability distributions to solve for other probabilities (or conditional probabilities).

Example: *Aaron goes outside*



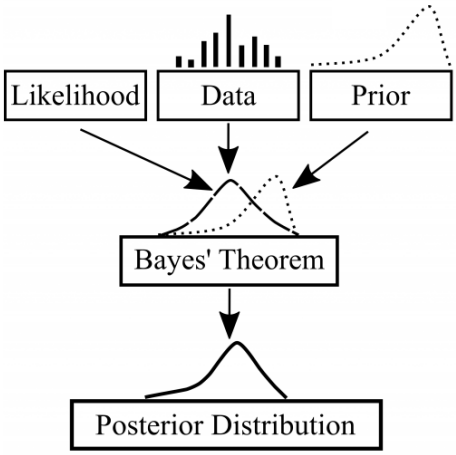
- We've been locked in our room all day and haven't seen the outdoors, we want to know if it's snowing. We have some **prior belief** of how likely it is to be snowing, but we also know this guy (let's call him Aaron) who is afraid of the snow.
- We heard Aaron is outside, let's call that our **evidence**, since we know that if Aaron is unlikely to be outside if it's snowing, what we will consider our **likelihood function**.



- Taken together, this information can give us a new **(posterior) probability** that it is probably not snowing if Aaron is outside.
- Note that depending on the evidence we have, we can make different inferences (if we knew whether it was snowing or not, we could calculate the probability that Aaron is outside)
- *Methodologically* it's important to know where you are getting each variable. This example would be fleshing out **assumptions** of the weather based on **sampled data** of Aaron being outside.

Continuous probabilities

- We mostly deal with discrete events, but this is helpful in the case that we want to know the probability of an event lying within some interval of assumptions
- You might see **PDFs (or probability density functions)** referenced in the literature, which refer to the likelihood that any evidence would lie within a certain **range** of the sample space
- The math gets a little more intense, but the assumptions and interpretations are in the same ballpark



Markov Decision Processes (Modeling Decision making)

In order to decide which kind of model to use, we must first consider what kind of evidence we are observing. Oftentimes we use **observed behavior** to guide our model assumptions.

Once we have a model, we can see how it **responds to new evidence** and how it **predicts behaviors**. Let's start with just modeling those behaviors.

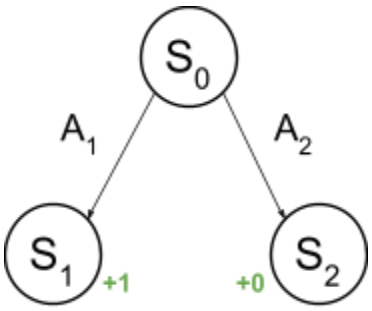
Okay, I'm going to build the model of a gambling agent, James Bot, step by step.

Rules:

- James will be gambling money with heroes and villains. Heroes double the money and villains take all the money.
- The agent only has 2 options, **bet or don't bet**.
- He will play against heroes, who will double the agent's bets, and villains, who will take the bet.

Example: Gambling Agent Part 1

- This model will predict how our agent, James Bot, will place $1 bets when the opponent is certainly a hero.
- We can model this scenario using states (S), actions (A), and utility (U):
  - **Start state: {S_0: before the bet}**
  - **Terminal states: {S_1: hero (with utility 1), S_2: no bet (with utility 0)}**
  - **{A_1: bet, A_2: don't bet}**
- *Note: action 1 will ALWAYS result in state 1*
- We can predict what the agent will do by assigning to it some **policy (π)**, or what action it would take in a given state
- This is where our theoretical assumptions get baked into the model, because we will assume the agent is a utility maximizer
  - This assumption shouldn't be taken for granted when considering these models
- James Bot's optimal policy in this case is always
  - $\pi = A_1$



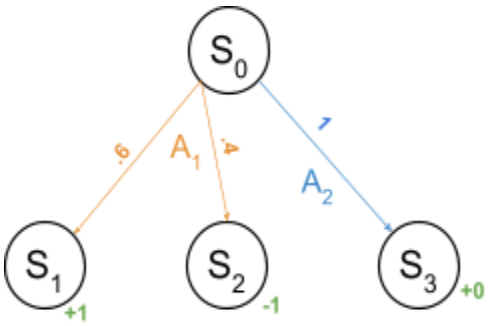
James Bot from here on out will be an agent that **maximizes its expected utility**, only in this case it is certain about its utility.

- For more information on how agent's being utility maximizers can be a theoretically robust assumption for modeling people, you can check out the Naïve Utility Calculus (Jara-Ettinger et al. 2016)

Next, James Bot will have to decide what to do when its actions can have multiple outcomes.

Example: Gambling Agent Part 2

- James Bot is now playing against an opponent and does not know whether the opponent is a hero or villain. The agent is told there is a 60% chance it is a hero and a 40% chance it is a villain.
- James' possible actions remain **{A_1: bet, A_2: don't bet}.**
- Whoever has the higher card takes home the pot. So the resulting state spaces have changed:
  - **Starting state: {S_0: before the bet}**
  - **Terminal States: {S_1: hero (with utility 1), S_2: villain (with utility -1), S_3: no bet (with utility 0)}**
- Now the starting state (S_0) carries with it **uncertainty** of how actions will transition to terminal states. Specifically, the belief in who our opponent is. So we add a **transition function T(s, a, s')** to describe how actions might end up
  - $T(S_0, A_1, S_1) = 0.6$ (belief that opponent is a hero)
  - $T(S_0, A_1, S_2) = 0.4$ (belief that opponent is a villain)
  - $T(S_0, A_2, S_3) = 1$ (always results in S_3)



- These transition functions are always found using probabilities, so in this case is it's useful to think of them as:
  - $T(S_0, A_1, S_1)$ = P(hero)
  - $T(S_0, A_1, S_2)$ = P(villain)
  - $T(S_0, A_2, S_3)$ = P(keep money)
- As a utility maximizer, we can assume our model finds an optimal policy by choosing the action that yields the highest expected utility.
  - The expected value of BET is P(hero)*1 + P(villain)*-1 = **0.2**
  - The expected value of NOT BET is P(keep money)*0 = **0**
  - Therefore the optimal policy is to **BET**, every time
  - *The expected value of taking an action from a given state is called a **Q-value (Q(s, a)).** We'll return to these soon.*

Note: **Markov Means Memoryless**

- Specifically this means that *future and past states are conditionally independent, given the present state*.
- So, if we know the present state, the past doesn't give us any more information about the future. It all gets baked in along the way.
- This is more important for the models we will look at that reach further (recursion!)
- This is our first encounter of something to keep an eye: assumptions of independence may be something people don't do in certain situations, leading to bias (getting "on a roll" in blackjack)
  - This is based on nothing but an intuition I have (I imagine there's work on this though)

Reinforcement Learning

Okay, we're talking about AI, so I know what you're thinking: What's smart about these? Well hold on to your hats because we're going to start getting *smart*.

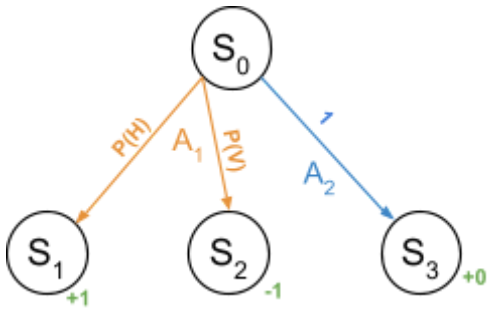
One of the primary features of both humans and AI algorithms is **the ability to adapt their strategies given new information;** and one of the primary uses for bayesian inference in the social sciences is to model **decision making under uncertainty**.

- *Side note: AI as a term does not necessarily entail this, all it has to do is maximize expected utility.*

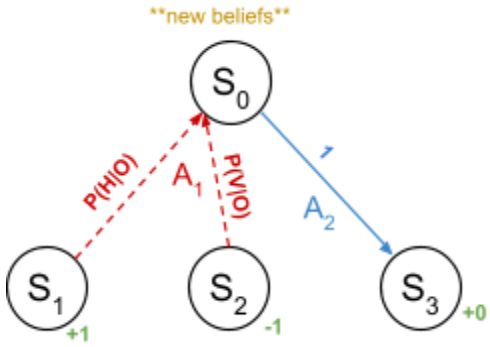
Most of the time there are parameters that we just don't know, and so we **estimate, observe, and re-estimate.** Which values we estimate and the assumptions we make will affect which model we use.

Let's check in with James Bot, who will be playing the same game as before. This time, however, **the agent is not told the likelihood of the opponent being a hero or villain**.

Example: Gambling Agent Part 3



- The agent's model remains the same as before:
  - **{A_1: bet, A_2: don't bet}**
  - **Starting state: {S_0: before the bet}**
  - **Terminal States: {S_1: hero (with utility 1), S_2: villain (with utility -1), S_3: no bet (with utility 0)}**
- This time, instead of calling them transition functions, we'll call them belief functions:
  - b(A_1, S_1) = P(H); belief that the opponent is a hero
  - b(A_1, S_2) = P(V); belief that the opponent is a villain
  - b(A_1, S_2) = P(K); belief that no bet will yield 0 (always 1)



- Because we **don't know the belief distributions,** one option is to initialize some prior belief distribution. This can be based on theoretical assumptions and be manipulated between models. For example, we can make them 50/50 (because why not).
- Then our model will make some decision, and we will observe some **outcome O.**
  - Now, we have an assumption and some observed evidence (sound familiar?)
- **Bayesian Updating:** We can use the observation to **update the belief distributions using Bayes Rule!**
  - Then the posterior distribution will become our new prior distribution.
  - NOTE that the beliefs only update for the action we took.

Parameter Updating

BIG DISCLAIMER: The more you have your model learn from evidence, the more evidence it needs to get good. We're talking big numbers, very fast. Oftentimes in the thousands/tens of thousands of iterations.

- This actually is a guiding concern for a lot of researchers: How do we make sense of novel scenarios given so little evidence.

As we go on, we have an idea of the components that make up these models, and so we'll abandon more specific equations (maybe a couple more) for the sake of high-order structure. Before we do I want to mention an important component of design: **the update functions**.

- Update functions can take many forms, based on the **unknown parameter it's estimating** and the **goal of the model**.
- For example, we just updated belief functions using a **bayesian update,** but that is by no means the only way a model will use new information to update its beliefs.
- ALSO, we are not always looking to update beliefs! Sometimes it's the utility function, other times it's the value of states themselves (which are similar).
    - *Although we are applying these models to social psychology, beliefs tend to be interesting (Lamba et al. 2020, Khalvati et al. 2019, Ray et al. 2008)*

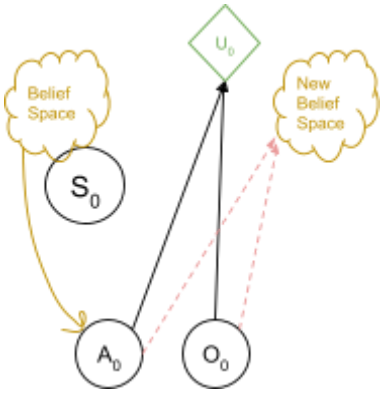
Just to give us an idea let's look at a form of value iteration called **temporal difference learning,** which tries to use experience to better estimate the **value of being in a particular state**. To do this, the model must combine new evidence with it's previous notion of the state's value:

$$V'(s) = (1-\alpha) * V(s) + \alpha * sample$$

Where V'(s) is the new value estimate at state s, V(s) is the current value estimate at state s, and α is the **learning rate.** Effectively what is happening is the model is using α to weight how important the new sample (evidence) should be in forming our new value estimate. The higher α, the more influence the new evidence will have.

The method of parameter updating is important to consider when choosing a model. The update function you choose and the values you assign to its constants (like α) will affect how your model behaves.

POMDP - Partially Observable Markov Decision Process



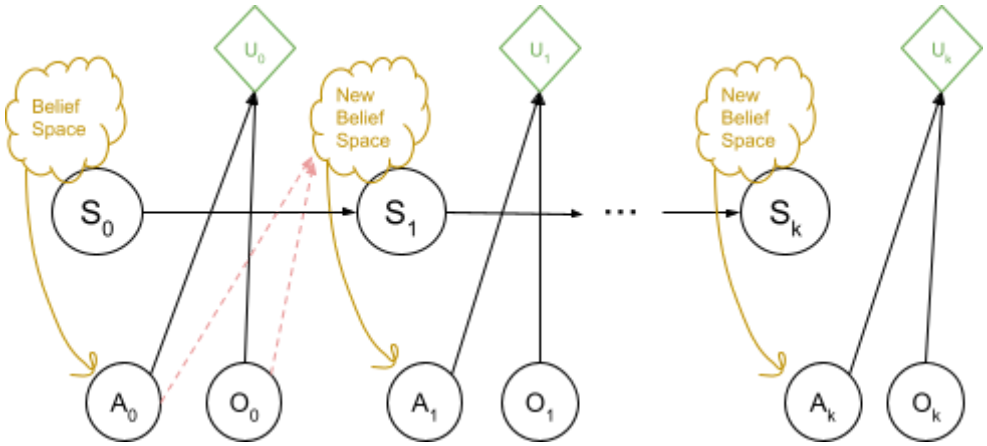
Let's take the model we did earlier and diagram it a little differently, but it's effectively the same as we saw before. Here we have:

- A state (S_0) that we're in, but we are not certain of the circumstances of the state (hero or villain?)
- A belief space consisting of a probability distribution to give us a clue as to our circumstances

- Actions (A_0) we take based on our beliefs (bet or no bet?)
- Observations (O_0) we make of what actually happened (was it a hero or villain?)
- And some utility we gained from those combined things
- Then the action and observations affect what our new beliefs will be the next time we play

This scenario, where we don't *really* know what state we're in and therefore have to make inferences about what state we're *probably* in is called a **partially observable markov decision process.**

Now for a big leap, we will look at how these models can look into future rounds to gain insight as to how they should act now. Consider the following diagram:



This is just the diagram we say before, but it stretches into future iterations of the game. Now our model can think about what *could happen later on*, simply by imagining what could happen if it were to take a particular sequence of actions. This is adding a **search** component to our model.

Search

The quick way to get a good idea of search is to think about chess. To make a descent move, you need to think about all your possible moves. Not only that, you need to think about how each of those will allow your opponent to move, and how each of *their* moves will affect your subsequent moves. This is search, but the problem shows up quickly… eventually there just gets to be too many possibilities.

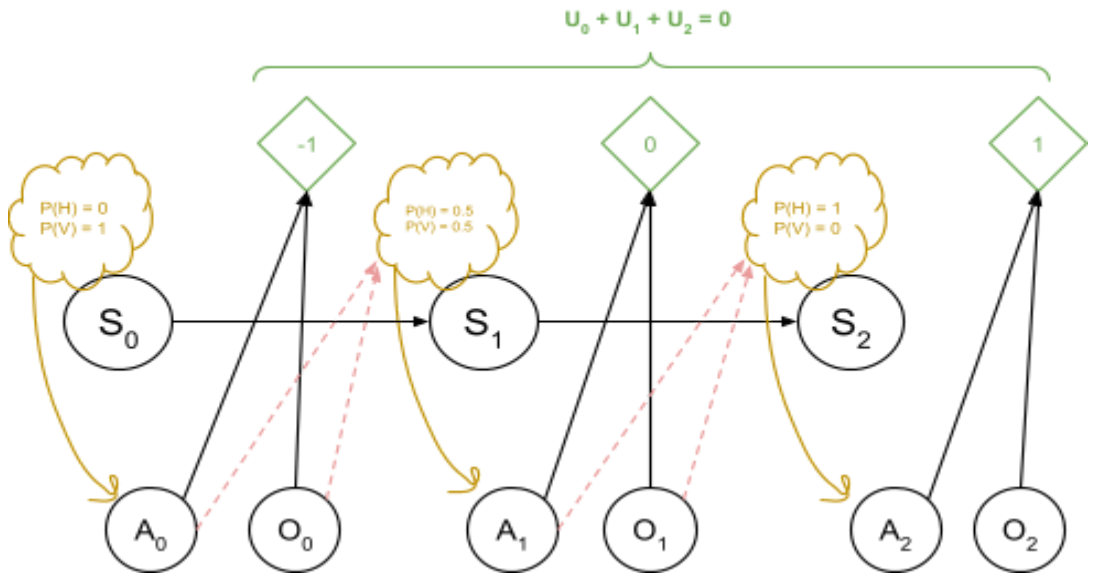
Luckily this is a problem for both humans and computers, and so our models will try and find ways to cope with it.

Example: Gambling Agent Part 3

Our agent (James Bot if you had forgotten) has a slightly new game to play. It will still be placing bets with opponents, but this time there are 2 opponents playing at once and only one chooses the fate of the bet. At the beginning, **both of the opponents are villains**. However, James notices that **each time a bet is placed, one of the opponents turns into a hero**.

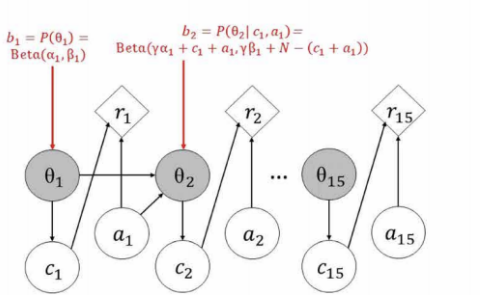
If we kept James as our model from before, it would *always choose no bet*, because placing a bet would certainly result in utility -1 as compared to no bet's 0. So in order to give James a better strategy, we need to give it an idea of the game over many rounds.

Using the same POMDP introduced earlier, let's see what James finds out when it only places bets:



So, only going 2 layers deep, James has found a policy that is *as optimal* as the no bet strategy. It would only need to go one layer deeper to find that it is the *actual optimal policy*.

This kind of POMDP is *very* similar to the one used in Modeling Other Minds (Khalvati et al. 2019) -- Image taken from Figure 3.



***Disclaimer**:* For our agent to actually arrive at this policy, it would need to try every move at every layer which, like chess, can blow up pretty quickly. Because of this, we get to do some more model design!!

Different models might have different **time horizons**, or how far ahead they are willing to look to make a decision. For example, James Bot had a time horizon of 2 -- look 2 steps ahead to consider.

- This is important because some people may use different time horizons for decision making in different scenarios (I know there's work on this but I don't have a paper right now)

This is just one way we can utilize recursive structures to model decision making and behavior. Some may get a little more involved.

Inverse Reinforcement Learning

In social domains, we are often not only concerned with our own behaviors and mechanisms, but also those of the people around us.

We have already seen how our agent's can make inferences about the behaviors of others, this is baked into the **belief functions**. Sure, it seemed like we were just figuring out the odds of there being a hero or villain across from James, but the idea can be extended far! We could have been finding the odds of someone accepting or rejecting an offer, the odds that a dog vomits after eating so much chocolate… but the point is that we were adjusting our beliefs of *occurrences*.

We can extend this further, by trying to make inferences about *motives*. To do this, consider what part of our model dictates motive. For the most part, the **utility function** has the greatest influence over our model's actions (in addition to transition functions, but let's focus on one thing at a time here).

If our model forms beliefs about the world, why not form beliefs about the **utility function** of another model, whose utility function we cannot directly observe. In other words, given some actions we can have our model **infer the utility function of another model.**

This is called **inverse reinforcement learning:** inferring unobservable features of a model (like utility function) based on some observable behaviors.

Example: Gambling Agent Part 4 (Double Agent)

This time, James Bot wants to figure how another agent (let's call it DA) is making decisions. But all James has access to is the decisions that DA makes. So, James makes the *assumption* that DA has a similar structure.

In a turn of events, DA will be making decisions to bet or not bet and James will accept or reject them. James' goal is to **figure out DA's utility function** (the reward for an accepted bet and the cost of a rejected bet). Like the other times, James will begin with a prior belief distribution (but doesn't necessarily have to). Namely, will assume that the utility function is the same as before:

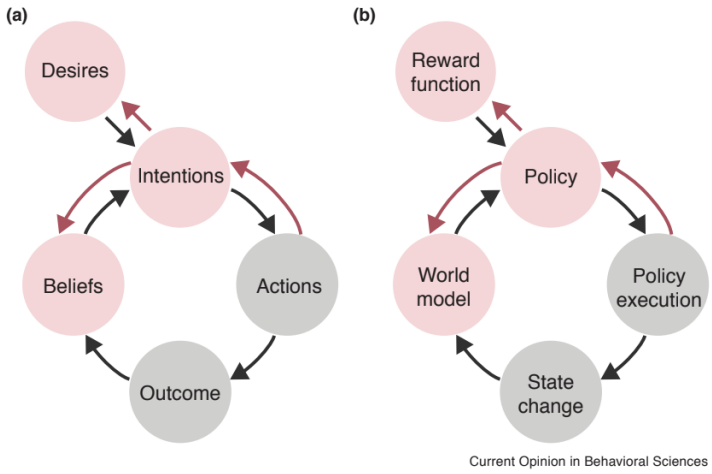
- U(accept) = +1
- U(reject) = -1
- U(no bet) = 0

Now James can accept and reject bets that DA makes, and based on whether or not DA *keeps betting or not*, James can make inferences about DA's utility functions. It might look something like this:

Here, James is using DA's actions to make inferences about its utility function.

This idea has been used for some Theory of Mind models (that are more theoretical than practical):

- Theory of Mind as Inverse Reinforcement Learning (Jara-Ettinger 2019):
    - From Figure 1:



- Pragmatic Language Interpretation as Probabilistic Inference (Goodman & Frank 2016)
    - This one doesn't explicitly mention the model but draws up something similar

Another paper:

Feature Based Reinforcement Learning

Basically manually extracting features of the environment based on which the agent makes decisions. The features are weighted and adjusted based on experience.

Based on this study presented in the comp modeling workshop.