CS 188 Introduction to Artificial Intelligence
Spring 2024 Note 16

Author (all other notes): Nikhil Sharma

Author (Bayes' Nets notes): Josh Hug and Jacky Liang, edited by Regina Wang

Author (Logic notes): Henry Zhu, edited by Peyrin Kao

Credit (Machine Learning and Logic notes): Some sections adapted from the textbook *Artificial Intelligence: A Modern Approach*.

Last updated: August 26, 2023

Particle Filtering

Recall that with Bayes' nets, when running exact inference was too computationally expensive, using one of the sampling techniques we discussed was a viable alternative to efficiently approximate the desired probability distribution(s) we wanted. Hidden Markov Models have the same drawback - the time it takes to run exact inference with the forward algorithm scales with the number of values in the domains of the random variables. This was acceptable in our current weather problem formulation where the weather can only take on 2 values, $W_i \in \{sun, rain\}$, but say instead we wanted to run inference to compute the distribution of the actual temperature on a given day to the nearest tenth of a degree.

The Hidden Markov Model analog to Bayes' net sampling is called **particle filtering**, and involves simulating the motion of a set of particles through a state graph to approximate the probability (belief) distribution of the random variable in question. This solves the same question as the Forward Algorithm: it gives us an approximation of $P(X_N|e_{1:N})$.

Instead of storing a full probability table mapping each state to its belief probability, we'll instead store a list of n **particles**, where each particle is in one of the d possible states in the domain of our time-dependent random variable. Typically, n is significantly smaller than d (denoted symbolically as $n << d$) but still large enough to yield meaningful approximations; otherwise the performance advantage of particle filtering becomes negligible. Particles are just the name for samples in this algorithm.

Our belief that a particle is in any given state at any given timestep is dependent entirely on the number of particles in that state at that timestep in our simulation. For example, say we indeed wanted to simulate the belief distribution of the temperature T on some day i and assume for simplicity that this temperature can only take on integer values in the range $[10, 20]$ ($d = 11$ possible states). Assume further that we have $n = 10$ particles, which take on the following values at timestep i of our simulation:

$$[15, 12, 12, 10, 18, 14, 12, 11, 11, 10]$$

By taking counts of each temperature that appears in our particle list and diving by the total number of particles, we can generate our desired empirical distribution for the temperature at time i:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_i)$	0.2	0.2	0.3	0	0.1	0.1	0	0	0.1	0	0

Now that we've seen how to recover a belief distribution from a particle list, all that remains to be discussed is how to generate such a list for a timestep of our choosing.

Particle Filtering Simulation

Particle filtering simulation begins with particle initialization, which can be done quite flexibly - we can sample particles randomly, uniformly, or from some initial distribution. Once we've sampled an initial list of particles, the simulation takes on a similar form to the forward algorithm, with a time elapse update followed by an observation update at each timestep:

- *Time Elapse Update* - Update the value of each particle according to the transition model. For a particle in state t_i, sample the updated value from the probability distribution given by $P(T_{i+1}|t_i)$. Note the similarity of the time elapse update to prior sampling with Bayes' nets, since the frequency of particles in any given state reflects the transition probabilities.
- *Observation Update* - During the observation update for particle filtering, we use the sensor model $P(F_i|T_i)$ to weight each particle according to the probability dictated by the observed evidence and the particle's state. Specifically, for a particle in state t_i with sensor reading f_i, assign a weight of $P(f_i|t_i)$. The algorithm for the observation update is as follows:

  1. Calculate the weights of all particles as described above.
  2. Calculate the total weight for each state.
  3. If the sum of all weights across all states is 0, reinitialize all particles.
  4. Else, normalize the distribution of total weights over states and resample your list of particles from this distribution.

  Note the similarity of the observation update to likelihood weighting, where we again downweight samples based on our evidence.

Let's see if we can understand this process slightly better by example. Define a transition model for our weather scenario using temperature as the time-dependent random variable as follows: for a particular temperature state, you can either stay in the same state or transition to a state one degree away, within the range $[10,20]$. Out of the possible resultant states, the probability of transitioning to the one closest to 15 is 80% and the remaining resultant states uniformly split the remaining 20% probability amongst themselves.

Our temperature particle list was as follows:

$$[15,12,12,10,18,14,12,11,11,10]$$

To perform a time elapse update for the first particle in this particle list, which is in state $T_i = 15$, we need the corresponding transition model:

T_{i+1}	14	15	16
$P(T_{i+1}\|T_i = 15)$	0.1	0.8	0.1

In practice, we allocate a different range of values for each value in the domain of T_{i+1} such that together the ranges entirely span the interval $[0,1)$ without overlap. For the above transition model, the ranges are as follows:

1. The range for $T_{i+1} = 14$ is $0 \leq r < 0.1$.
2. The range for $T_{i+1} = 15$ is $0.1 \leq r < 0.9$.
3. The range for $T_{i+1} = 16$ is $0.9 \leq r < 1$.

In order to resample our particle in state $T_i = 15$, we simply generate a random number in the range $[0,1)$ and see which range it falls in. Hence if our random number is $r = 0.467$, then the particle at $T_i = 15$ remains in $T_{i+1} = 15$ since $0.1 \leq r < 0.9$. Now consider the following list of 10 random numbers in the interval $[0,1)$:

$$[0.467, 0.452, 0.583, 0.604, 0.748, 0.932, 0.609, 0.372, 0.402, 0.026]$$

If we use these 10 values as the random value for resampling our 10 particles, our new particle list after the full time elapse update should look like this:

$$[15, 13, 13, 11, 17, 15, 13, 12, 12, 10]$$

Verify this for yourself! The updated particle list gives rise to the corresponding updated belief distribution $B(T_{i+1})$:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_{i+1})$	0.1	0.1	0.2	0.3	0	0.2	0	0.1	0	0	0

Comparing our updated belief distribution $B(T_{i+1})$ to our initial belief distribution $B(T_i)$, we can see that as a general trend the particles tend to converge towards a temperature of $T = 15$.

Next, let's perform the observation update, assuming that our sensor model $P(F_i|T_i)$ states that the probability of a correct forecast $f_i = t_i$ is 80%, with a uniform 2% chance of the forecast predicting any of the other 10 states. Assuming a forecast of $F_{i+1} = 13$, the weights of our 10 particles are as follows:

Particle	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
State	15	13	13	11	17	15	13	12	12	10
Weight	0.02	0.8	0.8	0.02	0.02	0.02	0.8	0.02	0.02	0.02

Then we aggregate weights by state:

State	10	11	12	13	15	17
Weight	0.02	0.02	0.04	2.4	0.04	0.02

Summing the values of all weights yields a sum of 2.54, and we can normalize our table of weights to generate a probability distribution by dividing each entry by this sum:

State	10	11	12	13	15	17
Weight	0.02	0.02	0.04	2.4	0.04	0.02
Normalized Weight	0.0079	0.0079	0.0157	0.9449	0.0157	0.0079

The final step is to resample from this probability distribution, using the same technique we used to resample during the time elapse update. Let's say we generate 10 random numbers in the range $[0,1)$ with the following values:

$$[0.315, 0.829, 0.304, 0.368, 0.459, 0.891, 0.282, 0.980, 0.898, 0.341]$$

This yields a resampled particle list as follows:

$$[13, 13, 13, 13, 13, 13, 13, 15, 13, 13]$$

With the corresponding final new belief distribution:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_{i+1})$	0	0	0	0.9	0	0.1	0	0	0	0	0

Observe that our sensor model encodes that our weather prediction is very accurate with probability 80%, and that our new particles list is consisistent with this since most particles are resampled to be $T_{i+1} = 13$.

Utilities

Throughout our discussion of rational agents, the concept of utility came up repeatedly. In games, for example, Utility values are generally hard-wired into the game, and agents use these utility values to select an action. We'll now discuss what's necessary in order to generate a viable utility function.

Rational agents must follow the **principle of maximum utility** - they must always select the action that maximizes their expected utility. However, obeying this principle only benefits agents that have **rational preferences**. To construct an example of irrational preferences, say there exist 3 objects, A, B, and C, and our agent is currently in possession of A. Say our agent has the following set of irrational preferences:

- Our agent prefers B to A plus \$1
- Our agent prefers C to B plus \$1
- Our agent prefers A to C plus \$1

A malicious agent in possession of B and C can trade our agent B for A plus a dollar, then C for B plus a dollar, then A again for C plus a dollar. Our agent has just lost \$3 for nothing! In this way, our agent can be forced to give up all of its money in an endless and nightmarish cycle.

Let's now properly define the mathematical language of preferences:

- If an agent prefers receiving a prize A to receiving a prize B, this is written $A \succ B$
- If an agent is indifferent between receiving A or B, this is written as $A \sim B$
- A **lottery** is a situation with different prizes resulting with different probabilities. To denote lottery where A is received with probability p and B is received with probability $(1-p)$, we write

$$L = [p, A;\ (1-p), B]$$

In order for a set of preferences to be rational, they must follow the five **Axioms of Rationality**:

- *Orderability*: $(A \succ B) \vee (B \succ A) \vee (A \sim B)$
  A rational agent must either prefer one of A or B, or be indifferent between the two.
- *Transitivity*: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
  If a rational agent prefers A to B and B to C, then it prefers A to C.

- *Continuity*: $A \succ B \succ C \Rightarrow \exists p\,[p, A;\ (1-p), C] \sim B$
  If a rational agent prefers A to B but B to C, then it's possible to construct a lottery L between A and C such that the agent is indifferent between L and B with appropriate selection of p.

- *Substitutability*: $A \sim B \Rightarrow [p, A;\ (1-p), C] \sim [p, B;\ (1-p), C]$
  A rational agent indifferent between two prizes A and B is also indifferent between any two lotteries which only differ in substitutions of A for B or B for A.

- *Monotonicity*: $A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A;\ (1-p), B] \succeq [q, A;\ (1-q), B]$
  If a rational agent prefers A over B, then given a choice between lotteries involving only A and B, the agent prefers the lottery assigning the highest probability to A.

If all five axioms are satisfied by an agent, then it's guaranteed that the agent's behavior is describable as a maximization of expected utility. More specifically, this implies that there exists a real-valued **utility function** U that when implemented will assign greater utilities to preferred prizes, and also that the utility of a lottery is the expected value of the utility of the prize resulting from the lottery. These two statements can be summarized in two concise mathematical equivalences:

$$U(A) \geq U(B) \quad \Leftrightarrow \quad A \succeq B \tag{1}$$
$$U([p_1, S_1;\ \dots\ ; p_n, S_n]) \quad = \quad \sum_i p_i U(S_i) \tag{2}$$

If these constraints are met and an appropriate choice of algorithm is made, the agent implementing such a utility function is guaranteed to behave optimally. Let's discuss utility functions in greater detail with a concrete example. Consider the following lottery:

$$L = [0.5, \$0;\ 0.5, \$1000]$$

This represents a lottery where you receive \$1000 with probability 0.5 and \$0 with probability 0.5. Now consider three agents A_1, A_2, and A_3 which have utility functions $U_1(\$x) = x$, $U_2(\$x) = \sqrt{x}$, and $U_3(\$x) = x^2$ respectively. If each of the three agents were faced with a choice between participting in the lottery and receiving a flat payment of \$500, which would they choose? The respective utilities for each agent of participating in the lottery and accepting the flat payment are listed in the following table:

Agent	Lottery	Flat Payment
1	500	500
2	15.81	22.36
3	500000	250000

These utility values for the lotteries were calculated as follows, making use of equation (2) above:

$$U_1(L) \quad = \quad U_1([0.5, \$0;\ 0.5, \$1000]) = 0.5 \cdot U_1(\$1000) + 0.5 \cdot U_1(\$0) = 0.5 \cdot 1000 + 0.5 \cdot 0 = \boxed{500}$$
$$U_2(L) \quad = \quad U_2([0.5, \$0;\ 0.5, \$1000]) = 0.5 \cdot U_2(\$1000) + 0.5 \cdot U_2(\$0) = 0.5 \cdot \sqrt{1000} + 0.5 \cdot \sqrt{0} = \boxed{15.81}$$
$$U_3(L) \quad = \quad U_1([0.5, \$0;\ 0.5, \$1000]) = 0.5 \cdot U_3(\$1000) + 0.5 \cdot U_3(\$0) = 0.5 \cdot 1000^2 + 0.5 \cdot 0^2 = \boxed{500000}$$

With these results, we can see that agent A_1 is indifferent between participating in the lottery and receiving the flat payment (the utilities for both cases are identical). Such an agent is known as **risk-neutral**. Similarly, agent A_2 prefers the flat payment to the lottery and is known as **risk-averse** and agent A_3 prefers the lottery to the flat payment and is known as **risk-seeking**.