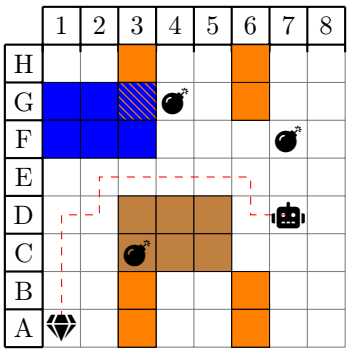
Project : Bayesian Inverse Reinforcement Learning

This project aims to code in Python a Bayesian approach to the Inverse Reinforcement Learning (IRL) problem. In the IRL problem, we try to learn the reward function R optimized by an agent called the tutor, given a partially described infinite horizon MDP $\langle S, A, T, \gamma \rangle$ (partially described because the reward function is not given), and observed state-action pairs $O = \{(s_i, a_i) | i \in [M]\}$ (with $M \in \mathbb{N}$) realized by the tutor. This project is inspired by the following two papers [1, 2]. We will use the following gridworld problem as running example :



where :
- S is the set compounded of the different cells, where each cell is identified by a letter in {A,B,C,D,E,F,G,H} and a number in $\{1, 2, 3, 4, 5, 6, 7, 8\}$. In the above example the initial state of the agent is D7 and A1 is a final state.
- $A = \{L, R, U, D\}$ is compounded of the four actions going *right*, *left*, *up*, or *down*.
- T, in each cell, if the agent tries to go in a direction, let's say left, this will succeed with probability 0.8. Yet, with probability 0.2 the agent slips and go in a perpendicular direction (i.e., here up with probability 0.5 and down with probability 0.5).
- γ is set to 0.9.

The dotted line in red represents the state-action pairs realized by the tutor, i.e., $O = \{(D7, L), (D6, U), (E6, L), (E5, L), (E4, L), (E3, L), (E2, D), (D2, L), (D1, D), (C1, D), (B1, D)\}$. Note that the tutor may provide demonstrations starting from different starting points.

Assumptions on the Reward function. Let $\Phi = \{\phi_1, \ldots, \phi_t\}$ be a set of binary features. We assume a function $\varphi : S \longrightarrow \{0, 1\}^{|\Phi|}$ which assigns to each state s the binary vector indicating the features that are present in state s. We assume that the reward $R(s)$ is a weighted sum of these features. Stated differently, we assume a reward function $R : \Phi \longrightarrow [-R_{\max}, R_{\max}]$ such that

$$R(s) = \varphi(s) \cdot (R(\phi_1), \ldots, R(\phi_t)),$$

where $R_{\max} \in \mathbb{R}^+$ is a parameter bounding the reward function (it can be set arbitrarily to 1). This assumption will help lower the dimensionality of the reward vector investigated.

In the gridworld shown above the features in Φ would be :
- ϕ_1 Treasure : if a jewel is present in the cell.
- ϕ_2 Bomb : if a bomb is present in the cell.
- ϕ_3 Mud : when the cell is orange.
- ϕ_4 Water : when the cell is blue.

— ϕ_5 Mountain : when the cell is brown.

For instance, for cells $G3$ and $C3$, we have $\varphi(G3) = (0, 0, 1, 1, 0)$, $\varphi(C3) = (0, 1, 0, 0, 1)$, $R(G3) = R(\phi_3) + R(\phi_4)$, and $R(C3) = R(\phi_2) + R(\phi_5)$.

1. Code a python class to represent a partially defined MDP.
2. Instantiate the gridworld running example.
3. Provide a method to create a similar but randomly generated gridworld MDP, where the size of the grid and the number of cells with each feature can be parameterized.
4. Code a policy iteration algorithm, to solve such an MDP if a reward function is provided.
5. Provide a method which, given a reward function R, a number M of timesteps, and an initial state s_0, provides a set O of M state-action pairs starting from s_0. To simulate an imperfect tutor, the optimal action will be chosen with probability 0.95, and a random action is chosen otherwise.

The Bayesian framework. The idea is that the tutor is behaving according to a close to optimal stationary policy. Hence, if she has performed action a_i in state s_i, than $Q^*(s_i, a_I)$ should have a high value compared to the ones in $\{Q^*(s_i, a)|a \in A\}$.

This assumption leads to the following conditional probability of observing the demonstrations of the tutor given a possible reward function :

$$\mathbb{P}(O|R) = \prod_{i=1}^{M} \mathbb{P}((s_i, a_i)|R) = \prod_{i=1}^{M} \frac{exp(\alpha Q^*(s_i, a_i, R))}{\sum_{a \in A} exp(\alpha Q^*(s_i, a, R))}.$$

where α is a parameter denoting how much we trust the tutor to behave optimally. Note that :

— if $\alpha = 0$, we assume the tutor is acting randomly.

— if α is large, the probability values will only be non negligible for optimal actions. Hence, we would assume the tutor is behaving (close) to optimally.

Using *Bayes rule*, we obtain :

$$\mathbb{P}(R|O) \propto \mathbb{P}(O|R)\mathbb{P}(R).$$

With $\mathbb{P}(R|O)$ and $\mathbb{P}(R)$ the *posterior* and *prior* probability distributions over the reward and $\mathbb{P}(O|R)$ is the *likelihood* function.

Several prior functions can be investigated on the reward function like :

— *Uniform* : Uniform prior on all reward functions R such that $\max_{\phi \in \Phi}\{|R(\phi)|\} \leq R_{\max}$.

— *Gaussian* : A Gaussian prior such that $\mathbb{P}(R) = \prod_{\phi \in \Phi} \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{R(\phi)^2}{2\sigma^2})$.

— *Beta* : A Beta distribution such that $\mathbb{P}(R) = \prod_{\phi \in \Phi} \frac{1}{\frac{R(\phi)}{R_{\max}}^{\frac{1}{2}}(1-\frac{R(\phi)}{R_{\max}})^{\frac{1}{2}}}$.

— Possibly a mixture of the previous priors or a more application-specific one.

6 Provide a method to compute the ratio $\mathbb{P}(R_1|O)/\mathbb{P}(R_2|O)$ given R_1, R_2, O, and a choice for the prior function. This is a meta step which will require to provide several classes and methods.

An MCMC algorithm to investigate the posterior distribution. To estimate $\mathbb{P}(R|O)$, one can use a Monte-Carlo Markov-Chain algorithm which relies on a sampling approach. The idea is to perform a random walk in the reward function space where the probability to move from one reward function R to a nearby reward function R' depends on the ratio $\mathbb{P}(R'|O)/\mathbb{P}(R|O)$. If one performs a very long random walk using such algorithm, then the frequency of occurrence of each reward function R will match her probability $\mathbb{P}(R|O)$.

To "walk" in the reward space $\mathcal{R}$ we discretize it using a grid of stepsize δ (where δ is a parameter of the approach). Stated differently,

$$\mathcal{R} = \{-R_{max}, -(\lceil \frac{R_{max}}{\delta} \rceil - 1)\delta, \ldots, -\delta, 0, \delta, \ldots, (\lceil \frac{R_{max}}{\delta} \rceil - 1)\delta, R_{max}\}^t.$$

In $\mathcal{R}$, two reward functions R and R' are neighbors if they differ only in one coordinate and of an amount of δ. For instance $(r_1, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_t)$ and $(r_1, \ldots, r_{i-1}, r_i + \delta, r_{i+1}, \ldots, r_t)$ are neighbors.

The MCMC algorithm used in [2] is the following PolicyWalk algorithm.

Algorithme 1 : PolicyWalk(Distribution P, MDP M, Step Size δ)

Pick a random reward vector $R \in \mathcal{R}$;
$\pi :=$ `PolicyIteration`(M,R) ;
repeat
  (a) Pick a reward vector R' uniformly at random from the neighbours of R in $\mathcal{R}$
  (b) Compute $Q^\pi(s, a, R')$ for all $(s, a) \in S \times A$
  (c) **if** $\exists (s, a) \in S \times A$, $Q^\pi(s, \pi(s), R') < Q^\pi(s, a, R')$ **then**
    i. $\pi' :=$ `PolicyIteration`(M, R' ,π) // Starting from policy π
    ii. Set $R := R'$ and $\pi := \pi'$ with probability $\min\{1, \mathbb{P}(R'|O)/\mathbb{P}(R|O)\}$
  end
  else
    i. Set $R := R'$ with probability $\min\{1, \mathbb{P}(R'|O)/\mathbb{P}(R|O)\}$
  end
until *A given number of iterations has been reached*
return R

This algorithm can be used :
— to estimate the distribution $\mathbb{P}(R|O)$ by counting the number of time each reward function appears ;
— to return a reward function R_{map} with high posterior probability by using a current maximum to store the reward function with highest a posteriori probability encountered during the walk.

7 Code the PolicyWalk algorithm to return a reward with high a posteriori probability.

Following [1], this algorithm can be modified using a decreasing "cooling" schedule function T. This modification consists in accepting the new reward function with probability $\min\{1, (\mathbb{P}(R'|O)/\mathbb{P}(R|O))^{\frac{1}{T_i}}\}$ instead of $\min\{1, \mathbb{P}(R'|O)/\mathbb{P}(R|O)\}$. This modification inspired from the simulated annealing algorithm makes it possible to concentrate the search over areas of high posterior probability. The idea is that T_i should first be a "high value" such that the probability of moving towards a neighbor will be higher. Then, the temperature should decrease. With low temperature, the algorithm will accept only moves toward high posterior probability rewards.

8 (Bonus) Provide such a modified version of your PolicyWalk algorithm.

To test your approach, you will use the method of question 5) to obtain a set of observations O. Using your PolicyWalk algorithm, you will obtain a reward R_{map} with optimal policy π_{map}. The performance of the algorithm will be computed using a 0-1 policy loss. This loss function incurs a cost of one each time a state-action pair in O is not optimal with respect to π_{map}.

9 Perform experiments on gridworld-like MDPs of varying characteristics. You will obtain a variety of plots giving the performance of your algorithms with different choices of parameters as a function of some other parameters (e.g., size of the MDP, number of features, number of iterations in the PolicyWalk algorithm, ...).

For more advanced students.

10 (Bonus) Propose improvements to the Bayesian IRL approach.

The Bayesian IRL approach makes it possible to learn a probable reward function given the demonstrations provided by the tutor. Active learning considers the problem of asking the tutor for new demonstrations. The idea is to ask the tutor for a demonstration in an area of the state space where the policy remains uncertain.

11 (Bonus) Design an active learning procedure using your ideas.

Rendering. The rendering of your project must include a report and an archive of your code. Your report will present your work. Notably it will present the experiments done for question 9. It will also contain the answers to questions 10 and 11, if you had time to address them. You will also mention any source or website that has been used to perform the project.

Références

[1] Bernard Michini and Jonathan P How. Improving the efficiency of bayesian inverse reinforcement learning. In *2012 IEEE International Conference on Robotics and Automation*, pages 3651–3656. IEEE, 2012.

[2] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.