R04/Sarthak Mittal/200050129

January 22, 2023

This paper discusses **inverse reinforcement learning (IRL)**, where the optimal actions are given and the reward function needs to be found for the system. They discuss both cases, where the **entire policy** is given and where we only have a **finite set** of **trajectories**. To handle degeneracy (having set of reward functions, all of which make the policy optimal), they use a **linear programming** formulation to pick a function which best differentiates the optimal policy from others. According to the paper, these algorithms could serve as models for animal and human learning, particularly when **multi-attribute** rewards are involved. It could also help in **apprenticeship learning**, such as "driving", where the expert's policy could be used to generate reward function for imitation by apprentice.

Their method relies on "Bellman Equations" and "Bellman Optimality" in **Markov Decision Processes (MDPs)**. They first formulate the **finite** state space case, and characterize the reward function using an inequality that is necessary and sufficient for the optimal policy to be unique. To choose the reward function from the feasible set, they add constraint to **penalize** deviation from optimal and **maximize** "sum of difference between quality of optimal action and next-best action". They added a **weight decay-like** penalty also, in order to have simpler (non-zero in a few states) reinforcements.

For the case where the state space is **infinite**, they assume that the value function of a policy can be approximated, and formulate a **linear approximation** for the reward function. The inequality is replaced by an **expectation maximization**, and is applied only over a large finite **subset** of states.

For the most general case (access to policy through a set of trajectories), they formulate it as (i) choosing a state distribution $\mathbf{D}$ and start state $\mathbf{s_0}$ (ii) maximizing $\mathbf{E_{s_0 \sim D}}[\mathbf{V}^{\pi}(\mathbf{s_0})]$. The linear approximation for reward carries over. For estimating the value function, they take an average of empirical return over m **Monte Carlo simulations** under the policy. The iterative algorithm proceeds by first finding **value function** using the given policy, and then finds the coefficients for the **reward function** using linear programming, and then finds **policy** that maximizes value function under reward, and continues.

They also bring forth a few questions about potential-based shaping rewards, noise in measurement of actions (or in its optimality), "locally consistent" rewards, "identifiability" and extension to partially observable environments.