Machine Learning

Markov Decision Processes

Alberto Maria Metelli and Francesco Trovò

Markov Decision Process

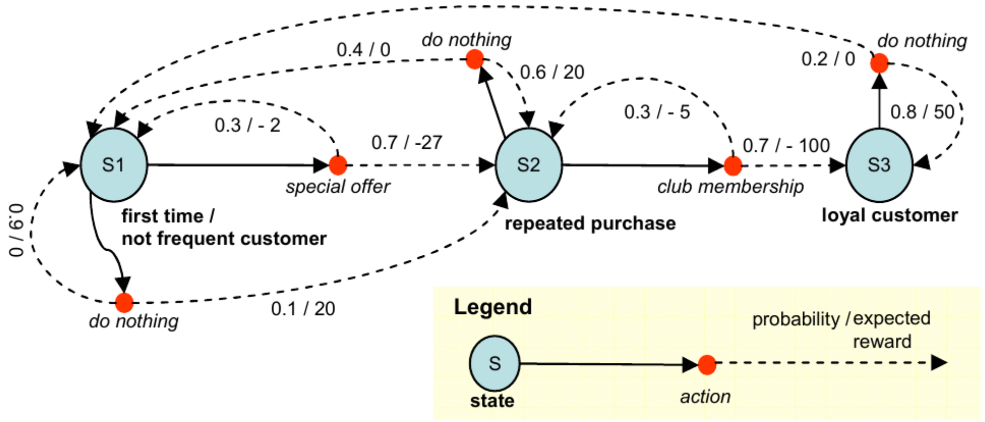
Two different problems

We want model the dynamics of a process and the possibility to choose among different actions in each situation

Two different problems:

- **Prediction**: given a specific behaviour (policy) in each situation, *estimate the expected long-term reward* starting from a specific state
- **Control**: learn the optimal behaviour to follow in order to *maximize the expected long-term reward* provided by the underlying process

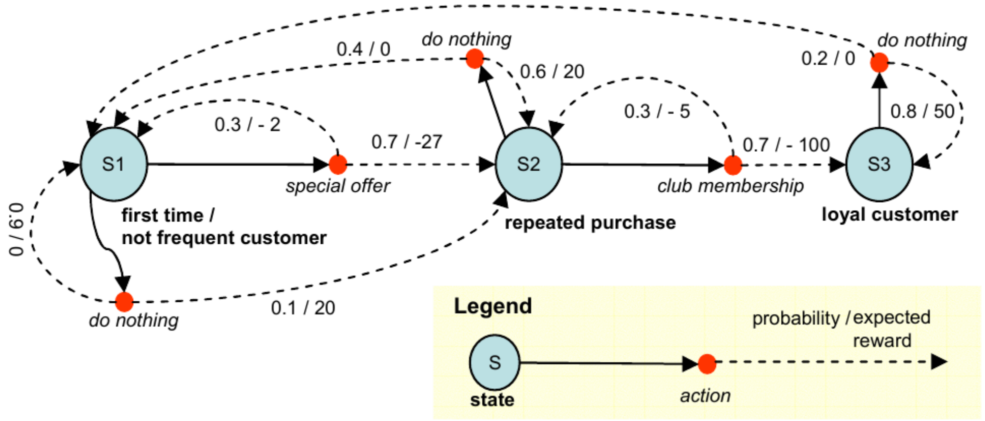
Example: Advertising Problem



- **Prediction**: given the actions in each state ($S1$, $S2$, $S3$) compute the value of a state
- **Control**: determine the best action in each state

Prediction

Prediction on the Advertising Problem



Given the policy (do nothing, do nothing, do nothing), compute the value of each state

Modeling the MDP

First, we model the MDP $\mathcal{M} := (\mathcal{S}, \mathcal{A}, P, R, \mu, \gamma)$ for the given problem:

- States: $\mathcal{S} = \{\text{first time}, \text{repeated purchaser}, \text{loyal customer}\}$
- Actions: $\mathcal{A} = \{\text{do nothing}, \text{special offer}, \text{club membership}\}$
- Transition model: $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, we need $\dim(P) = |\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ numbers to store it
- Reward function: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we need $\dim(R) = |\mathcal{S}||\mathcal{A}|$ numbers to store it
- Initial distribution $\mu \in \Delta(\mathcal{S})$, we need $\dim(\mu) = |\mathcal{S}|$ numbers to store it
- Discount factor: $\gamma \in (0, 1]$

where $\Delta(\cdot)$ represents the simplex over a set

We assume that all the customer are first timers $\mu = (1, 0, 0)$ and use $\gamma = 0.9$

Modeling the MDP

The agent's behavior is modeled by means of a **policy**:

$$\pi : \mathcal{S} \to \Delta(\mathcal{A})$$

Once we select a specific policy $\pi(a|s)$, P^π and R^π are defined as:

$$P^\pi(s'|s) = \sum_{a \in \mathcal{A}} \pi(a|s) P(s'|s,a) \qquad \dim(P^\pi) = |\mathcal{S}| \times |\mathcal{S}|$$

$$R^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) R(s,a) \qquad \dim(R^\pi) = |\mathcal{S}|$$

Computing the Value of the States

We have the Bellman expectation equation:

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t) | s_0 = s \right] = \sum_{a \in \mathcal{A}} \pi(a|s) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\pi(s') \right]$$
$$= R^\pi(s) + \gamma \sum_{s' \in \mathcal{S}} P^\pi(s'|s) V^\pi(s')$$

which we can rewrite in matrix form as:

$$V^\pi = R^\pi + \gamma P^\pi V^\pi \qquad \dim(V^\pi) = |\mathcal{S}|$$

Alternative 1: Closed-Form Solution

Thanks to the Bellman expectation equation:

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

Since P^π is a stochastic matrix, we have that the eigenvalues of $(I - \gamma P^\pi)$ are in $[1-\gamma, 1]$ for $\gamma \in [0, 1)$ and the matrix is invertible!

- Inverting matrix $(I - \gamma P^\pi)^{-1}$ costs $O(|\mathcal{S}|^3)$ with straightforward algorithm

Alternative 2: Recursive Solution

If we are not able to invert the matrix (the state space is too large), let us consider the recursive version of the Bellman expectation equation:

$$V^\pi = R^\pi + \gamma P^\pi V^\pi$$

```
V_old = np.zeros(nS)
tol = 0.0001
V = pi @ R_sa
while np.any(np.abs(V_old - V) > tol):
    V_old = V
    V = pi @ (R_sa + gamma * P_sas @ V)
```

Evaluating Different Policies

By changing the policy, which in matrix form is

$$\pi(a|s) = \Pi(s, a|s) \qquad \dim(\Pi) = |\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$$

we are able to compute the values of the states with different strategies:

- **myopic:** we do not want to spend any money in marketing

```
pi_myo = np.array([[1., 0., 0., 0., 0.],
                   [0., 0., 1., 0., 0.],
                   [0., 0., 0., 0., 1.]])
```

- **far-sighted:** we want to spend some money in marketing for the customer in both cases if she is a new customer or if she repeatedly purchased

```
pi_far = np.array([[0., 1., 0., 0., 0.],
                   [0., 0., 0., 1., 0.],
                   [0., 0., 0., 0., 1.]])
```

Results with Different Discounts

	$\gamma = 0.5$		$\gamma = 0.9$		$\gamma = 0.99$	
π	myopic	far-sighted	myopic	far-sighted	myopic	far-sighted
$V^\pi(S_1)$	**5.3333**	-47.6202	**36.3636**	-9.2889	396.0396	**785.3831**
$V^\pi(S_2)$	**18.6667**	-59.9347	**54.5455**	20.1890	415.8416	**824.8548**
$V^\pi(S_3)$	**67.5556**	58.7300	**166.2338**	136.8857	569.3069	**939.9320**

- For $\gamma = 0.5$ the myopic policy evidently outperforms the far-sighted one
- For $\gamma = 0.9$ the two policies are getting close
- For $\gamma = 0.99$ the far-sighted policy becomes the most rewarding one

Control

Select the Policy

- **Brute force**: enumerate all the possible policies, evaluate their values and consider the one having the maximum values
  - There exists a **deterministic** optimal policy
  - Requires evaluating $|\mathcal{A}|^{|\mathcal{S}|}$ policies
- **Dynamic Programming**
  - **Policy Iteration**: iteratively evaluate the current policy and update it in the greedy direction
  - **Value Iteration**: iteratively apply the Bellman optimality equation in its recursive form
    - we cannot solve the Bellman optimality equation in a closed form since the $\max$ operator is not linear!

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|a,s) V^*(s') \right\}$$

Policy Iteration

- Repeat until convergence:
    1. **policy evaluation**, where we compute the value V^{π_k} of the given policy π_k (as seen before)
    2. **policy improvement**, where we change the policy from π_k to π_{k+1} according to the newly estimated values (**greedy improvement**)

$$\begin{aligned}\pi_{k+1}(s) &= \arg\max_{a \in \mathcal{A}} Q^{\pi_k}(s,a) \\ &= \arg\max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|a,s) V^{\pi_k}(s') \right\} \quad \forall s \in \mathcal{S}\end{aligned}$$

- Guaranteed to converge to π^* in a **finite** number of steps!

Value Iteration

- Directly evaluate the optimal policy directly, i.e., compute $V^*(s)$
- Repeated application of the Bellman optimality equation:

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_k(s') \right\}$$

- Once we have $V^*(s)$, we can easily recover the optimal policy, i.e., the greedy one w.r.t. $V^*(s)$
- Guaranteed to converge to $V^*(s)$ **asymptotically**