1 Introduction

Consider a finite discrete Markov decision process (MDP) $\mathcal{M} = (\mathbf{s}, \mathbf{a}, \mathbf{P}, \mathbf{R}, \boldsymbol{\mu}, H)$ containing set of S states $s \in \mathbf{s}$, set of A actions $a \in \mathbf{a}$, state transition kernel $\mathbf{P}$, reward function $\mathbf{R}$, initial state distribution $\boldsymbol{\mu}$, and horizon H. The dynamics of an environment described by MDP is characterized by its state transition kernel $\mathbf{P} = \{\mathbf{P}^t\}_{t\in[H]}$ and immediate reward $\mathbf{R} = \{\mathbf{R}^t\}_{t\in[h]}$, where $\mathbf{P}^t : \mathbf{s} \times \mathbf{a} \times \mathbf{s} \to [0, 1]$ and $\mathbf{R}^t : \mathbf{s} \times \mathbf{a} \to [0, 1]$ are the state transition probability and corresponding reward in t-th time step over all trajectories. The policy $\pi(\cdot|s)$ gives the probability of choosing each action in state s. The expected cumulative reward for each state-action (s, a) pair according to policy π is given by $\mathbf{Q}^t_\pi(s, a) = \mathbb{E}_\pi[\sum_{i=t}^{H} \mathbf{R}^i(s^i, a^i)|s^t = s, a^t = a]$, and the total expected reward $J^\pi = \mathbb{E}_\pi[\sum_{t=1}^{H} \mathbf{R}^t(s^t, a^t)|s^1 \sim \boldsymbol{\mu}]$.

In offline reinforcement learning, the agent has access to the offline data generated from an unknown behavior policy π_β, which is a combination of a human expert, an existing system, and a random policy performing the task. Using this offline data agent's goal is to learn a target policy π_θ that generalizes the behavior pattern. Given offline data and a learned target policy π_θ, this work estimates the total expected reward $\widehat{J}$ when the MDP has a latent low-rank structure. Particularly, we assume that the state-action values of a $\mathbf{Q}$ matrix at each time step are low rank.

2 Algorithm

2.1 Data generation process

We consider a uniform state transition kernel and a reward value for each state-action pair uniformly sampled from [0,10] at each time step, and also assume that the initial state distribution is uniform over all states as defined in algorithm 2. We consider a policy π, which chooses an action uniformly from the subset of actions $\mathbf{a}_m \subseteq \mathbf{a}$, where the subset $\mathbf{a}_m$ is chosen uniformly at random amongst all subsets of size m as mentioned in algorithm 3. We assume both π_β and π_θ follows the same policy model. Following this process, we generate K trajectories each of length H as given in algorithm 1.

2.2 Policy evaluation algorithm

Given a dataset $\mathcal{D} = \{(s^t_k, a^t_k, r^t_k)\}_{t\in[H],k\in[K]}$, target policy π_θ, and the initial state distribution μ, we estimate the total expected reward $\widehat{J}$ by finding the state-action values ($\mathbf{Q}$ value) in backward from time step $t = H$ to $t = 1$ as described in algorithm 4. In each time step, the algorithm finds state-action pair visitation count n^t and estimates the state transition kernel $\widehat{\mathbf{P}}^t$ as given in algorithm 5. After that on the support of n^t, the state-action values $\mathbf{Z}^t$ are calculated using $\widehat{\mathbf{P}}^t$ based on the Bellman update, given by

$$\mathbf{Z}^t(s, a) = r^t(s, a) + \sum_{s',a'} \widehat{\mathbf{P}}^t(s'|a, s)\pi_\theta(a'|s')\widehat{\mathbf{Q}}^{t+1}_{\pi_\theta}(s', a') \;\forall\; (s, a) \in supp(n^t) \tag{1}$$

where $r^t(s, a)$ is the immediate reward for a state-action pair (s, a). Subsequently, to infer the $\mathbf{Q}$ value off-support, the algorithm solves the following optimization problem:

$$\underset{\mathbf{M}\in\mathbb{R}^{S\times A}}{\text{minimize}} \; \frac{1}{2}\|\mathbf{M}_{supp(n^t)} - \mathbf{Z}^t_{supp(n^t)}\|^2_F + \lambda\|\mathbf{M}\|_* \tag{2}$$

where λ is the regularization parameter. The algorithm to solve the optimization problem (2) is given in 6.

2.3 Simulations for policy evaluation algorithm

We consider a uniform state transition model with five states and four actions as in Table 1. We generate a dataset of five trajectories of length five as mentioned in Table 2. Based on this data, we estimate the state transition kernel $\mathbf{P}^t$ as given in Table 3. We found state-action values $\mathbf{Z}^t$ using 200 data trajectories without the matrix estimation step. The estimated state-action values are close to true, as given in Table 4. Table 4 justifies that we can accurately estimate the state transition kernel and corresponding state-action values with more trajectories. We considered 30 and 20 data trajectories

Algorithm 1: dataset generation

Input: S, A, K, H, m.
Output: $dataset\ \mathcal{D}$

$\mathcal{D}$ = [ ]
$(\mathbf{s}, \mathbf{a}, \mathbf{P}, \mathbf{R}, \boldsymbol{\mu})$ = MDP(S, A, H)
for $k \leftarrow 1$ ***to*** K **do**
  τ = [ ]
  s^0 = np.random.choice(**s**, p=$\boldsymbol{\mu}$)
  $i = 0$
  while $i < H$ **do**
    $a = \texttt{policy}(\mathbf{a}, m)$
    $r = \mathbf{R}[i][s^0, a]$
    s^1 = np.random.choice(**s**, p=$\mathbf{P}[i][s^0, a]$)
    τ.append((s^0, a, r))
    $s^0 \leftarrow s^1$
    i += 1
  end
  $\mathcal{D}$.append(τ)
end

Algorithm 2: Markov decision process

MDP(S, A, H)
s = np.arange(S)
a = np.arange(A)
$\boldsymbol{\mu}$ = np.ones(S)/S
$reward\ level$ = 10
R = [ ]
P = [ ]
for $i \leftarrow 1$ ***to*** H **do**
  R.append(np.random.rand(S, A)$*reward\ level$)
  P.append(np.ones((S, A, S))/S)
end
return $\mathbf{s}, \mathbf{a}, \mathbf{P}, \mathbf{R}, \boldsymbol{\mu}$

and found the state-action values with matrix estimation for each time step given in Table 5 and 6, respectively. From Table 5 and 6, we infer that the estimated state-action values are comparable with the true state-action values in each time step. We can also observe that the accurate matrix estimation step at every time step is crucial for finding the overall state-action values.

2.4 Simulations for matrix completion step

We generate a $m \times n$ low-rank matrix $\mathbf{M}$ of rank r as $\mathbf{M} = \mathbf{U}\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are of size $m \times r$ and $n \times r$ respectively with each entry being sampled from a uniform distribution of $[0, 10]$. We generate a mask $\mathbf{Z}$ of size $m \times n$ with each entry sampled from a Bernoulli distribution of

Algorithm 3: policy

policy($\mathbf{a}, m$)
$\mathbf{a}_m[\,]$ = list(itertools.combinations($\mathbf{a}, m$))
rnd = np.random.choice(len($\mathbf{a}_m$))
$\mathbf{a}_m$ = $\mathbf{a}_m[rnd]$
a = np.random.choice($\mathbf{a}_m$, 1)
return a

Algorithm 4: policy evaluation using matrix estimation

Input: dataset $\mathcal{D}$, π_θ, $\boldsymbol{\mu}$.

Output: $\widehat{J}$

Initialize $\widehat{\mathbf{Q}}_{\pi^\theta}^{H+1} \leftarrow 0_{S\times A}$
for $t \leftarrow H$ ***to*** 1 **do**
  $\{n^t, \widehat{\mathbf{P}}^t\} \leftarrow$ STKE($\mathcal{D}$, t, len($\mathcal{D}$))
  Ω = np.nonzero(n_t)
  for $(s,a) \in \Omega$ **do**
    $\sum = 0$
    for $(s',a') \in \Omega$ **do**
      if $(s',a') \neq (s,a)$ **then**
        $\sum$ += $\frac{1}{n^t(s,a)} \widehat{\mathbf{P}}^t(s,a,s')\pi_\theta(a'|s')\widehat{\mathbf{Q}}_{\pi_\theta}^{t+1}(s',a')$
      end
    end
    $\mathbf{Z}^t(s,a) \leftarrow r^t(s,a) + \sum$
  end
  $\widehat{\mathbf{Q}}_{\pi_\theta}^t \leftarrow$ ME($\mathbf{Z}^t$)
end
$\widehat{J} \leftarrow \sum_{s,a} \boldsymbol{\mu}(s)\pi_\theta(a|s)\widehat{\mathbf{Q}}_{\pi_\theta}^1(s,a)$

Algorithm 5: state transition kernel estimation

STKE(data, t, K)
Initialize $n^t \leftarrow 0_{S\times A}$
Initialize $\widehat{\mathbf{P}}^t \leftarrow 0_{S\times A\times S}$
for $k \leftarrow 1$ ***to*** K **do**
  $(s,a,s^{'}) =$ data(k,t)
  $n^t(s,a)$ += 1
  $\widehat{\mathbf{P}}^t(s,a,s^{'})$ += 1
end
$\widehat{\mathbf{P}}^t(s,a,s^{'}) \leftarrow \widehat{\mathbf{P}}^t(s,a,s^{'})/n^t(s,a)\ \forall\ (s,a) \in supp(n^t)$
return $\{n^t, \widehat{\mathbf{P}}^t\}$

probability p. We collect masked observations by applying a mask $\mathbf{Z}$ on the low-rank matrix $\mathbf{M}$ as $\mathbf{X} = \mathbf{Z} \circ \mathbf{M}$ and then added a masked noise sampled from Gaussian with -2 mean and variance one to the observations $\mathbf{X}$. From the noisy measurements $\mathbf{X}$, we estimate the low-rank matrix $\mathbf{M}$ using the nuclear norm minimization algorithm given in 6. To measure the performance, we use the normalized Forbenius norm error $\frac{\|\mathbf{M}-\widehat{\mathbf{M}}\|_F^2}{\|\mathbf{M}\|_F^2}$. We generate a 1000×1000 matrix of rank 5, a mask with probability $p = 0.1$, and collected the noisy observations $\mathbf{X}$ as defined above. Then, we estimate the low-rank matrix $\widehat{\mathbf{M}}$ using $\mathbf{X}$ and evaluate the normalized error as shown in Figure 2. The plot shows that the error converges to zero with iterations. We evaluate the above process over 20 realizations and observe the average normalized error as 3.15×10^{-2}.

References

[1] Xi, Xumei, Christina Lee Yu, and Yudong Chen. "Matrix Estimation for Offline Reinforcement Learning with Low-Rank Structure." arXiv preprint arXiv:2305.15621 (2023).

Algorithm 6: Nuclear norm minimization for matrix estimation

ME($\mathbf{Z}$)
Variable η
Set $K = 300$
λ = η*np.linalg.norm($\mathbf{Z}$,2)
$support$ = np.nonzero($\mathbf{Z}$)
$\mathbf{A}$ = np.zeros($\mathbf{Z}$.shape)
$\mathbf{B}$ = np.ones($\mathbf{Z}$.shape)
$\mathbf{A}[support] = 1$
$\mathbf{B}$ -= $\mathbf{A}$
$\mathbf{M}_{old}$ = np.zeros($\mathbf{Z}$.shape)
for $k \leftarrow 1$ ***to*** K **do**
  $\mathbf{Y} = \mathbf{Z} + \mathbf{B} * \mathbf{M}_{old}$
  $\mathbf{U}$, s, $\mathbf{V}$ = np.linalg.svd($\mathbf{Y}$, full matrices=False)
  s = np.maximum(s $- \lambda$, 0)
  $rank$ = sum(s > 0.0)
  $\mathbf{U} = \mathbf{U}$[:,:$rank$]
  $\mathbf{s} = \mathbf{s}$[:$rank$]
  $\mathbf{V} = \mathbf{V}$[:$rank$,:]
  $\mathbf{M}_{new} = \mathbf{U}$@np.diag(s)@$\mathbf{V}$
  $\mathbf{M}_{old} = \mathbf{M}_{new}$
end
return $\mathbf{M}_{new}$

Table 1: Uniform transition.

$\mathbf{s}$	0	1	2	3	4
$\mathbf{a}$	0	1	2	3	
$\mathbf{P}^t(\cdot\|s,a)$	0.2	0.2	0.2	0.2	0.2
$\mathbf{R}^t(s,\cdot)$	3.0	5.1	7.6	7.8	
$\boldsymbol{\mu}$	0.2	0.2	0.2	0.2	0.2
$\pi(\cdot\|s)$	0.25	0.25	0.25	0.25	

Table 2: dataset containing tuples in the form of (state,[action], next state, reward).

τ_1	(2, [3], 1, 5.38)	(1, [1], 1, 6.19)	(1, [2], 0, 9.68)	(0, [0], 4, 1.33)	(4, [3], 2, 8.33)
τ_2	(2, [3], 2, 5.38)	(2, [2], 4, 6.41)	(4, [1], 2, 5.12)	(2, [1], 3, 6.33)	(3, [3], 3, 6.79)
τ_3	(4, [0], 4, 9.62)	(4, [3], 3, 4.09)	(3, [0], 2, 1.86)	(2, [2], 0, 3.64)	(0, [2], 0, 4.09)
τ_4	(2, [3], 0, 5.38)	(0, [3], 3, 1.41)	(3, [0], 4, 1.86)	(4, [3], 4, 4.86)	(4, [0], 2, 9.13)
τ_5	(4, [3], 0, 4.85)	(0, [1], 0, 5.36)	(0, [1], 4, 8.54)	(4, [1], 2, 3.13)	(2, [1], 3, 0.61)

Table 3: STKE output.

(a) $\mathbf{n}^1$

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	3
3	0	0	0	0
4	1	0	0	1

(b) $\mathbf{P}^1(\cdot|2,a)$

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	1	1	1	0	0

(c) $\mathbf{P}^1(\cdot|4,a)$

	0	1	2	3	4
0	0	0	0	0	1
1	0	0	0	0	0
2	0	0	0	0	0
3	1	0	0	0	0

Table 4: state action (Q) values without matrix estimation over 200 trajectories.

(a) $\mathbf{Q}$ values using MDP($J = 23.82$)

	0	1	2	3
0	21.67	23.85	26.09	26.74
1	27.15	20.50	21.26	23.72
2	26.02	20.50	20.39	23.83
3	21.29	26.21	21.48	21.52
4	28.40	27.99	24.30	23.57

(b) $\mathbf{Q}_1$values using data($\widehat{J} = 23.66$)

	0	1	2	3
0	21.59	24.59	26.46	26.82
1	27.82	20.39	20.32	23.56
2	25.88	20.27	20.04	24.04
3	19.83	26.91	20.14	21.59
4	27.01	26.83	24.33	24.88

Table 5: state action (Q) values with matrix estimation over 30 trajectories.

(a) $\mathbf{Q}$ values using MDP

	0	1	2	3
0	2.71	9.45	4.09	2.67
1	8.63	4.61	7.00	4.53
2	7.45	0.61	0.66	2.96
3	6.12	0.13	3.82	6.79
4	9.13	2.57	1.03	8.33

	0	1	2	3
0	5.87	6.18	5.54	10.8
1	13.7	9.70	7.68	9.52
2	14.2	10.9	8.28	14.2
3	12.1	5.05	14.3	7.85
4	6.99	7.67	11.0	9.12

	0	1	2	3
0	14.8	17.7	10.2	12.5
1	18.6	18.2	18.8	9.58
2	17.6	16.7	11.5	9.87
3	10.8	16.8	18.3	17.0
4	11.0	14.2	9.81	14.3

	0	1	2	3
0	20.2	18.9	23.9	15.2
1	23.2	19.7	20.7	16.9
2	19.5	21.1	20.2	23.1
3	22.8	20.0	16.4	20.4
4	16.6	15.0	19.6	17.8

	0	1	2	3
0	21.6	23.8	26.0	26.7
1	27.1	20.5	21.2	23.7
2	26.0	20.5	20.3	23.8
3	21.2	26.2	21.4	21.5
4	28.4	27.9	24.3	23.5

(b)
$\mathbf{Q}$ values using data on support

	0	1	2	3
0	2.71	0	4.09	0
1	0	0	7.00	0
2	7.45	0.61	0.66	2.96
3	6.12	0	0	6.79
4	9.13	2.57	0	8.33

	0	1	2	3
0	5.27	0	1.84	8.32
1	11.7	0	0	7.53
2	12.6	6.76	5.40	10.0
3	0	3.30	0	0
4	5.69	5.82	0	6.57

	0	1	2	3
0	10.6	12.5	0	7.39
1	0	15.3	14.3	0
2	13.8	12.7	6.88	0
3	6.4	0	14.2	13.3
4	4.3	11.2	3.34	7.75

	0	1	2	3
0	0	8.94	14.9	8.94
1	20.2	14.2	12.0	10.3
2	0	15.0	13.0	0
3	14.6	0	9.60	11.7
4	7.98	7.05	10.5	11.3

	0	1	2	3
0	11.0	0	17.1	19.3
1	16.8	0	7.66	0
2	15.9	13.3	0	13.2
3	10.8	17.1	12.1	12.3
4	17.2	16.5	13.6	2.9

(c) $\mathbf{Q}$ values using data with `ME`

	0	1	2	3
0	2.68	0.36	4.01	1.29
1	1.23	0.04	6.85	0.09
2	7.29	0.65	0.68	3.01
3	6.10	2.09	0.27	6.66
4	9.06	2.49	0.77	8.24

	0	1	2	3
0	5.30	5.42	1.91	8.08
1	11.5	5.15	4.94	7.54
2	12.5	6.76	5.24	9.98
3	3.16	3.17	1.22	3.70
4	5.69	5.59	2.20	6.57

	0	1	2	3
0	10.4	12.4	6.89	7.40
1	9.48	15.2	14.1	12.6
2	13.5	12.6	6.89	6.31
3	6.44	14.1	14.0	13.0
4	4.46	11.0	3.51	7.61

	0	1	2	3
0	9.05	8.99	14.4	8.97
1	19.8	14.2	12.0	10.4
2	15.4	14.6	13.0	9.57
3	14.4	10.1	9.66	11.4
4	8.06	7.03	10.5	10.9

	0	1	2	3
0	11.1	16.6	16.9	18.9
1	16.4	9.49	7.73	6.71
2	15.7	13.4	12.6	13.0
3	10.8	16.6	12.2	12.2
4	17.0	16.3	13.3	13.0

Table 6: state action (Q) values with matrix estimation over 20 trajectories.

(a) $\mathbf{Q}$ values using MDP

	0	1	2	3
0	2.71	9.45	4.09	2.67
1	8.63	4.61	7.00	4.53
2	7.45	0.61	0.66	2.96
3	6.12	0.13	3.82	6.79
4	9.13	2.57	1.03	8.33

	0	1	2	3
0	5.87	6.18	5.54	10.8
1	13.7	9.70	7.68	9.52
2	14.2	10.9	8.28	14.2
3	12.1	5.05	14.3	7.85
4	6.99	7.67	11.0	9.12

	0	1	2	3
0	14.8	17.7	10.2	12.5
1	18.6	18.2	18.8	9.58
2	17.6	16.7	11.5	9.87
3	10.8	16.8	18.3	17.0
4	11.0	14.2	9.81	14.3

	0	1	2	3
0	20.2	18.9	23.9	15.2
1	23.2	19.7	20.7	16.9
2	19.5	21.1	20.2	23.1
3	22.8	20.0	16.4	20.4
4	16.6	15.0	19.6	17.8

	0	1	2	3
0	21.6	23.8	26.0	26.7
1	27.1	20.5	21.2	23.7
2	26.0	20.5	20.3	23.8
3	21.2	26.2	21.4	21.5
4	28.4	27.9	24.3	23.5

(b)

$\mathbf{Q}$ values using data on support

0	1	2	3
2.71	0	4.09	0
0	0	7.00	0
0	0.61	0.66	2.96
6.12	0	0	6.79
9.13	0	0	8.33

0	1	2	3
4.57	0	1.37	6.62
11.4	0	0	0
10.2	6.76	5.70	10.3
0	2.14	0	0
4.53	4.99	0	7.50

0	1	2	3
0	11.5	0	7.24
0	0	12.2	0
12.5	11.4	5.05	0
5.35	0	12.1	12.0
0	10.7	1.97	6.39

0	1	2	3
0	7.17	0	8.30
0	10.9	0	8.92
0	13.4	12.8	0
15.3	0	9.41	10.9
6.87	5.64	9.80	11.3

0	1	2	3
3.55	0	12.8	13.6
13.6	0	0	0
9.88	7.70	0	8.65
0	15.1	10.2	7.45
11.6	0	0	9.75

(c) $\mathbf{Q}$ values using data with `ME`

0	1	2	3
2.69	0.38	4.02	1.50
2.24	0.22	6.86	0.19
2.89	0.53	0.66	2.92
6.14	1.18	0.19	6.64
8.98	1.52	2.66	8.32

0	1	2	3
4.54	3.92	1.50	6.47
11.2	4.56	4.66	7.69
10.1	6.71	5.53	10.2
2.16	2.04	1.22	3.09
4.56	4.87	2.52	7.40

0	1	2	3
7.71	11.3	2.88	7.22
3.12	2.32	12.0	8.84
12.2	11.3	5.04	7.38
5.37	7.78	12.0	11.7
6.90	10.5	2.01	6.34

0	1	2	3
3.58	7.10	7.82	8.17
1.89	10.7	9.23	8.87
4.34	13.3	12.6	12.6
15.0	2.74	9.37	10.9
6.91	5.76	9.68	11.1

0	1	2	3
3.62	11.2	12.6	13.4
13.3	2.92	1.53	6.39
9.78	7.62	5.98	8.58
2.83	14.9	10.1	7.51
11.5	5.40	5.30	9.62