Machine Learning (2021) 110:2603–2640
https://doi.org/10.1007/s10994-021-05969-w

Partially observable environment estimation with uplift inference for reinforcement learning based recommendation

Wenjie Shang[1] · **Qingyang Li**[1] · **Zhiwei Qin**[1] · **Yang Yu**[2] · **Yiping Meng**[1] · **Jieping Ye**[1]

Received: 15 May 2020 / Revised: 6 February 2021 / Accepted: 4 March 2021 /
Published online: 14 April 2021

Abstract
Reinforcement learning (RL) aims at searching the best policy model for decision making, and has been shown powerful for sequential recommendations. The training of the policy by RL, however, is placed in an environment. In many real-world applications, the policy training in the real environment can cause an unbearable cost due to the exploration. Environment estimation from the past data is thus an appealing way to release the power of RL in these applications. The estimation of the environment is, basically, to extract the causal effect model from the data. However, real-world applications are often too complex to offer fully observable environment information. Therefore, quite possibly there are unobserved variables lying behind the data, which can obstruct an effective estimation of the environment. In this paper, by treating the hidden variables as a hidden policy, we propose a *partially-observed multi-agent environment estimation* (POMEE) approach to learn the partially-observed environment. To make a better extraction of the causal relationship between actions and rewards, we design a *deep uplift inference network* (DUIN) model to learn the causal effects of different actions. By implementing the environment model in the DUIN structure, we propose a *POMEE with uplift inference* (POMEE-UI) approach to generate a partially-observed environment with a causal reward mechanism. We analyze the effect of our method in both artificial and real-world environments. We first use an artificial recommender environment, abstracted from a real-world application, to verify the effectiveness of POMEE-UI. We then test POMEE-UI in the real application of Didi Chuxing. Experiment results show that POMEE-UI can effectively estimate the hidden variables, leading to a more reliable virtual environment. The online A/B testing results show that POMEE can derive a well-performing recommender policy in the real-world application.

Keywords Reinforcement learning · Environment estimation · Hidden state · Uplift modeling · Recommender system

Editors: Yuxi Li, Alborz Geramifard, Lihong Li, Csaba Szepesvari, Tao Wang.

✉ Wenjie Shang
shangwenjie@didiglobal.com

Extended author information available on the last page of the article

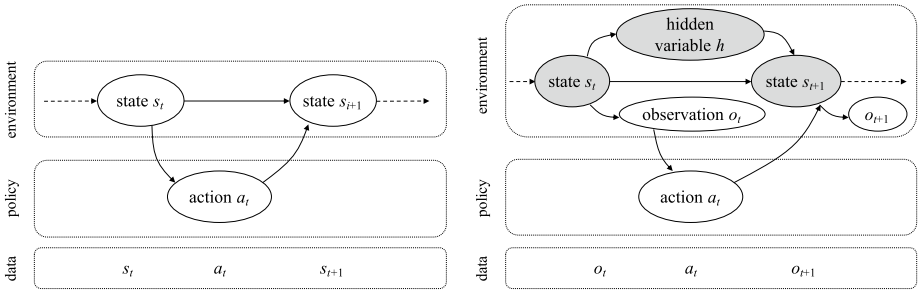
1 Introduction

In sequential recommendation problems (Ye et al. 2018, 2019), where the system needs to recommend multiple items to the user while responding to the user's feedback, there are multiple decisions to be made in sequence. For example, in our application of program recommendation to taxi drivers on the large-scale ride-hailing platform, the system recommends a personalized driving program to each driver, and a program consists of multiple steps, where each step is recommended according to how the previous steps were followed. Therefore, recommending the program steps is a sequential decision problem, and it can be naturally tackled by reinforcement learning (RL) (Sutton and Barto 2018).

As a powerful tool for learning decision-making policies, RL learns from interactions with the environment via trial-and-errors (Sutton and Barto 2018). In digital worlds where interactions with the environment are feasible and cheap, it has made remarkable achievements, (e.g., Mnih et al. 2015; Silver et al. 2016; Brown and Sandholm 2017; OpenAI et al. 2019). When it comes to real-world applications, physical environments in the real world are no longer as convenient as digital environments. It is not practical to interact with the real-world environment directly for training the policy, because of the high interaction cost, the potential unbearable risk and the huge amount of interactions required by the current RL techniques. A recent study (Shi et al. 2018) disclosed a viable option to conduct RL on real-world tasks, which is by estimating a virtual environment from the past data. Once a virtual environment is built, the RL process could be more efficient by interacting with it, and the physical cost in real-world environments could be avoided as well.

The environment estimation can be done by treating the environment as a policy that makes responses to the interactions, and employing the imitation learning methods (Schaal 1999; Argall et al. 2009) to learn the environment policy from the past data, which has drawn a lot of attention recently (Chen et al. 2019). Comparing with using supervised learning, i.e., behavior clone, to learn the environment policy, a more promising solution in Shi et al. (2018) is to formulate the environment policy learning as an interactive process between the environment and the system in it. The advantage of such a setting is that it could make a better generalization to evaluate a new system policy, especially when the environment policy changes over time and the distribution of new collected data shifts as well (Zhao et al. 2020). Take the example of the commodity recommendation system: the user and the platform could be viewed as two agents interacting with each other, where the user agent views the platform as the environment and the platform agent views the user as the environment. By this multi-agent view, Shi et al. (2018) proposed a *multi-agent adversarial imitation learning* (MAIL) method, extending the *generative adversarial imitation learning* (GAIL) framework (Ho and Ermon 2016), to learn the two policies simultaneously by beating the discriminator which aims to find the difference between the generated and the real interaction data.

However, the MAIL method (Shi et al. 2018) is under the assumption that the whole world only consists of two agents. From the perspective of the real users, they can receive much more information from the real-world that is not recorded in the data. Therefore, it is still quite challenging to build a realistic environment in real-world applications, since the real-world scenario is too complex to offer a fully observable environment, which means that there may exist unobservable variables that can implicitly affect the interaction. As shown in Fig. 1, in the classical setting, the next state in an MDP depends on the previous state and the executed action. While in most of real-world scenarios, the next state could be additionally influenced by some hidden variables, which result in the state being partially



(a) classical environment **(b)** environment with a hidden variable

Fig. 1 Illustration of the graph structure and the collected data (**a**) in the classical environment that assumes fully observable, and (**b**) in the more realistic environment with unobserved variables

observable. If we follow the assumption of a fully observable world, the estimation would be misled by the appeared fake associations in the data, which are commonly caused by the hidden variables. Thus, it is essential to take such hidden variables into consideration.

Hidden state problems arise in many real-world decision tasks. The state of the environment is only incompletely known to the learning agent. Partially observable MDPs (POMDPs) (Singh et al. 1994) are an appropriate model for hidden state problems. Most previous approaches to such problems have combined computationally expensive state estimation techniques with learning control (Kaelbling et al. 1998; Pineau et al. 2003; Cassandra et al. 2005). In control theory, it is widely accepted that learning a model of the environment is useful for policy control in such cases, which is called system identification. There has been some work on learning discrete-state models for the partially observable environment (Sallans 1999). While in many real-world applications, the state of the environment is commonly high dimensional and continuous. Little work has been done in this promising area. In this study, we try to use reinforcement learning to learn the continuous-state environment model for the partially observable tasks.

To involve hidden variables into the environment estimation, we propose a *partially-observed multi-agent environment estimation* method, named POMEE. First, we formulate two representative polices, the agent policy π_a and the environment policy π_e. Then, in order to simulate the effect of hidden variables, we add a hidden agent π_h into the interaction. According to the influence relationship, the hidden agent π_h interacts with the other two agents. Based on the formulation, we learn policies of these three agents only using the interaction data between π_a and π_e. Since hidden variables are unobservable, we propose two techniques to learn the policy of it: the partially-observed environment model and the compatible discriminator under the framework of GAIL (Ho and Ermon 2016). As the training converges, the partially-observed environment is successfully generated.

Based on the built virtual environment, RL algorithms can be used to optimize the agent policy. Policy optimization mainly includes two steps: policy evaluation and policy control. In the policy evaluation, the performance of the policy is evaluated according to the reward function in the simulator. When the hidden variables exist, the response of the environment could be additionally influenced by them. So the causal relationship between actions and responses must be depicted accurately in the simulator. Moreover, since the real-world application prefers online A/B testing to evaluate the improvement effect of the policy (Agarwal et al. 2016), it is more desirable to build a causal reward function in the

simulation environment. Generally, causal modeling aims at modeling the uplift, which is the incremental impact of an action or a treatment on an individual unit.

Recently, there are a lot of studies to learn the uplift model by tree-based methods (Hansotia and Rukstales 2002; Radcliffe and Surry 2011; Rzepakowski and Jaroszewicz 2012; Athey and Imbens 2015; Wager and Athey 2018), which demonstrate stable performance on many tasks. In this paper, to learn an uplift model compatible with the simulation environment, we propose a *deep uplift inference network* model, named DUIN, to infer the uplift of each action. By analogy with the experimental setting of online AB testing, the DUIN model has two output branches: the control branch and the treatment branch. The control branch is trained to predict the potential outcome of the control action. The treatment branch is trained to infer the uplift behind the observed outcome. When the model converges, the output of the treatment branch converges to the true value of the uplift.

By implementing the environment policy in the DUIN structure, we propose a *POMEE with uplift inference* approach, named POMEE-UI, to build a partially-observed environment with a causal reward mechanism. First, we use the randomized trial data to train the environment policy following the DUIN optimization. Then, the parameters of the treatment branch in environment policy are set to be fixed during the training process of POMEE. In this way, the learned environment policy can generate simulation data similar to the real data, and meanwhile offer the uplift value of each action as a causal reward.

To verify the effectiveness of POMEE-UI, we first use an artificial environment abstracted from the real-world application to conduct toy experiments. Then, we apply POMEE-UI to a large-scale recommender system for ride-hailing driver programs in Didi Chuxing. The results of toy experiments show that the environment learned by POMEE-UI can not only have a reliable causality, but also restore the real policy functions well. In the real-world application, POMEE-UI achieves a promising performance on both simulation evaluation and offline policy optimization. Finally, based on the virtual environment built by POMEE, a recommender policy is optimized and deployed online for A/B testing. The results of online experiment further demonstrate the effectiveness of applying our method to real-world applications. The contribution of this work is summarized as follows:

(1) We propose a novel environment estimation method POMEE to tackle the real-world situation where the state of the environment is partially observable.
(2) By treating the hidden variables as a hidden policy, we formulate the hidden effect into a multi-agent interactive environment. We define the partially-observed environment model and the compatible discriminator to learn policies effectively.
(3) We propose a novel deep uplift inference network DUIN model to learn the uplift effectively. Due to the flexibility in various settings, it makes deep neural networks a step further in uplift modeling.
(4) By implementing the environment policy in the DUIN structure, we propose the POMEE-UI approach to build a partially-observed environment with uplift inference. A general, feasible and reliable pipeline solution is built to enable RL to release the powerful sequential decision-making ability in real-world applications.
(5) We deploy the proposed framework to the program recommender system on a large-scale riding-hailing platform, and achieve significant improvements in the test phase.

The rest of this paper is organized as follows: we introduce the background in Sect. 2 and the proposed method POMEE in Sect. 3. The DUIN model and the POMEE-UI approach are proposed in Sect. 4. We describe the application of POMEE-UI to the driver program

recommendation system in Sect. 5. Experiment results are reported in Sect. 6. Finally, we conclude the paper in Sect. 7.

2 Background

2.1 Reinforcement learning

The problem to be tackled by Reinforcement Learning (RL) can usually be represented by a Markov decision process (MDP) quintuple (S, A, T, R, γ), where S is the state space and A is the action space and $T : S \times A \mapsto S$ is the state transition model and $R : S \times A \mapsto \mathbb{R}$ is the reward function and γ is the discount factor of cumulative reward. Reinforcement learning aims to optimize policy $\pi : S \mapsto A$ to maximize the expected γ-discounted cumulative reward $\mathbb{E}_\pi\left[\Sigma_{t=0}^{T}\gamma^t r_t\right]$ by enabling agents to learn from interactions with the environment. The agent observes state s from the environment, selects action a given by π to execute in the environment and then observes the next state, obtains the reward r at the same time until the terminal state is reached. Consequently, the goal of RL is to find the optimal policy

$$\pi^\star = \arg\max_\pi \mathbb{E}_\pi\left[\Sigma_{t=0}^{T}\gamma^t r_t\right], \tag{1}$$

of which the expected cumulative reward is the largest.

Partially observable Markov decision process The POMDP framework is general enough to model a variety of real-world sequential decision-making problems. The general framework of Markov decision processes with incomplete information was described by Astrom (1965) in the case of a discrete state space, and it was further studied in the operations research community where the acronym POMDP was coined. It was later adapted for problems in artificial intelligence and automated planning by Kaelbling et al. (1998). A discrete-time POMDP can formally described as a 7-tuple $(S, A, T, R, \Omega, O, \gamma)$, where S is a set of states and A is a set of actions and T is a set of conditional transition probabilities $T(s'|s, a)$ for the state transition $s \to s'$ and $R : S \times A \mapsto \mathbb{R}$ is the reward function and Ω is a set of observations and O is a set of conditional observation probabilities $O(o|s', a)$ and $\gamma \in [0, 1]$ is the discount factor.

At each time period, the environment is in some state $s \in S$. The agent chooses an action $a \in A$, which causes the environment transition to state $s' \in S$ with probability $T(s'|s, a)$. At the same time, the agent receives an observation $o \in \Omega$ which depends on the new state of the environment with probability $O(o|s', a)$. Finally, the agent receives a reward $r = R(s, a)$. Then the process repeats. The goal is for the agent to choose actions at each time step that maximizes its expected future discounted reward, which is the same as the goal of MDP defined in Eq. (1).

Imitation learning Learning a policy directly from expert demonstrations has been proven very useful in practice, and has made a significant improvement of performance in a wide range of applications (Ross et al. 2011). There are two traditional imitation learning approaches: behavioral cloning, which trains a policy by supervised learning over state-action pairs of expert trajectories (Pomerleau 1991), and inverse reinforcement learning (Russell 1998), which learns a cost function that prioritizes the expert trajectories over others. Generally, common imitation learning approaches can be unified as the follow formulation: training a policy π to minimize the loss function $l(s, \pi(s))$, under the discounted

state distribution of the expert policy: $P_{\pi_e}(s) = (1-\gamma)\Sigma_{t=0}^{T}\gamma^t p(s_t)$. The object of imitation learning is represented as

$$\pi = \arg\min_{\pi} \mathbb{E}_{s\sim P_{\pi_e}}[l(s, \pi(s))]. \tag{2}$$

2.2 Environment estimation

Reinforcement learning relies on an environment. However, when it comes to real-world applications, it is not practical to interact with the real-world environment directly to optimize the policy because of the low sampling efficiency and the high-risk uncertainty, such as online recommendation in E-commerce and medical diagnosis. A viable option is to build a virtual environment (Shi et al. 2018) for offline policy training. As a result, the training process could be more efficient by interacting with the virtual environment, and the interaction cost could be avoided as well.

Generative adversarial nets Generative adversarial networks (GANs) (Goodfellow et al. 2014) and its variants are rapidly emerging unsupervised machine learning techniques. GANs involve training a generator G and discriminator D in a two-player zero-sum game:

$$\arg\min_{G}\arg\max_{D\in(0,1)} \mathbb{E}_{x\sim p_E}\big[\log D(x)\big] + \mathbb{E}_{z\sim p_z}\big[\log(1-D(G(z)))\big], \tag{3}$$

where p_z is some noise distribution. In this game, the generator learns to produce samples (denoted as x) from a desired data distribution (denoted as p_E). The discriminator is trained to classify the real samples and the generated samples by supervised learning, while the generator G aims to minimize the classification accuracy of D by generating samples like real ones. In practice, the discriminator and the generator are both implemented by neural networks, and updated alternately in a competitive way. The training process of GANs can be seen as searching for a Nash equilibrium in a high-dimensional parameter space, so it has very strong ability of data representation. Recent studies (Menick and Kalchbrenner 2018) have shown that GANs are capable of generating faithful real-world images, demonstrating their applicability in modeling complex distributions.

Generative adversarial imitation learning GAIL (Ho and Ermon 2016) has become a popular imitation learning method recently. It allows the policy to interact with the environment but no reward signals. It was proposed to avoid the shortcoming of traditional imitation learning, such as the instability of behavioral cloning and the complexity of inverse reinforcement learning. It adopts the GAN framework to learn a policy (i.e., the generator G) with the guidance of a reward function (i.e., the discriminator D) given expert demonstrations as real samples. GAIL formulates a similar objective function like GANs, except that here p_E stands for the expert's joint distribution over state-action pairs:

$$\arg\min_{\pi}\arg\max_{D\in(0,1)} \mathbb{E}_{\pi}\big[\log D(s,\ a)\big] + \mathbb{E}_{\pi_E}\big[\log(1-D(s,\ a))\big] - \lambda H(\pi), \tag{4}$$

where $H(\pi) \triangleq \mathbb{E}_{\pi}\big[-\log \pi(a|s)\big]$ is the entropy of policy π.

GAIL allows the agent to execute the policy in the environment and update it with policy gradient methods (Schulman et al. 2015). The policy is optimized to maximize the similarity between the policy-generated trajectories and the expert trajectories measured by D. Similar to the Eq. (2), the policy π is updated to minimize the loss function

$$l(s, \pi(s)) = \mathbb{E}_{\pi}\left[\log D(s,a)\right] - \lambda H(\pi) \cong \mathbb{E}_{\tau_i}\left[\log \pi(a|s) Q(s,a)\right] - \lambda H(\pi). \quad (5)$$

where $Q(s,a) = \mathbb{E}_{\tau_i}\left[\log(D(s,a))|s_0 = s, a_0 = a\right]$ is the state-action value function. The discriminator is trained to predict the conditional distribution: $D(s,a) = p(y|s,a)$ where $y \in \{\pi_E, \pi\}$. In other words, $D(s, a)$ is the likelihood ratio that the pair (s, a) comes from π rather than from π_E. GAIL is proven to achieve similar theoretical and empirical results as IRL (Finn et al. 2016) while it is more efficient.

Recently, the multi-agent extension of GAIL (Shi et al. 2018) has been proven effective to build a virtual environment. A subset of this work in this paper has been published before (Shang et al. 2019). The previous publication proposed an environment reconstruction method to virtualize a real-world recommendation environment with a response model. In this paper, a causal uplift model is additionally designed to learn a more reliable environment model for better policy optimization. Additionally, we have revamped the exposition of our environment generation method from the POMDP perspective.

2.3 Causal inference and uplift modeling

Uplift modeling refers to the set of techniques used to model the incremental impact of an action or a treatment on a customer outcome. For example, a manager at an e-business company could be interested in estimating the effect of sending an advertising e-mail to different customers on their probability to click the links to promotional ads. With that information at hand, the manager is able to target potential customers efficiently.

Uplift modeling is both a causal inference and a machine learning problem (Gutierrez and Gérardy 2017). It is a causal inference problem because one needs to estimate the difference between two outcomes that are mutually exclusive for an individual (either a user receives a promotional e-mail or does not receive it). To overcome this counter-factual nature, uplift modeling crucially relies on randomized experiments. Uplift modeling is also a machine learning problem as one needs to train different models and select the one that yields the most reliable uplift prediction according to some performance metrics. More prerequisite knowledge can be seen in "Appendix A.1 and A.2".

The most popular methods for uplift modeling in the literature remain the tree-based ones (see Hansotia and Rukstales 2002; Radcliffe and Surry 2011; Rzepakowski and Jaroszewicz 2012; Athey and Imbens 2015; Wager and Athey 2018). However, little work (Johansson et al. 2016) has been done to release the strong representation ability of deep neural network for uplift modeling. In this paper, we make a further step to use the deep neural network for uplift modeling, which is also compatible with the training process of environment estimation.

3 Partially-observed multi-agent environment Estimation

To estimate the environments where hidden states exist, we propose a novel *partially-observed multi-agent environment estimation* (POMEE) method.

3.1 Formulation

In this study, by treating the hidden variables as a hidden policy, we formulate the partially-observed environment estimation as follows:

Partially-observed multi-agent environment.

- Observable agent A : known as the policy agent, denoted as π_a with observation o_A as input and action a_A as output.
- Observable agent E: known as the environment, denoted as π_e with observation o_E as input and action a_E as output.
- Unobservable agent H: known as hidden variables, denoted as π_h with observation o_H as input and action a_H as output. It plays a role of hidden effect in the sequential interactions between the policy agent A and the environment agent E.

Simulation of interaction.

- Start of the simulation trajectory: Given o_A (sampled from the initial states in the historical data) as the observation of agent A, it takes an action $a_A = \pi_a(o_A)$.
- Hidden effect for the action: the observation o_H of agent H is formatted as the concatenation $o_H =< o_A, a_A >$, and the action $a_H = \pi_h(o_H)$ has the same format as a_A.
- Hidden effect for the environment response: the observation o_E of agent E is formatted as the concatenation $o_E =< o_A, a_A, a_H >$, and its action is $a_E = \pi_e(o_E)$ which can be used to move forward to the new state for next step.

Goal We assume that the true policies $\pi_e^\star$ and $\pi_h^\star$ behind the observed trajectories are fixed in the time of a trajectory. The objective is to use only observable interactions, that is, trajectories $\tau_{real} = \{(o_A, a_A, a_E)\}$, to imitate the policies π_a and π_e, together with recovering the hidden effect of H by inferring the hidden policy π_h.

3.2 Objective function

The objective function of multi-agent imitation learning is defined analogy to Eq. (2):

$$(\pi_a, \pi_e, \pi_h) = \underset{(\pi_a, \pi_e, \pi_h)}{\arg\min}\, \mathbb{E}_{o_A \sim P_{\tau_{real}}} \big[L(o_A, a_A, a_E)\big], \tag{6}$$

where a_A, a_E depend on three policies. By adopting the GAIL framework, according to Eq. (5), we can get the imitation loss for environment estimation as

$$L(o_A, \pi_a, \pi_h, \pi_e) = \mathbb{E}_{\pi_a, \pi_h, \pi_e} \big[\log D(o_A, a_A, a_E)\big] - \lambda \Sigma_{\pi \in \{\pi_a, \pi_h, \pi_e\}} H(\pi). \tag{7}$$

We observe that π_a is independent with π_h and π_e given o_A and a_A, then using conditional independence rule, $D(o_A, a_A, a_E)$ under GAIL framework can be decomposed as

$$\begin{aligned} D(o_A, a_A, a_E) &= p(\pi_a, \pi_h, \pi_e | o_A, a_A, a_E) \\ &= p(\pi_a | o_A, a_A, a_E)\, p(\pi_h, \pi_e | o_A, a_A, a_E) \\ &= p(\pi_a | o_A, a_A)\, p(\pi_h, \pi_e | o_A, a_A, a_E) \\ &= D_a(o_A, a_A)\, D_{he}(o_A, a_A, a_E). \end{aligned} \tag{8}$$

where $D_a(o_A, a_A)$ denotes the imitation item of policy π_a, and $D_{he}(o_A, a_A, a_E)$ denotes the imitation item of policies π_h and π_e. Combining Eqs. (7) and (8), we can decompose the loss function as

$$\begin{aligned} L(o_A, \pi_a, \pi_h, \pi_e) &= \mathbb{E}_{\pi_a, \pi_h, \pi_e}\left[\log D_a(o_A, a_A) D_{he}(o_A, a_A, a_E)\right] - \lambda \Sigma_{\pi \in \{\pi_a, \pi_h, \pi_e\}} H(\pi) \\ &= \mathbb{E}_{\pi_a}\left[\log D_a(o_A, a_A)\right] - \lambda H(\pi_a) \\ &\quad + \mathbb{E}_{\pi_h, \pi_e}\left[\log D_{he}(o_A, a_A, a_E)\right] - \lambda \Sigma_{\pi \in \{\pi_h, \pi_e\}} H(\pi) \\ &= l(o_A, \pi_a(o_A)) + l((o_A, a_A),\ \pi_e \circ \pi_h((o_A, a_A))) \end{aligned} \tag{9}$$

which indicates that the optimization can be decomposed as optimizing policy π_a and joint policy $\pi_{he} = \pi_e \circ \pi_h$ individually by minimizing the loss functions

$$\begin{aligned} l(o_A, \pi_a(o_A)) &= \mathbb{E}_{\pi_a}\left[\log D_a(o_A, a_A)\right] - \lambda H(\pi_a) \\ &\cong \mathbb{E}_{\tau_i}\left[\log \pi_a(a_A|o_A) Q(o_A, a_A)\right] - \lambda H(\pi_a), \end{aligned} \tag{10}$$

where $Q(o_A, a_A) = \mathbb{E}_{\tau_i}\left[\log(D(o_A, a_A))|o_0 = o_A, a_0 = a_A\right]$ is the state-action value function of π_a, and

$$\begin{aligned} l((o_A, a_A), \pi_{he}((o_A, a_A))) &= \mathbb{E}_{\pi_h, \pi_e}\left[\log D_{he}((o_A, a_A), a_E)\right] - \lambda \Sigma_{\pi \in \{\pi_h, \pi_e\}} H(\pi) \\ &\cong \mathbb{E}_{\tau_i}\left[\log \pi_{he}(a_E|o_A, a_A) Q(o_A, a_A, a_E)\right] - \lambda \Sigma_{\pi \in \{\pi_h, \pi_e\}} H(\pi), \end{aligned} \tag{11}$$

where $Q(o_A, a_A, a_E) = \mathbb{E}_{\tau_i}\left[\log(D((o_A, a_A), a_E))|o_0 = o_A, a_{A0} = a_A, a_{B0} = a_E\right]$ is the state-action value function of π_{he}.

Based on this result, we propose the partially-observed environment model and the compatible discriminator to achieve the goal of imitating polices of agents A and E together with the hidden agent H, thus obtaining the POMEE approach.

3.3 Partially-observed environment model

In this study, the interaction between the agent A (known as the policy agent) and the agent E (known as the environment) could be observed, while the policy and data of the agent H (known as hidden variables) are unobservable.

Based on the decomposition result of objective function, we combine the hidden policy π_h with the observable policy π_e as a joint policy, named $\pi_{he} = \pi_e \circ \pi_h$. Under the GAIL framework, together with the policy π_a, the generator is formalized as an interactive environment of two policies as shown in the top of Fig. 2. The joint policy can actually be expressed as

$$\pi_{he}(o_A, a_A) = \pi_e(o_A, a_A, \pi_h(o_A, a_A)) \tag{12}$$

in which the input (o_A, a_A) and the output a_E are both observable in the historical data. Therefore, we can use imitation learning methods to train these two policies by imitating the observed interactions.

The policies in generator are updated alternatingly in each training step: first, the joint policy π_{he} is updated with the imitation reward r^{he} given by the discriminator. Second, the policy π_a is updated with the corresponding reward r^a given by the discriminator as well. Though there is no explicit updating step for the hidden policy π_h, it has been inferred potentially by these two steps. Intuitively, the generated hidden policy π_h is just like a by-product along with the process of optimizing policies π_a and π_{he} towards the truth, and consequently it can recover the real hidden effect to some extent. To make the training process more stable, we employ TRPO (Schulman et al. 2015) to update the two policies.

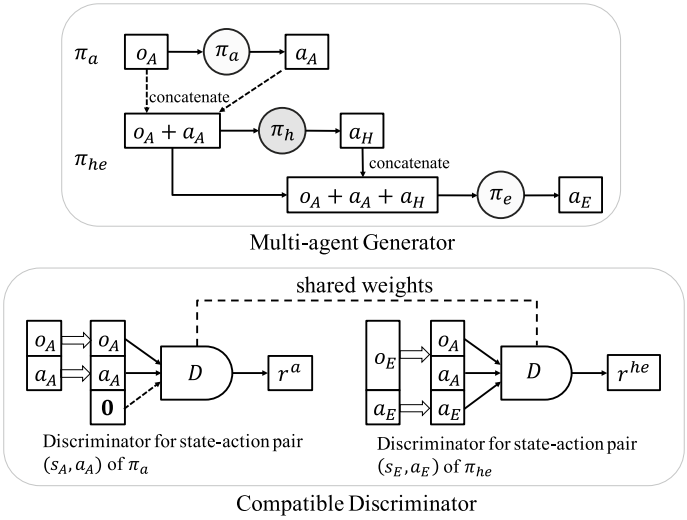


Fig. 2 The generator and the discriminator in POMEE. The multi-agent interactive environment plays a role of generator, and can generate simulation interaction data. The discriminator is designed to be compatible for classify the state-action pairs of both the policy π_a and the joint policy π_{he}

3.4 Compatible discriminator

In most of generative adversarial learning frameworks, there is only one task to model and learn in the generator. In this study, it is essential to simulate and learn different reward functions for the two policies π_a and π_{he} consisted in the generator, respectively.

We design the discriminator compatible with two classification tasks. As Fig. 2 illustrates, one task is designed to classify the real and generated state-action pairs of π_a while the other one is to classify the state-action pair of π_{he}. Correspondingly, the discriminator has two kinds of input: the state-action pair $(o_A,\ a_A,\ a_E)$ of policy π_{he} and the zero-padded state-action pair $(o_A,\ a_A,\ \mathbf{0})$ of policy π_a. This setting indicates that the discriminator splits not only the policy π_{he}'s state-action space, but also the policy π_a's state-action space. The loss function of each task is defined as

$$E_{\tau_{sim}}\left[\log(D_\sigma(o_A, a_A, a_E))\right] + E_{\tau_{real}}\left[\log(1 - D_\sigma(o_A, a_A, a_E))\right] \tag{13}$$

for π_{he} , and

$$E_{\tau_{sim}}\left[\log(D_\sigma(o_A, a_A, \mathbf{0}))\right] + E_{\tau_{real}}\left[\log(1 - D_\sigma(o_A, a_A, \mathbf{0}))\right] \tag{14}$$

for policy π_a.

The output of the discriminator is the probability that the pair data comes from the real data distribution. The discriminator is trained with supervised learning by labeling the real state-action pair as 1 and the generated fake state-action pair as 0. Then it is used as a reward giver for the policies while simulating interactions. The reward function for policy π_{he} can be written as:

$$r^{he} = -\log(1 - D(o_A,\ a_A,\ a_E)), \tag{15}$$

and the reward function for policy π_a is

$$r^a = -\log(1 - D(o_A,\ a_A,\ \mathbf{0})). \tag{16}$$

3.5 Simulation

We simulate interactions in the generator module. The simulated trajectory is generated as follows: First, we randomly sample one trajectory from the observed data and set its first observation as the initial observation o_0^A. Then we can use the two policies π_a, π_{he} to generate a whole trajectory triggered from o_0^A. Given the observation o_t^A as the input of π_a, the action a_t^A can be obtained. In consequence, the action a_t^E can be obtained from the joint policy π_{he} with the concatenation $< o_t^A, a_t^A >$ as input. Then we can get the imitation reward r_t^{he} by Eq. (15) and r_t^a by Eq. (16) which are used for updating policies in the adversarial training step. Finally, we can get the next observation o_{t+1}^A based on o_t^A and a_t^E by the predefined transition dynamics. This step is repeated until a terminal state, and a fake trajectory is generated.

Algorithm 1 POMEE

Input: $D_{real} = \{\tau_1, \tau_2, \ldots, \tau_n\}$: The observed real trajectories over T steps;
N: Number of trajectories generated in each iteration;
K: Steps of generator per discriminator step;
I: Number of iterations;

Initialize parameters θ^{he} and θ^a of policy π_{he} and π_a, parameters σ of discriminator D;
for $i = 1, 2, \ldots, I$ **do**
  for $k = 1, 2, \ldots, K$ **do**
    $\tau_{sim} = \varnothing$;
    for $j = 1, 2, \ldots, N$ **do**
      $\tau_j = \varnothing$;
      Randomly sample a trajectory τ_r from D_{real} and set its first state as the initial state o_0^A ;
      for $t = 0, 1, 2, \ldots, T-1$ **do**
        Simulate the action $a_t^A = \pi_a(o_t^A)$ and then the action $a_t^E = \pi_{he}(o_t^A, a_t^A)$;
        Get reward r_t^a and r_t^{he}, respectively, according to equations (16) and (15);
        Get next observation o_{t+1}^A given $o_t^A,\ a_t^E$ by predefined transition;
        Add $\{o_t^A,\ a_t^A,\ a_t^E,\ r_t^a,\ r_t^{he}\}$ to τ_j;
      end for
      Add τ_j to τ_{sim};
    end for
    TRPO update θ^a and θ^{he}, respectively, with τ_{sim} according to equations (10) and (11);
  end for
  Update the parameters σ of discriminator D by minimizing the loss in equation (13) and (14) in turn;
end for
return the trained environment policies $\pi_e,\ \pi_h$.

3.6 POMEE algorithm

Based on the partially-observed environment model and the compatible discriminator, we propose the POMEE method to achieve the goal of estimating environment with hidden variables from the observed data.

Algorithm 1 shows the details of POMEE. The whole algorithm adopts the generative adversarial training framework. In each iteration, firstly the generator simulates interactions

using policies π_a and π_{he} to collect the trajectory set τ_{sim} corresponding to Line 5 to Line 15. Then the policies π_a and π_{he} are updated in turn using TRPO with generated trajectories τ_{sim} in Line 16. After K generator steps, the compatible discriminator is trained by two steps as shown in Line 18. Specifically, the predefined transition dynamics in Line 11 depends on specific tasks. In this way, the algorithm can effectively imitate the policies of observed interactions and recover the hidden variables beyond observations.

4 Partially-observed environment estimation with uplift inference

In reinforcement learning, the environment model mainly consists of two parts: the state transition dynamics and the reward function. The POMEE approach introduced in the previous section achieves the modeling of transition dynamics. In this section, we will introduce a novel uplift model to build the reward function in the simulation environment. It is important to concern the causality between rewards and actions when hidden variables exist in the environment. Only when the causality of different actions is accurately depicted, can the policy optimization based on the simulator make sense. An illustration of the importance of uplift modeling can be seen in "Appendix A.3".

To learn a causal reward function in the virtual environment, we propose a novel *deep uplift inference network* model DUIN that applies to the training process of POMEE. In addition, the DUIN model can be used flexibly to binary treatment settings and multi-treatment settings, as well as the classification tasks and regression tasks.

4.1 DUIN model structure

The uplift modeling is generally based on a randomized trial experiments. Given the data of control and treatment groups, deriving a variant Eq. (17) from the Eq. (24) in "Appendix A.1", we propose the DUIN model trained on the randomized experiment data to infer the uplift. Figure 3 illustrates the detailed structure of this model. The inputs of this network are the observation X fed into the input layer and the treatment indicator t fed into the intermediate layer. The output is the predicted potential outcome under X and t. We use supervised learning method to train this model. We have the following relationship regarding the uplift inference.

$$\mathbb{E}\left[Y_i(t)|X_i\right] = \mathbb{E}\left[Y_i(0)|X_i\right] + \tau_t(X_i). \tag{17}$$

The whole network consists of two modules: the representation module and the inference module. The representation module is trained to learn high-level features that can effectively represent the potential outcome space. Based on the high-level features, the inference module is trained to predict the outcome. The inference module splits into two branches: the control branch and the treatment branch. The output of the control branch is the outcome if not treated, corresponding to the $\mathbb{E}\left[Y_i(0)|X_i\right]$ in Eq. (17). The output of the treatment branch is the uplift estimation for treatment t, corresponding to $\tau_t(X_i)$. The two branches are merged by adding the outputs of each branch like Eq. (17), and the output just becomes the outcome of treatment t, corresponding to the $\mathbb{E}[Y_i(t)|X_i]$ in Eq. (17).

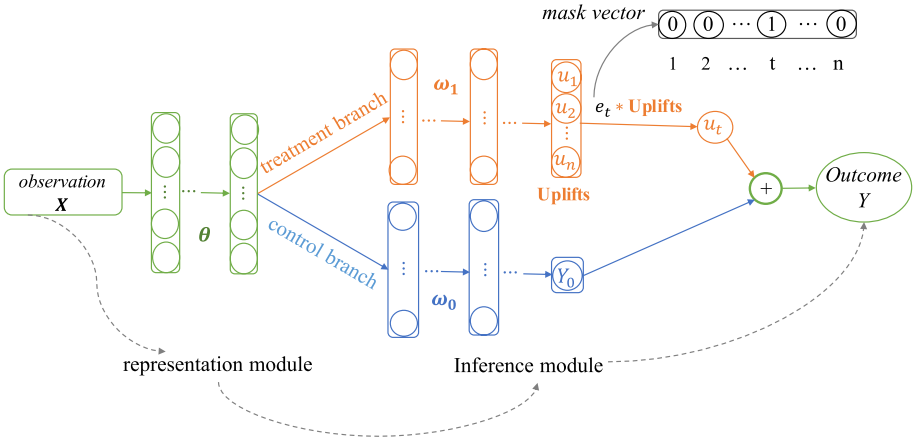


Fig. 3 The model structure of Deep Uplift Inference Network (DUIN) under the multi-treatment setting, and it will be the binary setting when $n = 1$. The observation X and the treatment t are fed as input and the the potential outcome Y is the output. The uplift outputs through an intermediate layer

4.2 DUIN optimization method

We use the supervised alternating optimization approach to train the DUIN model. We train the control branch together with the representation module on the control group data. Similarly, we train the treatment branch together with representation module on the treatment group data. The objective function can be formulated as

$$l(x, t, y, \theta, \omega_0, \omega_1) = L\big(y,\ \hat{y_0}(x, \theta, \omega_0) + e_t * u_n(x, \theta, \omega_1)\big), \tag{18}$$

where y is the ground truth outcome, $\hat{y_0}$ is the predicted outcome under the observation x with no treatment, u_n is the uplift vector under n different treatments and e_t is a mask row vector with the tth bit set as 1. Specifically, e_t is a zero vector when the treatment is control (not treated). The loss function L can be either regression loss, e.g., MSE and RMSE, or the classification loss, e.g., the logarithmic loss.

The whole training process of DUIN is shown in Algorithm 2. In each iteration, we update K steps for the parameters θ, ω_0 of the control branch, then update the same K steps for the parameters θ, ω_1 of the treatment branch. Experiment results show that the smaller K can make a better generalization and faster convergence under an ideal condition. As the model converges, the representation module and the treatment branch can be used as an uplift inference module. Intuitively, the uplift inference in DUIN is to fit the residual between the controlled outcome and the treated outcome.

Algorithm 2 DUIN training process

Input: $D = \{(x,t,y)\}$: The training data collected by randomized trial experiment;
K: Number of learning steps in one alternate round;
I: Number of total iteration steps;
n: the batch size;
γ: the learning rate;
$\theta, \omega_0, \omega_1$: the parameters of representation module, control branch and treatment branch.

Initialize parameters $\theta, \omega_0, \omega_1$ of the neural network;
for $i = 0, 1, \ldots, I-1$ **do**
  Randomly sample a mini-batch data $D_{batch}((x,t,y))$ with batch size n;
  if $i \mod 2K < K$ **then**
    update θ, ω_0 to minimize the loss in equation (18) using the control group data in D_{batch};
  else
    update θ, ω_1 to minimize the loss in equation (18) using the treated group data in D_{batch};
  end if
end for
return the parameters $\theta, \omega_0, \omega_1$ of representation module, control branch and treatment branch.

4.3 POMEE with uplift inference

By implementing the environment policy π_e in the DUIN structure, we propose a *POMEE with uplift inference* approach POMEE-UI as shown in Algorithm 3. Based on the POMEE framework, it can achieve the simulation of transition dynamics in a partially-observed environment. At the same time, due to the DUIN structure of the environment policy, a reward function with causality is also constructed. The integrated environment model can be more reliable for policy evaluation.

The computation graph of the environment policy is shown in Fig. 4. By analogy with the DUIN structure, the environment policy π_e also contains the representation module and the inference module. In the inference module, the output of the treatment branch u_E is the uplift value of action a_A under observation o_A. The output of the control branch a_{E0} is the potential outcome of the environment under none treatments. The final output of the environment policy a_E is calculated by a_{E0} plus u_E, which can be used to simulate the state transition process. The treatment branch acts as a reward function in the environment, of which the output u_E can be used as a reward for policy evaluation. In addition, considering the interaction relationship of the partially-observed environment, the output of the hidden policy a_H is fed into the control branch by splicing with the output of the representation module. Due to the unobservability of the hidden policy, the placeholder for a_H is fed with a zero vector during the training process of DUIN.

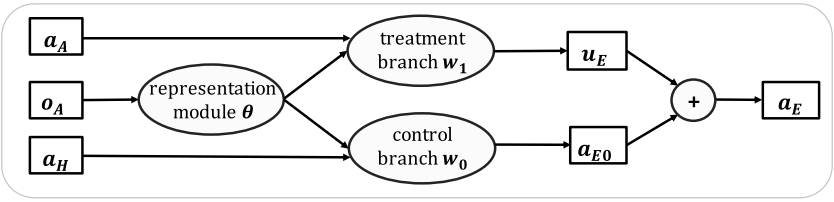


Fig. 4 The computation graph of the environment policy π_e implemented in the DUIN structure. The treatment action a_A is fed into the treatment branch and the hidden action a_H is fed into the control branch. The output a_E and u_E are the response action and the estimated uplift value, respectively

Algorithm 3 POMEE-UI

Input: $D_{rand} = \{(o_A, a_A, a_E)\}$: The training data collected by randomized trial experiment;
$D_{real} = \{\tau_1, \tau_2, \ldots, \tau_n\}$: The observed real trajectories over T steps.

1: Train a environment policy model π_e in the DUIN structure on the randomized trial dataset D_{rand} using the training method shown in Algorithm 2;
2: Set parameters θ, ω_1 of model π_e as untrainable variables, which are fixed in the following steps;
3: Run the training of POMEE on the observed dataset D_{real} as illustrated in Algorithm 1;
4: **return** the trained policies π_e, π_h: a partially-observed environment with an uplift inference module.

Algorithm 3 describes the training process of POMEE-UI. First, a DUIN-style environment policy model π_e is trained on the randomized trial dataset D_{rand}. Second, the representation module and the treatment branch of π_e remain fixed as an uplift model, and the parameters θ, ω_1 are set to be untrainable in the following step. Finally, the training process of POMEE is carried out on the observed dataset D_{real}. In other words, only the parameter ω_0 of the control branch in π_e is updated in the POMEE training. In addition, since the hidden action a_H is not observable, it is initialized as a zero vector during the DUIN training of π_e in the first step in Line 1.

5 Application in driver program recommendation

5.1 Driver program recommendation

We have witnessed a rapid development of on-demand ride-hailing services in recent years. In this economic pattern, the platform often need to recommend programs to drivers, aimed to help them finish more orders. Specifically, the platform would select the appropriate program to recommend the drivers to participate every day, and then adjust the program content according to the drivers' feedback behavior. This is a typical sequential recommendation task and can be naturally tackled by reinforcement learning (Qin et al. 2020). However, since the behavior of drivers is not only influenced by the recommended programs, but also influenced by some other unobservable factors, such as the response to special events and so on, that is, hidden variables exist in this application scenario. In order to optimize the recommender policy, it is essential to take into account the potential influence of hidden factors when recommending programs.

However, traditional reinforcement learning approaches are applied in these problems without exploring the impact of hidden variables, which would consequently degrade the learning performance. Thus, a more adaptive approach such as POMEE-UI proposed in this paper is desirable to tackle these problems.

In this paper, we propose a general pipeline for applying reinforcement learning to optimize a policy in a real-world application based on historical data. First and foremost, we build a virtual environment, namely simulator, to precisely recover the transition dynamics and reward mechanism of the real-world environment by using historical data. We then apply RL algorithms to optimize the system policy by interacting with the virtual environment. Such simulator-based RL method can be very efficient without any interaction cost with the real-world environment. A more detailed illustration of the pipeline work can be seen in Fig. 15 in "Appendix A.4".

5.2 POMEE-UI based driver program recommendation

As for the driver program recommendation, we apply POMEE-UI to build a virtual environment with hidden variables by using historical data. As shown in Fig. 5, there are three agents in the environment, representing driver policy π_d, platform policy π_p and hidden policy π_h. We can see that the driver policy and the platform policy have the nature of "mutual environment" from the perspective of MDP. From the platform's point of view, its observation is the driver's response, and its action is the recommendation program to the driver. Correspondingly, from the driver's point of view, its observation is the platform's recommendation program, and its action is the driver's response to the platform. The hidden variables are modeled as a hidden policy according to POMEE, so as to make a dynamic effect at each time step.

Data preparation Based on the real-world scenario, we integrated the historical data and then construct historical trajectories $D_{hist} = \{\tau_1, \ldots, \tau_i, \ldots, \tau_n\}$ representing trajectories of n drivers. Each trajectory $\tau_i = \{o_0^P, a_0^P, a_0^D, o_1^P, \ldots, o_t^P, a_t^P, a_t^D, o_{t+1}^P, \ldots, o_T^P\}$ represents the T steps of observable interactions between the driver d_i and the platform system. For the DUIN training, we collected some randomized trial data as $D_{rand} = \{(o^P, a^P, a^D)\}$ from the recommender system.

Definition of policies According to the interaction among agents in this application, the observation and action of each agent policy are defined as follows:

- platform policy π_p: The observation o_t^P consists of the driver's static characteristics (using real data) and the simulated response behavior a_{t-1}^D. The action a_t^P is the program information recommended for the driver, represented as a 2-tuple of (T, M) integers, where T indicates the target and M is the amount of bonus for achieving the target.
- hidden policy π_h: The observation o_t^H consists of o_t^P and a_t^P. The action a_t^H is the same format as a_t^P.
- driver policy π_d: The observation o_t^D consists of o_t^P, a_t^P and a_t^H. The action a_t^D is the simulated driver's behavior at the current step, which indicates the completion degree of the recommended program a_t^P.

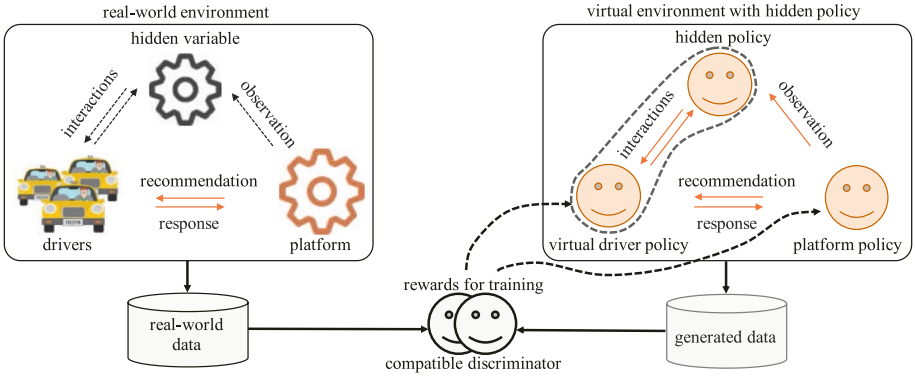


Fig. 5 The POMEE-UI framework applied in the driver program recommendation. While real-world data only collects the interactions between the drivers and the Didi Chuxing platform, the virtual environment contains three policies simulating the drivers, the platform, and the hidden variables

Analogy from the POMEE-UI method, we implement the driver policy π_d in the DUIN structure, and further combine the policies π_h, π_d into a joint policy. We then apply POMEE-UI to train π_d and π_h. Afterwards, the partially-observed environment of driver program recommendation is reconstructed.

5.3 RL in the virtual environment

Once the virtual environment is built, we can perform RL efficiently to optimize the policy π_p by interacting with the environment. The challenge with simulated training is that even the best available simulators do not perfectly capture reality, which is often called the "reality gap". Models trained purely on static data fail to generalize to the real world, as there is a discrepancy between simulated and real environments in terms of some physical properties. A number of related works have sought to address the reality gap in robotics, such as domain adaptation (Tzeng et al. 2016) and randomization of simulated environments (Sadeghi and Levine 2016), but they are not verified in real-world environments.

In this work, we design these mechanics to try to close the gap in this application. With the uplift model embedded in the virtual environment, we can design the recommendation reward with uplift values, which have a good causal relationship with the recommended program. In addition, due to the simulated hidden variables in the environment, the reinforcement learning approach could learn a more robust policy with improved performance in the real world.

6 Experiments

In this section, we conduct two groups of experiments to verify the effectiveness of the proposed POMEE-UI method. The first is a group of toy experiments in which a rule-based environment is designed, the second is a real-world application of driver program recommendation in Didi Chuxing.

6.1 Toy experiments

We firstly expect to design an artificial environment to verify the effectiveness of the proposed method POMEE-UI. However, it is rather difficult to design such an artificial environment that can verify both the hidden effects and the uplift learning performance. Considering that the uplift model produced by the DUIN training remains fixed during subsequent POMEE training in POMEE-UI, we firstly design a randomized trial experiment to evaluate the learning performance of uplift model independently. We then validate the policy simulation effects of POMEE-UI in a well-defined artificial environment.

6.1.1 DUIN on synthetic data

We design separately an artificial randomized trial dataset to verify the effectiveness of the DUIN model. All function rules and parameter values are designed to mimic the real-world environment. Three rule-based functions are defined: the artificial control

outcome function f^C, the artificial uplift function f^U, and the artificial treatment outcome function $f^T = f^C + f^U$ like Eq. (17). We conduct DUIN and two other meta-algorithms (Künzel et al. 2019) of uplift modeling as a comparison:

- *S-Learner* the treatment is included as a feature similar to the observation features to estimate a combined outcome function. It is a "single" response estimator.
- *T-Learner* the control response estimator and the treatment response estimator are learned separately, "T" being short for "two".
- *DUIN* the uplift modeling method proposed in this paper.

Rule-based artificial randomized trial data The observation is simplified as a two-dimensional vector, and the treatment is binary of 0 or 1. We first sample individual units from the observation space randomly, and then randomly target each unit as 0 for control and 1 for treatment. Based on the observation and treatment action, we generate the simulation data by the following rule-based outcome functions.

Denote the observation as (x_1, x_2), and constrain x_1, x_2 between -1 and 1. Figure 6 illustrates the three function spaces. The treated function f^T is represented as $f^T = f^C + f^U$. The controlled function f^C is defined as a a hemispherical surface with radius 1 above the XOY plane. It can be formulated as

$$f^C = \max\left(0, \sqrt{1 - x_1^2 - x_2^2}\right).$$

The uplift function, named f^U, is defined as a weighted sum of two conjugate two-dimensional Gaussian functions. The formulation is

$$f^U = \frac{3}{4}\left(\frac{\exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu_1}\right)^T \Sigma^{-1}\left(\mathbf{x}-\boldsymbol{\mu_1}\right)\right)}{2\pi\sqrt{|\Sigma|}}\right) - \frac{1}{2}\left(\frac{\exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu_2}\right)^T \Sigma^{-1}\left(\mathbf{x}-\boldsymbol{\mu_2}\right)\right)}{2\pi\sqrt{|\Sigma|}}\right),$$

where

$$\mu_1 = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \end{pmatrix},\ \mu_2 = \begin{pmatrix} -\frac{1}{3} \\ -\frac{1}{3} \end{pmatrix},\ \Sigma = \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix}.$$

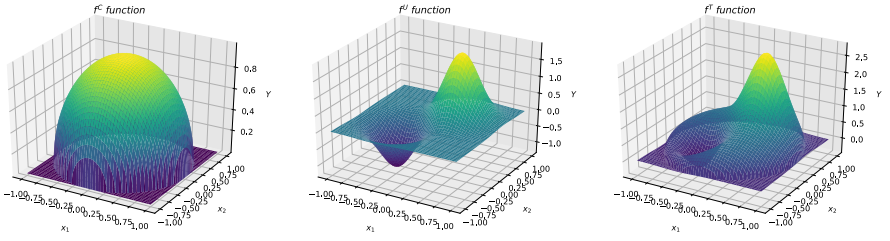


Fig. 6 Illustration of the uplift toy experiment settings.The f^C function is the control outcome function. The uplift function f^U is defined to mimic the uplift under different conditions as shown in Fig. 14 in "Appendix A.3". The treated function f^T is defined by adding the uplift function f^U to the control function f^C

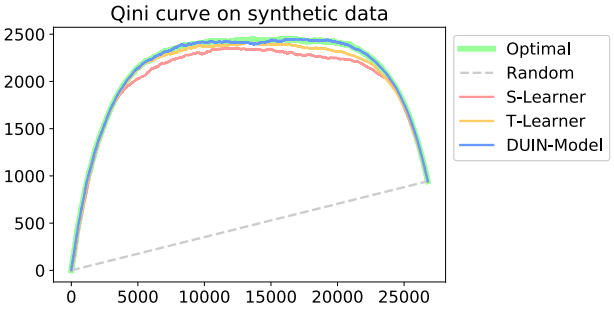


Fig. 7 Qini curves of three models evaluated on testing dataset: S-Learner, T-Learner and DUIN. Besides, the ground truth, as the Optimal model, and the random baseline are also plotted in this figure

Table 1 Comparison of Qini-Coefficient and $Q^{os,TO}$ on three models

Methods	Qini-Coefficient	Q^{TO}
S-Learner	1.6342	2.1553
T-Learner	1.6919	2.0548
DUIN	**1.7342**	**2.0155**
GROUND-TRUTH	1.7363	2.0126

The bold in tables indicates that the current method has the best performance of all the comparison methods on the current metric

The GROUND-TRUTH row represents the best performance that one model might achieve

Results. Uplift evaluation differs drastically from the traditional machine learning model evaluation, because of the invisibility of the ground truth. Here, we use Qini curve/coefficient (Radcliffe 2007) and Q^{TO} (Athey and Imbens 2015) to evaluate uplift models under binary treatment settings. Qini-Coefficient is an indicator to measure the ranking performance of the causal effect estimated by a model. The larger Qini-Coefficient, the better performance. Q^{TO} is a measure similar to MSE in supervised learning by exploiting the transformation of the potential outcome. The smaller Q^{TO}, the better performance. The detailed introduction to the two uplift evaluation metrics can be seen in "Appendix A.4".

The Qini curves of three models are shown in Fig. 7, which demonstrate the quality of uplift ranking inferred by the causal models. Although the rule-based setting is simple, the DUIN model has a significant out-performance than the other models on both metrics. The area under the uplift curve of the DUIN model is significantly larger than those of S-Learner and T-Learner methods, and this curve almost coincides with the Optimal one.

The performance of quantitative metrics are shown in Table 1. The Qini-Coefficient is the area between the Qini curve and the random curve. Q^{TO} is a measure similar to MSE in supervised learning. It is consistent with the uplift curves that the DUIN model has a larger Qini-Score and a smaller Q^{TO} than the other models. Furthermore, the gap between the DUIN model and the GROUND-TRUTH is very small, which potentially shows a strong causality of the DUIN model.

The uplift function space learned by the three models are shown in Fig. 8. The uplift function space, inferred by DUIN, is precisely close to the real defined one as shown in Fig. 6, while the ones learned by S-Learner and T-Learner approaches are deviated severely and they are not smooth which means a higher variance. As a result, we further demonstrate the ability of the DUIN model to infer the uplift function precisely and smoothly.

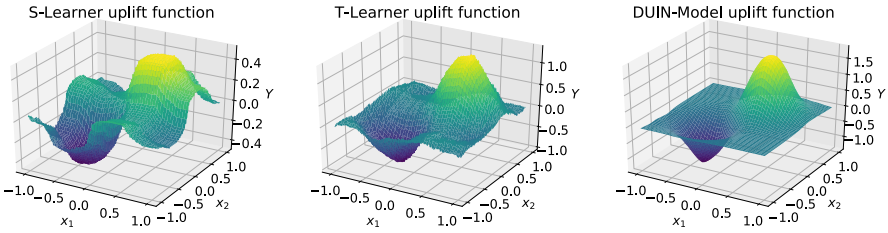


Fig. 8 Uplift function spaces learned by three different methods: S-Learner, T-Learner and DUIN-Model

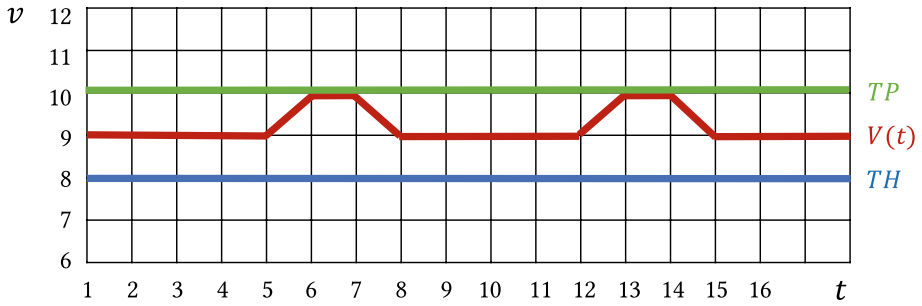


Fig. 9 Schematic drawing of interaction in the toy environment: t represents the time step and v is a variant affected by all three policies. TP and TH are the thresholds for policies taking effect, and $V(t)$ describes the intrinsic evolution trend of the artificial driver policy π_d

6.1.2 Artificial environment for POMEE-UI

We hand-craft an artificial environment with deterministic rules, consisting of the artificial platform policy π_p, the artificial driver policy π_d, and the artificial hidden policy π_h. In the same way, all function rules and parameter values are designed to mimic the real-world environment. We use POMEE and POMEE-UI to learn the policies and compare them with the real ones. Additionally, we conduct MAIL and MAIL-UI methods, without modeling hidden variables, as a comparison.

Description of the artificial environment Similar to the interaction in the driver program recommendation, we define a triple-agent environment to simulate a partially observable Markov decision process (POMDP). The semantic drawing of this toy experiment is shown in Fig. 9. In POMDP, the key variant v (denotes the driver's response) is affected by three policies at each time step. The policy π_d has an intrinsic evolution trend on the variant v in the period of 7 time steps, as defined in Eq. (22). The policy π_p has a positive effect on the variant v if the value of v is under the green line, otherwise no effect. Oppositely, the policy π_h has a negative effect on the variant v if the value of v is above the blue line, otherwise no effect. The green and blue lines can be seen as the thresholds of π_p and π_h to make effect on the evolution trend of v. Here we set the policy π_h as a role of hidden variables in this environment, of which the effect on the interaction would not be observed.

POMDP definition. All the hyperparameters in the following rule-based functions are selected randomly from an appropriate range of values.

The *observation* o is a tuple (tw, r, v), in which $tw \in \{1, 2, \ldots, 7\}$ is the time step in one period, r is a static factor used to make a difference on the effect of each agent and v is the key variant in the interaction process. The initial value v_0 is sampled from a uniform distribution $U(9 + wave, 9 - wave), wave = 1.2$, where $wave$ denotes the sampling range of v_0. We add the static factor $r = 1 - 0.5 \times \frac{v_0 - 9}{wave}$ into the state to make the episodes generated by this setting more diverse.

The *action* is defined as the output of the deterministic policy. The thresholds of green line TP and blue line TH are 10 and 8 correspondingly. Then we define the deterministic policy rule of each agent as follows:

$$a_p = \pi_p(tw, r, v) = \max\left(0, \min\left(1, r \times (TP - v) \times \frac{tw}{7}\right)\right), \tag{19}$$

$$a_h = \pi_h(tw, r, v, a_p) = \max\left(-1, \min\left(0, r \times \left(TH - v - \frac{a_p}{2}\right) \times \frac{tw}{7}\right)\right), \tag{20}$$

$$a_d = \pi_d(tw, r, v, a_p, a_h) = \Delta V(tw) + a_p + a_h. \tag{21}$$

where

$$\Delta V(tw) = \begin{cases} 1 & \text{if tw} = 5; \\ -1 & \text{else if tw} = 7; \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

The *transition dynamics* is simply defined as: $v_{t+1} = v_t + a_d^t$ and r is a constant once initialized. tw is a timestamp indicator cycling in the sequence $[1, 2, \ldots, 7]$. In this experiment, we set the length of trajectory T to 8.

By running the defined rules in the toy environment, we collect many episodes as training data $D_{real} = \left\{\left(o_p^0, a_p^0, a_d^0, o_p^1, \ldots, o_p^T\right)\right\}$. By randomly sampling from the observation space and the platform action space, we generate a randomized trial dataset $D_{rand} = \left\{\left(o_p, a_p, a_d\right)\right\}$. Based on these two datasets, we can perform the comparative algorithms to verify the effectiveness of POMEE-UI.

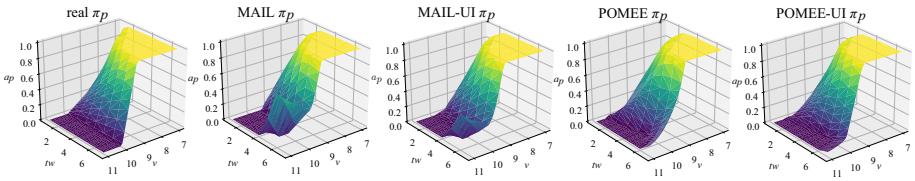
Implementation details We conduct four training methods on this artificial environment: POMEE, POMEE-UI, MAIL and MAIL-UI. The main difference between the first two methods and the second two methods is that there is no hidden policy in the MAIL and MAIL-UI settings. The main changes of MAIL-UI and POMEE-UI methods with respect to MAIL and POMEE methods are that the environment policy is implemented in the DUIN structure and the training process follows Algorithm 3. We aim to compare the similarity between the generated policies and the defined rules.

In detail, each policy or module is embodied by a neural network with 2 hidden layers and combined sequentially into a joint policy network illustrated in Fig. 2. There are 64 neurons in each hidden layer activated by tanh functions. To control the same complexity of the policy model, the joint policy networks in these four methods have the same number of hidden layers. The discriminator network adopts the same structure as each policy network. Different from GANs training, we perform $K = 3$ generator steps per discriminator step, and sample $N = 200$ trajectories per generator step. The detail of the training process is described in the previous sections.

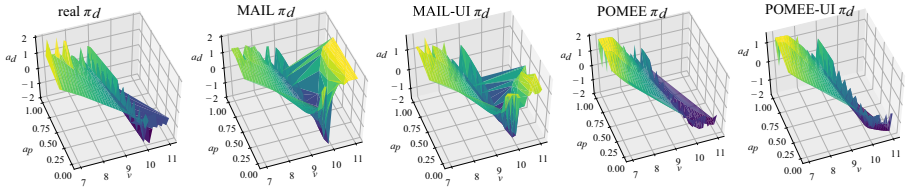
Results The generated policy functions trained by these four methods are shown in Fig. 10. First of all, from the perspective of the two observable policies, the policy function maps of π_p and π_d produced by POMEE-type methods are both more similar to the real function spaces than those by MAIL-type methods, as shown in Fig. 10a, b. MAIL-type methods produce sharp distortion shape locally when r is large. We believe that this is because the hidden variables have a greater impact on the interaction as r increases, and a large unobservable bias has reached a point where it cannot be neglected.

Additionally, compared with the basic methods MAIL and POMEE, the MAIL-UI and POMEE-UI methods can restore the policy function spaces more realistically. In particular, MAIL-UI can significantly alleviate the distortion in the policy function space learned by MAIL, which probably implies that the environment policy model implemented by a causal DUIN model can alleviate the hidden bias to some extent in the learning process.

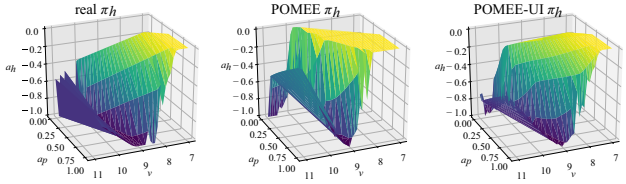
Then we further compare the similarity between the hidden policies generated by POMEE-type methods and the true policy π_h. In Fig. 10c, it can be seen that the generated hidden policies can describe threshold effects well and match the real function map roughly, although it is difficult under the setting of fully unobservable variables. Similar to the results of π_p and π_d, the hidden policy learned by POMEE-UI is closer to the real policy π_h than that learned by POMEE. Our results show the potential of using observational data to infer the hidden effect model.



(a) Artificial platform policy: the ground-truth, the learned by MAIL, the learned by MAIL-UI, the learned by POMEE and the learned by POMEE-UI.



(b) Artificial driver policy: the ground-truth, the learned by MAIL, the learned by MAIL-UI, the learned by POMEE and the learned by POMEE-UI.



(c) Artificial driver policy: the ground-truth, the learned by POMEE and the learned by POMEE-UI.

Fig. 10 Visualization and comparison of policy functions, with $r = 1.3$. More visualizations with various of r values are presented in "Appendix A.4"

6.2 Experiments on real world applications

Similar to the toy experiments in the previous subsection, the experiment on real-world application data is also divided into two steps: First, we evaluate the learning performance of the DUIN model on the randomized trial data collected from the real application system. Then, we apply POMEE-UI and several comparative methods to the real-world application data, and evaluate the performances of simulation and policy optimization. Finally, we deploy a recommender policy online, which is optimized in the POMEE-based environment, and results of online A/B test are reported at the end.

6.2.1 DUIN on real-world data

We apply DUIN to the real-world randomized trial dataset that is collected from the real-world recommender system. The dataset has 1.16 million recommendation record samples. Although the huge dataset can release the power of deep models, it involves a lot of noise data and a large randomness lies behind the observed outcome. It is still very challenging to infer the uplift effect from such real-world data.

We perform the Causal Forest method (Wager and Athey 2018), a popular algorithm for uplift modeling in observational studies, on this real-world dataset as a comparison. The Qini-Coefficient and Q^{TO} are used to evaluate the performance of two models.

Implementation details In the training of the DUIN model, we find that the frequency of alternate optimization, that is, the number of learning steps in one alternate round K in Algorithm 2, can affect the model performance to some extent. The model trained under $K = 5$ can have a better performance and stability than that under $K = 1$. We believe that the lower frequency of alternate optimization, that is, the larger value of K, can help the model eliminate the influence of noise and randomness.

Results The Qini curves of Causal Forest model and DUIN model are shown in Fig. 11. It can be seen that the Causal Forest model trained on the real-world dataset can only have a very small performance improvement compared with the random model. The DUIN model has a better performance overall despite poor performance in the middle part.

The values of Qini-Coefficient and Q^{TO} metrics are listed in Table 2. The Qini-Coefficient of the DUIN model is larger than that of the Causal Forest model, which shows a

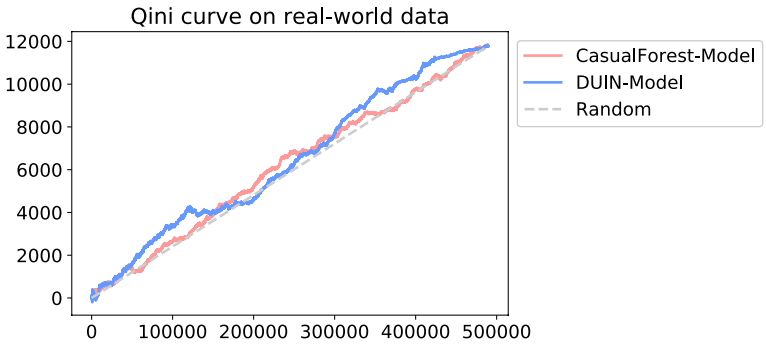


Fig. 11 Qini curves of two models evaluated on testing dataset: the Causal Forest model and the DUIN model. The Qini curve of the random model is also plotted as baseline to be compared

better ability to rank uplift. The Q^{TO} of the DUIN model is smaller than that of the Causal Forest model, which means a lower estimation error of the uplift value. These results can further demonstrate the ability of the DUIN model to infer the uplift effect.

6.2.2 Real-world experiment for POMEE-UI

In this part, we apply POMEE-UI to a real-world application of driver program recommendation as introduced in Sect. 5.1. We first use historical data to build different virtual environments by six comparative methods. We then evaluate these environments from various statistical measures. Finally, we train different recommender policies in these environments by the same training method, and evaluate these policies in offline and online environments. Specifically, we include six methods in our comparison:

- *SUP* Supervised learning of the driver policy with historical state-action pairs, i.e., behavioural cloning;
- *GAIL* GAIL to learn the driver policy, given the historical record of program recommendation as a static environment;
- *MAIL* Multi-agent adversarial imitation learning, without modeling the hidden variables.
- *MAIL-UI* MAIL-type method, in which the environment policy is implemented in the DUIN structure. The main difference between it and POMEE-UI is that it does not model the hidden variables, just like MAIL compared to POMEE;
- *POMEE* The proposed method described in Algorithm 1;
- *POMEE-UI* The proposed method described in Algorithm 3.

We evaluate the models by different statistical metrics.

Log-likelihood of real data on models We evaluate the learned policy distribution of six different models by the mean log-likelihood (MLL) of real state-action pairs on both training set and testing set. As shown in Table 3, the models trained by POMEE-type methods achieve the highest mean log-likelihood on both data sets. Since the evaluation is on the view of each state-action pair, the behavioural cloning method SUP achieves a better performance than MAIL-type methods. Meanwhile, the POMEE-type methods make a significant improvement compared with the MAIL-type methods, which indicates the positive influence of our hidden variables setting.

Correlation of key factors trend Another important measurement of generalization performance is the trend of drivers' response. We use the trend lines of two indicators to compare different simulators: number of Finished Orders (FOs) and Total Driver Incomes (TDIs). The same as above, we apply the simulator to a subsequent testing data and simulate the trends of FOs and TDIs. Then we calculate the Pearson correlation coefficient

Table 2 Comparison of Qini-Coefficients and Q^{TO} on real-world data by two uplift models: the Causal Forest model and the DUIN model

Methods	Qini-Coefficient	$\mathbf{Q^{TO}}$
Causal Forest	0.0137	2.4089
DUIN	**0.0809**	**2.3754**

The bold in tables indicates that the current method has the best performance of all the comparison methods on the current metric

Table 3 Comparison of mean log-likelihood by six different methods on the real-world test set

Methods	MLL on training set	MLL on testing set
SUP	17.09	18.00
GAIL	18.43	17.85
MAIL	15.27	14.52
MAIL-UI	16.22	15.48
POMEE	**21.74**	21.21
POMEE-UI	21.54	**21.32**

The bold in tables indicates that the current method has the best performance of all the comparison methods on the current metric

(PCC) between the simulated trend line and the real one. As shown in Table 4, the simulated trend lines of two indicators by POMEE and MAIL achieve high correlations to the real ones, with Pearson correlation coefficient of 0.8 approximately. While the methods SUP and GAIL, trained directly with static data, get lower performance in this evaluation. Though the PCC by the MAIL-UI and POMEE-UI methods is not the highest, these two methods still have a decent performance on this metric.

Distribution of driver response To further compare the generalization performance of models, we apply the built simulators to subsequent program recommendation records. We simulate the drivers' responses by using real program records on testing data, then compare the simulated distribution of drivers' responses with the real distribution. Here we use FOs as an indicator. Figure 12 shows the error of FOs distributions simulated in six simulators. The simulation distributions by SUP and GAIL are biased apparently when FOs are low. The reason is that these two methods use static real data directly for building simulators, which could limit the generalization performance of simulators, and the lower FOs mean the higher uncertainty, especially zero. The FOs distribution by POMEE is closer to the real one than that by MAIL, where the hidden variables setting makes difference explicitly. The same applies to POMEE-UI and MAIL-UI. In addition, it can be seen that the FOs distributions by MAIL-UI and POMEE-UI are respectively more realistic than those by MAIL and POMEE, which also shows the effect of the DUIN structure.

Policy evaluation results in offline environments In this part, we evaluate the effect of different simulators for policy optimization. First, we use the policy gradient method TRPO (Schulman et al. 2015) to optimize a recommender policy in each simulator. Then, by using

Table 4 Comparison of Pearson correlation coefficients on FOs and TDIs trend lines by six different methods

Methods	PCC on FOs	PCC on TDIs
SUP	−0.0213	0.0010
GAIL	0.4987	0.4252
MAIL	**0.8129**	0.7861
MAIL-UI	0.8023	0.7952
POMEE	0.7945	**0.8596**
POMEE-UI	0.8018	0.8342

The bold in tables indicates that the current method has the best performance of all the comparison methods on the current metric

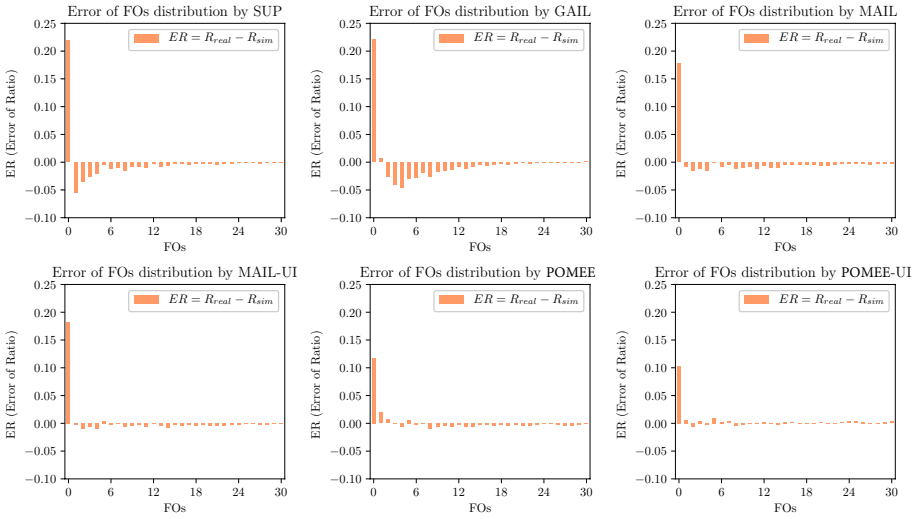


Fig. 12 Error of FOs distribution generated by six different methods on testing data. Y-axis is the error of FOs distribution between the simulation and the real. The original FOs distribution is presented in "Appendix C"

testing data, we build four virtual environments for policy evaluation by four methods, named *EvalEnv-MAIL*, *EvalEnv-MAIL-UI*, *EvalEnv-POMEE* and *EvalEnv-POMEE-UI* respectively. Given these four environments, we execute the optimized policies under a constrained budget, and compare the improvement of mean FOs. It would be expected that the simulator built by SUP or GAIL method would produce a policy that performs badly in the real environment because it is trained on static data.

As shown in Fig. 13, the policy $\pi_{POMEE-UI}$ optimized in the simulator built by POMEE-UI achieves best performance in all environments, while the policies π_{SUP} and π_{GAIL} perform bad in these environments. The promotion by π_{POMEE} compared to π_{MAIL} can further verify that training in a virtual environment with hidden variables can bring better performance to traditional reinforcement learning. Compared with MAIL and POMEE, the improvements by MAIL-UI and POMEE-UI demonstrate that an uplift model, used as a reward function in a simulator, could improve the performance of policy optimization than a response model. Additionally, the performance of policies π_{SUP}, π_{GAIL} shows a

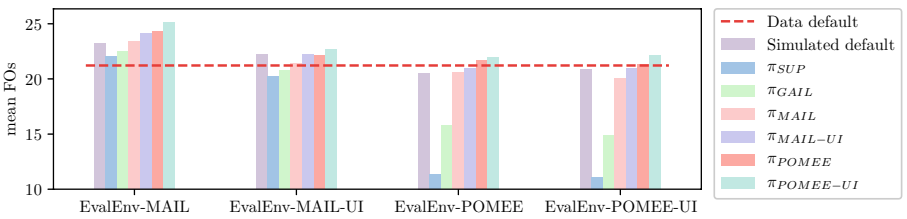


Fig. 13 Comparison of performance of different policies trained from different simulators in four evaluation environments: *EvalEnv-MAIL*, *EvalEnv-MAIL-UI*, *EvalEnv-POMEE* and *EvalEnv-POMEE-UI*. Y-axis is the mean FOs by executing different policies. The Data default is the mean FOs in the real testing data. The Simulated default is the mean FOs of the original simulation in each evaluation environment

significant degradation in *EvalEnv-POMEE* and *EvalEnv-POMEE-UI*, while not shown up in *EvalEnv-MAIL* and *EvalEnv-MAIL-UI*, which also indicates that the environment built with modeling hidden variables can recover the real environment more precisely.

Policy evaluation results in online A/B tests We further conduct online A/B tests to evaluate the effect of the policy π_{POMEE}. The online tests are conducted in three cities of different scales. The drivers in each city are divided randomly into two groups of equal size, namely the control group and the treatment group. The programs for the drivers in the control group are recommended by an existing recommendation policy, which can be viewed as a baseline policy. The drivers in the treatment group are recommended by π_{POMEE}. The results of online A/B tests are shown in Table 5. The policy π_{POMEE}, optimized in the simulator built by the proposed method POMEE in this study, achieves significant improvements on FOs and TDIs in all three cities, and the overall improvements are **11.74%** and **8.71%**, respectively.

7 Conclusion

This paper explores how to estimate a partially observable environment with uplift inference from the past data. We first propose the POMEE method following the generative adversarial training framework. We design the partially-observed environment model as an important part of the generator and make the discriminator compatible with two different classification tasks so as to guide the imitation of each policy precisely. To build a causal reward function in the virtual environment, we then propose a novel DUIN model to learn the uplift effect of each action. By implementing the environment policy in the DUIN structure, we propose the POMEE-UI approach to estimate the partially observable environment with an uplift inference module. Further, we apply POMEE-UI to build a virtual environment of driver program recommendation system on a large-scale ride-hailing platform, which is a highly dynamic and partially observable environment. Experiment results verify that the policies generated by POMEE-UI can be very similar to the real ones and have better generalization performance in various aspects. Furthermore, the simulator built by POMEE-type methods can produce a better policy with common RL training methods. It is worth noting that the proposed method POMEE-UI can be used not only in this task, but also in many other real-world partially observable environments.

Appendix A: Prerequisite knowledge

A.1 Causal inference

We consider a framework with N individuals indexed by i. Denoting $Y_i(1)$ the person i's outcome when he receives the active treatment and $Y_i(0)$ the person i's outcome when he receives the control treatment, the *causal effect*, τ_i, of the active treatment versus the control treatment is given by:

$$\tau_i = Y_i(1) - Y_i(0). \tag{23}$$

This causal effect is also named Individualized Treatment Effect (ITE). Researchers typically pay more attention to estimate the conditional Average Treatment Effect (cATE), that is, the expected causal effect of the active treatment for a subgroup in the population:

$$cATE : \tau(X_i) = \mathbb{E}[Y_i(1)|X_i] - \mathbb{E}[Y_i(0)|X_i], \tag{24}$$

where X_i is a representation vector of random variables (features). Of course, we will never observe both $Y_i(1)$ and $Y_i(0)$. Letting $W_i \in \{0, 1\}$ be a binary variable taking on value 1 if person i receives the active treatment, and 0 if person i receives the control treatment, the person i's observed outcome is actually:

$$Y_i^{obs} = W_i Y_i(1) + (1 - W_i) Y_i(0). \tag{25}$$

A popular but unfortunately wrong belief is that one can always estimate the cATE from the observational data by simply computing the empirical counterpart of

$$\mathbb{E}[Y_i^{obs}|X_i = x, W_i = 1] - \mathbb{E}[Y_i^{obs}|X_i = x, W_i = 0]. \tag{26}$$

This won't identify the cATE unless the assumption holds true that W_i is independent of $Y(1)$ and $Y(0)$ conditional on X_i. This assumption is the so-called *Unconfoundedness Assumption* or the *Conditional Independence Assumption* (CIA) commonly used in the social science and medical literature. This assumption holds true when treatment assignment is strictly random conditional on X_i:

$$CIA : \{Y_i(1), Y_i(0)\} \perp\!\!\!\perp W_i \,|\, X_i. \tag{27}$$

In causal inference, there is also an important concept, the propensity score $p(X_i) = P(W_i = 1|X_i)$, which represents the probability of treatment given X_i (Rosenbaum and Rubin 1983), which is a key to one direction of uplift modeling.

A.2 Uplift modeling

Uplift modeling amounts to estimating a cATE. Although companies can easily conduct randomized experiments so as to ensure that the CIA holds, the fact that we never observe the true $\tau(X_i)$ makes it seemingly impossible to use standard supervised learning algorithms to estimate it. The uplift literature has proposed three main approaches to estimate $\tau(X_i)$ despite the absence of the ground truth.

(1) Two-Model approach. It consists of modeling $\mathbb{E}[Y_i(1)|X_i]$ and $\mathbb{E}[Y_i(0)|X_i]$, one using the treatment group data and the other using the control group data, exclusively. This approach has been applied in several uplift works (Radcliffe 2007; Nassif et al. 2013) and is often used as a baseline model. The advantage of the Two-Model approach resides in its simplicity. Because inference is done separately in two models, state-of-the-art machine learning algorithms such as Random Forest (RF) (Breiman 2001) or XGBoost (Chen and Guestrin 2016) can be used on both the regression or the (multi-) classification settings. Although this approach has been shown well-performing (Zaniewicz and Jaroszewicz 2013; Athey and Imbens 2015), it may miss the "weaker" uplift signal, which is illustrated in simulation study (Radcliffe and Surry 2011).

(2) Class Transformation method. It was introduced by Jaskowski and Jaroszewicz (2012) in the case of binary outcome variable $Y_i^{obs} = \{0, 1\}$. This method needs to create a new target variable:

$$Z_i = Y_i^{obs} W_i + (1 - Y_i^{obs})(1 - W_i). \tag{28}$$

Under the assumption that control and treated groups are balanced across all profits of individual, that is, $p(X_i = x) = 0.5$ for all x, Jaskowski and Jaroszewicz (2012) proved that:

$$\tau(X_i) = 2P(Z_i = 1|X_i) - 1. \tag{29}$$

Uplift modeling then becomes to model $P(Z_i = 1|X_i)$, (i.e., $\mathbb{E}[Z_i = 1|X_i]$). The Class Transformation method is popular because it tends to show better performance than the Two-Model approach while still remaining simple. However, the two assumptions (binary outcome variable and balanced dataset between control and treatments) might seem to be restrictive. A generalization to unbalanced treatment assignment and to regression setups can be borrowed from Athey and Imbens (2015).

(3) Modeling uplift Directly. This approach generally modifies existing machine learning algorithms to directly infer a treatment effect. Lo (2002) proposed a strategy based on logistic regression, Su et al. (2012) and Guelman et al. (2014) focused on k-nearest neighbors while Zaniewicz and Jaroszewicz (2013) proposed a modification of the SVM model. Finally, Wager and Athey (2018); Guelman et al. (2015) provided a generalization to ensemble methods. Formally, in the case of a balanced randomized experiment, where the propensity score $p(X_i = x) = 0.5$ for all x, the estimator of the average treatment effect $\hat{\tau}$ is given by:

$$\hat{\tau} = \frac{\Sigma_i Y_i^{obs} W_i}{\Sigma_i W_i} - \frac{\Sigma_i Y_i^{obs}(1 - W_i)}{\Sigma_i (1 - W_i)}, \tag{30}$$

which corresponds to the difference in the sample average outcome between treated and untreated observations.

A.3 Uplift for target selection

The uplift under different observations can be split into four classes as shown in Fig. 14 by Michel et al. (2019). Each class is explained in detail as follows:

- A *Sure-thing* is the observation that would respond positively either treated or controlled, treating these observations might be a waste of resource.
- A *Sleeping* is the observation that would react negatively if treated but not if controlled. An example would someone that forgot a website subscription he was not using and just received an e-mail about it. Treating these observations would be a departure from the goal.
- A *lost cause* is the observation that would respond negatively no matter what happens. Treating these observations might also be a waste of resources.
- A *persuadable* is the observation that react positively to a treatment but would react negatively if controlled. These observations are the ones we should spend resources on.

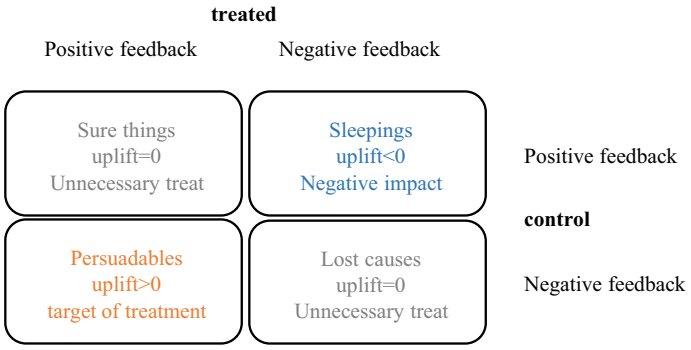


Fig. 14 The illustration of uplift value under different observation types. The vertical direction represents the potential outcome when treatment is applied, and the horizontal direction represents the potential outcome when controlled

Table 5 Results of online A/B tests on the platform of Didi Chuxing

Cities	ΔFOs(%)	ΔTDIs(%)
City A	+10.73	+6.16
City B	+10.16	+9.38
City C	+18.47	+17.84
Overall	+11.74	+8.71

Improvements of FOs and TDIs by policy $\boldsymbol{\pi}_{POMEE}$ in the treatment group comparing to the control group

A.4 Uplift evaluation metrics

The Qini curve is introduced in Radcliffe (2007) as a parametric curve with the following equation:

$$g(t) = Y_t^T - \frac{Y_t^C N_t^T}{N_t^C}.$$

where Y^T (respectively Y^C) and N^T (respectively N^C) are the sum of the treated (respectively control) individual outcomes and the number of treated (respectively control) individual units, and the t subscript indicates that the quantity is calculated for the first t units, sorted by the inferred uplift value, and $g(t)$ is the cumulative incremental gains of the first t units.

The calculation of Qini curve depends on gain charts, which are built by sorting the main population from the best to the worst lift performance and partitioning in segments. The Y-axis represents the cumulative incremental gains, that is $g(t)$ and the X-axis is the proportion of the population targeted, represented as t. There is an uplift curve and a random curve based on the calculation of every segment. The Qini-Coefficient is the difference between the area under the Qini curve and the random curve. The larger Qini-Coefficient, the better performance of the uplift model.

The Athey measure Q^{TO} proposed by Athey and Imbens (2015) is a measure similar to MSE in supervised learning, by exploiting the cATE-generating transformation. It is based on the fact that the expectation of cATE-generating transformed outcome $Y_i^\star$ is

equal to the true uplift if the assumption of unconfoundedness holds. The cATE-generating transformation of the outcome is defined as

$$Y_i^\star = Y_i^{obs} \cdot \frac{W_i - e(X_i)}{e(X_i) \cdot (1 - e(X_i))},$$

where $e(x) = Pr(W_i = 1|X_i = x)$ is the conditional treatment probability, or the propensity score. In the case with complete randomization the propensity score is constant $e(x) = p$ for all x, and the transformation simplifies to

$$Y_i^\star = Y_i^{obs} \cdot \frac{W_i - p}{p \cdot (1 - p)},$$

where $p = \mathbb{E}[e(X_i)] = \mathbb{E}[W_i] = pr(W_i = 1)$ is the common probability of assignment to the treatment. If there is a variation in the treatment effect, the estimation of $\mathbb{E}[Y_i^\star|X_i = x]$ based on the values of the pairs $(Y_i^\star, X_i)$ will be biased. Then the out-of-sample goodness-of-fit measure $Q^{os,TO}$ was proposed as

$$Q^{TO}(Y_i^\star, \tau(\hat{X}_i)) = \frac{1}{N} \sum_{i=1}^{N} \left(Y_i^\star - \hat{\tau}(X_i)\right)^2.$$

The smaller Q^{TO}, the better performance.

Appendix B: Illustration of the pipeline work

The approach proposed in this paper is a pipeline work demonstrated in Fig. 15, and can be summarized as following steps Fig. 16:

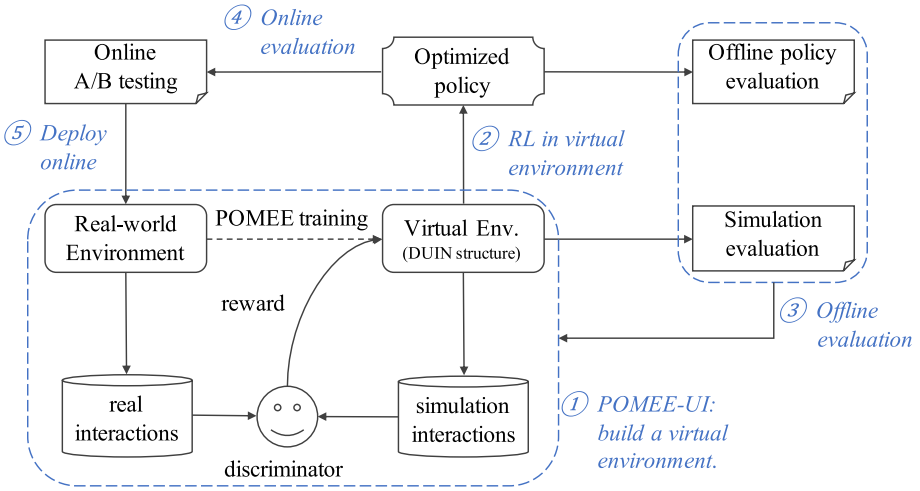


Fig. 15 Illustration of the pipeline work presented in this paper. There are mainly five steps for applying reinforcement learning methods to real-world applications on historical data, which are sorted by blue labels (Color figure online)

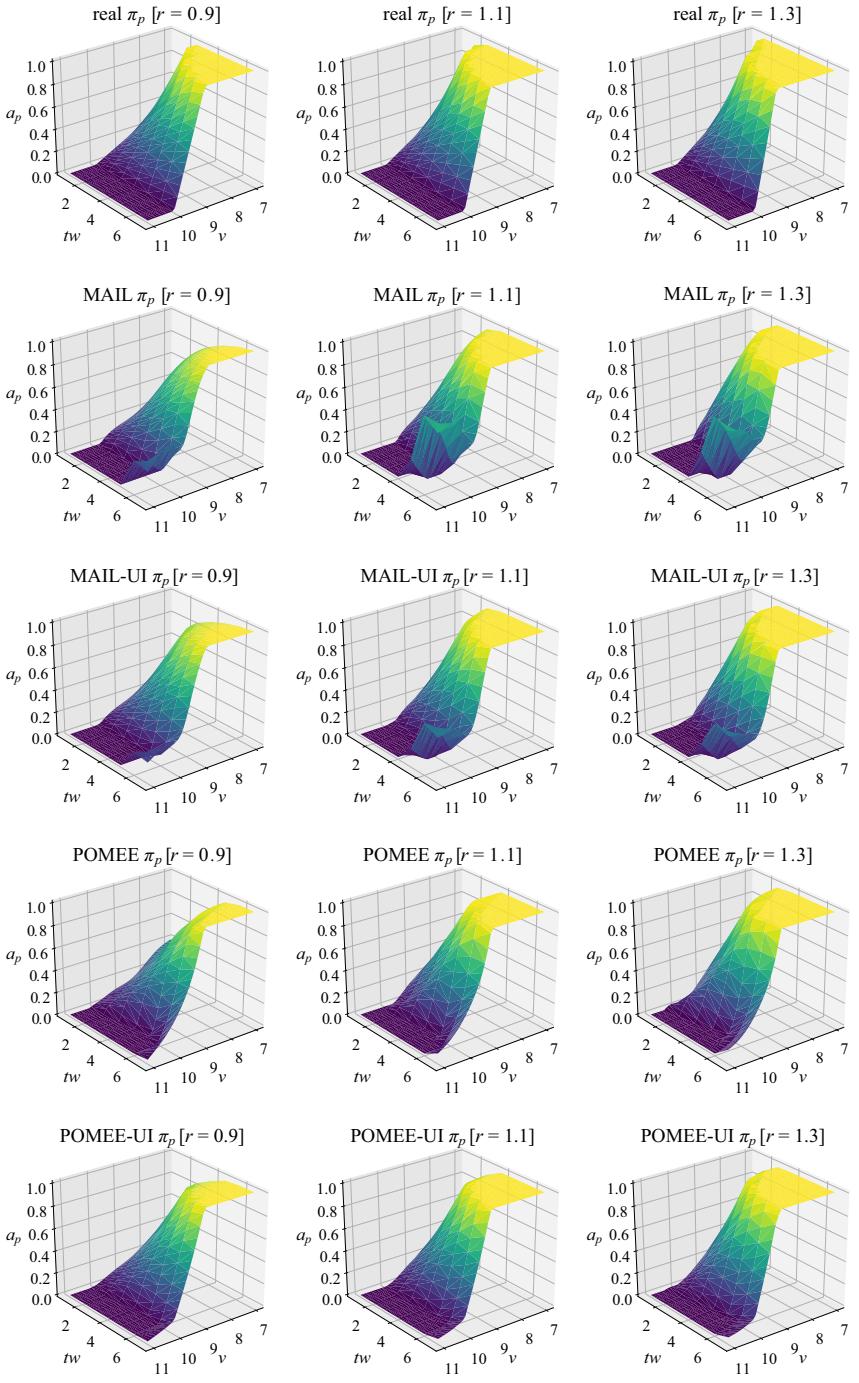


Fig. 16 Visualization of the artificial platform policy function π_p with respect to v and tw on different values of r. The first row is the ground-truth rule function. The second to the fourth rows are the platform policy functions generated by MAIL, MAIL-UI, POMEE and POMEE-UI respectively

(1) Performing POMEE-UI to generate a virtual environment: learn an environment model by POMEE using the real interactions, in which the environment model is implemented in the DUIN structure to make a causal reward mechanism.
(2) Conducting RL in the virtual environment: optimize the recommender policy by interacting with the virtual environment to maximize the cumulative reward.
(3) Offline evaluation: evaluate the simulation effect of the generated virtual environment model from several aspects of statistics to measure the gap from simulation to reality. The offline policy evaluation is conducted in the virtual environment built on the data of new phase.
(4) Online evaluation: online A/B testing to evaluate the policy performance in the real-world environment. The optimized policy is applied to the treatment group, and the control group is deployed with the existing policy as a comparison to evaluate the improvement effect of the optimized policy.
(5) Online deployment: a policy that has been validated by all of the above steps would be deployed online. Then, new interaction data collected in the real environment can be used to fine-tune the virtual environment model, thus forming a policy optimization closed-loop.

Appendix C: More experiment results

In this section, we show more results for experiments in Sect. 6.1.2 and 6.2.2 (Figs. 17, 18, 19). The function space learned by MAIL and POMEE under various r value are listed here. The original FOs distributions generated by different methods are shown in the final.

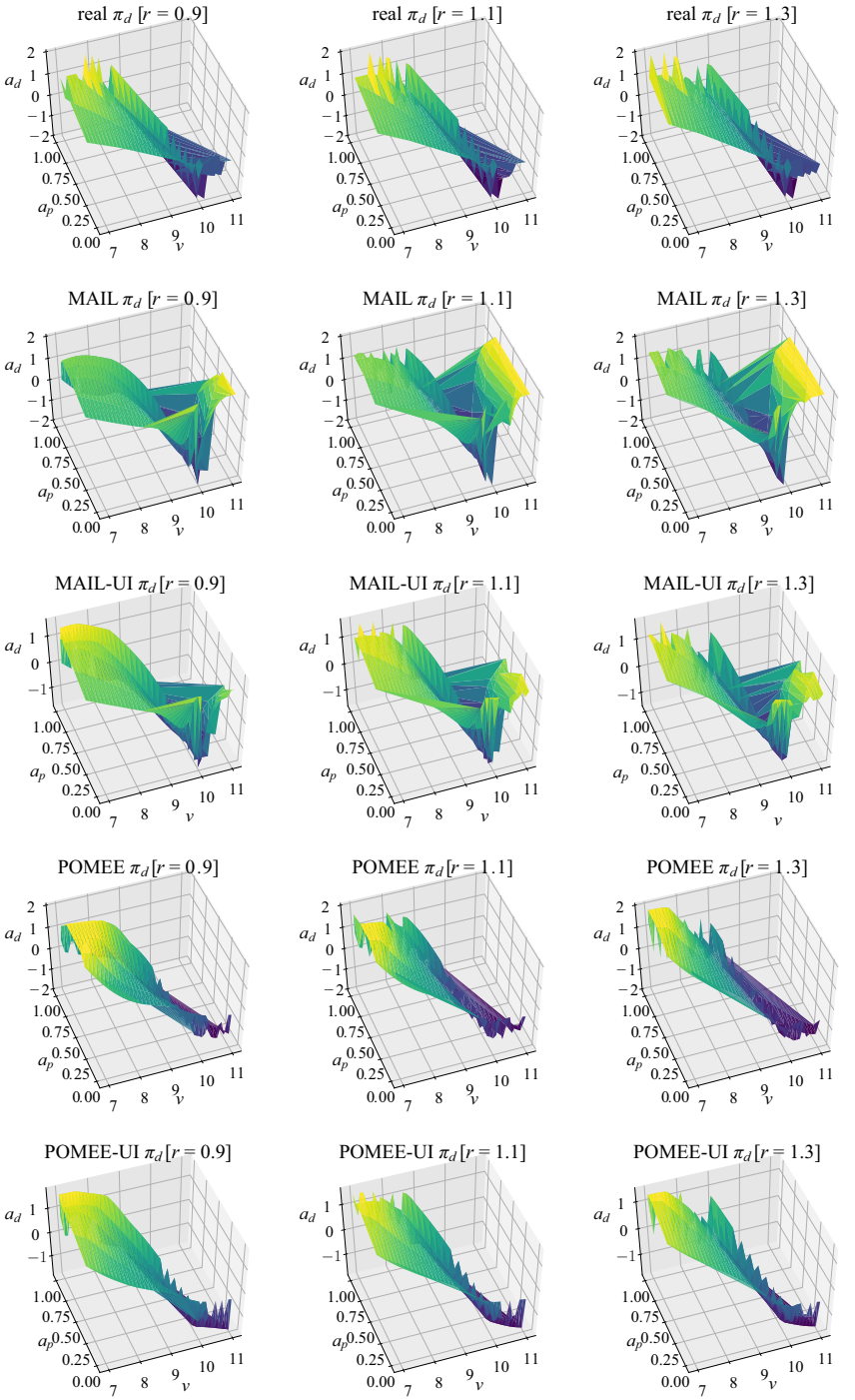


Fig. 17 Visualization of the artificial platform policy function π_d with respect to v and a_p on different values of r. The first row is the ground-truth rule function. The second to the fourth rows are the driver policy functions generated by MAIL, MAIL-UI, POMEE and POMEE-UI respectively

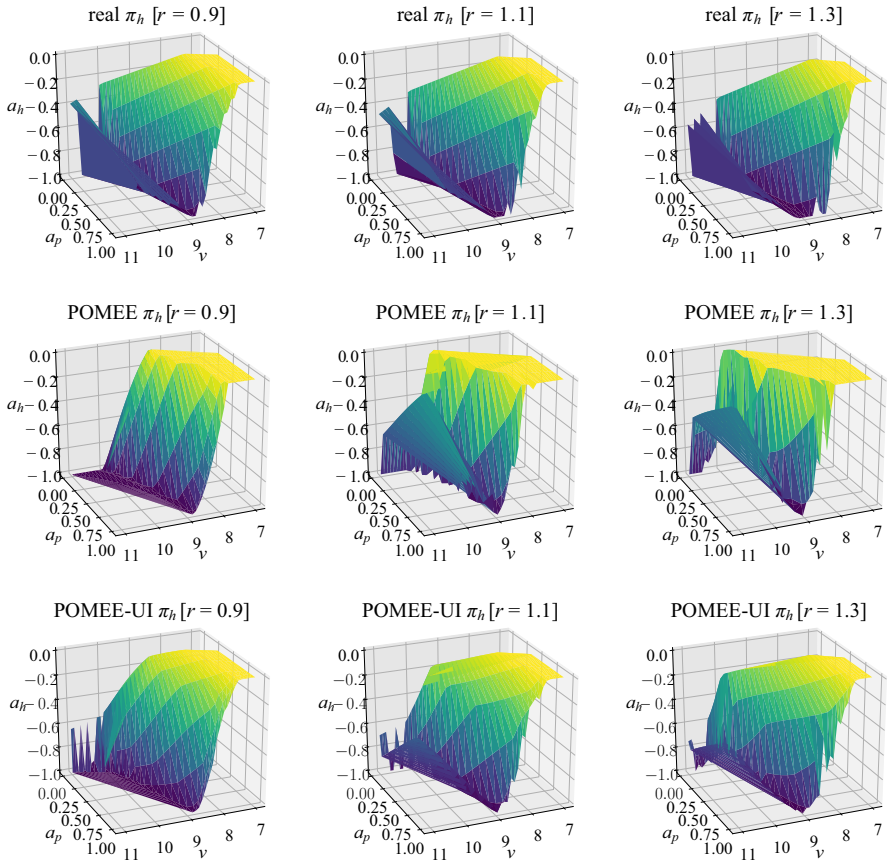


Fig. 18 Visualization of the artificial hidden policy function π_h with respect to v and a_p on different values of r. The first row is the ground-truth rule function. The second row is the hidden policy function generated by POMEE and the third row corresponds to POMEE-UI

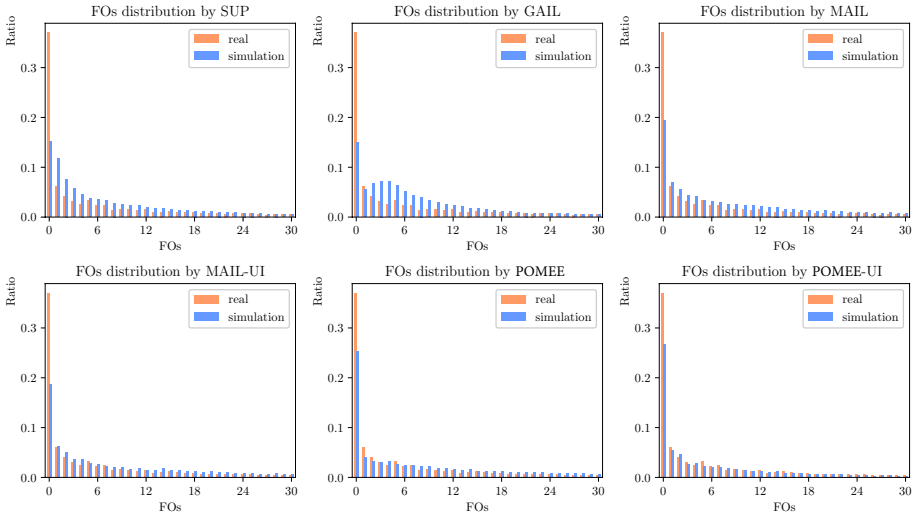


Fig. 19 The original FOs distribution generated by six different methods on testing data. Y-axis is the ratio of FOs distribution

Acknowledgements We would like to thank Prof. Yuan Jiang for her support to this work. We would also like to thank the anonymous reviewers for their very constructive comments. This work is supported by the National Key R&D Program of China (2018AAA0101100), NSFC (61876077), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

Agarwal, A., Bird, S., Cozowicz, M., Hoang, L., Langford, J., Lee, S., et al. (2016). Making contextual decisions with low technical debt. *CoRR* (abs/1606.03966).

Argall, B., Chernova, S., Veloso, M. M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems, 57*(5), 469–483.

Astrom, K. J. (1965). Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications, 10*(1), 174–205.

Athey, S., & Imbens, G. W. (2015). Machine learning methods for estimating heterogeneous causal effects. *Stat, 1050*(5), 1–26.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.

Brown, N., & Sandholm, T., (2017). Safe and nested subgame solving for imperfect-information games. In *Advances in neural information processing systems 30: annual conference on neural information processing systems 2017*, pp. 689–699.

Cassandra, A., Nodine, M., Bondale, S., Ford, S., & Wells, D. (2005). Using pomdp-based state estimation to enhance agent system survivability. In *IEEE 2nd symposium on multi-agent security and survivability*, pp. 11–20.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 785–794.

Chen, X., Li, S., Li, H., Jiang, S., Qi, Y., & Song, L. (2019). Generative adversarial user model for reinforcement learning based recommendation system. In *Proceedings of the 36th international conference on machine learning, ICML*, Vol. 97, pp. 1052–1061.

Finn, C., Christiano, P. F., Abbeel, P., & Levine, S. (2016). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *CoRR* abs/1611.03852.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems, 27,* 2672–2680.

Guelman, L., Guillén, M., & Pérez Marín, A. M. (2014). Optimal personalized treatment rules for marketing interventions: A review of methods, a new proposal, and an insurance case study. UB Riskcenter Working Paper Series.

Guelman, L., Guillén, M., & Pérez-Marín, A. M. (2015). Uplift random forests. *Cybernetics and Systems, 46*(3–4), 230–248.

Gutierrez, P., & Gérardy, J. Y. (2017). Causal inference and uplift modelling: A review of the literature. In *International conference on predictive applications and APIs*, pp. 1–13.

Hansotia, B., & Rukstales, B. (2002). Incremental value modeling. *Journal of Interactive Marketing, 16*(3), 35.

Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. *Advances in Neural Information Processing Systems, 29,* 4565–4573.

Jaskowski, M., & Jaroszewicz, S. (2012). Uplift modeling for clinical trial data. In *ICML Workshop on clinical data analysis*.

Johansson, F. D., Shalit, U., & Sontag, D. A. (2016). Learning representations for counterfactual inference. In *Proceedings of the 33nd international conference on machine learning, ICML*, Vol. 48, pp. 3020–3029.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*(1–2), 99–134.

Künzel, S. R., Sekhon, J. S., Bickel, P. J., & Yu, B. (2019). Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences, 116*(10), 4156–4165.

Lo, V. S. Y. (2002). The true lift model—a novel data mining approach to response modeling in database marketing. *ACM SIGKDD Explorations Newsletter, 4*(2), 78–86.

Menick, J., & Kalchbrenner, N. (2018). Generating high fidelity images with subscale pixel networks and multidimensional upscaling. *CoRR* abs/1812.01608.

Michel, R., Schnakenburg, I., & Martens, T. (2019). *Targeting uplift: An introduction to net scores*. Springer

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518,* 529–533.

Nassif, H., Kuusisto, F., Burnside, E. S., & Shavlik, J. W. (2013). Uplift modeling with ROC: An SRL case study. In *Late breaking papers of the 23rd international conference on inductive logic programming, CEUR workshop proceedings*, Vol. 1187, pp. 40–45.

OpenAI, B. C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *CoRR, 1912,* 06680.

Pineau, J., Gordon, G. J., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for pomdps. In *Proceedings of the eighteenth international joint conference on artificial intelligence*, pp. 1025–1032.

Pomerleau, D. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation, 3*(1), 88–97.

Qin, Z. T., Tang, X., Jiao, Y., Zhang, F., Xu, Z., Zhu, H., & Ye, J. (2020). Ride-hailing order dispatching at didi via reinforcement learning. *INFORMS Journal on Applied Analytics, 50*(5), 272–286.

Radcliffe, N. J. (2007). Using control groups to target on predicted lift: Building and assessing uplift models. *Direct Marketing Analytics Journal, 1,* 1421.

Radcliffe, N. J., & Surry, P. D. (2011). Real-world uplift modelling with significance-based uplift trees. White Paper TR-2011-1, Stochastic Solutions pp. 1–33.

Rosenbaum, P. R., & Rubin, D. B. (1983). The central role of the propensity score in observational studies for causal effects. *Biometrika, 70*(1), 41–55.

Ross, S., Gordon, G. J., & Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635.

Russell, S. J. (1998). Learning agents for uncertain environments (extended abstract). In *Proceedings of the eleventh annual conference on computational learning theory*, pp. 101–103.

Rzepakowski, P., & Jaroszewicz, S. (2012). Decision trees for uplift modeling with single and multiple treatments. *Knowledge and Information Systems, 32*(2), 303–327.

Sadeghi, F., & Levine, S. (2016). CAD^2RL: Real single-image flight without a single real image. *CoRR* abs/1611.04201.

Sallans, B. (1999). Learning factored representations for partially observable Markov decision processes. *Neural Information Processing Systems, 12,* 1050–1056.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences, 3*(6), 233–242.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., & Moritz, P. (2015). Trust region policy optimization. In *Proceedings of the 32nd international conference on machine learning*, pp. 1889–1897.

Shang, W., Yu, Y., Li, Q., Qin, Z., Meng. Y., & Ye, J. (2019). Environment reconstruction with hidden confounders for reinforcement learning based recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 566–576.

Shi, J., Yu, Y., Da, Q., Chen, S., & Zeng, A (2018) Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. *CoRR* abs/1805.10000.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature, 529*(7587), 484–489.

Singh, S. P., Jaakkola, T. S., & Jordan, M. I. (1994). Learning without state-estimation in partially observable markovian decision processes. In *Proceedings of the eleventh international conference on machine learning*, pp. 284–292.

Su, X., Kang, J., Fan, J., Levine, R. A., & Yan, X. (2012). Facilitating score and causal inference trees for large observational studies. *Journal of Machine Learning Research,* 13(Oct):2955–2994.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

Tzeng, E., Devin, C., Hoffman, J., Finn, C., Abbeel, P., Levine, S., et al. (2016). Adapting deep visuomotor representations with weak pairwise constraints. *Algorithmic foundations of robotics XII, Proceedings of the twelfth workshop on the algorithmic foundations of robotics, WAFR,* Vol. 13, pp. 688–703.

Wager, S., & Athey, S. (2018). Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association, 113*(523), 1228–1242.

Ye, Z., Zhang, L., Xiao, K., Zhou, W., Ge, Y., & Deng, Y. (2018). Multi-user mobile sequential recommendation: An efficient parallel computing paradigm. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2624–2633.

Ye, Z., Xiao, K., Ge, Y., & Deng, Y. (2019). Applying simulated annealing and parallel computing to the mobile sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering, 31*(2), 243–256.

Zaniewicz, L., & Jaroszewicz, S. (2013). Support vector machines for uplift modeling. In *13th IEEE international conference on data mining workshops, ICDM workshops*, pp. 131–138.

Zhao, P., Cai, L. W., & Zhou, Z. H. (2020). Handling concept drift via model reuse. *Machine Learning, 109*(3), 533–568.

Authors and Affiliations

Wenjie Shang[1] · Qingyang Li[1] · Zhiwei Qin[1] · Yang Yu[2] · Yiping Meng[1] · Jieping Ye[1]

Qingyang Li
qingyangli@didiglobal.com

Zhiwei Qin
qinzhiwei@didiglobal.com

Yang Yu
yuy@nju.edu.cn

Yiping Meng
mengyipingkitty@didiglobal.com

Jieping Ye
yejieping@didiglobal.com

[1] AI Labs, Didi Chuxing, Beijing, China

[2] National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China