BCORLE(λ): An Offline Reinforcement Learning and Evaluation Framework for Coupons Allocation in E-commerce Market

Li Taishan [*1] **Yang Hongdi** [*1] **Zhao Yibo** [*1] **Zhao Yida** [*1]

Abstract

In this project, we dive into the paper "BCORLE(λ): An Offline Reinforcement Learning and Evaluation Framework for Coupons Allocation in E-commerce Market" (Zhang et al., 2021). This paper formulates the coupon allocation problem in the e-commerce market and solves it using an offline reinforcement learning strategy. We thoroughly study and comprehend the methodology in it. Besides, we conduct experiments to evaluate the method and reproduce the result presented in the paper.

1. Introduction

As the competition in e-commerce platforms becomes more and more violent, those e-commerce giants such as Taobao try to use monetary rewards to encourage users to log in to their platform and consume. The problem of Coupon allocation emerges when we want to motivate users by sending coupons every day under a limited budget.

We can formulate the coupons allocation problem as a constrained Markov decision process(CMDP), which can be converted into a Lagrangian dual problem. Then we can solve it by reinforcement learning (RL) methods. In particular, due to the fact that we cannot test any policy under the high financial risk, we can only use an offline learning strategy. There are many troubles when we intend to solve this problem by offline RL. This paper proposes several mechanisms to solve them. The main contribution of it is:

- It proposes an offline RL framework called BCORLE(λ) to solve the coupons allocation problem.
- An improved offline RL algorithm called resemble batch-constrained Q-learning (R-BCQ) is proposed. It is a hybrid of two popular offline RL algorithm: batch-constrained Q-learning (BCQ) and random ensemble mixture (REM).
- A model-free evaluation method called random ensemble mixture evaluator (REME) is proposed.

2. Problem Formulation

This section formulates the coupons allocation problem as a CMDP and then converts it into a Lagrangian dual problem.

2.1. Coupons allocation as CMDP

Suppose we want to allocate coupons to users once a day. And the period of the allocation is T days. The coupon allocation with budget b can be formulated as a CMDP as $< S, A, P, R, C, \gamma, b >$. S is the state space consisting of the user's information and the historical retention information of the users on the shopping platform. A is the action space where we choose the value of a coupon. P are the transition probability. R is the reward function which is -1 if a user does not log in to the platform, and 0 otherwise. C is the cost function which is the value of the coupons. γ and γ' are the discount factors for reward and cost respectively. b is the total budget of coupon allocation in T days.

As stated before, we can only do offline learning here. Namely, we learn from a collected dataset D, where $D = (s_i, a_i, r_i, c_i, s_{i+1})_{i=1}^{M}$. Then, formulate the problem as:

$$max_{\pi \in \Pi} J(\pi) = \mathbb{E}_{\tau \sim \pi, \mu} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

$$s.t. \quad C(\pi) = \mathbb{E}_{\tau \sim \pi, \mu} \left[\sum_{t=0}^{T} \gamma'^t c_t \right] \leq b,$$

where π is the learned policy, Π is the set of all policies, and τ is the distribution of the interaction between the agent and the environment when the initial state distribution is μ and the agent follows the policy π.

2.2. Lagrangian Dual Problem

To solve the above problem, we resort to the Lagrangian Multiplier method. Then, our problem is converted into:

$$\begin{aligned}\max_{\pi\in\Pi} L(\pi,\lambda) = \quad & \mathbb{E}_{\tau\sim\pi,\mu}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \\ & -\lambda\left(\mathbb{E}_{\tau\sim\pi,\mu}\left[\sum_{t=0}^{T}\gamma'^t c_t\right]-b\right) \\ s.t. \quad & \lambda\geq 0\end{aligned}$$

It is reasonable that we assume there is a policy π satisfying the constraint $C(\pi) < b$. Then prior problem is equivalent to the Lagrangian dual problem:

$$\min_{\lambda\in\mathbb{R}_+}\max_{\pi\in\Pi} L(\pi,\lambda).$$

Thus we can solve it by finding an optimal vaqlue of λ^* and the corresponding π^*_λ. Now we give a proof of the existence of the solution. First we make the following assumptions which are reasonable in real world.

Assumption 1. Given λ_a and λ_b, if $C(\pi^*_{\lambda_a}) \geq C(\pi^*_{\lambda_b})$, then $J(\pi^*_{\lambda_a}) \geq J(\pi^*_{\lambda_b})$, where $\pi^*_{\lambda_a} = \arg\max_\pi L(\pi,\lambda_a)$ and $\pi^*_{\lambda_b} = \arg\max_\pi L(\pi,\lambda_b)$.

This assumption means that the greater the distributed coupon amount is, the greater the user's retention rate will be.

Assumption 2. There exists λ_a and λ_b , making the condition $C(\pi^*_{\lambda_a}) < b$ and $C(\pi^*_{\lambda_b}) > b$ hold.

This assumption is true in the real world. When $\lambda \to 0$, there is no constraint on the budget, then the optimal policy will allocate the largest amount coupon to users, which makes the total cost exceeds the budget b. When $\lambda \to \infty$, the optimal policy will allocate the smallest amount of coupons to users, causing the total cost is less than the budget b.

Theorem 1. $C(\pi^*_\lambda)$is monotonically non-increased with the increase of λ, i.e., If $\lambda_A \leq \lambda_b$, then $C(\pi^*_{\lambda_a}) \geq C(\pi^*_{\lambda_b})$.

Proof. By the definition of π^*_λ, we have

$$\begin{aligned}J(\pi^*_{\lambda_a}) - \lambda_a\left(C(\pi^*_{\lambda_a})-b\right) &\geq J(\pi^*_{\lambda_b}) - \lambda_a\left(C(\pi^*_{\lambda_b})-b\right) \\ J(\pi^*_{\lambda_b}) - \lambda_b\left(C(\pi^*_{\lambda_b})-b\right) &\geq J(\pi^*_{\lambda_a}) - \lambda_b\left(C(\pi^*_{\lambda_a})-b\right)\end{aligned}$$

By adding the two sides of the above two inequalities, we have

$$\begin{aligned}&J(\pi^*_{\lambda_a}) - \lambda_a\left(C(\pi^*_{\lambda_a})-b\right) + J(\pi^*_{\lambda_b}) - \lambda_b\left(C(\pi^*_{\lambda_b})-b\right) \\ &\geq J(\pi^*_{\lambda_b}) - \lambda_a\left(C(\pi^*_{\lambda_b})-b\right) + J(\pi^*_{\lambda_a}) - \lambda_b\left(C(\pi^*_{\lambda_a})-b\right)\end{aligned}$$

After some reduction, we have

$$(\lambda_a-\lambda_b)(C(\pi^*_{\lambda_a}) - C(\pi^*_{\lambda_b})) \geq 0$$

Because $\lambda_a \leq \lambda_b$, we can conclude that $C(\pi^*_{\lambda_a}) \leq C(\pi^*_{\lambda_b})$.

Theorem 2. Under Assumption 2 and Assumption 3, their exists an optimal Lagrangian multiple variable λ^* which can make its corresponding optimal policy π^*_λ maximize the objective function $J(\pi)$ while satisfying the budget constraint.

Proof. According to the Theorem 1 and Assumption 2, we can find a λ^* which satisfies the condition $C(\pi^*_{\lambda^*}) = b$. And according to the Assumption 1, the objective $J(\pi)$ is maximized when the cost $C(\pi)$ equals to the budget b under the constraint $C(\pi) \leq b$. Therefore, we can conclude that there exists λ^* and its corresponding optimal policy $\pi^*_{\lambda^*}$, which makes the objective $J(\pi)$ is maximized and the constraint is satisfied.

3. Methodology

We have shown the problem we face and the existence of the solution. In this section, we introduce the BCORLE(λ) framework, which contains the R-BCQ offline learning method, REME method for evaluation, and λ-generalization method that enables the policy learning and evaluation can be performed with different values of λ.

3.1. λ-Generalization

First, we transfer the Lagrangian problem into a RL problem:

$$\begin{aligned}L(\pi,\lambda) &= \mathbb{E}_{\tau\sim\pi,\mu}\left[\sum_{t=0}^{T}\gamma^t r_t\right] - \lambda\left(\mathbb{E}_{\tau\sim\pi,\mu}\left[\sum_{t=0}^{T}\gamma'^t c_t\right]-b\right) \\ &= \mathbb{E}_{\tau\sim\pi,\mu}\left[\sum_{t=0}^{T}\gamma^t r_t - \lambda\gamma'^t c_t\right] + \lambda b \\ &= \mathbb{E}_{\tau\sim\pi,\mu}\left[\sum_{t=0}^{T} r^\lambda(s_t,a_t)\right] + \lambda b,\end{aligned}$$

where $r^\lambda(s_t,a_t) = \gamma^t r_t - \lambda\gamma'^t c_t$. Then we can solve the Lagrangian problem by solving the corresponding RL problem whose goal is to maximize $\mathbb{E}_{\tau\sim\pi,\mu}\left[\sum_{t=0}^{T} r^\lambda(s_t,a_t)\right]$ under the budget limit b.

What is unusual in this RL problem is that the reward function varies in terms of λ, which is bound to whether the policy exceeds the budget limit. Prior work deals with this by an iterative method. It first fixes λ and learns the optimal policy. Then it checks whether the cost of the policy is within the range of the budget. If not, it updates λ, then

learns the optimal policy again until it matches the budget. This method has high computation overhead as it must re-learn the optimal policy when λ is updated.

λ-generalization method solves this problem by incorporating λ into the transition tuple, which integrates the policy learning processes for different values of λ. Here are the details.

Suppose we have a dataset $D = \{(s_i, a_i, r_i, c_i, s_{i+1})\}_{i=1}^{M}$. It first incorporates λ's into the transition tuples. For each $(s_i, a_i, r_i, c_i, s_{i+1})$, we extend it to $(s_i, a_i, r_i, c_i, s_{i+1}, \lambda_j)$ for each $\lambda_j \in \{\lambda_1, \lambda_2, \ldots, \lambda_L\}$, where the possible values of λ can be predefined. Then our dataset becomes $D\prime = \{(s_i, a_i, r_i, c_i, s_{i+1}, \lambda_j)\}_{i,j=1,1}^{M,L}$, which is L times the original one.

In this way, we can learn policy for different λ's simultaneously. Then, for each policy $\pi(s, \lambda)$, the evaluation network estimates its cost $\hat{C}(\pi(s, \lambda))$. In the end, we can choose a policy $\pi(s, \lambda)$ such that $\mathbb{E}\left[\hat{C}(\pi(s, \lambda))\right] \in [b - \delta, b]$.

So far, we have combined different values of λ with policy learning. Then the question is how do we choose a set of possible values of λ, i.e. $\{\lambda_1, \lambda_2, \ldots, \lambda_L\}$. We obtain the values by first finding an appropriate interval $[\lambda_{min}, \lambda_{max}]$ and then sampling from it. Here we set λ_{min} to 0, otherwise the cost function may play a positive role in the reward r^{λ}. To determine λ_{max}, we need some extra inferences.

Define our action set as $\{a_1, a_2, \ldots, a_k\}$. Suppose that the $a_i < a_j$ for $i < j$. It is obvious that the retention and the reward are monotonically ascending in terms of the action, i.e. $r_i < r_j, c_i < c_j,$ for $\forall a_i < a_j$.

Theorem 3. When $\lambda \geq \max_{1<i\leq k} \frac{r_1 - r_i}{c_1 - c_i}$, the optimal coupons allocation policy for λ is always distributing coupons with minimum value to users.

Proof. By Theorem 1, the cost is non-decreasing with respect to λ. Thus we only need to prove that when $\lambda = \max_{1<i\leq k} \frac{r_1 - r_i}{c_1 - c_i}$, the optimal policy is distributing the minimum value c_1 to users.

As $\lambda = \max_{1<i\leq k} \frac{r_1 - r_i}{c_1 - c_i}$, we have

$$\lambda \geq \frac{r_1 - r_i}{c_1 - c_i} \text{ for } \forall 1 < i \leq k,$$

which is equivalent to

$$r_1 - \lambda c_1 \geq r_i - \lambda c_i.$$

This means that the optimal policy will always choose action a_1, which has the minimum cost.

Thus we set $\lambda_{max} = \max_{1<i\leq k} \frac{r_1 - r_i}{c_1 - c_i}$, in which case the optimal policy will distribute the minimum value to users and the cost will exceed the budget. Finally, we choose the interval $[0, \max_{1<i\leq k} \frac{r_1 - r_i}{c_1 - c_i}]$, and sample from it to obtain $\{\lambda_1, \lambda_2, \ldots, \lambda_L\}$.

3.2. Offline Policy Learning: R-BCQ

Up to now, we have converted the coupons allocation problem into an RL problem. It is time to figure out how to learn the optimal policy. As stated before, if we use online learning, there potentially exists financial risks as we need to know the response of users in the real world after we take an action. For this reason, we can only do offline learning here. There are two kinds of common offline RL methods: batch-constrained Q-learning (BCQ) and random ensemble mixture (REM). The former can alleviate the mismatch between the state-action pairs generated in the learned policy and in the training dataset. The latter gives us a better generalization.

R-BCQ method just combines the strength of the above two. First, it introduces a generative model that controls how the state-action pairs are generated. Denote $G(a \mid s, \lambda; \omega)$ as this model, where ω is its parameter. Given state s and Lagrangian variable λ, we only consider action a such that $G(a \mid s, \lambda; \omega) / \max_{a\prime} G(a\prime \mid s, \lambda; \omega) > \beta$, where β is a hyper parameter which controls how many pairs are dropped.

Secondly, R-BCQ also uses the same idea in REM method. It uses multi-head network to estimate the Q-value. Specifically, given a tuple of (s, a, λ), each head of the network gives a Q-value $Q_i(s, a, \lambda)$. Then we estimate the Q-value as $Q(s, a, \lambda) = \sum_i \alpha_i Q_i(s, a, \lambda)$, where $\sum_i \alpha_i = 1, \alpha_i > 0, \forall i$.

Combining the two ideas, the learned policy is chosen as:

$$\pi(s, \lambda) = \underset{a|G(a|s,\lambda;\omega)/\max_{a\prime} G(a\prime|s,\lambda;\omega)>\beta}{\arg\max} \sum_i \alpha_i Q_i(s, a, \lambda).$$

The loss of the policy learning network with parameter θ is:

$$L(\theta) = \mathbb{E}_{s,a,r^{\lambda},s\prime,\lambda\sim D\prime} \left[l_k \left(r^{\lambda} + \gamma \max_{a\prime \in A} \sum_i \alpha_i Q_i\prime(s\prime, a\prime, \lambda) - \sum_i \alpha_i Q_i(s, a, \lambda) \right) \right]$$

, where $A = \{a\prime \mid G(a\prime \mid s, \lambda; \omega) / \max_{\hat{a}} G(\hat{a} \mid s, \lambda; \omega) > \beta\}$. And the loss of the generative model is $L(\omega) = \mathbb{E}_{s,a,\lambda\sim D\prime}[-\log G(a \mid s, \lambda; \omega)]$.

3.3. Policy Evaluation: REME

After learning a policy, we need to evaluate its cost and utility. We train an evaluation network that is similar to the multi-head learning network to estimate the expected cost and utility respectively. What is different here is that we

compute the target value using the evaluated policy π rather than taking $\max$ over all possible actions. Thus the loss is defined as:

$$L(\theta) = \mathbb{E}_{s,a,r,s',\lambda\sim D'} \left[l_k \left(r + \gamma \sum_i \alpha_i \hat{Q}_i(s', \pi(s', \lambda), \lambda) - \sum_i \alpha_i \hat{Q}_i(s, a, \lambda) \right) \right]$$

. The reward function is replaced by cost function when we want to evaluate cost.

4. Experiments

The original paper showes their results of conducting experiments on both simulation environment and real-world platform Taobao Deals. As we don't have access to platforms such as Taobao Deals, we decide to reproduce the results in the simulation environment.

Here is the link for our code: https://github.com/zhaoyd1/CS282_Final

4.1. Simulation Environment

We follow the original setting of 10000 users, 30 days time span, and the coupons ranging from 0.1 Yuan to 2.1 Yuan with the interval size of 0.1 Yuan.

Also for each user, the probability of her logging in is uniformly distributed with the assumption: $P_{c_i} \geq P_{c_j}$ if $c_i \geq c_j$ where each c_i means the value of coupon she received the day before. The probability of her logging in again will decrease with proportion to the time span from her last logging in.

Each state can be defined as the combination of days that the user received coupons of different values and the reward can be defined as 0 and -1, each representing whether the user logs in or not the day after receiving a coupon.

4.2. Evaluation

We use the same evaluation metrics with the original paper. There are three metrics in all which are **AvgLogins**, **AvgCost**, **ROI** (defined as the **AvgLogins/AvgCost**). For **AvgLogins** and **ROI**, the larger means the better performance. For **AvgCost**, the smaller, the better.

4.3. Python Environment Setup

When trying to run the code given by the author, we find the python environment setup is quite hard because the **requirements.txt** in the original file was totally wrong and a lot of modules needed are not included in the requiring list.

Therefore, we try to set up a suitable environment from the very beginning and found that the author imported many old APIs, which are either outdated or complicated to use. We update some APIs and modified the corresponding codes. However, the codes live in an inherently conflicting environment, making it quite hard to determine which dependencies should be solved and which could be ignored.

After our work, eventually, we ignore some dependency problems which won't affect our code and manage to set up the environment to run the corresponding code and provided new **Readme.md** and **requirement.txt** to specifically describe how to set up the environment correctly.

4.4. Codes Modification

The original experiment uses the codes from **offlinerl** (Polixir, 2022) and some other existing works, but due to the version updating of these codes, we have to modify the training part and data generator to fit in the new APIs and merge the APIs from different versions to get our own **offlinerl**.

The author also uses some closed source extensions of TensorFlow only accessible on the PAI platform of Alibaba, which also needs modifications so that we could train the model on our local machines or the school cluster.

Moreover, we have already fixed many obvious and critical bugs, including

- Tens of variables being referenced before defined.
- Incorrect number and order of initialization arguments of **DQNAgent** when evaluated by **REME**.
- using inbuilt functions of class **String** to obvious **np.ndarray** type variables and directly using **float()** on a list.
- using the absolute path of the author's computer for every file.

Unfortunately, there are still some bugs and hazards that we failed to solve, like the unknown assertion error when training MOPO and the unaligned tensors during evaluation.

To sum up, we mainly modifies the code in **offlinerl**, **utils**, **train.py** and **evaluation.py**.

4.5. Results

We generate the simulated user data by **generator.py** and trained it by different methods. We get similar results as the original paper and we find not only did R-BCQ have better performance but also cost less time for training, which also proves the strength of R-BCQ in computation time.

For different off-policy evaluation methods simulation, we only manage to modify and run the code for **REME** and

Method	
R-BCQ	0.359h
REM	0.492h
BCQ	1.933h

FQE while failed to evaluate by **IS**, **DM** and **DR** as there are too many bugs and misleading comments mentioned above.

For **REME** and **FQE**, we get almost the same results on the errors of Q-value of **AvgCost** when the learning step is 1000 (**REME**: 0.0065 and **FQE**: 0.0744 for $\lambda = 0$) and it shows that **REME** achieves less error than **FQE**. Combined with the results of the original paper, the errors seem to drop when the learning step increase from 1000 to 2000 while rising again if the learning step is set to 4000. It may imply that learning too many times will lead to the error rising again, which may be seen as a kind of overfitting.

5. Comments

5.1. Innovations

This paper gives us a new way to think about the Lagrangian dual problem. It transforms the coupon allocation problem with a limited budget into a Lagrangian problem. As there are no ways to optimize the objective directly, it resorts to reinforcement learning. In particular, due to the fact that bad plans may cause giant financial loss, we can only apply the offline learning method. This paper proposes several methods to improve the performance of offline learning on the task of coupon allocation. The main innovations of this paper are as below.

- λ-Generalization. It enables us to learn policies for different λ values, which avoids re-learning policies when we change λ or the budget. This can significantly reduce the computation overhead. As we often need to adjust our budget according to the market environment, this is a great contribution of this paper.

- Offline policy learning method R-BCQ. This is a combination of two popular offline RL methods, BCQ and REM. In this way, we can enjoy both benefits of them. Thus, it not only can address the problem of mismatch problem between the state-action pairs in the learned policy and the training dataset but also can avoid the Q-value estimation bias problem, which increases the robustness and generalization performance of the model.

- Policy evaluation method REME. This is a model-free evaluation method that can evaluate the cost and expected utility of policy. Combined with the λ-generalization method, the policy evaluation network is able to evaluate the policy π with different values of λ.

The experiment results presented in this paper are also quite clear and organized. The author carries out the experiments on both the simulation environment and then the Taobao Deals platform, which ensures the security of the experiment and avoids unaffordable costs on real-world testing.

The experiment includes comparisons on the performance with three evaluation metrics both considering cost and logging in rate, which evidently shows the strength of the innovative BCORLE(λ) algorithm and makes it more applicable in different circumstances. Whether pursuing a lower cost or higher retention rate, BCORLE(λ) seems to be a good choice for online coupon allocation.

Though reinforcement learning is often criticized for great time consumption, the experiment shows that with λ-Generalization method, the algorithm can give a relatively fast response. Moreover, due to the limited state dimension and number of actions, the algorithm doesn't depend on an unsatisfying amount of data, which overcomes one general weakness of reinforcement learning.

5.2. Code Implementation

Compared to the fascinating contributions in the theoretical part, the code implementation of this paper is somewhat disappointing. Besides the dependency conflicts and bugs referenced in Sec 4.4, the author's bad code habit also contributes to the code's mess and ambiguity, such as duplicated lengthy code segments, pinyin style naming of variables, using the same name for different variables, and misleading comments, which make the code even harder to understand and debug.

We have summarized several problems and confusions in code implementation in an e-mail and sent it to the author to seek advice and ask for a new version of code, since we believe the author can hardly produce the results on his local machine based on the code provided.

Although we did have a tough time reproducing the experiment results, we have a deep recognition of the importance of a decent coding style. We are still waiting for the reply of the author and hopefully, we can ultimately reproduce the experiment's result with the help of the author.

5.3. Further Improvement

The paper makes use of offline reinforcement learning in coupon allocation circumstances, which is quite innovative. However, compared to the existing method used by Alibaba such as Logistics regression + Linear programming (LR+LP) or Gradient boost decision tree + Linear programming (GBDT+LP), the RL method is less explainable. Both

existing methods predict the distribution of users' retention intent and then choose an action while RL methods just take action based on the learned policy. An example is that If someone tries to attack the coupon mechanism, it could be easier to detect by the prediction+action choosing model as we will find unusual changes in the distribution of users' retention intent. For RL methods, it is hard to detect and interpret the changes in the policy, so we couldn't carry out efficient adversarial plans. Maybe it still relies on more advancement in RL explainability and adversarial model to use RL in such circumstances.

The simulation environment setting is also worth more consideration. The environment assumes all users are the same and just uses uniformly distributed probability to represent their logging in or not. While in real-world settings, users vary in gender, age and these properties will be related to their retention probability. The simulation environment ignores these relevant information. Although in the real-world platform, gender, age, and location are set as one dimension of state each, it increases the risk and uncertainty of applying the algorithm in a real-world platform after testing on a simulation environment as a different definition of the state is used in the experiment on these two different backgrounds. RL methods generally don't have good properties of generalization and some small changes in the environment may lead to great loss. If the simulation environment setting can be more complicated so as to simulate the real world better, the performance on an application may also be significantly improved.

For the experiment results, we think there are still several aspects to discuss. An example is how the evaluation errors of different OPE methods change as the learning steps increase. From the results in paper appendix G, we can conclude that it is not always decreasing. There might be overfitting problems or other factors leading to the error rising when learning steps are large. If more experiment was done with the work, some more meaningful findings would be made.

References

Polixir. Offlinerl. https://github.com/polixir/OfflineRL, 2022.

Zhang, Y., Tang, B., Yang, Q., An, D., Tang, H., Xi, C., LI, X., and Xiong, F. Bcorle(\lambda): An offline reinforcement learning and evaluation framework for coupons allocation in e-commerce market. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 20410–20422. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/ab452534c5ce28c4fbb0e102d4a4fb2e-Paper.pdf.