A decision support method to increase the revenue of ad publishers in waterfall strategy

A Decision Support Method to Increase the Revenue of Ad Publishers in Waterfall Strategy

Reza Refaei Afshar*, Yingqian Zhang†, Murat Firat‡ and Uzay Kaymak §
School of Industrial Engineering, Eindhoven University of Technology
Eindhoven, The Netherlands
Email: *r.refaei.afshar@tue.nl, †yqzhang@tue.nl, ‡m.firat@tue.nl, §U.Kaymak@tue.nl

***Abstract*—Online advertising is one of the most important sources of income for many online publishers. The process is as easy as placing slots in the website and selling those slots in real time bidding auctions. Since websites load in few milliseconds, the bidding and selling process should not take too much time. Sellers or publishers of advertisements aim to maximize the revenue obtained through online advertising. In this paper, we propose a method to select the most profitable ad network for each ad request that is built upon our previous work [1]. The proposed method consists of two parts: a prediction model and a reinforcement learning modeling. We test two strategies of selecting ad network orderings. The first strategy uses the developed prediction model to greedily choose the network with the highest expected revenue. The second strategy is a two-step approach, where a reinforcement learning method is used to improve the revenue estimation of the prediction model. Using real AD auction data, we show that the ad network ordering obtained from the second strategy returns much higher revenue than the first strategy.**

***Index Terms*—Reinforcement Learning, Real time bidding, ad networks, Predictive model**

I. Introduction

Today, most companies operating online rely on advertising and a remarkable amount of their income is obtained through online advertising. In the website of these companies, there are some blocks called *ad slots* for rendering advertisements. While opening the website by a user, the ad slots are proposed to auction platforms to find advertisements within few milliseconds length of time. Real time bidding (RTB) is the most common method for filling the ad slots. Based on this method, Supply Side Platforms (SSP) on behalf of publishers and Demand Side Platform (DSP) on behalf of advertisers interact through ad networks. Ad networks are responsible to run auctions to sell publishers' ad slots to advertisers. In other words, SSPs manage ad requests and send them to ad networks to find appropriate advertisements through auction. The entire process is done in a few milliseconds which is the time a browser loads a website.

Waterfall strategy and Header Bidding are the two common ways that an SSP participates in the auction. In Header bidding, the ad requests are sent to several ad networks simultaneously [2]. By contrast, in waterfall strategy, ad requests are sent to ad networks sequentially and the process continues until finding an advertisement or reaching timeout [3]. Successful ad requests fill the ad slots. If an ad request is unsuccessful, SSP initializes and sends another ad request. In this paper, we focus on the waterfall strategy.

In many RTB systems that work based on waterfall strategy, requests are sent to ad networks in a predefined order. In other words, the first ad request always is sent to ad network 1, the second to ad network 2 and so on. This is not an efficient way because the first ad network is not always the best choice depending on the combination of user, website content and so on. Since ad slots should be filled in few milliseconds, SSPs aim to reduce the number of unsuccessful ad requests. Therefore, reducing the time of finding advertisements is an important issue for SSPs. Furthermore, online advertising is one of the main sources of income for companies and earning the highest possible amount of money due to selling ad slot is important in maximizing the revenue of a company. Therefore, selecting ad networks that increase the revenue and provide advertisements in shortest time possible is a main objective for SSPs [4].

We present a method for ad network ordering that provides the advertisements in the shortest time and with the highest revenue. Our method uses historical RTB data and consists of two steps. In the first step, a prediction model predicts a revenue value for all possible ad request-ad network pairs. In this paper, unlike our recent work [1], we do not start from RL and use prediction model as approximation. Instead, we can state that it is natural to use machine learning methods to build a prediction model in order to estimate whether ad requests will be successful if a specific ad network is used. Then, based on such estimations, we have two ways to derive the ad network ordering: one is to use prediction models straightforwardly, and the other is to use RL. Therefore, in this paper, our goal is to investigate the "good use" of the prediction model for our particular optimization problem. On one hand, the prediction model could be used as an ordering strategy and the ordering is based on the success probabilities [1]. On the other hand, the prediction model can provide initial revenue for RL step to improve the revenue estimation. This paper focuses on the suitable usage of the prediction model and we show that using the prediction model followed by the RL step provides the increased amount of revenue in the lowest number of ad requests.

The problem is a sequential decision making problem and at each time step a certain ad network is selected to send the ad requests and receive rewards. Therefore, reinforcement learn-

ing (RL) is a proper modeling for this problem. Ad requests and ad networks convert to states and actions respectively. The initial state-action values obtained in the first step are used in the learning part of the second step. We show that the reinforcement leaning step increases the expected revenue drastically in comparison with a predefined ordering and an ordering based on the prediction model.

The paper is organized as follows. Section II reviews some related previous works. Section III presents the method for ad networks ordering. Section IV discusses the results of using our proposed method as an ordering strategy on a RTB dataset. Finally, Section V concludes this paper.

II. Literature Review

Many studies focus on real time bidding and develop a framework to help publishers manage ad slots and increase their revenue. In case of waterfall strategy, SSPs on behalf of publishers involve in some decision making tasks. Adjusting the floor price and ordering of ad networks are two important problems [5].

There are different methods developed to set the floor price (reserve price). Floor price is the minimum price that a publisher expects to earn through selling impressions [6]. In [7], a method for adjusting optimal prices for high value inventories is developed. Based on this method, first high value inventories are predicted using a set of classifiers. The authors used the idea in [8] to cascade the classifiers and reduce the false positive rate of the prediction. Then, another cluster of classifiers is developed to predict the separation between top and second bid for high value inventories. Using this information, the floor price is set based on the two highest bids. In [9], an equation is defined in which the floor price is equal to inner product of two vectors. One of them is feature values and the other is weights of the features. This paper uses an approximated gradient vector to determine a good floor price based on historical data. A limitation of this work is that a single floor price is found for all auctions which is too restrictive. In [10], the floor price is set in case of minimizing regret in second price auction. The paper presents an online algorithm that optimizes the floor price based on the actual revenue. This method is general and is not specific to real time auctions. In [11]–[13], the authors propose a method which is useful in non-stationary environments to make better decisions for reserve prices through considering the gap between highest bid and second bid. There are some other studies that present methods for adjusting the floor price such as [14] which models the real time bidding environment as a dynamic game.

Reinforcement learning is a popular method in the context of real time bidding. However, almost all of previous research model the environment from bidder side. In [15] a reinforcement learning method for helping bidders or advertisers to find the best bidding strategy is developed. In [16] and [17], the contribution is assisting advertisers to follow optimal action selection policy. In our paper, we focus on the environment from seller side and the reinforcement learning modeling of RTB environment is from the publisher's point of view.

The ad network ordering problem has gained less attention in recent years in comparison to dynamic pricing. Nonetheless, it is an important issue and has high impact on the seller's revenue. Sometimes, the contracts between publishers and advertisers determine part of ad allocations and the other ad slots are filled through real time auctions [2]. In [18], a method is developed to decide which ad slots should be sold in real time auctions and which should be filled to satisfy contracts. Based on [19] the revenue is the most important factor when a publisher wants to select one ad network. However, the best ad network does not provide the maximum revenue for some ad requests. Therefore, a dynamic ordering is necessary. In [1] we proposed a reinforcement learning method to derive optimal ordering of ad networks. In this paper, we extend this approach and explore the significance of each part of the method. We present a two step ordering method and we show that the importance of the method is in the reinforcement learning part.

III. Methodology

In this section, the ad network ordering method is presented. This method consists of two steps. The first step is a prediction model to find a success probability for each ad network, and the second step includes two methods for ordering.

A. Prediction Model

As a black box, the first step module receives an ad request and returns the expected value of the revenue of that ad request when it is sent to a certain ad network. The expected revenue is defined by multiplication of floor price and the success probability of the ad request which is obtained from the prediction model. The binary target value of the prediction model is called *event state* which indicates whether an ad request is successful (value 1) or not (value 0) and is explained in Table I. For simplicity, in the rest of this paper class 0 refers to the ad requests that their event state is 0. Similarly, class 1 contains ad requests with event state equal to 1. The prediction model is trained with historical data as input. The output of this step could be used either as an output for the second step or as a standalone ordering strategy. As it is mentioned earlier, there are not enough data to learn the revenue obtained from each ad network for each ad request. Therefore, the RL part itself cannot be used as an ordering method.

The historical data samples are obtained from a RTB system and contains information about ad requests. Each ad request has set of features that are listed in table I.

Before training the prediction model with the dataset, some processing and cleaning on the data was necessary to obtain sufficient data quality. Usually, RTB datasets do not have any information about sequences of ad requests. Based on waterfall strategy and the features that is shown in Table I, an ad request is initialized with request order 1 and is sent to the first ad network. If the response indicates failure, another ad request with request order 2 and lower floor price is sent to the next ad network. This sequence ends with or without an impression. Those sequences that the event state value of their last ad request is 0 are called incomplete sequences. Timeout might

TABLE I
INFORMATION IN EACH AD REQUEST.

Field name	Definition	Type
Event state	The result of attempt: 0: fail, 1: success	Binary
Timestamp	time of ad request (hour of a day)	Numerical
Opportunity order	shows how many times a user has entered our system	Numerical
Country code	A code specify country of the user visiting publisher's website.	Nominal
Ad tag id	A unique string corresponds to an advertisement slot	Nominal
Ad network id	Id of each ad network (Ad exchange, AdSense, AOL, ...)	Nominal
Referrer URL	URL of the server that shows the ad	Nominal
Referrer domain	Domain of the server that shows the ad	Nominal
Page URL	URL of the webpage containing the ad slot	Nominal
Page domain	Domain of publisher's website	Nominal
Device name	Name of user device	Nominal
OS name	User's operating system	Nominal
Browser name	User's browser	Nominal
Floor price	The amount of floor price (reserve price)	Numerical
Request order	Order of current attempt in a sequence of attempts.	Numerical

be a reason that make a sequence incomplete. These sequences would not help in predicting the success probability because we do not know the exact reason and the distribution. In other words, it is possible to find two ad requests with the same features except in the time that one of them is successful and the other is unsuccessful. Therefore, the correct label for a train sample may not be deterministic and is different for two identical data samples. To solve this problem, all incomplete sequences are removed from the dataset.

The other issue about the dataset is the number of successful and unsuccessful ad requests. Event state is the target value and for each sequence of ad requsts there is at most one impression. Therefore, the number of impressions (class 1 ad requests) is lower than class 0 samples and we need to balance the dataset. Oversampling extends the dataset and generate new data samples for the smaller class. In this problem, each sequence has between one to eight ad requests. At most one ad request in each sequence is in class 1. Generating new ad requests to have equal number of samples in class 0 and class 1 increase the time of training drastically and make the process not applicable in reasonable time. Hence, we use under-sampling [20]. Because the class 1 samples are more important and have more influence on the revenue, randomly selecting class 0 samples do not have serious effect on the classifier output. Therefore, random under-sampling is chosen to balance the dataset.

Each ad request is represented by a feature vector and these feature vectors are used as inputs to the prediction model. The components of feature vectors are shown in Table I. Event state is the target value and the rest construct the feature vector. In order to use these feature values for prediction model, nominal features should be converted to numerical. One-Hot Encoding is used for this conversion [21]. However, features like URL contain many distinct values and One-Hot Encoder assigns a column for each value. Therefore, the number of columns of the feature vector increase drastically. To solve this problem, a certain number of values with high frequency is selected to generate columns and the other values are updated to a single name, e.g. *low frequencies*.

Many classification method like Support Vector Machine, Bayesian classifier, random forest and decision tree are tested during our preliminary work and random forest classifier works best in case of performance measures. Hence, this classification method is selected to predict the success probability of each ad request. The prediction model provides a success probability which is the probability of event state equal to one for each ad network. The multiplication of this success probability and the floor price provides an estimated value of revenue when a certain ad network is selected to send the ad request. Equation (1) shows the expected revenue obtained from the first step. Fig. 1 illustrates the first step of the proposed method.

$$E[\underline{R}(x_i, a(x_i))] = P(event\_state(x_i) = 1 | x_i, a(x_i)) \times floor\_price(x_i) \quad (1)$$

Where $E[\underline{R}(x_i, a(x_i))]$ is the expected revenue of ad request x_i when it is sent to ad network $a(x_i)$, and $floor\_price$ is a feature of the ad request. $P(event\_state = 1 | x_i, a(x_i))$ is the probability of event state being one for a given ad slot and floor price value and obtained from the prediction model.

B. Prediction model as an ordering method

In the second step, two ordering strategies are possible. The first one is using the output of the prediction model for

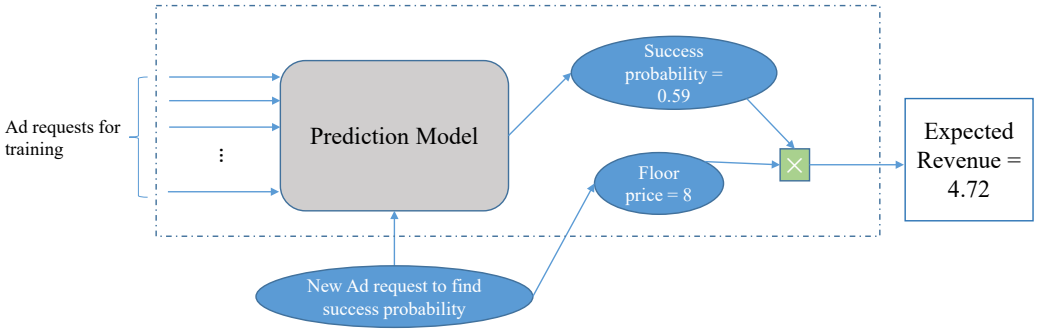

Fig. 1. An Example of the first step of the proposed method. Ad requests are used to train the classifier and then success probabilities are obtained for each new ad request. Multiplication of success probability and floor price yields the expected revenue.

ordering ad networks. In other words, the first ordering strategy is to use (1) and sort the ad networks based on these values.

An interesting question is what the significance of each step is. It is clear that because the historical data do not provide enough state-action pairs to learn all values, the prediction model is necessary for finding unobserved values. However, if the prediction model can estimate all state-action values could it be used as the ordering strategy? To answer this question, we use the multiplication of success probability and floor price as state-action values and follow a greedy policy to select ad networks. This method will provide advertisements in the lowest number of requests. However, the revenue obtained in this way is not optimal because the ordering does not care about next states and their revenue. We evaluate this method in Section IV.

C. Reinforcement Learning

The second approach is to modify the revenue estimation to obtain more revenue. For this purpose, a reinforcement learning modeling is developed. The problem is sequential decision making. At each time step, the agent (decision maker or SSP) selects an ad network and receives the reward. The next state depends on whether this ad request is successful or not. In the rest of this section, states, actions, reward function and state transition are defined.

1) States: In the RL modeling of real time bidding, states are pertinent to ad requests. One approach is to consider all features of ad requests as states. As it is impossible to interact with the real RTB environment at the moment we use historical data to learn state-action values. Since the historical data are obtained from a fixed and predefined ordering of ad networks, the observed state-action space is sparse and a high percent of state-action values are the expected revenue of the first step without any modification. Therefore, using ad requests as states results in sparse state-action values because there is only one observed ad network for each ad request in the historical data. Each state should be a subset of features to overcome the sparsity. Among different subsets of features, the combination of floor price, request order and ad tag id well represents the nature of states because ad tag id is the ad slot identifier in each state and in each state transition all features are fixed except request order and floor price.

The sparsity problem is not solved yet because the predefined ordering of ad networks results in one observed ad network in each combination of request order and floor price. Hence, it is necessary to define some ranges in the values of request order and floor price instead of exact values. In this definition, states corresponds to sets of values of selected features. A trade-off emerges from this definition. On one hand, if the number of ranges are too high the observed state-action space is sparse. On the other hand, if there are few ranges the states are too general and the decision maker do the same for a large number of ad requests. Therefore, the ordering is similar to the predefined method. We test different number of categories and realize that dividing values into two ranges keep balance between sparsity and generalization. We define two thresholds on the values of request order and the floor price to divide each of them into two categories. These thresholds are called t_{ro} and t_{fp} respectively for request order and floor price. Therefore, each state consists of range of floor price, range of request order, and the value of ad tag id. Equation (2) defines the state of a given data instance. Note that the space still has unobserved state-action pairs which are initialized by the prediction model of the first step.

$$\begin{aligned} s(x_i) = (&Ad\_tag\_id(x_i), floor\_price\_range(x_i), \\ &request\_order\_range(x_i)), \\ &x_i \in D \ : \ i^{th} ad\ request, \end{aligned} \tag{2}$$

$$floor\_price\_range(x_i) = \begin{cases} 0 & if\ floor\_price(x_i) \in [0, t_{fp}) \\ 1 & if\ floor\_price(x_i) \in [t_{fp}, m_f] \end{cases} \tag{3}$$

$$request\_order\_range(x_i) = \begin{cases} 0 & if\ request\_order(x_i) \in [0, t_{ro}) \\ 1 & if\ request\_order(x_i) \in [t_{ro}, m_r] \end{cases} \quad (4)$$

$$m_r = \max_{x_i \in D}(request\_order(x_i)) \quad (5)$$

$$m_f = \max_{x_i \in D}(floor\_price(x_i)) \quad (6)$$

where m_r and m_f are the maximum values of the request order and the floor price that are obtained from the historical data and D is the set of all ad requests.

2) Actions: In our modeling actions are ad networks. In each state a certain number of ad networks are available and choosing each one results in either a state transition or termination of current episode. Assume that there are N possible actions in each state. This number is fixed for all ad requests because available ad networks are known and fixed. Equation (7) shows the actions in the reinforcement learning modeling of this paper.

Equation (7) is the definition of the possible actions in each state. Since some combinations of states and actions do not exist in the historical data, the actions set of each state is a subset of all actions.

$$a(x_i) \in \{a_1, a_2, ..., a_N\}, \quad (7)$$

where $a_1, a_2, ..., a_N$ are ad networks.

3) Reward Function: The reward value should take two objectives into account. Action selection policy should select actions which provide highest revenue in shortest time. Floor price is the lowest bound for the revenue. Therefore, reward should be related to the floor price. Besides, in order to prevent unsuccessful attempts a penalty should be assigned to ad requests of class 0. Therefore, if the ad request is successful the reward is the value of floor price and if it is not successful the reward is -1. Equation (8) shows this definition.

$$Reward^{x_i}(s,a) = \begin{cases} -1 & if\ event\_state(x_i) = 0 \\ floor\_price(x_i) & if\ event\_state(x_i) = 1 \end{cases} \quad x_i \in D(s,a), \quad (8)$$

$$D(s,a) = \{x_i \in D | (s(x_i), a(x_i)) = (s,a)\}, \quad (9)$$

where $floor\_price$ and $event\_state$ are features of ad request x_i. $D(s,a)$ is the set of all ad requests whose corresponding state and action are (s,a).

4) State Transition: Based on the definition of the states, the state transitions are updating of the values of floor price and request order. When an ad request is successful there is no next state and the current state is a terminal. Otherwise, the next state is obtained by increasing the request order and decreasing the floor price. Request order is an integer value that shows the ordering of an ad request in a sequence. Therefore, the request order increases by one in a state transition. Based on the RTB historical data, the floor price decreases when the request order increases. In sum, a next state is a new ad request followed by an unsuccessful ad request which its request order is higher and its floor price is lower than current state's.

5) Learning State-Action values: The model of environment is unknown and also it is not possible to interact with the environment. Hence, sequences of ad requests to fill a certain ad slot are considered as episodes. We use Monte Carlo method to learn state-action values by averaging of the returns. We modify the typical Monte Carlo method to take initial state-action values obtained from the first state into account. In the Monte Carlo method [22], arbitrary values are assigned as initial state-action values. In our method, the initial values are obtained from the first step, and their weights in the averaging part are the number of each state-action pairs in the training data. Equation (10) shows the updating rule of the Monte Carlo algorithm.

$$Q(s,a) = (\sum_{j=1}^{n_1(s,a)} Reward_j^{x_i}(s,a) + E[\underline{R}(x_i,a)] \times n_2(s,a)) / (n_1(s,a) + n_2(s,a)), s.t.\ \ x_i \in D(s,a), \quad (10)$$

where n_1 is the number of (s,a) observed so far and n_2 is the number of (s,a) used for training the prediction model.

The proposed method sorts ad network for each coming ad request in order to have maximum revenue in the lowest number of requests. The ordering is based on the state-action values. Figure 2 presents an overview of our proposed method.

IV. Experiments and Results

In this section, the result of using our proposed method to find the most profitable ordering of ad networks are discussed. The first part of this section deals with the performance of the prediction model and the second part presents the comparison between the expected revenue obtained by our method, real revenue and the revenue acquired by using the prediction model as a decision method. The dataset is a set of RTB ad requests in a period of one week from Monday to Sunday (20^{th} to 26^{th} of November 2017). Each day contains between 300000 to 400000 ad requests. In order to keep the chronological ordering of ad requests, we use the ad requests of Monday for training and testing of the prediction model. Then, the ad requests of Tuesday to Saturday are used for learning state-action values of the second step. Finally, the ad requests of Sunday serves as validation data to evaluate the

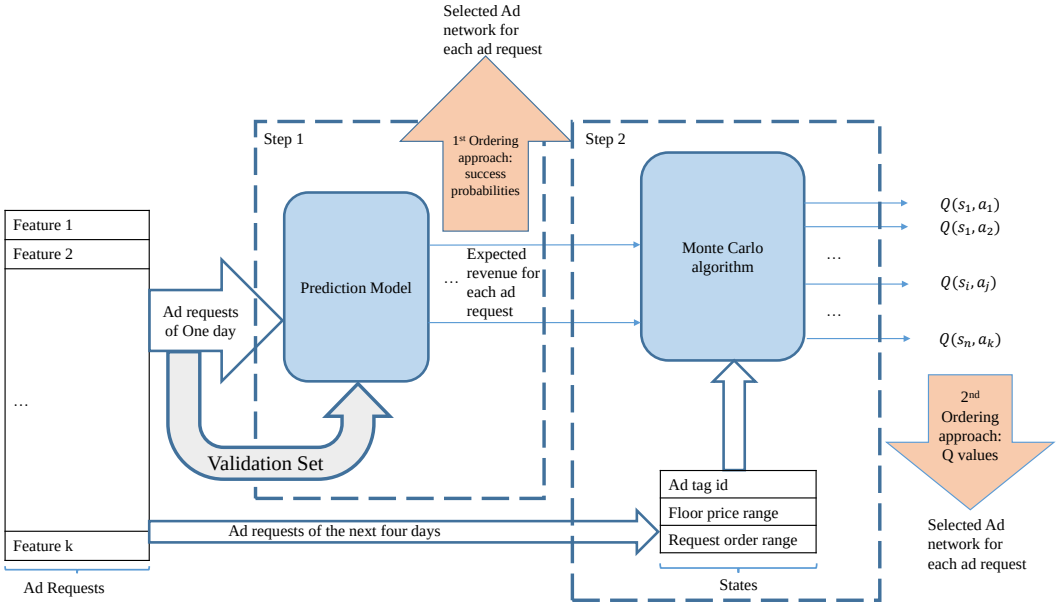

Fig. 2. Two ad network ordering methods. First one is to the use success probability multiplied by the floor price and the second is a two-steps method. Ad requests of the first day of the week are used for training the prediction mode. Then, the second step uses the ad requests of the next five days to learn state-action values. Selecting the ad networks greedily based on state-action values provides the maximum revenue.

expected revenue obtained by different methods of ad network ordering.

A. Prediction Model evaluation

The prediction model predicts the success probability of ad requests to provide initial values for the reinforcement learning part. As it is mentioned earlier, the ad requests of 20^{th} of November 2017 were used for this step. Nonetheless, the ad requests of all days are used for training and testing separately to show that the method does not perform well on just one day and is not biased. In order to keep the chronological ordering of ad requests, we used a cross validation method on the sequences of ad request that are sent to fill a certain ad slot. Sequences are independent and splitting the dataset to training and validation sets based on the sequences would not violate the ordering of ad requests. In other words, we do not use any related ad requests of future to predict current ad requests.

Table II shows the performance measures of predicting the success probabilities of ad requests. As it is illustrated in this table, the prediction model predicts whether an ad network is successful to find an advertisement for an ad slot or not, with a high precision and recall. Fig. 3 shows the ROC curve of the prediction model for predicting the event state of ad requests [23]. The curves are close to each other and it shows that the prediction model is not biased to one specific date. The higher curves correspond to $25th$ and $26th$ of November and these days have more ad requests.

In sum, the prediction model is trained on the ad requests of Monday and used as initial values for the second step which is trained on the ad requests of $21th$ and $25th$.

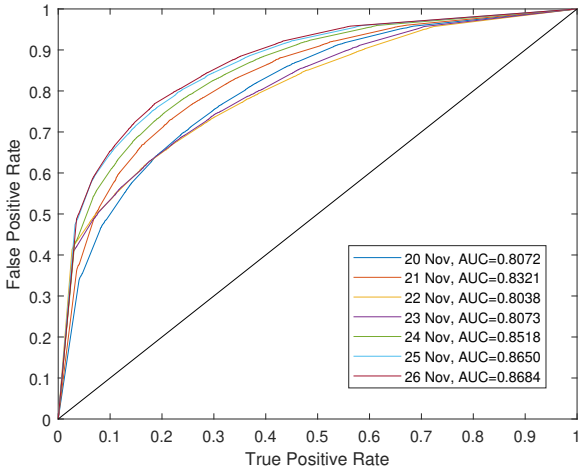

Fig. 3. ROC curve for ad requests of seven days.

B. Ordering Methods evaluation

We extend the recent work [1] and two ordering methods have been defined in sections III-B and III-C. In this section, the results of using these methods to derive the best ordering of ad networks are discussed.

The ad requests of five consecutive days are used to learn state-action values. In the dataset, five different ad networks are observed. Therefore, in each state there are five possible actions. Each sequence of ad requests that starts with request order equal to one and ends with or without an impression serves as an episode in the Monte Carlo method. For each ad request of an episode, first it is converted to a state based

TABLE II
PERFORMANCE MEASURES FOR THE PREDICTION MODEL

Event state = 1	Nov 20	Nov 21	Nov 22	Nov 23	Nov 24	Nov 25	Nov 26
Precision	0.7388	0.7668	0.7468	0.7382	0.7816	0.7991	0.8012
Recall	0.7165	0.7291	0.6781	0.6967	0.7486	0.7598	0.7662
F1	0.7275	0.7475	0.7108	0.7168	0.7647	0.7790	0.7833
Accuracy	0.7314	0.7549	0.7240	0.7261	0.7700	0.7844	0.7879
Kappa	0.4628	0.5098	0.4480	0.4521	0.5400	0.5689	0.5758

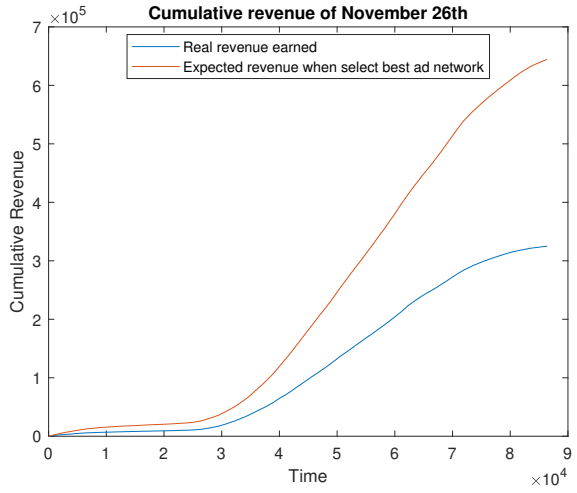

Fig. 4. Expected revenue vs. real revenue.

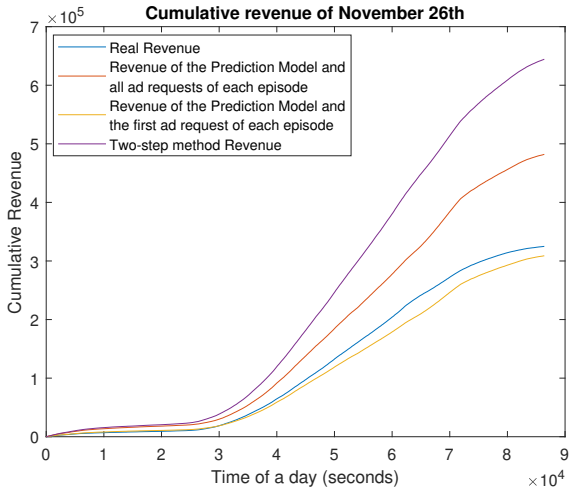

Fig. 5. Comparison of four ad network selection policies.

on (2) and then for this state and its observed action, $Q(s, a)$ is updated based on (10). By sorting the action values for each state, a greedy method selects the action with the highest state-action value. For each sequence, the first ad request is considered to estimate the revenue because the value of each state is the average of returns obtained from that state. The value of the first ad request of each sequence shows the expected revenue of that sequence.

In order to have a reference revenue, the real revenue of ad requests is obtained through multiplication of event state and floor price. If event state is one, the ad request is successful and the floor price is the lower bound of the revenue. Fig. 4 illustrates cumulative revenue that is actually obtained in the data and the cumulative expected revenue of the proposed method and is discussed in our recent paper [1]. The evaluation is performed on the ad requests of 26^{th} of November. We can infer from this figure that using our method will increase the revenue drastically.

As it is mentioned in section III-B, we use the prediction model as an ordering strategy to find out the significance of the first step. First, we consider the first ad request of each sequence and select the ad network with the maximum success probability as the selected action.

The second approach to evaluate the method is to find the expected revenue of all ad requests of a sequence. In this approach the policy is still greedy. Assume that P_i is the maximum success probability when request order is i. First, the ad network corresponding to P_1 is selected. This ad request is unsuccessful with the probability $1 - P_1$ and in the next attempt the ad network corresponding to P_2 will be selected. In this way the expected values of revenue are obtained. Equation (11) shows this expected value.

$$\begin{aligned} E(e) = {} & P_1 \times floor\_price(e_1) + \\ & (1 - P_1) \times P_2 \times floor\_price(e_2) + \\ & \ldots + (\prod_{i=1}^{t-1}(1 - P_i)) \times P_t \times floor\_price(e_t), \end{aligned} \quad (11)$$

where e is an episode or sequence of ad networks, e_i is the i^{th} ad request of episode e and t is the maximum request order or the length of episode e. Fig. 5 illustrates the revenue of all four methods. This figure shows that the two-step method provides a much higher revenue than the real revenue and also than the revenue of using prediction model as a selection policy. Furthermore, based on this figure, ad network ordering strategy derived by the prediction model when only the first ad request of each sequence is considered can provide an estimation of real revenue obtained from a predefined ordering.

V. CONCLUSION

In this paper, we proposed a method to select ad networks in the waterfall strategy. The method consists of two steps. As there are not sufficient data samples to learn the value of sending requests to ad networks, we develop a prediction model to overcome the sparsity of the dataset. The prediction model is a necessary step to deal with unobserved state-action values in the dataset. However, we showed that this approach is not enough for ordering the ad networks and we need the reinforcement learning part to improve the estimation and increase the revenue. The values obtained in the first step serve as initial state-action values for the second step which is a reinforcement learning modeling of the real time bidding. We showed that the power of our method lies in the reinforcement learning part and selecting ad networks based on a greedy policy on the state-action values provides the maximum revenue and appropriate advertisements in the lowest number of attempts.

Monte Carlo method was used because we do not have any information about the model of the environment. The state-action space was sparse and to deal with the sparsity we used the prediction model to provide initial state-action values. In the future, we will use other methods like function approximation to overcome the sparsity problem [22], [24] [25]. Furthermore, we aim to use our method in the real environment to obtain the real revenue.

In addition, We will investigate how to model different players as agents, like those done in B2B and B2C online auctions [26], [27], and recently in ad auctions [28]. We will consider DSPs, which act on behalf of advertisers, as an agent, and model the environment as a multi-agent system. We will study learning algorithms that optimize publishers' strategy by taking into account other players in the system.

ACKNOWLEDGMENT

This work was supported by NWO (project 628.010.001) and EU EUROSTARS (Project E! 11582).

REFERENCES

[1] R. R. Afshar, Y. Zhang, M. Firat, and U. Kaymak, "A reinforcement learning method to select ad networks in waterfall strategy," in *11th International Conference on Agents and Artificial Intelligence (ICAART)*. IEEE, 2019.

[2] A. Sayedi, "Real-time bidding in online display advertising," *Marketing Science*, vol. 37, no. 4, pp. 553–568, 2018.

[3] J. Wang, W. Zhang, S. Yuan *et al.*, "Display advertising with real-time bidding (rtb) and behavioural targeting," *Foundations and Trends® in Information Retrieval*, vol. 11, no. 4-5, pp. 297–435, 2017.

[4] S. Muthukrishnan, "Ad exchanges: Research issues," in *International Workshop on Internet and Network Economics*. Springer, 2009, pp. 1–12.

[5] O. Busch, "The programmatic advertising principle," in *Programmatic Advertising*. Springer, 2016, pp. 3–15.

[6] W. Zhang, S. Yuan, and J. Wang, "Optimal real-time bidding for display advertising," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1077–1086.

[7] Z. Xie, K.-C. Lee, and L. Wang, "Optimal reserve price for online ads trading based on inventory identification," in *Proceedings of the ADKDD'17*. ACM, 2017, p. 6.

[8] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[9] D. Austin, S. Seljan, J. Monello, and S. Tzeng, "Reserve price optimization at scale," in *2016 IEEE 3rd International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016, pp. 528–536.

[10] N. Cesa-Bianchi, C. Gentile, and Y. Mansour, "Regret minimization for reserve prices in second-price auctions," *IEEE Transactions on Information Theory*, vol. 61, no. 1, pp. 549–564, 2015.

[11] J. Rhuggenaath, A. Akcay, Y. Zhang, and U. Kaymak, "Optimizing reserve prices for publishers in online ad auctions," in *2019 IEEE Conference on Computational Intelligence for Financial Engineering and Economics (CIFEr)*, 2019.

[12] ——, "A pso-based algorithm for reserve price optimization in online ad auctions," in *2019 IEEE Congress on Evolutionary Computation (IEEE CEC)*, 2019.

[13] ——, "Fuzzy logic based pricing combined with adaptive search for reserve price optimization in online ad auctions," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2019.

[14] S. Yuan, J. Wang, B. Chen, P. Mason, and S. Seljan, "An empirical study of reserve price optimisation in real-time bidding," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1897–1906.

[15] D. Vengerov, "A gradient-based reinforcement learning approach to dynamic pricing in partially-observable environments," 2007.

[16] H. Cai, K. Ren, W. Zhang, K. Malialis, J. Wang, Y. Yu, and D. Guo, "Real-time bidding by reinforcement learning in display advertising," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017, pp. 661–670.

[17] V. Nanduri and T. K. Das, "A reinforcement learning model to assess market power under auction-based energy pricing," *IEEE transactions on Power Systems*, vol. 22, no. 1, pp. 85–95, 2007.

[18] J. Rhuggenaath, Y. Zhang, A. Akcay, and U. Kaymak, "Optimal display-ad allocation with guaranteed contracts and supply side platforms," 2018, working paper.

[19] A. Ghosh, P. McAfee, K. Papineni, and S. Vassilvitskii, "Bidding for representative allocations for display advertising," in *International Workshop on Internet and Network Economics*. Springer, 2009, pp. 208–219.

[20] N. Japkowicz *et al.*, "Learning from imbalanced data sets: a comparison of various strategies," in *AAAI workshop on learning from imbalanced data sets*, vol. 68. Menlo Park, CA, 2000, pp. 10–15.

[21] D. Harris and S. Harris, *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[22] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[23] T. Fawcett, "Roc graphs: Notes and practical considerations for researchers," *Machine learning*, vol. 31, no. 1, pp. 1–38, 2004.

[24] E. Peer, V. Menkovski, Y. Zhang, and W.-J. Lee, "Shunting trains with deep reinforcement learning," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2018, pp. 3063–3068.

[25] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[26] S. Verwer, Y. Zhang, and Q. C. Ye, "Auction optimization using regression trees and linear models as integer programs," *Artificial Intelligence*, vol. 244, pp. 368–395, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2015.05.004

[27] S. Verwer and Y. Zhang, "Revenue prediction in budget-constrained sequential auctions with complementarities," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 1399–1400.

[28] J. Jin, C. Song, H. Li, K. Gai, J. Wang, and W. Zhang, "Real-time bidding with multi-agent reinforcement learning in display advertising," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 2193–2201.