From Kernel to Permutation Estimator: Applying SHAP to Explain Reinforcement Learning in Online Advertising

Jingxuan Chen

Master of Science
School of Informatics
University of Edinburgh
2023

Abstract

Reinforcement learning (RL) is an effective technique to optimise a bidding strategy in online advertising auctions. However, RL models based on neural networks are known as "black boxes", which hide the decision-making process away from human users. Such a lack of transparency prevents humans from trusting and using RL models effectively. To mitigate this issue, we employ an eXplainable Artificial Intelligence approach - SHapley Additive exPlanations (SHAP), to explain how RL bidders make decisions. Given the computational complexity of precise SHAP value calculations, we focus on two SHAP value estimators: Kernel and Permutation. Through a series of experiments, we delve into the interplay between RL bidding policies and SHAP values. To validate our explanations without the requirement of real data, we propose a simple yet intuitive validation method through synthetic simulation data in an auction simulation environment - AuctionGym. Our findings suggest that SHAP values derived from the Permutation estimator present a feasible approach for uncovering the feature importance underlying bidding policies.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jingxuan Chen*)

Acknowledgements

Firstly, I would like to express my gratitude to my supervisors, Ben Allison, Doudou Tang and Robert Hu, for their constant help and guidance throughout this project. Thank Prof Iain Murray, for coordinating this amazing project with Amazon. Thank my colleagues, Keith, Miltiadis and Radhikesh, for insightful discussions. Thank Christoph Molnar, for allowing me to be a beta reader of his inspirational SHAP book. Thank ChatGPT, for saying "Good luck with your dissertations!". Thank Kai, for all the hugs and assistance in this challenging but sweet year.

Secondly, I want to thank my family, friends and lovely cats - Capu and Cino, for their unconditional love, virtually or physically.

Lastly, thank my favourite band, Mayday, for always making my day.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project focuses on the application of an eXplainable Reinforcement Learning (XRL) technique in the realm of online advertising. In this chapter, we provide an overview of the project and outline the structure of this dissertation.

1.1 Motivation

Reinforcement learning (RL) is a machine learning paradigm where an agent learns a policy - which action to take in a particular state, in order to yield the most rewards [18]. RL has applications in various fields, including robotics, autonomous vehicles, and recommendation systems. [4] argues that the interactive and reactive characteristics of the bidding problem also align with an RL formulation. Each bidder participating in such an auction can be recognised as an RL agent, whose goal is to optimise its utility (i.e., the utility is the reward). To be more specific, for an agent (or a bidder model), its state is an advertisement impression opportunity and its action is a bid. The main difficulty in the bidding problem is the intractability of precisely computing the expectation of utility. To tackle this issue, [4] states three utility estimators and proposes an auction simulation environment - AuctionGym[1]. AuctionGym enables producing reliable offline validation of utility estimators, without the need for proprietary and sensitive data. Based on the result in AuctionGym, RL is proven to be effective for solving the bidding problem, which might inspire human bidders or domain researchers.

However, there is a barrier between RL and human bidders - intransparency. These RL bidders are neural networks that are known as "black boxes". It is because humans can only know the input and output of the models, but obtain no information about how a

[1] https://github.com/amzn/auction-gym

prediction is made. The absence of intermediate steps and decision logic makes RL models challenging for humans to understand, and thus prevents humans from trusting and using them [13]. In contrast, if explanations are attached, humans will be able to judge models' predictions by comparing explanations with their basic knowledge. This can further bring insights and benefits for humans. Therefore, it is essential and meaningful to explain RL models. As the foundation of XRL, eXplainable Artificial Intelligence (XAI) concentrates on explaining Artificial Intelligence (AI) models' behaviours. A way to categorise XAI methods is according to whether a method is applicable to any AI model (model-agnostic) or is only tailored to a particular model (model-specific). One of the model-agnostic techniques is SHapley Additive exPlanations (SHAP). For each instance to be explained, SHAP generates one value per feature that signifies the contribution of that feature to the model's prediction. The challenge here is that accurately computing SHAP values is impractical due to the exponential complexity. To address this, SHAP value estimators are developed, which can speed up calculations while maintaining a certain level of accuracy.

In this project, we explore how SHAP can shed light on the behaviours of RL bidders, with a specific focus on two SHAP value estimators - Kernel and Permutation. AuctionGym is used to train RL bidders and validate SHAP results. To better understand both bidding policies and SHAP value estimators, we designed and conducted three sets of experiments in varying settings. In addition, the explanation pipeline and all experiments aim to be reproducible, and we hope it can assist other researchers in the future.

1.2 Objectives

This project aims to investigate how RL bidders work in AuctionGym and the difference between SHAP estimators. We propose three research questions:

1. **Learning to bid**: To what extent can SHAP reflect different configurations within AuctionGym?

2. **SHAP**: How do explanations change when adjusting SHAP estimators?

3. **Context feature**: What insights can be derived from SHAP when context features are modified?

1.3 Structure of Dissertation

The structure of this dissertation is outlined below:

- Chapter 1 introduces this project and formulates three research questions for exploration.
- Chapter 2 presents the background and related work of this project, covering RL bidding policies and XRL methods.
- Chapter 3 describes our methodology, containing the utilisation of AuctionGym, SHAP value estimators and our visualisation choices.
- Chapter 4 details the implementation of our three-step pipeline: bidder training, simulation data generation and SHAP.
- Chapter 5 reports our experiment details and results, accompanied by discussions of our findings.
- Chapter 6 concludes this project by addressing research questions, indicating the limitation of our work, and suggesting future research direction.

Chapter 2

Background

In this chapter, we start with the fundamental concepts of RL and a specific RL problem - contextual bandits. Then we show that the bidding problem (learning to bid) can be formulated as offline learning in contextual bandits, and introduce a simulation environment - AuctionGym that enables offline validation for this problem. We next move on to XRL and present the explanation technique we used in this project - SHAP. We also provide a review of previous applications of SHAP.

2.1 Reinforcement Learning

An RL agent learns a policy by interacting with the environment. For a single action, the environment gives the agent numerical feedback indicating rewards or penalties, where penalties can be recognised as negative rewards. The amount of reward over the long run is defined as a value function, which is vital in RL as it guides the agent to discover proper actions so that the agent can optimise its policy[18].

Bandit learning is a simplified framework of RL, which only cares about immediate rewards. One of the bandit learning problems is contextual bandits, whose award following each action depends on its input context. A context can be recognised as a state in the general RL framework and is observed by the agent before deciding an action [17]. [3] studies offline learning in contextual bandits. Here, "offline" means that the learning is based on historical data (consisting of contexts, actions and rewards) and is unable to gather new data. There are two existing approaches to deal with the reward in offline learning: **Direct Method (DM)** and **Inverse Propensity Score (IPS)**. DM approximates the reward function for each action according to the given data, and then uses all these approximations to estimate the expected reward of the full policy. The

limitation of DM is that it might suffer from a high bias when the modelling of reward is not accurate. The other choice, IPS, eliminates such bias of the reward estimation through importance sampling on the logged policies. An importance weight is the ratio of the learnt policy's probability density to that of a logged policy. However, IPS can lead to a high variance, especially when the past policies greatly differ from the current one. In order to mitigate the shortcomings of DM and IPS, [3] develops a novel method: **Doubly Robust (DR)**. DR takes DM as a baseline and leverages IPS to rectify its errors. It has been proven that DR will be unbiased if either DM or IPS is correct. [2] further demonstrates the capability of DR in optimising contextual bandits policy.

2.2 Learning to Bid

[4] introduces online advertising auctions. In a single auction, an ad exchange takes on the role of the auctioneer, who presents an online advertisement impression opportunity to potential advertisers. The opportunity is described by context features (embeddings), while advertisers can only observe a subset of these context features represented as x. Then, advertisers participate in the auction as bidders, with the goal of maximising their own cumulative utility. Each bidder needs to make two decisions:

1. **The allocation problem**: Which advertisement in its catalogue $\mathcal{A}$ will be shown[1]?
2. **The bidding problem**: How much to bid for this impression opportunity?

After collecting bids from all advertisers, the ad exchange will select the advertiser who places the highest bid as the winner of this auction, and inform the winner how much it will be charged as per the auction type. There are two common auction types: **First-Price** and **Second-Price**. The winner will pay as much as it bids in a first-price auction, while in a second-price one, the winner's payment will be the second-highest bid. Finally, only the winner will know a non-zero outcome, by comparing the amount it paid with how much it earns from this impression. All other bidders will obtain zero outcomes since they did not provide any payment and also receive nothing.

As mentioned above, a bidder mainly needs to solve two problems during an auction. Regarding the allocation problem, [4] defines that every advertisement $a \in \mathcal{A}$ is linked to a private valuation $v_a \in \mathbb{R}^{+}$[2]. v_a specifies the advertiser's willingness to pay for a

[1] In [4], there is an additional step that involves selecting a subset of advertisements $\mathcal{A}_x \subseteq \mathcal{A}$, which contains all advertisements that are eligible to be displayed given the current context features x. However, for the sake of simplicity, our project only discusses $\mathcal{A}$ instead.

[2] In this project, we use $\mathbb{R}$ to represent real numbers.

potential conversion event (click-through rate) of a. There is also a binary[3] random[4] variable C denoting whether a conversion event has happened after an advertisement impression. The conversion estimator for the impression of advertisement a_i and the context x can be written as $\widehat{\mathrm{P}}(C|A=a_i;X=x)$[5]. The advertiser now can estimate its expected welfare ω given such impression and context:

$$\widehat{\mathbb{E}}[\omega|A=a_i;X=x] := v_{a_i} \cdot \widehat{\mathrm{P}}(C|A=a_i;X=x). \tag{2.1}$$

[4] further assumes that the advertiser will always choose the advertisement $a^\star$ that can maximise its estimated expected welfare $\widehat{\omega}$ among all advertisements:

$$a^\star = \arg\max_{a_i \in \mathcal{A}} \widehat{\mathbb{E}}[\omega|A=a_i;X=x]. \tag{2.2}$$

For the bidding problem, a bidder's optimal strategy is related to the auction type. [21] shows that bidding truthfully (i.e., reporting the expectation of v_a) is a dominant strategy (the most profitable for both auctioneers and bidders) in second-price auctions, under certain conditions. One condition is that all competitors can access the same information. However, this is not usually satisfied in real-world auctions. This is because bidders might have varying levels of information, such as the item being auctioned and the market environment. Other conditions can also violated in actual auctions. Therefore, second-price auctions can no longer maximise auctioneers' revenue, which stops auctioneers from sticking to second-price auctions (e.g., instead of second-price, auctioneers might choose first-price or a combination of first-price and second-price auctions). In fact, bidders always have no information about the auction type, so they should move away from truthful bidding and find another way to maximise their utility.

2.2.1 Bidding Policy

[4] states that the bidding problem can be framed in an RL formulation, or more precisely, a contextual bandits one. A bidder can be recognised as an RL agent, whose state is an observable context x and action is a bid $b \in \mathbb{R}^+$. After placing a bid, the bidder will be notified whether it wins or not. If it wins, it will get charged a price $p \leq b$, otherwise not. Then it can know its reward, which is the utility according to the auction outcome. Other bidders' behaviours or settings (e.g., the auction type) are unknown environmental factors. In addition, how the bidder determines a bid can be regarded as

[3] In this project, for binary variable, we default $1 \equiv$ True/Yes and $0 \equiv$ False/No.

[4] In this project, we use uppercase letters to distinguish random variables from lowercase values.

[5] Same as [4], we use $\widehat{Q}$ to denote estimated quantities Q in our project.

sampling a value from its policy π, where $\pi(b|a;x)$ denotes $\mathrm{P}(B=b|A=a;X=x;\Pi=\pi)$. The goal of the bidder is to maximise the expectation of its utility U.

[4] factorises U by three notations. The first one W is a binary random variable, showing whether the bidder wins the auction. The second one $V \equiv \omega$ is the welfare that the bidder gains from the auction, and the last one P represents the amount of payment for the auction. With these notations, the bidder's utility can be calculated as:

$$U = W(V - P). \tag{2.3}$$

After each auction, all of W, V and P become observable. When $W = 0$, it means the bidder loses the auction, and thus $V = P = 0$ as well.

[4] further computes the expected U for a bidding policy π by integrating over all contexts x, values v and prices p:

$$\begin{aligned} \mathop{\mathbb{E}}_{b\sim\pi(B|A;X)}[U] = \int \mathrm{P}(W=1|X=x;B=b)(v-p) \\ \mathrm{P}(V=v|A=a;X=x)\mathrm{P}(P=p|X=x;B=b)\mathrm{d}x\mathrm{d}v\mathrm{d}p. \end{aligned} \tag{2.4}$$

[4] makes two assumptions on this integration: (1) $\mathrm{P}(W)$ and $\mathrm{P}(P)$ are conditionally independent of A given X and B; (2) V is conditionally independent of B given A and X. Although these assumptions simplify this integration, it is still intractable, thus the estimation of utility expectation becomes necessary.

2.2.2 Utility Estimators

Since the bidding problem is a contextual bandits problem and the reward is the utility, three reward estimators (DM, IPS and DR) mentioned in Section 2.1 can be applied to estimate the utility. In addition, [4] draws a similar conclusion as [3]: DM might suffer from a high bias and IPS might cause a high variance, while DR can achieve the overall best performance since it combines the advantages of DM (low variance) and IPS (unbiased).

Direct Method (DM): [4] first defines a utility estimator $\widehat{u}$ based on the historical context-advertisement-bid triplet samples:

$$\widehat{u}(x,a,b) \approx \mathbb{E}[U|X=x;A=a;B=b]. \tag{2.5}$$

For a single triplet, the estimation can be derived from Equation (2.3) by splitting W, V and P. Then [4] argues that if utility estimators for each bid in each impression

opportunity are available in historical data $\mathcal{D}$, an expected utility estimator for a policy can be obtained as well:

$$\mathop{\mathbb{E}}_{b\sim\pi(B|A;X)}[U] \approx \widehat{U}_{\mathrm{DM}}(\pi,\mathcal{D}) = \sum_{(x,a,b,u)\in\mathcal{D}} \int \widehat{u}(x,a,b')\pi(b'|a;x)\mathrm{d}b'. \tag{2.6}$$

Inverse Propensity Score (IPS): [4] supposes that instead of explicitly modelling the utility, there is another approach to optimise a bidding policy. That is, directly applying importance sampling on historical policies to maximise the integral in Equation (2.4):

$$\mathop{\mathbb{E}}_{b\sim\pi(B|A;X)}[U] \approx \widehat{U}_{\mathrm{IPS}}(\pi,\mathcal{D}) = \sum_{(x,a,b,u)\in\mathcal{D}} u\frac{\pi(b|a;x)}{\pi_0(b|a;x)}. \tag{2.7}$$

Doubly Robust (DR): [4] proposes the last estimator which takes the complementary advantages of the previous two estimators, by selecting part of the samples to learn a utility estimator and using the remaining to calibrate the learnt policy:

$$\begin{gathered}\mathop{\mathbb{E}}_{b\sim\pi(B|A;X)}[U] \approx \widehat{U}_{\mathrm{DR}}(\pi,\mathcal{D}) = \\ \sum_{(x,a,b,u)\in\mathcal{D}} \Big(\int \widehat{u}(x,a,b')\pi(b'|a;x)\mathrm{d}b' + (u-\widehat{u}(x,a,b))\frac{\pi(b|a;x)}{\pi_0(b|a;x)} \Big).\end{gathered} \tag{2.8}$$

[4] notes that although combining DM and IPS can make DR outperform other models in the bidding problem, it does not guarantee performance improvements in all applications.

2.2.3 AuctionGym

Beyond figuring out how to formulate a reward utility, validating a learnt bidding policy's performance is also a vital but challenging task. This is because purely depending on logged data can not capture how an updated bidding policy will react to new context features, while running online experiments is too expensive and might lead to financial losses as sub-optimal bidding strategies are kept being tried. As a result, [4] proposes an auction simulation environment - AuctionGym, which enables producing reliable offline validation without the need for proprietary and sensitive data.

AuctionGym can simulate online advertising auctions end-to-end, following the same process as described in Section 2.2. An auctioneer will present an impression opportunity via context features, then all bidders jointly make their own allocation and bidding decisions. Finally, the auctioneer will notify all bidders of their individual outcomes. From the view of a single bidder (an RL agent), it can only access the

information related to itself, including its bidding policy and rewards of previous state-action pairs. In contrast, the researchers who operate AuctionGym can control the environment configurations, such as the representation of context features, auction type, number of participating bidders and their utility estimators. Researchers can also observe all data about bidders and auctions that were experienced.

2.3 Explainable Reinforcement Learning

As discussed in Section 1.1, XRL is a sub-field of XAI so it can always use model-agnostic XAI methods and follows XAI taxonomy. [12] presents two perspectives to categorise XAI methods. The first one is the time of explanation generation, which can be Intrinsic (during building models) or Post-hoc (after building models). In addition, intrinsic methods are always model-specific, while post-hoc methods can be either model-specific or model-agnostic. The second category is based on the scope of explanation. If an explanation can interpret the overall performance of a model, then it is called a Global method. Otherwise, if an explanation focuses on a particular data instance, it is a Local method. Table 2.1 shows four XRL methods with their categories that fall into such XAI taxonomy[6].

	Global Explanation	**Local Explanation**
Intrinsic	Programmatically Interpretable RL [20]	Hierarchical Policies [16]
Post-hoc	Reward Decomposition [6]	SHAP [14]

Table 2.1: Example XRL methods and their categorisation according to the time and scope of the explanation, adapted from [12].

2.4 SHapley Additive exPlanations (SHAP)

As the above table shows, SHAP is a post-hoc and local explanation method. It is also a model-agnostic method that can be applied to all models, including RL ones. Proposed by [7], SHAP draws inspiration from Shapley value [15] in cooperative game theory. Shapley value is designed to address a distribution problem: “When a coalition

[6]Since RL models are a subset of AI (or machine learning) models, all model-agnostic XAI methods are also model-agnostic XRL methods, and they are applicable to RL models. On the other hand, model-specific XRL methods can also be identified as model-specific XAI methods. However, the converse does not hold for either case.

of players achieves a payout together, how can the payout be fairly distributed among all these players?" In response to this question, [15] defines a fair distribution through four axioms: **Efficiency** (the sum of contributions equals to the payout), **Symmetry** (identical players have equal contributions), **Dummy** (a player who has no impact on the payout contributes zero), and **Additivity** (a player's contribution for the sum of the games equals to the sum of the player's contributions for all games). These four axioms[7] yield a unique solution known as Shapley value. [10] summarises that Shapley value of a player can be recognised as the weighted average of its marginal contributions across all possible coalitions it involves. Here, a player's marginal contribution to a coalition is calculated as the difference between the value functions of this coalition with and without this player's presence[8]. Such a marginal contribution's weight depends on the size of the coalition and the value function of a coalition represents the payout generated by that coalition. If a coalition is empty (i.e., has no players), its value function should be zero.

[7] transforms Shapley value from game theory to machine learning. In this context, a machine learning prediction is treated as a cooperative game, by viewing each input feature as an individual "player". And the payout is now the predicted value minus the average model prediction. This is because when there are no known features (i.e., each feature is absent and is replaced by a random variable), the model outputs the average prediction rather than zero. Regarding this game-like setting, the above axioms can also guarantee a unique solution in machine learning, namely SHAP [10]. Compared with the question addressed by Shapley value, SHAP aims to answer: "Given a model prediction, how can the contribution of each input feature be fairly distributed?"

Assuming there is an input data instance $x^{(i)}$ with j features and the model prediction is $f(x^{(i)})$, X stands for a random variable of the input data instance. Table 2.2 shows other notations that are used for calculating SHAP values[9] and supporting axioms, with their meanings in game theory and machine learning [10].

The value function is the core of SHAP. Given that a machine learning model has a fixed number of input features, even when certain features are absent (i.e., in C), their indices still need to be accounted for. The solution is randomly sampling values for these indices and then integrating them over their distributions. [10] denotes $x_S^{(i)} \cup X_C \in \mathbb{R}^p$ as

[7] There was a fifth axiom called Linearity, which has been shown that it can be derived from the other axioms, thus bringing no new fairness requirements.

[8] A new coalition is created when adding a player to an existing coalition without this player.

[9] SHAP value(s) means the value(s) generated by SHAP, while Shapley value can refer to the method or a single value.

Game Theory	Machine learning	Notation
Player	Feature index	j
Coalition	Set of features	$S \subseteq \{1,...,p\}$
Not in coalition	Features not in coalition S	$C := \{1,...,p\} \setminus S$
Coalition size	Number of features in coalition S	$\lvert S \rvert$
Number of players	Number of features	p
Payout	Prediction for $x^{(i)}$ minus average prediction	$f(x^{(i)}) - \mathbb{E}(f(X))$
Value function	Prediction for feature values in coalition S minus average prediction	$v_{f,x^{(i)}}(S)$
Shapley value of the jth player	SHAP value of feature j	$\phi_j^{(i)}$

Table 2.2: Notations for SHAP with meanings in game theory and machine learning, adapted from [10].

a p-dimensional real number feature vector, where $x_S^{(i)}$ represents the values at indices S coming from $x^{(i)}$ and X_C stands for the remaining values at indices C that are random variables from X with distribution $\mathbb{P}_{X_C}$. The value function is derived as follows [10]:

$$v_{f,x^{(i)}}(S) = \int f(x_S^{(i)} \cup X_C) d\mathbb{P}_{X_C} - \mathbb{E}(f(X)). \tag{2.9}$$

This value function ensures that an empty coalition has a value of 0 (i.e., $v(\emptyset) = 0$). And this value function can also determine feature j's marginal contribution to S [10]:

$$\begin{aligned} v_{f,x^{(i)}}(S \cup j) - v_{f,x^{(i)}}(S) &= \int f(x_{S\cup j}^{(i)} \cup X_{C\setminus j}) d\mathbb{P}_{X_{C\setminus j}} - \mathbb{E}(f(X)) \\ &\quad - \left(\int f(x_S^{(i)} \cup X_C) d\mathbb{P}_{X_C} - \mathbb{E}(f(X)) \right) \\ &= \int f(x_{S\cup j}^{(i)} \cup X_{C\setminus j}) d\mathbb{P}_{X_{C\setminus j}} - \int f(x_S^{(i)} \cup X_C) d\mathbb{P}_{X_C}. \end{aligned} \tag{2.10}$$

Furthermore, the SHAP value of feature j of the instance $x^{(i)}$ is [10]:

$$\phi_j^{(i)} = \sum_{S \subseteq \{1,...,p\} \setminus j} \frac{|S|!(p-|S|-1)!}{p!} \left(\int f(x_{S\cup j}^{(i)} \cup X_{C\setminus j}) d\mathbb{P}_{X_{C\setminus j}} - \int f(x_S^{(i)} \cup X_C) d\mathbb{P}_{X_C} \right). \tag{2.11}$$

Similar to Shapley value, SHAP value of a feature can also be recognised as the weighted average marginal contribution of all its possible coalitions, which satisfies the four axioms [10].

Efficiency: the sum of SHAP values of a data instance equals the difference between its model prediction and the average prediction: $\sum_{j=1}^{p} \phi_j^{(i)} = f(x^{(i)}) - \mathbb{E}(f(X))$.

Symmetry: if two features j and k equally contribute to all possible coalitions, their SHAP values should be the same. In other words, if $v_{f,x^{(i)}}(S \cup \{j\}) = v_{f,x^{(i)}}(S \cup \{k\})$ for all $S \subseteq \{1,...,p\} \setminus \{j,k\}$, then $\phi_j^{(i)} = \phi_k^{(i)}$.

Dummy: if a feature can not change the model predictions for all possible coalitions, its SHAP value should be zero. That is, if $v_{f,x^{(i)}}(S \cup \{j\}) = v_{f,x^{(i)}}(S)$ for all $S \subseteq \{1,...,p\} \setminus j$, then $\phi_j^{(i)} = 0$.

Additivity: a feature's SHAP value for the sum of the models equals the sum of the feature's SHAP values for all models (e.g., ensemble models [10]).

2.4.1 Estimating SHAP Values

Although SHAP can fairly distribute contributions of each feature, there are two difficulties that make the exact calculation of SHAP values infeasible, so we have to use approximations [10].

The first difficulty is about the value function defined in Equation (2.9), which contains an integration over the distribution of absent features. However, this integral remains uncomputable as the feature distribution is unknown and only data instances are observable. This issue can be tackled by estimation methods, for example, Monte Carlo integration. Monte Carlo integration approximates the integration over distributions by randomly sampling data instances and then calculating their average. In SHAP, there is a term called "background data" or "masker", referring to a dataset used to draw samples to replace absent features. By averaging the predictions of such replacements, the integral can be estimated. Following the law of large numbers, a larger number of random samples is more likely to have an accurate estimation.

The second difficulty is that the number of coalitions grows exponentially with the number of features. If a model has p input features, there will be 2^p possible coalitions. This problem causes the summation in Equation 2.11 time-consuming, especially when the model has many input features. The solution is to apply SHAP value estimators to sample coalitions instead of summing over all coalitions. Although SHAP is a model-agnostic method, there are also model-specific SHAP value estimators that are designed to speed up estimation (e.g., Linear SHAP for linear models, Deep SHAP for deep networks [7]). Our project focuses on applying model-agnostic estimators, and will introduce more information in Section 3.2.

2.4.2 Related Work

Given that our project aims to explore a SHAP application, we review other SHAP applications, as listed in the following examples:

- Traffic light control [14]: SHAP explains the decision-making process of Policy Gradient RL agents.
- Anomalies detection [1]: SHAP is used to uncover why an instance is classified as anomalous by Autoencoder.
- Real-time accident detection [11]: SHAP aids in analysing feature importance and dependency when predicting the occurrence of accidents by XGBoost.
- Process management of wastewater treatment plants [22]: SHAP facilitates performance comparison and model interpretation of Random Forest, XGBoost and LightGBM.
- NO_2 forecasting [19]: SHAP provides insights into the predictions of a pollution time series by LSTM.

Chapter 3

Methodology

In this chapter, we present the methodology employed in this project. We use AuctionGym to train bidders and validate explanations, and apply several estimators to approximate SHAP values. A description of how to visualise our results is also provided.

3.1 AuctionGym

As discussed in Section 2.2.3, AuctionGym supports simulations of online advertising auctions under different configurations, without the need for real data. Instead, our input data - context features, are randomly sampled from predefined data distributions. We use fictitious names to refer to such features based on their indices: "Feature 1", "Feature 2" and so on and so forth. By manipulating context features and other settings in AuctionGym, we can train RL bidder models and validate SHAP results.

3.1.1 Bidder Training

Before the training process, we provide AuctionGym with a configuration file that includes settings for both auctions and bidders. Each bidder is represented by a unique number, allowing us to obtain information about a specific bidder model through its number. We also specify arguments for a training mode, initial model parameters, and data distributions to sample context features. For each training iteration, several rounds of auctions take place, and all bidders are updated accordingly after each iteration. During an auction round, a subset of these bidders is chosen as participants and follows the auction process described in Section 2.2. Performance evaluation metrics for bidder models are recorded per iteration, yet typically only models from the final iteration are

saved, unless the training mode suggests storing checkpoint models (i.e., models from intermediate interactions) as well.

3.1.2 Explanation Validation

Since all context features in this project are synthetic, human-centred evaluation (comparing explanations with human experts' knowledge in this domain) can not be applied. Alternatively, we validate our explanations by using AuctionGym. [4] shows the simulation of a conversion event in AuctionGym: the probability of this event occurring depends on a sigmoid function of the dot product between a context feature vector and advertisement-specific parameters (real numbers). As each feature in a context vector comes from a data distribution, when the range of a feature's data distribution widens, it results in a wider range of the probability for conversion events. Consequently, utilities vary widely, prompting bidders to learn bids across a wider range to optimise their policies (i.e., a wider range of model predictions). Therefore, if we widen each feature's range differently, we anticipate that a feature with a wider range contributes more to a model prediction, as it can widen the range of the prediction further. Since SHAP values denote a feature's contribution to its model prediction, we assume that the range of features can be reflected by SHAP values.

In our project, we control the range of features via "weights". We define "weights" (of features) as a vector that has the same size as the number of features and is element-wise multiplied by the context feature vector. As we can control the distribution that a set of context features follows in AuctionGym, we can introduce a weight vector to manually modify context feature values. Thus, we expect that a higher weight of a feature will lead to a higher SHAP value. Also, as a sanity check, we state that a feature with zero weight will have a SHAP value of zero due to its lack of contribution. However, considering the random nature of our context features and the complexity of our model (e.g., interactions between item embeddings and private valuations), we accept that the influence of weights on SHAP values (1) might be not obvious especially when the difference between feature weights is slight; (2) is not linear and can only be presented as a rough trend. In addition, the effect of weights can be better shown when grouping SHAP values of several data instances that come from the same distribution[1], as a single instance is more likely to be unstable or sensitive to noise.

[1] Although SHAP is a local explanation method, such a grouping is able to provide a "global" overview.

3.2 SHAP Value Estimators

As mentioned in Section 2.4.1, it is impractical to sum over all coalitions when generating SHAP values. Instead, we need to sample parts of coalitions. The authors of [7] implemented a Python package - `shap`[2], which supports various SHAP value estimators. Even though model-specific SHAP value estimators are generally better (faster and relatively more accurate) [10], we use model-agnostic estimators since `shap` does not contain a model-specific estimator that can be directly applied to our bidder models. So far, there are five model-agnostic estimators available in `shap`: Kernel, Permutation, Exact, Sampling and Partition. This project focuses on the first two estimators but also tries the others for comparison purposes.

3.2.1 Kernel Estimator

Kernel estimator is originally proposed in [7], which is described as the combination of Shapley Values and Linear LIME (Local Interpretable Model-Agnostic Explanations [13]). LIME is another model-agnostic XAI approach that trains a local surrogate model g whose individual prediction is close to the one given by the original model f, and g is allowed to use a simplified input compared with f's [9]. [7] introduces Kernel SHAP as a special application of LIME. In Kernel SHAP, to explain how a model f behaves on an instance x with M features, a surrogate model g is constructed as a linear model[3]. Such g can yield estimated SHAP values of x. That is, a coefficient of an index in g can be recognised as an estimated SHAP value of the feature in x with the same index. Since SHAP utilises the concept of coalitions, both f and g should be able to deal with coalitions as their inputs. If an original coalition input for f is z, then g can use its simplified version z'. This further asks an auxiliary function h_x that can covert the simplified coalition input back to the original one: $h_x(z') = z$. Specifically, assuming all sampled coalitions are stored in Z, each input is sampled as a binary coalition instance $z' \in \{0,1\}^M$. In this context, for a specific index, if its corresponding feature value in z' is 1, then it means that the feature in z with the same index is present in this coalition while 0 means absent.

In order to have an accurate estimation, g is optimised by minimising its loss function. This loss function L is defined as a weighted sum square error over Z, where the error of z' is the difference between the original model prediction $f(z)$ (or $f(h_x(z'))$)

[2] `https://github.com/shap/shap`

[3] Linear models are supposed to be explainable since their coefficients can indicate model behaviours.

and the surrogate one $g(z')$. The weights are determined by a kernel π_x that shows how important a coalition z' is, denoting $|z'|$ as the occurrence of non-zero elements in z' (i.e., the number of features present in z):

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)} \tag{3.1}$$

With this kernel, [7] formulates the loss function as:

$$L(f,g,\pi_x) = \sum_{z' \in Z} (f(h_x(z')) - g(z'))^2 \pi_x(z') \tag{3.2}$$

The downside of Kernel SHAP is that its explanation speed will become much slower, when the number of features or background data instances increases. To mitigate this issue, `shap` suggests using k-means to summarise background data when there are more than 100 instances. K-means is an unsupervised machine learning clustering algorithm that groups data points into distinct clusters based on their similarities. The goals of k-means are minimising the variance within each cluster and maximising the variance between different clusters.

3.2.2 Permutation Estimator

Although Kernel SHAP is the original model-agnostic estimator, the Permutation estimator becomes a better option as it is faster and also more accurate [10]. The core concept behind the Permutation estimator involves randomly sampling feature permutations to create a subset of coalitions and then applying Monte Carlo integration to these coalitions. For each permutation, features are successively added to their preceding coalition according to their order. Taking the idea of antithetic sampling[4] [8], this process of adding features operates both forwards and backwards. Therefore, each feature obtains two marginal contributions from a permutation. By averaging all marginal contributions of a feature, its SHAP value can be estimated. Denoting m as the number of permutations, $o(k)$ as the k-th permutation and $-o(k)$ as its reverse, and $\widehat{\Delta}$ is the corresponding marginal contribution [10]:

$$\widehat{\phi}_j^{(i)} = \frac{1}{2m} \sum_{k=1}^{m} (\widehat{\Delta}_{o(k),j} + \widehat{\Delta}_{-o(k),j}) \tag{3.3}$$

[4] Antithetic sampling is used to reduce variance in Monte Carlo integration, by generating paired samples who are symmetry around a certain point.

3.2.3 Other Estimators

There are three other model-agnostic estimators supported by `shap` but are not preferable in our project due to their limitations.

Exact estimator ignores the second difficulty mentioned in Section 2.4.1 and simply sums all 2^p coalitions in Equation (2.11). It is inefficient and not scalable, as it can only handle cases with fewer than approximately fifteen features (the exact number might vary depending on the size of the background data).

Sampling estimator randomly selects part of coalitions and uses Monte Carlo integration to approximate SHAP values. Although it offers efficient computation, it suffers from inaccuracy due to its random sampling.

Partition estimator takes a hierarchy of features into consideration, which groups features and assigns contributions to features at a group level. However, as our features are entirely independent, it is unsuitable and might lead to inaccurate estimation as well.

3.3 Visualisation

After estimating SHAP values, we visualise them through plots and dashboards.

3.3.1 Plot

Beyond estimating SHAP values, `shap` also supports generating plots that display SHAP values. The two used in our project are **waterfall** and **summary** (also known as beeswarm) plots. A waterfall plot focuses on the local behaviour, which draws SHAP values for a single data instance and demonstrates how much a feature contributes to the model prediction compared with the average prediction. In contrast, a summary plot provides a global view by presenting SHAP values for a bunch of data instances (usually another dataset that differs from the background data) together.

3.3.2 Dashboard

We use another Python package - `shapash`[5], to develop web-based dashboards. Although SHAP value estimation in `shapash` simply calls the one deployed by `shap`, it visualises SHAP values in another way. Instead of a single static plot, it integrates several plots and enables user interactions.

[5] `https://github.com/MAIF/shapash`

Chapter 4

Implementation

In this chapter, we introduce our three-step pipeline: training a bidder in AuctionGym, generating data for explanation and applying SHAP to explain the bidder's behaviours.

4.1 Bidder Training

Before training our bidder models, we first reproduce an initial experiment (described in [4]) in AuctionGym to ensure the reliability of our environments. We then follow a similar structure as the original implementations, but we only train models once instead of multiple runs. This is because this project focuses on a particular model's behaviours rather than the average and we prefer to run various experiments given the time limitation. The original implementations contain functions to parse configuration files, simulate advertisement opportunities based on configurations and generate model evaluation metrics. Given our research questions in Section 1.2, we implement functions to provide various data distributions that context features can sample from. Notably, we stick to the default setting in AuctionGym that all bidder models (shallow multi-layer perceptrons [4]) are trained from scratch simultaneously and use the same utility estimators. Although we can access all bidders' information, we only record one bidder's[1] details (e.g., surplus and bidding policy) for explanation purposes.

We adjust the training by controlling these arguments: the utility estimator for each bidder, auction type, whether to save checkpoint models, data distribution of context features and the embedding size of context features. The unchanged training settings include a random seed of 0, 50 iterations, 5000 auction rounds per iteration, 3 bidders

[1]We always select the bidder with an index of 0 to ensure its validity (i.e., its index will not be out of range), regardless of the total number of bidders.

in the environment, 2 out of 3 bidders participating in each auction, 12 items in each bidder's catalogues (as we fix the random seed, item embeddings in bidder's catalogues also remain the same), a Bayesian logistic regression with Thompson sampling allocator for each bidder [4] and a ratio of 0.8 for observable to unobservable context features.

4.2 Simulation Data Generation

Regarding data for explanations, we implement functions that support generating different sizes of data that follow the same distribution as the one used to train the bidders. This approach prevents out-of-distribution data from compromising model performance. Then, we randomly split the data into background and explanation datasets with a ratio of 0.8. In addition, we set a fixed seed of 0, to ensure the reproducibility of the datasets.

4.3 SHAP

We use `shap` for estimating SHAP values. Before explaining bidder models, we test it on a toy linear example to ensure SHAP value estimators can work as we expected. To build a SHAP explainer, we need to specify a model prediction function, the background dataset (Section 4.2) and the type of value estimator. We develop the model prediction function utilising a trained RL bidder (Section 4.1), which can input context features and subsequently output a bid. We also set a fixed seed of 0 for the explainer to enable reproducibility. Then we use the built explainer to estimate SHAP values for each instance in the explanation dataset, which involves predicting the sampled coalitions. After such estimation, we visualise SHAP values via plots and dashboards as mentioned in Section 3.3. As the summary plot can provide insights from a global perspective, we only present one waterfall plot for a local instance as an example for each bidder-dataset-explanation triplet.

Chapter 5

Experimental Results and Discussion

In this chapter, we group our experiments into three categories based on our research questions in Section 1.2: learning to bid, SHAP and context feature. Each experiment is presented following a structure of **Description** - **Assumption** - **Result and Discussion**.

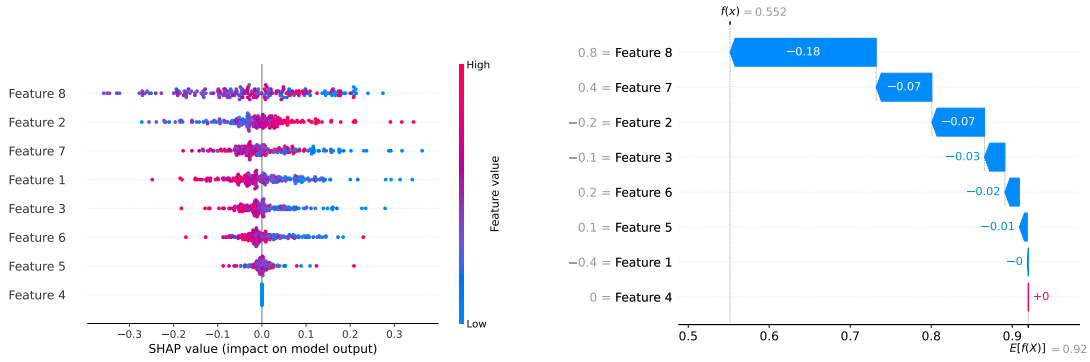
Our experiments were conducted on the teaching cluster[1]. Unless otherwise specified, we employed the following default settings: DR utility estimator for bidder models, first-price auctions, the final iteration model for explanation, Permutation SHAP value estimator, 800 background data instances and 200 explanation data instances (0.8 ratio in Section 4.2). Each data instance comprised 8 observable context features and 2 unobservable features (0.8 ratio in Section 4.1). All observable features follow a normal distribution with a mean of zero and a variance of one (i.e., $N(0,1)$), weighted by a vector of $[-4,-2,-1,0,1,2,4,8]$. For simplification, all unobservable features in our experiments also follow $N(0,1)$ but without additional weights. We suppose such a setting is reasonable for three reasons: (1) most settings[2] supported by [4] and [10] due to its relevance and common use; (2) 1000 data instances in total and 8 features ensures efficient SHAP value estimation within a reasonable time (approximately 4.5 hours); (3) a feature weight vector can be used for validation as discussed in Section 3.1.2.

Figure 5.1 illustrates two SHAP value plots (introduced in Section 3.3.1) under the default experiment setting. Figure 5.1a is a summary plot that visualises 200 data instances. Each data point represents a SHAP value for a specific feature of an instance. The colour of the point indicates how this feature's value compares to the values of the same feature in other instances. Features are ordered according to their importance rankings, which are determined by summing the absolute SHAP values across each

[1] `https://computing.help.inf.ed.ac.uk/teaching-cluster`

[2] They are: DR utility estimator, first-price auctions, the final iteration model, Permutation SHAP value estimator, two 0.8 ratios and $N(0,1)$.

feature. Here, "Feature 8" is ranked as the most important feature while "Feature 4" is the least important one. Such ranking provides a basic idea of the validation process: features with the highest and lowest weights (are likely to) contribute the most and least to model predictions based on their SHAP values. Rankings of other features are not as obvious as these two. One potential reason is that their weights are relatively similar. To be noted, we can not observe the effect of weights' signs here. We will further discuss this problem in Section 5.3.1. Figure 5.1b is a waterfall plot that presents the local explanation for an instance of $[-0.4, -0.2, -0.1, 0, 0.1, 0.2, 0.4, 0.8]$. The order of features is also ranked by feature importance (i.e., the absolute value of SHAP values), accompanied by feature values on the side. We find that the rankings of "Feature 8" and "Feature 4" remain consistent with those in the summary plot, while the rankings of other features show variation.



(a) A summary plot that shows SHAP values of 200 data instances.

(b) A waterfall plot that displays SHAP values of $[-0.4, -0.2, -0.1, 0, 0.1, 0.2, 0.4, 0.8]$.

Figure 5.1: SHAP value plots using the default experiments setting, including DR utility estimator, first-price auctions, Permutation SHAP value estimator and 800 background data. All context features come from $N(0, 1)$ and are weighted by $[-4, -2, -1, 0, 1, 2, 4, 8]$.

Considering the scope of this dissertation, we only present representative experiments and their results in this chapter. Table 5.1 shows our selected experiments and their descriptions. The complete experiment lists can be found in our appendices: Appendix A for bidder training and Appendix B for SHAP information.

Additionally, to avoid Redundant expressions, we will occasionally refer to a model by its most distinct characteristics. For instance, a model utilising DR utility estimators might simply be denoted as "DR".

	Experiment		Description
Section 5.1 **Learning to Bid**	Utility Estimators		DR *versus* DM and IPS
	Auction Type		First-Price *versus* Second-Price
	Model Convergence		The Final Iteration Model *versus* Checkpoints
Section 5.2 **SHAP**	SHAP Value Estimators (4 and 8 features)		Permutation *versus* Kernel, Exact, Sampling and Partition
	Background Data Size (Kernel Estimator)		800 *versus* 80 and 800 into 80 clusters using k-means
Section 5.3 **Context Feature**	Validation: Feature Weight		Details in Table 5.4
	Feature Distribution	Variance of Normal Distribution	Details in Table 5.5
		Uniform Distribution	Details in Table 5.6
		Mixture Distribution	Details in Table 5.7
	Number of Features (Mixture Distribution)		8 *versus* 16, 24, 32 and 40 features

Table 5.1: Experiments with descriptions that are grouped by research questions in Section 1.2, all other settings are the default ones.

5.1 Learning to Bid

5.1.1 Utility Estimators

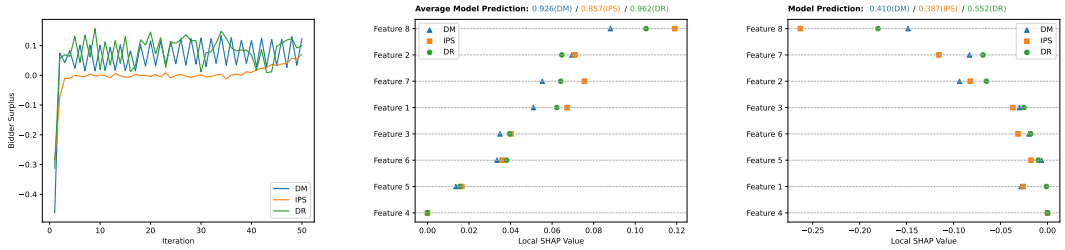
Description: As mentioned in Section 2.2.2, [4] supposes that the bidder models utilising DR utility estimators outperform those using DM or IPS, and validates this argument by running experiments in AuctionGym. In the experiments conducted by [4], model performance is evaluated by three metrics: Social Welfare (the overall value generated in auctions), Auction Revenue (the value earned by the auctioneer) and Social Surplus (the average value gained by all bidders). As our project focuses on a single bidder's behaviours, we defined a metric called Bidder Surplus to present the surplus obtained by this particular bidder. In this experiment, we would like to explore whether the model using DR also has the best performance in our Bidder Surplus metric and figure out how utility estimators affect SHAP values from both global and local views.

Assumption 1: The model using DR will yield the most bidder surplus.

Assumption 2: When other settings are kept the same, models with different utility estimators will provide similar average predictions and feature importance rankings.

Result and Discussion: Figure 5.2 shows the comparison among models that use different utility estimators (DM, IPS and DR). From Figure 5.2a, we find that the models using DR and DM achieve similar bidder surpluses higher than IPS. This can not fully support our Assumption 1. We suppose one potential reason is that we only run the training process once, which contains a random effect. Regarding Assumption 2, Figure 5.2b displays the model average prediction and the absolute mean SHAP values of 200 instances across features. We note that the model using IPS gives a lower average bid prediction than DR and DM. We hypothesise that this might explain why the model

using IPS also has a relatively lower bidder surplus. We also observe that the majority of points in the upper row are positioned to the right of points with the same colour. As features are ranked by importance in DR, this observation implies that the models using DM and IPS rank features similarly to DR. However, since points are not completely overlapped, the exact values of how much they assign to each feature are varied. Turning into a local view, Figure 5.2c presents local explanations for the same instance in Figure 5.1b. Although some features have similar SHAP values, their predicted bids and other features yield different results, especially for the most important feature. We accept that a local instance might be more unstable than the average of all instances. Hence, we believe that the models using DM, IPS, and DR allocate feature contributions distinctively while still sharing similarities in decision-making.



(a) Evolution of the bidder surplus for utility estimators during training iterations.

(b) Absolute mean SHAP values for utility estimators, ordered by DR's rankings.

(c) SHAP values of a local instance for utility estimators, ordered by DR's rankings.

Figure 5.2: Comparison of models using different utility estimators (DM, IPS, and DR) in terms of bidder surplus, absolute mean SHAP values, and SHAP values for an instance of $[-0.4, -0.2, -0.1, 0, 0.1, 0.2, 0.4, 0.8]$.

5.1.2 Auction Type

Description: We describe two auction types in Section 2.2. Even though first-price auctions are more common than second-price auctions in concurrent online advertising and bidders do not know the auction type beforehand, we are still interested in understanding how the influence of second-price auctions on bidder decisions differs from that of first-price auctions. We suppose that our RL bidders can learn about the auction type since it impacts their payment and subsequently their utility. An experiment conducted by [4] demonstrates that models participating in second-price auctions fluctuate less (i.e., the models' behaviours are more likely to remain consistent) after converging and

can achieve a higher surplus than in first-price auctions. In this experiment, we further explore these observations using a similar approach to our previous experiment.

Assumption 3: After converging, models participating in second-price auctions will be more stable than in first-price auctions.

Assumption 4: Models participating in second-price auctions can obtain higher bidder surplus than in first-price auctions.

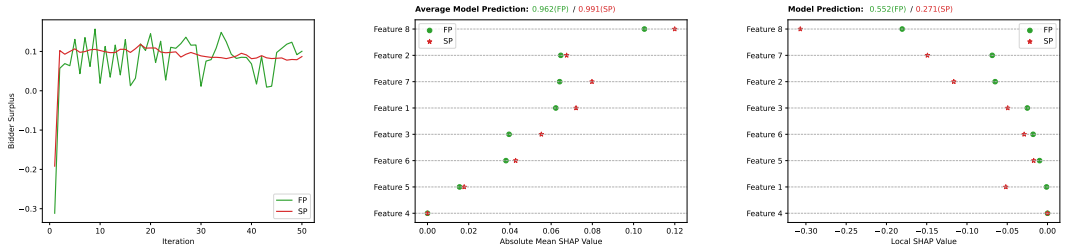
Assumption 5: Models participating in both second-price and first-price auctions tend to rank feature importance similarly. However, their bidding behaviours might vary due to the differences in their rewards influenced by the auction types.

Result and Discussion: Figure 5.3 presents the comparison between models participating in first-price and second-price auctions. Figure 5.3a illustrates that the bidder surplus in second-price auctions remains relatively stable after the initial iterations, whereas in first-price auctions, it continues to fluctuate. This observation supports Assumption 3. Although the bidder surplus in second-price auctions fluctuates less, we identify that it is only around the average of the surplus in first-price auctions, rather than consistently higher. A potential reason might be the same random effect in Section 5.1.1. Thus, Assumption 4 can not be supported by our experiment. Regarding Assumption 5, Figure 5.3b and Figure 5.3c show that from both global and local views, the rankings of features slightly differ between models in first-price and second-price auctions. However, their exact SHAP values vary more obviously. In addition, the model in second-price auctions tends to place a much lower bid for this specific local instance, in contrast to its behaviours for the overall model predictions that average out. As a result, we argue that the models in first-price and second-price auctions maintain similarities mainly at the feature ranking level, indicating a likely difference in the assignment of specific feature importance.

5.1.3 Model Convergence

Description: Model convergence has been observed in the work by [4] during the training of bidders. We would like to delve into this observation from an explanatory perspective. In this experiment, we re-train our bidders using a "progress saving" mode[3] that saves checkpoint models per three iterations, and the zero iteration is stored without any training. Every checkpoint model is then explained using the Exact estimator, given its faster explanation speed (we will discuss more about estimators in Section 5.2.1).

[3]This mode introduces additional randomness, which makes its results slightly different from the default ones.



(a) Evolution of the bidder surplus for auction types during training iterations.

(b) Absolute mean SHAP values for auction types, ordered by first-price's rankings.

(c) SHAP values of a local instance for auction types, ordered by first-price's rankings.

Figure 5.3: Comparison of models participating in first-price and second-price auctions in terms of bidder surplus, absolute mean SHAP values, and SHAP values for an instance of $[-0.4, -0.2, -0.1, 0, 0.1, 0.2, 0.4, 0.8]$.

Assumption 6: If a model converges, its model behaviours (average prediction and feature importance) should also be stable.

Result and Discussion: Figure 5.4 shows the evolution of the bidder surplus (indicating model convergence), average model prediction, absolute mean SHAP values and SHAP values for a local instance. We observed that the bidder starts to converge after approximately nine iterations. In the meantime, both the average model predictions and SHAP values (from both global and local views) tend to be stable. Thus, we suppose that this result can support Assumption 6.

5.2 SHAP

5.2.1 SHAP Value Estimators

Description: In Section 3.2, we present model-agnostic SHAP value estimators available in the `shap` package. Among these, the Permutation estimator is more suitable than the Kernel estimator, whereas the Exact, Sampling and Partition estimators are even less suitable for our project. In this experiment, we explore this argument by comparing their accuracy and the time cost to explain 200 instances. Since fully accurate SHAP values cannot be obtained due to the first difficulty mentioned in Section 2.4.1, we consider the SHAP values estimated by the Exact estimator as the baseline for the accuracy metric. The accuracy of each estimator is calculated as the average similarity across all 1600 SHAP values (8 features multiplied by 200 instances). Each similarity is determined by

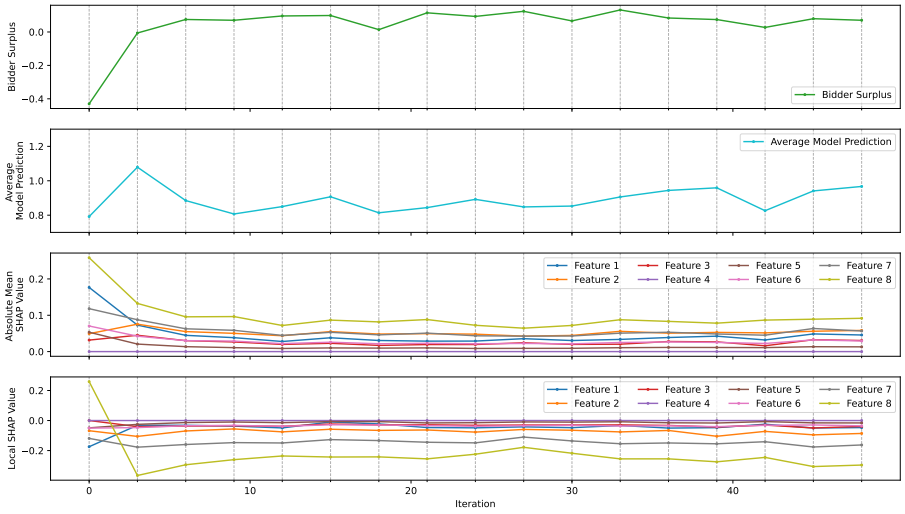


Figure 5.4: Evolution of the bidder surplus, average model prediction, absolute mean SHAP values and SHAP values for an instance of $[-0.4, -0.2, -0.1, 0, 0.1, 0.2, 0.4, 0.8]$ across all eight features during every three training iterations.

the absolute value of the difference between the values computed by the Exact estimator and the current estimator, divided by the value yielded by the Exact estimator. We not only compare the default settings with eight features, but also examine a scenario with four features whose distributions are $N(0,1)$ and a weight vector of $[2, 0, -4, 8]$.

Assumption 7: The Permutation estimator has the overall best performance among the five estimators.

Assumption 8: The Kernel estimator is more suitable than the Exact, Sampling and Partition estimators in our project.

Result and Discussion: Table 5.2 presents the comparison of SHAP value estimators. Based on the accuracy metric, we detect that the Permutation estimator is always better than the Kernel, Sampling and Partition estimators. Except for the Exact estimator, all other estimators have a lower accuracy when the number of features grows. Also, when there are eight features, the Kernel estimator obtains a higher accuracy than the Sampling and Partition estimators. Regarding the explanation time, we observe that both the Permutation and Sampling estimators maintain similar results for four and eight features. This observation aligns with their implementations in `shap`, which involves a fixed number of permutations or coalitions to be sampled. The time for other estimators increases when increasing four to eight features, especially the Kernel

one. Beyond this table, we find that when we set sixteen as the number of features, the explanation time for the Kernel estimator becomes longer than 40 hours while the Exact one is not supported as mentioned in Section 3.2.3. The failure of the Exact estimator prevents us from comparing the Permutation, Sampling and Partition estimators. However, we hypothesise that the Permutation estimator is likely to be more accurate than the others based on our current results. Considering Assumption 7, we suppose that the Exact estimator is the best choice when it is feasible, but the Permutation estimator can be the most appropriate replacement when we can not apply the Exact estimator in our project. This finding aligns with the principle of how the "auto" option chooses which estimator to use in `shap` [10]. Although Exact is applicable when there are eight features, we still choose Permutation as our default setting because of its flexibility for more features. In addition, the Kernel estimator can achieve higher accuracy than the Sampling and Partition estimators, but also costs a longer time. So Assumption 8 can only be supported by our results, when we define "suitable" as the accuracy.

Number of Features	SHAP Value Estimator	Accuracy (Compared with Exact)	Time for 200 Instances
4	Permutation	99.02%	04:22:18
	Kernel	85.26%	00:20:45
	Exact	100.00%	00:06:22
	Sampling	84.15%	00:18:04
	Partition	95.15%	00:07:20
8	Permutation	94.46%	04:24:39
	Kernel	87.58%	07:03:42
	Exact	100.00%	01:13:19
	Sampling	81.74%	00:18:43
	Partition	81.23%	01:56:48

Table 5.2: Performance of SHAP values estimators for 4 and 8 context features. Accuracy is compared against the Exact estimator, and the time taken (in hours:minutes:seconds) is for explaining 200 instances using 800 background data.

5.2.2 Background Data Size

Description: Our previous experiment shows that the Kernel estimator is the most time-consuming SHAP value estimator among the five. As mentioned in Section 3.2.1, one possible solution is to use k-means to reduce background data size by grouping similar instances into clusters. In this experiment, we would like to explore whether k-means is helpful in our case. We compare the default background data size of 800,

with the background size of 80 and the one that uses k-means to group 800 into 80 clusters.

Assumption 9: K-means will reduce the time for explanation compared with the original background data size. Even though it decreases the accuracy, it should still be more accurate than directly sampling fewer instances, as it theoretically contains more information about the data distribution.

Result and Discussion: Table 5.3 displays how background data size affects explanations generated by the Kernel estimator. Our observations are that a smaller size of background data[4] (10% of the original one) obviously shortens explanation time but drops the accuracy as well. Regarding Assumption 9, applying k-means only costs about one-seventh of explanation time compared with not using it. Although it reduces the accuracy by about 20%, it is still 5% more accurate than directly sampling 80 background data instances.

Background Data Size	Accuracy (Compared with Exact)	Time for 200 Instances
800	87.58%	07:03:42
80	62.11%	00:42:44
800 into 80 clusters using k-means	67.12%	00:55:38

Table 5.3: Performance of different background data sizes for the Kernel estimator. Accuracy is compared against the Exact estimator, and the time taken (in hours:minutes:seconds) is for explaining 200 instances.

5.3 Context Feature

In contrast to the previous experiments, experiments in this section are not built upon theories. Instead, we are trying to "synthesise" or simulate different context features based on our knowledge, which might be able to complement the lack of real data. We first explore the effect of feature weights and treat it as a validation step (Section 3.1.2). Then, we design experiments to compare how different data distributions affect explanations. Finally, as there will be more than eight features and contexts features are complex in the real world, we further extend our research to forty features that come from a "mixture" distribution. We monitor the time taken by every experiment, and find all of them can be finished within five hours.

[4]In fact, the background data size of "800 into 80 clusters using k-means" is also 80.

5.3.1 Validation: Feature Weight

Description: At the beginning of this chapter, we observe two limitations of SHAP from Figure 5.1. The first limitation is that SHAP can only reflect the magnitudes of feature weights, not their signs. The second one is that when the weights between features only differ slightly, we cannot identify an obvious difference in feature ranking. In this experiment, we aim to determine if these limitations apply to other scenarios as well. In addition, to maintain the consistency of the validation process, we would like to check whether the position of a weight makes a difference (e.g., whether swapping the indices of the highest and lowest weights will change the corresponding features' importance accordingly). We are also interested in investigating how a local instance can represent weight, as we believe a global perspective is more suitable for validation, as discussed in Section 3.1.2. Table 5.4 shows our experiment settings[5], with the alias[6] of each feature weight vector and its corresponding local instance to be explained[7].

Alias	Feature Weight	Local Instance
Default	$[-4,-2,-1,0,1,2,4,8]$	$[-0.4,-0.2,-0.1,0,0.1,0.2,0.4,0.8]$
Opposite	$[4,2,1,0,-1,-2,-4,-8]$	$[0.4,0.2,0.1,0,-0.1,-0.2,-0.4,-0.8]$
Random	$[1,4,-2,8,-4,-1,0,2]$	$[0.1,0.4,-0.2,0.8,-0.4,-0.1,0,0.2]$
Identical	$[1,1,1,1,1,1,1,1]$	$[0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]$
Two	$[1,1,1,1,1,1,1,2]$	$[0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.2]$
Five	$[1,1,1,1,1,1,1,5]$	$[0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.5]$
Ten	$[1,1,1,1,1,1,1,10]$	$[0.1,0.1,0.1,0.1,0.1,0.1,0.1,1]$

Table 5.4: The feature weight experiment, which describes the alias of each experiment, and the corresponding feature weight vector and local instance to be explained. Other conditions follow the default settings.

Assumption 10: SHAP can reflect feature weights at a certain level.

Assumption 11: The position of a weight does not make a difference.

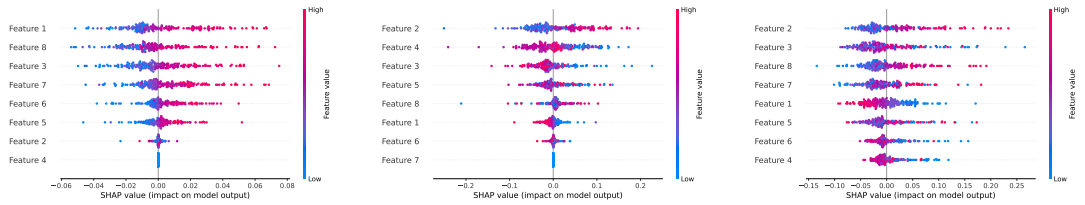
Assumption 12: When a feature holds greater importance than others, a local explanation can show its impact.

[5]The default setting is for comparison purposes.

[6]We will use these aliases to refer to specific experiments once defined, as indicated below.

[7]Beyond experiments in Table 5.4, we also run others experiments (e.g., using absolute values for all feature weights and only changing the position of the most important feature). However, as these experiments did not yield additional insights, they are not presented. This situation also occurred in the experiments discussed in Section 5.3.2.2

Result and Discussion: Figure 5.5 shows summary plots of Opposite, Random and Identical. Comparing Figure 5.5a with Figure 5.1a, even though their weights are opposite numbers and other data or settings are kept the same, we can not observe the opposite feature importance (i.e., the opposite colour spread for the same feature). Given that the embeddings of items in our bidder's catalogues might also affect the signs, we try to set all of them positive or the same value, but still fail to have a valid observation[8]. Therefore, we suppose Assumption 10 is not true in our case. Two potential reasons are the effect of signs is too complex for SHAP and we do not fully control all other influential factors. Assumption 11 can be supported by comparing Figure 5.5b and Figure 5.1, as we find a feature with higher weight tends to be more important no regardless of its position. One possible cause of why the feature importance does not rank exactly as the absolute feature value, is pointed out by Figure 5.5c. Even when features are equally weighted, their importance is not exactly the same due to the other parts of our model. Regarding Assumption 12, Figure 5.6 displays three waterfall plots of Two, Five and Ten, we observe that the highest weighted feature ("Feature 8") is the most important feature for all three plots. We also apply Two, Five and Ten to explain an instance that each feature has the same value (e.g., $[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]$) and the finding remains the same. Therefore, we believe that Assumption 12 holds for us.



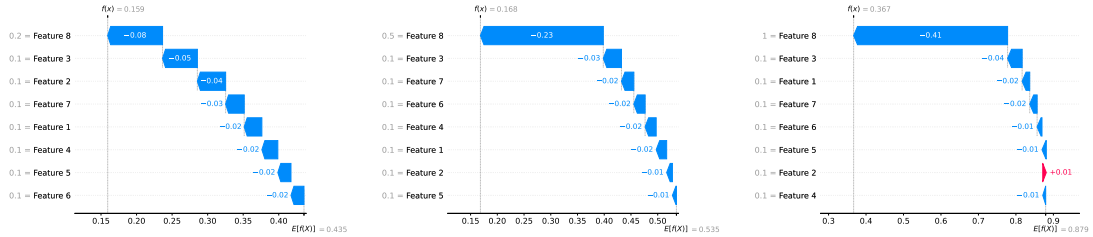
(a) Summary plot of Opposite $[4, 2, 1, 0, -1, -2, -4, -8]$.

(b) Summary plot of Random $[1, 4, -2, 8, -4, -1, 0, 2]$.

(c) Summary plot of Identical $[1, 1, 1, 1, 1, 1, 1, 1]$.

Figure 5.5: Three summary plots with different feature weight vectors as described in Table 5.4, each of which shows SHAP values of the same 200 data instances.

[8] In Section 5.3.2.2, we even restrict all the context features to be positive as well.



(a) Waterfall plot of Two $[1,1,1,1,1,1,1,2]$.

(b) Waterfall plot of Five $[1,1,1,1,1,1,1,5]$.

(c) Waterfall plot of Ten $[1,1,1,1,1,1,1,10]$.

Figure 5.6: Three waterfall plots with different feature weight vectors as described in Table 5.4, each of which shows SHAP values of a local instance whose features values are 0.1 element-wise multiply the weight vector.

5.3.2 Feature Distribution

5.3.2.1 Variance of Normal Distribution

Description: As discussed in Section 3.1.2, we suppose that a wider range of feature values is likely to increase the importance of features. In this experiment, instead of weights, we would like to explore whether the variance of a normal distribution has a similar impact on feature importance. Table 5.5 shows our settings, and we assume that a higher variance brings a wider range or a higher spread of values.

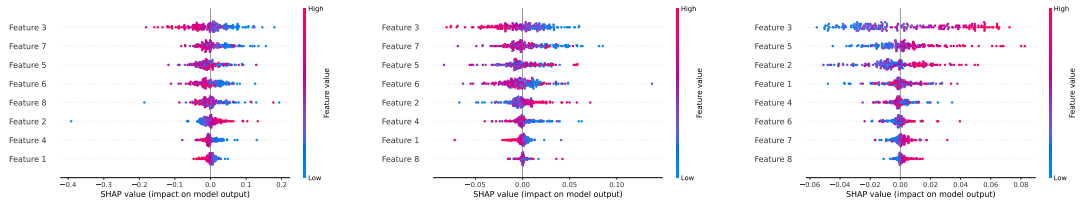
Alias	Data Distribution
Variance	$[N(0,1),N(0,2),N(0,3),N(0,4),N(0,5),N(0,6),N(0,7),N(0,8)]$
Variance Replaced	$[N(0,1),N(0,2),N(0,3),N(0,4),N(0,5),N(0,6),N(0,7),N(0,1)]$
Variance Reverse	$[N(0,8),N(0,7),N(0,6),N(0,5),N(0,4),N(0,3),N(0,2),N(0,1)]$

Table 5.5: The variance of normal distribution experiment, which describes the alias of each experiment and the corresponding data distribution (a normal distribution with a mean of zero but a different variance). Each feature is equally weighted, while other conditions follow the default settings.

Assumption 13: When a feature follows a normal distribution with a mean of zero, a higher variance tends to make the feature more important.

Result and Discussion: Figure 5.7 presents our results by three summary plots. "Feature 3" is the most important feature for all plots, though its variance is not the highest. However, if we group "Feature 1, 2, 4" ("Group A") and "Feature 5, 6, 7, 8" ("Group B") respectively, we observe that "Group A" is less important in Variance

(Figure 5.7a, with lower variance) and more important in Variance Reverse (Figure 5.7c, with higher variance) than "Group B". In addition, by comparing Figure 5.7a and Figure 5.7b, we find that "Feature 8" becomes the least important when switching its variance from eight to one. Therefore, we argue that Assumption 13 is partially true, given that "Feature 3" exhibits an unconventional behaviour.



(a) Summary plot of Variance. (b) Summary plot of Variance Replaced. (c) Summary plot of Variance Reverse.

Figure 5.7: Three summary plots whose feature distribution is different but every feature is equally weighted as described in Table 5.5, each of which shows SHAP values of the same 200 data instances.

5.3.2.2 Uniform Distribution

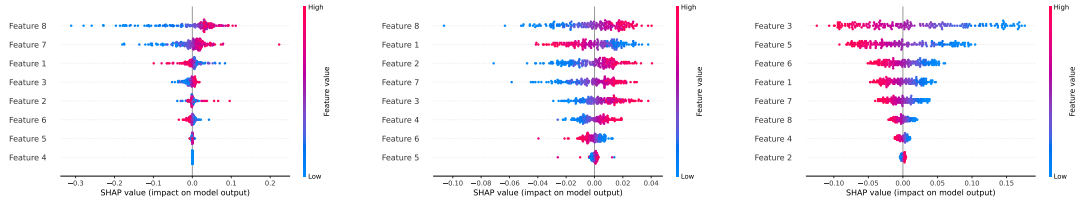
Description: Although normal distribution is common for real data, the occurrence of uniform distribution is also reasonable. In this experiment, we explore how feature weights affect a uniform distribution. Given that our normal distributions have a mean of zero (which implies features have both positive and negative values), for the purpose of comparison, we decide to set data here as only positive. Table 5.6 outlines our feature weights for the same feature vector, with each feature sampling from a uniform distribution between zero and one (i.e., $U(0,1)$).

Alias	Feature Weight
Uniform	$[-4,-2,-1,0,1,2,4,8]$
Uniform Positive	$[1,1,1,1,1,1,1,1]$
Uniform Negative	$[-1,-1,-1,-1,-1,-1,-1,-1]$

Table 5.6: The uniform distribution experiment, which describes the alias of each experiment and the corresponding feature weight vector. Each feature follows a uniform distribution $U(0,1)$, while other conditions follow the default settings.

Assumption 14: The magnitudes of weights can be generally observed, but the signs can not (modified from Assumption 10).

Result and Discussion: Figure 5.8 displays summary plots for Uniform, Uniform Positive and Uniform Negative. According to Figure 5.8a, we identify that the three highest weighted features are also the three most important and zero weight still leads to no impact. Through the comparison between Figure 5.8b and Figure 5.8c, the opposite spread can still not be detected. Thus, we argue that Assumption 14 can be supported by these results.



(a) Summary plot of Uniform $[-4,-2,-1,0,1,2,4,8]$.

(b) Summary plot of Uniform Positive $[1,1,1,1,1,1,1,1]$.

(c) Summary plot of Uniform Negative $[-1,-1,-1,-1,-1,-1,-1,-1]$.

Figure 5.8: Three summary plots whose feature weight vectors are different but every feature follows $U(0,1)$ as described in Table 5.6, each of which shows SHAP values of the same 200 data instances.

5.3.2.3 Mixture Distribution

Description: In addition to normal and uniform distribution, context features can also consist of several different distributions, which we call "Mixture Distribution". In this experiment, we have a preliminary attempt at mixture distribution. Beyond normal and uniform distribution, we define the distribution whose samples are either zero or one as "Binary (Distribution)", and the one where each sample is an integer between one and five as "5-Class (Categorical Distribution)". Since we would further increase feature numbers by repetition, Table 5.7 shows the distribution of each feature in this experiment and the next experiment.

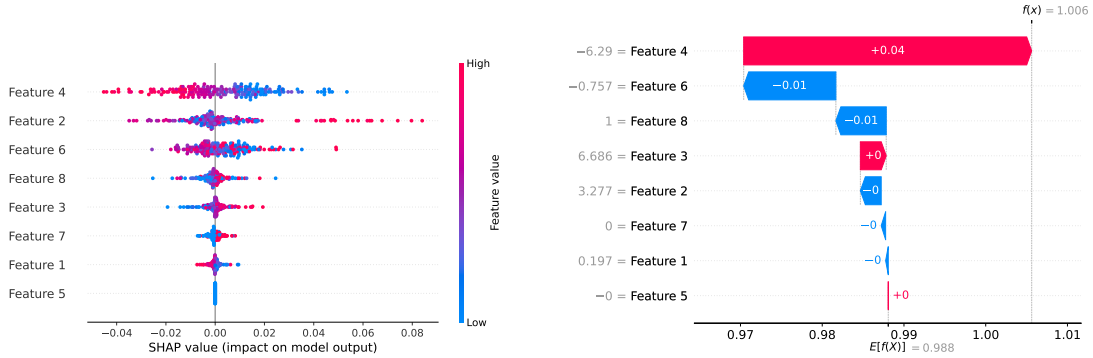
Assumption 15: A higher spread of feature values will likely result in higher feature importance (concluded from Assumption 10, 13 and 14).

Result and Discussion: Figure 5.9 shows two SHAP value plots under such mixture setting. Figure 5.9a supports Assumption 15, demonstrating that the four most important features correspond to distributions with the four highest spread. In terms of the local

Feature 1 (9/17/25/33)	**Feature 2** (10/18/26/34)	**Feature 3** (11/19/27/35)	**Feature 4** (12/20/28/36)
$N(0,1)$	$N(0,5)$	$N(5,1)$	$-5*U(-2,2)$
Feature 5 (13/21/29/37)	**Feature 6** (14/22/30/38)	**Feature 7** (15/23/31/39)	**Feature 8** (16/24/32/40)
$0*U(-2,2)$	$5*U(-2,2)$	Binary	5-Class Categorical

Table 5.7: The mixture distribution (and the number of features 16/24/32/40) experiment information, which describes the data distribution that each feature comes from. A scalar before $*$ and a distribution is regarded as the weight of this feature.

impact, Figure 5.9b illustrates a random example from the explanation dataset, where the three most significant features align with Assumption 15 as well.



(a) A summary plot that shows SHAP values of 200 data instances.

(b) A waterfall plot that displays SHAP values of a random local instance.

Figure 5.9: SHAP value plots whose experiment settings are eight features that follow the data distribution as presented in Table 5.7.

5.3.3 Number of Features

Description: To assess the scalability of our methods and the stability of our findings, we investigate scenarios where the number of features increases. In this experiment, as shown in Table 5.7, we generate 16, 24, 32, and 40 context features by replicating the earlier mixture distribution 2, 3, 4, and 5 times. Since validating explanations becomes more complex with an increased number of features, this experiment serves as a preliminary exploration, aiming to offer insights for future research.

Assumption 16: A higher spread of feature values is likely to result in higher feature importance (inherited from Assumption 15).

Result and Discussion: Table 5.8 displays the ten most important features and their respective distributions given 16, 24, 32 and 40 features (with the scenario of 8

features is for comparison), where the same colour indicates features sampling from the same distribution. Considering 8 features, we find that the distributions of the three most important features are: $-5*U(-2,2)$, $N(0,5)$ and $5*U(-2,2)$. These contributions also exhibit the three widest ranges of values (as observed in Figure 5.9a). As the number of features increases, we consistently notice that features with these three distributions tend to hold more importance than others. The distribution of the fourth most important feature in the case of 8 features, only appears when there are 16 and 24 features, and it is ranked lower than all three of the previously mentioned distributions. Based on these findings, we suppose that they support Assumption 16.

Importance Ranking	Number of Features									
	8		16		24		32		40	
	Name	Distribution	Name	Distribution	Name	Distribution	Name	Distribution	Name	Distribution
1	Feature 4	$-5*U(-2,2)$	Feature 10	$N(0,5)$	Feature 4	$-5*U(-2,2)$	Feature 12	$-5*U(-2,2)$	Feature 4	$-5*U(-2,2)$
2	Feature 2	$N(0,5)$	Feature 12	$-5*U(-2,2)$	Feature 12	$-5*U(-2,2)$	Feature 14	$5*U(-2,2)$	Feature 22	$5*U(-2,2)$
3	Feature 6	$5*U(-2,2)$	Feature 2	$N(0,5)$	Feature 22	$5*U(-2,2)$	Feature 22	$5*U(-2,2)$	Feature 20	$-5*U(-2,2)$
4	Feature 8	5-Class	Feature 4	$-5*U(-2,2)$	Feature 20	$-5*U(-2,2)$	Feature 20	$-5*U(-2,2)$	Feature 34	$N(0,5)$
5	Feature 3	$N(5,1)$	Feature 6	$5*U(-2,2)$	Feature 10	$N(0,5)$	Feature 4	$-5*U(-2,2)$	Feature 28	$-5*U(-2,2)$
6	Feature 7	Binary	Feature 14	$5*U(-2,2)$	Feature 2	$N(0,5)$	Feature 26	$N(0,5)$	Feature 12	$-5*U(-2,2)$
7	Feature 1	$N(0,1)$	Feature 8	5-Class	Feature 14	$5*U(-2,2)$	Feature 10	$N(0,5)$	Feature 26	$N(0,5)$
8	Feature 5	$0*U(-2,2)$	Feature 15	Binary	Feature 18	$N(0,5)$	Feature 6	$5*U(-2,2)$	Feature 18	$N(0,5)$
9	/	/	Feature 16	5-Class	Feature 6	$5*U(-2,2)$	Feature 18	$N(0,5)$	Feature 10	$N(0,5)$
10	/	/	Feature 3	$N(5,1)$	Feature 8	5-Class	Feature 30	$5*U(-2,2)$	Feature 30	$5*U(-2,2)$

Table 5.8: Results of the number of features experiment, which presents the ten most important features (based on absolute mean SHAP values) and their distributions when there are 8, 16, 24, 32 and 40 context features. The same colour of features indicates that they follow the same data distribution.

5.3.3.1 Dashboard

In addition, we developed an interactive dashboard that visualises the case of 40 features. Figure 5.10 is a screenshot of it. This dashboard contains four interactive units:

- **Upper-left corner** displays the ranking of the twenty most important features, based on the absolute mean feature contribution (i.e., SHAP value).
- **Bottom-left corner** displays a specific feature's SHAP value (this feature can be selected from the upper-left corner). The x-axis represents the feature value, while the y-axis indicates the feature's contribution. Each data point corresponds to this feature in an instance from the explanation dataset, with the colour denoting the model prediction for that particular instance. In this screenshot, "Feature 4" ($-5*U(-2,2)$) is presented here. Analysing these data points, we note that when

the "Feature 4" value in a given instance is positively higher, it tends to contribute more negatively to the model prediction of that instance. This more negative contribution tends to bring the prediction value closer to zero. Conversely, a more negative value of this feature, is more likely to contribute positively, thus pushing the prediction value closer to 1.

- **Upper-right corner** displays the dataset of all explained data instances, including indices of instances, model predictions and feature values.
- **Bottom-right corner**: displays the local explanation for a specific data instance (this instance can be chosen from the upper-right corner), which presents the feature contributions for that particular instance. The occurrence of features can be controlled by adjusting the threshold of SHAP values or the number of features to be displayed.

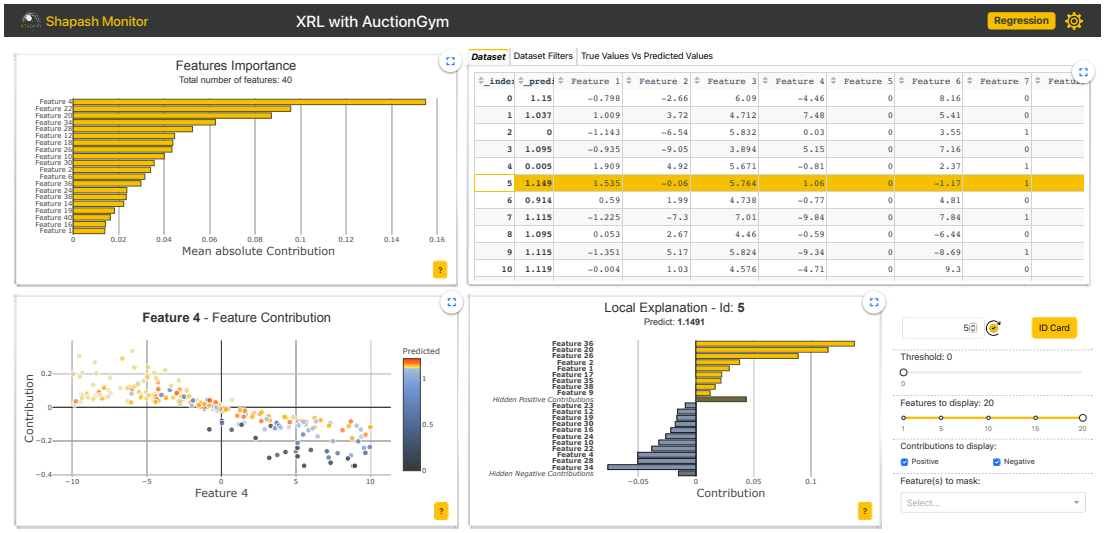


Figure 5.10: A screenshot of our interactive dashboards that displays global feature importance, single feature contribution, explanation dataset and local explanation.

Chapter 6

Conclusion

In this chapter, we summarise this project and answer the research questions proposed in Section 1.2. We also indicate limitations and future work of this project.

6.1 Summary

In this project, we implement a three-step pipeline enabling the application of SHAP on RL bidder models. Based on our experimental results, our answers to the research questions are as follows:

1. **Learning to bid**: By applying consistent bidding configurations with assigned feature weights, SHAP indicates that regardless of the utility estimator employed or the auction type encountered, each bidder model obtains a similar feature ranking. Such ranking is aligned with the order of the absolute values of weights. However, the precise degree of feature importance varies between models. Additionally, after a bidder model converges, its SHAP values also become stable.

2. **SHAP**: The modification of SHAP estimators brings changes in explanations. When using 800 background data to approximate the payout of a coalition, all model-agnostic SHAP value estimators demonstrate similar performance. Notably, the Permutation estimator overall performs the best in our case considering its accuracy, explanation time and scalability. Concerning the reduction of the size of background data for the Kernel estimator, either directly or via k-means, this action compromises its accuracy but leads to shorter explanation time.

3. **Context feature**: When other conditions are kept the same, a feature with a wider range of values (which can result from either a higher feature weight or a higher

variance of the normal distribution) tends to exhibit higher importance, especially when there is a noticeable difference in spread.

6.2 Limitation

We present three limitations of our work. Firstly, due to the lack of real data, we are unable to conduct a comprehensive evaluation with expert insights or delve into in-depth feature analysis. Instead, our validation process relies on synthetic simulation data, which falls short of the complexity and depth that real data would offer. Secondly, due to time constraints, we could not conduct multiple experimental runs as demonstrated by [4], which might introduce higher variance and bias to our results. Lastly, our exploration is confined to a limited number of AuctionGym settings, leaving ample room for future investigation.

6.3 Future Work

This study has shed light on SHAP's potential in explaining bidder behaviours using AuctionGym. To further extend our research, we propose three possible directions. Firstly, introducing real data would provide an opportunity to assess the practical impacts of our methods. If real data is still unavailable, creating more complex simulated data could also be helpful to enhance the validity and generalisability of our findings. Next, there are more AuctionGym configurations to be further explored. [5] recently updates their previous work [4], presents a more detailed analysis of RL bidder policies and the application of AuctionGym. Leveraging these advancements, our study can be extended for a more comprehensive view of SHAP's capabilities and limitations. Finally, developing a model-specific SHAP estimator tailored to our bidder models could further enrich our explanatory capabilities. Such a development can deepen our understanding of how SHAP enhances the explanation of RL bidders' behaviours within the broader context of AuctionGym and beyond.

Bibliography

[1] Liat Antwarg, Ronnie Mindlin Miller, Bracha Shapira, and Lior Rokach. Explaining anomalies detected by autoencoders using Shapley Additive Explanations. *Expert Systems with Applications*, 186:115736, December 2021.

[2] Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. Doubly Robust Policy Evaluation and Optimization. *Statistical Science*, 29(4), November 2014.

[3] Miroslav Dudík, John Langford, and Lihong Li. Doubly Robust Policy Evaluation and Learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 1097–1104, Madison, WI, USA, June 2011. Omnipress.

[4] Olivier Jeunen, Sean Murphy, and Ben Allison. Learning to Bid with AuctionGym. In *Proc. of the AdKDD Workshop at the 28th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, AdKDD '22, 2022.

[5] Olivier Jeunen, Sean Murphy, and Ben Allison. Off-Policy Learning-to-Bid with AuctionGym. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4219–4228, Long Beach CA USA, August 2023. ACM.

[6] Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable Reinforcement Learning via Reward Decomposition. In *IJCAI/ECAI Workshop on explainable artificial intelligence*, 2019.

[7] Scott M. Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 4768–4777, Red Hook, NY, USA, December 2017. Curran Associates Inc.

[8] Rory Mitchell, Joshua Cooper, Eibe Frank, and Geoffrey Holmes. Sampling Permutations for Shapley Value Estimation. *The Journal of Machine Learning Research*, 23(1):43:2082–43:2127, January 2022.

[9] Christoph Molnar. *Interpretable Machine Learning*. Lulu. com, 2020.

[10] Christoph Molnar. *Interpreting Machine Learning Models With SHAP*. Leanpub, August 2023.

[11] Amir Bahador Parsa, Ali Movahedi, Homa Taghipour, Sybil Derrible, and Abolfazl (Kouros) Mohammadian. Toward safer highways, application of XGBoost and SHAP for real-time accident detection and feature analysis. *Accident Analysis & Prevention*, 136:105405, March 2020.

[12] Erika Puiutta and Eric M. S. P. Veith. Explainable Reinforcement Learning: A Survey. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl, editors, *Machine Learning and Knowledge Extraction*, Lecture Notes in Computer Science, pages 77–95, Cham, 2020. Springer International Publishing.

[13] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier, August 2016. arXiv:1602.04938 [cs, stat].

[14] Stefano Giovanni Rizzo, Giovanna Vantini, and Sanjay Chawla. Reinforcement Learning with Explainability for Traffic Signal Control. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3567–3572, Auckland, New Zealand, October 2019. IEEE.

[15] Lloyd Shapley. A value for n-person games. *Contributions to the Theory of Games*, pages 307–317, 1953. Publisher: Princeton University Press.

[16] Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and Interpretable Skill Acquisition in Multi-task Reinforcement Learning, December 2017. arXiv:1712.07294 [cs].

[17] Aleksandrs Slivkins. Introduction to Multi-Armed Bandits, January 2022. arXiv:1904.07272 [cs, stat].

[18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. MIT Press, November 2018. Google-Books-ID: uWV0DwAAQBAJ.

[19] María Vega García and José L. Aznarte. Shapley additive explanations for NO2 forecasting. *Ecological Informatics*, 56:101039, March 2020.

[20] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically Interpretable Reinforcement Learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018.

[21] William Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance*, 16(1):8–37, 1961. Publisher: [American Finance Association, Wiley].

[22] Dong Wang, Sven Thunéll, Ulrika Lindberg, Lili Jiang, Johan Trygg, and Mats Tysklind. Towards better process management in wastewater treatment plants: Process analytics based on SHAP values for tree-based machine learning methods. *Journal of Environmental Management*, 301:113941, January 2022.

Appendix A

Bidder Training Information

This table shows the complete experiment list of settings and time for training bidders.

Observable Features	*Utility Estimator*	*Auction Type*	*Checkpoint*	*Feature Weight*	*Feature Distribution*	*Training Time*
8	DR	FP	FALSE	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*01:22:58*
4	DR	FP	FALSE	**[2, 0, -4, 8]**	All *N(0, 1)*	*01:02:40*
8	**DM**	FP	FALSE	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*01:12:09*
8	**IPS**	FP	FALSE	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*01:32:12*
8	DR	**SP**	FALSE	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*00:50:55*
8	DR	FP	**TRUE**	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*01:03:38*
8	DR	FP	FALSE	**[4, 2, 1, 0, −1, −2, −4, −8]**	All *N(0, 1)*	*01:43:19*
8	DR	FP	FALSE	**[1, 4, -2, 8, -4, -1, 0, 2]**	All *N(0, 1)*	*01:19:09*
8	DR	FP	FALSE	**[1, 1, 1, 1, 1, 1, 1, 1]**	All *N(0, 1)*	*01:06:00*
8	DR	FP	FALSE	**[1, 1, 1, 1, 1, 1, 1, 2]**	All *N(0, 1)*	*01:04:34*
8	DR	FP	FALSE	**[1, 1, 1, 1, 1, 1, 1, 5]**	All *N(0, 1)*	*01:03:52*
8	DR	FP	FALSE	**[1, 1, 1, 1, 1, 1, 1, 10]**	All *N(0, 1)*	*01:15:24*
8	DR	FP	FALSE	**[1, 1, 1, 1, 1, 1, 1, 1]**	**[*N(0, 1)*, *N(0, 2)*, *N(0, 3)*, *N(0, 4)*, *N(0, 5)*, *N(0, 6)*, *N(0, 7)*, *N(0, 8)*]**	*01:35:54*
8	DR	FP	FALSE	**[1, 1, 1, 1, 1, 1, 1, 1]**	**[*N(0, 1)*, *N(0, 2)*, *N(0, 3)*, *N(0, 4)*, *N(0, 5)*, *N(0, 6)*, *N(0, 7)*, *N(0, 1)*]**	*01:29:58*
8	DR	FP	FALSE	**[1, 1, 1, 1, 1, 1, 1, 1]**	**[*N(0, 8)*, *N(0, 7)*, *N(0, 6)*, *N(0, 5)*, *N(0, 4)*, *N(0, 3)*, *N(0, 2)*, *N(0, 1)*]**	*01:34:01*
8	DR	FP	FALSE	**[1, 1, 1, -5, 0, 5, 1, 1]**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class]**	*01:33:29*
8	DR	FP	FALSE	[-4, -2, -1, 0, 1, 2, 4, 8]	**All *U(0, 1)***	*01:20:59*
8	DR	FP	FALSE	**[1, 1, 1, 1, 1, 1, 1, 1]**	**All *U(0, 1)***	*01:10:47*
8	DR	FP	FALSE	**[-1, -1, -1, -1, -1, -1, -1, -1]**	**All *U(0, 1)***	*01:08:07*
16	DR	FP	FALSE	**[1, 1, 1, -5, 0, 5, 1, 1] (2 repetitions)**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class] (2 repetitions)**	*01:30:19*
24	DR	FP	FALSE	**[1, 1, 1, -5, 0, 5, 1, 1] (3 repetitions)**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class] (3 repetitions)**	*01:33:58*
32	DR	FP	FALSE	**[1, 1, 1, -5, 0, 5, 1, 1] (4 repetitions)**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class] (4 repetitions)**	*01:30:45*
40	DR	FP	FALSE	**[1, 1, 1, -5, 0, 5, 1, 1] (5 repetitions)**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class] (5 repetitions)**	*01:28:59*

Appendix B

SHAP Information

This table shows the complete experiment list of settings and explanation time for SHAP applications.

Observable Features	*Utility Estimator*	*Auction Type*	*Checkpoint*	*SHAP Estimator*	*Background Data Size*	*Feature Weight*	*Feature Distribution*	*Explanation Time*
8	DR	FP	FALSE	Permutation	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*04:24:39*
4	DR	FP	FALSE	**Permutation**	800	**[2, 0, -4, 8]**	All *N(0, 1)*	*04:22:18*
4	DR	FP	FALSE	**Sampling**	800	**[2, 0, -4, 8]**	All *N(0, 1)*	*00:18:04*
4	DR	FP	FALSE	**Partition**	800	**[2, 0, -4, 8]**	All *N(0, 1)*	*00:07:20*
4	DR	FP	FALSE	**Exact**	800	**[2, 0, -4, 8]**	All *N(0, 1)*	*00:06:22*
4	DR	FP	FALSE	**Kernel**	800	**[2, 0, -4, 8]**	All *N(0, 1)*	*00:20:45*
8	DR	FP	FALSE	**Sampling**	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*00:18:43*
8	DR	FP	FALSE	**Partition**	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*01:56:48*
8	DR	FP	FALSE	**Exact**	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*01:13:19*
8	DR	FP	FALSE	**Kernel**	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*07:03:42*
8	DR	FP	FALSE	**Kernel**	80	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*00:42:44*
8	DR	FP	FALSE	**Kernel**	80 (k-means)	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*00:55:38*
8	**DM**	FP	FALSE	Permutation	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*04:16:55*
8	**IPS**	FP	FALSE	Permutation	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*04:37:02*
8	DR	**SP**	FALSE	Permutation	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*04:24:35*
8	DR	FP	**TRUE**	**Exact**	800	[-4, -2, -1, 0, 1, 2, 4, 8]	All *N(0, 1)*	*00:43:19 (iteration 0) 01:12:69 (others average)*
8	DR	FP	FALSE	Permutation	800	**[1, 4, -2, 8, -4, -1, 0, 2]**	All *N(0, 1)*	*04:23:14*
8	DR	FP	FALSE	Permutation	800	**[4, 2, 1, 0, −1, −2, −4, −8]**	All *N(0, 1)*	*04:24:57*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, 1, 1, 1, 1, 1]**	All *N(0, 1)*	*04:24:45*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, 1, 1, 1, 1, 2]**	All *N(0, 1)*	*04:29:43*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, 1, 1, 1, 1, 5]**	All *N(0, 1)*	*04:29:40*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, 1, 1, 1, 1, 10]**	All *N(0, 1)*	*04:27:37*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, 1, 1, 1, 1, 1]**	**[*N(0, 1)*, *N(0, 2)*, *N(0, 3)*, *N(0, 4)*, *N(0, 5)*, *N(0, 6)*, *N(0, 7)*, *N(0, 8)*]**	*04:33:50*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, 1, 1, 1, 1, 1]**	**[*N(0, 1)*, *N(0, 2)*, *N(0, 3)*, *N(0, 4)*, *N(0, 5)*, *N(0, 6)*, *N(0, 7)*, *N(0, 1)*]**	*04:30:21*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, 1, 1, 1, 1, 1]**	**[*N(0, 8)*, *N(0, 7)*, *N(0, 6)*, *N(0, 5)*, *N(0, 4)*, *N(0, 3)*, *N(0, 2)*, *N(0, 1)*]**	*04:24:04*
8	DR	FP	FALSE	Permutation	800	[-4, -2, -1, 0, 1, 2, 4, 8]	**All *U(0, 1)***	*04:26:15*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, 1, 1, 1, 1, 1]**	**All *U(0, 1)***	*04:23:15*
8	DR	FP	FALSE	Permutation	800	**[-1, -1, -1, -1, -1, -1, -1, -1]**	**All *U(0, 1)***	*05:12:58*
8	DR	FP	FALSE	Permutation	800	**[1, 1, 1, -5, 0, 5, 1, 1]**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class]**	*04:00:32*
16	DR	FP	FALSE	Permutation	800	**[1, 1, 1, -5, 0, 5, 1, 1] (2 repetitions)**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class] (2 repetitions)**	*04:14:02*
24	DR	FP	FALSE	Permutation	800	**[1, 1, 1, -5, 0, 5, 1, 1] (3 repetitions)**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class] (3 repetitions)**	*04:09:10*
32	DR	FP	FALSE	Permutation	800	**[1, 1, 1, -5, 0, 5, 1, 1] (4 repetitions)**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class] (4 repetitions)**	*04:02:00*
40	DR	FP	FALSE	Permutation	800	**[1, 1, 1, -5, 0, 5, 1, 1] (5 repetitions)**	**[*N(0, 1)*, *N(0, 5)*, *N(5, 1)*, *U(-2, 2)*, *U(-2, 2)*, *U(-2, 2)*, Binary, 5-Class] (5 repetitions)**	*04:42:12*