Use of sample-splitting and cross-fitting techniques to mitigate the risks of double-dipping in behaviour-agnostic reinforcement learning

Comparative Analysis

Yaren Aslan[1]
Supervisor(s): Frans Oliehoek[1]**, Stephan Bongers**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Yaren Aslan
Final project course: CSE3000 Research Project
Thesis committee: Frans Oliehoek, Stephan Bongers, Catholijn Jonker

Abstract

This paper addresses the issue of double-dipping in off-policy evaluation (OPE) in behaviour-agnostic reinforcement learning, where the same dataset is used for both training and estimation, leading to overfitting and inflated performance metrics especially for variance. We introduce SplitDICE, which incorporates sample-splitting and cross-fitting techniques to mitigate double-dipping effects in the DICE family of estimators. Focusing specifically on 2-fold and 5-fold cross-fitting strategies, the original off-policy dataset is partitioned with random-split to get separate training and evaluation datasets. Experimental results demonstrate that SplitDICE, particularly with 5-fold cross-fitting, significantly reduces error, bias, and variance compared to naive DICE implementations, providing a more doubly-robust solution for behavior-agnostic OPE.

Key words: DICE, off-policy evaluation, behaviour-agnostic, sample-splitting, cross-fitting, 2-fold, 5-fold, variance, double-dipping, overfitting

1 Introduction

In reinforcement learning, for each state of the environment, the agent takes a certain action based on the policy, retrieves a reward based on this action and transitions into a new state. The overarching goal of reinforcement learning methodologies is to acquire an optimal policy which maximizes the long-term cumulative rewards [18]. In the context of policy evaluation, *off-policy evaluation* (OPE) refers to the setting where the agent estimates the value of a target policy by referring only to a dataset of experience previously collected by other policies in the said environment [17]. The objective of OPE is to estimate the expected cumulative (discounted) reward that a new policy (namely, the target policy) would achieve if deployed in the environment. This is important for understanding how well the new policy might perform before actually deploying it [21]. However, it is essential to note that this logged experience is collected by potentially multiple and possibly unknown behavior policies which requires the embodiment of the concept known as *behavior-agnostic*.

Behaviour-agnostic OPE specifically denotes an approach where the learning algorithm does not make any assumptions about the behavior policy that generated the dataset [23]. Recent advancements in addressing the unknown bounds of behaviour-agnostic OPE have led to the development of various estimators collectively referred to as the "DICE" family which stands for *DIstribution Correction Estimation* [13], [22], [26], [23]. These estimators are used to display the ratio between the propensity of the target policy to visit distinct state-action pairs compared to their occurrence likelihood in the off-policy data. DICE-based estimators exercise "a single marginal ratio to re-weight the rewards for each state-action pair", consequently achieving a relatively low variance for the estimate values [3].

It is worth highlighting that policy evaluation in general requires tedious consideration of the model complexity to prevent pitfalls such as overfitting or underfitting the data. Overfitting occurs when a modeling approach mirrors every underlying pattern of the data, resulting in high accuracy when applied to the original dataset. However, it fails to generalize well to unseen and foreign datasets. This is mainly caused by data with many features and/or an excessively large neural network. In this context, *double-dipping* refers to the practice of overfitting a model by building (training) and evaluating (estimating) it on the same dataset [2]. This then causes misleadingly high performance metrics with artificially inflated statistical significance since the model is being evaluated on the data it was trained on, leading to "circular logic".

In the current implementation of the neural estimators, DICE algorithms employ the same dataset to conduct the training process and the estimation of the target policy. This practice is the driving factor for double-dipping. Another pressing issue related to the double-dipping behaviour is that the DICE family employs primal and dual regularization techniques as a regression method to tackle high variance and therefore to avoid overfitting [23]. The main concern regarding this choice is that this introduces a trade-off, as regularization can bias the parameter of interest with the aim of mitigating overfitting.

This paper aims to answer the following research question: *"How does the use of sample-splitting and cross-fitting techniques mitigate the effects of 'double-dipping' in behavior agnostic reinforcement* *learning?"*. To break it down more clearly, the aim of this research is to adopt commonly used techniques in double/debiased machine learning (DML), namely *sample-splitting* and *cross-fitting*, for the DICE estimators and analyze how effective they are in mitigating the risks associated with double-dipping. DML's "double" behaviour comes from simultaneously estimating two predictive models: one for the primary target of interest and another for auxiliary outcomes [7]. When compared against naive ML estimators, their fast rates of convergence and robust behaviour with respect to a broader class of probability distributions makes the objective of this research even more striking [7]. In this research, we introduce SplitDICE, which incorporates sample-splitting and cross-fitting into the existing implementation of the DICE estimator.

1.1 Structure

The structure of this paper is as follows. In section 2, we explain previous work relevant to the topic of research from different research papers whether they are in the context of double machine learning methods and/or DICE estimators for behaviour agnostic off-policy evaluation. We list the points of improvement and how this research contributes to mitigate such knowledge gap. In section 3, we explain the methods in hand in technical terms with necessary mathematical observations and calculations regarding the techniques of sample-splitting and cross-fitting.

After setting up a clear background for the integration of OPE, DICE and DML, we then go into section 4 which demonstrates the experimental setup with regards to data generation, environment specifications and choice of configurations. Next, section 5 presents the results of the three estimator models and discusses the performance on a predetermined category of metrics, these being convergence, relative and mean-squared error, variance and bias. We also provide a statistical analysis on the significance of results for the relative error. After reflecting on the achieved results and discussing the limitations of the bounds of the research in section 6, we touch upon the ethical aspects considered during the process and the societal impact of the research in section 7. Finally, we conclude the paper by summarizing that 5-fold cross-fit DICE estimator has a notable improvement over the naive implementation of BestDICE achieving lower rates of error, bias and most importantly variance.

2 Related Work

The study that sustains the backbone of this research is that of Yang et al. on off-policy evaluation via the regularized Lagrangian [23]. By showing that the previous DICE formulations are all equivalent to regularized Lagrangians of the same linear program (LP), they specifically investigate the dual form, namely $d{-}LP$, for off-policy evaluation. They then identify a list of choices with regards to translating this formulation into a stable minimax optimization problem. These choices consist of the specification of redundant constrains and the regularization of primal and dual variables. This optimization unifies all the variants of the DICE estimators under one framework by selecting an appropriate regularization configuration. These are listed as DualDICE [13], GenDICE [25], GradientDICE [26], DR-MWQL and MWL [22], LSTDQ [11], Algae $Q{-}LP$ [14] and BestDICE with it being the variant that achieves the best performance out of all. This said, the estimator introduced by our paper builds upon BestDICE as well.

There has also been studies in developing doubly-robust estimators inspired from the methods used in machine learning. In a recent study of Kallus and Uehara, with the objective of achieving true off-policy evaluation in time-invariant Markov processes, they develop a new estimator based on double reinforcement learning [10]. They design an estimator that employs both estimated stationary density ratios and q-functions. This design maintains efficiency even when both components are estimated slowly and ensures consistency when either component is estimated accurately. The strategy used for the double reinforcement learning also sets the initiative behind this research. In their comparative analysis of the estimator architecture, one of the estimators used for comparison is DualDICE which is a part of the DICE family as mentioned before. This variant uses dual regularization parameters similar to BestDICE which makes their research especially valuable to give close attention to the nature of dual regularized DICE estimators. However, their study lacks a comparative analysis of different k-fold cross-fitting proportions, which our paper aims to address.

In Chernozhukov et al.'s study for double machine learning for treatment and causal parameters, they dive into the details of Double ML and the methodology behind it in order to improve the performance of naive ML estimators [6]. Their objective differs from previously mentioned studies since it does not encompass DICE estimators or off-policy evaluation in its scope. By setting up a background for why and how modern supervised statistical/machine learning fails to provide accurate estimators of causal parameters, they provide reasoning behind the poor performance of naive estimators one of which being regularization bias. They come to point out that while regularization helps with stability and convergence, it introduces a bias into the estimator. This is especially relevant to the current research since DICE estimators use techniques of primal and/or dual regularization in order to avoid overfitting. It is essential to note that the main objective of this study is to demonstrate the orthogonalization of double machine learning, driven by the need to counteract bias introduced by non-orthogonal ML estimators. However, our study is specifically motivated by addressing the phenomenon of double-dipping, which naturally focuses on reducing variance. Since within the DICE family, some estimators such as BestDICE have already been tailored to counteract the negative effects of certain regularization choices on bias, our study prioritizes variance as a more valuable performance metric than bias.

In Jacob's study for cross-fitting and averaging for machine learning estimation of heterogeneous treatment effects, they investigate the performance of twelve different estimators on four different meta-learners [9]. In the aim of finding a correlation between the learning process of the meta learners and the procedure of chosen doubly robust machine learning techniques, these estimators have varying qualities regarding the types of sample-splitting, cross-fitting and averaging procedures used. It is again worth to mention that this research is restrained by the bounds of machine learning and does not concern off-policy evaluation. However, the intricacies they provide regarding the procedure used for conducting research with varying categories of estimators and performance metrics are as much valuable for the research at hand to establish a fair comparative analysis. Although our study does not share the same depth for the number of techniques employed, we use the same proportions as they do in their experiment for the cross-fit estimator, these being 2-fold and 5-fold cross-fitting. It is essential to point out that the results achieved by the study are more significant than those of ours, primarily because the regularization choices within the DICE framework can potentially decrease the efficiency of double machine learning techniques. Thus, combining behaviour-agnostic off-policy evaluation with double machine learning addresses distinct challenges separate from those of Jacob's.

3 Background and Methodology

In this section, we start with background information that forms the basis of this research which consists of reinforcement learning and policy evaluation. We then explain the motivation behind off-policy evaluation and a family of OPE estimators known as DICE which are designed to operate in behaviour-agnostic settings. In subsection 3.1 and subsection 3.2, we introduce the equations and the formulations provided by Yang et al. [23]. Next, we present the problem of double-dipping which forms the primary focus of this research and explain the proposed solution with regards to sample-splitting. The last subsection introduces the technicalities behind SplitDICE which is the estimator built to address the research question. In subsection 3.3 and subsection 3.4, with regards to sample-splitting and cross-fitting, we introduce the concepts provided by Jacob [9] and equations provided by Chernozhukov et al. [6].

3.1 Policy Evaluation in Reinforcement Learning

In a reinforcement learning (RL) setting, we consider an infinite-horizon Markov Decision Process (MDP) [18]. This process is defined by the tuple $M = \langle S, A, R, T, \mu_0, \gamma \rangle$. This consists of a state space (S), action space (A), reward function (R), transition probability function (T), initial state distribution (μ_0), and a discount factor $(\gamma \in [0, 1])$. In RL, a policy (π) defines the agent's strategy or behavior in an environment. It is a mapping from states to actions, denoted as $\pi(a \mid s)$, which specifies the probability distribution over actions A that the agent selects when in state s at step $t \geq 0$. The environment yields a scalar reward $r_t = R(s_t, a_t)$ and then transitions to a new state $s_{t+1} \sim T(s_t, a_t)$.

We define the value of a policy (π) by the normalized expected per-step reward it receives:

$$\rho(\pi) := (1-\gamma)\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 \sim \mu_0, \forall t, a_t \sim \pi(s_t), s_{t+1} \sim T(s_t, a_t)\right] \quad (1)$$

In RL, the behavior policy denoted as μ refers to the policy that the agent is currently following to interact with the environment. It is a distinct concept from the target policy π which the agent seeks to optimize and improve during its learning (training) process. Therefore, naturally, in the context of policy evaluation, the policy being evaluated is the target policy. In a more concrete way, with policy evaluation, we aim to evaluate how effectively the training of the behavior policy aligns with the target policy. The value of a target policy can be expressed equivalently in two manners using different functions:

$$\begin{aligned} \rho(\pi) &= (1-\gamma) \cdot \mathbb{E}_{a_0 \sim \pi(s_0)} \left[Q^\pi(s_0, a_0) \right] \\ &= \mathbb{E}_{(s,a) \sim d^\pi} \left[R(s,a) \right], \quad s_0 \sim \mu_0 \end{aligned} \tag{2}$$

where Q^π stands for the state-action values and d^π represents the visitations of π. More specifically,

- the function Q^π represents the Q-values associated with policy π. These values are retrieved by mapping state-action pairs (s,a) to the expected value under policy π when executed in the environment, starting from state s and taking action a.
- the function d^π represents the on-policy distribution of π. Keep in mind that this distribution is normalized over all state-action pairs (s,a) to indicate the probability of encountering (s,a) under π. It is also averaged over the entire duration with γ-discounting strategy.

3.2 Using DICE Estimators to Achieve Behaviour-agnostic Off-policy Evaluation

Off-policy evaluation (OPE) is a crucial technique in RL that aims to estimate the value of a policy $\rho(\pi)$ by using only a fixed dataset of experiences, without further interactions with the environment. This aspect is particularly important in settings where interacting with the real environment is costly, risky or even infeasible [21]. Therefore, OPE makes it much more practical to improve policies in a controlled, off-line manner before any real-world deployment, thereby mitigating possible risks and reducing high costs.

In this context, we assume that we have access to a finite dataset $D = \{(s_0^{(i)}, s^{(i)}, a^{(i)}, r^{(i)}, s'^{(i)})\}_{i=1}^N$, where $s_0^{(i)} \sim \mu_0$, $(s^{(i)}, a^{(i)}) \sim d_D$ are samples from some unknown distribution d_D. Estimators using DICE methods employ the following expression to derive an estimate average per-step reward value of the target policy:

$$\rho(\pi) = \mathbb{E}_{(s,a,r) \sim d_D}[\zeta^*(s,a) \cdot r] \text{ where } \zeta^*(s,a) = \frac{d^\pi(s,a)}{d_D(s,a)} \tag{3}$$

where $(s,a,r) \sim d_D$ is used as an abbreviated form of $(s,a) \sim d_D, r = R(s,a), s' \sim T(s,a)$. In other words, this simulates sampling from the dataset D when using a finite number of samples. Most importantly, $\zeta^*(s,a)$ stands for the distribution correction ratio. The main objective of the DICE estimators is that they aim to approximate this correction ratio without requiring the knowledge of d^π or d_D. This is where the concept of behaviour-agnostic arises from.

3.3 Double-dipping and Sample Splitting with k-fold Cross-fitting

Double-dipping, in the context of machine learning, is a term for overfitting a model through both building and evaluating the model on the same dataset [2]. Consequently, while the model may exhibit low error rates within the sample, it will have high variance and poor generalizability. We observe in the neural network of the DICE estimators that the same dataset is used for training the estimator and estimating the value of the target policy. Keep in mind that in order to introduce more stability into the optimization, DICE employs mechanisms to apply regularization techniques and redundant constraints [23]. Regularization works by adding a penalty term to the loss function used to train the model which increases the loss for larger model coefficients, thereby discouraging the model from fitting too closely to the training data [5]. However, since now the model is constrained to be simpler, this can prevent it from capturing the true underlying patterns in the data, and consequently increasing bias. Therefore, the main objective of this research is to see whether the adaptation of a fundamental strategy from the methods of double/de-biased machine learning known as sample-splitting is effective in reducing variance while keeping the estimator as (un)biased as before or even less biased.

Allowing for the unbiased assessment of model performance, the technique of sample-splitting mainly involves partitioning the original dataset into distinct subsets, in technical terms this is known as K folds. For example, consider a 50:50 sample splitting strategy where the data is split into two equal folds, called an auxiliary sample (A) and the main sample (M) for the estimation of the parameter of interest. In the context of DICE estimators, training is regards to the Q-value functions and visitation densities whereas the parameter of interest refers to the estimate value of the target policy, $\rho(\pi)$. It is important to keep in mind that sample splitting reduces the available amount of data used for both training of the estimator and the estimation of the value of the target policy. As pointed out by Jacob, this leads to “a loss in efficiency and statistical power in finite samples” [9]. This is where the concept of cross-fitting comes along. To make use of the full sample and to restore efficiency, the roles of the samples are switched thereby using sample M for training and sample A for estimation.

As suggested by Chernozhukov et al., we assume that a random sample $(W_i)_N^{i=1}$ from the distribution of W is available for evaluation and training [6]. For two fittings, we build two prediction models based on the roles of the samples, these samples being I and I^c. The true value μ_0 of the nuisance parameter μ is estimated by $\hat{\mu}_0(I^c)$ using the training sample $(W_i)_{i \in I^c}$. The true value θ_0 of the target parameter θ is estimated by the estimator $\check{\theta}_0(I, I^c)$ using the evaluation sample $(W_i)_{i \in I}$. Here, the nuisance parameter refers to the distribution correction ratio and the target parameter refers to the average per-step reward value, as mentioned before in Equation 3. In the aggregation of the results, the calculation of the joint estimate value is given by:

$$\tilde{\theta}_0 = \frac{\check{\theta}_0(I, I^c)}{2} + \frac{\check{\theta}_0(I^c, I)}{2} \tag{4}$$

where I and I^c represent a random 50-50 split of the dataset. Over a batched number of episodes $\{1, \ldots, N\}$, the size of I is n, the size of I^c is also n, and the total sample size is $N = 2n$. We then construct an estimator $\check{\theta}_0(I, I^c)$ that employs the nuisance parameter estimator $\hat{\mu}_0(I^c)$ where I is used for evaluation dataset and I^c is used for training dataset. This can be interpreted as building the prediction model (neural network) of the estimator on the training dataset, I^c. Then, we reverse the roles of I and I^c and construct an estimator $\check{\theta}_0(I^c, I)$ that employs the nuisance parameter estimator $\hat{\mu}_0(I)$ where I^c is used for evaluation dataset and I is used for training dataset. We are now building the prediction model (neural network) of the estimator on the dataset I that was used as evaluation dataset in the first fitting. The results of the two estimators are then aggregated into a final estimate value, $\tilde{\theta}_0$ by taking the average.

For a fold number that is larger than 2, the same process can be followed by assigning $(100 - \frac{100}{K})\%$ of the observations to sample A and $(\frac{100}{K})\%$ to sample M and iterating till every fold is used for estimation. The calculation for the joint estimate value is then given by:

$$\tilde{\theta}_0 = \frac{1}{K} \sum_{k=1}^{K} \check{\theta}_0(I_k, I_k^c) \tag{5}$$

which involves a K-fold random split of the entire sample with $k = 1, \ldots, K$. Then, for each fold we construct an estimator $\check{\theta}_0(I_k, I_k^c)$ that employs the nuisance parameter estimator $\hat{\mu}_0(I_k^c)$ where I_k is used for evaluation dataset and I_k^c is used for training dataset. In this case, each training set $I_k^c = \bigcup_{m \neq k} I_m$ has size $N \cdot \left(\frac{K-1}{K}\right)$ and each evaluation set I_k has size $\frac{N}{K}$ and the total sample size is N. As an example, we present the diagram in Figure 1 for a visual representation of the splitting process that applies 5-fold cross-fitting. Each box colored with blue represents the evaluation dataset of the corresponding iteration whereas the grey colored boxes are merged altogether to represent the training dataset.

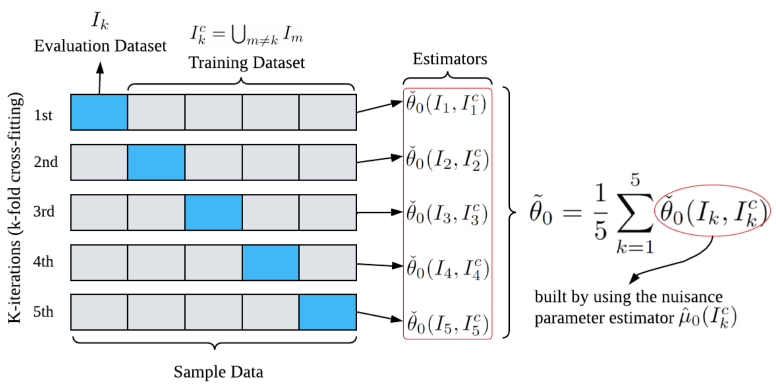

I_k
Evaluation Dataset
$I_k^c = \bigcup_{m \neq k} I_m$
Training Dataset
Estimators
K-iterations (k-fold cross-fitting)
1st
2nd
3rd
4th
5th
$\check{\theta}_0(I_1, I_1^c)$
$\check{\theta}_0(I_2, I_2^c)$
$\check{\theta}_0(I_3, I_3^c)$
$\check{\theta}_0(I_4, I_4^c)$
$\check{\theta}_0(I_5, I_5^c)$
$\tilde{\theta}_0 = \frac{1}{5}\sum_{k=1}^{5}\check{\theta}_0(I_k, I_k^c)$
built by using the nuisance parameter estimator $\hat{\mu}_0(I_k^c)$
Sample Data

Figure 1: **Model representation for a 5-fold cross-fit estimator**. The sample data is split into 5 folds. During each iteration of cross-fitting, one fold, I_k, serves as the evaluation dataset, while the remaining folds are merged into a training dataset, I_k^c. In each iteration, a new estimator is built using the nuisance parameter estimator $\hat{\mu}_0(I_k^c)$. The final estimate value $\tilde{\theta}_0$ is obtained after running all the iterations and averaging out the results retrieved from each estimator.

3.4 DICE using Double/De-biased ML: SplitDICE

We now introduce SplitDICE which implements k-fold cross-fitting for the DICE estimators. The pseudo-code provided by Algorithm 1 demonstrates the main concept of sample-splitting technique. It is important to note that the splitting of the dataset is done by keeping the episode structure intact, this means the original off-policy data is split by episodes rather than steps, to keep the sequential logic of the behaviour policy same as the original. For both folding proportions, the iteration continues until each subset is used to form a dataset for estimation whereas the leftover subsets are merged into one training set. To ease the process of splitting, all the possible combinations of splitting are done via the method provided in Algorithm 1 and appended to the final list of train-eval dataset pairs.

The pseudo-code provided by Algorithm 2 and Algorithm 3 shows the implementation of cross-fitting and the training process of the DICE estimator respectively. Before running the folds, Algorithm 2 calls `random_split()` as provided by Algorithm 1 to retrieve all the train-eval dataset pairs. The calculation of the final joint estimate value is performed at the end of the last fold. This occurs after the estimator has been trained for a number of steps with the training dataset for all the K folds and the list of estimate values is retrieved per fold. These values, stored in `joint_estimates`, are calculated over the full batch of the evaluation dataset at every 100 step intervals of the training process. Keep in mind that the fold number (K) is set as a flag, it is provided as an input in the command line to run the file and initialized as a global variable.

Algorithm 1 Sample-splitting of the dataset with fold number K

```
1:  procedure RANDOM_SPLIT(dataset $\mathcal{D}$, training ratio $\alpha$)
2:      Retrieve total number of samples $N$ from $\mathcal{D}$
3:      Calculate number of evaluation samples $n_1 \leftarrow \alpha \times N$
4:      Calculate number of training samples $n_2 \leftarrow N - n_1$
5:      Retrieve all episodes from the dataset as a list
6:      Shuffle the episodes at random                          ▷ to account for a fully random split
7:      Initialize an empty list, $\mathcal{F}$                  ▷ to store pairs of train-eval datasets
8:      Calculate the size of each fold (subset) $f \leftarrow \frac{N}{K}$
9:      for $k \leftarrow 0$ to $K-1$ do
10:         Determine start index $s \leftarrow k \times f$
11:         Determine end index $e \leftarrow (k+1) \times f$ if $k < K-1$ else $N$
12:         Initialize a new off-policy dataset $\mathcal{D}_{eval}$ with capacity $n_1$     ▷ evaluation sample, $I_k$
13:         Initialize a new off-policy dataset $\mathcal{D}_{train}$ with capacity $n_2$    ▷ training sample, $I_k^c$
14:         Add episodes from $[s : e]$ to $\mathcal{D}_{eval}$
15:         Add remaining episodes to $\mathcal{D}_{train}$
16:         Append the pair $(\mathcal{D}_{eval}, \mathcal{D}_{train})$ to $\mathcal{F}$
17:     end for
18:     return $\mathcal{F}$
19: end procedure
```

Algorithm 2 k-fold cross-fitting with fold number k

```
1: procedure RUN_CROSS_FITTING
2:     Load the original dataset $\mathcal{D}$ from the directory
3:     Set training ratio $\alpha \leftarrow \frac{1}{k}$
4:     Retrieve a list of all the fold pairs $\mathcal{F}$ by calling RANDOM_SPLIT($\mathcal{D}$, $\alpha$)
5:     Initialize an empty list, $\mathcal{J}$                    ▷ to store estimate values received per fold
6:     for $k \leftarrow 0$ to $K-1$ do
7:         Split into training dataset $\mathcal{D}_{eval}$ and evaluation dataset $\mathcal{D}_{train}$ using $\mathcal{F}[i]$
8:         Build a DICE estimator $\check{\theta}_0(\mathcal{D}_{eval}, \mathcal{D}_{train})$   ▷ using the nuisance parameter estimator $\hat{\mu}_0(\mathcal{D}_{train})$
9:         Update $\mathcal{J}$ by calling RUN_TRAINING_AND_ESTIMATION($\check{\theta}_0$, $\mathcal{D}_{train}$, $\mathcal{D}_{eval}$, $\mathcal{J}$, $k+1$)
10:    end for
11:    return $\mathcal{J}$
12: end procedure
```

Algorithm 3 Run training and estimation per each fold with fold number K

```
1: procedure RUN_TRAINING_AND_ESTIMATION(estimator $\check{\theta}_0$, training dataset $\mathcal{D}_{train}$, evaluation dataset
   $\mathcal{D}_{eval}$, joint estimates $\mathcal{J}$, fold index $k_{index}$)
2:     Retrieve the target dataset, $\mathcal{D}_{target}$
3:     Initialize an empty list, $\mathcal{R}$                    ▷ to store estimate values received over training
4:     for each step from 0 to 10000 do
5:         Retrieve a batch of transitions $\mathcal{T}$ from $\mathcal{D}_{train}$
6:         Retrieve a batch of initial steps $\mathcal{S}$ from $\mathcal{D}_{train}$ and preprocess initial steps batch
7:         Perform a training step for $\check{\theta}_0$ using $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{D}_{target}$
8:         if step is a multiple of 100 or step is the last step then
9:             Estimate average per-step reward $r$ using $\mathcal{D}_{eval}$ and $\mathcal{D}_{target}$        ▷ via Equation 3
10:            Append estimate $r$ to $\mathcal{R}$
11:            if step is the last step then
12:                Update $\mathcal{J}$ by calling CALCULATE_JOINT_ESTIMATE($\mathcal{R}$, $\mathcal{J}$, $k_{index}$)
13:            end if
14:        end if
15:    end for
16:    return $\mathcal{R}$
17: end procedure
18:
19: procedure CALCULATE_JOINT_ESTIMATE(running estimates $\mathcal{R}$, joint estimates $\mathcal{J}$, fold index $k_{index}$)
20:    Append $\mathcal{R}$ to $\mathcal{J}$
21:    if $k_{index}$ equals $K$ then
22:        Compute the mean $\tilde{\theta}_0$ of all the folds $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k$ in $\mathcal{J}$          ▷ via Equation 5
23:        Log the results for $\tilde{\theta}_0$
24:    end if
25:    return $\mathcal{J}$
26: end procedure
```

4 Experimental Setup

For the purposes of this research, Google's DICE-RL codebase was forked for own use [24], it is made publicly available on the GitHub platform [1].

The datasets used for the experiment were generated using OpenAI Gym which is an open source Python library and a standard API for reinforcement learning. It provides simulated training environments to train as well as test reinforcement learning agents. The environment simulated to create datasets for the experiment is `Frozenlake-v0` which features discrete action and observation space. In the FrozenLake environment, the agent navigates a grid world represented as a frozen lake, encountering holes and frozen surface (which leads to failure with the reward of 0) and a goal (which leads to success with the reward of +1). The visual representation of this environment can be seen via Figure 2.

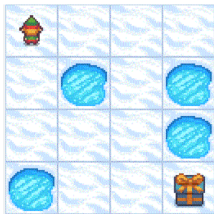


Figure 2: Grid view

This process can be reproduced by running the following command for all the defined seed numbers:

```
for alpha in {0.0, 1.0}; python3 scripts/create_dataset.py
   --save_dir=./tests/testdata --load_dir=./tests/testdata
   --env_name=frozenlake --num_trajectory=200 --max_trajectory_length=100
   --alpha={alpha} --seed={seed} --tabular_obs=0
```

where $\alpha = 0.0$ refers to the dataset for the behaviour policy and $\alpha = 1.0$ refers to the dataset for the target policy. The aim is to estimate the average per-step reward value of the target policy (also referred as cumulative normalized expected reward) using the behaviour policy. In addition to the environment specifications, as given in the default experimental setup of Nachum et al., 20 datasets are created with 20 unique seed numbers (from 0 to 19, inclusive) for generalization purposes [13]. The number of trajectories and the maximum length of each trajectory are set to 200 and 100 respectively for all replications.

For all the datasets separately, the estimator is trained for 10000 steps and the value of the target policy is estimated at intervals of 100 training steps. Initially set as default at 100000 training steps with estimations made every 500 steps, this configuration was adjusted due to the Frozenlake environment's faster convergence, necessitating fewer training steps for accurate estimator performance. This process can be reproduced by running the following command for all the defined seed numbers. Keep in mind this is for running SplitDICE and the fold number is adjusted accordingly depending on the choice of k-fold strategy. For the experiment at hand, we use SplitDICE with 2-fold and 5-fold cross-fitting:

```
python3 scripts/run_neural_dice_split.py
   --save_dir=./tests/testdata --load_dir=./tests/testdata
   --env_name=frozenlake --num_trajectory=200 --max_trajectory_length=100
   --alpha=0.0 --seed={seed} --tabular_obs=0 --fold_number={k-fold}
```

For the estimator configurations, we use the default choice of parameters as provided in the codebase. More specifically, from the unified framework of DICE estimators, the configurations used for the experiment are those belonging to BestDICE, this consists of the following arguments which can be attached to the command given above if deemed necessary:

```
--primal_regularizer=0. --dual_regularizer=1. --zero_reward=0
--norm_regularizer=1. --zeta_pos=1
```

5 Results

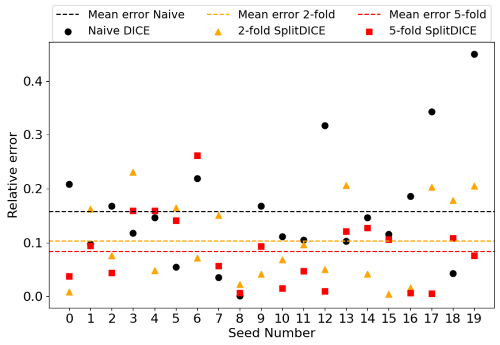
In this part of the paper, we systematically present the results of the experiment and analyze what they indicate for the objectives of the research. Separated into three subsections, we start with 5.1 which exhibits results for relative error calculations obtained per seed for all the considered estimator models with a statistical analysis on significance. Then, 5.2 provides further analysis on other performance metrics measured as single value generalized from all the 20 seeds. Finally, 5.3 demonstrates levels of convergence to the ground truth and a final remark on the most significant metric of the research, which is variance.

5.1 Error Calculation and Comparison

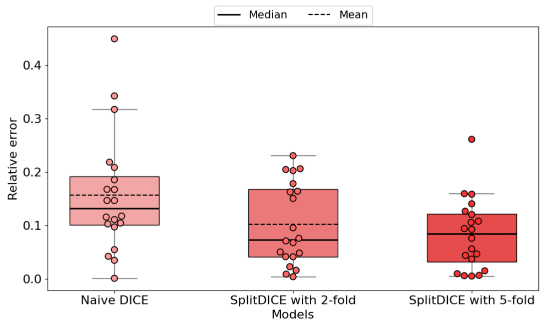
In order to evaluate the estimator's error, (absolute) relative error is chosen as a metric. The motivation behind this choice was that relative error normalizes the absolute error by the true value. This provides a dimensionless measure, which is particularly useful when comparing errors across different scales. This is indeed the case for the experiment in hand since the datasets created with different seeds may have different ground truth values for the average per-step reward of the target policy. This makes it evident that the significance of an error is not just in its absolute magnitude but in how it compares to the true value.

The results shown via Figure 3a demonstrate the relative error for each scenario at a given seed number starting from 0 to 19 (inclusive). The comparisons are made between the three values plotted per seed since datasets with different seeds have different ground truths. The error is calculated by comparing the value of ground truth (retrieved from the target policy) against the final estimated value for average per-step reward (retrieved from the trained behaviour policy).

For a clear comparison of distribution of points, Figure 3b provides a categorical whisker diagram for these error values. It indicates that DICE estimators used with 2-fold and 5-fold cross-fitting exhibit lower rates of error mostly in the spread of the central portion of the data. As seen from shorter whisker lengths, the variance in error values exhibits notable reduction with SplitDICE, particularly when applied with a 5-fold cross-fitting. Although 2-fold SplitDICE is the only model without any outliers, the error values are concentrated towards the edges of the box on the 1st and 3rd quartiles, indicating a skewness in the distribution. Another important finding is that the median line overlaps the mean for 5-fold cross-fitting, this implies that the data is not skewed heavily in one direction and that there are no significant outliers pulling the mean away from the median. This is noteworthy to mention since it confirms that higher-fold cross-fitting provides better estimates due to more thorough validation, demonstrating its validity [6].



(a) The results are categorized by the considered estimator models, with the mean relative error displayed for each as a general summary of all the data points.



(b) The results are categorized by the considered estimator models, with the (non-outlier) data points displayed as a randomized swarm to avoid overlaps.

Figure 3: **Scatter (a) and box-whisker (b) plots showing the relative error between the final estimated average per-step reward value and the ground truth.** The results are obtained for all the 20 seeds from 0 to 19, inclusive.

Before moving on with any statistical analysis on significance, we conducted Anderson-Darling and Shapiro-Wilk tests to determine whether the data is normally distributed and thereby decide whether to continue with parametric or non-parametric approaches. Since the results of the test determined at least one of the datasets to be not normally-distributed, non-parametric approaches were considered and therefore Kruskal-Wallis test was determined suitable. This analysis aim to compare different estimators' performance and determine whether there exists statistically significant disparities in the results of the relative error values. The results for Kruskal-Wallis test-statistic and p-value are reported as 5.73377 and 0.05688 respectively, with the alpha value set to 0.1 (i.e. significance level of 10%). A higher test-statistic suggests a greater likelihood of significant differences among the groups. The obtained p-value indicates that there is a 5.688% chance of reaching the observed test-statistic value if the null hypothesis were true (i.e. there are no significant differences between groups). However, given that the p-value is less than the predetermined significance level, the null hypothesis is rejected. Therefore, we infer that at least one of the estimator categories exhibits a mean that is significantly different from the others.

However, since Kruskal-Wallis is an extension of Mann-Whitney U test for three or more categorical and independent groups, we also conducted Mann-Whitney U test for post-hoc pairwise comparisons to identify which group(s) contributes to the significance. The results of this test are reported via Table 1. From all the pairwise comparisons, the only significant result comes from Naive versus 5-fold. With the adjusted p-value surpassing the significance level and a considerably high U-value, the difference in the median values of of the two estimator models in terms of rank sums shows a significant disparity. Keep in mind that if the significance level was set to 5%, the results would show no statistically significant difference for any of the pairwise comparisons.

Table 1: Mann-Whitney U Test Results for Relative Error Comparison

Group 1	**Group 2**	**Test-statistic (U-value)**	**p-value**	**Adj. p-value after Holm correction**	**Reject**
2-fold SplitDICE	5-fold SplitDICE	230.0	0.42488	0.42488	False
2-fold SplitDICE	Naive DICE	258.0	0.11986	0.23971	False
5-fold SplitDICE	Naive DICE	286.0	0.02073	0.06220	True

5.2 Evaluation of Performance Metrics

Additionally, other metrics such as mean-squared error, bias and variance have been calculated from the average per-step reward values of 2-fold and 5-fold cross-fit estimators and a naive single estimator. Keep in mind the final values were obtained after having all the aforementioned estimators trained for a given number of training steps, in this case this number was set to 10000 for all. As suggested by Jacob, the formula used for these performance metrics are provided via Equation 6 for mean-squared error, bias and variance respectively [6]. Keep in mind the formula for variance calculation is adjusted to account for the subtle fluctuations in the ground truth values such that the the outcome is relative to the truth.

$$\text{MSE} = \frac{1}{S}\sum_{s=0}^{S}\left[\tilde{\theta}_0^s - \theta_0^s\right]^2, \quad \text{Bias} = \left|\frac{1}{S}\sum_{s=0}^{S}\tilde{\theta}_0^s - \theta_0^s\right|, \quad \text{Var} = \frac{1}{S}\sum_{s=0}^{S}\left[\frac{\tilde{\theta}_0^s}{\theta_0^s} - \frac{\overline{\tilde{\theta}_0^s}}{\theta_0^s}\right]^2 \tag{6}$$

where θ_0^s refers to ground truth obtained from the target policy and $\tilde{\theta}_0^s$ refers to the estimate value obtained from the model estimators, in the case of cross-fit estimators, this value is the average over K folds. s here is for the seed number since per each seed, the calculated final estimate value and the ground truth might differ. These values have been provided in Table 2.

We can say that the results advocate for the advantages of using cross-fitting methods over the naive estimator. The naive estimator has the highest MSE, marking less accuracy in prediction. On the contrary, 2-fold cross-fitting significantly reduces the MSE value reflecting a nearly 55% improvement. As we move on to 5-fold cross-fitting, we see even a further reduction in the MSE performance with approximately a 70% improvement over the naive estimator and a 25% improvement over 2-fold. Additionally, bias is substantially lower with cross-fitting methods compared to the naive estimator, although surprisingly enough 5-fold demonstrates a slight increase in bias compared to the 2-fold approach. As expected so, variance also show improvements with cross-fitting; again with 5-fold cross-fitting achieves the lowest in both, establishing more consistent and precise results of estimation.

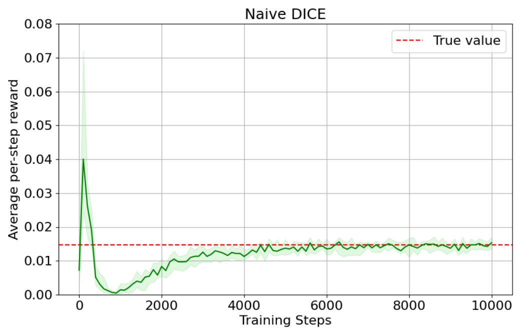
Table 2: Performance measures for the considered estimators

Scenarios	**MSE**	**Bias**	**Variance**
Naive estimator	7.773960e-06	0.000309	0.035533
2-fold cross-fit	3.424816e-06	0.000144	0.015937
5-fold cross-fit	2.487964e-06	0.000172	0.011163

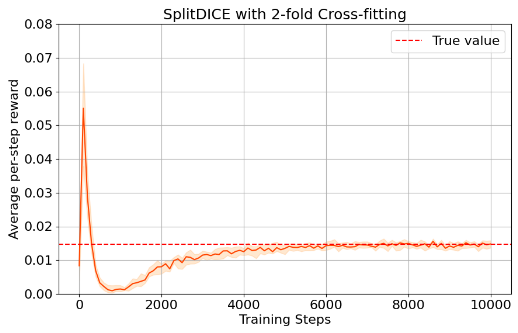
5.3 Convergence of Average Per-Step Reward over Training

Plots provided by Figure 4 demonstrate the trend for average per-step reward over 10,000 training steps. Since the experiment is repeated over 20 datasets each with a distinct seed value, the results are then generalized with the median plotted and error bars at 25th and 75th percentiles. The average per-step reward value of the target policy is used as the ground truth (true value), in order to account for all the 20 seeds we use the mean of average per-step reward values of all target policies.

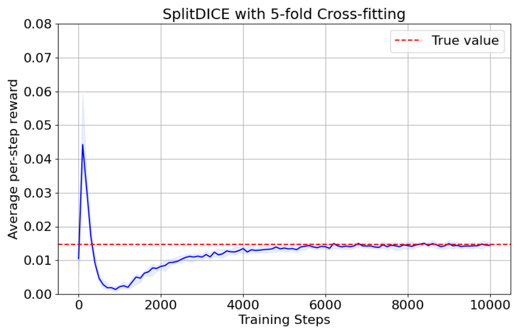
As it can be seen clearly, all three estimators show convergence to the true value of the target policy. The convergence has a more fluid movement for SplitDICE however whereas for the naive implementation, there seems to be more extremities and peaks throughout the training process. The deviation in the convergence trend of Naive DICE shown via Figure 4a is smoothed out when applied cross-fitting. Another noticeable remark is that both folds of SplitDICE start with a relatively high peak at the start of the training process, although variance is generally high this is not the case for Naive DICE. This could be attributed to sampling variability meaning the training set created as an outcome of random split of the original dataset may not generalize well to the overall dataset thereby leading to estimations that are far from the ground truth until the estimator adjusts to the true distribution during the later stages of training. This is also an apparent occurrence between the two different folds since 2-fold SplitDICE shown via Figure 4b reaches a higher initial peak (between 0.05 and 0.06) compared to the 5-fold SplitDICE shown via Figure 4c (between 0.04 and 0.05). This would also indicate that by splitting of the data into higher number of folds, the training set is more likely to be a better representative of the original dataset's variability.



(a) The sample data is used both as a training and an evaluation dataset.



(b) The sample data is split into two distinct subsets at random, namely training and evaluation. Then, the roles of the two datasets are reversed for the second fitting. The final estimate value is calculated as the average of the two fittings.



(c) The sample data is split into five distinct subsets at random. The fitting is continued until each subset is assigned the role for estimation whereas the leftover four subsets are merged into one for training. The final estimate value is calculated as the average of the five fittings.

Figure 4: **Convergence to the true value (average per-step reward of the target policy) over the entire training process of Naive DICE (a), 2-fold SplitDICE (b) and 5-fold SplitDICE (c)**. The estimation of the average per-step reward value is done at every 100-step intervals of the entire training process, which totals 10000 steps overall.

In order to truly understand the effects of the double-dipping behaviour, we also need to touch upon variance observed in the results of average per-step reward. According to the hypothesis of this research, we expect to observe a descending trend in variance when comparing the naive DICE estimator to SplitDICE (with former exhibiting higher variance). Naturally, this also proves that the regularization strategies employed by the generic neural DICE estimator in order to avoid the possibly overfitted results that would arise from using the same dataset for training and estimation, can be strengthened by use of double machine learning methods. As seen from Figure 5, there is an apparent difference between the interquartile ranges of the estimators. Both folds of SplitDICE exhibit a concentration of points near the median whereas Naive DICE shows this pattern only for a few data points. This observation indicates not only that there is a descending trend in variance from Naive to 5-fold SplitDICE but also that SplitDICE (more significantly for the 5-fold version) densely clusters data points around the desired range achieving a more stable and focused distribution. Additionally, for SplitDICE the gap between the mean and the median is much smaller suggesting that distribution of reward values is more symmetric and centered around the true value.

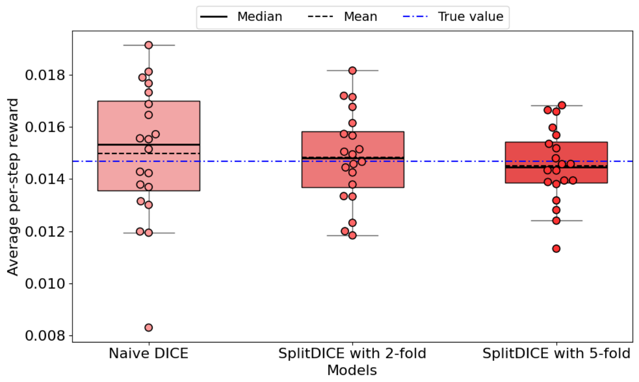


Figure 5: **Box-whisker plot showing the estimated of average per-step reward value (calculated at step=10000) for each seed.** The results are categorized by the considered estimator models, with the (non-outlier) data points displayed as a randomized swarm to avoid overlaps. Additionally, the mean ground truth value is displayed as a reference criterion for comparison.

6 Discussion

The detailed analysis of the results allowed for us to make conclusions from different aspects. While variance holds particular significance in investigating the risks of double-dipping, it was definitely worthwhile to explore how the cross-fit strategy affects bias, given the inherent trade-off between the two. The difference between the results for bias was relatively smaller compared to variance but still there was a downwards trend from Naive DICE to SplitDICE. Another interesting finding was between the two whisker plot diagrams as provided in Figure 3b and Figure 5. Although the error values retrieved by the estimators had relatively similar interquartile range widths with each other, the differences between the overall group of values was notable. However, this pattern was conceptually reversed for the average per-step reward values. In this case, the estimate values for each model showed similar closeness to the true value, but their interquartile ranges differed from each other much more significantly. To summarize, this indicates that variance in error was (nearly) unaffected changing to SplitDICE but the variance in the average per-step reward values decreased noticeably.

Now, we move on to discussing the limitations behind the research starting with the environment chosen for the experiment which was `Frozenlake-v0` featuring a discrete action and observation space. This environment was selected due to its considerable simplicity and the manageable computational effort and resources required for experimentation. However, more complex environments such as `Reacher-v2` and `Cartpole-v0` were excluded from the scope of this study. These environments, while potentially offering richer datasets with discrete action space and continuous observation space, require significantly higher computational efforts due to the relative increase in the number of training steps needed to achieve convergence to the true value as witnessed in the results of the previous DICE papers [23]. This decision was made to ensure that the experiments could be completed within a reasonable time-frame and with available computational resources.

When splitting the dataset, the episodes retrieved from the off-policy dataset are shuffled in order to achieve a fully random split. This is because random shuffling helps prevent any bias that might arise from the order in which the data was collected initially. In order to mitigate the possible fluctuations that might be caused by the variability of the shuffling process, the results were generalized across multiple seeds as suggested in previous experimentation conducted on DualDICE [13]. It is noteworthy that although the episodes were shuffled for randomness, the seeds used for generating random numbers during the experiments were not randomized. Instead, they were systematically defined to cover a range of values, from 0 to 19 inclusive. This approach was chosen to ensure consistency across different runs of the experiment and the reproducibility of the setup. However, a randomized approach for the collection of seed values could improve the reliability of the results by giving a more stable analysis that is less susceptible to randomness.

Before conducting the statistical analysis for the relative error comparison, first determining the normality of the datasets required choosing between several normality tests, primarily the Shapiro-Wilk test [19] and the Kolmogorov-Smirnov (K-S) test [4] as the two most popular techniques. Main limitation of the K-S test is its high sensitivity to extreme values. Thus, it is recommended to use the K-S test for sample sizes ≥ 50 [12] or to apply the Lilliefors correction, which makes the test less conservative [8]. Another pressing issue is that if parameters such as location, scale, and shape are estimated from the data, the critical region of the K-S test becomes invalid, necessitating simulation of such variables [16]. Unfortunately, both these limitations are pertinent to the results of the current experiment at hand. Considering that previous studies show that The Shapiro-Wilk test is more suitable for smaller sample sizes, [8] and to still account for the empirical power of the K-S test, we employed both Anderson-Darling test [20] (a refinement of the K-S Test) and Shapiro-Wilk test. The former revealed that the data for 2-fold and Naive were not normally-distributed. The latter also concluded that only 2-fold was not normally distributed (bearing in mind that the p-value calculated for the naive dataset was right above the boundary of significance). The results of these normality tests made it necessary to choose non-parametric approaches for further analysis, this being Kruskal-Wallis. However, to account for false negatives, under the assumption that all the groups are in indeed normally-distributed, we would rely on parametric approaches: one-way ANOVA could be conducted with Tukey's HSD Test for post-hoc [15]. The results of this statistical analysis concludes that there exists a significant difference between the Naive estimator and 5-fold SplitDICE at the significance level of 5%. This approach offers better significance since with the non-parametric methods, the best significance level achieved was 10%.

7 Responsible Research

To ensure the reproducibility of the results, the data was generated with seed specifications as described in section 4 and then the final results for MSE, bias and variance were retrieved as the average across all the output. For the convergence of average per-step reward over the number of training steps, the median was selected. This measure made it possible that the presented results accurately reflected the performance of the estimators, instead of merely depicting a favorable outcome. Additionally, given the precision of the values obtained at the end of the experiment and the close comparisons, the results required careful analysis and interpretation. Statistical analyses were conducted when applicable to determine the significance of the results and to compete against an academically accepted and well-known criteria. The outcome of these tests can be seen via Table 1 by means of ensuring a fair and informed critique.

From a broader societal perspective, especially the objective behind this research has high relevance for societal trust in AI applications with regards to conducting better off-policy evaluation and better estimating the outcome of a strategy. Considering certain critical sectors, more reliable OPE metrics can lead to better treatment recommendations in healthcare, ensure safer decision-making processes in autonomous driving and lead to more accurate risk assessments in finance. By creating a more accessible and discussable framework for combining behaviour-agnostic reinforcement learning with double machine learning, SplitDICE contributes to the development of more trustworthy AI systems. In such hopes, the field of the research and the motivation behind it benefit society by promoting more effective technology applications.

8 Conclusion

In this research, we aimed to answer the following research question: *"How does the use of sample-splitting and cross-fitting techniques mitigate the effects of 'double-dipping' in behavior agnostic reinforcement learning?"*. This was in the scope of integrating the techniques of sample-splitting and cross-fitting with the current implementation of the DICE estimators which have had inspiring results for the estimation of the target policy in off-policy and behaviour-agnostic settings. This approach was proposed under the name SplitDICE and for the results of this research the estimator was built on the configurations of BestDICE.

As the fundamental stage of the research, the data was split into a a number of subsets with the specific number determined by the type of cross-fitting employed. The process of cross-fitting necessitated several iterations of training on the estimator till each subset was used for the estimation of the average per-step reward value of the target policy. In order to conduct the experiment, DICE-RL codebase created by Google Research was forked for own use to make alterations to the implementation of the neural DICE estimator. For the scope of this research, three different estimator models were considered: Naive DICE (or in other words, BestDICE), 2-fold SplitDICE and 5-fold SplitDICE. Results divided into three main focus points (5.1, 5.2, 5.3 respectively) indicated that: 1) 5-fold SplitDICE has lower rates of relative error than the Naive DICE at a significance level of %10 2) in overall performance measures for MSE, bias and variance generalized from the final estimate value of average per-step reward obtained from 20 seeds show a descending trend from Naive DICE to 5-fold SplitDICE and most importantly; 3) variance calculated at the end of 10000 steps from all the observations show a descending trend going from Naive DICE to 2-fold SplitDICE and from 2-fold SplitDICE to 5-fold SplitDICE, aligning with the trend of smoothness seen from the plots.

8.1 Future Work

Future work into the field could include use of more noise or more complex environments with larger number of trajectories and trajectory lengths. The initial phase of the research focused on getting familiar with the structure of the DICE estimators due to the extensive and broad setup of the DICE-RL codebase. Therefore, simplicity and computational efficiency was a priority when selecting the type of environment. However, it is noteworthy to mention that there is potential for scalability and enhanced performance by testing these methods in more challenging environments. Testing the methods on continuous observation spaces that features more customized and unique reward functions can yield more scalable and reliable results.

The methods of sample-splitting and cross-fitting were only adopted for BestDICE variant of the unified DICE family. It is also worthwhile to extend these techniques by applying other existing variants of DICE such as DualDICE, GradientDICE etc. since they have varying constraints on primal/dual regularization as well as other parameters of interest for the neural network of the estimator.

References

[1] Yaren Aslan. Use of sample-splitting and cross-fitting techniques to mitigate the risks of double dipping in behaviour-agnostic reinforcement learning. https://github.com/compScienceYaren/dice_rl, 2024.

[2] Tali M. Ball, Lindsay M. Squeglia, Susan F. Tapert, and Martin P. Paulus. Double dipping in machine learning: Problems and solutions. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, 5(3):261–263, March 2020.

[3] Zhepeng Cen, Zuxin Liu, Zitong Wang, Yihang Yao, Henry Lam, and Ding Zhao. Learning from sparse offline datasets via conservative density estimation. (arXiv:2401.08819), March 2024. arXiv:2401.08819 [cs].

[4] I. M. Chakravarti, R. G. Laha, and J. Roy. *Handbook of Methods of Applied Statistics, Volume I.* John Wiley and Sons, Hoboken, 1967.

[5] Richard Cheng, Abhinav Verma, Gabor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel W. Burdick. Control regularization for reduced variance reinforcement learning. (arXiv:1905.05380), May 2019. arXiv:1905.05380 [cs, stat].

[6] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, and Whitney Newey. Double machine learning for treatment and causal parameters. July 2016.

[7] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 2018.

[8] Asghar Ghasemi and Saleh Zahediasl. Normality tests for statistical analysis: A guide for non-statisticians. *International Journal of Endocrinology and Metabolism*, 10(2):486–489, 2012.

[9] Daniel Jacob. Cross-fitting and averaging for machine learning estimation of heterogeneous treatment effects. (arXiv:2007.02852), August 2020. arXiv:2007.02852 [stat].

[10] Nathan Kallus and Masatoshi Uehara. Efficiently breaking the curse of horizon in off-policy evaluation with double reinforcement learning. *Operations Research*, 70, February 2022.

[11] Michail Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, January 2003.

[12] Prabhaker Mishra, Chandra M Pandey, Uttam Singh, Anshul Gupta, Chinmoy Sahu, and Amit Keshri. Descriptive statistics and normality tests for statistical data. *Annals of Cardiac Anaesthesia*, 22(1):67–72, 2019.

[13] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. (arXiv:1906.04733), November 2019. arXiv:1906.04733 [cs, stat].

[14] Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. Algaedice: Policy gradient from arbitrary experience. (arXiv:1912.02074), December 2019. arXiv:1912.02074 [cs].

[15] Anita Nanda, Abikesh Mahapatra, Bibhuti Mohapatra, and abinash mahapatra. Multiple comparison test by tukey's honestly significant difference (hsd): Do the confident level control type i error. *International Journal of Applied Mathematics and Statistics*, 6:59–65, January 2021.

[16] Guangbin Peng. Po 04 testing normality of data using sas ®. 2004.

[17] Doina Precup, Richard Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, pages 759–766, June 2000.

[18] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA, 1994. First published: 15 April 1994.

[19] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[20] M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974.

[21] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High-confidence off-policy evaluation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(11), February 2015.

[22] Masatoshi Uehara, Jiawei Huang, and Nan Jiang. Minimax weight and q-function learning for off-policy evaluation. (arXiv:1910.12809), October 2020. arXiv:1910.12809 [cs, stat].

[23] Mengjiao Yang, Ofir Nachum, Bo Dai, Lihong Li, and Dale Schuurmans. *Off-Policy Evaluation via the Regularized Lagrangian*. July 2020.

[24] Sherry Yang, Rebecca Chen, Yilei Yang, Peter Hawkins, Eugene Brevdo, and Scott Zhu Qianli. Dice: The distribution correction estimation library. https://github.com/google-research/dice_rl, 2023.

[25] Ruiyi Zhang, Bo Dai, Lihong Li, and Dale Schuurmans. Gendice: Generalized offline estimation of stationary values. (arXiv:2002.09072), February 2020. arXiv:2002.09072 [cs, stat].

[26] Shangtong Zhang, Bo Liu, and Shimon Whiteson. Gradientdice: Rethinking generalized offline estimation of stationary values. (arXiv:2001.11113), November 2020. arXiv:2001.11113 [cs, stat].