Technische Hochschule Deggendorf
Fakultät Angewandte Informatik

Studiengang Master Künstliche Intelligenz und Data Science

Deep Reinforcement Learning für Entscheidungsneurowissenschaften

Deep Reinforcement Learning for Decision Neuroscience

Masterarbeit zur Erlangung des akademischen Grades:

Master of Science (M.Sc.)

an der Technischen Hochschule Deggendorf

Vorgelegt von:
Faizan Shaikh Abdul Khalil Shaikh
Matrikelnummer: 22102199

Am: August 10th 2023

Prüfungsleitung:
Prof. Dr. Patrick Glauner

Ergänzende Prüfende:
Prof. Dr. Christoph Korn
UKHD, Heidelberg

Erklärung

Name des Studierenden: Faizan Shaikh Abdul Khalil Shaikh

Name des Betreuenden: Prof. Dr. Patrick Glauner

Thema der Abschlussarbeit:

Deep Reinforcement Learning für Entscheidungsneurowissenschaften . . . .

. . . .

. . . .

. . . .

1. Ich erkläre hiermit, dass ich die Abschlussarbeit gemäß § 35 Abs. 7 RaPO (Rahmenprüf-ungsordnung für die Fachhochschulen in Bayern, BayRS 2210-4-1-4-1-WFK) selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Deggendorf,   *10.08.2023*
. . . .
　　　Datum　　　　　　　　　　　　　　　Unterschrift des Studierenden

2. Ich bin damit einverstanden, dass die von mir angefertigte Abschlussarbeit über die Bib-liothek der Hochschule einer breiteren Öffentlichkeit zugänglich gemacht wird:

○ Nein

● Ja, nach Abschluss des Prüfungsverfahrens

○ Ja, nach Ablauf einer Sperrfrist von . . . Jahren.

Deggendorf,   *10.08.2023*
. . . .
　　　Datum　　　　　　　　　　　　　　　Unterschrift des Studierenden

Bei Einverständnis des Verfassenden vom Betreuenden auszufüllen:

Eine Aufnahme eines Exemplars der Abschlussarbeit in den Bestand der Bibliothek und die Ausleihe des Exemplars wird:

○ Befürwortet

○ Nicht befürwortet

Deggendorf, ...
　　　　　　　　　Datum　　　　　　　　　　　　　Unterschrift des Betreuenden

Acknowledgements

I would like to express my heartfelt gratitude to the following individuals who have played instrumental roles in the successful completion of this thesis. First and foremost, I extend my deepest appreciation to my first supervisor, Prof. Dr. Patrick Glauner. Your unwavering guidance, insightful feedback, and dedication to my academic growth have been invaluable. Your mentorship has truly enriched my understanding of the subject matter and has inspired me to strive for excellence. I am also deeply thankful to my external supervisor, Prof. Dr. Christoph Korn. Your expertise, constructive feedback, and unique perspective have greatly enriched the quality of this work. Your contributions have expanded the scope of my research and added depth to my analysis, for which I am truly grateful. I would like to extend my heartfelt gratitude to the co-author of the work, PhD candidate Sergej Golowin. Your collaboration has been pivotal in shaping the trajectory of this research. From the inception of the problem definition to the meticulous process of data collection, your involvement has been pivotal. Your guidance in data exploration, predictive modelling, and your consistent feedback throughout the entire thesis journey have been invaluable. I am also indebted to my colleagues, friends, and family, whose unwavering support, encouragement, and understanding have sustained me throughout this academic endeavour. Your belief in my abilities has been a constant source of motivation, and I am deeply grateful for your presence in my life. To everyone who has contributed, in ways big and small, to the realization of this thesis, I extend my heartfelt thanks. Your collective influence has shaped not only the outcome of this research but also my growth as a scholar. I am truly blessed to have had such an incredible support network, and I look forward to carrying the lessons learned from this journey into the next chapter of my academic and professional life.

Abstract

The study investigates human decision-making behaviour within a game-based context and endeavours to replicate said behaviour using the Generative Adversarial Imitation Learning (GAIL) technique. In this gamified environment, inspired by a hunter-gatherer scenario, players have to ensure their survival in a challenging environment while accounting for their episodic homeostasis and factoring in current and future climatic conditions, necessitating the estimation of stochastic trade-offs. The initial phase of the study primarily centres on the collection and analysis of data from healthy participants, thereby yielding valuable insights into their gameplay dynamics and the cognitive processes underpinning their decision-making. An overarching observation is that participants can seemingly differentiate between cases where simple heuristics advance the game and those cases where prior and present information is necessary for informed action selection. Subsequently, the study pivots toward predictive modelling, firstly by considering the task in a supervised learning paradigm as the basis for behavioural cloning by utilizing a white-box decision tree algorithm. The trained decision tree model attained a survival rate of 20%, proximity to the human benchmark of 21%, whereas comparison with imitation-based evaluation metric, namely Monte Carlo Distance (MCD) registered a score of 8.92 - a decent score considering the stochasticity of the game and variability in human behaviour. On the other hand, GAIL regards the task in an inverse reinforcement learning framework and attempts to imitate the behaviour, achieving a score of 16% as a survival rate and an MCD score of 8.83, showcasing competitive performance and effective imitation capabilities. The study also looks closely at how the GAIL model works using post hoc model explainability, more specifically, utilizing Shapley analysis and training a decision tree on GAIL's synthetic behavioural data, suggesting that GAIL is capable of learning complex strategies that are similar to those used by humans. This research contributes to the understanding of resource management, risk assessment, and strategic thinking within a game environment, and demonstrates the potential of GAIL for imitating human behaviour in a tabular setting. Code can be found here: `https://github.com/faizankshaikh/ForaGym`

Contents

List of Figures

1 Introduction

1.1 Motivation

Human decision-making behaviour has long been a subject of great interest and significance in various domains, including psychology, economics, and artificial intelligence. Understanding how individuals make choices and respond to dynamic environments is essential for developing effective strategies and decision-support systems. In the realm of artificial intelligence and machine learning, the ability to emulate human decision-making processes can lead to more robust and adaptive algorithms.

The central focus of the game designed for the study is to ensure the survival of players by maintaining their health above a specified threshold for a specific duration in an episode. Accomplishing this goal necessitates making foraging decisions based on current and future climate factors, which adds a layer of complexity and uncertainty to the decision-making process. This thesis then delves into the investigation of human decision-making behaviour within the game and aims to emulate this behaviour using Deep Learning, more specifically, Generative Adversarial Imitation Learning.

1.2 Outline

The study begins with an introduction (Chapter 1) that outlines the motivation behind this research and provides an overview of the entire thesis structure. Following the introduction, Chapter 2 presents the background knowledge necessary for understanding the subsequent chapters. This includes an explanation of the basics of machine learning, deep learning, reinforcement learning, and imitation learning. Furthermore, it delves into the important hyperparameters associated with decision trees, multi-layer perceptrons (MLP), and Generative Adversarial Imitation Learning (GAIL). Chapter 3 offers a literature review, providing insights into prior work and research relevant to the thesis. Chapter 4 elaborates on the methodology employed in this study. It describes the problem design and the process of predictive modelling, while also exploring various methods for explainability in the context of the models utilized. The empirical results and data analysis are presented in Chapter 5. This chapter includes details about data collection, exploratory data analysis, data preprocessing, and feature engineering techniques applied to the raw and derived features. Additionally, it discusses the model training and evaluation process, focusing on decision tree and GAIL models. The chapter concludes with an in-depth evaluation of the model's performance and explainability. Chapter 6 is dedicated to the discussion of the obtained results and their implications, providing a critical analysis of the research findings. Furthermore, it discusses potential future work and directions for further research in the field of imitation learning, predictive modelling, and explainability.

Finally, Chapter 7 concludes this thesis by summarizing the key findings and contributions made in the study. It reiterates the importance of predictive modelling and explainability in imitation learning and outlines the broader implications of the research.

2 Background Knowledge

2.1 Basics of Machine Learning

A learning algorithm in machine learning refers to a computational method or process that enables a system to automatically acquire knowledge or make predictions based on data. It involves constructing a mathematical model by learning patterns and relationships from the given input data. Generalization of new examples is crucial in machine learning because it ensures that the learned model can accurately predict outcomes for unseen or future data points. The goal is not merely to fit the training data perfectly but to capture underlying patterns that hold true for new instances as well.

Supervised learning is a category of machine learning where the algorithm learns from labelled examples, where the input data is accompanied by corresponding desired outputs. The algorithm's objective is to learn a mapping function that can predict the correct output for unseen inputs.

There are several examples of supervised learning algorithms, including linear regression, logistic regression, support vector machines, and random forests. Each algorithm has its own characteristics and is suitable for different types of problems. In the context of supervised learning, an optimization algorithm aims to find the optimal values of the model's parameters by minimizing a specific objective function. This process involves adjusting the parameters iteratively based on the error or loss between the predicted and actual outputs.

Decision Trees are a popular machine learning algorithm that uses a hierarchical structure to make predictions. They work by splitting the input space into regions based on feature values, forming a tree-like structure. At each internal node, a splitting criterion is used to decide which feature to split on. The process continues recursively until the tree reaches a certain depth or a stopping criterion.

2.1.1 Hyperparameters of Decision Trees

Hyperparameters are adjustable parameters that are not learned from the data but are set by the user. These parameters control the behaviour of the learning algorithm and can significantly impact the performance and generalization ability of the model. Examples of hyperparameters include the learning rate, regularization strength, and the number of hidden layers in a neural network. Hyperparameters in Decision Trees play a significant role in controlling the behaviour, complexity, and performance of the algorithm.

Let's delve into the significance and use of the specific hyperparameters commonly associated with Decision Trees that are used in this work.

1. **Splitting Criteria**: The splitting criteria determine how the algorithm measures the quality of a split at each internal node of the tree. Two commonly used criteria are Gini

impurity and information gain. Gini impurity measures the probability of misclassifying a randomly chosen element from a node if it were randomly labelled according to the distribution of classes in that node. Information gain, on the other hand, quantifies the reduction in entropy (or uncertainty) achieved by splitting on a particular feature. The choice of splitting criteria depends on the problem and the desired behaviour of the Decision Tree.

2. **Max Depth**: The max depth hyperparameter limits the maximum depth or length of the Decision Tree. It controls the complexity of the tree and helps prevent overfitting, where the model memorizes the training data too closely and fails to generalize well to unseen data. By setting an appropriate max depth, we can find a balance between capturing intricate relationships and avoiding excessive complexity.

3. **Minimum Weight Fraction Leaf**: This hyperparameter specifies the minimum fraction of samples required to be present at a leaf node for further splitting to occur. It helps control the granularity of the Decision Tree. Setting a higher value for this hyperparameter can result in fewer and larger leaf nodes, leading to a simpler tree structure. Conversely, a lower value can create more leaf nodes, potentially capturing more specific patterns in the data. It is used to control the trade-off between model complexity and capturing fine-grained details.

4. **Minimum Impurity Decrease**: The minimum impurity decrease hyperparameter determines the threshold for splitting a node based on the decrease in impurity achieved. It quantifies the minimum improvement required to justify a split. A higher value for this hyperparameter results in fewer splits and a simpler tree, as only significant impurity decreases will be considered for further partitioning. Lower values allow for more splits, potentially capturing finer distinctions in the data. This hyperparameter helps control the trade-off between simplicity and sensitivity to small impurity improvements.

By carefully tuning these hyperparameters, we can optimize the performance of Decision Trees for a specific problem. It often involves a trial-and-error process or using techniques like grid search or randomized search to explore different combinations of hyperparameter values and select the ones that yield the best performance in terms of accuracy, precision, recall, or other evaluation metrics.

2.2 Basics of Deep Learning

Deep learning is a subfield of machine learning that focuses on training artificial neural networks with multiple layers to learn and extract complex patterns from data. Neural networks are computational models inspired by the structure and function of biological neurons in the brain. They consist of interconnected layers of nodes (neurons) that process and transform input data to produce output predictions.

Deep learning differs from traditional machine learning approaches by its ability to automatically learn hierarchical representations of data. Deep neural networks can automatically

learn and extract higher-level features from raw data, enabling them to capture intricate relationships and patterns that may be challenging to specify manually.

A typical structure of a multilayer perceptron (MLP) model, a type of neural network, involves an input layer, one or more hidden layers, and an output layer. Each layer contains multiple nodes (neurons) that perform computations on the input data. The hidden layers act as intermediate representations, progressively transforming the input data to produce the final output predictions.

Backpropagation is a key algorithm used to update the weights of the neural network based on the calculated errors. It propagates the error from the output layer back to the hidden layers, allowing the network to adjust its weights and improve its predictions.

2.2.1 Hyperparameters of MLP

Hyperparameters play a critical role in deep learning models, allowing fine-tuning and optimization of the model's performance. Let's elaborate on the significance and use of the specific hyperparameters mentioned in the context of multilayer perceptron (MLP) models:

1. **Number of Hidden Layers**: The number of hidden layers in an MLP determines the depth of the network. Adding more hidden layers increases the model's capacity to learn complex patterns and representations from the data. However, an excessively deep network may lead to overfitting, especially when training data is limited. Balancing model complexity and generalization is crucial when deciding the appropriate number of hidden layers.

2. **Optimization Algorithm**: The optimization algorithm determines how the network's weights are updated during training. Gradient descent is a common optimization algorithm that calculates the gradient of the loss function with respect to the model's parameters and adjusts the weights in the opposite direction of the gradient to minimize the loss. Adam is a popular variant of gradient descent that adapts the learning rate for each weight during training. It combines the advantages of different adaptive learning rate methods and uses estimates of the first and second moments of the gradients to adaptively adjust the learning rate. Adam tends to provide faster convergence and improved performance compared to traditional gradient descent methods, especially in large-scale deep learning tasks.

3. **Loss Function**: The loss function quantifies the discrepancy between the predicted outputs and the actual ground truth. Cross-entropy or log loss is commonly used in classification tasks. It measures the difference between predicted class probabilities and the true class labels. Cross-entropy loss encourages the model to assign higher probabilities to the correct classes, thereby promoting accurate classification.

4. **Learning Rate**: The learning rate determines the step size taken during weight updates. Selecting an appropriate learning rate is crucial for effective training. If the learning rate is too high, the model may fail to converge or exhibit unstable behaviour. On the other hand, if the learning rate is too low, the training process can become slow or get

stuck in suboptimal solutions. Techniques such as learning rate schedules or adaptive methods like Adam can help dynamically adjust the learning rate during training for better convergence.

5. **Batch Size**: The batch size refers to the number of training examples processed in each iteration of the optimization algorithm. It affects both computational efficiency and the quality of weight updates. Larger batch sizes provide smoother convergence and more accurate gradient estimates but require more memory and computational resources. Smaller batch sizes can lead to faster computations but may have more noisy updates due to smaller sample sizes. Determining an appropriate batch size involves finding a balance between computational efficiency and the convergence characteristics of the model.

By carefully selecting and tuning these hyperparameters, practitioners can optimize the performance of deep learning models for specific tasks. Often, a combination of domain knowledge, experimentation, and techniques like grid search or random search is employed to find the optimal set of hyperparameters that result in improved model accuracy, convergence speed, and generalization ability.

2.3 Basics of Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning that focuses on training agents to make sequential decisions in an environment to maximize a cumulative reward signal. In RL, an agent interacts with an environment, taking actions based on its current state and receiving feedback in the form of rewards or penalties.

Reinforcement learning differs from traditional machine learning approaches in that it involves learning through trial and error. Instead of being provided with labelled examples or explicit instructions, RL agents learn by exploring and receiving feedback from the environment.

A fundamental concept in RL is the Markov Decision Process (MDP). An MDP is a mathematical framework that models the RL problem. It consists of a set of states, actions, transition probabilities, rewards, and a discount factor. The MDP assumes the Markov property, which states that the future state depends only on the current state and action, disregarding the history of previous states and actions.

Deep reinforcement learning (DRL) combines RL with deep learning techniques to tackle complex, high-dimensional problems. It involves using deep neural networks as function approximators to learn policies or value functions directly from raw sensory inputs.

There are various algorithms in deep RL, including Proximal Policy Optimization (PPO). PPO is an actor-critic algorithm that uses an actor network to estimate the policy and a critic network to estimate the value function. It improves the policy by iteratively updating the actor using the policy gradient and the critic using the value loss.

PPO is trained through a process of collecting trajectories in the environment, computing advantages based on the value estimates, and optimizing the policy using stochastic gradient ascent. It balances exploration and exploitation by clipping the policy update to prevent large policy changes.

Hyperparameters of PPO include the learning rate, batch size, and the number of epochs which are already explained in the Deep Learning subsection. These hyperparameters influence the convergence and performance of the algorithm. Additionally, PPO uses the value loss instead of log loss as the loss function for training the critic network. The value loss measures the mean squared error between the predicted and target values, helping the critic to estimate the expected return accurately.

By understanding the concepts of RL, MDPs, and DRL, practitioners can leverage algorithms like PPO to develop agents that can learn and make optimal decisions in complex, dynamic environments. Tuning the hyperparameters and selecting appropriate loss functions are essential for achieving good performance in reinforcement learning tasks.

2.4 Groundwork for Imitation Learning

Imitation learning is a machine learning approach that aims to learn policies or behaviours by imitating expert demonstrations. It leverages the availability of expert demonstrations to guide the learning process and facilitate the acquisition of optimal behaviours.

Imitation learning can be beneficial in solving reinforcement learning problems by providing a shortcut to policy learning. Rather than relying solely on trial and error exploration, imitation learning utilizes the knowledge and expertise of humans or expert demonstrators to bootstrap the learning process and accelerate convergence.

There are different types of imitation learning, namely behavioural cloning and inverse reinforcement learning. Behavioural cloning involves learning a policy by directly mapping states to actions using the expert's demonstrations as labelled training data. In contrast, inverse reinforcement learning seeks to infer the underlying reward function from the expert's behaviour and then optimize a policy based on this inferred reward function.

Generative adversarial imitation learning (GAIL) is an imitation learning method that combines generative adversarial networks (GANs) with reinforcement learning. GANs are neural network models that consist of a generator and a discriminator. The generator attempts to generate synthetic data that resembles the expert demonstrations, while the discriminator aims to distinguish between the generated data and the expert demonstrations.

In GAIL, the generator model learns to generate action sequences based on observed states, aiming to imitate the expert's behaviour. The discriminator model, on the other hand, learns to distinguish between the generated actions and the expert's actions. The training process involves iteratively updating both the generator and discriminator models to improve the quality of generated actions and enhance the discriminator's ability to differentiate between real and generated actions.

2.4.1 Hyperparameters of GAIL

Hyperparameters play a crucial role in the effectiveness and performance of GAIL. Let's elaborate on the significance and use of the specific hyperparameters mentioned in the context of GAIL:

1. **Architecture of Generator and Discriminator Models**: The architecture of the generator and discriminator models determines their complexity and representational power. A more complex architecture, such as deeper or wider neural networks, can capture more intricate patterns and generate more accurate actions. However, overly complex models may also lead to overfitting or slow convergence. Finding the right balance between model complexity and generalization is essential when selecting the architectures for the generator and discriminator.

2. **Number of Updates per Round of Discriminator**: GAIL uses an adversarial training approach, where both the generator and discriminator models are updated iteratively. The number of updates per round of the discriminator affects the training dynamics and the equilibrium between the generator and discriminator. Increasing the number of updates per round may enhance the discriminator's ability to discriminate between real and generated actions. However, it can also make the training process more computationally expensive. Finding the optimal number of updates per round is crucial for achieving a stable and effective training process.

Tuning these hyperparameters is essential to optimize the performance of GAIL. It often involves a trial-and-error process and experimentation to find the right values that lead to improved action generation and discriminator performance. It is important to strike a balance between model complexity, computational efficiency, and the ability to generate actions that resemble expert demonstrations accurately.

Moreover, hyperparameter tuning can be guided by domain knowledge and previous experience with similar tasks. Techniques such as grid search or random search can be employed to explore different combinations of hyperparameter values and select the ones that yield the best performance, such as generating actions that closely match the expert's behaviour and achieving effective discrimination between real and generated actions.

By leveraging the concepts of imitation learning and techniques like GAIL, practitioners can learn from expert demonstrations and accelerate the acquisition of optimal policies in reinforcement learning settings. Careful tuning of hyperparameters ensures effective training and improves the quality of generated actions and the discriminator's discrimination ability.

3 Literature Review

Decision-making under uncertainty has received greater attention in cognitive neuroscience [1], with several lines of evidence elucidating different variants of uncertainty, such as risk, ambiguity, and expected and unexpected forms of uncertainty [2]. Markov Decision Processes (MDPs) are one of the analytical tools used for enlightening decision-making under uncertainty, especially when considering the sequential dynamics of decision-making [3]. The first part of our study is derived from Korn and Bach [4] as a hunter-gatherer game (i.e. MDP task), where they provide evidence for a trade-off between heuristic and optimal decision policies in human decision-making. The second part of our study focuses on modelling human behaviour through Deep Reinforcement Learning (DRL) and Decision Trees. Recent advances of DRL have led to artificial agents capable of producing behaviour that meets or exceeds human-level performance in a wide variety of tasks [5]. We use DRL, more specifically, Generative Adversarial Imitation Learning [6] in order to mimic human behaviour. Decision trees [7] on the other hand, are widely used in settings where interpretable machine learning models are required [8]. Combining DRL with decision trees [9] can provide both performance and interpretability that can be used to study human behaviour. This methodology is inspired by Pan, Menghai, et al [10], but applies it to a text-based game [4] through the lens of VIPER [9].

4 Methodology

4.1 Problem Design

We developed a text-based hunter-gatherer game that simulates the challenges faced by individuals striving to survive. The primary objective of the game is for players in a particular forest to maintain their health above a certain threshold for a certain number of days. To achieve this, players must make critical decisions on a daily basis, choosing between foraging for food or waiting for the next day in hopes of better environmental conditions, such as greater availability of food or less chance of facing threats. The game operates in an RL setting, building upon the work of Korn et al [4]. The stimuli (i.e. the frontend) given to the participants is given in figure 4.1. The goal is to create an engaging and realistic environment where we can experimentally test the decision-making abilities of players.
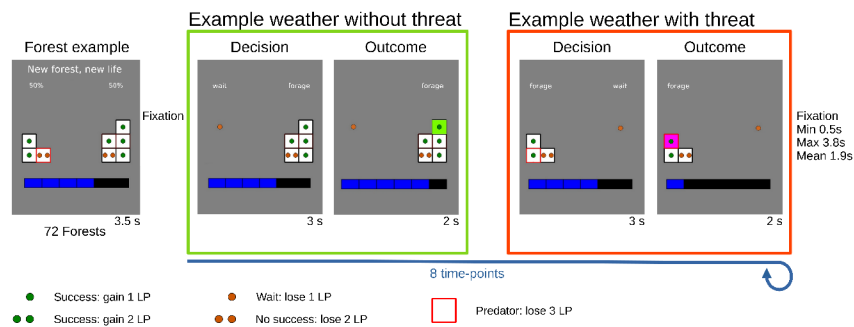

Figure 4.1: Task figure

The game design encompasses five key components:

1. **Displaying the Initial State**: At the beginning of each gameplay session, the game presents players with the initial state, which includes two climate conditions which is chosen from one of the 72 forests. These climate conditions are categorized as relatively good or bad in comparison to each other. Each climate condition is characterized by three properties: forest quality (which is a probability value ranging from 0 to 1), threat encounter (which is again a probability value ranging from 0 to 1), and nutritional quality

(either 1 or 2 which signifies higher or lower gain in health). This initial state sets the foundation for the players' survival journey.

2. **Displaying the Current State**: The current state provides essential information for decision-making, which includes the number of days (ranging from 0 to 8, initialized as 8), the player's life points (ranging from 0 to 6, initialized as 4 or 5 randomly), and a randomly chosen climate condition for the particular forest chosen in the previous step. This information enables players to assess their current situation and strategize accordingly.

3. **Prompting User Action**: After displaying the initial and current state, players are prompted to take action. They can choose between two options: foraging for food or waiting until the next day. This decision is crucial, as it directly affects the player's chances of survival. The prompt serves as a pivotal point where players must weigh the risks and benefits of each action in light of the current state and alternative forest conditions.

4. **Displaying Consequence of Actions**: Once players make their decision, the game reveals the consequences of their particular action. There are a total of seven possible outcomes that players may experience. For each climate condition, players may encounter successful foraging (the result of which is that the player gains life points based on nutritional quality), failed foraging attempts (the result of which is that life points decrease by 2), or threat encounters (the result of which is that life points decrease by 3). Additionally, there is a separate consequence for choosing to wait, where life points decrease by 1. The numerical value of gain/loss of life points regulates the amount of risk the player is allowed to take depending upon the current state. These consequences provide immediate feedback to players, shaping their understanding of the impact of their decisions.

5. **Iterative Gameplay**: If the player survives (i.e. life points greater than 0) and the episode has not yet ended (i.e. days left greater than 0), the game proceeds to the next step, allowing the player to continue their survival journey. This iterative gameplay structure allows for multiple opportunities for players to adapt their strategies, learn from their experiences, and explore different approaches to survival.

Overall, the game design aims to provide an engaging and realistic simulation of the challenges faced by hunter-gatherers in a forest environment. By incorporating various environmental factors, decision-making scenarios, and consequences, the game seeks to test players' survival instincts and strategic thinking, offering an immersive and rewarding experience.

4.2 Predictive modelling

We explored four distinct approaches, each contributing to our understanding and solution development. These approaches can be categorized as follows:

1. **Heuristic Methods**: To establish a benchmark, we employed heuristic methods that encompassed several simple strategies, most notably: "just forage," "just wait," "random action," and "hail mary." The first three heuristics involve straightforward decision-making based on fixed rules. The "hail mary" heuristic, on the other hand, involves waiting until the player's life points reach a specific threshold, defined as 3 (see Appendix). These heuristic methods provided initial insights into survival strategies and offered comparative metrics for evaluating more advanced approaches.

2. **Dynamic Programming (DP)**: Dynamic Programming (or more specifically, backwards induction) is an exact solution method that involves iteratively updating the action-value function for each state. This process begins from the final state and progresses towards the initial state. The update is based on the Bellman equation, which incorporates the immediate reward (R) and the discounted value (V) of successor states, as mentioned in the equation below

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + v_\pi(s')] \tag{4.1}$$

where,

   - s is the current state and s' is the next state
   - $v_\pi(s)$ is the value of state s under policy π.
   - $\pi(a|s)$ is the probability of taking action a in state s.
   - $p(s',r|s,a)$ is the probability of transitioning to state s' and receiving reward r after taking action a in state s.
   - r is the reward received for taking action a in state s and transitioning to state s'.
   - $v_\pi(s')$ is the value of state s' under policy π.

By leveraging DP, we aimed to optimize the decision-making process and determine the most effective actions for each state, taking into account the long-term consequences and potential rewards. We set the discount factor γ to 1 and provide the full knowledge about the temporal domain of the task to the DP model

3. **behavioural Cloning**: behavioural cloning entails training an agent to imitate the behaviour of an expert in a supervised manner. This approach involves learning a policy directly from expert demonstrations without explicitly considering the environmental dynamics or rewards. Our implementation of behavioural cloning utilized Decision Trees as the base algorithm, allowing us to gain white-box access to the internal working mechanism. By building a mapping between states (S) and actions (A) using a supervised learning approach, the agent attempted to replicate the expert's actions for each encountered state.

4. **Generative Adversarial Imitation Learning (GAIL)**: GAIL utilizes a combination of generative adversarial framework with imitation learning to learn a policy from expert demonstrations, where the expert is actual human behavioural data. The generator part

of GAIL strives to generate actions that are indistinguishable from those of the expert, whereas the discriminator has to find the irregularities in the demonstrations and classify if they are generated or real. The pseudocode for GAIL can be described as follows

---

Algorithm 1 Generative Adversarial Imitation Learning

1: Input: Expert trajectories τ_e, initial policy π_0 and discriminator parameters θ_0

2: **for** $i = 0, 1, 2, \ldots$ **do**

3:     Sample trajectories τ_g from π_i

4:     **Discriminator update:**

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_w \log\left(D_w(s,a)\right)\right] + \hat{\mathbb{E}}_{\tau_E}\left[\nabla_w \log\left(1 - D_w(s,a)\right)\right] \tag{4.2}$$

Explanation: * The first term in the update equation is the expected log-likelihood of the discriminator over the expert trajectories. This term encourages the discriminator to learn to classify real trajectories as real. * The second term in the update equation is the expected log-likelihood of the discriminator over the generated trajectories. This term encourages the discriminator to learn to classify generated trajectories as fake. * The ∇_θ term takes the gradient of the update equation with respect to the discriminator parameters θ.

5:     **Policy update:**

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a \mid s)Q(s,a)\right] - \lambda\nabla_\theta H\left(\pi_\theta\right), \tag{4.3}$$

where
$$Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}\left[\log\left(D_{w_{i+1}}(s,a)\right) \mid s_0 = \bar{s}, a_0 = \bar{a}\right]$$

Explanation: * The first term in the update equation is the expected log-likelihood of the discriminator over the expert trajectories. This term encourages the policy to generate trajectories that are more likely to be classified as real by the discriminator. * The second term in the update equation is the entropy of the policy π_i. The entropy is a measure of how random the policy is.

6: **end for**

It is worth noting that the generator part of GAIL is built upon the base algorithm of Proximal Policy Optimization (PPO), which is a popular reinforcement learning algorithm. PPO optimizes the agent's policy using a surrogate objective function, which ensures that the policy update remains within a specified proximity bound to prevent drastic policy changes. The pseudocode of PPO as described in [11] is listed in figure 4.2

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6:     Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

    typically via stochastic gradient ascent with Adam.
7:     Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

    typically via some gradient descent algorithm.
8: **end for**

Figure 4.2: Pseudocode for PPO

The combination of PPO and the generative adversarial framework in GAIL enhances the agent's ability to learn from expert demonstrations and refine its policy through the adversarial training process

4.3 Methods for Explainability

In order to enhance the interpretability of deep reinforcement learning models, we adopted two methods that shed light on the inner workings of these complex systems. These methods enable us to gain insights into the learned policies and understand the contributions of different features within the model.

1. **Shapley values**: Shapley values are a powerful technique that assigns importance or contribution scores to the features or components of a deep neural network model. This approach quantifies the marginal contributions of each feature towards the model's output prediction. This helps us understand which features are more influential and how they contribute to the overall predictions.

2. **Policy Extraction using Decision Trees**: Drawing inspiration from explaining human behaviour through Decision Trees, we apply a similar methodology to train a Decision Tree model that emulates the behaviour of the trained deep reinforcement learning model. This process involves using the learned deep reinforcement learning model as a teacher and generating a dataset of state-action pairs. We then train a Decision Tree model in a supervised learning manner using this dataset, aiming to capture the decision rules and policies learned by the deep model. The resulting Decision Tree model serves as a transparent and interpretable representation of the complex deep reinforcement learning model. By analyzing the structure of the Decision Tree and examining the

decision rules, we gain a deeper understanding of how the deep model makes decisions in different states.

4.4 Evaluation Metrics

To assess the effectiveness of our learning algorithms in the RL environment and evaluate their ability to imitate expert behaviours, we employed two key metrics that provide valuable insights into their performance.

1. **Survival Rate**: The survival rate metric quantifies the agent's ability to survive until the last day in the RL environment. For artificial agents, we calculated the mean and standard deviation of survival rate over 10,000 episodes over 10 intervals. A higher survival rate indicates a more successful and adaptive agent that can navigate the environment effectively, make optimal decisions, and ensure its own survival. By measuring the survival rate, we can gauge the proficiency of the learning algorithms in tackling the challenges posed by the environment and their capacity to learn from past experiences to improve future decision-making.

2. **Monte Carlo Distance**: Monte Carlo Distance is a way to measure how similar an agent's behaviour is to that of an expert. It helps us see how well the agent imitates the expert.

   Imagine the expert and the agent are both navigating through a series of situations, making decisions at each step. The Monte Carlo Distance compares the actions they take in these situations. Here's how it works:

   a) We gather trajectories (sequences of actions and observations) from both the expert and the agent.

   b) For each time step in the expert's trajectory, we calculate the probability of observing that specific action in the expert's trajectory.

   c) Next, we compare these probabilities to the actions taken by the agent. If the agent's actions are similar to the expert's actions, the Monte Carlo Distance will be smaller.

   Monte Carlo distance can be mathematically defined as follows

$$H\left(T, T'\right) = -\frac{1}{n} \sum_{i=1}^{n} \log \left[\frac{\sum_{j=1}^{m} \Pi_{\{o'_j\}}\left(o_i\right) + 1}{m + |S| \times |A|} \right]$$

where:

T and T' are the expert and generated trajectories, respectively

n = number of timesteps in each trajectory

m = number of expert trajectories

o_i = observation at timestep i in the expert trajectory

$\Pi(o'_j)$ = probability of observing o_i in the jth expert trajectory

$|S|$ = number of states in the environment

$|A|$ = number of actions in the environment

5 Data Analysis and Results

5.1 Data Collection

A total of 29 healthy individuals (14 male, 15 female) with a mean age of 23.93 ± 3.73 (mean ± standard deviation) participated. Each experiment for an individual consisted of four mini-blocks, each containing 18 forests. Participants had the opportunity to win an incentive of €0.50 if they successfully survived a forest without being eliminated, as indicated in the problem design. Each participant had the potential to complete 576 trials (72 forests multiplied by 8 days). The average number of trials per participant, excluding "none" responses, was 361.45 ± 23.22, resulting in a total of 10,482 decisions. These "none" responses are marked when the participants go beyond the time limit to respond, and for these cases, a default "wait" action is applied. On average, the probability of successfully surviving a forest was 22% ± 2.25. Responses categorized as "none" accounted for 0.02 ± 2.69 of all recorded responses.

5.2 Exploratory Data Analysis

To gain deeper insights into the collected data and understand the patterns and relationships within the variables, we employed various data visualization techniques. These visualizations allow us to explore the characteristics of the game environment and the decision-making behaviours of the human participants. The following visualizations were generated:

1. **Frequency plot of the number of days left**: This plot illustrates the distribution of the number of days remaining in the game across the dataset. It provides an overview of how frequently each number of days occurs and allows us to analyze the duration of the game episodes as experienced by the human participants.
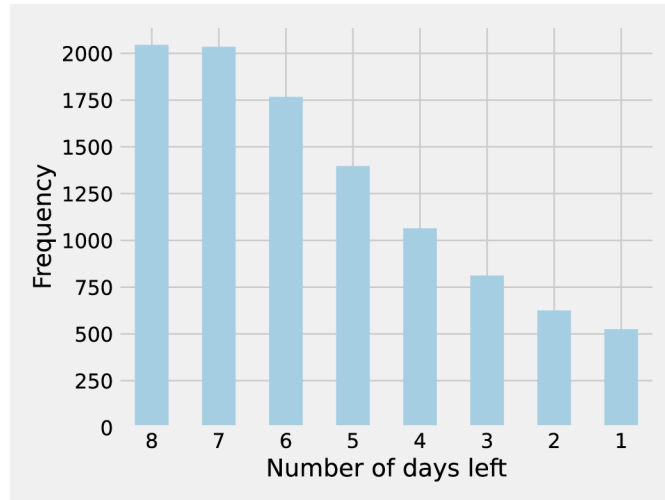
Figure 5.1: Frequency plot of the number of days left

2. **Frequency plot of the number of life points left**: This plot showcases the distribution of the number of life points remaining in the game. It provides insights into the variations in life points and helps identify any trends or patterns in the health status of the human participants throughout the episodes.
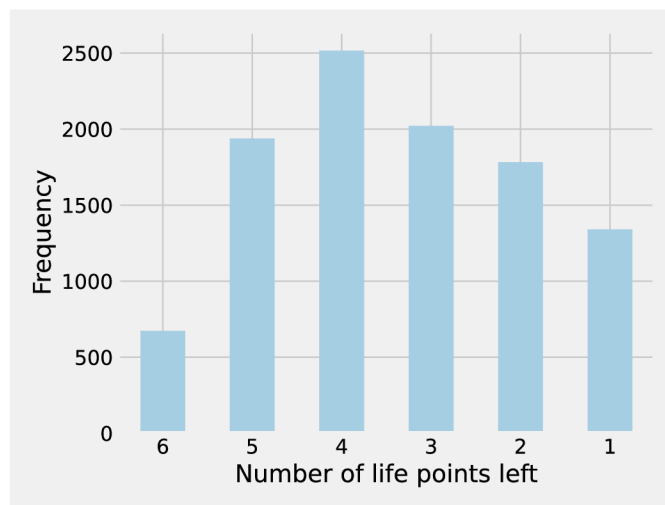

Figure 5.2: Frequency plot of the number of life points left

3. **Stacked frequency plot of the number of days left along with the number of life points left**: This stacked plot presents the joint distribution of the number of days and the number of life points remaining. It allows us to observe the relationship between these two variables and explore how their values interact with each other during the game as experienced by the human participants.
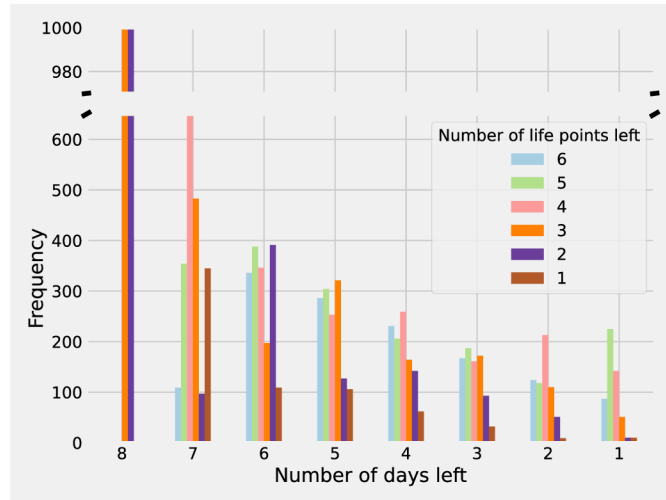
Figure 5.3: Stacked frequency plot of the number of days left along with the number of life points left

4. **Stacked frequency plot of the number of days left for a particular action taken**: This visualization focuses on the distribution of the number of days remaining in the game based on the actions taken by the human participants. It provides insights into how their decision to forage or wait influences the duration of the game episodes.
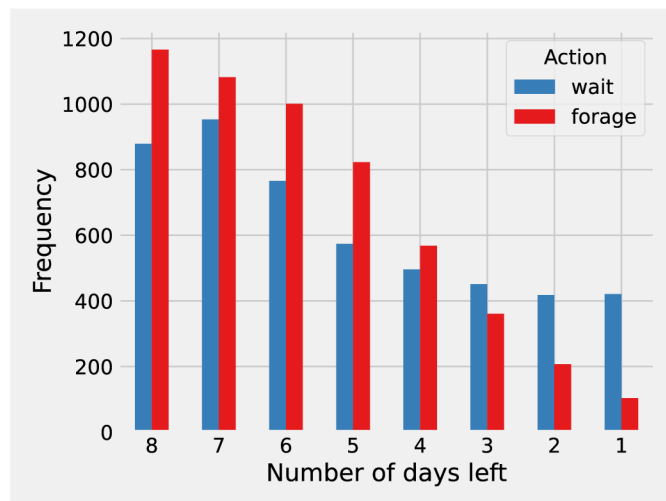

Figure 5.4: Stacked frequency plot of the number of days left for a particular action taken

5. **Stacked frequency plot of the number of life points left for a particular action taken**: Similar to the previous visualization, this plot showcases the distribution of the number of life points remaining based on the actions taken by the human participants.

It allows us to examine how their decisions impact their health status throughout the game.
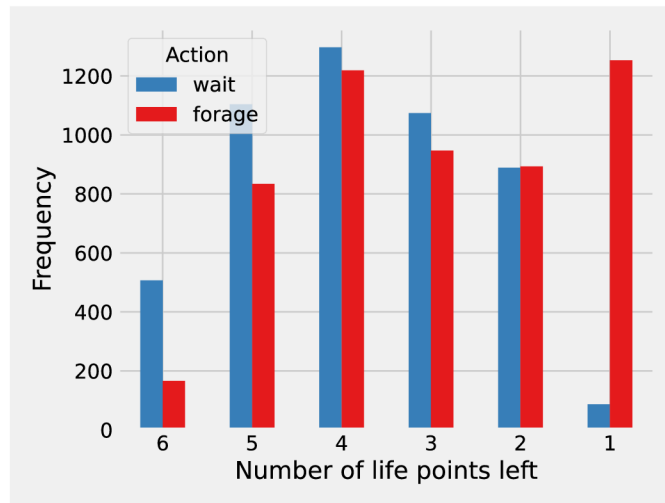

Figure 5.5: Stacked frequency plot of the number of life points left for a particular action taken

6. **Violin plot of the probability of success for a particular action taken**: This violin plot displays the distribution of the probability of success for a specific action taken by the human participants. It provides a visual representation of the variability in their success rates and allows us to compare the effectiveness of different actions as observed in their gameplay.
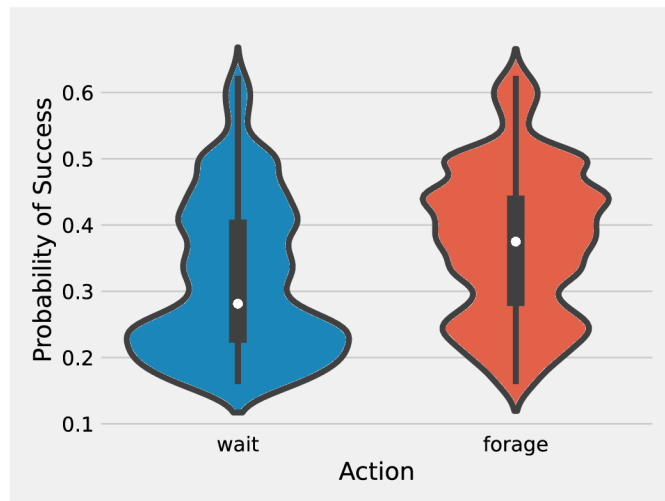

Figure 5.6: Violin plot of the probability of success for a particular action taken

7. **Violin plot of the probability of threat for a particular action taken**: This Violin

plot presents the distribution of the probability of encountering a threat for a specific action taken by the human participants. It helps us understand the potential risks associated with different actions and evaluate the participants' ability to mitigate threats based on their chosen actions.
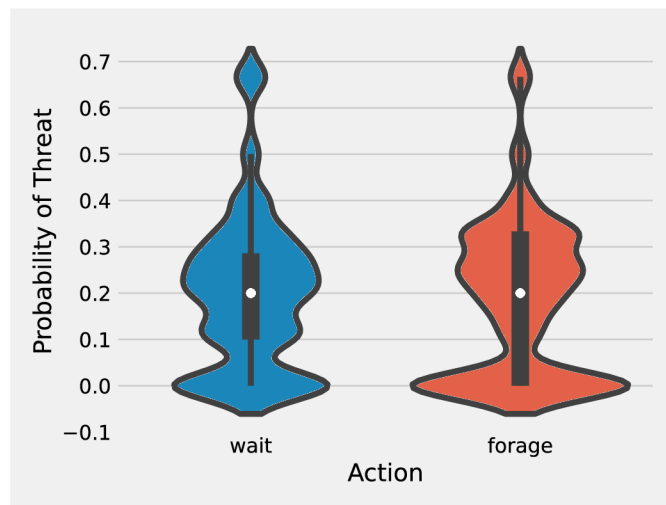

Figure 5.7: Violin plot of the probability of threat for a particular action taken

8. **Correlation plot of all independent features**:

Figure 5.8: Correlation plot of all independent features

By utilizing these data visualizations, we gain a better understanding of the game environment and the decision-making process of the human participants. These visual representations allow us to identify patterns, trends, and correlations within the data, enabling us to make informed interpretations and draw valuable insights from the behaviour and gameplay of the human participants.

5.3 Data Preprocessing and Feature Engineering

Minimal preprocessing was done to the collected data in general. For the decision tree, we prepared the data by performing a stratified train-test split with a ratio of 80:20. This resulted in approximately 8,400 state-action pairs in the training set. Whereas for training the GAIL model, we scaled all the features to a range of 0 to 1. Feature engineering plays a crucial role in extracting relevant information from the raw data and transforming it into meaningful features that can enhance the performance of machine learning models. In our study, we employed feature engineering techniques to derive insightful features from the data collected during the game and from the previous work from Korn et al [4]. These features can be categorized into two main groups: raw features and derived features.

5.3.1 Feature Engineering - Raw Features

1. **Number of days left**: This feature represents the remaining number of days in the episode. It provides information about the temporal aspect of the gameplay and allows us to analyze how the human participants strategize their actions based on the limited time available

2. **Number of life points left**: This feature indicates the remaining life points of the participants. It serves as a measure of their health status and reflects their ability to sustain themselves throughout the game. Analyzing this feature helps us understand how the participants manage their resources and make decisions to ensure their survival.

3. **Probability of threat encounter**: This feature quantifies the likelihood of encountering threats in the game environment. It is derived from the game's climate conditions and reflects the level of risk faced by the participants. Examining this feature enables us to assess how the participants consider the potential threats when deciding whether to forage or wait.

4. **Nutritional quality**: This feature represents the nutritional value of the food obtained by foraging. It is an important factor in maintaining the health and well-being of the participants. Analyzing this feature allows us to explore the participants' prowess to ascertain the dynamicity of the climate.

5.3.2 Feature Engineering - Derived Features

1. **Probability of successfully foraging**: This derived feature is calculated as the product of the probability of forest quality and the complement of the probability of threat encounter. It provides an estimate of the likelihood of successfully obtaining food while considering the risks involved. This feature offers insights into the participants' evaluation of the forest conditions and their ability to overcome potential threats when foraging

2. **Safe state**: This derived feature is defined as "True" if the number of life points left is greater than the number of days left, indicating that the participants have sufficient resources to survive without the need to forage further. Conversely, if the number of life points is equal to or less than the number of days left, the safe state is labelled as "False." This feature helps identify situations where participants can adopt a more conservative strategy and avoid unnecessary risks by waiting instead of foraging.

By incorporating these raw and derived features, we aim to capture various aspects of the gameplay and the decision-making process of the human participants. These features provide valuable insights into their resource management, risk assessment, and strategic thinking, enabling us to build more robust and interpretable machine learning models for analyzing their behaviour in the game environment.

5.4 Model Training and Evaluation

5.4.1 Model Architecture and Hyperparameters - Decision Tree

We trained the decision tree with a maximum depth of 5 and used entropy as the splitting criterion. Additionally, to ensure interpretability and minimize feature redundancy [1], we pruned the tree by setting the minimum weighted fraction of the sum total of weights of leaf nodes to 5% and the minimum impurity decrease required to induce a split to 0.02.

5.4.2 Model Architecture and Hyperparameters - GAIL

Training the GAIL model was challenging due to the stochasticity of the task. After numerous experiments, we settled on the architecture described as follows

- The generator architecture was based on a modified version of PPO as described in Stable-Baselines3 [2]. The diagram 5.9 visualizes the network architecture of the generator.

- The network architecture of the discriminator was kept simple - an MLP model with two hidden layers, each with 32 neurons.

- The batch size for training GAIL was set to 256

- Learning rate of the generator was set to 3x10-4 and the learning rate of the discriminator was set to 1x10-3 and the optimization algorithm for both was Adam

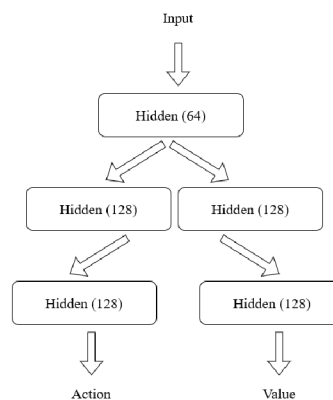- The number of updates per round for the discriminator was set to 2

Figure 5.9: Architecture of GAIL's generator

---

[1] Refer the documentation for Decision Trees here `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

[2] Refer the documentation for PPO here `https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html`

The generator was first pretrained for 50 epochs on the environment without relying on human behavioural data. This ensured that the model gets reliably trained and the mode collapse issue is reduced. After this, the learning rate for generator was reduced by a factor of 10, and the generator was trained in tandem with the discriminator for the remaining 250 epochs. the discriminator model reached 0.55 log loss and the generator model reached 0.43 value loss. The loss graphs of the two models can be viewed in 5.10 and 5.11
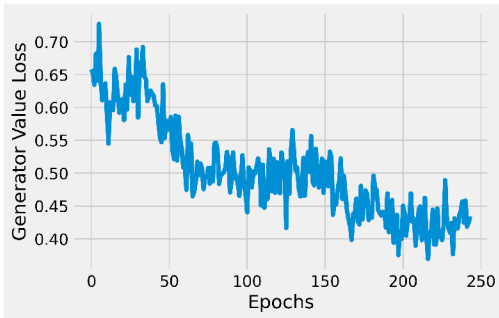
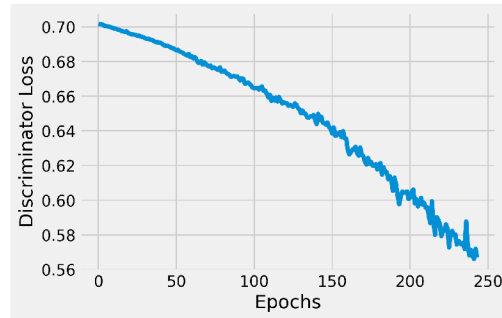Figure 5.10: Generator Value loss   Figure 5.11: Discriminator Logloss

5.5 Model Evaluation

Table 5.1 includes several approaches and algorithms, highlighting the following key points:

- The first four entries represent heuristic approaches that serve as a benchmark for comparison. These heuristics are utilized to establish a baseline, and the objective is for the trained machine learning models to outperform these approaches.

- The Backwards Induction approach is regarded as the best algorithm due to its possession of complete information about the environment. This approach achieves a survival rate of 28%, indicating the maximum achievable performance for an agent.

- Humans demonstrate a survival rate of approximately 21%. The goal for the trained models is to approach this level of performance, as it represents the capacity of human decision-making within the given problem domain.

- The Decision Tree algorithm trained with behavioural Cloning achieves a score of 16, which is considered a respectable performance. This indicates that the model is capable of capturing patterns and making informed decisions based on the observed behaviour from the training data.

- GAIL achieves a survival rate of 15%. Although GAIL's survival rate is slightly lower in comparison to Decision Trees, its ability to closely imitate the desired behaviour makes it a competitive approach as demonstrated by the Monte Carlo distance score.

Sr. No.	Approach	Algorithm	Survival Rate (%)	MCD
1	Heuristic	Always Wait	0	-
2	Heuristic	Always Forage	12	-
3	Heuristic	Hail Mary Play	14	-
4	Heuristic	Random Actions	9	-
5	Backwards Induction	Dynamic Programming	28	-
6	Human benchmark	-	21	-
7	behavioural Cloning	Decision Tree	20	8.92
8	Inverse RL	GAIL	16.5	8.83

Table 5.1: Comparison of approaches

5.6 Model Explanability

5.6.1 Explaining Decision Tree

After training the decision tree on the human participants, we can visualize it in order to describe the collective human strategy as seen 5.12. As decision trees are inherently explainable, they are extremely useful tools when it comes to explainable ML
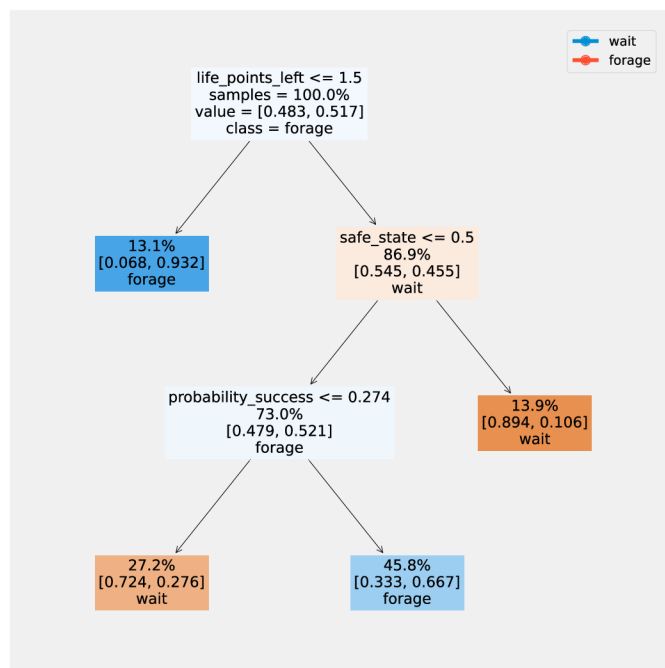

Figure 5.12: Decision Tree trained on human behavioural data

The decision tree shows two main sub-strategies:

- Players are more likely to forage if they have very low life points, indicating that they are on the verge of death. This suggests that players are willing to take risks in order to increase their chances of survival.

- Players are more likely to wait if they have enough life points to wait out the episode, also known as being in a safe state. This suggests that players are more risk-averse when they have a higher chance of survival.

After these two main sub-strategies, the decision tree becomes less clear. However, one insight is that players seem to depend on the probability of success when making their decision. For example, if the probability of success is high, players are more likely to forage, even if they have a safe amount of life points. This suggests that players are more likely to take risks if they believe that they are likely to be successful.

5.6.2 Explaining GAIL

The decision tree trained on synthetic data from the GAIL model reveals two main sub-strategies similar to the strategy of human participants: forage when low health and wait when safe. The GAIL model also considers other factors which might be important to survive (and to imitate human behaviour), such as low probability of threat encounters, high probability of success and higher number of days left.

This finding suggests that GAIL agents are capable of learning complex strategies that are similar to those used by humans. This has important implications for the development of artificial intelligence agents that can interact with the environment in a safe and efficient manner.
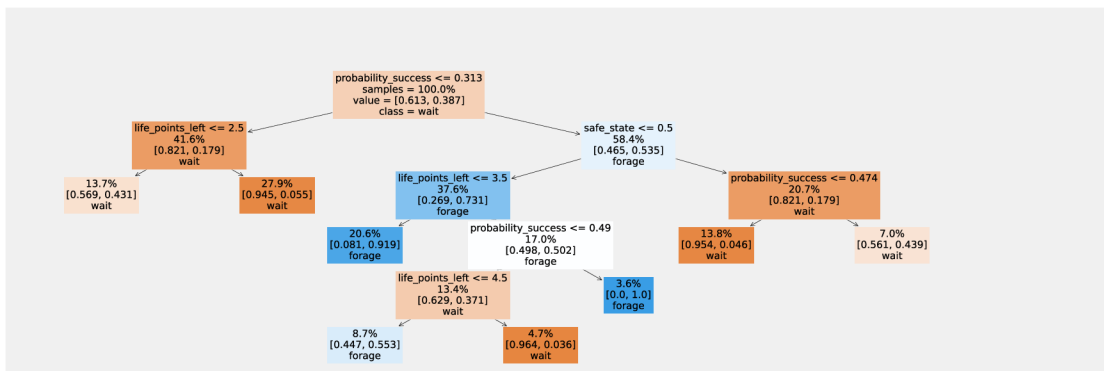

Figure 5.13: Decision Tree trained on synthetic data obtained from GAIL

The Shapley values in figure 5.14 represent the marginal contribution of each feature to the GAIL agent's decision-making process. The top three features are life points left, days left, and probability of success. As can be seen, the life points left have the largest Shapley values, indicating that it is one of the most important features for the GAIL. Days left and probability of success are also important features, but they have a marginally smaller impact on the decisions.
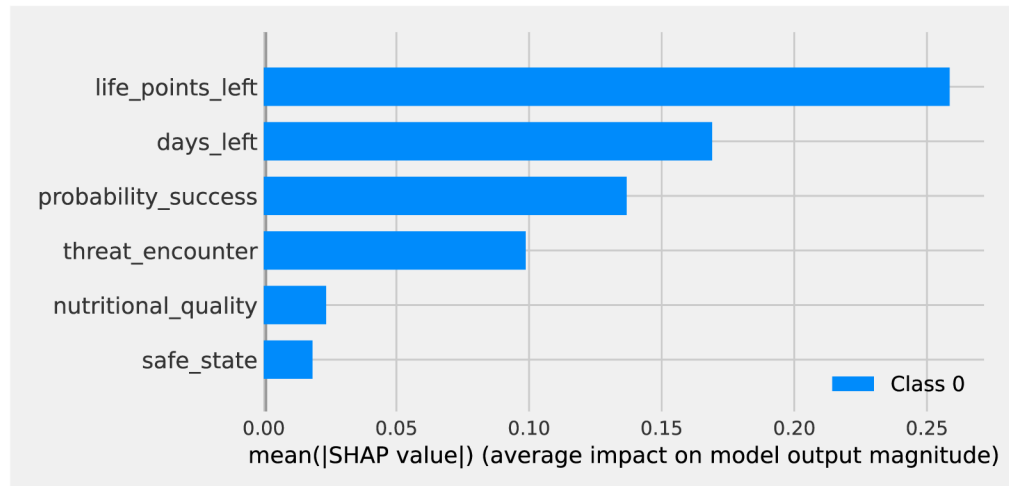
Figure 5.14: Shapley analysis of GAIL

6 Discussion

The discussion section encapsulates the outcomes and implications derived from the study's application of the Reinforcement Learning (RL) Paradigm to assess human decision-making capabilities within the intricate framework of a challenging game environment. The utilization of this paradigm yielded insightful revelations regarding the diverse array of strategies employed by players. It became evident that the likelihood of survival was notably influenced by stochastic elements, emphasizing the necessity for a larger participant cohort to enhance the robustness and precision of population-level estimations.

One of the key takeaways from the study was the efficacy of Decision Trees, specifically when equipped with well-crafted derived features. These Decision Trees exhibited a dual advantage: not only did they provide understandable and interpretable explanations of player strategies, but they also demonstrated a level of performance that could be considered on par with the more complex GAIL model. The Decision Tree's capacity to distill intricate decision-making dynamics into comprehensible insights is noteworthy and lends itself to practical applications where transparent explanations of strategic choices are imperative.

Conversely, the Generative Adversarial Imitation Learning (GAIL) model showcased a distinct strength in capturing the underlying behavioral intricacies of the participants. However, this heightened capacity was accompanied by a trade-off in terms of immediate interpretability. The GAIL model's intricate nature necessitated post-hoc efforts to expound upon its outcomes, which could potentially limit its real-time applications in scenarios requiring prompt insights.

An innovative avenue emerged through the process of amalgamating the GAIL model's findings with the Decision Trees, demonstrating a promising approach to balance the dual aspects of performance and explainability. This harmonious synthesis offered a potential bridge between the complex modeling techniques and the need for transparent understanding of decision-making dynamics.

In conclusion, the study not only shed light on the multifaceted nature of decision-making strategies in challenging environments but also highlighted the strengths and trade-offs associated with different modeling approaches. The combined insights from Reinforcement Learning, Decision Trees, and GAIL techniques contribute to a more comprehensive understanding of human decision-making, paving the way for informed applications in various fields.

7 Conclusion and Future Work

This study investigated human decision-making behaviour in a game-based setting and demonstrated the potential of GAIL for imitating human behaviour in a tabular setting. The study also that the application of the RL Paradigm effectively evaluated human decision-making capabilities in the challenging game environment. However, players exhibited diverse strategies, and survival odds were influenced by chance, necessitating a larger sample size for better population estimation. The results also showed that GAIL is capable of learning complex strategies that are similar to those utilized by humans, but it lacks immediate interpretability. Distilling GAIL through Decision Trees improved interpretability, presenting a promising approach for balancing performance and explainability.

Evident ways of extension of the work would be to first gather more informative data by prompting players to explain their decisions, providing valuable context for deeper insights into decision-making processes. One line of work which is gaining traction in the RL community is to train the models in a framework called Reinforcement Learning from Human Feedback, where the data collected from the participants is the explicit ranking of the model's actions instead of supervised data. We attempted to transform the supervised data to a feedback mechanism, which showed promising preliminary results, but a more thorough exploration is necessary.

Secondly, from the model perspective, investigating GAIL's instability is a must. The architecture of adversarial training is inherently unstable as mentioned in the GAN literature, but we attempted to remedy it by pretraining the generator on the game environment. Another suggestion would be to carry out trials with diffusion models, which is the current state-of-the-art techniques for generative modelling for computer vision. Although there have been attempts to replicate their success for RL, our usecase is a bit more nuanced (aka RL for tabular data) and so we decided not to test it for the scope of the thesis. Another cutting-edge work is to utilize foundation models as a baseline pretrained model and finetune them on the downstream task. We aim to examine foundation models for our task in the near future. From an explainability perspective, one extension could be to explore soft decision trees.

Thirdly, the game could be modified to encompass multi-agent learning. This would uncover complex decision-making dynamics in group settings and would help understand agent interactions and their impact on outcomes. This is our current focus and preliminary work has been described here (link to slides).

By addressing these future research directions, we can gain a deeper understanding of decision-making in game environments and advance the development of more interpretable and human-aligned AI systems.

Bibliography

[1] D. D. Bourgin, J. C. Peterson, D. Reichman, S. J. Russell, and T. L. Griffiths, "Cognitive model priors for predicting human decisions," in *International conference on machine learning*. PMLR, 2019, pp. 5133–5141.

[2] A. R. Bland and A. Schaefer, "Different varieties of uncertainty in human decision-making," *Frontiers in neuroscience*, vol. 6, p. 85, 2012.

[3] O. Alagoz, H. Hsu, A. J. Schaefer, and M. S. Roberts, "Markov decision processes: a tool for sequential decision making under uncertainty," *Medical Decision Making*, vol. 30, no. 4, pp. 474–483, 2010.

[4] C. W. Korn and D. R. Bach, "Heuristic and optimal policy computations in the human brain during sequential decision-making," *Nature communications*, vol. 9, no. 1, p. 325, 2018.

[5] L. M. Rigoli, G. Patil, H. F. Stening, R. W. Kallen, and M. J. Richardson, "Navigational behavior of humans and deep reinforcement learning agents," *Frontiers in psychology*, vol. 12, p. 725932, 2021.

[6] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[7] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees–crc press," *Boca Raton, Florida*, 1984.

[8] G. Blanc, J. Lange, and L.-Y. Tan, "Provable guarantees for decision tree induction: the agnostic setting," in *International Conference on Machine Learning*. PMLR, 2020, pp. 941–949.

[9] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," *Advances in neural information processing systems*, vol. 31, 2018.

[10] M. Pan, W. Huang, Y. Li, X. Zhou, and J. Luo, "xgail: Explainable generative adversarial imitation learning for explainable human decision analysis," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1334–1343.

[11] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.