Demonstrating & Implementing the Hidden Markov Model in Opponent Modeling for Two Player Zero-Sum Games

Sina Azartash

Final Project Paper

Whiting School Of Engineering, Johns Hopkins University

EN.605.724.81.SP22 Applied Game Theory

Dr. Richard Cost

May 3, 2022

Abstract

This paper discusses Opponent Modeling(OM) applied to Game Theory. OM is the capability to to learn about the strategies of another player. In short, OM uses prior knowledge and/or observations to model the behavior of an opponent. This model can be used to predict the behavior of other players, exploit those predictions, and defend against exploitation. Here, we explain some theory in the OM field and then give some practical examples. Specifically, we summarize research, introduce models and architectures, and discuss challenges and limitations in OM. Also, we discuss applications of OM, demonstrate a fundamental algorithm applied to a game, discuss our results, and give some ideas for future work. Among OM, our application scope is two-player, zero-sum games. Additionally, we limit our focus to the generative model domain specifically, Hidden Markov Model(HMM) and its algorithms. We use HMM to infer the most likely strategy an adversary holds given a sequence of actions. To demonstrate HMM, we analyze the Forward Algorithm applied to the Iterated Prisoner's Dilemma. Next, we present a detailed walkthrough of the Forward Algorithm with illustrations to explain the fundamental ideas behind HMM in OM. Lastly, we discuss how this model can be extended for further capability, and its ramifications.

Table of Contents

Introduction

The Ideal Design

The most ideal implementation of Opponent Modeling(OM) has some very powerful capability. The objective of OM is to learn the strategy of an opponent, to understand what another player is planning and thinking, and to learn about the person's style behind the strategy.

Imagine playing a game of chess against an AI with OM capability. The AI scrutinizes your every move to improve its understanding of what type of player you are. At each turn, the AI attempts to improve its model about you by comparing your moves against the moves of other players in a database of millions of previous chess games. Just after a couple turns, the AI begins to see patterns in your play that you might not even be aware you are projecting. These patterns are used to fit you into an archetype. That archetype is used to anticipate your behavior and predict how you will react to various situations in the game. The AI might test and bluff several different strategies to see how you will react in order to further refine its archetype and to search for weakness in your play. Several turns later, the game has been fiercely, evenly matched and

you decide to attempt to trap the AI's king with a bishop and a knight. You start moving your chess pieces to the right location to set up your strategy. When you finally execute your plan, the AI presents you with a trap. The movement of the bishop and knight created an opportunity to eliminate the chess pieces guarding your king. You instantly lose the game.

The above hypothetical example illustrates the power of OM. Collecting knowledge about an adversary game is an invisible, quiet process, but can produce very real results. Throughout the entire game, the AI was studying the player and gaining advantage in the game by building a model. The other player does not see any indication of this other than a few seemingly random test strategies to defend against. The first advantage gained by this archetype model is to search for where the player's strategy has taken risk. Almost all strategies that deviate from the long-term standard carry some type of risk. The AI then capitalizes on the opportunity presented by the player's risk. The second advantage gained by this archetype model is a probabilistic estimate of possible strategies in the future. The AI combined the archetype's model with the observed sequence of actions related to setting up a strategy to recognize what strategy the player was preparing for after a couple turns. This allowed the AI to react to the other player's strategy earlier and begin a counter strategy before the player had even finished setting up their strategy. The third advantage gained by the archetype model is understanding which strategy is most effective against the player. There may have been several possible counter strategies against the player's plan. The AI compares the player's reactions and style in this game with its database of reactions and style in other games to generate a Bayesian probability distribution. In other words, given the archetype of the player and given the matching archetypes in the database, the probability of each possible counter strategy is calculated. Lastly, the fourth

advantage is defense against exploitation and hiding risks. The AI can play strategies in which the risk has the lowest probability of being capitalized by the player. For example, the AI knows that the player is less likely to respond to an attack when the player is just about to execute a plan. This capability allows the AI to take more risk, but to hide their risk.

Ramifications

Situations in which AI predicts opponent behaviors based on archetype analysis does not just apply to chess, but can virtually apply to any game. Although the above example is hypothetical, it is not hard to imagine that this technology is already being used. Powerful organizations that can afford to hire teams of AI and Game Theory experts to work together to produce applications to virtually any multi-agent domain. Some examples include cyber-security, stocks, sports, distributed systems, recruiting, business strategy, realestate, customer service, and of course the military (Nashed and Zilberstein 2022a). As the developments in this field continue and applications extend to consumer facing applications and the social space, the potential of unintended side effects grows. That is, the person wielding such AI has significant power over ordinary people. AI can learn preferences of those ordinary people and find the most optimal strategy suited towards those preferences.

As such, the motivation for this paper is to raise awareness about OM. Future developments in OM have the potential to contribute to several great advancements to improve the quality of life, but may give birth to significant new societal problems. Regardless of whether society is ready, advancements in OM are coming and are only accelerating. The objective of this paper attempts to address “what is OM and how does it generally work?” This paper explains

fundamental concepts, introduces modern scientific research, discusses modern day challenges and limitations, and demonstrates its use for application to simple games.

Problem and Scope

The building of an archetype of the opponent and updating throughout the game represents an invisible growing utility in games. There are four such main advantages of OM applied to game theory. First, AI uses the archetype model by OM to learn where the player's strategy deviates from the long-term standard to exploit the risk. Second, the AI can respond to player strategies even before they are executed and use the extra time to build a trap. Third, the AI will play the strategy that has the highest probability of causing the player to struggle. Fourth, the AI can hide their risks and take risks when the player is least likely to respond.

The aforementioned OM AI would actually be very difficult to implement. Creating such an AI using state-of-the-art advancement in OM technology would likely take a few years of development. Therefore, explaining and demonstrating the capabilities of such a system are not discussed in this paper. Instead, we focus specifically on strategy matching. We demonstrate how an OM takes a given sequence of decisions and matches it to candidate strategies.

OM is in fact a large area consisting of many different approaches and many different classes of algorithms. Surveying just a subsection of the OM field has frequently been chosen as a PhD dissertation topic. We choose to focus on Hidden Markov Models (HMM) because it is a popular and fundamental approach to solving OM problems. The purpose of this study is not to

statistically prove the effectiveness of HMM's. Instead, we analyze its use in simple games. Furthermore, we specifically focus on two-player, zero sum games because that is generally the easiest games to run. Lastly, the purpose of this paper is not to demonstrate scalability and robustness in HMM. Instead, we evaluate the different HMM algorithms and discuss how HMM compares to the other OM methods.

OM Capability and Limitations

OM seeks to understand the adversary's strategy. Samer B. Nashed and Shlomo Zilberstien define OM to be "the ability to use prior knowledge and observations in order to predict the behavior of an opponent" (Nashed and Zilberstein 2022b). OM builds an archetype model of the adversary to see how it compares with other adversaries and how its previous moves compare with its current move. The structure of the model varies, but generally consists of a list of attributes that describe the adversary decided through classification and clustering.

In the context of zero-sum games, OM is used when game theory practitioners are interested in finding holes in adversaries' strategies. The model generally serves two purposes for the protagonist: to exploit when the adversary takes risk in the most effective way possible and to prevent being exploited when taking a risk. In order to detect when an adversary is taking risk, strategy is suboptimal and then plays the most optimal move to exploit the adversary's suboptimal move. This requires that observing the adversary's moves is possible and that the observed information provides meaningful information that can be used for exploitation. Encountering uncertainty or noise complicates and can undermine OM because it lowers the confidence that the adversaries moves are related to its intentions(Mandziuk 2010).

Consequently, algorithms that better handle uncertainty and noise in OM are an area of frequent research.

OM in advanced modern applications is used to develop models for Partially Observable Stochastic Games (POSG). POSG describes the dynamic scenarios in which a game of multiple players contains stochastic events, partial information of players, partial information of the environment, and no fixed horizon (Horák and Bošanský 2019). These situations are a modern research challenge because the complexity of the problem often makes the algorithms intractable, or determining the payoff function has too much uncertainty associated with it. Other modern challenges include continuous actions, stochastic actions, concurrent actions, sparse payoff structure and decentralized coordination of multiple agents. OM is often used as a solution to reduce computational complexity. It uses data from past experiences, online datasets, and similar pre-built models to create archetype models of the agents in POSG. This model is then used as a priori information to guide the development of other game theory algorithms (Nashed and Zilberstein 2022b).

Alternative OM Architectures and Algorithms

OM is a large field with a diversity of architecture designs and implementation patterns with the goal of computing the best response. In general, the numerous techniques differ on three principal axes: the method in which data is collected, the learning algorithm used, and the level of decision making abstraction(Nashed and Zilberstein 2022b). OM architectures use different methods for extracting and processing observation data that represents the adversaries behavior.

The learning component used is usually one or an ensemble of Support Vector Machines, Decision Trees, Neural Networks, and Multi-Agent Reinforcement Learning. The level of abstraction varies from approximating low-level optimal decisions for a single agent to high-level optimal decision for collective population behavior. In a nutshell, it involves a trade-off in which the choice of the OM architecture depends on the quality of the variables in the game, is constrained by the data and preprocessing limitations, is constrained by the quality of expert domain knowledge, and needs to output significant, meaningful action selections. Furthermore, several OM methods are often swapped, modified, and combined together in an ensemble or a complementary pipeline.

There also exists a large diversity of learning algorithms that are used by OM depending on the goals of the OM architecture. OM algorithms generally fall into three categories: Discriminative Role or Strategy Classification (DRSC), Goal-Based Generative Models (GBGM), and Policy Approximation (Nashed and Zilberstein 2022c). DRSC is a supervised approach that classifies each agent into several classes. Those classes are used to make nuanced predictions about future behaviors. DRSC algorithms include Support Vector Machines, Case-based Reasoning, Expert Systems, and Metrics, and Game Theoretic Approaches. Next, GBGM reports how agents achieve their goals using the likelihood distribution of agents playing a strategy given an action and the probability distribution of a strategy given the action. Other algorithms include Hidden Markov Models (HMM). Bayesian Networks and black-box simulation or expert knowledge. Finally, Policy Approximation is a reinforcement learning technique which approximates the strategy of an opponent based on a defined states space and action space. The approximated strategy, also called the policy, is used to predict sequences of

opponent actions. These algorithms include decision trees, Abstract Markov Models, Deterministic Automata, Deep Neural Networks, and Partially Observable Markov Decision Processes. In this paper, we focus on HMM which belongs to the generative model approach.

Hidden Markov Models

HMM algorithms represent the goals of agents in multiplayer games and delineate player intention from player action. The term 'hidden' refers to events that are not observable such as actions done in secret or plans in the player's thoughts. HMM's are similar to bayesian networks in that it involves iteratively updating conditional probabilities. Furthermore, HMM's all use the markov decision process which allows recurrence relations when calculating current state. HMM's are characterized by three fundamental problems: likelihood computation, decoding, and learning. Likelihood Computation is done by the Forward Algorithm, Decoding is performed by the Viterbi Algorithm and the HMM model learns by the Forward-Backward Algorithm.

The Markov Property is fundamental to HMMs. In the Markov Chain, the next state depends only on the current state, not the complete sequence of states. Each transition to a new state has a probability associated with it and the sum of the transition probabilities for a given state will sum to one(Keselj 2009). The probabilities of each state can be displayed by an adjacency matrix. The transition matrix specifically describes the adjacency matrix that describes the hidden state. The advantage of using HMMs is that the intentions of the adversary can change during a game and can even fluctuate. The opponent's intentions constantly changing severely complicates the problem, but HMM easily handles this by modeling those changes

through the transition matrix. Meanwhile, the emission matrix is the term used to describe the adjacency matrix of probabilities learned from observing the change in states. The transition matrix is useful in that if the transition matrix is multiplied by a vector representing the states, then that outputs a vector of probabilities of states. When this future state is multiplied by the transition matrix repetitively this eventually leads to stationary distribution. The stationary distribution describes a vector distribution of state probabilities that no longer change; that is, they are at equilibrium.

In OM, the hidden states can represent the model of the adversary, while the observed variables represent the adversaries' actions which can be used as a proxy to predict the hidden states. HMMs attempt to maximize the conditional posterior probability of an observed state given the hidden state. Simply put, HMM in OM attempts to find the most optimal action to take from the given hidden state. The goal is to calculate this posterior probability and it is not known because initially the hidden states are not given. Consequently, Bayes Rule is used to reverse this and split the posterior probability into the likelihood of the hidden state given the observed state multiplied by the probability of an observed state. This can be calculated because the observed states are known.

Central to the three core HMM process, the current hidden state is calculated based on the observed state in a chain. The hidden state is represented as the conditional probability of an observed state given the previous hidden state and is calculated in a chain. In this way, at each state in the chain, the likelihood of the hidden state is calculated at each level. Going down the chain and reaching the end of the chain is the initial hidden state. To calculate this, the stationary

distribution serves as an approximation. The core HMM process is described more in detail using the Forward Algorithm walkthrough application below.

Modern Research Endeavors

Ganzfried and Sandholm developed Deviation-Based Best Response (DBBR) on Texas Hold'em. Texas Hold'em is an extensive form game that is difficult to compute so they use abstraction. DBBR works by comparing the opponent's strategy with a precomputed approximated equilibrium strategy. Deviations between the strategy of the opponent and the precomputed optimum strategy were exploited. The best response was continually updated to counter the opponent's strategy in real time (Ganzfried and Sandholm 2011).

Burkhard Von der Osten, Kirley, and Miller recently introduced a Theory of Mind framework for multi-agent systems. Theory of Mind (ToM) refers to OM that is capable of predicting how its opponent thinks, uncovering the opponents rationale for decisions and predicting how the antagonist would view the protagonist. Some ToM experiments take this even farther by focusing on how the protagonists think that the antagonist is being viewed by the protagonist. Briefly, an abstract model was used to predict actions of an adversary using nested beliefs and this was applied to all agents. Then, a stereotyping mechanism segmented agents based on similar behavior (Osten et al. 2017).

Knegt, Durgan, and Wiering used OM via Deep Reinforcement Learning. A neural network was created to predict which move a player will most likely make when in a specific state based on opponent history. Knegt et. al created a probability distribution for each decision to represent the model of the opponent. After generating a likely move by the opponent based on

the model, next following future reactions by the opponent were considered. Monte Carlo simulations were conducted to approximate how the expected payoff of a specific decision made against an opponent would compare against other decisions at the end of the game (Knegt et al. 2018).

Explaining HMM Decision Process on the Iterated Prisoner's Dilemma

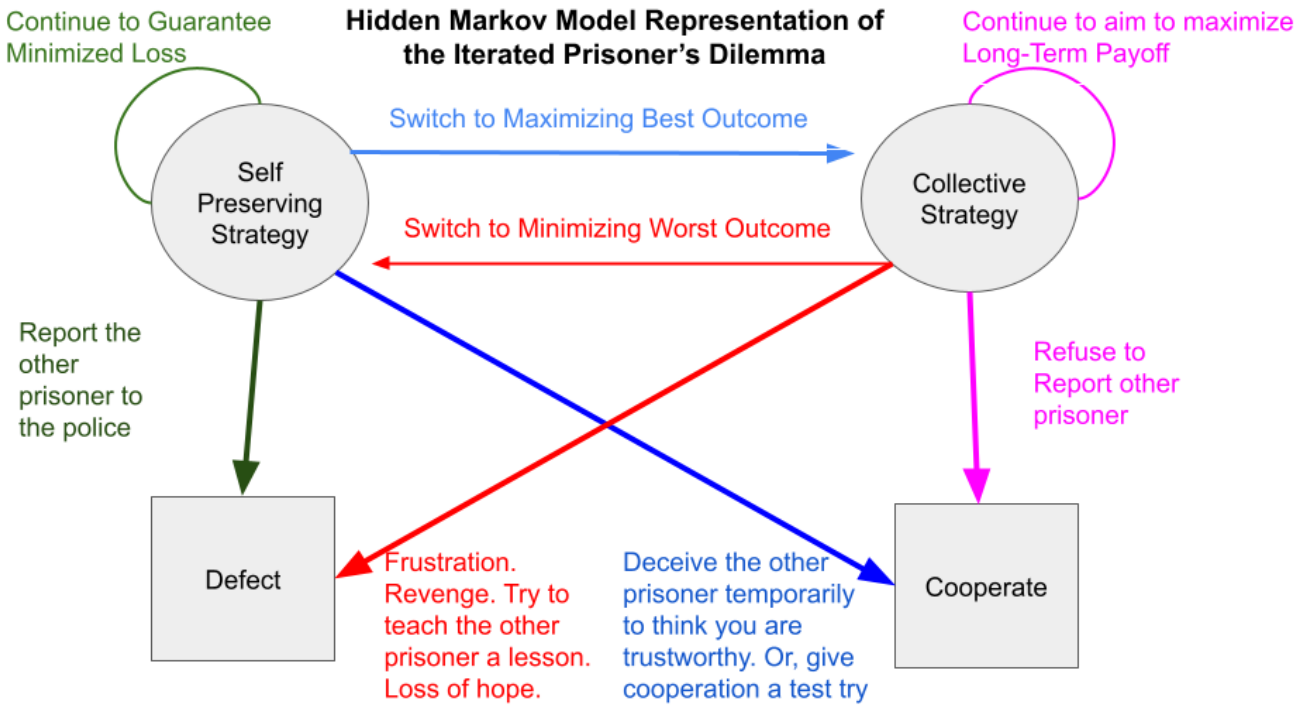
The Forward Algorithm

The Forward Algorithm(FA) uses the HMM principles for OM using dynamic programming. Dynamic programming is a method for reducing the time complexity of an algorithm by avoiding redundant calculation. FA aims to find the probability of a sequence of opponent actions given the hidden strategy they are using. In other words, a probability value is assigned to each predicted sequence of opponent actions. We can never truly know the intentions of our opponent and therefore FA used in OM and therefore will have several errors. The goal in FA is to achieve a significant minimizing of those errors.

We demonstrate how the Forward Algorithm achieves minimized errors on the iterated prisoner's dilemma. The iterated prisoner's dilemma is a classic, standard game in which two prisoner's are caught by the police and are given the option to be set free if they rat on the other prisoner. If a prisoner defects on the other prisoner, the prisoner is set free, but the other prisoner suffers a much greater jail time. If both prisoner's defect, both prisoners suffer increased jail time. However, if both prisoners cooperate by choosing to not rat each other out, then they suffer only a minor sentence. The strategy to receive the lowest sentence would be to try to coordinate cooperation, but the player cannot talk. Since players cannot count on the other player being to

cooperate, players are better off defecting. As a result players are stuck between minimizing loss by defection or maximizing gain by cooperation.

Below, the picture demonstrates the prisoner's dilemma through the iterated prisoner's dilemma. The advantage of the HMM is that the player's strategy does not always match its actions. In other words, players sometimes choose moves contrary to their own actions. HMM attempts to distinguish the adversary's true strategy from the observed actions.



Algorithm Walkthrough

FA aims to find the probability an opponent would play a sequence of actions if they were using a specific hidden strategy. Using the example of the iterated prisoner's dilemma, let us say that our protagonist agent observes the following sequence in the adversary: defect,

defect, then cooperate. OM aims to uncover the opponent's hidden strategy inorder to predict the opponent's next action. Let us say that the adversary has either a selfish strategy or a collective strategy. To determine the probability of this sequence of actions, we multiply the probability of a defect in state 1 by the probability of defect in state 2 by the probability of cooperation in state 3. The probability of each action can be denoted as Y0, Y1, and Y2 as shown below.

$$P(Y = Y_0, Y_0, Y_1) = ?$$

Y_0 indicates a defection and Y_1 indiciates cooperation. As a reminder, the sequence of these observed states are used to calculate the probability of candidate strategies. The candidate strategy with the highest probability can be inferred to be the hidden strategy the opponent is using.

However, the probability of the action sequence cannot be calculated because the probability of a single action is not currently known. In order to calculate this, we multiply the probability of an action given the strategy at the current state multiplied by the probability of a strategy at the current state. This is illustrated in the picture below where X_0 is the selfish strategy and X_1 is the collective strategy. As a reminder, X refers to the hidden states and Y refers to the observed states. The picture communicates that the probability of a selfish strategy is being multiplied by the posterior probability of defect given a selfish strategy at state #1.

$$P(X_0)\,P(Y_0|X_0)$$

To calculate the probability of actions for the other remaining states we need to follow the aforementioned hidden markov process using the transition matrix. That is, we need to consider the possibility the adversary had a change of heart and transitioned their strategy in the next stage. In stage 1, defect was played and in stage 2 since defect was played again. For this reason, we may assume that the strategy of the adversary did not change. Therefore, the transition probability would be the new X_0 given the old X_0. Below, we see that the probability of selfish strategy is being multiplied by the conditional probability of a defection given the selfish strategy multiplied by the transition probability of the selfish strategy given the selfish strategy.

$$P(X_0)\ P(Y_0|X_0)\ P(X_0|X_0)$$

We might be tempted to assume that the strategy did not change. However, in reality, there is the possibility the adversaries strategy did change and the adversary chose to play an action in contrast to their strategy. There can be many game theoretic reasons for doing so such as deception, acting out emotionally, making threats, or being threatened. FA has to keep track of both possibilities. Below, we show how the probability calculation would look like if we did change strategies. The transition matrix would switch to X_1 given the old X_0.

$$P(X_0)\ P(Y_0|X_0)\ P(X_1|X_0)$$

Next, state 2 is calculated for both strategy assumptions. The probability that the opponent decided to defect given that he stayed on a selfish strategy is shown below.

$$P(X_0)\ P(Y_0|X_0)\ P(X_0|X_0)\ P(Y_0|X_0)$$

Using this logic, the probability that the opponent decided to defect given that he switched to a collective strategy is also exemplified below. We previously mentioned that FA needs to keep track of all possibilities. However, we do not need to calculate the probability of cooperating given each strategy because the cooperation action was simply not taken.

$$P(X_0)\ P(Y_0|X_0)\ P(X_1|X_0)\ P(Y_0|X_1)$$

In stage 3, we did see the adversary play the cooperate observed action. This action could have occurred because either the adversary had switched to a collective strategy or the adversary already was on a collective strategy. Below shows: defect given selfish → switch to collective → defect given collective → stay collective.

$$P(X_0)\ P(Y_0|X_0)\ P(X_1|X_0)\ P(Y_0|X_1)\ P(X_1|X_1)$$

Below shows: defect given selfish → stay selfish → defect given selfish → switch to collective.

$$P(X_0)\ P(Y_0|X_0)\ P(X_0|X_0)\ P(Y_0|X_0)\ P(X_1|X_0)$$

Above, shows the possibility of a collective strategy and then switching to selfish again. Or, perhaps the whole time the adversary was actually playing a collective strategy and chose to defect the first two times because he/she was afraid or nervous. All these scenarios show how the number of possibility sequences that need to be considered grow exponentially. Since there were

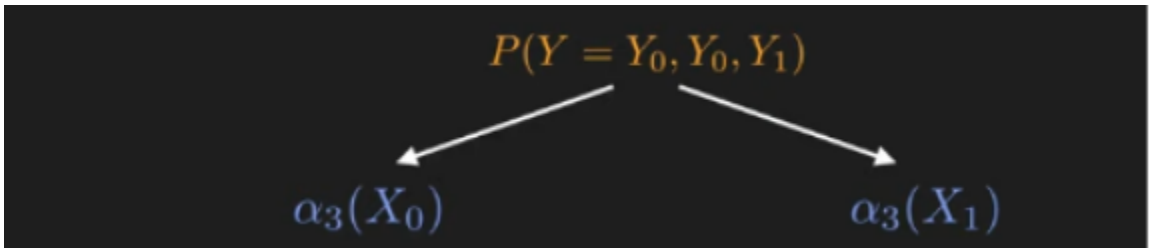
3 stages and 2 strategies, we have $2^3 = 8$ total possible probability calculations. One of those final example calculations is shown below. Below exemplifies defect given selfish → stay selfish → defect given selfish → switch to collective → cooperate given collective.

$$P(X_0)\ P(Y_0|X_0)\ P(X_0|X_0)\ P(Y_0|X_0)\ P(X_1|X_0)\ P(Y_1|X_1)$$

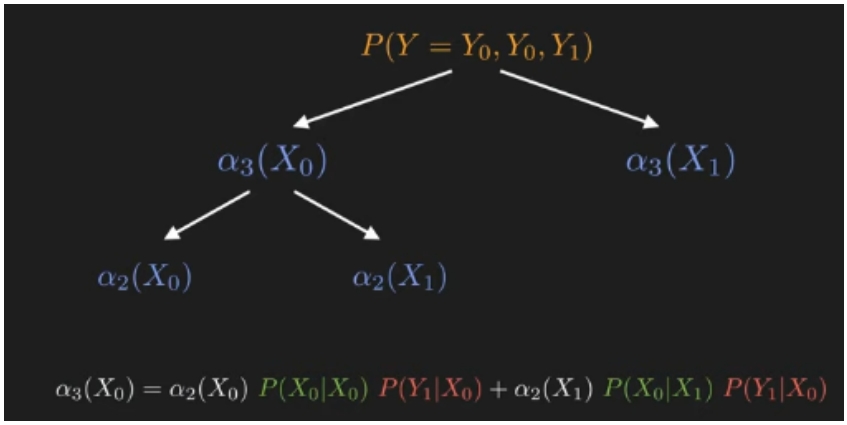
Recurrence Relations

The generalized formula for calculating the number of probability calculations is strategies to the power of the number of states (policy $^{\text{number of states}}$). The significance of this formula is the time complexity of this algorithm becomes $O(n^T)$. For each observation in the sequence, there is an exponential increase in calculations. To make this calculation tractable, the number of calculations can be reduced by recurrence relations.

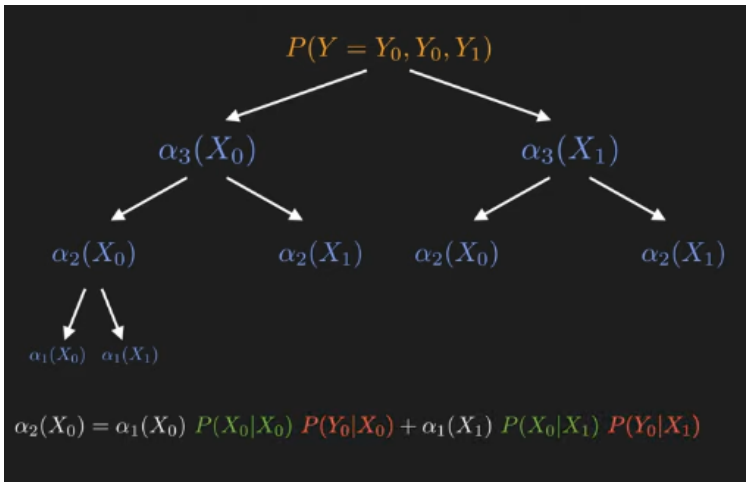
In recurrence relations, the nth term of a sequence can be calculated based on a function of the n-1 term(the previous state). Fortunately, this was made possible because of the Markov Property and is why we chose the HMM model. This is precisely the reason why the problem was orginally constrained to have the current state depend on only the previous relation. In the image below, we illustrate splitting the 8 possible sequences based on the final state. $\propto_3(X_0)$ is a mathematical representation that indicates that the third state is defect. The subscript indicates that this is the third state.



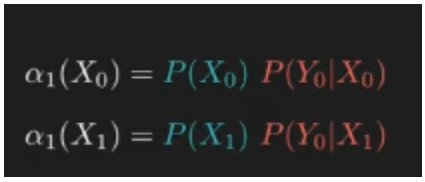
Together $\propto_3(X_0)$ and $\propto_3(X_1)$ can be used to calculate the probability of the actions given the strategy. In turn, $\propto_3(X_0)$ can be calculated using the recurrence relation: $\propto_2(X_0)$ and $\propto_2(X_1)$. Below, we show that $\propto_3(X_0)$ can be calculated using the action and strategy of state 3. In other words, $\propto_3(X_0)$ is the marginal probability of staying selfish(X_0) and cooperating for stage 3 and switching to selfish (X_1) from a collective strategy and cooperating (Y_1) for stage 3. The color red indicates the observed actions as the emission matrix while green indicates the strategy we are inferring as the transition matrix.



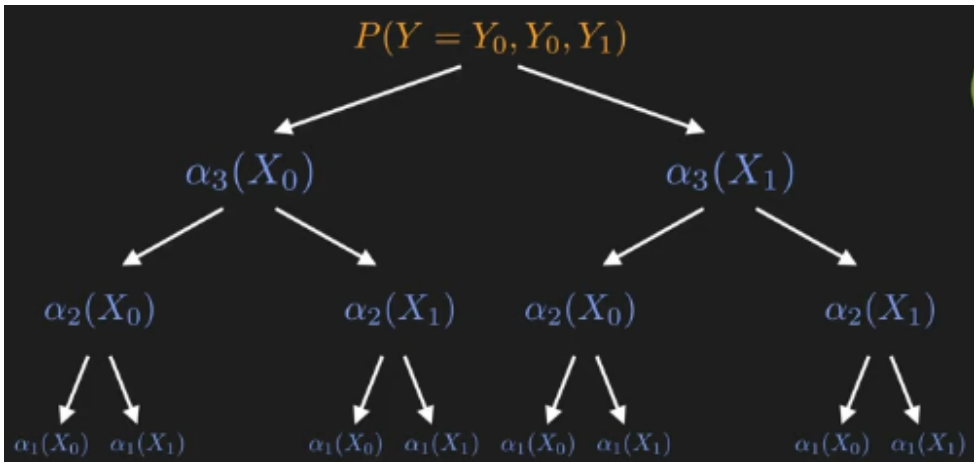
In this way, the states are recursively in a tree which creates opportunity for dynamic programming. Likewise, below, we illustrate $\propto_2(X_0)$ can be calculated using the marginal probabilities of $\propto_1(X_0)$ and $\propto_1(X_1)$.



Since $\propto_1(X_0)$ does not have a previous state, it cannot be further decomposed. For that reason, the state is approximated using the stationary distribution. As mentioned earlier, the stationary distribution is just the transition matrix infinitely multiplied against itself until convergence of values. Below, the color blue represents the stationary probability for state 1 and red represents the observation of state 1 given the two possible strategies.



Once the problem is fully decomposed into a complete tree, we can see that the same equations show up repetitively several times. We can avoid unnecessary and redundant calculation by just plugging in our previous value for $\propto_1(X_1)$ and $\propto_1(X_0)$. Below, we see that we can also plug in values for previous calculated $\propto_2(X_0)$ and $\propto_2(X_1)$.



General HMM Formula

Altogether, the general form of an alpha term is $\propto_t(X_i)$ where t is the state number and i = the index of the strategy used. How to calculate the alpha term of any number of states and any number of strategies per state is shown in a formula below. In order to calculate an alpha term, we take the product of the previous alpha term (yellow), the transition probability (green) from the transition matrix, and posterior probability of observed action given a hidden state (red) from the emission matrix. We multiply every action for every state denoted by the posterior probability. Finally, index j is the strategy of the previous state. In the same way that $\propto_1(X_1)$ added to $\propto_1(X_0)$ equals $\propto_2(X_0)$, we have to sum all the previous strategies of the previous state to get the current state.

$$\alpha_t(X_i) = \sum_{j=0}^{n-1} \alpha_{t-1}(X_j) P(X_i|X_j) P(Y^t|X_i)$$

The formula for the final calculation is shown below. The probability of the observed actions is simply the sum of the alpha terms.

$$P(Y^1, Y^2, ...Y^t) = \sum_{i=0}^{n-1} \alpha_{t-1}(X_i)$$

Inferring a Strategy

Above, demonstrated how to calculate the probability that a sequence of actions is played given hidden adversaries intentions that can change. The assumption made was that the transition probabilities were known, but that is most likely not the case. Knowing the transition probabilities is knowing the probability an adversary has a change of heart and knowing the probability the adversary decides to keep to the same strategy. To solve this problem, we compare several candidate transition matrices. We would run the entire FA algorithm again on each transition matrix and then compare the results. Each transition matrix represents a strategy of strategies, a meta-strategy. The action sequence that has the highest probability associated with the transition matrix would be our selected strategy. In other words, FA works by discretizing possible strategies. Instead of determining the strategy of the adversary, we select the most likely strategy in the pool of candidates. Some examples of transition matrices are shown below.

Always Defect	New Strategy		
Old Strategy		Selfish	Collective

	Selfish	1.0	0.0
	Collective	1.0	0.0

Above, we have a 100% chance of staying on a selfish strategy and a 100% chance of switching to a selfish strategy. We have no chance in staying on a collective strategy, and no chance to switch to a collective strategy.

Always Cooperate	New Strategy		
Old Strategy		Selfish	Collective
	Selfish	0.0	1.0
	Collective	0.0	1.0

Above, we have a 100% chance of staying on a collective strategy and a 100% chance of switching to a collective strategy. We have no chance in staying on a selfish strategy, and no chance to switch to a selfish strategy.

Average	New Strategy		
Old Strategy		Selfish	Collective
	Selfish	0.85	0.15
	Collective	0.15	0.85

Once a player has chosen a strategy, the probability that they stay on that same strategy is generally much higher than them switching. That is, humans experience strategic inertia. This is displayed above.

Stubborn	New Strategy

Old Strategy		Selfish	Collective
	Selfish	0.95	0.05
	Collective	0.05	0.95

Above is an example of a stubborn strategy. We have a high chance of staying on a collective strategy, but in the event the agent becomes cynical, the agent has a high chance of getting stuck on a selfish strategy.

Ambivalent	New Strategy		
Old Strategy		Selfish	Collective
	Selfish	0.65	0.35
	Collective	0.35	0.65

Above is an example of an ambivalent strategy. We have a high chance of switching strategies and a low chance of staying on any strategy because the agent is unsure of the most optimal strategy to play. This can be expected strategy at the beginning of the game.

Predicting Actions

In the above examples, we calculated the probability of the observed sequence containing three states. If we want to predict the most likely action of our opponent, then we would have to extend our observation sequence to four states. Since we do not know what the fourth state is, we calculate the probabilities for two different observation sequences: one in which the final state is to defect and one in which the final state is to cooperate. We run the FA algorithm for each

observation sequence, but we have to specify a transition matrix. For example, this approach would yield the probability of a defect in the next state, assuming the past three states have shown an ambivalent strategy.

We can also predict several steps ahead. There can be many game theoretic reasons for wanting to think more steps ahead such as limited access to computation, taking advantage of down time, and estimating what path an action will lead to considering chain reaction of optimal responses from agents. Below, we exemplify the use of the FA algorithm to calculate two turns ahead by comparing four observations.

$\text{argmax}(P_1, P_2, P_3, P_4)$

$P_1 = P(Y_0, Y_0, Y_1, Y_0, Y_0 \mid \text{ambivalent})$ $P_2 = P(Y_0, Y_0, Y_1, Y_1, Y_0 \mid \text{ambivalent})$

$P_3 = P(Y_0, Y_0, Y_1, Y_0, Y_1 \mid \text{ambivalent})$ $P_4 = P(Y_0, Y_0, Y_1, Y_1, Y_1 \mid \text{ambivalent})$

Discussion: Limitations Hidden Markov Models

There are some disadvantages to our approach. We can continue to think several turns ahead, but as we predict farther into the future, our prediction becomes less accurate, and the time complexity increases exponentially. Furthermore, the most probable action sequence might not be statistically significant. In other words, there is a possibility that the predicted most optimal action or action sequence does not differ from the other action(s) when more states and actions are sampled. Another problem not accounted for above is that the transition matrices can change overtime. An adversary can start with an ambivalent approach early in the game and then transition to a stubborn strategy late in the game. One approach to solve this would be to

continually rerun the FA algorithm and determine the most appropriate meta-strategy after every new observation. However, this alone does not suffice because subsets of a sequence of actions could conflict. For example, the first half of the action observation sequence could indicate the ambivalent meta-strategy and the second half could indicate the stubborn meta-strategy. However, the HMM algorithm might misinterpret this as an overall 50/50% balanced meta-strategy. The HMM algorithm would need to segment the action sequence and model transition matrices as a function that dynamically changes with each new observed action. Finally, rerunning the FA algorithm several times to find the best candidate strategy, rerunning the FA algorithm to compare future states and sementing transition matrix functions can quickly become too computationally expensive for real-world applications.

Demonstrating HMM

Experimental Design

Below, we demonstrate Hidden Markov Models using python, with the library HMMLearn. A supplementary notebook Azartash_Sina.ipynb file is provided to show how the calculations are made and the implementation in code. A zero indicates a defection and a one indicates cooperation. In this section, we demonstrate how five different hidden play strategies are detected. Each hidden strategy guides the hidden states and are described in the Inferring a Strategy section above. In short, the always_defect player always chooses to switch to a defection; the always_cooperate player always chooses to switch to a cooperation strategy; the average player is more likely to stay on their strategy than switch; the stubborn play strategy is

much less likely to make a change to their hidden state; and the ambivalent play strategy is more likely than the average player to switch hidden states.

In the figure below, we implement these strategies. At the top, we state the strategy name. The matrix underneath represents the transition matrix. The transition matrix represents the probability of switching to a different state or staying in the current state. For example, for the stubborn play strategy, there is a 95% chance to stay on the current strategy and a 5% chance to switch strategies. Next, the hidden state row represents a randomly generated sample of each strategy. We can see that the hidden state array of the always-defect player is majority zero and the hidden state array of the always-cooperate player is majority one. Also, we see that stubborn play strategy is least likely to change their hidden state and the ambivalent strategy is most likely to change their hidden state. Beneath the hidden state row is the results row, this shows the action taken based on the hidden state. An important note is that the action taken will not always match the hidden state. For example, a hidden state of zero indicates the intention to defect, but in reality a cooperation may result. This represents the unpredictability of real life situations and is encoded in the emission matrix. We arbitrarily set the emission matrix to have an 80% chance of following a hidden state. The true probability that a human decides to follow through on their intentions is a difficult number to obtain, is contextual, and is always changing. Since we chose 80%, we see that the majority of time, the players will follow their hidden state. Lastly, at the bottom row is the OM accuracy. In this row, we predict how accurately we can predict the strategy of the player by measuring the agreement between the observed actions in results and the predicted hidden state in the hidden state row. Since hidden intentions do not always match the action observed in a game, and due to random sampling, the accuracy levels will be around

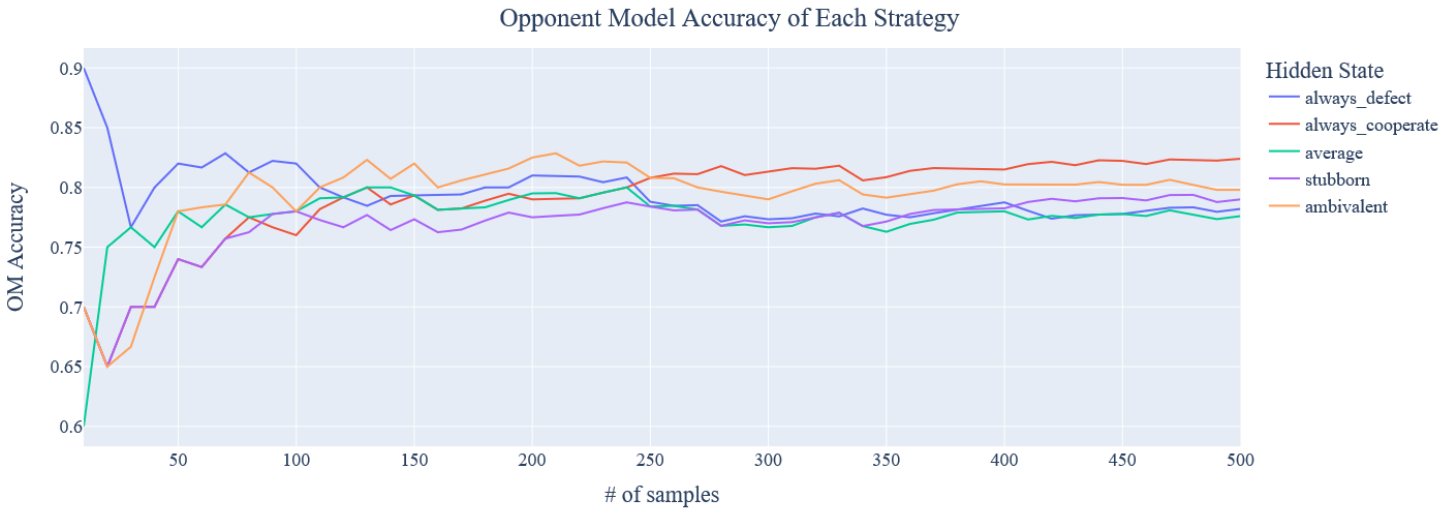
the emission percentage. Furthermore, since our sample size is 10, the accuracy levels will more likely be less than the emission percentage.

```
always_defect player
[[1 0]
 [1 0]]
Hidden State: [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Result      : [1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0]
Opponnent Model Accuracy = 0.7666666666666667

always_cooperate player
[[0 1]
 [0 1]]
Hidden State: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Result      : [1 0 1 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0]
Opponnent Model Accuracy = 0.7

average player
[[0.85 0.15]
 [0.15 0.85]]
Hidden State: [1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1]
Result      : [1 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 0]
Opponnent Model Accuracy = 0.7666666666666667

stubborn player
[[0.95 0.05]
 [0.05 0.95]]
Hidden State: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Result      : [1 0 1 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0]
Opponnent Model Accuracy = 0.7

ambivalent player
[[0.65 0.35]
 [0.35 0.65]]
Hidden State: [1 1 1 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 1]
Result      : [1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 1 1 1 0]
Opponnent Model Accuracy = 0.6666666666666666
```

Modeling Behavior of Each Strategy

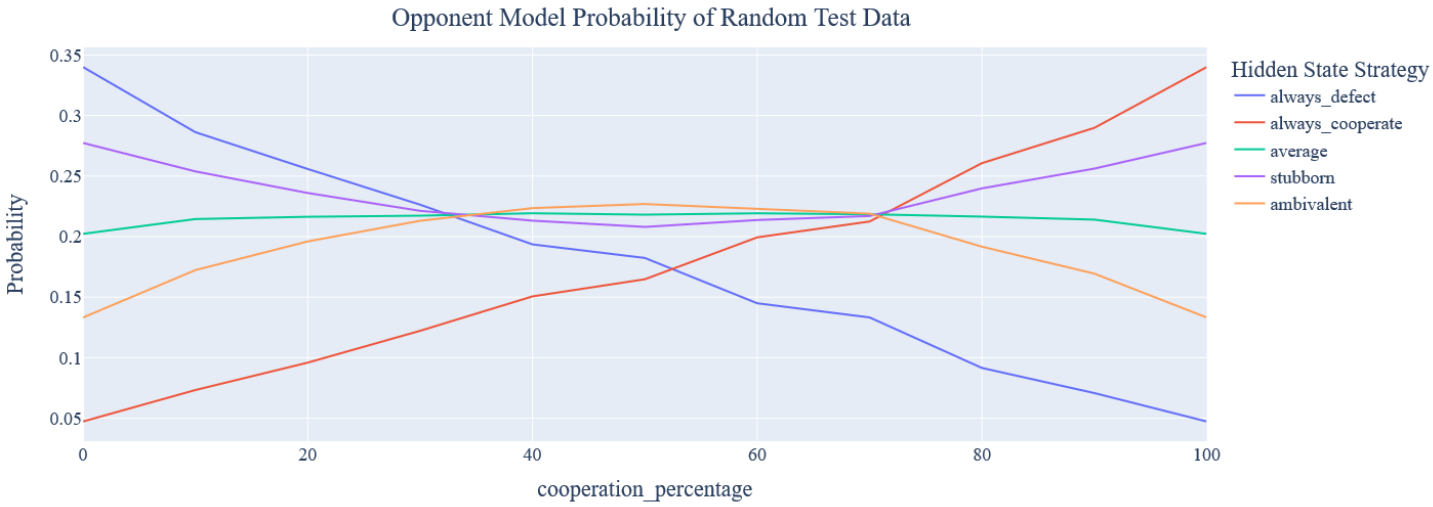
We investigated how the OM detection of each strategy would change over a larger sample space. Below, the number of samples represented the number of observed states sampled

from each hidden state strategy. The hidden strategy is unknown, and we attempt to predict the right strategy using OM HMM. We see that with an increasing number of samples, the OM prediction accuracy fluctuates around 80 percent with the majority of fluctuations below 80. After 150 iterations of the iterated prisoner's dilemma, we start to converge on accuracy. This indicates that it takes on average 150 trials of the game before our software can detect which strategy the opponent is playing. A human is much more likely to recognize a strategy in many less iterations. The model used in this study is for demonstration purposes while more sophisticated, modern, industrial models should require many less trials. Even though the accuracy levels do not converge to the emission percentage, the model is still usable and very functional. For example, at 100 samples, we have approximately 83% chance of detecting the always-defect strategy and 76% chance of detecting the always-cooperate strategy. Finally, even after 500 trials of the game, the model does not fully converge and some uncertainty remains. Further testing showed that the accuracy displays asymptotic behavior, never fully converging at 1000 and 10000 trials.



Detecting Strategy in Randomly Generated Test Data

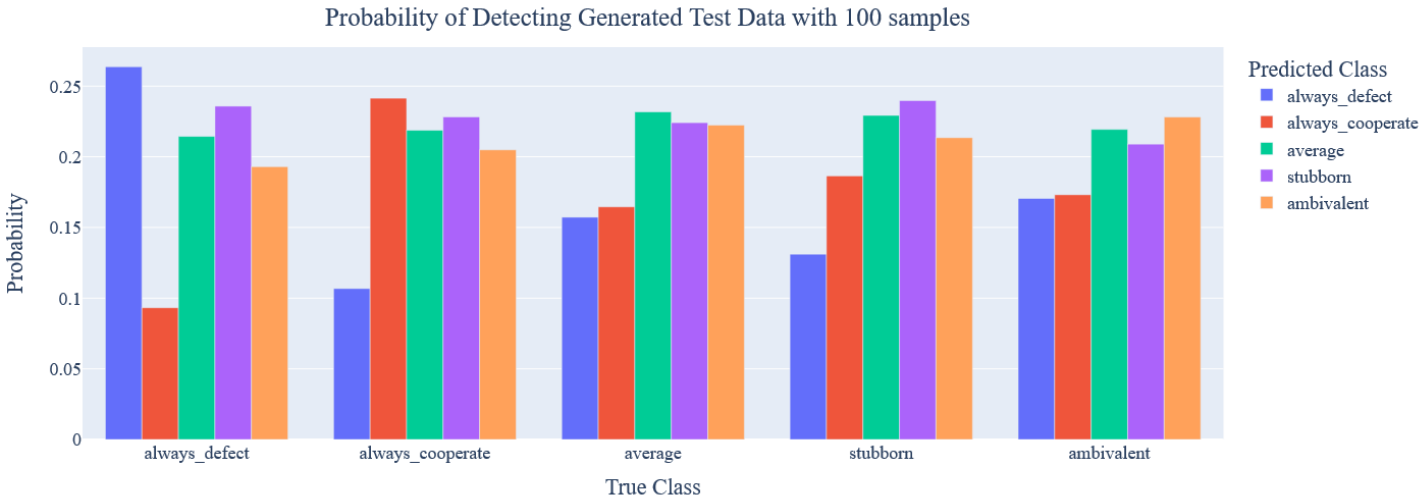
We investigated the ability of our HMM model to detect the hidden strategy of a player on randomly generated test data with a sample size of 100 trials. The randomly generated test data is generated with a probability of a random defection or cooperation using a uniform probability. Therefore, only the percentage of defection and percentage of cooperation were investigated. In other words, there was no pattern between states and the states were completely random. The sequence ordering of test data was not investigated in this section, but it is investigated in the later sections. In the figure below, we see that as the percentage of cooperation increases, the always cooperate strategy is more likely to display the highest probability. The always-cooperate strategy becomes the most likely strategy after the randomly generated test data shows about 70% cooperation. Similarly, the always-defect strategy is the most likely strategy when the generated test data shows about 30% cooperation. The other strategies were more difficult to detect. This makes sense because sequential patterns of switching states were not encoded into the randomly generated test data. Interestingly, the stubborn strategy was the second most likely strategy when there was a large disproportion of strategies. This also marks sense because the stubborn strategy is the most likely to not have the hidden state switch. Likewise, the ambivalent strategy is the highest when there is an equal proportion of strategies because the ambivalent strategy is constantly switching states



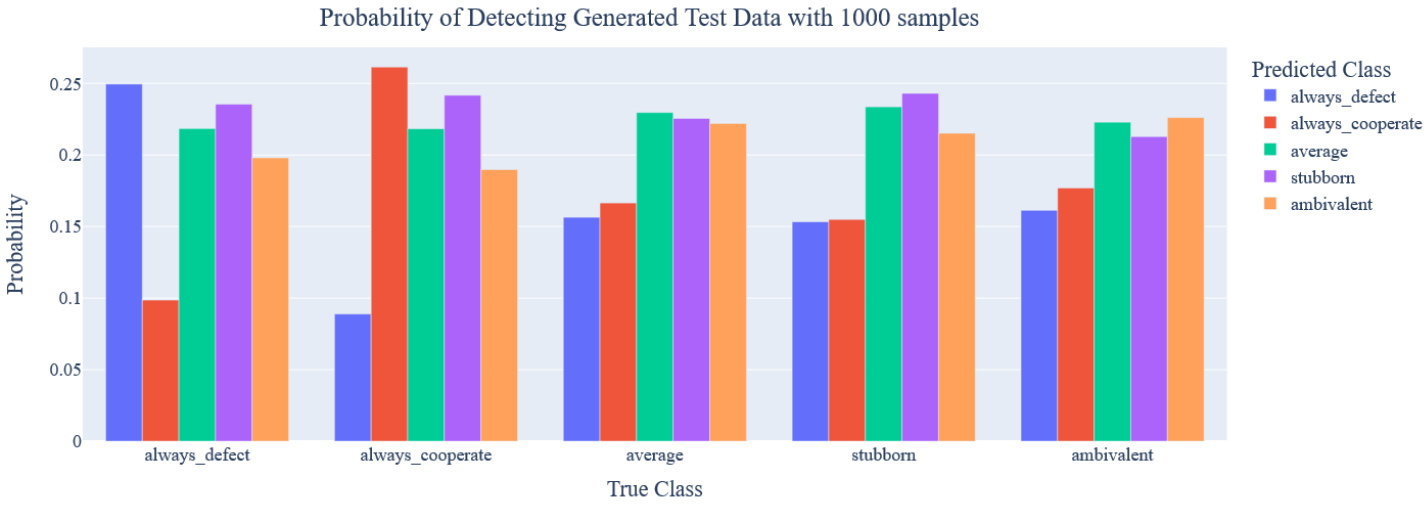
Detecting Strategy in Generated Data From Unknown Strategy Class

In this section we did encode sequential information into our test cases. We generated test data by randomly choosing a strategy and then sampling from that strategy. Our OM model then has to predict from which strategy the generated test data came from. This represents real life situations in which the AI has a preconceived model of the human player based on experience. The experience has allowed the AI to create categories of different play styles. In this situation, we are matching the play style of the randomly generated data to five different preconceived categories. In the figure below, we test our OM capability on one-hundred trials of the iterated prisoner's dilemma game. Our results are generally positive and successful, for each true strategy, the hidden strategy predicted displays the highest probability. While our OM model is successful at a sample size of 100, the probability values were very close to each other especially for the non-extreme strategies. The always-defect and the always-cooperate strategies were the easiest to detect, and the average play style was the most difficult to detect. When the test data is

sampled from the average play style strategy, the stubborn and ambivalent strategies were very close. This makes sense because the average strategy is between these stubborn and ambivalent strategies.



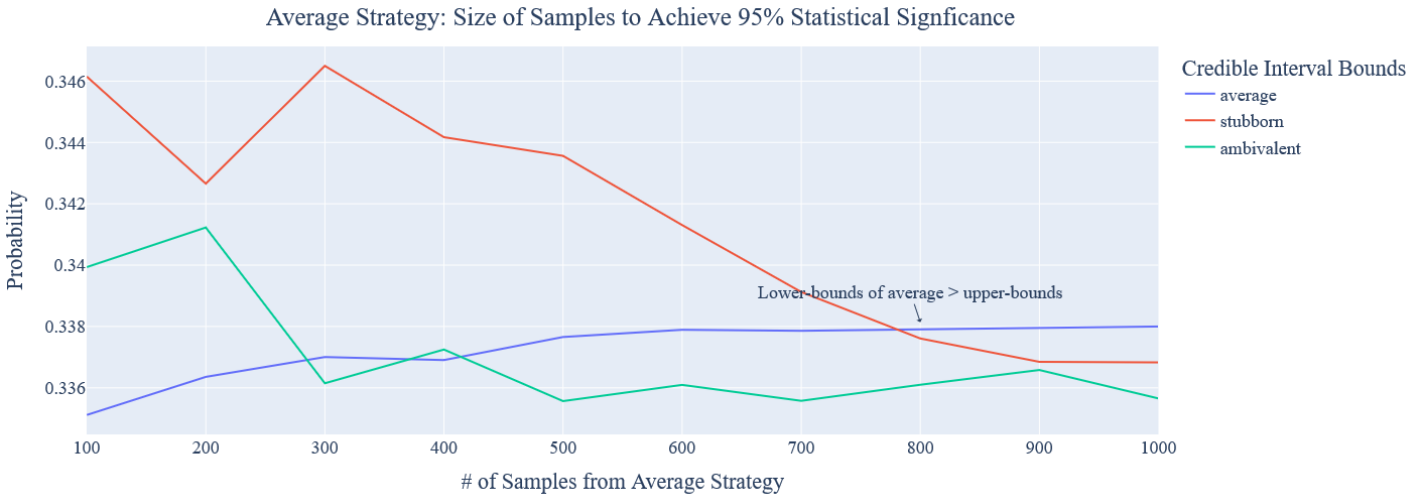
We repeated this experiment with a larger sample size to see if the predicted probabilities of each hidden strategy would start to distinguish. Indeed, this is the case, but the effects are slight. Further experimentation showed that exponentially more samples are needed to achieve a further distinguishing of predicted probabilities.



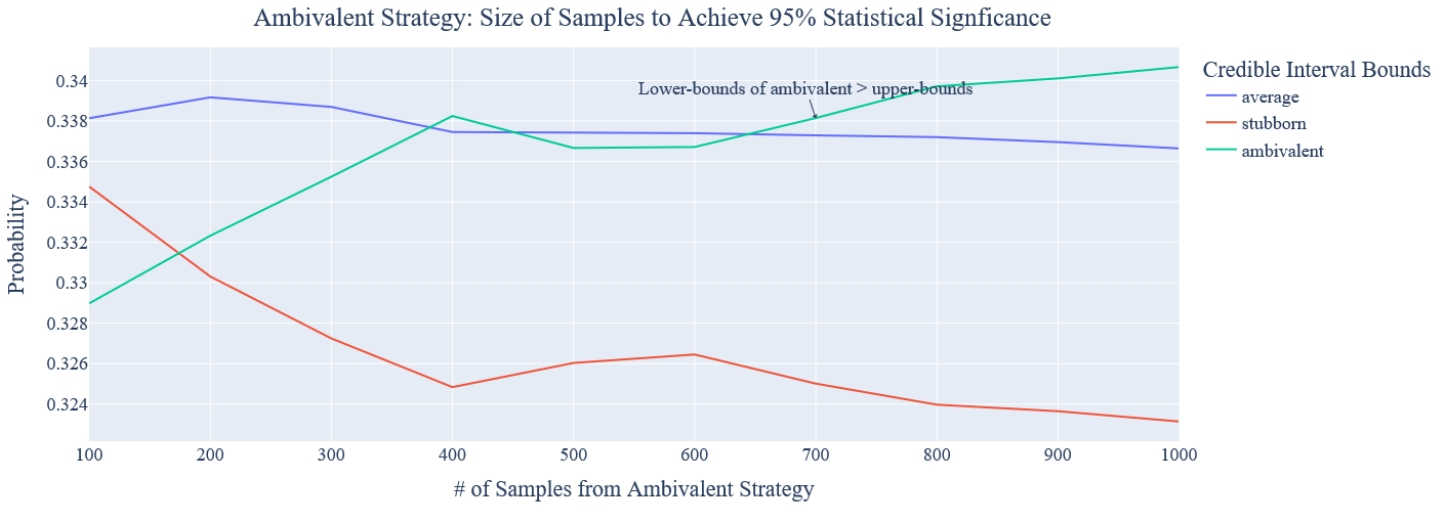
Quantifying Strength of Our with Statistical Significance

Finally, we were interested to see if the predicted probabilities are statistically significant. Since the test data was randomly generated, there is the chance that our OM displaying a high performance simply due to luck. We wanted to control for that so we calculated the 95% credible interval. The credible interval is a Bayesian confidence interval technique based on bootstrapping. In short, we ran our OM model on several different sample sizes ranging from 100 to 1000 with an increase in 100 samples at each split. For each sample size, we ran our OM model 30 times to generate a probability distribution. We used this probability distribution to test for statistical significance at each step. If the predicted hidden strategy had its lower bounds at the 5% quantile than the predicted hidden strategy of the other methods at the 95% quantile, then we said that our OM was credible in its prediction.

Below, we see that it takes 800 samples for the OM model to become credible in detecting the average strategy. This is expected as the average strategy is the most difficult to detect.



Next, we see that it takes 700 samples for the OM model to become credible in detecting the ambivalent strategy. This is expected as the ambivalent strategy is the second most difficult strategy to detect.



Lastly, for some reason, the stubborn strategy was the most difficult to detect. It took a little less than 900 trials of the game, for the OM to be credible in its prediction of the hidden strategy.

Conclusion

We have introduced Opponent Modeling and described several foundational background concepts. We gave examples of modern research endeavors, discussed the ramifications of the OM, and described some limitations of OM. Next, we explained Hidden Markov Models as a solution to OM and described its algorithmic foundations. We applied the Hidden Markov Model to the iterated prisoner's dilemma and described how the decision process to implement it. Finally, we gave an example of python code. At 100 samples of the game, our OM is able to detect the strategy of the other player. At 900 samples of the game, our OM is credible at detecting the correct strategy 95% of the time.

References

Ganzfried, and Sandholm. 2011. “Game Theory-Based Opponent Modeling in Large Imperfect-Information Games.” ACM Other Conferences. 2011. https://dl.acm.org/doi/abs/10.5555/2031678.2031693.

Horák, Karel, and Branislav Bošanský. 2019. “Solving Partially Observable Stochastic Games with Public Observations.” *Proceedings of the AAAI Conference on Artificial Intelligence*. https://doi.org/10.1609/aaai.v33i01.33012029.

Keselj, Vlado. 2009. “Speech and Language Processing (second Edition) Daniel Jurafsky and James H. Martin (Stanford University and University of Colorado at Boulder) Pearson Prentice Hall, 2009, Xxxi 988 Pp; Hardbound, ISBN 978-0-13-187321-6, $115.00.” *Computational Linguistics*. https://doi.org/10.1162/coli.b09-001.

Knegt, Stefan J. L., Madalina M. Drugan, and Marco A. Wiering. 2018. “Opponent Modelling in the Game of Tron Using Reinforcement Learning.” *Proceedings of the 10th International Conference on Agents and Artificial Intelligence*. https://doi.org/10.5220/0006536300290040.

Mandziuk, Jacek. 2010. *Knowledge-Free and Learning-Based Methods in Intelligent Game Playing*. Springer.

Nashed, Samer, and Shlomo Zilberstein. 2022a. “A Survey of Opponent Modeling in Adversarial Domains.” *Journal of Artificial Intelligence Research*. https://doi.org/10.1613/jair.1.12889.

———. 2022b. “A Survey of Opponent Modeling in Adversarial Domains.” *Journal of Artificial Intelligence Research*. https://doi.org/10.1613/jair.1.12889.

———. 2022c. “A Survey of Opponent Modeling in Adversarial Domains.” *Journal of Artificial Intelligence Research*. https://doi.org/10.1613/jair.1.12889.

Osten, Friedrich Burkhard von der, Friedrich Burkhard von der Osten, Michael Kirley, and Tim Miller. 2017. “The Minds of Many: Opponent Modeling in a Stochastic Game.” *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. https://doi.org/10.24963/ijcai.2017/537.