**Machine Learning Iterated Prisoner’s Dilemma Strategy**

James Wallace

MIS 4300

4/22/2025

Rough Draft

## Introduction

The iterated prisoner’s dilemma is designed to simulate recurring interactions between parties. It consists of two parties who are not allowed to communicate with one another and are given the choice to either cooperate or defect. If one party cooperates while the other defects it will score 0 points while the defector scores 5, if both parties defect, they will both earn 1 point, lastly if both cooperate, they will both earn three points. With these rules the most obvious strategy is to defect, because regardless of the other party’s decision, defecting will always give you more points than cooperating. However, this introduces the core dilemma of the situation, if both parties make the best choice for themselves and defect the result is worse than if they had cooperated with one another. This problem becomes more difficult with more iterations of the dilemma as any action will influence future choices by the two parties.

David Axelrod’s 1984 book The Evolution of Cooperation details the first iterated prisoner’s dilemma tournament in which he took 14 submitted strategies and had them compete against one another to score the most points. Each match between strategies would consist of 200 turns and each strategy would have a match vs each other strategy, itself, and a purely random strategy. The results were surprising as the simplest strategy submitted, tit-for-tat, proved to be the most effective. Due to the long length of the match, strategies that emphasized cooperation tended to outperform those that did not. The results of the tournament led to Axelrod creating four rules for a successful strategy, the first being to be nice, or start by cooperating until the other strategy forces you not to. The second rule is to be retaliatory, when your opponent defects you need to follow suit at least once. Next, the strategy needs to be forgiving, it must be able to cooperate again after defecting. Finally, the strategy needs to be clear and predictable to allow the other strategy to be able to plan its moves around yours. While this last one seems counterintuitive for a competition, if both strategies earn the most points by cooperating, being able to plan your actions around the other strategy becomes mutually beneficial.

The purpose of this project is to take the results of an iterated prisoner’s dilemma tournament and use them to create a machine learning model to compete in an iterated prisoner’s dilemma tournament to see how it performs, if it follows the four rules laid out by David Axelrod, and to see if it proves to be more complex than simpler deterministic strategies such as tit-for-tat.

## Creating the Data

To train a model to play in the tournament I need to start by creating a data file from my own iterated prisoner’s dilemma tournament to train it on. To run the tournament, I will be using the Axelrod python package as it contains many strategies and can save the raw turn data. This package includes just under 250 strategies and the capabilities to run any of them in an iterated prisoner’s dilemma tournament against one another. For this tournament I will use the strategies included in Axelrod’s first tournament. After running the tournament, the raw data from each strategies action in each match is saved to a new csv file. The raw data includes all 200 decisions made by each strategy in each match it plays throughout the tournament as well as the final scores. It includes additional data that might prove useful in the future, but raw match data is the most important for training the machine learning model. The goal is to create a model that can look at the overall moves played so far in a match and determine if it will score more points on average if it defects or cooperates on each turn.

I will start by using the fourteen strategies and a pure random strategy that were included in Axelrod’s first tournament to create my first data file to train models on. My hypothesis is that more strategies will result in a better model due to the extra data it will have access to. However, additional strategies will also lead to longer computing times while creating the model itself. This has given me the idea to start building and testing my model in a tournament with only a few strategies, such as Axelrod’s first tournament, and then giving it access to more data once the model is refined. This will provide several benefits, primarily saving time but also to see how much the model is improved with more data, or if it will become too complicated and perform worse as Axelrod’s fourth rule warns.

After running an iteration of Axelrod’s first tournament, I was left with a csv file showing the raw tournament match data for each strategy. This file contains 240 rows and the important Actions column. This column includes all 200 moves played by each strategy for each match played in the tournament. To make the dataset suitable for training the Action column needs to be split from one column with 200 characters to 200 columns, one for each turn. This was accomplished by importing the data set into Python and using the pandas Series() function to split the string into an array. From here the data set was appended with the new array and the C and D characters, for cooperate and defect, were replaced with 1 for C and 0 for D. From here I used a PostgreSQL server hosted on my machine to further trim the csv down to only include the columns that would be important for creating the model. This included the 200 columns for each turn, the index, interaction index, and the final scores.

## Model Creation

Once the raw tournament data was refined into a new csv file, model\_base.csv, I was able to start testing various models. My first thought for a model was a linear regression model with the following form:

Where are the corresponding turn numbers being used to predict the final scores. While this model might work well as a method to predict a final score it is not able to be implemented into a strategy very well. To create a strategy from a model the model needs to be able to be filled in with existing data for each up to where = the next turn. This would mean that the tournament would not be able to predict a final score until the last turn of the match, making it useless. Additionally, this model fails to incorporate the opponents moves into the data which is very important in being able to predict a final score since each turn relies on the combination of actions from each strategy to determine scoring.

In an attempt to make it easier to include both of the competing strategies moves in the model I attempted to create models after transforming the data from a two-dimensional array with dimensions 240x200 into a three-dimensional array with dimensions 120x2x200. This stored each pair of 200 turns with one another rather than in separate rows. This allowed me to create models using moves from both strategies instead of just one, but encountered issues because I could only look at the most recent N moves, where N is the number of turns I wanted the model to look back at. As a result, while I was able to create models that could be used with fewer variables and incorporated into a strategy, I was only able to use data from the most recent turns. This meant that if I would only see the last 20 turns from each strategy for each match eliminating 90% of the raw data from the data set.

At this point I decided to create a few models to gain insight into the type of model I should try to implement once I was able to sort my raw data in a more suitable way. I started by creating a logistic regression model to predict the opposing strategies move. I assumed logistic regression would be ideal for this because it would be predicting a categorical result of either cooperate or defect. I was not sure if this would be very useful, but it would give me a good starting point that I was confident would be accurate since the deterministic strategies were inherently predictable. Additionally, it served as proof of concept if I was unable to find a model capable of predicting the final score well. The model was trained with and only looked at the five most recent turns, but despite the small sample size was able to predict opponent moves with 83.78% accuracy against the test data.

From here I wanted to test two models against each other. A linear regression model was still my first choice since it is easy for me to understand how to use it to create a strategy, but I also wanted to try creating a neural network because a lot of the similar projects I researched use a type of neural network as the base for their strategies. To start I set as an arbitrary choice because I want to be able to use the model’s strategy for a majority of the match. From there I created a linear regression model that would take the last 10 moves from each strategy to predict the final score. Alternatively, I created a neural network using the TensorFlow Python package. I created a neural network with three dense layers with the mean squared error (MSE) as the loss function and mean absolute error (MAE) as the metric. From here I trained it over one thousand epochs to see how well it developed. The resulting neural network model had a final MAE of 47.59. Similarly, I created a linear regression model with the same value. Surprisingly, the linear regression model had a lower MAE value of 45.49. With these results I decided to stick with a linear regression model in the future because I am more comfortable working with these types of models and they can be trained much faster.

The last step in model creation was figuring out a way to use more than just the 5% of the data set that the models were trained on in the prior section. To make use of all the data I needed to be able to not only look at the 10 prior turns but start from turn 11 and take the move from the 10 previous turns for each subsequent turn. Professor Beaty had the idea to stick with a two-dimensional array instead of the three-dimensional array and use as a lookback variable to loop through the data set for each turn. This allowed me to expand my training data from one set of values for each match into as many as 190 sets of values for each match. This allowed me to use nearly all of the data outside of the initial turns to train my linear regression models on. The figure below shows the new shape of the data at . A screenshot of a computer program

AI-generated content may be incorrect.

This was a significant improvement from the initial 240x200 dimensions of the data and was reflected in the resulting models. The new models had MAE scores ranging from 52.85 to as low as 17.83 as the value increased. I used this technique to crate a new data set for the logistic regression model as well, but this did not boost the accuracy much.

## Strategy Implementation

One important difference between my first models and the lookback model is that when I trained the linear regression model on the three-dimensional array looking at only the last turns, when I increased the value to try to create a model with more data this did not necessarily result in a more accurate model. However, using the lookback method to loop through the data allowed for the model to get progressively more accurate with higher values, until I reach at least at which point the errors reach the millions. Depending on the lookback value, , of my model I will always need to wait turns before using the model to make predictions on the current match. This presents the issue of how many turns do I want to wait for a slightly more accurate model? With the models becoming progressively more accurate I did not want to wait for too long to implement the model’s strategy, but I also wanted to use the best model possible for the strategy. In the end I elected to use multiple models with different values so that I could start with a less accurate model using starting on turn 9, then transition to more accurate model at . This will allow me to use a progressively more accurate model the further into the game the strategy progresses, similar to how a person will be able to recognize more patterns in the strategies the longer a match goes.

For the first 9 turns I decided to use a copy of the tit-for-tat strategy because it is easy to implement and has proven to be effective. After turn 9 I have enough data to start using my model. Each model would take the prior turns to use for predictions then fill in the remaining two values with a prediction of the opponent’s next move using a logistic regression model and either a cooperate or defect for its own move. The model would then be used to make two predictions for each outcome, cooperate or defecting into the predicted opponents move, and choose the one that resulted in a higher predicted final score.

My initial plan was to take advantage of the fact that the Axelrod package has a GitHub repository with all the files included. I created a fork of this repository to allow me to edit the files while maintaining some version control in case I broke anything while adding the strategy. While this worked effectively, it proved to be a case of over complication as I was able to create the same strategy inside of my Jupyter Notebook and work to fix bugs without having to commit changes and repeatedly clone the forked repository for each change.

## Tournament Results

While researching how others had approached using machine learning techniques to approach the iterated prisoner’s dilemma problem I saw a lot of papers with very impressive results and was hoping to find similar success with my models. However, after 100 iterations of Axelrod’s first tournament with my custom strategy added, it was only able to finish in tenth place on average. The following table shows the result summary:

|  |  |  |  |
| --- | --- | --- | --- |
| Rank | Name | Average Score | Average Wins |
| 1 | First by Stein and Rapoport: 0.05: (D, D) | 2.590833 | 11 |
| 2 | Grudger | 2.531 | 6 |
| 3 | First by Grofman | 2.528333 | 1 |
| 4 | First by Davis: 10 | 2.5255 | 5 |
| 5 | Tit For Tat | 2.523 | 0 |
| 6 | First by Shubik | 2.521667 | 3 |
| 7 | First by Tideman and Chieruzzi: (D, D) | 2.487667 | 11 |
| 8 | First by Nydegger | 2.467333 | 0 |
| 9 | First by Graaskamp: 0.05 | 2.340333 | 4 |
| 10 | My Strategy | 2.288833 | 7 |
| 11 | First by Downing | 2.2765 | 7 |
| 12 | First by Feld: 1.0, 0.5, 200 | 2.0255 | 10.5 |
| 13 | First by Tullock | 1.876 | 9 |
| 14 | First by Joss: 0.9 | 1.856667 | 11.5 |
| 15 | First by Anonymous | 1.732167 | 4 |
| 16 | Random: 0.5 | 1.729333 | 4 |

While I am somewhat disappointed to have not designed a groundbreakingly good machine learning strategy I am very happy to have been able to create a strategy that is able to be competitive and not simply make the same decision every turn. That being said the strategy did perform better when sorted by the amount of games won in each tournament rather than average score. The following graph shows the distribution of wins across 100 tournaments for each strategy:A graph of a diagram

AI-generated content may be incorrect.

When evaluated by this metric the strategy places in 5th while remaining very consistent. I was surprised that there was this significant difference between the strategies placements. I suspect that this may be a result of the strategy being able to consistently outperform cooperative strategies in head to heads but doing a poor job of maximizing its score vs more aggressive strategies that took advantage of how much it cooperated. Its preference for cooperation can be seen when looking at the strategies decision matrix shown below. This shows the rates at which my strategy’s actions resulted in each of the four possible outcomes and how it reacted to each outcome:

The strategy clearly prioritized cooperation as it cooperated on over 84% of each of its turns. Interestingly, its decision making seems to rely more on its own moves that those of its opponents as its has a very high cooperation rate of 97.5% and 98.5% after it cooperated the prior turn when the opponent both cooperated and defected respectively. Meanwhile, it only cooperated 11.5% and 4% of the time after it had defected when the opposing strategy had cooperated and defected respectively. An additional surprise was that the strategy proved to be one of the most consistently scoring strategies, even more so than the entirely predetermined strategies like tit-for-tat. This was the most surprising result from the tournament as I had some expectations that there would be some strategies that it would struggle to perform against. Particularly with the high cooperation rate my strategy aimed for, I am surprised it was able to maintain such a consistent average score across 100 tournaments as shown below:A graph of statistics

AI-generated content may be incorrect.

## Conclusion

I was happy to find that creating a model to predict the final score of an iterated prisoner’s dilemma match is certainly possible and can lead to an effective strategy. I anticipated that the neural network would create a more accurate model than a linear regression model but was surprised to see the linear regression model outperform it. This of course carries the caveat that I only tested these models against each other once and do not know enough about neural networks to know if I properly optimized this model. Regardless, it was a good illustration that a simpler strategy is not always a worse strategy. Which was further proven as the simplest strategy, tit for tat, outperformed my still much more complicated strategy. That being said, this project was incredibly successful in creating an effective strategy as it was able to consistently perform at a good level and never seemed to get abused by other strategies. It also seemed to agree with the rules outlined by Axelrod in prioritizing cooperation as shown by its high cooperation rate. The strategy did not seem to be too retaliatory though it did tend to cooperate less if the opposing strategy defected. While the strategy seemed to be capable of forgiveness though not very frequently, the strategy seemed to prefer to keep doing the same thing if it was able to, but once it started defecting it was much more willing to cooperate once again if the opponent did as well. This leaves the most difficult to analyze assertion by Axelrod, that predictability was very important so that opponents would know they can cooperate. However, this strategy shows either the flaw in that thinking or the flaw in the tournament itself. The strategy was predictable in that it would tend to cooperate but when it defected it continued to do so for a while. More interestingly though it appeared to be able to tell when the game was nearing the end despite not being able to know how long the game was going on for. Almost without fail while inspecting the raw data from a single iteration of the tournament would show it defecting repeatedly at the end of each match. This is interesting, because the big flaw with a prisoner’s dilemma tournament is that it is always the optimal move to defect on the last turn if you know the game is going to end as the opponent will not have a change to retaliate, and my strategy seemed to be able to anticipate the game ending and start defecting more often as a result. This could be explained by my strategy changing models at turn 179 which is around where a lot of these defections started, but this was also the most accurate model which makes me wonder if it would continue to act more aggressively if the games lasted longer or if the model’s proximity to the end of the game allowed it to pick up on this pattern while others could not differentiate between he beginning and end of the game was effectively.