# Project 3: Decision Trees for Expert Iteration

Francisco Parrilla Andrade 1900939

**Abstract**—Reinforcement learning has been a field that has shown promising results for the gaming industry. There was systems that are able to play games such as Go, Tic-Tac-Toe, Chess, and are able to beat human world champions. Monte Carlo Tree Search has helped in the creation of such agents, by exploring the different possibilities and make a decision based on it. There are different methods that help to ease the exploration done by MCTS, which uses a random policy. This project presents an implementation of MCTS in the game of Tic-Tac-Toe, that uses a trained classifier, decision tree, to explore the nodes, and create an expert agent in the game. The proposed implementation is able to increase the number of positives outcomes (wins) agents have in the game.

**Index Terms**—Reinforcement Learning, Monte Carlos Tree Search Algorithm, Supervised Learning, Decision Trees, Tic-Tac-Toe

## 1 Introduction

Sometimes, players that play against a computer get frustrated when the difficulty of the game is none, and the difficulty level is top to the max settings. Nonetheless, building intelligent agents within games that can mimic a human-mind decision making criteria is something from the past. To be able to do this, researchers have devoted time in finding and developing algorithms that can optimally create systems that are able to cope and outperform an expert human player. For example, games such as Tic-Tac-Toe, Othello, Chinese Checkers, and Go, have seen the use of Monte Carlo Tree Search (MCTS) algorithm to be able to find the optimal move at each stage of the game. This is done by exploring the different possible combinations from a current state, and take an action that brings more rewards [1], [2]. Moreover, deep learning has allow a better exploration during the MCTS, and has shown to perform well [1], [2]. Classical classifiers such as decision trees have also been used during MCTS algorithm, such as [3]. This project will use a supervised learning algorithm, decision tree, as a classifier that replaces the way MCTS chooses the child node to expand. Instead of randomly doing it, the classifier will predict which child node to explore. The experiment will use the game Tic-Tac-Toe.

The following sections of this report are: Background Literature Survey, Methodology, Results, Discussion and Conclusion.

## 2 Background

### 2.1 Monte Carlo Tree Search Algorithms

Monte-Carlo Tree Search algorithm, also known as MCTS, iteratively builds a search tree, and stops until a constraint is reached, such as, number of iterations, time, memory [4]. Each iteration consists of four stages: selection, expansion, simulation and backpropagation. MTCS are well known for effectively finding an optimal action based on a game state, and there have been situations where MCTS has proved to increase performance in games such as Pac-Man, Go [2].

### 2.2 Related Work

MCTS algorithm has helped researchers as a starting point for their systems, as they play by themselves, data is generated and it can be used by another method to learn better [1]. AlphaGo Zero is an example of this, by using the MCTS, the system was trained by self-play reinforcemtn learning that took random actions in the Go [1].There is a major difference between AlphaGo Zero and previous similar works, while the previous systems rely on annotated data by an expert, AlphaGo Zero initially relies in MCTS algorithm for an initial training [1].Neural networks have been considering to guide the tree search, where a board representation and previous representations are taken, and it guides the MCTS algorithm to take the best policy and decide what move to take [1].AlphaGo Zero competed against previous similar systems, one of them is AlphaGo Master which initially uses human annotated data. AlphaGo Zero won 60 games out of 60, showing that taking a reinforcement learning approach at the beggining is optimal and alleviates the task to rely on human annotated data [1]. Please refer to [1] for a better explanation of the methods and results. Another system, ExIT (Expert Iteration), is based on the premise that in order to solve a problem where a decision needs to be taken, a planning policy and a generalization is needed [5].A tree search algorithm, MCTS, was used for this policy generalization, where each move was simulated for a high number of times [5]. Later, a convolutional neural network was used a classifier to take an action using the data collected.For a more detailed explanation refer to [5].

MCTS is not only applicable to board games, it also has been applied in rehabilitation games that pursue to alleviate the effects caused by strokes and brain injuries, such as physical impairment of limbs [6]. This game uses kinetic devices that record the muscle's movements made by the player, this includes muscle orientation and joint activity. Depending on the measurements, the difficulty of the game is set by using MCTS, by determining the amount and distance of objects to present to the player [6].From this research, it was found that the game was able to adapt the intensity of the game based on the physical capability of the player, which makes it an useful and possible therapy game that can be used to help people affected by strokes and brain injuries [6].

MCTS algorithm is helpful for games that have a definite number of states, however, with games such as checkers, where the number of possible move is unbounded [2]. A solution for this, can be the use of heuristics, or rules known by the expert in addition to using MCTS [2]. [2] presented an approach based on this, in addition to using deep learning to train an agent that plays Chinese Checkers. All of the previous works used deep learning, but one of the disadvantages of deep learning methods is that their interpretability is difficult [3]. In [3], a learning approach using MCTS and based on a decision tree is proposed, that helps MCTS in the exploration by balancing the nodes that it explores. It was found that by using this approach, the MCTS improved the performance [3].

## 3 Methodology

The main objective of this project is to test the decision making capability of a classical machine learning algorithm, when playing a game of Tic-Tac-Toe. Currently, the code provided by [7] uses Monte Carlo Tree search algorithm to perform the best action at all points in the game.

In order to train a classifier, data is needed, therefore is necessary to design how data will be collected, more specifically, what information from a Tic-Tac-Toe game is important. The code provided by [7] represent the Tic-Tac-Toe board as an array of length 9, which reflects the state of the board at any given time. Every game is started as:

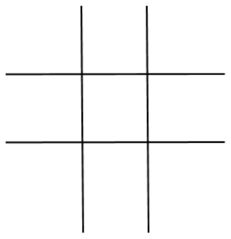

Fig. 1: Initial state of the board at start of the game

And the array will have the following values:

$$0, 0, 0, 0, 0, 0, 0, 0, 0$$

As the game progresses, each player will take an action, and this will be reflected with either 1 (player 1) or 2 (player 2). For example, if the action by player 1 is to put an X in the middle box of the board, the state of board will change to:

And the array will change to:

$$0, 0, 0, 0, 1, 0, 0, 0, 0$$

where the 1 represents that player 1 filled the middle box (index 4) with an X. As the game progresses, the array

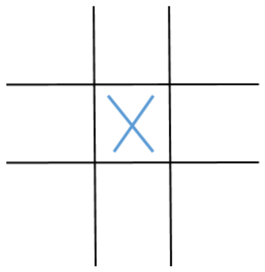

Fig. 2: State of the board after first action

can end up looking like:

$$1, 2, 1, 1, 1, 2, 2, 1, 2$$

which represent a finished game where players draw.

From this information, the state of the board at each given time will be collected, and also the player that took the action, and the action (where did the player decided to put an X or O) will be included. For example, if player 1 decides to put X in the middle of the board, the array will look like:

$$0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4$$

the collected array will only represent the state of the board before a player takes an action (indices 1-9),the player that took the action (1) and in which box will be affected (4). The state of the board and the player that has the action will be the independent variables, and the action taken will be the dependent variable.With that being said, the code *UCT_task1.py* to play and generate data from 300 games using only the MCTS algorithm, will be collected and saved to an output file named *data_t1.csv* in the project repository [1].

After training the first classifier with the saved data, the code used to play the game will changed in a way where the action taken by the program will not be entirely random, but instead, it will use the trained classifier to predict an action 90% of the time, while using the random choice 10% of the time. The trained classifier will be assigned to both players, and 300 games will be played and the data from them will be collected, and this process will be repeated 500 times. Every 10 iterations, the latest classifier will be tested by playing 50 games against each of the 10 previous classifiers, i.e. classifier 10 will played 50 games against classifier 1-9. For this, player 1 will always have the latest classifier. From the 450 games played every 10 iterations, the number of wins, losses and draws will be recorded, this is from player's 1 point of view. It is expected that from these games, the latest classifier wins the majority of time, however, it is possible that when classifier 10 faces classifier 9, the game ends up in a draw.

At the same time, every 10 iterations, 450 games will be played using no classifier, this is to see the difference in

1. https://github.com/fjpa121197/DataScienceDM/tree/master/Project/Assignment%202

results compared to using a classifier. The results from this analysis will be based on the number of times a player (1 or 2) wins a game, and how many times the game ends up in a draw. See Fig 3 for a graphical representation of the pipeline.

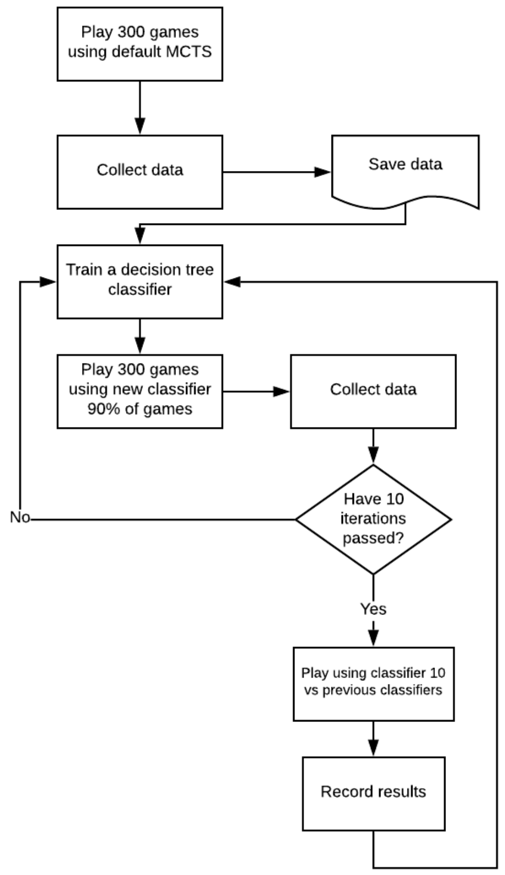
Play 300 games using default MCTS
Collect data
Save data
Train a decision tree classifier
Play 300 games using new classifier 90% of games
Collect data
Have 10 iterations passed?
No
Yes
Play using classifier 10 vs previous classifiers
Record results

Fig. 3: Pipeline of the experiment

Based on this information, the following hypothesis can be presented:

- H0: Using a trained classifier to select a child node during MCTS algorithm, instead of randomly choosing one, makes no difference in the amount of times a player wins.
- H1: Using a trained classifier to select a child node during MCTS, and iterating multiple times, it improves the decision making of the player, and can be seen by an increase in the number of wins.

In order to prove this hypothesis, a permutation test, with significance level of 5% will be calculated. The data that will be used for this is the one generated every 10 iterations, that is why every 10 iterations, 500 games using only MCTS random choice will be used. It will be compared with the data from

## 4 Results

This section has the results based on the experiments mentioned in the methodology section. First, the results from using the default MCTS algorithm that randomly chooses the child node, were the ones that it was expected. From Figure 4, it can be seen that the outcome from using default MCTS in most cases is a draw. Every 10 iterations, 500 games were played, which in total, this was done 50 times.

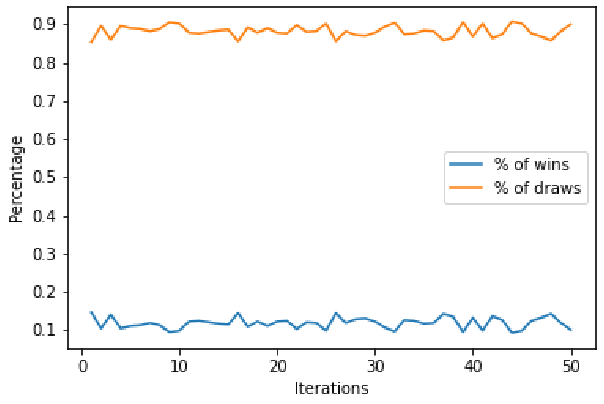

Fig. 4: Percentage of wins vs draws after 500 games per iteration

From this graph, the number of wins for player 1 and player 2 have been combined, this is to compare wins and draws alone. Additionally, it was found that the mean number of draws (out of 500 games) was 441, while the mean number of times an agent won (out of 500 games) was 58. It is evident that using default MCTS algorithm does not allow any of the players to win. To be confident about how many times a game ends up in a tie (out of 500 games), bootstrap (100,000 iterations and 90% confidence interval) was performed, and it was restated that the mean number of games is 441, and the lower and upper bound is 439 and 442 respectively.

On the other hand, Figure 5 shows the percentage of wins and draws when using a classifier, decision tree, to decide which node to expand during MCTS.

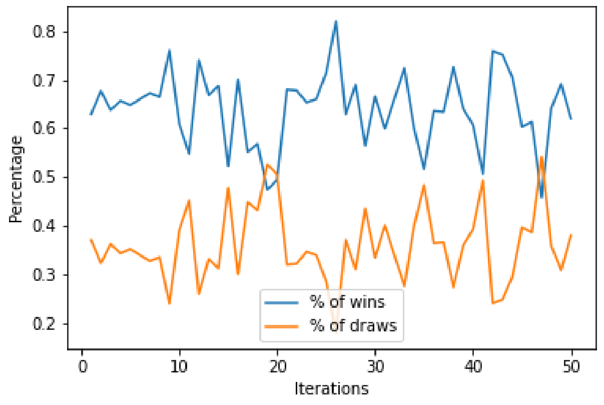

Fig. 5: Percentage of wins vs draws after 500 games per iteration

It can be seen that the amount of wins (wins from player 1 and 2) have increased significantly, and also the number of draws have decreased (out of 500 games per iteration).

Additionally, it was found that the mean number of draws (out of 500 games) was 163, while the mean number of times an agent won (out of 500 games) was 336. It is evident that using a classifier within MCTS algorithm allows to reduce the number of ties when playing many games. To be confident about how many times a game ends up in a tie (out of 500 games), bootstrap (100,000 iterations and 90% confidence interval) was performed, and it was restated that the mean number is 163 games, and a lower and upper bound of 155 and 171 respectively. Comparing the values from both configurations (default MCTS vs enhanced MCTS), it can be seen that a classifier makes a player become expert in game, which is reflected in how many times the game ends with a positive result (not a draw), and how in Figure 5 ,both lines (% wins and draws) are closer.

To be completely correct and refute our null hypothesis, a permutation test (200,000 iterations) was made, using the number of wins from both configurations and the resulting p value was 0.0. This allows to be sure that using a classifier to select a child node during MCTS, improves the decision making of an agent as the number of wins is significantly higher.

At the same time, the improvement of classifiers along iterations was recorded. Only the graphs that are considered important to show here are presented, and the remaining graphs can be found inside the project repository.

For the first iteration, classifier 10 competed against classifier 1-10 for 50 games each, and the results from this are shown in Figure 6.It is expected that classifier 10 wins the majority of games against classifier 1, meaning that there is a negative relationship. However, it might not be the case when playing against classifier 9 or itself.

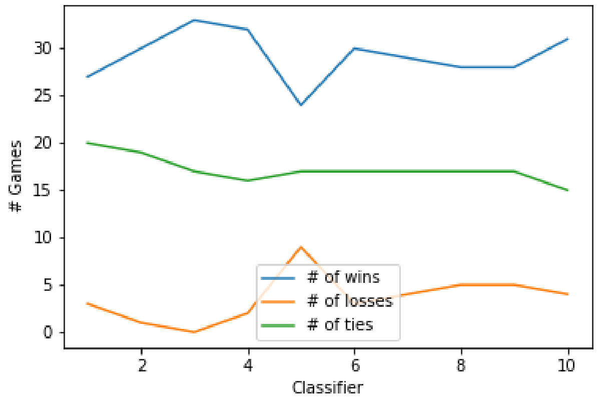

Fig. 6: Classifier 10 vs Classifier 1-10

However, it was found that this behaviour was not present, even to the point, where classifier 10 won more games against itself, than classifier 1. A Pearson's correlation test was performed for the 50 times classifier M played against its 10 previous classifiers, and out of the 50 iterations,there were only two cases (see Fig 7 and Fig 8) where there was a moderate negative relationship as expected, however, the p-value for these cases was not lower than 0.05, which can mean that the results are by chance.

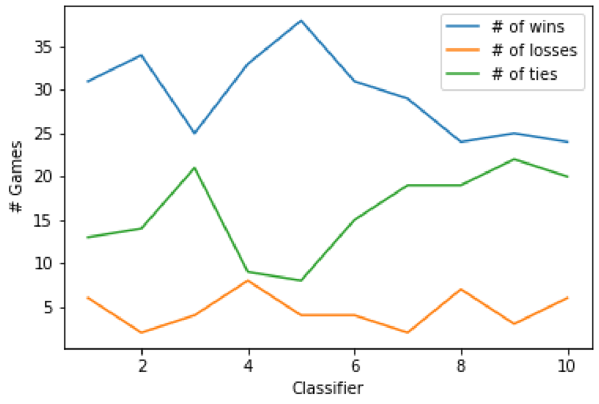

Fig. 7: Classifier 50 vs Classifier 41-50

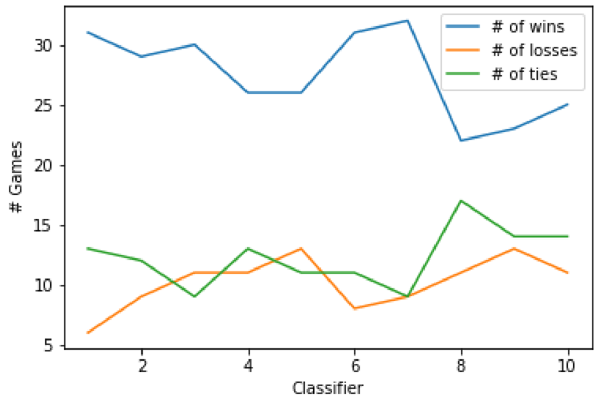

Fig. 8: Classifier 490 vs Classifier 481-490

On the other hand, there were cases where a strong and significant (p-value less than 0.05) positive relationship was found. These cases are shown by Fig 9 and Fig 10.

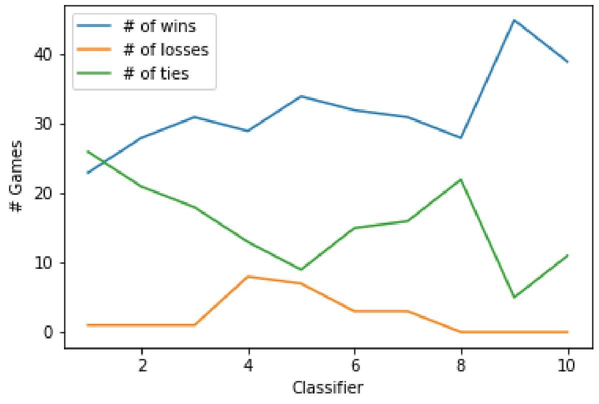

Fig. 9: Classifier 70 vs Classifier 61-70

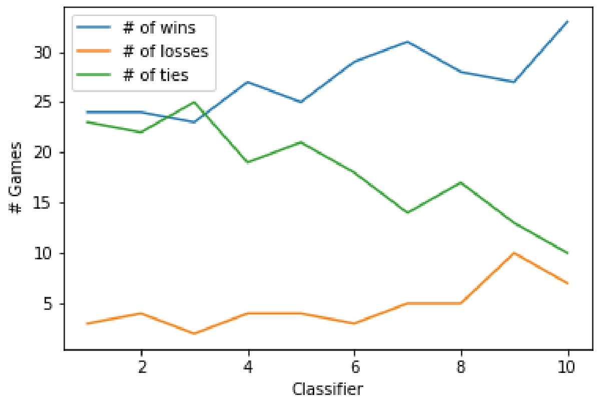

Fig. 10: Classifier 340 vs Classifier 331-340

## 5 Discussion

As mentioned before, the objective of this work was to show how a classifier can make an agent expert in Tic-Tac-Toe. From the results, it can be seen that the default code that use MCTS algorithm, the one used to generate the first dataset, is not able to make a player an expert in the game. This is shown by the number of times a game ends up in a draw. On the other hand, the results show that when using a trained classifier to select the child node that the MCTS algorithm expands, the number of times an agent win is higher. After making a permutation test, it can be concluded that using the classifier, makes the agents perform better.

Also, comparing a classifier to his predecessors, it was expected to see that the latest classifier perform better and have a negative relationship as his opponent change to a higher classifier. Nonetheless, it was found that this was not the case, and in most cases, there was no correlation. It was also found that there were cases that the predicted outcome was the opposite. This might be because of the number of iterations that the MCTS simulate games, which it was set to 50 for all cases.

For future implementations, it will be interesting to see if a trained classifier can beat an agent that use a MCTS algorithm that randomly selects the child node. Also, to see if it is possible to replace the MCTS algorithm, by a trained classifier that decently perform against a MCTS algorithm.

## 6 Conclusion

In retrospect, MCTS algorithm has been used to explore possible combinations in games that work with combinations. In this project, MCTS algorithm was used to play games of Tic-Tac-Toe, however, it was found that the agents (players) were not experts in the game. This was shown by the number of times the games ended up in a draw. The random policy that MCTS operates is not sufficient to make an agent expert in this game, as the child node to explore was picked randomly. To solve this limitation, a decision tree was used to help the MCTS algorithm to select the correct child node and improve the chances to create an expert agent. It was proven that by using a classifier, trained with data from the game, the agents improved their decision making and more games were won, and also the number of draws decreased. It was expected that when comparing classifiers among themselves, the latest classifier outperforms his predecessors and show a negative correlation as the opponent changed. However, this was not the case, but it was able to significantly improve the number of positive outcomes when playing a high number of games.

## References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[2] Z. Liu, M. Zhou, W. Cao, Q. Qu, H. W. F. Yeung, and V. Y. Y. Chung, "Towards understanding chinese checkers with heuristics, monte carlo tree search, and deep reinforcement learning," *arXiv preprint arXiv:1903.01747*, 2019.

[3] C. Nunes, M. De Craene, H. Langet, O. Camara, and A. Jonsson, "A monte carlo tree search approach to learning decision trees," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 429–435.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[5] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," in *Advances in Neural Information Processing Systems*, 2017, pp. 5360–5370.

[6] S. S. Esfahlani and G. Wilson, "Development of rehabilitation system (rehabgame) through monte-carlo tree search algorithm," *arXiv preprint arXiv:1804.10381*, 2018.

[7] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.