

CE811: Assignment 1

Tomas Oplatek OPLAT85906

School of Computer Science and Electronic Engineering, University of Essex, UK

To17758@essex.ac.uk

During this project, a learning agent for the board game “The Resistance” was created and analyzed. The agent used a decision tree classifier as a way to learn to predict another player’s behavior. The project describes the motivation behind this decision and a comparison between expected and actual behavior. The resulting agent can improve its overall win-rate over multiple consecutive games, which indicates that decision tree classifier can be used effectively when creating an AI for this game.

1 Introduction

The purpose of this project was to create an artificial intelligence agent capable of playing the board game “The Resistance”.

This report presents two agents, one of which is playing the game just by following predefined actions, while the other one tries to learn on its own by predicting player’s behaviors based on their previous actions.

The performance of the learning agent is then analyzed and compared to other existing agents in multiple runs of 5 player games.

Finally, the self-learning agent will be pitted against other agents created for the assignment.

2 The Resistance

2.1 Game description

The Resistance is a 5-10 players game made by Don Eskridge where people try to guess each other’s identities. The players are split into two groups, where one group represents the resistance, while the other one represents the spies. The game is set in a Sci-Fi universe where a small group of rebels wants to end the government reign. The government knows about this plot and tries to stop it by infiltrating the resistance.

The spies know each other, but the resistance does not.

2.2 Game rules

The game consists of 5 missions. Each mission can result in failure or success. The spies need 3 failed missions to win and the resistance needs 3 missions to succeed. Spies also win automatically after 5 consecutive mission rejections.

Each turn a leader is appointed that must prepare a team that will go on the mission. The rest of the players then vote in public on whether this team should go on the mission or if a different leader should be selected. If the majority of the players votes for the mission to go on, the players on the team vote in secret on the outcome of the mission. The resistance players must vote for the mission success, and the spies can choose to sabotage the mission.

2.3 Game AI research

There have been many attempts to create an AI for this game, mainly because it was made popular by an AI competition in August 2012, which was organized by Alex J. Champanard. [1] The API in Python he created for this competition is the one that will be used for the purpose of this project, and will be described in the detail in the later sections. Some of the bots that were submitted to this competition will be used as part of this project when testing the newly created bot's performance.

While there have been attempts to create AI's for The Resistance, there are only few studies that do closely investigate the available AI techniques for this game. [5]. Other board games and card games have recently been mastered by AI, employing different techniques.

In 2017, the best ranked player of the ancient board game Go was beaten by the DeepMind AI developed by Google. This AI uses a combination of neural networks and Monte Carlo tree search algorithm. [6]

In 2016, the algorithm DeepStack managed to beat professional players Poker players, although only at the Heads-Up (meaning 1 versus 1) No-Limit Texas Hold'Em version: "DeepStack avoids reasoning about the full remaining game by substituting computation beyond a certain depth with a fast-approximate estimate. Automatically trained with deep learning, DeepStack's "intuition" gives a gut feeling of the value of holding any cards in any situation." [7]

3 Background

Both agent's decision making is based upon trying to predict the chances of other players being spies. The probability of being spy is then used when selecting a team as a leader and when voting whether other selected teams can be trusted.

3.1 Simple agent

The first agent does not use any sort of machine learning and just try to play based on the information it gathers during a single game, it does not keep any statistics between

the games. A dictionary is used to keep track of the count of sabotaged missions that each player was part of. When the agent is a leader, it tries to assemble the team from the players with lowest count of sabotaged missions.

If the agent is a spy, it will only vote for a team that has at least one spy in it.

3.2 Learning agent

This agent tries to learn other players behaviour by observing and saving the data. It uses a decision tree, which is a type of supervised learning.

3.3 Decision tree

A decision tree allows us to represent a dataset in the form of a tree. We need to support it with a dataset that is separated into N predictors and 1 target. There are many algorithms that can be used to create such a tree, but the basic principle is that the algorithm divides the dataset into smaller subsets and creates decision nodes and leaf nodes. Decision node split the dataset further, while leaf node represents the final target classification (Figure 1). [2]



Figure 1 - Decision Tree Example [2]

By using this process, we should be able to predict target value based on the predictors. In the case of the project, this technique should allow us to predict whether the player is a spy or not. The accuracy of this prediction depends on many factors, ranging from the quality and quantity of the data provided, to the actual algorithm used when constructing the tree.

The big advantage of decision trees is their interpretability, they can be used for both classification and regression, and they can be easily converted to if/else statements which are easy to program. [8]

4 Techniques Implemented

The learning agent uses a decision tree as a way of predicting whether the other player is a spy.

4.1 Scikit-learn decision tree

The decision tree of the agent is constructed by the open-source python library scikit-learn [3]. The agent uses a `DecisionTreeClassifier` class which, after being fitted with data, allows us to predict whether the player is a spy. The algorithm used by scikit to construct the tree is called CART (Classification and Regression Trees). [4]

4.2 Data collection

The agent collects data throughout that game and stores it in `GameStats` class. After the game is completed, the agent goes through the collected data and creates a data structure for each player that can be used by scikit or Weka. This structure for each player is then saved in a static class called `GameBlackboard`, where it is available to be accessed between the games.

The data that are collected during a single game are stored in separate arrays and consist of:

- Players – The players that are taking part in this game instance
- Teams – List of teams that were proposed in chronological order
- Votes – 2D array of all votes for each team in chronological order, the index of the vote is the same as the player it belongs to from the players array
- Leaders – Who was the leader of each team
- Spies – Who the spies were
- Player stats – A class that contains all predictors and targets relevant to each player generated from `GameStats` (Table 1)

Predictors							Target
Voted up	Is leader	Team size	Mission result	Is on team	Turn	Try	Is spy
False	False	2	'Completed'	False	1	1	True
True	False	3	'Sabotaged'	True	2	1	True
False	False	2	'Failed'	False	3	1	True
False	False	2	'Completed'	False	3	2	True
False	False	2	'Completed'	False	4	1	True
True	True	3	'Failed'	True	4	2	True
False	False	3	'Failed'	False	4	3	True
True	False	3	'Failed'	False	4	4	True
True	False	2	'Completed'	False	1	1	False
True	False	3	'Failed'	True	2	1	False
True	True	3	'Sabotaged'	True	2	2	False
True	True	3	'Sabotaged'	True	3	1	False
True	False	2	'Sabotaged'	False	4	1	False
True	False	3	'Failed'	True	4	2	False
True	False	2	'Sabotaged'	False	1	1	False

Table 1 - Example data set

4.3 Learning process

The decision tree classifier does currently not use any previously collected data from previous games at the start of the game. This decision was made because there will be different bots during the final competition than were available during the testing; therefore, it made more sense to let the agent learn on its own.

This of course brings the challenge of deciding when we have enough data for the classifier to make the decision accurately. After some testing, it has been decided that the algorithm will not build the classifier until it has at least 20 data samples from each player. Until then, it uses the same algorithm as the simple agent.

When it has 20 or more data samples from each player in the current game, the classifier can predict whether each player is a spy based on their actions in the previous turns/tries.

At the start of the game, a dictionary is created with each player's name, and each player is assigned a probability of being a spy, which is zero at the start. After each mission attempt, the classifier decides whether it thinks the player is a spy or not. If it thinks the player is a spy in this attempt, it will increment a `spy_chance_total`

counter in the local PlayerStats class. After this process, the actual probability of the player being a spy is calculated as:

$$\text{spy_chance_total} / (\text{turn} + \text{tries} - 1)$$

4.4 Decision influence

There are two main areas in which the spy probability influences the agent's decisions. Firstly, it changes what players the agent picks during the team selection. This only applies when the agent plays as a resistance. Secondly, it changes how the agent votes.

4.4.1 Selecting a team

The process that happens during team selection is very straightforward. The agent always selects itself as a part of the team. If the agent itself is a spy, it randomly selects the remaining members of the team from other players who are not spies.

However, if the agent is part of the resistance, it will try and add other members that have the lowest probability of being spies. The dictionary with player's spy probabilities is sorted by the probabilities and players are added as members in that order, until there are enough members in the team.

This should provide the best chances of success for the resistance and for the agent.

4.5 Voting on a mission attempt

When it comes to voting, the agent will always vote positively for mission attempts where he is the leader. If he is not a leader, then the agent's actions depend on whether he is a spy or a member of the resistance. The spy will only vote for missions that he knows have spies in the team.

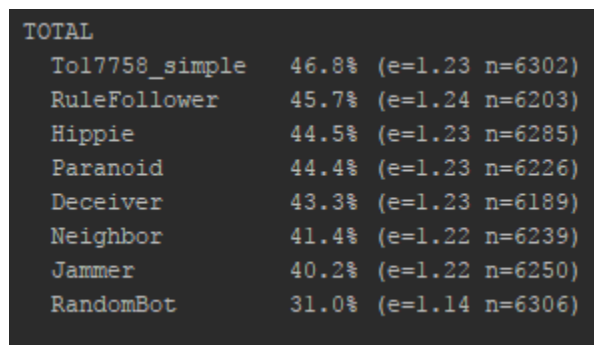
On the other hand, the resistance agent will check the members of the team, and see what their spy probabilities are. If it finds out that the player with the highest chance of being a spy is on the team, it will vote against it. Otherwise, it will vote positively.

5 Experimental Study

During the development of my agent it has been tested extensively against other bots that were provided during the course. Bots of different types and quality were available for competition. Both the simple agent and learning agent were tested in batches of 1000 games and the results were compared. The learning agent output was also provided in the form of an arff file for further analysis in the Weka software tool.

5.1 Simple agent

The simple agent was created during the first labs. The initial experiments were run against the bots from `beginners.py`. The goal was to create an agent that would defeat the beginner bots regularly and that would use common sense while making its decisions. This just meant creating an agent that would keep track of the players that were on the team during a sabotaged mission, following the logic that these players have a higher likelihood of being spies. Just this simple modification allowed the agent to beat the other beginner bots fairly consistently (Figure 2).



TOTAL		
Tol7758_simple	46.8%	(e=1.23 n=6302)
RuleFollower	45.7%	(e=1.24 n=6203)
Hippie	44.5%	(e=1.23 n=6285)
Paranoid	44.4%	(e=1.23 n=6226)
Deceiver	43.3%	(e=1.23 n=6189)
Neighbor	41.4%	(e=1.22 n=6239)
Jammer	40.2%	(e=1.22 n=6250)
RandomBot	31.0%	(e=1.14 n=6306)

Figure 2 - Simple bot against beginners.py in 6302 games

This was considered a good enough result and the focus of the project then shifted towards creating a machine learning agent.

5.2 Learning agent

This agent was supposed to learn on its own between the games, therefore the first thing necessary was to create a data structure in which to store all the data about the game and players. The final dataset is described in chapter 5.2.

At first it was attempted to make a single tree for all the game results and players, where player name was one of the predictors. This tree would grow unnecessarily large and would not be as effective as a unique tree for each player. Nonetheless, when removing the player name predictor, this tree could be used to predict the accuracy of the model. The tool Weka was used to do this (Figure 3 and Figure 4).

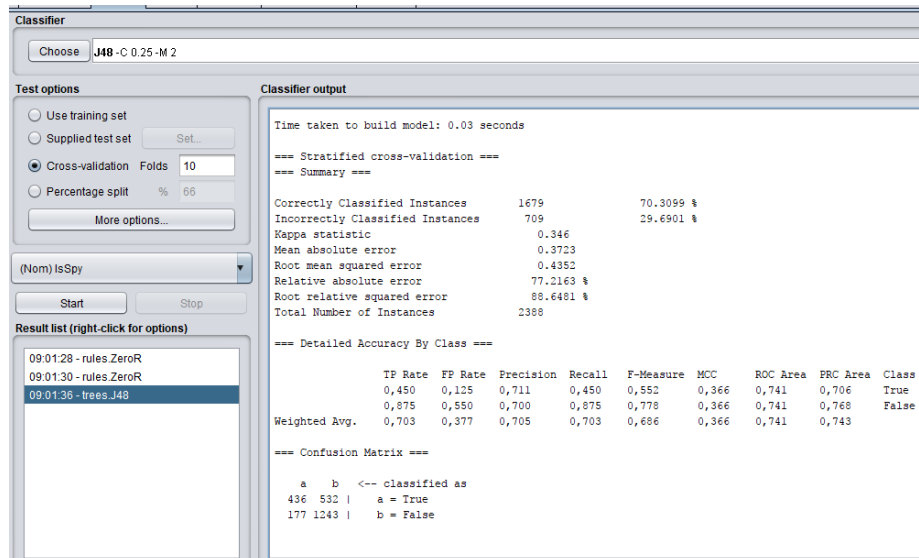


Figure 3 - Weka J48 Tree classifier

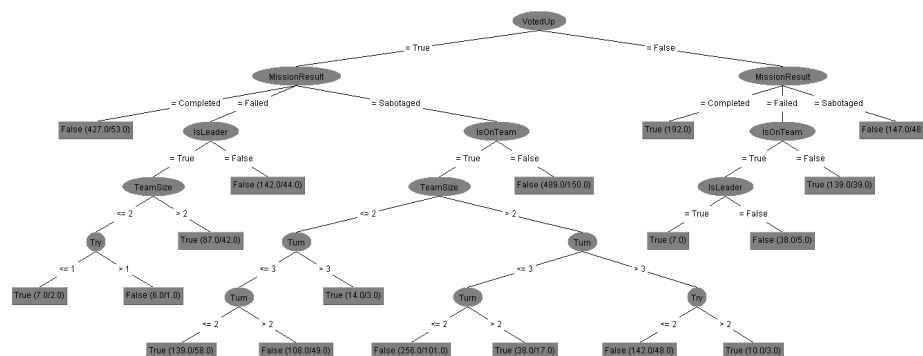


Figure 4 - Weka J48 Tree visualization

These results were obtained by a 10-fold Cross-validation. This means that the data provided is split into 10 folds, where 1 fold is used to construct the decision tree and the remaining 9 folds are used as a test set. The model was able to correctly classify 70.3099% percent of instances, and this result was considered to be adequate.

The confusion matrix shows that the classifier is very good when predicting if the player is not a spy, but it has a rather large number of false-positives, meaning that it presumes that player is a spy when in reality he is not.

5.2.1 Performance

Since this agent learns from other players and then tries to guess their identity, its performance should largely depend on how many games in sequence is it allowed to play. The lower the number of games, the more will the performance be similar to the simple agent.

6 Analysis

6.1 Development process

During the development process of the simple agent, even the small change of tracking players who were on sabotaged missions improved the agents score. That is why this simple agent was used as a basis for the learner agent.

For the learner agent, tracking the chance of players being spies and using that information when selecting a team worked very well, leading to an improvement in the agents score. This agent focused heavily on playing as a resistance, the spies mechanics are the same as for the simple agent.

One thing that did not work very well was the implementation of a mechanism that will randomly decide whether to sabotage the mission in the first round when the agent is a spy. This mechanism was tried because it was assumed that it could mask the identity of the spy agent in the first round, where players would normally always sabotage the mission. However, this did not produce the intended results, the agent scored worse than normally. The reason for that might be that most of the bots in the competition do not automatically assume that the player is a spy if he is in a team that sabotaged the first mission. On the other hand, this approach might work better when the agent is pitted against other learning agents, because it will disrupt the pattern in his behavior, making it harder to predict it.

6.1.1 Voting

During the development and testing, it turned out that the system implemented in the voting method only decreased the agent's performance. It was surprising that this occurred, because voting true for every team produced far better results. It seems that voting false increases the likelihood of being labeled as a spy, which lead to the resistance agent bad performance (Figure 5).

TOTAL		
Suspicious	52.5%	(e=4.00 n=594)
Logicalton	47.6%	(e=3.88 n=634)
Bounder	46.1%	(e=3.90 n=623)
Simpleton	44.1%	(e=3.87 n=627)
Statistician	43.4%	(e=3.82 n=643)
Tol7758_simple	40.9%	(e=3.95 n=590)
Tol7758	38.7%	(e=3.72 n=655)
Trickerton	37.8%	(e=3.76 n=634)

Figure 5 - 600 games with modified voting

After checking the algorithm, it turned out that it was assigning the spy probabilities when voting in the opposite order. This mistake was fixed, which resulted in a much better performance. (Figure 6).

TOTAL		
Suspicious	50.5%	(e=3.81 n=659)
Logicalton	50.4%	(e=3.94 n=615)
Tol7758	46.3%	(e=3.90 n=624)
Simpleton	45.8%	(e=3.87 n=632)
Bounder	44.4%	(e=3.88 n=626)
Statistician	41.5%	(e=3.84 n=629)
Trickerton	38.9%	(e=3.79 n=633)
Tol7758_simple	38.9%	(e=3.95 n=582)

Figure 6 - 600 games with correct voting

6.2 Methodology for obtaining results

The results are presented in the form of graphs, split into 3 categories. The agent should show an increase in performance and score stability the longer it can run and gather data. To test this assumption, the agent was pitted against the bots in files experts.py, learners.py, intermediates.py and tol7758_simple.py (which is the simple version of the agent). The first competition was 500 consecutive games, the second one was 1000, and the last one was 3000. The competition ran in one thread, so that the agent could access data from each game. Since there were 8 competitors, each player took part in approximately 320, 620, and 1860 games. Each one of these competitions was completed 10 times and the results from each are provided in the graphs, as well as a total average. The graphs track the overall win percentage, the percentage of voting for a resistance team, and the percentage of voting against a mission with spies in the team.

6.3 Results

batch	overall win-rate	voting for resistance	voting against spies
1	45,80%	92,10%	32,70%
2	44,70%	92,70%	36,00%
3	45,20%	92,70%	35,40%
4	45,30%	92,10%	35,00%
5	45,00%	91,50%	34,80%
6	44,10%	91,40%	33,70%
7	44,20%	91,10%	33,40%
8	45,00%	91,60%	33,50%
9	44,40%	91,80%	33,60%
10	44,60%	91,80%	33,80%
AVG	44,83%	91,88%	34,19%

Table 2 - 310 Consecutive Games

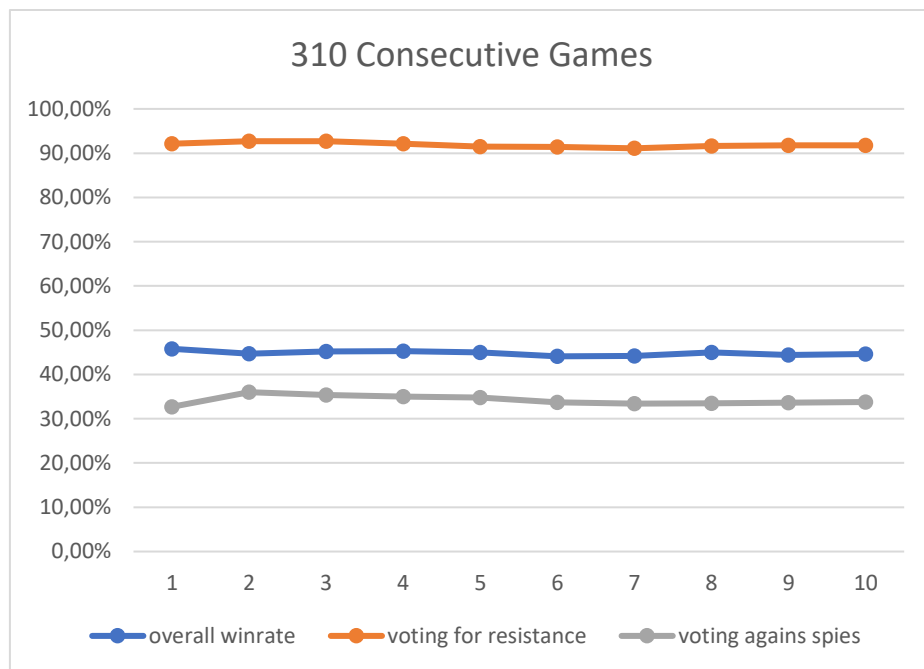


Figure 7 - 310 Consecutive Games Graph

batch	overall win-rate	voting for resistance	voting against spies
1	44,90%	90,80%	32,60%
2	45,20%	91,10%	32,80%
3	44,80%	91,20%	32,40%
4	45,50%	91,10%	32,70%
5	45,40%	90,70%	33,00%
6	45,40%	91,20%	33,10%
7	45,90%	91,50%	33,20%
8	46,00%	91,40%	32,90%
9	46,20%	91,50%	33,00%
10	46,30%	91,90%	33,30%
AVG	45,56%	91,24%	32,90%

Table 3 - 620 Consecutive Games

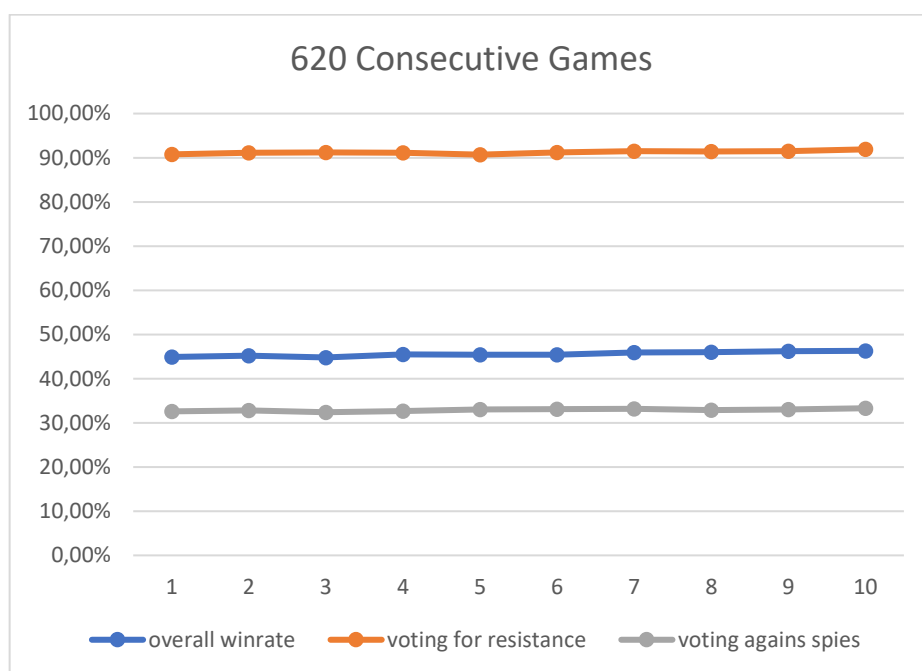


Figure 8 - 620 Consecutive Games Graph

batch	overall win-rate	voting for resistance	voting against spies
1	46,50%	92,50%	31,90%
2	46,20%	91,80%	33,40%
3	45,50%	91,70%	33,90%
4	45,00%	91,50%	33,90%
5	45,10%	91,30%	33,60%
6	45,40%	91,40%	33,70%
7	45,70%	91,70%	33,50%
8	45,50%	91,60%	33,50%
9	45,70%	91,70%	33,60%
10	45,70%	91,70%	33,40%
AVG	45,63%	91,69%	33,44%

Table 4 - 1860 Consecutive Games

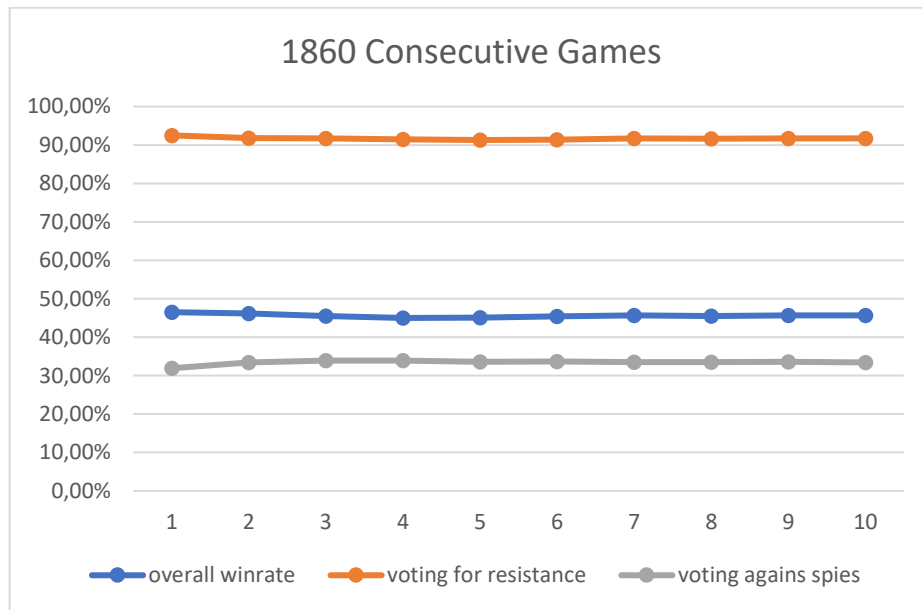


Figure 9 - 1860 Consecutive Games Graph

6.4 Results analysis

The results show that the agent is improving its performance between games, which is shown by the average increase in win percentage between the tests (Figure 10). After playing 620 consecutive games instead of 310, the agent did improve the win

percentage by 0,73%. After playing 1860 games, the agent also showed an increased win-rate, but now the improvement was much lower at 0,07%.

The results stayed very stable during each batch, showing no drastic differences (Figure 7, 8, 9). However, the voting accuracy went down between the first two tests, and only started improving again in the last test run (Table 2, 3, 4).

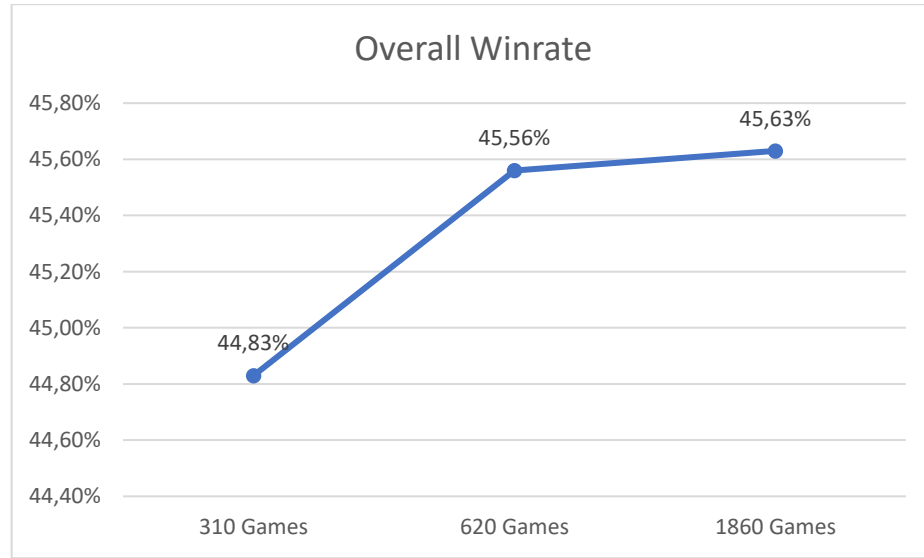


Figure 10 - Overall Win-rate Percentage

The system for selecting players for a team with a decision tree is working well, because the overall win-rate is increasing with more consecutive games played. The voting system on the other hand is not working as expected. It should get more precise the more the agent plays, but in fact it is doing the opposite. This might be caused by the rather large number of false-positives that the classifier produces, and by its very simple algorithm.

7 Conclusions and Future Work

This project has showed that decision tree classifier can be effectively used in an AI for The Resistance, resulting in an agent that can improve its win-rate by viewing other player's actions.

The work could be further extended in many ways. Firstly, this agent focused mainly on playing as a resistance player and identifying spies. Improving the ability to play as a spy might show further improvements. Secondly, the voting algorithm of this agent is very simple and could be expanded upon. Finally, there should be ways to improve the decision tree classifier, either by providing it more data about player's behavior, or by improving the tree itself by optimizing it with pruning.

References

1. AiGameDev.com Announcing THE RESISTANCE A.I. Competition: Build Your Spy Bot! Online at: <http://aigamedev.com/open/upcoming/resistance-competition/> [Accessed on: 06 November 2017]
2. Saedsayad.com Decision Tree - Classification Online at: http://www.saedsayad.com/decision_tree.htm/ [Accessed on: 06 November 2017]
3. Scikit-learn Machine Learning in Python Online at: <http://scikit-learn.org/stable/> [Accessed on: 06 November 2017]
4. Scikit-learn Decision Trees Online at: <http://scikit-learn.org/stable/modules/tree.html/> [Accessed on: 06 November 2017]
5. TAYLOR, Daniel Peter (2014) Investigating Approaches to AI for Trust-based, Multi-agent Board Games with Imperfect Information; With Don Eskridge's ``The Resistance''. BSc thesis. University of Derby
6. David Silver & Aja Huang (2016) 'Mastering the game of Go with deep neural networks and tree search' *Nature* **529**, 484–489
7. DeepStack Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker Online at: <https://www.deepstack.ai/> [Accessed on: 09 November 2017]
8. Alpaydin E (2014) *Introduction to machine learning* Cambridge, Massachusetts: The MIT Press