AI Agent for ”The resistance” board game

Gonzalez, Ricardo J

School of Computer Science and Electronic Engineering
University of Essex, UK
rg20332@essex.ac.uk

Abstract. During this project an agent was developed for the game The Resistance using techniques of Neural Networks, Learning Decision Trees and Genetic Algorithms, with the objective of being a good spy player by hiding its true purpose from the resistance members.
The resulting agent ended as a spy very capable of not showing that it was the spy, but lack aggressiveness and therefore didn't had a very good win-rate

1 Introduction

The objective of this research is to develop an agent for the game: ”Avalon: The Resistance” the development of the overall agent is centered around the idea of make a agent which is a very good spy, but that it keeps itself well hidden among the resistance members.

2 The resistance

2.1 What is the resistance?

Avalon: The Resistance is a party game with social interaction. Designed to be played with 5 to 10 players, these are later on divided randomly in members of the *resistance* or *spies* [1].

Mechanics Part of the mechanics of the game is that the spies know who are the other spies, while the resistance members doesn't know who may or not be a spy. Thus, being the discovery of who is spy and who's not an essential part of the game as a resistance member.

Flow The flow of the game consists of three to five rounds/missions. In each of these missions phases happens in the following order:

1. From all the players a leader is chosen
2. The leader selects players to be part of the mission (being either the player itself and other players)
3. All the players votes if they agree with the selection of the players.

4. If most of the players votes negatively to the team, then another leader is chosen, thus we return to step 2.
5. If most of the players votes positive, then the players "go" to the mission
6. While at the mission, resistance members give a card that represents that they will not sabotage the mission. Nevertheless, if one of the player is a spy, he can instead decide to give a card that represents that they will sabotage the mission. *Note*. No one knows who votes what, they just know that the cards exists overall
7. If at least one player has voted to sabotage the mission the mission fails

Who wins? If three missions are sabotaged the team of spies win the game. Otherwise, the resistance wins it

2.2 Previous research surrounding the game

One important research done in The Resistance: Avalon was done by MIT and Harvard academics. In the they explain that using **DeepRole** algorithm to create a multi-agent reinforcement learning agent that's able to learn the game.

The way this agent learns is by playing several games against itself named as counterfactual regret minimization (CFR), with an augmentation of *deductive reasoning*. At each point in the game, the CFR looks ahed to create a decision tree, so it can plan an optimal strategy where at worst it ties against any opponent [2].

Nevertheless, nothing is perfect, as Serrino says "Language is definitely the next frontier." Mainly because in this type of games, one important way to bluff the other players of the reality is by communication with them. And at the same time the communication that other players may have with you could be a hint to what is they're actual alignment

3 Background

3.1 Agent

The agent decides some of its decision by using ML or NN models/solutions to know the probability of a good outcome for itself or for the team overall. But also it has some behavior of only acting of the current situation of the game, without taking in consideration his previous saved matches.

3.2 Supervised Learning

With supervise learning the model has the objective of approximating itself to an underlying function based on some labeled data, being tested its effectiveness with unseen examples.

3.3 Feed Forward Neural Networks

Is a type of Artificial Neural Network (ANN), being this that it's a network represented by layers that contains nodes and those nodes are connected with the nodes of the next layer. The Feed Forward behavior happens when no backward-facing connections exist, thus we are only facing forwards in the network [3].

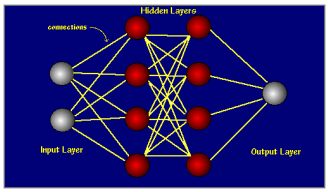

Fig. 1: Example of a Neural Network

3.4 Learning Decision tree

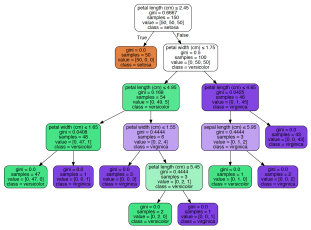

Fig. 2: Example of a Learning Decision Tree

A modification of a normal decision tree that consists of decision nodes and leaf nodes, but instead of the decision tree being populated by human interaction it's instead done by using an ML algorithm that populates the decision nodes based on data provided [4].

The biggest advantage of using a decision tree is being able to be understood very easy by it simple structure and logic.

3.5 Genetic Algorithm

A Genetic/Evolutionary Algorithm has its origin of the concept of *Natural Selection* where a *population* of *individuals* with random *attributes* compete to be able to *reproduce* based on their fitness resulting into an offspring. Those offspring being a combination of its parents, are able to mutate by a random chance leading to a different fitness, leading to a new *generation* of new individuals competing with each other [5].

This solution has the objective of reducing human interaction for looking for ideal combination of fine variables, being a specific number or specific set of numbers in an specific order.

4 Techniques Implemented

The agent was developed with the idea to perform decently when playing as a spy, no matter who is playing against, thus molding the overall algorithms related to this. The

algorithm of the bot can be diversified mainly in the 3 principal actions *sabotage*, *vote* and *select*

4.1 Sabotage

The sabotage behavior when being a spy didn't required extra techniques, mainly done with decision tree that says that the agent will only sabotage if this is the last mission or the team is of at least of 3 players and there's no other spy in it. The objective of this behavior is:

1. Reduce the possibility that the other players know that the agent's the spy
2. Avoid the possible coordination problem between the other spy and itself, and that both players actually sabotage thus providing of more knowledge to the other players than necessary.

4.2 Vote

The vote behavior is based if the user is or not a spy. If it's not a spy it will follow a normal decision tree algorithm, voting positive if either of the following happens:

- Team is of two people
- The agent is on the team

But, if the agent's a spy it will instead based its decision using a decision tree classifier.

4.3 Selecting

This behavior is overall the same either if the agent is a spy or not, the algorithm does the following:

1. If the agent's a spy it will not consider the other spies as part of its possible team
2. Calculates the possibility that a player is a spy
3. Orders the player by its possibility of being a spy in an ascendant order
4. The final team will be himself in addition of the first n - 1 players of the ordered list of players, being n the size of the team for the mission

4.4 Management of state

The state that will be managed are:

- `missions_been_on`: A dictionary that the player has been
- `failed_missions_been_on`: A dictionary that the player has been and has failed
- `suspect_missions_voted_up`: A dictionary that contains an array of the missions that the user has voted up, being their indexes the amount of missions the team has failed

- `suspect_missions_voted_down`: A dictionary that contains an array of the missions that the user has voted down, being their indexes the amount of missions the team has failed
- `perfect_team_counts`: An array of 5 elements representing each mission. Each of the elements of the array is a tuple containing the average spy probability of the team in the mission, the average spy probability of all the players, if the agent is a spy and if the mission was successful for the agent (Was sabotaged if spy, was not sabotage if resistance), those elements in that order

The way this state is managed is as the following:

1. When the game starts the state's variables are initialized with their initial value
2. When vote was completed the state will be updated for `suspect_missions_voted_down` and `suspect_missions_voted_up` with the new values for each player, according to the suspect value of the team
3. After the mission is completed. It will update the values of `missions_been_on`, `failed_missions_been_on` and `perfect_team_counts`

4.5 Data collection

With the state, the agent can log it state at the end of the game into 2 different CSVs. One CSV has the objective of figuring out to predict if a player is a spy or not. While the other has the objective of predicting if it's the best to vote positive or negative for this team

This data was gather by playing against the following set of opponents 10,000 games each:

- Beginners
- Intermediates
- Experts

The first csv consisted of the use of the information of the following attributes of the state:

- `missions_been_on`
- `failed_missions_been_on`
- `suspect_missions_voted_up`
- `suspect_missions_voted_down`

While the second csv uses only the state's attribute of `perfect_team_counts`in addition of the number of the mission it was. Each mission being a different row of the csv, an example of this data can be shown in figure 3

Tournament Number	Win Percentage as Spy	Selection Percentage	General win Percentage
1	68.9%	52.4%	35.7%
2	69.2%	52.5%	37.0%
3	69.8%	52.1%	37.2%
4	70.2%	51.8%	37.8%
5	70.9%	52.3%	38.3%
6	71.1%	52.1%	39.0%
7	71.3%	51.8%	36.4%
8	72.0%	52.5%	36.2%
9	72.6%	52.7%	40.4%
10	72.8%	53.0%	35.3%
AVG	**70.88%**	**52.32%**	**37.33%**

Fig. 3: Example of information of the vote info data

4.6 Data preprocessing

After the data was collected, it was then preprocessed, deleting the duplicates of the data and creating new sets of data containing the same data but normalized. This was with the objective of reducing the possible noise of some values, being because they were several rows containing the same data, thus the model risking itself to become overfited

4.7 Decision Tree Learning

A decision tree classifier was developed using the scikit-learn library. The specific tree used was a `DecisionTreeClassifier`, that classified based on the voting-related data if the agent should vote towards or against the mission.

4.8 Neural Network Architecture

Using the tensorflow and keras library, a NN was developed which architecture consisted of an input layer of 16 nodes, three hidden layers each of them with 10 nodes and an output layer of 2 nodes. This neural network was trained with the information related to identifying if a player was a spy or not.

The hidden layer's nodes had a *tanh* activation function, while the output layer had a *softmax* activation function. For fitting the Network it was used an *Adam* optimizer with a learning rate of **0.001** and the use of *SparseCategoricalCrossentropy* as loss function

4.9 Genetic Algorithm

Nevertheless, a secondary objective of this research was to see how a neural network could be optimized using a GA, being the objective of the GA to found the best weights for the NN.

Genes The genes were represented as a matrix, being each row the weights for each layer of the individual

Crossover For the crossover, it was used the technique of a single point crossover, being that before a random point of the gene of the offspring there would be all the genes of one of the parents, and after that random point the genes of the other parent.

Mutation The mutation algorithm used has an incident rate of 10% per element of the gene, aka. per layer. If the mutation happens the weights of that layer will be modified by a random factor between 0 and 5 inclusive. *Note* an individual can suffer of 1 or more mutations.

Pool It was used a pool which contained the top 100 individuals between all generations since the start, from this pool each generation is chosen both the top 5 individuals and the 4 partners of those best individuals selected by our selection algorithm.

Selection The type of selection chosen for the mating of the top 5 individuals was a tournament selection between the best of a random sample of the pool representing the 20% of the pool

Number of population and generations The initial and future populations will always be of 20 individuals, and this happens during 100 generations

Work done with the best individual After running the 100 generations the best configuration of the model is ran through 20 epochs.

5 Experimental Study

5.1 Splits of the data

The way all the of the data was splitted for both the NN and the decision tree

5.2 Perfect Genetic Algorithm

Most of the parameter tuning was done with the genetic algorithm, as it was the first step of the development.

Originally there weren't any pruning for the best individuals in the pool, this decision was made later on to improve the time complexity of the algorithm, also to made sure that no progress was lost.

Another aspect of constant change in the algorithm was the amount of population and generations to process. Originally having little amount of both variables to be able to quickly test of any bug. But later on due to the poor performance decided to instead increase to 80 population and 200 generations, this led to almost the same result, but

was obviously more time consuming. Thus decide to reduce both variables to a 20 population and 50 generations set. This was decent, but I noticed that probably after the 50th generation there would be more individuals that may be more suited. Thus decided instead to keep the population but increase the generations. This resulted in a very efficient code in time complexity and that gave an individual with a good fitness.

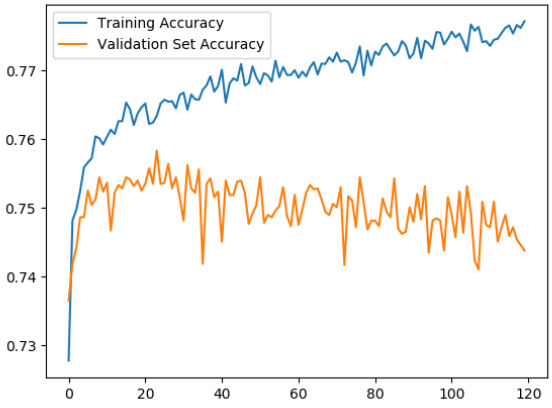

Fig. 4: Accuracy of the final model for classifying spy

This code resulted in a model with the accuracy shown in the figure 4, as we can see it has an overall good accuracy. But, it seems based on the information of the validation data, that it may be a little bit overfitted more towards the training data.

5.3 Decision Tree

The `DecisionTreeClassifier` used was tweak specially in which would be the max-depth of the model. After some experimentation, the conclusion was that the best would be to have a maximum of 10 levels of depth. This lead to an accuracy of around 90% and a final tree as shown in figure 5.

5.4 Change of behavior

Originally when the agent was a member of the resistance it would also use the classifier. Nevertheless, when testing it performance the winning percent when playing against intermediate players were less than one percent.

I noticed that the main reason was because of the voting action, thus decided to instead use a different behavior for voting in this scenario. This resulted in a 20% winning rate when playing as the resistance.

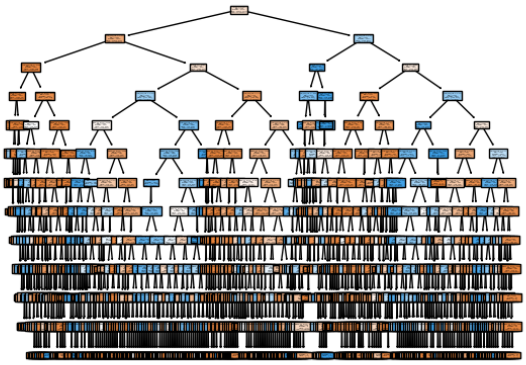

Fig. 5: Final Decision Tree

6 Analysis

6.1 Development process

With the objective that the agent as a spy was able to disguise itself as best as possible was interesting to see how sabotaging or not in certain situations change how the other players may see you and the team overall.

Thus, there were an importance overall in deciding which would be the best behavior for the agent when being a spy and disguising itself as a resistance member. This seems to be an overall combination of all the actions: *vote*, *sabotage* and *select*.

Nevertheless for also seeing how well overall the agent performed, the agent should also perform decently as a resistance. This led to a change previously mentioned of instead of using the vote classifier the same as with spy, change it with another behavior. As it seems like the other agents were very sensitive to my agent voting against a team, because the classifier said that it would not be positive for the resistance's objective.

6.2 Methodology for obtaining results

The main objective of the agent is to perform decently against any player while also keeping itself hidden from suspicion of being a spy. Thus, for taking metrics of the results the agent will play in 10 tournaments of 5000 matches each against beginners, intermediates and experts. From this matches, the primarily focus will be win percentage of the agent while being a spy and the percentage of being selected as part of the team. The former matters because is how I will measure how good the agent is to disguise itself while playing as the spy role

6.3 Results

Mission #	Team Avg Prob	Players Avg Prob	Spy	Success
1	0.7629306316375732	0.3743550598621369	1	0
2	0.7391705214977264	0.5136944055557251	1	1
3	0.9265740215778352	0.8634344816207886	1	0
4	0.9801148970921836	0.9303136706352234	1	1
5	0.9907341996828716	0.9839640498161316	1	1

Fig. 6: Experiment results of the final agent version

6.4 Results analysis

Based on the previous data it seems like the most of the time the agent is able to keep itself disguised as a resistance member, because he's selected to be part of the team most of the time he plays, being selected in average 52.32% of time. More than the average of a naive or brute agent playing spy.

Nevertheless the overall win rate, in general and as spy it's something worrying. When playing with the other players the agent was most of the time among the worst three players of the three categories (Spies, Resistance and General). Thus it could be concluded that probably the reason why it keeps so well its disguise of a resistance member is that it doesn't play as aggressively as it should, leading to a lower win rate and thus performing poorly on that.

7 Overall Conclusions

7.1 Main take-aways

An overall important conclusion everyone can have of this experiment and development of the agent, is that even if the tools and techniques are overall considered advanced and very powerful, they do not necessarily guarantee a perfect agent. It was not an hypothesis that was stated initially, nevertheless the results were not as good as expected. This can lead to the conclusion that more than an advanced/powerful technique or combination of them, the researcher should instead focus on maybe the development of a good architecture or design if the primarily objective is to create a *perfect* agent.

7.2 Difficulties faced

As any other iterative project of try and error, the development of the agent lead to some problems while doing the development. First big problem made was that originally the idea was that the genetic algorithm objective was to estimate the best amount of neurons when having 2 layers. This lead mainly to a hardware limitation, where the time complexity of the solution was not suitable for my hardware (a 2012 laptop with

dual core and without GPU), this happened because for each individual I had to fit the model in order to avoid the possibility of the individual's fitness was biased due to the weights it randomly had at the start, rather than the amount of neurons.

Further difficulties happened when developing my custom logic for the Neural Network Sequential, as originally I had the class as it inheritated from the tensorflow Sequential class, in order to reuse the functions of the class in my cases. Nevertheless, this lead to a bug when saving and then loading the model, thus decided to use instead a composition style for handling the overall operations of the class.

7.3 Future work

Future work that can be done regarding the development of this bot, is experimenting with the performance that it may have when the GA instead of looking for the perfect weights looks instead for the perfect structure design. This inspiration came to me after reading the paper regarding this same topic, when it's not only about the amount of neurons as my original idea was. But instead is the whole architecture of the NN that's being optimized, involving in general more complex algorithms and management of the individual's chromosome [6].

References

1. BoardGame. (2020) The resistance: Avalon. (https://boardgamegeek.com/boardgame/41114/resistance). [Online; Accessed 10-11-2020].
2. J. Serrino, M. Kleiman-Weiner, D. C. Parkes, and J. B. Tenenbaum, "Finding friend and foe in multi-agent games," *arXiv*, 2019, [Online; Accessed 10-11-2020].
3. U. of Wisconsin, "The basics of neural networks," 2020, [Online; Accessed 10-11-2020].
4. S. Savad, "Decision tree - classificacion," 2017, [Online; Accessed 10-11-2020].
5. W. Roetzel, D. Chen, and X. Luo, "Design and operation of heat exchangers and their networks," 2020, [Online; Accessed 10-11-2020].
6. S. Mizuta, T. Sato, D. Lao, M. Ikeda, and T. Shimizu, "Structure design of neural networks using genetic algorithms," 2019, [Online; Accessed 10-11-2020].