Universiteit Leiden
The Netherlands

Informatica en Economie

How to create an agent for the game of Klaverjas using Random Forests

Lennard Hordijk

Supervisors:
J.N. van Rijn, J.K. Vis & P. Koning

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01-08-2022

Abstract

Klaverjas is a trick-taking card game played in two teams of two players. The goal of the game is to earn as many points as possible. In this thesis we create an agent that, using a random forest, predicts which card is the best card in a situation. We do this by first predicting whether a card is playable followed by predicting the card a human player would play in a situation. We analyse two versions of the game Klaverjas: The Amsterdam version and the Rotterdam version. The agent is able to predict the card a human player would play in 81.4% and 80.7% of the game states for the Amsterdam and Rotterdam version respectively. Furthermore, the agent is able to outperform two other agents with different strategies. The agent that plays random cards from the playable cards is beaten in around 59% of the games. A rule-based agent is beaten in around 54% of the games. Additionally, the agent is able to score an average of 101 points per game, while the rule-based agent only manages to score 88 points on average. We expected that our agent was able to win more games against the simple strategies than it did. However, on average the random forest agent scores significantly more points than the rule-based agent each round. We expect that there is room for improvement for this agent to beat the other agents in even more games, but the agent is sufficient at maximising his score in each round of Klaverjas.

Contents

1 Introduction

Nowadays artificial intelligence is a topic of interest. The number of applications for artificial intelligence is increasing and it is used for increasing difficult problems. A few examples of these applications are the use of artificial intelligence in self-driving cars and allowing for personalised advertisements. Artificial intelligence can also be used in games. With the use of reinforcement learning an agent can learn to play a game in the best way possible. This method was applied to the game of Chess and Shogi in 2017 [SHS+17]. Before this application, there were agents that could also play chess better than humans. These agents were mostly rule-based. In [SHS+17] however, the agent created these rules by itself. It was surprising that this agent was able to beat the rule-based agents. In this thesis, we will be looking at the game of Klaverjas. We investigate whether we are able to create an agent that is able to identify which card is best playable in a situation. Just like the game of chess, we need to play the best move as much as possible to be able to outperform humans. In contrast to chess, Klaverjas is a game of imperfect information, while in the game of chess, all possible different game states are visible at any given moment. In Klaverjas the cards of the three other players are not known by the current player and therefore we have a game of imperfect information. To create an agent that is able to predict the best card most of the time, we use a supervised learning method. We apply this to Klaverjas to determine how well it can predict the 'goodness' of a card and in this way determine which card should be played. The goodness of a card represents the likelihood that a human player will play this card. Because a human player always tries to play the best card, we are looking for the best card in a given game state. We define the following research question:

How to create an agent that, using random forests, predicts which card a human would play?

To create this agent we will follow two steps. First, we are going to apply a random forest to predict whether or not a card can be played by the rules of Klaverjas in a certain game state. If we want to apply a random forest to a game state, we need to create a representation of the game state that is readable by the random forest. Furthermore, we need to determine which attributes of a game state determine whether a card can be played or not. We expect that we can reach an accuracy of 100% on this step as a random forest should be able to distinguish the relatively easy rules of Klaverjas. After applying a random forest on predicting whether a card can be played, we create an agent that predicts the card a human player would play in a given game state. As we expect that a human player always chooses to play the best card in a situation, we will now refer to this as choosing the best card instead of the card a human player would play. In order to predict the best card, we use some information from the previous step and create new attributes of a game state that aim to determine the goodness of a card. After the first step, we have a better understanding of the database and some attributes from that step can contribute to the process of predicting the goodness of the card. We now create additional attributes that determine the goodness of a card even further. Due to the imperfect information of the game of Klaverjas, it is not always clear what the best card is in each situation. This is also because different players could use different strategies. In some situations a human player may think that another card is better than another human player with a different strategy. After creating this agent, we test how many game states it predicts the card that a human player would play. Furthermore, we test the agent against two other agents with relatively simple strategies. We test the percentage of games the random forest agent

wins against the other two agents and give a distribution of the points obtained by the agents in each round.

1.1 Game of Klaverjas

In this thesis we will analyse the trick-taking card game of Klaverjas. The game of Klaverjas will now be explained in more detail.

1.1.1 Cards

Klaverjas is a trick-taking game played in teams of two people. The goal is to score more points than the other team. For this game, the top eight cards of a French deck are used. Each card has a face-value (7,8,9,10,J,Q,K,A), a suit (clubs, hearts, diamonds and spades) and a rank (1-8). Each card also has a value that is awarded when you win the trick this card has been played. The rank and value of a card depend on whether or not the suit of the card is a so-called trump suit. The cards of a trump suit follow a different rank and value distribution. A card from a trump suit always has a higher rank than a card from a normal suit. In this way, any trump card always wins from any regular suited card. The points and rank distribution are given in Table 1.

Rank	Regular		Trump	
8	A	11	J	20
7	10	10	9	14
6	K	4	A	11
5	Q	3	10	10
4	J	2	K	4
3	9	0	Q	3
2	8	0	8	0
1	7	0	7	0

Table 1: Rank and value for each card

1.1.2 Round

A game of Klaverjas consists of several rounds. In theory, this could be unlimited, but a common number of rounds is 16. In this thesis, we neglect the notion of rounds, because this has a confounding effect on the statistical results. At the beginning of each round of Klaverjas, all players receive eight random cards from the deck and a random trump suit is chosen. A round consists of eight tricks. In a trick, all players play one card. The player that played the card with the highest rank wins the trick and plays the first card of the next trick. The total value of all the cards in a trick is added to the score of the team that had the winning card. In Klaverjas we distinguish two main versions of the game: The Amsterdam - and the Rotterdam version. The rules that decide which cards can be played are almost the same but are different in one rule. The rules for the Rotterdam version are given below. If you can not follow a rule, the next rule needs to be applied

1. If there has not yet been played a card, all cards may be played.

2. The card must be of the same suit as the first played card.
3. If another trump card is already played, a higher trump card must be played.
4. A trump card must be played.
5. Each card can be played.

For the Amsterdam version the following rules apply:

1. If there has not yet been played a card, all cards may be played.
2. The card must be of the same suit as the first played card.
3. If the teammate is currently winning the trick, each card can be played.
4. If another trump card is already played, a higher trump card must be played.
5. A trump card must be played.
6. Each card can be played.

In this thesis both versions are used. A round is over when eight tricks are played, meaning all players have played their cards.

Figure 1: A typical game of Klaverjas.

1.1.3 Meld

There is an additional way to receive points. Meld points are given for certain combinations of cards in a trick. The winner of the trick receives these points. The following combinations are worth meld points:

- **10 meld points** — winning the last trick
- **20 meld points** — three consecutive cards of the same suit
- **50 meld points** — four consecutive cards of the same suit
- **100 meld points** — four cards of the same face value
- **200 meld points** — four cards of value Jack
- **additional 20 meld points** — King and Queen of trump suit
- **100 meld points** — all tricks are won by trump suit playing team

It is not always wise to play a card that allows for meld points. For example, if you are sure that the other team will win this trick. Playing cards that create meld points are not mandatory.

1.1.4 Bidding process

At the start of each round, a random trump suit is chosen. Now the bidding process takes place. The first player can choose whether or not he wants to play this trump suit. Playing a trump suit means you decided that this trump suit is good enough for you to win this round. If you play a trump suit and fail to score more than half of the total points scored by both teams in a round, you lose all points to the opposing team. If the first player decides they will not play this trump suit, the next player may decide if he will play it or not. If all players decide to not play the chosen trump suit, a new trump suit is chosen and the process begins again.

2 Related Work

Klaverjas is mostly played in the Netherlands and therefore there are not many studies done on this specific game. This thesis expands on previous research [vRTV19]. In this research, a method is presented in which a combination of machine learning and an exact approach is able to predict whether or not a combination of hands is winning. This is done using perfect information and therefore all hands were visible for the algorithm. We now try to predict this using imperfect information by only knowing the hand of the player.
Artificial intelligence has been used before in AlphaZero [SHS+17]. In this research, a reinforcement algorithm is used to master the game of Chess and Go. By simulating games against itself it was able to play these games increasingly better. This agent eventually beat the world champion of Chess and Go and the very high-level algorithm 'Stockfish'. Chess and Go are games of perfect information and therefore it would be interesting to see how well it performs on games with imperfect information.
A common way to solve trick-taking card games is with the use of the mini-max algorithm. This has been done for several card games such as Bridge [FB98]. Klaverjas is a game of imperfect information: a player only knows his own cards and not the cards of the other players. This causes that the game state space is very large. To still be able to determine the best card to play, Perfect Information Monte Carlo (PIMC) can be used [Gin01]. This technique uses a game tree that has a probabilistic node as starting node. This node branches to all possible game states. This technique

is a convenient use when the possible game state space is large.
In [Cha18] a method is presented to create an environment in which a reinforcement algorithm can be used on the imperfect information game 'Big2'. This method builds further upon [SHS+17] as it now creates an environment where not two but four players are active. The trained agent is able to beat amateur players in a relatively short training time.

3 Data

For this thesis, a database provided by P. Koning [Kon] is used. This database contains 2,465,352 game states. A game state is a representation of the game with the currently available information. This is different than a representation of the game itself. A game state represents the information known by the current player, such as its current hand and whether other players still have a card of a suit. A game representation is a more general way to represent the game. One can choose to represent the whole game with all hands known from each player or only use information known to all players, such as the trump suit and already played cards. In this research, we only want to use game states as we try to predict which card a human player would play in a situation. The data is created by analysing games on www.klaver.live. Klaver.live is a website where everyone can play an online game of Klaverjas. The database is therefore created by analysing good and relatively bad players. All the data attributes consist of strings. The database contains the following attributes:

- **Cards** — The cards the current player has.
- **Center** — The cards already played by other players this trick.
- **CardPlayedx** — The cards already played by player x in previous tricks.
- **HasColorx** — Represents whether player x has a suit. This is 0 if a player was not able to follow a suit. The order of suits: clubs, hearts, diamonds, spades.
- **Troef** — The trump suit in this round.
- **Variant** — The version of the game that is played. This could be Rotterdam or Amsterdam. If it was not possible to determine the version of the game from the played cards or people did not follow either of the rules this is set to unset.
- **PlayCard** — The card that is actually played by the player. This is our target for predicting which card to play see Section 4.
- **PlayableCardBits** — A string that corresponds to cards, which shows the cards that are allowed to be played. This is our target for predicting whether a card can be played see Section 4.

Both versions of Klaverjas can be played on the website. It is not obliged to enter the version of the game you play. This causes that we could not always determine the version of the game that was played. The dataset therefore contains games that have neither of the versions. These games are deleted. The database contains 797,260 different game states for the Rotterdam version and 1,255,644 for the Amsterdam version. We use the same dataset for both steps taken in this thesis.

In this research, we want to evaluate each card on its own. This allows us to determine whether or not we are allowed to play this card and if that is the case, if we want to play this card. To be able to do this, we need to split the database from hands to individual cards. The dataset also contains cards that have already been played. These cards are marked with a '.' and we can delete these cards from the database. Therefore, after we split the data into single cards we have 3,587,804 cards in the Rotterdam version and 5,652,348 cards in the Amsterdam version.

After analysing this database which is used for creating our agent that predicts which card we need to play, we need to create a set of game representations to test the quality of this agent. We will test the quality of the agent by performing simulations against other agents with different strategies. To compare these agents fairly, we need to test the same games in the simulations. In Klaverjas the chance of winning a round is highly correlated to the quality of cards that are dealt to all players. By using the same games we mitigate this factor as all agents will have the same starting configuration and we can compare the agents more fairly on performance. We create this database by randomly assigning cards to players. In a normal game of Klaverjas, the bidding process starts at the start of each round. As the scope of this thesis is not to create an agent that optimises the bidding process, we make a basic algorithm that determines if a player bids or not. We create a database with random games where a game is only allowed if the first player to play a card has at least 25 points of the trump suit.
In Klaverjas the dealt cards have a big impact on the odds of winning a round or not. Even the best strategy loses when he has very bad cards and plays against very good cards. Therefore we only create games where the first player has good cards. We create 5,000 games of Klaverjas where the cards, trump and starting player is given. In Klaverjas, the first player has more strategic options as he can choose the first card that will be played. Therefore being the first player to move has a strategic benefit. To mitigate this factor, we create a database in which the first player to play switches between the teams. For both teams, we create 2,500 games in which that team has the starting player. We therefore end up with 5,000 different games.

4 Methods

In this section, the methods used to create the agent that plays Klaverjas using a random forest are explained. The methods can be divided into two different parts, the first step is looking at the methods used to create an agent that successfully predicts which card can be played. The second step is explaining the methods to create the agent that predicts the best card in hand in a game state. Lastly, we will look at some measurements to get a deeper understanding of the performance of our agent.

4.1 Predict which card can be played

To predict if a card can be played, we need to transform the dataset into states that can be processed by a random forest. In Section 1 we saw that whether or not a card may be played depends on different rules. By using these rules, we can determine which attributes of the cards determine if a card can be played or not. The following attributes are needed:

- **The suit of the first played card** — To determine which suit we need to follow.
- **The other played suits** — To determine whether a trump card is played.
- **The rank of all cards** — To determine whether we have a higher trump card if needed.
- **Which player is currently winning the trick** — To determine whether a trump card needs to be played or not. This is only applies to the Amsterdam version.

To create these attributes out of the dataset, we first perform some preprocessing steps to the data. We split the cards into a list which allows us to evaluate them more easily. Furthermore, we split the already played cards in this trick into separate cards and we extract the number from the trump. We then replace the value of each card with a rank based on the suit of the card, this makes it easier to compare cards to each other. After the aforementioned step, we split the center cards into suit and rank.
After the preprocessing steps, we can create the needed attributes. Because we want to evaluate the cards one by one, we eventually need to split the cards. By doing this, we only have the information of the current card and we do not know anything about the other cards that were in the current hand, which causes problems. For example, a trump card can only be played if the player does not have the leading suit, but based on the aforementioned data we do not know if we have such a card. Therefore we need to create features that represent this kind of information.
The features that we create are the following:

- **Has_suit** — Is 1 if player has the leading suit in hand, 0 otherwise.
- **Has_trump** — Is 1 if player has a trump card, 0 otherwise.
- **Has_higher_trump** — Is 1 if player has a trump card that is higher than already played trump cards, 0 otherwise.
- **Is_higher** — Is 1 if the current trump card is higher than the already played trump cards, 0 otherwise.

For the Amsterdam version we create two extra features.

- **Is_winning** — Is 1 if teammate is winning the current trick, 0 otherwise.
- **other_suits** — Is 1 if number of suits in hand > 1, 0 otherwise.

After we created the features, we change the appearance of the player suit and the suits of the center cards. For the player suit, we change the suit according to the leading suit of the trick into three different features. If it is the same, it gets changed to *p_s*, if it is equal to the trump suit it changes to *p_t*, if this is not the case it is changed to *p_none*. If a card has already been played, the leading suit is changed to *1_t*. If this leading suit is equal to the trump suit and if this is not the case it is changed to *1_nt*. The second and third cards have the same technique that is applied to the player cards.
The rank of all cards is also taken into account as sometimes it is necessary to play a higher trump card. This gives four extra features. After creating these features we now have eleven features representing the suits of the cards, four representing the rank of the cards and six and four extra

features based on the current situation for the Amsterdam and Rotterdam version respectively. This leads to twenty-one features for the Amsterdam version and nineteen features for the Rotterdam version.
Once all features are created, we can train the random forest on this data. We split the data into a train and test set. After training the random forest on the train set, we evaluate the test set and compare the prediction for every card with the actual target for that card. In this way, we get a percentage of how many cards are predicted right. We expect to get 100% accuracy in predicting which card can be played as the rules for playing cards are relatively easy. The random forest should be able to distinguish patterns that determine whether a card can be played or not according to the applicable rules.

4.2 Predicting the best card

This task is not as straightforward as the previous step. Some general rules are needed to determine if a card is good in different situations of the game. The 'goodness' of the card depends on the context, one of the examples being the rank of the card, but it could also depend on different combinations of the center cards. For example, if three cards are already played and the player has two cards that are able to win this trick, the player sometimes needs to choose to play the second-best card in order to win another trick with the best card.
Just like the method described in Section 4.1 we need to evaluate each card individually to predict the goodness of the card. The best card will then be chosen and evaluated. A more extensive explanation of this process will be given later on in this thesis. If we split the database into cards instead of hands, we lose information about the other cards. So, as seen before in Section 4.1 we need to create features that transfer this information. We will therefore create features in three different groups: features created by evaluating the whole hand, features created by evaluating a single card and features created by evaluating both.

Before creating these features, we must first create class instances of all the cards in the database. In this way, we could more easily use these cards for the functions we use. After that, we start creating the features. We use the same features as used in Section 4.1.
We first create features by evaluating the whole hand.

- **tX** — Represents the current turn with x being the number of cards in the center increased by one.
- **PlayerX_has_trump** — Is 1 if player x has a trump card based on current information, 0 otherwise.
- **No_playable_cards** — Represents the number of legal moves for the current player.
- **Teammate_played** — Is 1 if player 2 has already played a card, 0 otherwise.
- **Teammate_winning** — Is 1 if player 2 is currently winning the trick, 0 otherwise.
- **Can_create_street** — Is 1 if the current player has a card that can create a street, 0 otherwise.

- **Has_higher** — Is 1 if current player has a card that is higher than all cards already played, 0 otherwise.

The features that are created by evaluating a single card:

- **Is_highest_suit** — Is 1 if this card is the highest card left of this suit, 0 otherwise.
- **Is_second_highest_suit** — Is 1 if this card is the second highest card left of this suit, 0 otherwise.
- **Is_lowest_value** — Is 1 if this card has the lowest value of all cards in hand, 0 otherwise
- **Is_highest_value** — Is 1 if this card has the highest value of all cards in hand, 0 otherwise.
- **Creates_street** — Is 1 if this card creates a street, 0 otherwise.

The last created features depend on the suit of the first played card. If the current player is the first player to play a card, these features are created after splitting the database into cards. The suit of each card will then be the suit determining the value of the features.

- **PlayerX_has_suit** — Is 1 if player x has the leading suit or the suit of the currently evaluated card, 0 otherwise.
- **Has_highest_suit** — Is 1 if the currently player has the highest card left of the leading suit or the suit of the current evaluated card, 0 otherwise.
- **Has_second_highest_suit** — Is 1 if the current player has the second highest card left of the leading suit or the suit of the currently evaluated card, 0 otherwise.
- **Still_in_game** — Is 1 if the highest card left of the leading suit or suit of the currently evaluated card is not in the center or in hand of the current player, 0 otherwise.

We now have created 42 features to put into our random forest. Unlike in Section 4.1, we now use soft-prediction. We predict the likelihood of a card being the best card and choose the card with the highest probability. This card is compared to the actual card played and this allows us to determine if our prediction is correct. As we created many features that influence the goodness of the card and the amount of data we used to train the random forest, we expect a high accuracy in predicting which card is played. However, this data also contains cards played by bad players.
To test the quality of our prediction, we check what the performance is for a rule-based agent. This rule-based agent follows some simple rules to determine which card is best to play in a given situation. Our random forest agent should be able to beat this rule-based agent in this task as it takes more features into account and is trained on a big data set. The rule-based bot is given in Figure 2

```
if first turn or second turn then
    if Has highest card of cards still in game of suit then
        Play highest card
    else
        Play card with lowest value
    end if
else if third turn then
    if Can't follow asked suit then
        Play lowest card
    else if Teammate played highest card left of suit then
        Play highest card
    else
        Play lowest card
    end if
else
    if Teammate is currently winning trick then
        Play highest card
    else if Has card that wins trick then
        Play this card
    else
        Play lowest card
    end if
end if
```

Figure 2: Pseudo code of the rule-based agent

4.3 Simulations

We now have a random forest agent that based on its predictions can play a game of Klaverjas. We know how good it is at predicting cards in the dataset, but we do not know how good the agent is at actually playing the game. We therefore need to compare this agent with other agents with different strategies in actually playing the game. We create two agents with simple tactics and simulate games against them.

- **Random agent** — This agent chooses a random card from all the legal cards. This could barely be called a real strategy as it just plays random cards. Therefore the random forest agent should be able to beat this agent.
- **Rule-based agent** — This agent plays cards based on some simple rules. The rules are not that advanced, so the random forest agent should be able to defeat this strategy. The pseudo-code of this rule-based agent is given in Figure 2.

We simulate 5,000 games in which the teams are one of the agents. In this way, we can evaluate whether the random forest agent actually outperforms the basic agents. A game consists of playing one round of Klaverjas to avoid confounding factors. The team with the most points after this round is the winner. We use a set with pre-determined games of Klaverjas. We first let one agent

be Player 1 and Player 3. After simulating the 5,000 games from the dataset, we let this agent be Player 0 and Player 2 for the same 5,000 games. In this way, we mitigate the factor that the database could be biased. The database could contain cards that always favor the first player, but by switching the first player we can mitigate this risk. After all simulations are done, we evaluate the number of won games by each agent. We also see the number of won games in which the agent was the starting player or not.
In total we simulate 5,000 games six times, in which each agent plays against another agent. We do this for both versions of the game, hence six times 5,000.
We expect that the random forest agent is able to beat the random agent, which has no logic-based strategy. The rule-based agent will likely be stronger, but due to the amount of data the random forest agent is trained on and the more attributes the random forest agent takes into account, it should still be able to beat the rule-based agent.
The last step is evaluating the points scored each round by the random forest agent and the rule-based agent. As Klaverjas is typically played in 16 rounds and the points carry over from round to round, it may be that even if an agent wins more rounds, it could end up losing. If an agent only barely wins his rounds and loses all the points if he loses a round, the agent could still lose. We therefore want to evaluate the number of points scored in each round. We choose to only evaluate the games played by the rule-based agents against the random forest agents as these are expected to have the best strategies.

5 Algorithms

In this section the used programming language and the random forests will be explained with more detail.

5.1 Programming libraries

For this thesis, we need to apply some preprocessing steps and apply a random forest. The programming language used for this thesis is Python version 3.10.5. For the data processing, we use the Pandas library version 1.3.5. This library has many built-in functions that allow for more easy data processing. For the random forest, we use the Scikit-learn library version 1.0.2 [PVG+11] [BLB+13]. SKLearn is a commonly used library for machine learning. We use the *RandomForestClassifier* with the default parameters. Furthermore, we use the *train_test_split* function from this library to split the dataset into a train and test set. We use 75% of the dataset to train the random forest and 25% to test the random forest. In the end, Joblib version 1.1.0 is used to store the random forest, this allows us to use it for the simulations.

5.2 random forest

In this thesis, we aim to build a classifier that determines whether a card is playable and which card should be played. This is a binary classification task. The dataset contains examples of cards that are playable and cards that are played. We therefore make this a supervised learning task. One of the most robust algorithms to use for supervised classification problems is a random forest. Random forests consists of multiple decision trees [Bre01]. These decision trees are created using

different samples of the dataset. The output of a random forest is the majority vote of all the decision trees. The decision trees are all created on random samples of the data. In this way, they all have some bias. By combining these decision trees, we are able to cancel out most of this bias by using the other decision trees. By doing this, we are able to get a more robust classification algorithm.
We use a random forest because attributes of a game state are easily transformed in either categorical data or binary data. A random forest is particularly good at handling these kinds of data and is able to process big amounts of data.

6 Results

We will now show the results of the performance of the two different agents that we created. We will now take a look at how often the agent correctly predicts whether a card may be played or not. After that, we will show how sufficient the other random forest agent is in predicting the best card out of a player's hand. We then show the results of the simulations against agents with different strategies: The random agent and the rule-based agent. At last, we will look at how many points the random forest and rule-based agents score on average in a round of Klaverjas.

6.1 Results predict which card can be played

For this step, the random forests for both versions are trained on 75% of the dataset and 25% of the dataset is used for testing. As the confusion matrices show, and as expected, the random forest

	Amsterdam			Rotterdam		
	Predicted playable	Predicted not playable	Total	Predicted playable	Predicted not playable	Total
Playable	856,351	0	856,351	536,044	0	536,044
Not playable	0	556,736	556,736	0	360,907	360,907
Total	856,351	556,736	1,413,087	536,044	360,907	896951

Table 2: Confusion matrix for the Amsterdam and Rotterdam version in predicting whether a card can be played or not.

is able to classify 100% of the cards right for both versions. This means we created a random forest agent that can predict which card is allowed to be played. We will now look at which features most contributed to the fact that is able to predict whether a card is playable or not. A graph showing the feature importances is given in Figure 3. By looking at the feature importances we can see that whether having the leading suit or not is a very important feature. It has an importance of 35.2% and 35.7% for the Amsterdam and Rotterdam version respectively. This makes sense because whether you have the suit has influence on which suit you can play. Furthermore, the player suit is very important. This has combined importance of 31.4% and 34.3% for the Amsterdam and Rotterdam version respectively. These features, just like the *Has_suit* feature, determine whether the suit can be played or not. Features that are not important are the features representing the third card. This could be easily explained, as the third card is not played that often. This means it will not be taken into account in many game states. For the Amsterdam version, we see that whether the teammate is winning and whether we have other suits in hand are also a little bit important. This could be explained by saying that these features represent the exceptions of the rules. These features are decisive when such an exception occurs.

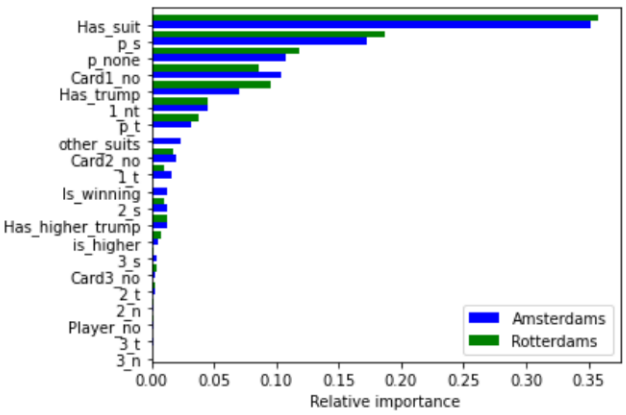

Figure 3: Feature importance for predicting whether a card is playable.

6.2 Results predict which card is best

We will first look at the performance of our rule-based agent at predicting the best card in a gamestate. The results for this agent is given in Table 3. In Figure 4 the percentages are visualised.

	Amsterdam				Rotterdam			
Number of playable cards	Predicted good	Predicted wrong	% predicted correctly	Total	Predicted good	Predicted wrong	% predicted correctly	Total
1	455,412	931	99.80%	456,343	294,267	73	99.98%	293,340
2	202,440	120,176	62.75%	322,616	125,648	80617	60.92%	206,265
3	81,715	75,045	52.13%	156,760	49,841	49,722	50.06%	99,563
4	40,533	43,407	48.29%	83,940	23,877	28,432	45.65%	52,309
5	30,293	33,835	47.24%	64,128	17,551	22,333	44.01%	39,884
6	27,202	29,385	48.07%	56,587	15,365	21,164	42.06%	36,529
7	27,540	37,710	42.21%	65,250	14,108	24,317	36.72%	38,425
8	14,443	35,577	28.87%	50,020	8,926	21,019	29.81%	29,945
Total	879,578	376,066	70.05%	1,255,644	549,583	247,677	68.93%	797,260

Table 3: Results for predicting the best card in hand for both versions for different number of legal moves for the rule-based agent.

As we can see the rule-based agent is able to predict the card that a human would play in 70.05% and 68.93% of the times for the Amsterdam and Rotterdam version respectively.
As we can see in Figure 4 we can see that the percentage of predicting the card a human would play drops as the number of playable cards increases. The results of this agent gives us a lower-bound which our random forest should be able to beat.
To get the results for our random forest agent, we first need to predict the probability that this card is the card a human player would play in a situation. Then for every hand, we choose the card with the highest probability. After that, we compare this card to the actual card played for that hand and we are able to determine whether it has been predicted correctly. We distinguish between the total playable cards in the hand, as it becomes more difficult to predict which card is played when more cards are available. The results for both versions are given in Table 4. Table 4 shows that for the Amsterdam version a total of 1,021,833 from the 1,255,644 hands are predicted correctly. This corresponds to an 81.4% accuracy score. For the Rotterdam version 643,403 of the 797,260 hands are predicted correctly. This corresponds to an 80.7% accuracy score. In Figure 5 the percentages for the number of playable cards are shown. In this figure, we see that when the

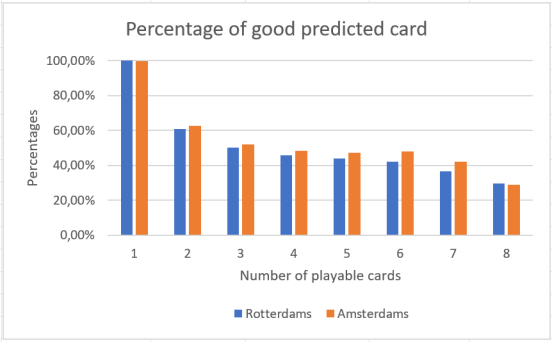

Figure 4: Percentage correctly predicted for the number of playable cards.

number of playable cards becomes higher, the prediction score becomes lower. This makes sense as it becomes more difficult to predict which card is best when there are more options to choose from.

	Amsterdam				Rotterdam			
Number of playable cards	Predicted good	Predicted wrong	% predicted correctly	Total	Predicted good	Predicted wrong	% predicted correctly	Total
1	455,412	931	99.80%	456,343	294,267	73	99.98%	294,340
2	273,198	49,418	84.68%	322,616	175,277	30,988	84.98%	206,265
3	119,115	37,645	75.99%	156,760	75,031	24,532	75.36%	99,563
4	54,716	29,224	65.18%	83,940	32,665	19,644	50.75%	39,884
5	35,687	28,441	55.65%	64,128	20,240	19,644	44.74%	36,529
6	27,910	28,677	49.32%	56,587	16,343	20,186	44.74%	36,529
7	28,050	37,200	42.99%	65,250	15,823	22,602	41.18%	38,425
8	27,745	22,275	55.47%	50,020	13,757	16,188	45.94%	29,945
Total	1,021,833	233,811	81.38%	1,255,644	643,403	153,857	80.70%	797,260

Table 4: Confusion matrix for predicting the best card in hand for both versions for different number of legal moves for the random forest agent.

It outperforms choosing random cards as for example 40% of the hands with seven available cards are predicted correctly, while randomly choosing a card would lead to an accuracy of 14%.
As we can see, our random forest agent is able to beat our rule-based agent. 81.4% and 80.7% accuracy for the Amsterdam and Rotterdam version respectively is significantly higher than the 70.05% and 68.93% from the rule-based agent. We created an agent that is better at predicting the best card in a given situation than a simple rule-based agent.
However, our agent still has difficulty predicting the card a human player would play in certain situation. Especially in the cases in which many cards are evaluated the accuracy score is considered quite bad. Having eight playable cards normally means that you are the first player of a round to play a card. This is a very important game state and can determine whether you win or lose that round. We still are only able to predict 55.47% and 45.94% of the cards right with eight playable cards for respectively the Amsterdam and Rotterdam version. A good agent for Klaverjas should be able to make the right decisions in difficult situations. This is not what our agent is doing consistently.
Figure 6 shows the feature importances for both versions. Here we see that for both versions, Rotterdam and Amsterdam, the number of playable cards is a very important feature. That could be explained by the fact that there are many hands in the dataset with just one playable card. If the number of playable cards is equal to one, this card should always be played. Furthermore,

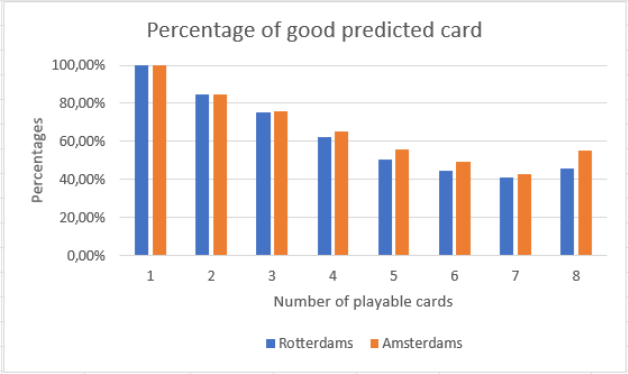

Figure 5: Percentage correctly predicted for the number of playable cards.

whether a card has the lowest or highest value is important. When you are playing the third or fourth card in a trick this could be important, because then it is known whether your teammate is winning or not. This could determine whether you want to play a card with a high or low value. Furthermore, the rank of a card is important. This makes sense because for the card itself it is a good measure of the value of the card and for the cards in the center it gives a good overview of the cards that need to be beaten. Just like with the prediction in Section 6.1 the suit of the second and third card are not that important. We do not see big differences between the two versions.

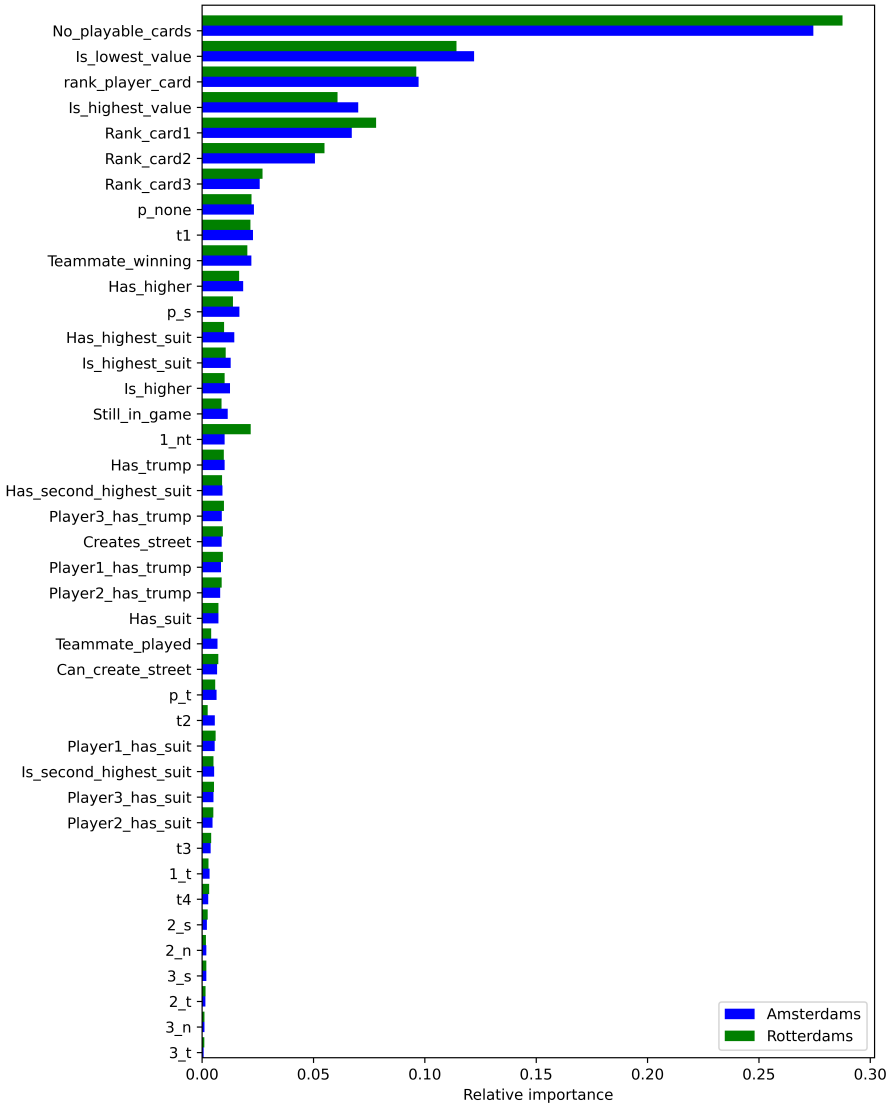

Figure 6: Feature importances for both versions.

6.3 Simulations

We divide this section into three subsections in which first the rule-based agent plays against the random agent, and after that the random agent plays against the random forest agent. Lastly, the random forest agent plays against the rule-based agent.

6.3.1 Random agent vs rule-based agent

The total games won from the simulations for the random agent versus the rule-based agent are shown in Table 5. As we can see in Table 5 the rule-based agent wins 56.43% of the games from the Amsterdam version and 58.15% of the games from the Rotterdam version. Despite being small, there is a difference in performance between the versions. It is surprising that the rule-based agent

	Amsterdam					
	Random agent			Rule-based agent		
	Won while starting	Won while not starting	% won	Won while starting	Won while not starting	% won
First 5,000 games	1,054	1,039	41.86%	1,461	1,446	58.14%
Second 5,000 games	1,102	1,162	45.28%	1,338	1,398	54.72%
Total	43.12%	44.02%	43.57%	55.98%	56.88%	56.43%

	Rotterdam					
	Random agent			Rule-based agent		
	Won while starting	Won while not starting	% won	Won while starting	Won while not starting	% won
First 5,000 games	954	1,055	40.18%	1,445	1,546	59.82%
Second 5,000 games	1,093	1,083	43.52%	1,417	1,407	56.48%
Total % won	40.94%	42.76%	41.85%	57.24%	59.06%	58.15%

Table 5: Games won by the random agent and the rule-based agent for both versions.

wins fewer games in the Amsterdam version than in the Rotterdam version. In the Amsterdam version, there are situations in which there are more legal moves. In general, if there are more legal moves, the random agent is less likely to pick the best card making the agent's strategy worse, resulting in a lower win rate for the random agent. However, our results show the opposite. This can be explained by the fact that the rule-based agent does not account for special trump cases. It

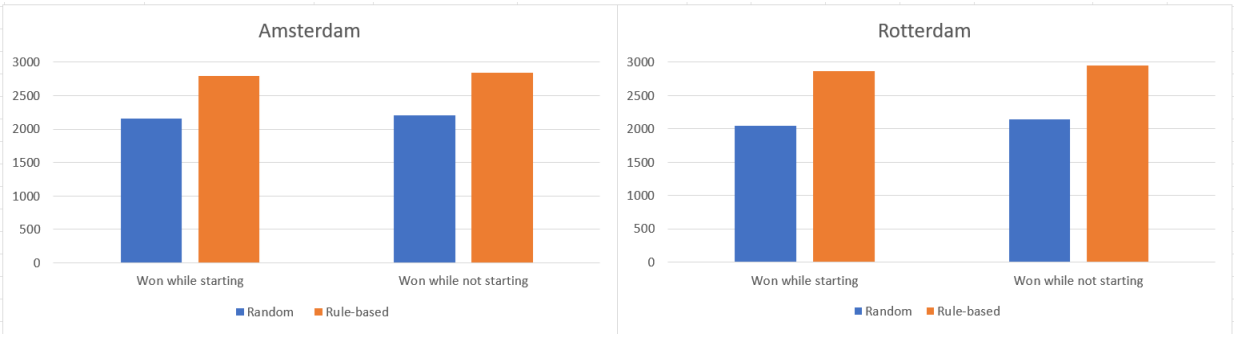

Figure 7: Graph of total games won for each version

could be that because of this, the rule-based agent becomes 'more bad' than the random agent does, resulting in a lower win rate for the rule-based agent. Another thing that stands out analysing the win rates is that the second run of the database produces significantly different performance results. The win-rate for the Amsterdam version differs from 58.1% to 54.7% and for the Rotterdam version, it differs from 59.8% to 56.5%. This means that the database contains biased hands. The database contains hands that favored the rule-based agent more in the first run than in it did the second run. In Figure 7 a graph is shown for the games won for each version by each agent and whether it was starting or not. This is a visual graph of Table 5.
This figure shows that there is only a minimal difference between starting and not starting the round. The rule-based agent wins in total 56.6% of the hands where it is the starting player and 57.95% where it is not the starting player. This is surprising as the first player has a confirmed good hand. It has at least two good cards of the trump suit. It makes sense that this does not matter for the random agent, but the rule-based agent should be expected to make more use of this factor. Furthermore, we see that the rule-based agent is able to beat the random agent.

6.3.2 Random agent vs random forest agents

We will now look at the results of the simulations of the random agent against the random forest agent. The rule-based agent gave us a lower-bound win-rate of 56.4% and 58.2% so we should be able to win more than that. The total number of games won by each agent is shown in Table 6.

	Amsterdam					
	Random agent			Random forest agent		
	Won while starting	Won while not starting	% won	Won while starting	Won while not starting	% won
First 5000 games	1,010	1,030	40,80%	1,470	1,490	59,20%
Second 5000 games	1,092	1,020	42,24%	1,480	1,408	57,76%
Total	42,04%	41,00%	41,52%	59,00%	57,96%	58,48%

	Rotterdam					
	Random agent			Random forest agent		
	Won while starting	Won while not starting	% won	Won while starting	Won while not starting	% won
First 5000 games	951	1,062	40,26%	1,438	1,549	59,74%
Second 5000 games	1,016	1,060	41,52%	1,440	1,484	58,48%
Total	39,34%	42,44%	40,89%	57,56%	60,66%	59,11%

Table 6: Games won by the random agent and the random forest agent for both versions.

The random forest agent is able to win 58.5% of the games for the Amsterdam version, for the Rotterdam version this is 59.1%.
Just like with the simulations in Section 6.3.1, we see that the second run of the 2,500 games is worse for the random forest agent. The accuracy for both versions differs from 59.2% to 57.7% and from 59.7% to 58.5%. In Figure 8 the number of games is shown for each version and whether it was starting or not.
Just like we have seen before, there is not a huge performance difference between starting or not. In Figure 5 is shown that the random forest has difficulty predicting the best card when many cards are playable. Whilst being a following player with fewer available cards to play, he becomes better at predicting which card is best.

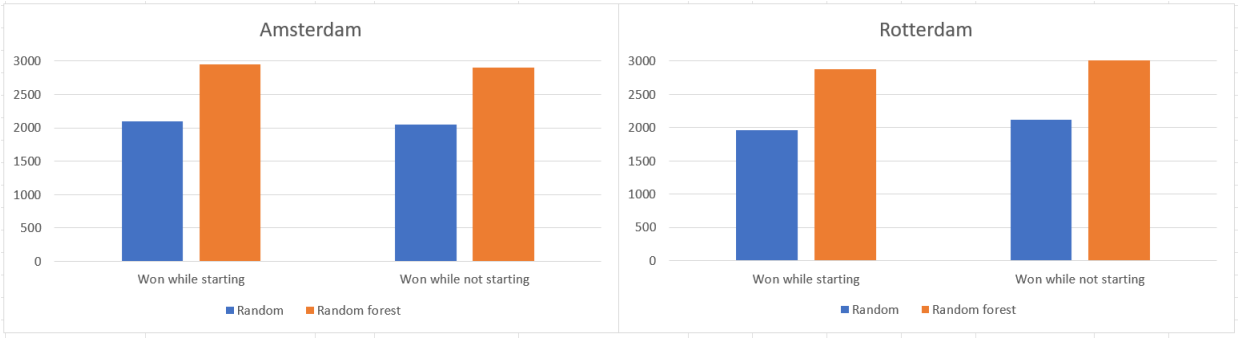

Figure 8: Graph of total games won for each version

Therefore it could be that the fact that starting the game with a good hand and therefore having a strategic benefit is overruled by the fact that the random forest agent is not able to predict the best card with many cards to play. A good agent should still be able to win more times when it starts as losing when starting is very bad. All points are given to the opponent and therefore losing while playing the trump suit should be minimized and that is not what our agent is doing. It is clear that

the random forest agent outperforms the random agent. However, the difference in performance between the rule-based agent and the random forest agent against the random agent differs less than expected. There is a difference so we do expect that the random forest agent is able to beat the rule-based agent.

6.3.3 Rule-based agent versus random forest agent

We will now look at the results for our most advanced strategies: Rule-based and random forest. In Table 7 are the number of games won by each agent shown. As we can see the random forest

	Amsterdam					
	Rule-based agent			Random forest agent		
	Won while starting	Won while not starting	% won	Won while starting	Won while not starting	% won
First 5000 games	1,071	1,063	42.68%	1,437	1,429	57.32%
Second 5000 games	1,260	1,195	49.10%	1,305	1,240	50.90%
Total	46.67%	45.16%	45.89%	54.18%	51.38%	54.11%

	Rotterdam					
	Rule-based agent			Random forest agent		
	Won while starting	Won while not starting	% won	Won while starting	Won while not starting	% won
First 5000 games	1,046	1,090	42.72%	1,410	1,454	57.28%
Second 5000 games	1,277	1,201	49.56%	1,299	1,223	50.44%
Total	46.46%	45.82%	46.14%	54.18%	53.54%	53.86%

Table 7: Games won by the rule-based agent and the random forest agent for both versions.

agent outperforms the rule-based agent by winning 54.1% of the games for the Amsterdam version and by winning 53.9% of the games for the Rotterdam version. Again we see huge differences in the first run of 5,000 games and the second run. In Figure 9 the total games won for each version and whether it was the starting player or not are shown. Once again, we see that whether starting

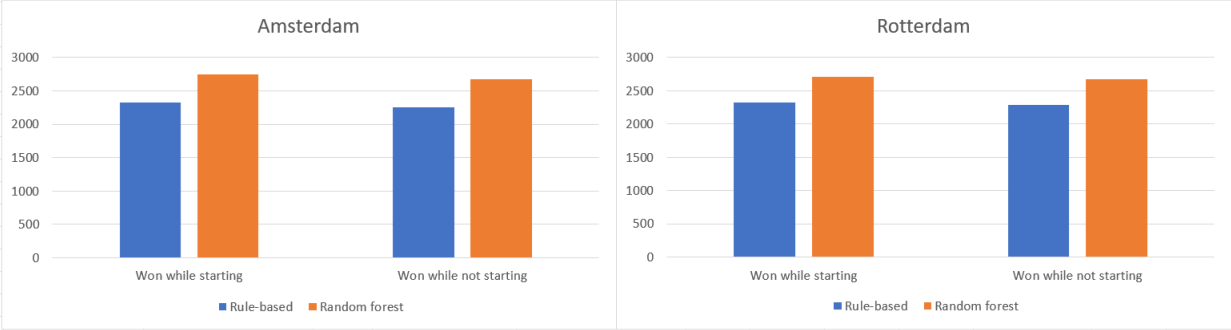

Figure 9: Graph of total games won for each version

the game or not has not that much influence on winning the game or not. For the Amsterdam version, the random forest agent wins 54.8% of the games while starting and 53.4% when it does not start. For the Rotterdam version, this is 54.2% and 53.5%. By analyzing the figures we see that the random forest agent outperforms the rule-based agent. However, the difference is small. The rule-based agent is very simple and the random forest agent is trained on a big dataset and takes more attributes into account. The random forest agent should beat the rule-based agent more than it has done now. It is expected that after improving the rule-based algorithm it would even be able to beat the random forest agent.

6.3.4 Overall

We will now take a look at a comparison between all the agents. In Table 8 a table is shown with the performances of each agent against the other agents.

	Random	Rule-based	Random forest
Random	—	42,71%	41,20%
Rule-based	57,29%	—	46,01%
Random forest	58,80%	53,99%	—

Table 8: Performance of each agent against the other agents.

In the table, the percentages represent the win-rate of the agent on the left against the agent at the top. In this table, we see that the random forest agents outperforms both other agents. However, the win-rates are too small. As expected the random agent has the worst strategy.

6.3.5 Points

Figure 10 shows how often the rule-based and random forest agent obtain a number of points in the 10,000 simulated games against each other. The outliers 0, 162, 182 and 202 are not shown completely as the other data points would not be visible anymore. In both versions, the trend line

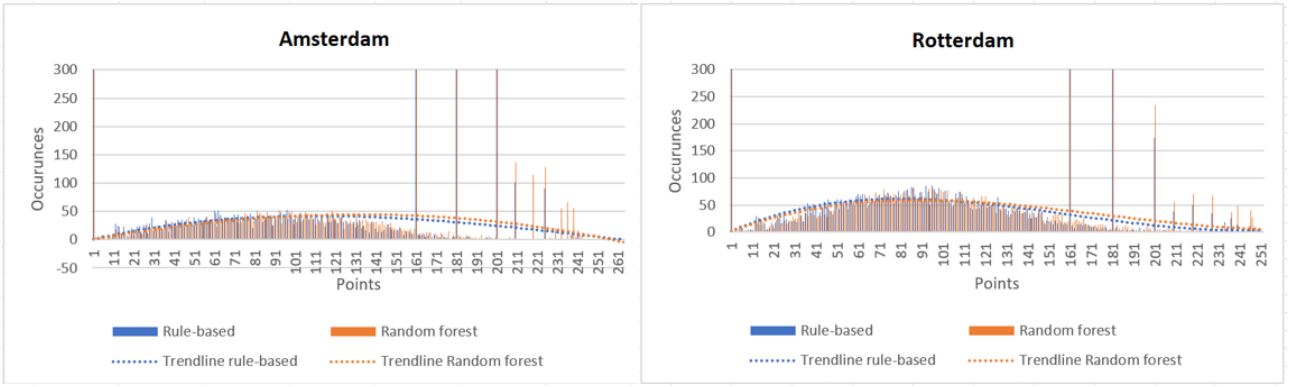

Figure 10: Total points obtained by the rule-based and random forest agent

for both strategies is a bit different. We see that the trend line of the random forest agent is more skewed to the right, but only slightly. Table 9 shows the average and standard deviations for both strategies for both versions. Here we see that for both versions the random forest agent gets more

	Amsterdam		Rotterdam	
	Rule-based	Random forest	Rule-based	Random forest
Average	87,91	100,1	87,46	101,95
Standard deviation	75,17	77,89	60,25	63,50

Table 9: Average points obtained and standard deviation by each agent in a round of Klaverjas.

points on average. 100.1 for the Amsterdam version and 101.95 for the Rotterdam version against

87.91 and 87.46 for the rule-based agent. Furthermore, we see that for the Amsterdam version the density of points is more spread out due to the higher standard deviation. We see that the random forest agent not only wins more games but also scores more points on average than the rule-based agent. This is a significant difference in performance between the rule-based agent and the random forest agent.

7 Limitations

We will now discuss some limitations of this research.
At first, the database is shared by the maintainer of the website www.klaver.live. This is a free-to-play Klaverjas website where everyone is always able to play a game of Klaverjas. This means that a top-level player could be among the data entries, but also mediocre players of Klaverjas. It is hard to distinguish these played hands from the other hands that might be played by the top-level player. This means our random forest is trained on hands where the best card is chosen, but also on hands where this is not the case. This makes our random forest less reliable. In the future, a subset of this dataset could be chosen. By filtering out cards that are obviously not the best card a new dataset could be created. Determining whether a card is obviously not the best card could be done with some basic rules, such as playing a mid-value card of a suit from a hand with many cards of that suit. In some cases, this is a bad card.

An important strategic part of Klaverjas is the bidding process. This could give strategic advantages as the trump suit is an important factor in the game. In this research, we hardly took this into account, only by allowing games where the starting player had enough points of the given trump suit. As this still could mean that in the real world a player still would not have played the trump suit, the outcome of the games becomes less reliable.

Klaverjas always has a luck component. It does matter whether you get very good or very bad cards. Even with the best strategy you still can not win with very bad cards against very good cards. This is shown in Table 5 where the random agent still wins around 41% of the games. The random agent has generally been considered a very bad strategy. Because of this luck element, it is difficult to predict the actual performance of our agent.
There are some cases in which multiple cards could be considered the best card. For example, if you are the first player to play a card and you have two aces from a regular suit, both cards could be considered the best card. Our agent does not take this into account as is always chooses the same card as the best card. This decreases the accuracy of the random forest. However, for the game simulations, this should not matter, because in that case, both cards have equal winning chances.

8 Conclusion

In this thesis, we created an agent that is able to predict which card a human player would play in a game of Klaverjas using random forests. We created this agent by first creating an agent that is able to predict whether a card is playable or not. A card is playable when the rules of Klaverjas allow the card to be played. We trained a random forest with supervised learning to reach this task. This agent reached a 100% accuracy score. We then used this approach to create another agent

that could predict which card is the best card to play in a given situation. We created 42 features to make sure this could happen. For the Amsterdam version of this game, we were able to predict 81.4% of the hands correctly and for the Rotterdam version this was 80.7%. This agent is able to beat a simple rule-based agent for this task as that agent only reached 70.05% and 68.93% accuracy for the Amsterdam and Rotterdam version respectively. However, the agent has a lot of trouble determining which card to play if there are many cards playable. Hence, it is mediocre at making good decisions in complicated situations. We then used this random forest agent to simulate games against two different agents: an agent with a random strategy and an agent with a rule-based strategy. The random forest agent was able to beat the strategies with respectively 58.8% and 54.0%. Furthermore, we saw that on average the random forest obtains 100.1 and 101.95 points against 87,9 and 84,5 points for the rule-based agent for the Amsterdam and Rotterdam version respectively.

The win-rate of the random forest agent against the very simple rule-based agent is lower than expected. By improving the rule-based agent it is expected that it can beat the random forest agent. A good Klaverjas algorithm should be able to win more games against a simple rule-based algorithm and therefore the used method is not the way to go. This is surprising as random forest should be performing very well on these kinds of tasks.

Improvements to this agent could be made in filtering out bad entries in the database. As the database consists of games by players with all skill-levels, the database can consist game states where players have made a very bad decision. By filtering out these entries the random forest is trained on better data. Furthermore, other methods could be used like neural networks. Neural networks may be better at solving this task as they may be better at dealing with imperfect information.

We showed how to create an agent that predicts whether a card is playable or not and how well it performs on this task. We showed that a game of imperfect information could become solvable by using a random forest. The random forest is able to beat the rule-based agent on winning more games and scoring more points each game, but we expected to see a larger difference in performance between the two agents.

9 Further research

As discussed in Section 7, this research obliges players to play with a trump suit if they had enough points of this suit. The process of bidding whether you want to play a trump suit could be further analysed to make an even better agent with an even better strategy.

Furthermore, AlphaZero [SHS+17] could be applied to create an agent that learns the game of Klaverjas by itself. In this research, we created a representation of the game that could help with this process. As Alphazero can learn the game by only knowing the rules of the game, this is not particularly necessary, but it could help the learning process. This representation can be created without using the used random forest. The representation contains more information than a simple representation of the game does. In this way, AlphaZero should learn the game much easier.

Another option to improve this research could be by using neural networks instead of a ran-

dom forest. Neural networks could also be used by solving these kinds of problems, but configuring a neural network takes a lot of time. In [KLHG21] a method is showed which creates a neural network for tabular data. This should perform very well on our data as we also use tabular data. By applying this method, we should see better performances for the agent.

At last, this approach could be used in similar card games like Bridge. A strategy could be developed for these games in which the best card is chosen and this can be used to create an agent which people can play against.

References

[BLB+13] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[Cha18] Henry Charlesworth. Application of self-play reinforcement learning to a four-player game of imperfect information. *CoRR*, abs/1808.10442, 2018.

[FB98] Ian Frank and David Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1-2):87–123, 1998.

[Gin01] Matthew L Ginsberg. Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14:303–358, 2001.

[KLHG21] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 23928–23941. Curran Associates, Inc., 2021.

[Kon] Peter Koning. Live amp; online klaverjassen met vrienden.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SHS+17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[vRTV19] Jan N. van Rijn, Frank W. Takes, and Jonathan K. Vis. Computing and predicting winning hands in the trick-taking game of klaverjas. In Martin Atzmueller and Wouter Duivesteijn, editors, *Artificial Intelligence*, pages 106–120, Cham, 2019. Springer International Publishing.

10 Appendix

Link to GitHub repository: https://github.com/lennardhordijk/Klaverjas