Computing and Predicting Winning Hands in the Trick-Taking Game of Klaverjas

J.N. van Rijn[2,4], F.W. Takes[3,4], and J.K. Vis[1,4]

[1] Leiden University Medical Center
[2] Columbia University, New York
[3] University of Amsterdam
[4] Leiden University

Abstract. This paper deals with the trick-taking game of Klaverjas, in which two teams of two players aim to gather as many high valued cards for their team as possible. We propose an efficient encoding to enumerate possible configurations of the game, such that subsequently $\alpha\beta$-search can be employed to effectively determine whether a given hand of cards is winning. To avoid having to apply the exact approach to all possible game configurations, we introduce a partitioning of hands into 981,541 equivalence classes. In addition, we devise a machine learning approach that, based on a combination of simple features is able to predict with high accuracy whether a hand is winning. This approach essentially mimics humans, who typically decide whether or not to play a dealt hand based on various simple counts of high ranking cards in their hand. By comparing the results of the exact algorithm and the machine learning approach we are able to characterize precisely which instances are difficult to solve for an algorithm, but easy to decide for a human. Results on almost one million game instances show that the exact approach typically solves a game within minutes, whereas a relatively small number of instances require up to several days, traversing a space of several billion game states. Interestingly, it is precisely those instances that are always correctly classified by the machine learning approach. This suggests that a hybrid approach combining both machine learning and exact search may be the solution to a perfect real-time artificial Klaverjas agent.

Keywords: trick-taking card games, alpha-beta search, computational complexity, machine learning, AI

1 Introduction

A substantial part of artificial intelligence deals with investigating the extent to which machines are able to perform nontrivial complex human tasks. One of such tasks is playing games, a topic which over the years has received a lot of attention in artificial intelligence research [8,9,10], leading to a number of breakthroughs. A recent example is AlphaGo [21], where a combination of search algorithms and machine learning techniques is used to effectively beat humans at the highly complex game of Go. In general, a key problem in such games is that the search space of all possible game configurations is extremely large. This makes it difficult for algorithms to choose for example the next best move, whereas such a decision is often without much effort successfully taken by a human. In this paper we aim to explore this difference in competence of machines and humans, in particular in automatically assessing if a given instance of the game of Klaverjas can be won.

Klaverjas is a trick-taking (card) game, played with the top eight cards from each suit of the French deck. Each card has a face $f \in F = \{7, 8, 9, 10, \mathrm{J}, \mathrm{Q}, \mathrm{K}, \mathrm{A}\}$, a suit $s \in S = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$ and a rank $r \in R = (1, \ldots, 8)$. Cards with higher ranks are considered more powerful. One suit is designated the *trump* suit; cards from this suit are considered more powerful than all cards from other suits. Each card has a specific number of points associated to it, based on the rank and whether it is part of the trump suit or not. Table 1 displays for each card its value in points. Note that the rank of a card depends on the face value and whether it is part of the trump suit.

Table 1. Rank and number of points per card.

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

The game is played with four players that form two teams: one team consists of player N (north) and S (south) whereas the other consists of player E (east) and W (west). Each player starts with eight cards, to be played in each of the eight tricks. A trick is a part of the game in which each player plays one card. The first card of a trick can be freely chosen by the starting player. The other players must follow the leading card. If such a card is not available in a player's hand, a trump card must be played. Whenever a player must play a trump card, and a trump card has already been played in that trick, if possible, a higher rank trump card should be played. If the player can neither follow suit nor play a trump card, it is allowed to play any card that the player has left. It should be noted that there are also versions of the game in which always playing a (higher) trump card is not mandatory if a team mate has already played a trump card, referred to as "Amsterdams" rather than the version which we consider, which is "Rotterdams" Klaverjas.

Once the fourth player has played his card, the trick has ended, and the winner of the trick is determined. If trump cards have been played, the trump card with the highest rank wins the tricks. If not, the card with the highest rank of the leading suit wins the trick. The player who played this card takes all the cards, his team receives the associated points and will start the next trick. The team that wins the last of the eight tricks is awarded 10 additional points. To win the game the team that started the game has to accumulate more points than the opposing team. If they fail to do so, i.e., they lose the game, which is referred to as *nat*, 162 points are awarded to the opposing team. Note that draws do not exist. When the starting team manages to win all eight tricks of the game they are awarded 100 bonus points, referred to as *pit*.

Additionally, special *meld* points can be claimed by the team winning the trick when cards with adjacent face values are in the trick. These are for three ascending face values 20 meld points and for four ascending face values 50 meld points. For the King and Queen of trump, players can claim 20 meld points, in addition to other meld points already claimed in that trick. Finally, when four cards of the same face are played in the same trick, the team winning the trick can claim 100 meld points. For determining meld points, the order in which the players have played these cards is irrelevant. The addition of meld changes the dynamics of the game drastically, as players might sometimes be inclined to play a good card in a trick that is already lost, to prevent conceding meld points to the opposing team. Teams can choose not to claim meld points, for example when they already know they will lose the game.

In this paper we consider the task of determining and predicting whether an instance of the game of Klaverjas is winning for a variant of the game, in which complete information on a team's cards is available. In addition, we assume that there is no bidding process of determining which player starts the game; the first player always starts and determines the trump suit. For this simplified version of the game, we consider the decision problem of, given a particular distribution of cards over players, determining whether this hand is winning for the starting player. We do so using both an exact algorithm based on $\alpha\beta$-search, as well as using a machine learning algorithm that based on feature construction mimics how a human decides whether a hand would be winning, for example based on counting high value cards.

The results presented in this paper are useful for at least two types of new insights. First, in the real game, determining the starting player is done based on bidding, where players assess the quality of their dealt hand, based on whether they think they can win that hand. The approaches presented in this paper essentially perform this type of hand quality assessment. Second, as we employ both an exact approach and a machine learning approach, we can investigate the extent to which both are able to efficiently solve the game of Klaverjas. This will allow to investigate whether exact algorithms have the same difficulties with certain hands as an exact algorithm faces.

The remainder of this paper is organized as follows. After discussing related work in Section 2, we introduce various definitions and necessary notation in Section 3. Then, an exact algorithm for solving a game of Klaverjas is presented in Section 4. Next, a machine learning approach is presented in Section 5. A comparison between the two is made in Section 6. Finally, Section 7 concludes the paper and provides suggestions for future work.

2 Related work

Klaverjas is an example of the Jack-Nine card games, which are characterized as *trick-taking* games where the the Jack and nine of the trump suit are the highest-ranking trumps, and the tens and aces of other suits are the most valuable cards of these suits [16]. Trick-taking games are games of finite length, where players have a hand of cards, and in each round (called a trick) all players play a card from their hand; the player that played the best card according to the rules wins the trick. As Jack-Nine games are not extensively studied in literature, we review some seminal research on solving games, as well as as relevant literature on trick-taking games in general.

Exhaustive search strategies have extensively been applied in a number of games [9,19]. For two-player games, the minimax algorithm and its extension $\alpha\beta$-pruning, together henceforth referred to as $\alpha\beta$-search, traverse a game tree to evaluate a given game position. In practice, this is often combined with a *heuristic* evaluation function, in case the game tree is too massive to traverse to all relevant leafs. Historically, much research has been conducted to handcraft *static* heuristic functions that capture human knowledge about the game. Alternative to using a static heuristic function, induction techniques can be used to learn such functions based on previous games. Recently, this strategy has been successfully employed within AlphaGo, an artificial intelligence agent that has beaten the human world champion at Go [21,22]. When no heuristic function is used, and the relevant part of the game tree is completely traversed, the minimax algorithm results in the game-theoretical value, i.e., the outcome of the game assuming perfect play, of this game state [12,17]. This is one of the principal aims of this work.

When agents are confronted with imperfect information, Perfect Information Monte Carlo (PIMC) search is a practical technique for playing games that are too large to be optimally solved [7,15]. PIMC builds a game tree starting with a probabilistic node, branching to all possible configurations. For each configuration, it assumes perfect information and uses common search algorithms, such as minimax with $\alpha\beta$-pruning. It has been noted that the correct card to play might be based on information that the player can not possibly know [6].

Several computational complexity results have been obtained for trick-taking games where all players have perfect information [2,23,24]. According to [2], the natural decision question for trick-taking games is whether a given team can obtain a given number of tricks. In order to obtain computational complexity results, generalizations need to be made. In the case of trick-taking games this can be done over the number of suits, the number of cards per suit, the number of players and the way these players are assigned to two teams. In [23] it was shown that trick-taking games with two players and one suit are in P. In [24], it was proven that trick-taking games with two players, where both players have for each suit an equal number of cards, are in P, for an unbounded number of suits.

Later, the authors of [2] showed that all generalizations are in PSPACE, and when generalizing over the number of teams and the number of suits, trick-taking games are PSPACE-complete. Furthermore, they showed that a game with six players and unbounded number of suits and cards per suit is PSPACE-complete. It is shown that the obtained complexity results also apply to trick-taking games that feature a trump suit, such as the Nine-Jack games. The biggest difference between the obtained complexity results and the main interest of our work, Klaverjas, is that the natural decision question of Klaverjas involves a threshold on the number of points, rather than the number of obtained tricks. This decision question requires a different approach.

Other related research regarding trick-taking games focuses on combining and applying search and heuristic evaluation techniques to Skat, a related trick-taking game. We review some examples. In [13], an agent is proposed that uses exact search, assuming perfect information. The implementation features a specific move ordering, transposition tables and adversarial heuristics to speed up the procedure. The main claim is that a typical instance of Skat (with perfect information) can be solved within milliseconds. This conclusion is in line with our (Klaverjas) experiments in Section 6.2. The outcomes of individual games can be combined using Monte Carlo techniques. In [5] an agent is proposed that applies inference techniques to both heuristic evaluations and the bidding in Skat. Finally, in [14] a post processing opponent modelling technique is introduced, determining the skill level of an opponent based on earlier games. This is based on the premise that when playing against weaker opponents, moves that have a higher risk and reward pay-off can be played. All mentioned agents use a variant of PIMC to traverse the search space.

3 Preliminaries

We allow ourselves to build upon the excellent definition given by the authors of [2]. The game is played with 32 cards, each player obtaining 8 cards. For each card c, $\mathit{face}(c)$ denotes the face, $\mathit{suit}(c)$ denotes the suit, $\mathit{rank}(c)$ denotes the rank, and $\mathit{points}(c)$ denotes the number of points associated with this card. Note that each card is defined by its suit and face value. The rank and the score follow from this.

A position p is defined by a tuple of hands $h = (h_N, h_E, h_S, h_W)$, where a hand is a set of cards, a trump suit $\phi \in \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$ and a *lead player* $\tau \in P$, where $P = \{N, E, S, W\}$. Let h_p with $p \in P$ be the set of all cards (i.e., the hand) of player p. All players have an equal number of cards, i.e., for all $i, j \in P$ it holds that $|h_i| = |h_j|$. Furthermore, the hands of players do not overlap, i.e., for all $i, j \in P$ (with $i \neq j$) we have $h_i \cap h_j = \emptyset$.

We define $h_{p,s}$ with $p \in P$ be the set of all cards of player p that are of suit s, i.e., $H_{p,s} = \{c \in H_p : \mathit{suit}(c) = s\}$. Let o_p with $p \in P$ be the set of all cards in the opposing team of player p, i.e., $o_N = o_S = h_W \cup h_E$ and $o_W = o_E = h_N \cup h_S$.

Problem statement. The goal of this paper is to compute the game-theoretical value of the game Klaverjas $\textsc{KlaverjasOpen}(h, \phi, \tau)$ with hands

h, trump suit ϕ and starting player τ, where the players have full information. Team NS aims to maximize the the score of team EW subtracted from the score of team NS. Team EW aims to minimize this value. In case of a positive value, team NS has an optimal strategy to win the game, whereas in case of a negative value, team EW has an optimal strategy to win the game.

$\textsc{KlaverjasOpen}(h, \phi, \tau)$ returns tuple $K = (K_{NS,t}, K_{NS,m}, K_{EW,t}, K_{EW,m})$ of four values, respectively the trick points obtained by team NS, the meld points obtained by team NS, the trick points obtained by team EW and the meld points obtained by team EW. As such, the total score of team NS can be defined as $K_{NS} = K_{NS,t} + K_{NS,m}$, and similarly the score of team EW can be defined as $K_{EW} = K_{EW,t} + K_{EW,m}$. Note that there are sometimes multiple sequences of moves may lead to the same outcome. Including the definition of *nat*, the result of the game is determined by $\textsc{Outcome}(\textsc{KlaverjasOpen}(h, \phi, \tau)) =$

$$\textsc{Outcome}(K) = \begin{cases} K_{NS} - K_{EW} & \text{if } K_{NS} > K_{EW}, \\ -(162 + K_{EW,m}) & \text{otherwise.} \end{cases}$$

This value is to be maximized by team NS and minimized by team EW. Note that team NS needs to obtain more points that team EW cf. the definition in Section 1, otherwise all 162 points are awarded to team EW (i.e., team NW is *nat*). In this case, the optimal outcome does never include any meld points from team NS, as the team obtaining meld points can choose to not declare these. Also note that the definition of *pit* from Section 1 is implicitly captured in this formalization through the meld points.

4 Exact approach

After briefly discussing the combinatorics behind solving the game of Klaverjas in Section 4.1, after which the main approach is outlined in Section 4.2 and explored further in Section 4.3.

4.1 Combinatorics

In this section we restrict ourselves, without losing generality, to configurations of the game of Klaverjas with a fixed trump ($\diamondsuit$) and starting player (N). The number of different configurations is given as the number of ways of dealing 32 cards over 4 hands of 8 cards: $\binom{32}{8}\binom{24}{8}\binom{16}{8}$. Note that given a fixed trump suit some of the above configurations will be equivalent as the order of the non-trump suits is of no consequence. We omit the removal of these symmetrical configurations for the sake of simplicity.

We use a *Combinatorial Number System* [1,11] of degree k to define a bijection between any number N to the k-th combination (in lexicographic order) of $\binom{n}{k}$:

$$N = \binom{c_k}{k} + \cdots + \binom{c_2}{2} + \binom{c_1}{1}.$$

The process of mapping the number N to its corresponding combination is commonly referred to as *unranking*, while the inverse operation is called *ranking*.

It is trivial to combine combinatorial number systems (of the same degree). This allows us to easily enumerate the total number of configurations. Moreover, the total number of configurations is less than 2^{64} making the implementation trivial.

4.2 Solving approach

Calculating the game-theoretical value of a given configuration with perfect information, a fixed trump and starting player can be done with various existing techniques, in particular minimax search with $\alpha\beta$-pruning [12]. In practical approaches, this search technique is often equipped with an intermediate evaluation function that allows for partial exploration of the search tree as well as various heuristics aimed towards reducing the statespace. In contrast to most practical implementations, we are interested in the game-theoretical value of a configuration, the search procedure needs to traverse the complete statespace (unless it can theoretically determine for a given branch that it will never be relevant, cf. $\alpha\beta$-pruning). In our implementation we use the classical minimax search with $\alpha\beta$-pruning, but without any additional heuristics. This approach is rarely practical as the statespace for most games is too large.

An upper bound on the statespace for a given configuration is $8!^4$, where in every trick each player is able to select any of his cards in hand. Although rare, this situation can occur in practice, e.g., when every player is dealt cards from only one suit.

The distribution of the suits over the hands is the main contributing factor to the number of legal moves during a particular game (note that in case of the trump suit also the face value of the card can be of influence). Therefore, the size of the search space highly depends on this distribution.

4.3 Equivalence classes

In order to show the practicality of the solving approach presented in Section 4.2, we partition the total number of configurations into configurations with the same distribution of the number of cards from the same suit in each of the hands:

$$\begin{pmatrix} 8\,0\,0\,0 \\ 0\,8\,0\,0 \\ 0\,0\,8\,0 \\ 0\,0\,0\,8 \end{pmatrix}, \ldots, \begin{pmatrix} 2\,2\,2\,2 \\ 2\,2\,2\,2 \\ 2\,2\,2\,2 \\ 2\,2\,2\,2 \end{pmatrix}, \ldots, \begin{pmatrix} 0\,0\,0\,8 \\ 0\,0\,8\,0 \\ 0\,8\,0\,0 \\ 8\,0\,0\,0 \end{pmatrix},$$

where the hands are represented in rows and the columns represent the number of cards of each suit (both the order of players as well as the order of suits is unimportant). There are 981,541 of such equivalence classes [1], each containing a highly variable number of equivalent configurations ranging from 1 to

[1] See also: N.J.A. Sloane. The On-Line Encyclopedia of Integer Sequences, `https://oeis.org`. Sequence A001496.

40,327,580,160,000. Note that equivalence does not imply a similar play out of the game nor a similar outcome. It merely fixes the distribution of cards of certain suits over the hands with the intent of obtaining a similar average branching factor during the $\alpha\beta$-search.

This partitioning focuses on exploring the effect of the various statespace sizes on the performance of our exact solving approach as well as yielding an interesting dataset for the machine learning approach described in Section 5.

5 Machine learning approach

In this section, we elaborate on our machine learning approach to predict the outcome of a particular configuration.

The classification problem is as follows. Given full information about the hands of all players, predict whether the starting team will win or not, i.e., whether for a given deal h, $\textsc{Outcome}(\textsc{KlaverjasOpen}(h, \phi, \tau)) > 0$. Like in Section 4, we fix $\phi = \diamondsuit$ and $\tau = N$.

In order to apply supervised machine learning techniques to this problem, we need to have access to a dataset of generated games, and their outcome. For this we use all 981,541 games that were analyzed in Section 6.1. Generally, machine learning models are induced based on a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$ to map an input $\mathbf{x}$ to output $f(\mathbf{x})$, which closely represents y. Here, n represents the number of games in the dataset, $\mathbf{x}_i$ is a numerical representation of one such game and y_i is the outcome of that game. As such, y_i represents whether team *NS* will obtain more points than team *EW*, i.e, whether $\textsc{Outcome}(\textsc{KlaverjasOpen}(h, \phi, \tau)) > 0$.

The main challenge is representing a game g as feature vector $\mathcal{F}(g)$. This has been done for other games, see for example Dou Shou Qi [20]. We note that more complex algorithms, e.g., convolutional neural networks, can implicitly learn this mapping. However, it has been noted in [8] that card games do not have a clear topological structure to exploit. As such, defining convolutions on card games is a research question in its own right and beyond the scope of this work. Table 2

Table 2. Features constructed for the machine learning approach.

Name	Size	Parameters	Definition
card ownership	32	$\forall s \in S, \forall f \in F$	$p : \exists c : c \in h_p \wedge suit(c) = s \wedge face(c) = f$
suit counts	16	$\forall p \in P, \forall s \in S$	$\lvert h_{p,s} \rvert$
rank counts	32	$\forall p \in P, \forall r \in R$	$\lvert \{c \in H_p : rank(c) = r\} \rvert$
points	4	$\forall p \in P$	$\sum_{c \in h_p} points(c)$
stdev per player	4	$\forall p \in P$	$\text{stdev}(\forall s \in S : \lvert h_{p,s} \rvert)$
stdev per suit	4	$\forall s \in S$	$\text{stdev}(\forall p \in P : \lvert h_{p,s} \rvert)$
stdev (game)	1		$\text{stdev}(\forall p \in P, \forall s \in S : \lvert h_{p,s} \rvert)$
top cards	16	$\forall p \in P, \forall s \in S$	$\lvert \{c \in H_{p,s} : rank(c') < rank(c) \vee suit(c') \neq suit(c)\} \rvert$ with $c' \in o_p$

shows an overview of the handcrafted features that we defined. The first column defines a name for each group of features, the column 'size' denotes how many of those features can be generated. The column 'parameters' defines how this number of features can be generated. The last column defines how each feature can be generated.

Card ownership is a perfect mapping from a configuration containing all cards to a tabular representation. It describes for each card to which player it belongs. Suit counts, rank counts and points represent some basic countable qualities of the hand that a given player has. The standard deviation (stdev) gives a measure of how equal the suits are spread (per player, per suit and for the whole game). Note that the maximum obtainable standard deviation (per player, per suit and per game) is 3.46. If the game has a standard deviation of such value, this means that all players have all cards from a given suit (and the player with the cards from the trump suit will win the game). If the game has a standard deviation of 0, this means that all players have exactly two cards of each suit. The top cards denote how many cards of a given suit a player has that can not be beaten by the other team (except using trump cards).

Additionally, we can construct several convenience features oriented on teams, that can be exploited by the classifier to assess the card quality of the teams at once. For suit count (per suit), rank count (per rank), points and top cards (per suit), the appropriate team feature is the sum of both players for that feature. For example, the feature 'suit count of $\diamondsuit$ for team NS', is the sum of 'suit count of $\diamondsuit$ for player N' and 'suit count of $\diamondsuit$ for player S'.

6 Experiments

In this section we present results of using the exact approach and the machine learning approach, as well as a comparison between the approaches.

6.1 Exact approach results

As discussed in Section 4, the number of configurations is $\binom{32}{8}\binom{24}{8}\binom{16}{8}$, which is approximately $9.956 \cdot 10^{16}$. Therefore it is nontrivial to solve a representative sample of all configurations. Instead, we sample according to the equivalence classes as defined in Section 4.3. From each equivalence class we randomly select one configuration yielding a set of 981,541 configurations. For all of these configurations the game-theoretical score is calculated using minimax with $\alpha\beta$-pruning. No transposition tables were used. Note that when the team of the lead player can no longer obtain more than half of the points, all points will be assigned to the other team. Note that the leaf count within an equivalent class can differ significantly due to i) the rule that a player needs to play a higher trump card if possible and ii) the dynamics of $\alpha\beta$-pruning combined with how the cards are delt.

Figure 1 shows a CDF of the leaf count of the $\alpha\beta$-algorithm. This leaf count is directly propertional to the running time. Most configurations can be calculated in

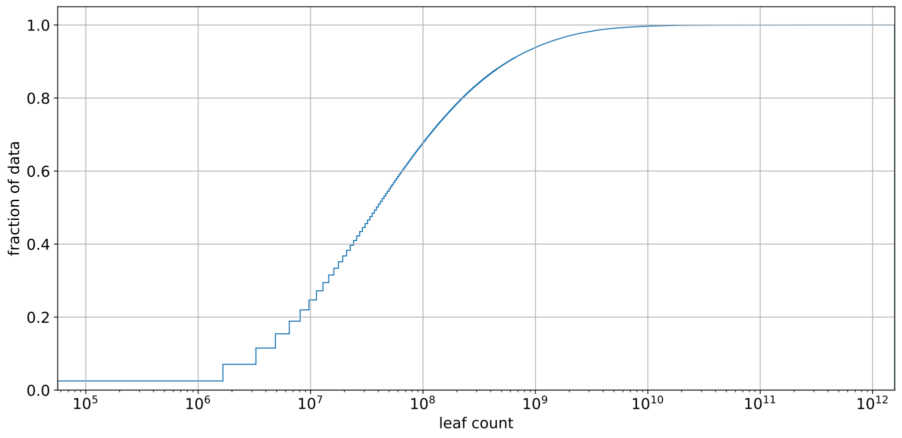

Fig. 1. Cumulative Distribution Function (CDF) of the leaf count of the exact algorithm for each of the 981,541 instances (see Section 4.3) of Klaverjas.

on average 1.5 CPU minutes, having between 10^6 and 10^9 leafs. A few instances with around 10^{11} leafs took around 45 minutes. However, one configuration required up to 4 CPU days, visiting $1.5 \cdot 10^{12}$ leafs.

6.2 Machine learning results

In this section we evaluate the performance of the machine learning techniques on the task to predict for a given configuration whether the team of the lead player can obtain more points than the other team. Our main interests are to evaluate the handcrafted features (as proposed in Section 5) and to compare machine learning approaches with exact search techniques (see Section 6.3). We use standard machine learning techniques, i.e., decision trees and random forests [3]. These have the advantage that they are interpretable and relatively insensitive to hyperparameter values. We use random forests with 64 trees. The other hyperparameters were set to their defaults, as defined in scikit-learn 0.20.0 [18].

For each configuration, we extract the following sets of features: card ownerships (first row in Table 2), the handcrafted features (all other rows in Table 2 and convenience team features) and all features (all rows in Table 2 and convenience team features). Clearly, the set of all features contains a richer source of information than the set of just the card ownerships. We evaluate the algorithm using 2-fold cross-validation. The problem is well-balanced (528,339 positive vs 453,202 negative observations). We record the predictive accuracy of the classifiers on all three feature sets. Predictive accuracy is the percentage of correctly classified observations.

The results are presented in Table 3, displaying the accuracy for different algorithms and feature sets. We note the following observations. As expected,

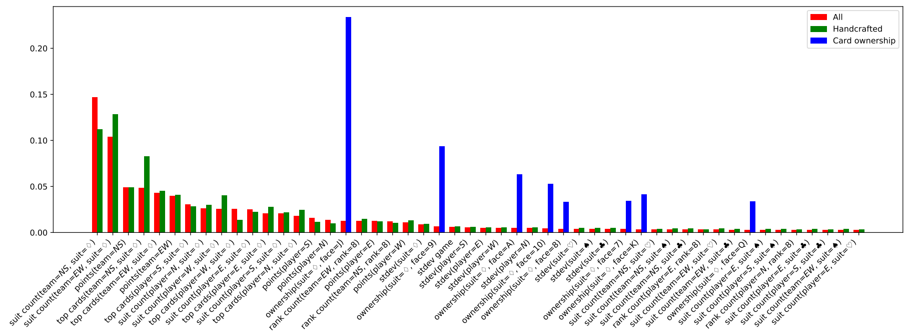

Fig. 2. Gini importance according to the random forest classifier.

the set of all features outperforms the set of just the card ownership features on both classifiers. Interestingly, the performance of the single decision tree on the handcrafted (and all) features almost equals that of random forest on just the card ownership features. Finally, the set of handcrafted features outperforms the set of all features for both classifiers. These observations lead us to belief that the handcrafted features on their own provide a more useful signal to learn from, in the context of tree-based models.

In order to study the behaviour of the classifier a bit better, we analyze the Gini importance as defined in [4]. Gini importance is defined as the total decrease in node impurity (averaged by all trees in the ensemble). Intuitively, a high Gini importance resembles that the feature was important for classification. Figure 2 shows a bar plot of the 50 most important features (sorted according to the feature set 'All'). We show Gini importance for the three feature sets. Note that each feature is applicable to either the 'card ownership' set (blue bars) or the 'handcrafted set' (green bars). As a limitation of this feature importance analysis, we note that the notion of feature importance is rather subjective, and that there is no guarantee that this directly correlated with the predictive accuracy of a model.

The results seem to confirm several expected patterns. First, from the handcrafted features that focus on a suit, the highest rated ones focus on the trump suit (♢). Second, from the card ownership features, the highest rated ones are

Table 3. Accuracy of machine learning algorithms on different feature sets.

Feature subset	Decision tree	Random forest
Card ownership	82.44	88.16
Handcrafted	88.02	91.98
All	87.96	91.79

the ones that focus on the top two trump cards (♢J and ♢9). Finally, from the rank count features, the highest rated ones focus on the highest rank (rank 8).

We note the following observations. First, when provided with all features, the handcrafted features provide the highest Gini importance. Second, the random forest makes proper use of the convenience team features (cf. top three features). Finally, suit counts, top cards and points seem to be strong features, often used in the top ranked features.

6.3 Comparison of exact approach and machine learning approach

The two experiments above highlight how an exact algorithm based on $\alpha\beta$-search as well as a machine learning approach are both independently able to assess whether a certain hand of Klaverjas is winning. A comparison of which approach is "better" in terms of determining whether a hand is winning may at first glance seem uninteresting, as the exact algorithm always returns the correct answer. However, the number of leafs in the search tree that is traversed by the exact algorithm can be seen as an indicator of how difficult it is to exactly determine whether a hand is winning. A distribution of these leaf counts was presented in Section 6.1. Here, we compare this leaf count between four result sets from the machine learning model, namely the correctly classified win and loss instances, and the incorrectly classified win and loss instances.

Figure 3 presents results of this comparison. The figure shows on the vertical axis the number of leafs that were traversed by the exact algorithm, for all instances in each of the four result classes described above. Horizontal Gaussian noise was added to highlight the density of each result set at different leaf count values. Note the logarithmic vertical axis.

From the figure, we see that indeed, as discussed in Section 6.2, the majority of instances is correctly classified. More importantly, we observe two interesting patterns for the win instances (depicted in red and blue). First, it appears that the win instances require exploration of a larger search space than lost instances of the game. We believe that this is due to the fact that such instances require the algorithm to explore the maximum score up until the last trick of the game, whereas for lost games, the search tree can be pruned much earlier in the game when no more cards of substantial value are left. Second, we observe how for the correctly classified win instances (depicted in red), the number of leafs is significantly higher (note the logarithmic vertical axis). In fact, for incorrectly classified win instances (depicted in green) only substantially lower leaf counts are observed. It turns out that the machine learning algorithm is able to correctly classify the instances that are difficult to solve exactly (requiring up to four days of computation time, see Section 6.1). One possible explanation for this is the different objective of both approaches. Given a deal, the exact approach aims to find the set of moves that leads to the best possible score, whereas the machine learning approach only aims to classify whether a deal is winning or not. In future work we aim to study this in more detail, for example by comparing the exact approach agains a supervised regression model.

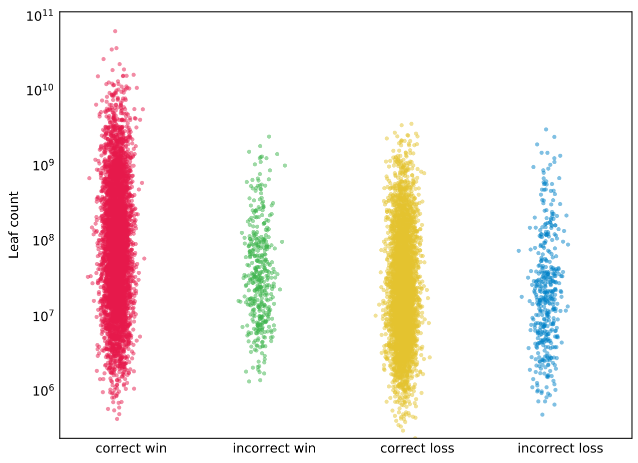

Fig. 3. Leaf count (according to exact algorithm) of correctly and incorrectly classified (according to machine learning approach) win and loss instances (1% sample).

7 Conclusion

In this paper, we have presented both an exact algorithm as well as a machine learning approach to solving the game of Klaverjas. In particular, we addressed the task of assessing whether a given distribution of cards over the hands of teams of players, is winning. It turns out that an exact algorithm based on $\alpha\beta$-search is able to determine this on average in a matter of minutes. In addition, the proposed machine learning approach employing simple features based on card ownership is able to predict whether a hand is winning with 88% accuracy. Adding more complex aggregated features and statistics related to meld points increases this accuracy to almost 92%. Interestingly, many of the cases where the machine learning algorithm consistently performs well, are in fact instances where the computation time (as a result of the number of leafs in the search tree) of the exact algorithm is longest (up to several days), evaluating over $1.5 \cdot 10^{12}$ play outs. This suggests that games that are difficult to assess for an algorithm, are in fact easy for humans, who typically use features similar to the machine learning approach in their decision making.

The findings presented in this paper highlight how in the future, a real-time artificial agent playing the game of Klaverjas may benefit from combining an exact approach with a machine learning approach, depending on what type of hand it is evaluating. A key question to then address is how we can a priori determine which component of such a hybrid algorithm we should use. The explored partitioning of the set of equivalence classes presented in this paper may provide a first basis of determining this.

In future work, we want to investigate if we can determine whether a hand is winning based on only one player's cards, rather than based on perfect information on all of the the team's cards. A starting point may be the PIMC approach discussed in [15], where to solve a game with imperfect information, typically many games with perfect information are played out. In addition, we want to investigate the use of transposition tables in order to reduce the search space even further.

Acknowledgements The second author was supported by funding from the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (grant agreement 638946). We thank F.F. Bodrij and A.M. Stawska for assistance with qualitative real-world validation of a relevant feature subset.

References

1. Beckenbach, E.F.: Applied Combinatorial Mathematics. Krieger Publishing Co., Inc. (1981)
2. Bonnet, É., Jamain, F., Saffidine, A.: On the Complexity of Trick-Taking Card Games. In: IJCAI. pp. 482–488 (2013)
3. Breiman, L.: Random Forests. Machine learning **45**(1), 5–32 (2001)
4. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.: Classification and regression trees. Chapman and Hall/CRC (1984)
5. Buro, M., Long, J.R., Furtak, T., Sturtevant, N.R.: Improving State Evaluation, Inference, and Search in Trick-Based Card Games. In: IJCAI. pp. 1407–1413 (2009)
6. Frank, I., Basin, D.: Search in games with incomplete information: a case study using Bridge card play. Artificial Intelligence **100**(1-2), 87–123 (1998)
7. Ginsberg, M.L.: GIB: Imperfect Information in a Computationally Challenging Game. Journal of Artificial Intelligence Research **14**, 303–358 (2001)
8. Hearn, R.A.: Games, Puzzles, and Computation. Ph.D. thesis, Massachusetts Institute of Technology (2006)
9. van den Herik, H.J., Uiterwijk, J.W., van Rijswijck, J.: Games solved: Now and in the future. Artificial Intelligence **134**(1-2), 277–311 (2002)
10. Hoogeboom, H.J., Kosters, W.A., van Rijn, J.N., Vis, J.K.: Acyclic Constraint Logic and Games. ICGA Journal **37**(1), 3–16 (2014)
11. Knuth, D.E.: The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions. Addison-Wesley Professional (2005)
12. Knuth, D.E., Moore, R.W.: An Analysis of Alpha-Beta Pruning. Artificial intelligence **6**(4), 293–326 (1975)
13. Kupferschmid, S., Helmert, M.: A Skat Player Based on Monte-Carlo Simulation. In: International Conference on Computers and Games. pp. 135–147. Springer (2006)
14. Long, J.R., Buro, M.: Real-Time Opponent Modeling in Trick-Taking Card Games. In: IJCAI. vol. 22, pp. 617–622 (2011)
15. Long, J.R., Sturtevant, N.R., Buro, M., Furtak, T.: Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search. In: AAAI (2010)
16. Parlett, D.: The Penguin Book of Card Games. Penguin UK (2008)
17. Pearl, J.: The Solution for the Branching Factor of the Alpha-Beta Pruning Algorithm and its Optimality. Communications of the ACM **25**(8), 559–564 (1982)

18. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
19. van Rijn, J.N., Takes, F.W., Vis, J.K.: The Complexity of Rummikub Problems. In: Proceedings of the 27th Benelux Conference on Artificial Intelligence (2015)
20. van Rijn, J.N., Vis, J.K.: Endgame Analysis of Dou Shou Qi. ICGA Journal **37**(2), 120–124 (2014)
21. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
22. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of Go without human knowledge. Nature **550**(7676), 354–359 (2017)
23. Wästlund, J.: A solution of two-person single-suit whist. The Electronic Journal of Combinatorics **12**(1), paper #R43 (2005)
24. Wästlund, J.: Two-person symmetric whist. The Electronic Journal of Combinatorics **12**(1), paper #R44 (2005)