Analysing the performance impact of differences within feature vector in the machine learning agent playing game of Schnapsen.

Vrije Universiteit Amsterdam
Intelligent Systems 2020

Table of Contents

1 Abstract 1
2 Introduction 1
3 Background Information 1
3.1 General 1
3.2 Bots 2
4 Research Question 2
5 Experimental setup 3
5.1 General 3
5.2 Bots 3
5.2.1 Rand bot 3
5.2.2 Rdeep bot 3
5.2.3 Bully bot 3
5.2.4 Minimax bot 4
5.2.5 Alpha-beta Pruning 4
5.2.6 Machine Learning 5
6 Results 6
7 Findings 8
7.1 Phase 1&2 8
7.1.1 Rand bot 8
7.1.2 Bully bot 8
7.1.3 Rdeep bot 8
7.2 Phase 2 8
7.2.1 Rand bot 8
7.2.2 Bully bot 8
7.2.3 Rdeep bot 9
7.2.4 Minimax bot 9
7.2.5 Alpha-beta bot 9
8 Conclusion 9
9 References 10
10 Appendices 11
10.1 Appendix A 11
10.2 Appendix B 13

1 Abstract

The paper examines and describes the effect of different sizes and types of feature vectors in the machine learning bot. To reach our goal, we have created four machine learning bots with different sizes of the feature vectors. Firstly, machine learning bots were trained. Secondly, they competed against each other and five other agents, aiming to draw conclusions based on the win and loss rate from each bot depending on the phase they were playing. Outcomes were provided and examined using tables which reveal the number of wins and losses and the average improvement or deterioration of each machine learning bot. As a result, we found out that the ratio of performance win/loss stays consistent between testing in phase 1 and 2 and testing in phase 1. We concluded that depending on the phase, the performance of individual bots, such as ml or ml_extended, is better in comparison to the rest of the variations.

2 Introduction

According to the Cambridge dictionary[1], Artificial intelligence is a field which aim is to create and design systems that would automate the process and solve a different kind of tasks that humans are capable of doing. One of these tasks concern processing voice, image, text recognition and learning. Intelligent systems are becoming an essential part of our everyday life: starting from smart home assistants and finishing with self-driving cars. Nowadays, a large part of artificial intelligence relies on machine learning which changes the approach of the computers performing the task and learning new information from the data. The aim of the paper is to analyze and understand how different features can affect the machine learning agent's performance.

3 Background Information

3.1 General

Schnapsen or Schnapser is an Austrian card game, consisting of 20 cards with different ranks from highest to lowest: Ace, 10, King, Queen and Jack. Each of them gives points as follows: 11, 10, 4, 3, 2. You always have to have five cards in your hand. The goal of the game is to obtain 66 points as fast as you can by taking tricks and bidding: you are getting points if the rank of your hard is higher than the opponent's, which would mean you are winning the round. One of the ways to obtain a large number of points is by marriages: if a player has a king or queen of the same suit, he or she can play a card and show another to the opponent. The marriage of cards that are non-trump-suit gives 20 points, of trump-suit gives 40 points. It is not always straightforward due to the heuristics

[1] `https://dictionary.cambridge.org/dictionary/english/artificial-intelligence`

of the game: opponent always interferes with your objectives. Thus, a player cannot set a specific objective. Schnapsen contains imperfect information, which leads to an increase in uncertainty in the game. Thus, hundreds of situations may occur. Building a bot that will win based on some heuristics and strategies is complex and challenging for multiple reasons. In order to design it, the game needs to be analyzed and multiple algorithms and heuristic strategies discussed.

3.2 Bots

The provided framework includes five agents to play against different versions of machine learning bot:

1. Rand bot:
   Rand bot is a simple agent: it enumerates legal moves and chooses a random move.
2. Rdeep bot:
   The rdeep bot assumes that players have the rand strategy and samples N amount of random games from the given move, ranks move by averaging heuristics of the following state.
3. Bully bot:
   The bully bot makes 3 checks: if it has trump suit, it will play it. If it does not have, it will play the card of the same suit. Otherwise, if neither is found, it will make a move with the highest rank available.
4. Minimax and alpha-beta bot:
   The bot is working only in phase 2 due to the fact that the information at this moment is perfect. It estimates the states, makes the best move and minimizes the outcome for the player.

4 Research Question

The research aims to answer the question: How is the performance of machine learning agent affected by different feature vectors in the game of schnapsen? The goal is to build a machine learning bot with additional features, make an investigation on how different features and lengths of feature vectors affect the game performance, time and results of the game. There are three possible outcomes of our research: when changes have no effect, positive effect, negative effect. The hypothesis is that the longer the feature vector, the more advanced cases of the game states the bot can comprehend and as a result, have a better winning rate is. An investigation is going to be made by comparing agents such as rand, bully, minimax, rdeep, alpha-beta, against the machine learning bot through tests that show how many wins each strategy has after playing against each other. As a result, machine learning bot has managed to get the highest win rate against other players. Moreover, the amount of time taken to win has been increasing. The results of the tests provided a strong incentive to conduct research about the machine learning algorithms and specific parts of the current state that had an impact on the winning rate. In the research, there are

going to be used two definitions: default features and additional features. The default features are player's 1 points, player's 2 points, the player's 1 pending points, player's 2 pending points, trump suit, phase of the game, the size of the stock, leader, whose turn and card played by an opponent. Additional features are called all the features that were not mentioned in the default features.

5 Experimental setup

5.1 General

In order to answer the research question, the contents of ML bot's feature vector have to be altered. Since that would be the only thing that has changed about the environment, causal links between the changes in feature vector and performance of the ML bot can be investigated. Firstly, the variations of ML bot will play against each other, as well as against rand, bully, rdeep, minimax, alpha-beta in order to examine the impact of adjustments to feature vectors in terms of performance against other types of agents. Latter versions of games were played starting from phase 1. Now, the attention will be drawn to the performance difference of new agents depending on which phase of the game will be played. In order to observe that new ML agents will play against the rand, bully, rdeep, minimax and alpha-beta in phase 2.

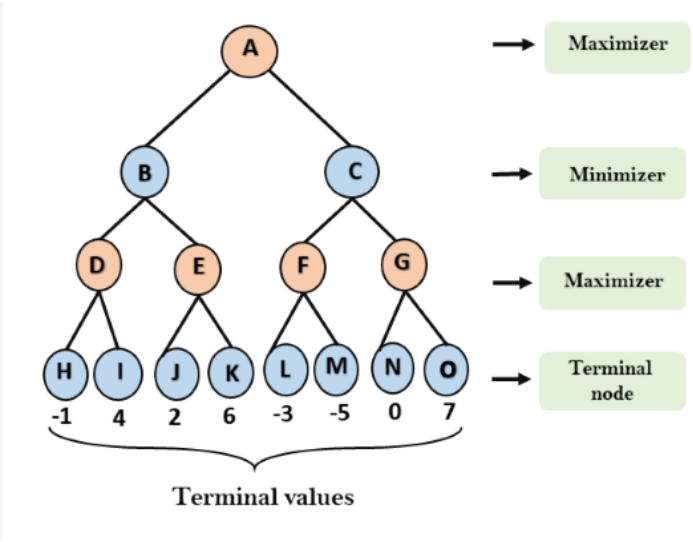
5.2 Bots

5.2.1 Rand bot is very simple and straightforward method that enumerates all legal moves and chooses a random card to play. This method is not complex at all and the win or loss depends only on pure luck, there is no algorithm.

5.2.2 Rdeep bot is based on the rand bot, however, a bit more complex in comparison to rand bot. It assumes that players follow the rand strategy and samples N amount of random games from the given move and ranks it by averaging heuristics of the evaluated value of the given state. The win or loss does not mainly depend on pure luck as in rand bot.

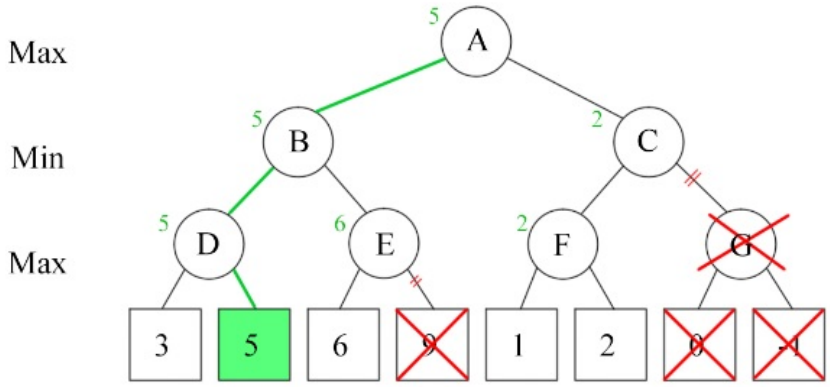
5.2.3 Bully bot takes into consideration 3 scenarios: having trump suit on hand, having a card of the same suit as the one that opponent has played, having neither of mentioned before. In the first scenario, if the bot has a trump suit, it will play it. In the second scenario, if the bot does not have a trump suit, it checks if it has the same card suit as the played card by the opponent and plays it. In the third scenario, if the bot has neither the trump suit nor card of the same suit, it will play a card with the highest rank.

5.2.4 Minimax bot is formulated for two-player games, covering cases where players take alternate moves and where they make simultaneous moves. It has been reproduced for the game with a higher level of complexity and decision-making processes with uncertainty. In computer science, minimax is used in several programming procedures.
When it comes to schnapsen, minimax is an observing algorithm that takes into consideration all the possible moves of the game. After testing, it was found that minimax bot had the highest win rate against bots such as rdeep and bully. However, the win rate against bully and rdeep does not define minimax as the top algorithm to be used in schnapsen game. The limitation of the minimax algorithm means that every state has to be visited twice: one time to find its children and a second time to evaluate the heuristic value. This results in slowness in complex games such as schnapsen. Minimax is optimal, complete with the time complexity $\mathcal{O}(b^m)$ and space complexity $\mathcal{O}(b^m)$.
In the figure below a classical search tree of a minimax algorithm is illustrated. The first step consists of the comparison of each value in the terminal state with the initial value of maximizer, which determines the higher nodes values. The purpose of it is to find the maximum among all.



5.2.5 Alpha-beta Pruning could be described as an algorithm which aims to minimize the number of nodes that are given by the minimax in its search tree. It contains two values within the recursive functions which are named Alpha and Beta. Moreover, it is mostly used for 2 player machine games. To elaborate, the primary purpose of its use is to reduce the pruning of the search tree. Thus, it

could be described as a modified version of the minimax algorithm, which is more optimal. Without that, applying alpha-beta pruning is always the most optimal approach to the task. Alpha-beta is considered faster than minimax because it does not explore some branches of the tree, which will not affect the value[2]. Thus, it will look twice as far in comparison to minimax in the same period of time.



5.2.6 Machine Learning (ML) is a branch of computer science intent on developing algorithms that can automatically learn from the input data. It generates algorithms that can learn a specific task without being programmed explicitly for it. It is mostly used for classification, regression, and clustering tasks[3]. To elaborate, an ML bot would aim to "learn" by identifying new patterns through the data given in a process and create new strategies to improve the performance of a specific task and lead to the desired results[4]. Machine learning algorithms are used in an increasing amount of different tasks such as email filtering, image processing and game theory (poker, schnapsen, chess). In schnapsen, Machine learning bot can learn the heuristic function rather than implementing. The advantage of machine learning is that in schnapsen, is needed only to calculate one heuristic function.
In our research, we are using altering vectors of machine learning to achieve our goal. Vector machine is a supervised algorithm that is used for classification or regression problems where the dataset teaches SVM about the classes so that it can classify all new data. It works by arranging the information into various classes by finding a line (hyperplane) which isolates the data set into classes.

[2] Slagle, J. R 1963
[3] M. G. Bobra and S. Couvidat (2015)
[4] Karlijn, Willems (2018)

This algorithm aims to maximize the distance between several different classes that are involved and is referred to as margin maximization. If the line that maximizes the distance between the classes is identified, the probability to generalize well to unseen data is increased.

There are four machine learning bots:

1. ml — this is the default bot that has the default features: the default features player's 1 points, player's 2 points, player's 1 pending points, player's 2 pending points, trump suit, phase of the game, the size of the stock, leader, whose turn and card played by an opponent
2. ml_limited[5] — the additional bot with a reduced feature vector, including the features such as perspective, player's 1 and 2 points, player's 1 and 2 pending points
3. ml_extended[6] — the additional bot that consists of default features and features such as lowest card value for player, lowest card value for the opponent (phase 2), ace count for the player, ace count for opponent (phase 2), hand strength for player (number of points/max possible points), hand strength for opponent (number of points/max possible points, phase2), number of cards of opponent's suit in player's hand, point difference, pending point difference
4. ml_mix[7] — the additional bot that consists of the feature vector, combined from ml_limited and ml_extended

6 Results

	rand		**bully**		**rdeep**		**ml**		**ml_extended**		**ml_limited**		**ml_mix**		**Total**	
	wins	losses	wins	losses	wins	losses	wins	losses	wins	losses	wins	losses	wins	losses	wins	losses
ml	1534	264	1179	373	660	884			756	714	838	660	858	608	5165	3503
ml_extended	1533	256	1249	273	690	783	746	729			847	648	832	632	5897	3321
ml_limited	1422	311	1155	320	584	910	692	788	651	819			787	674	5291	3822
ml_mix	1361	358	1157	348	622	850	603	908	575	860	713	735			5031	4059

Fig. 1. Overview of wins and losses (given in points) for each variation of ML agent in phase 1 & 2

[5] Detailed practical code implementation can be found in Appendix B
[6] Detailed practical code implementation can be found in Appendix A
[7] Detailed practical code implementation can be found in both Appendix A and B

	rand	**bully**	**rdeep**	**ml_extended**	**ml_limited**	**ml_mix**	**Average**
ml_extended	0.00%	5.60%	4.35%		1.06%	-3.13%	1.58%
ml_limited	-7.30%	-2.08%	-13.01%	-16.13%		-9.02%	-9.51%
ml_mix	-11.28%	-1.90%	-6.11%	-31.48%	-17.53%		-13.66%

Fig. 2. Overview of performance gain or lose for each new implementation of ML agent against the default model in phase 1 & 2

	rand		**bully**		**rdeep**		**minimax**		**alpha-beta**		**Total**	
	wins	losses	wins	losses	wins	losses	wins	losses	wins	losses	wins	losses
ml	673	479	766	476	447	788	478	751	493	728	2857	3222
ml_extended	681	492	717	545	447	770	471	778	474	729	2790	3314
ml_limited	611	548	656	582	452	771	475	788	464	758	2658	3447
ml_mix	574	566	644	576	423	810	506	739	445	762	2592	3453

Fig. 3. Overview of wins and losses (given in points) for each variation of ML agent in phase 2

	rand	**bully**	**rdeep**	**minimax**	**alpha-beta**	**Average**
ml_extended	1.17%	-6.83%	0.00%	-1.49%	-4.01%	-2.23%
ml_limited	-10.15%	-16.77%	1.11%	0.63%	-6.25%	-6.29%
ml_mix	-17.25%	-18.94%	-5.67%	5.53%	-10.79%	-9.42%

Fig. 4. Overview of wins and losses (given in points) for each variation of ML agent in phase 2

7 Findings

7.1 Phase 1&2

7.1.1 Rand bot

The ml bot has slightly better results in comparison to the ml_extended bot, mainly due to the randomness of the strategy that rand is uses. The worst performance was shown by ml_mix, which combines properties of ml_extended and ml_limited. All the variations of the ml bot had at least 3 times more wins than losses.

7.1.2 Bully bot

The ml_extended has noticeably better results from the games than other variations of the ml bot. Ml_limited showed the worst performance. All the variations of the ml bot had at least three times more wins than losses.

7.1.3 Rdeep bot

In the situation with rdeep, the results are entirely different. First of all, in all scenarios, rdeep wins more than variations ml bots. If we are comparing ml bots between each other, then the ml_extended has slightly better performance than ml. The worst rate of wins was demonstrated by ml_limited.

7.2 Phase 2

The overall performance of all variations of ml agent is significantly worse while playing matches in phase 2. The default agent had approximately 59.59% win rate while starting the match in phase 1, and when it starts in phase the win rate drops to 47.00%. Moreover, on average there is more narrow difference between the performance of new implementations of ml agent against the default ml agent. That suggests that new implementations of ml agent are closer to each other in overall performance while playing only in phase 2.

7.2.1 Rand bot

In direct comparison of default ml agent's performance against rand and new implementations of ml agent it can be observed that depending on the feature vector of ml agent there is a significant deviation between results. Ml_extended is the only bot outperforming default agent. Both ml_limited and ml_mix have remarkably low performance. Those results suggest that while playing in phase 2 it is beneficial to provide ml agent with additional features in feature vector, however, it does not provide noticeable performance gains.

7.2.2 Bully bot

In contrast to rand bot, in the case of bully bot all variations of ml agent perform worse than default agent. The ranking of which agent performs the best stays

consistent rating ml_extended as the best, followed by ml_limited and ml_mix being at the end. The performance loss is likely due to implementing features which do not allow for prediction of bully's actions.

7.2.3 Rdeep bot

Ml_extended doesn't display deviation from default agent's performance. Ml_limited peculiarly outperformed ml_extended as well as the default ml agent. Ml_mix continuously underperforms.

7.2.4 Minimax bot

The performance of new ml agents differs from the usual representation. The unusual results might be the outcome of the unique way minimax acquires it's strategies for move choice. For the first and only time ml_mix outperforms all other ml variations. That would suggest that the new features have considerable advantage over the default features that were removed in the ml_limited.

7.2.5 Alpha-beta bot

Unexpectedly, alpha-beta bot doesn't follow the outcomes observed with minimax. Instead, the standard order of performance is seen as well as a performance loss in all three new ml bot implementations in comparison to default one.

8 Conclusion

The paper aimed to answer how is the performance of machine learning agent affected by different feature vectors in the game of schnapsen. One of the most critical steps in the direction of forming the conclusion were the results from the bots' plays against each other and results. During the competition between these bots, the connection was drawn between the results and the phase in which they were playing. So, in phase 1 and 2, the ml_extended version of the bot that is built on top of the default performed better than the default agent. The ml_limited, the bot with a shorter feature vector, had considerably worse performance in comparison to the default bot. Ml_mix bot has the worst performance. When it comes to only phase 2, the moment when both players reveal their cards, all bots performed worse in comparison to the default bot. The ratio of performance gain/loss stays consistent between testing in phase 1 and 2 and testing in phase 1. When it comes to the comparison of variations of ml bots to rand, rdeep, minimax and alpha-beta, ml_extended has better performance on average. There is some connection found between the number of features implemented and win/loss ratio. However, the final conclusion related to that fact cannot be drawn at that moment due to the limited time. In order to prove that the larger feature vector, the better performance is, we would need to make more detailed research with a bigger amount of tournaments, feature vectors involved. However, the conclusion can be drawn that depending on the feature vector size and the phase being played, the performance of the bot will be different.

9 References

- Hazewinke, Michiel, ed. (2001), “Minimax principles”, Encyclopedia of Mathematics, Springer+Business Media B.V/ Kluwer Academic Publishers, ISBN.
- M. G. Bobra and S. Couvidat (2015).“SOLAR FLARE PREDICTION USING SDO/HMI VECTOR MAGNETIC FIELD DATA WITH A MACHINE-LEARNING ALGORITHM“.The Astrophysical Journal.
- Karlijn, Willems (2018). “Machine Learning in R for beginners“.
- DE Knuth, RW Moore (1975). Artificial intelligence, www-public.imtbs-tsp.eu.
- Slagle, J. R. (November, 1963). "Game Trees, m and n Minimaxing, and the m and n Alpha-Beta Procedure,11 Al Group Rep. No. 3, UCRL-4671, Lawrence Radiation Laboratory, University of California.
- Kasneci, G., Stern, D. H., Graepel, T. K. H., & Herbrich, R. (2012). U.S. Patent Application No. 12/975,177.
- Liu, H., & Motoda, H. (2012). Feature selection for knowledge discovery and data mining (Vol. 454). Springer Science & Business Media.
- Joachims, T. (1998, April). Text categorization with support vector machines: Learning with many relevant features. In European conference on machine learning (pp. 137-142). Springer, Berlin, Heidelberg.

10 Appendices

10.1 Appendix A

Code implementation of new features. They are implemented in ml_extended as well as ml_mix.

```
#init required vars to be appended to feature set
player_hand = state.hand()
opponent_hand = state.hand_opponent()
cards_in_hand = 5 # there are 5 cards in a hand at all times
max_hand_points = cards_in_hand * 11 # ace is worth 11 points

# Append lowest card value in player hand && ace count && hand strength
hand_points = 0.0
ace_count = 0.0
current_lowest = 0.0
for card in player_hand:
    if Deck.get_rank(card) == "A":
        if current_lowest < 1.0:
            current_lowest = 1.0
        ace_count += 0.2
        hand_points += 11
    elif Deck.get_rank(card) == "10":
        if current_lowest < 0.75:
            current_lowest = 0.75
        hand_points += 10
    elif Deck.get_rank(card) == "K":
        if current_lowest < 0.5:
            current_lowest = 0.5
        hand_points += 4
    elif Deck.get_rank(card) == "Q":
        if current_lowest < 0.25:
            current_lowest = 0.25
        hand_points += 3
    elif Deck.get_rank(card) == "J":
        if current_lowest < 0.0:
            current_lowest = 0.0
        hand_points += 2

feature_set.append(current_lowest)
feature_set.append(ace_count)
feature_set.append(hand_points/max_hand_points)

# Append lowest card value in opponent's hand && ace count && opponent hand strength
hand_points_opp = 0.0
ace_count_opp = 0.0
```

```
current_lowest_opp = 0.0
if (phase == 2):
    for card in opponent_hand:
        if Deck.get_rank(card) == "A":
            if current_lowest_opp < 1.0:
                current_lowest_opp = 1.0
            ace_count_opp += 0.2
            hand_points_opp += 11
        elif Deck.get_rank(card) == "10":
            if current_lowest_opp < 0.75:
                current_lowest_opp = 0.75
            hand_points_opp += 10
        elif Deck.get_rank(card) == "K":
            if current_lowest_opp < 0.5:
                current_lowest_opp = 0.5
            hand_points_opp += 4
        elif Deck.get_rank(card) == "Q":
            if current_lowest_opp < 0.25:
                current_lowest_opp = 0.25
            hand_points_opp += 3
        elif Deck.get_rank(card) == "J":
            if current_lowest_opp < 0.0:
                current_lowest_opp = 0.0
            hand_points_opp += 2
feature_set.append(current_lowest_opp)
feature_set.append(ace_count_opp)
feature_set.append(hand_points_opp/max_hand_points)

# Append how many cards of opponent's suit do you have
same_suit_cards = 0.0

for card in player_hand:
    if opponents_played_card is not None:
        if Deck.get_suit(card) == Deck.get_suit(opponents_played_card):
            same_suit_cards += 1
feature_set.append(same_suit_cards/cards_in_hand)

# Append one-hot encoded points difference
p_diff = p1_points - p2_points
feature_set += [p_diff, 0] if p_diff > 0 else [0, -1 * p_diff]

# Append one-hot encoded pending points difference
pp_diff = p1_pending_points - p2_pending_points
feature_set += [pp_diff, 0] if pp_diff > 0 else [0, -1 * pp_diff]
```

10.2 Appendix B

Code implementation of ml_limited and ml_mix. Showing which features were left.

```
# Perform one-hot encoding on the perspective.
perspective = [card if card != 'U'   else [1, 0, 0, 0, 0, 0] for card in perspective]
perspective = [card if card != 'S'   else [0, 1, 0, 0, 0, 0] for card in perspective]
perspective = [card if card != 'P1H' else [0, 0, 1, 0, 0, 0] for card in perspective]
perspective = [card if card != 'P2H' else [0, 0, 0, 1, 0, 0] for card in perspective]
perspective = [card if card != 'P1W' else [0, 0, 0, 0, 1, 0] for card in perspective]
perspective = [card if card != 'P2W' else [0, 0, 0, 0, 0, 1] for card in perspective]

# Append one-hot encoded perspective to feature_set
feature_set += list(chain(*perspective))

# Append normalized points to feature_set
total_points = p1_points + p2_points
feature_set.append(p1_points/total_points if total_points > 0 else 0.)
feature_set.append(p2_points/total_points if total_points > 0 else 0.)

# Append normalized pending points to feature_set
total_pending_points = p1_pending_points + p2_pending_points
feature_set.append(p1_pending_points/total_pending_points if total_pending_points > 0 else
feature_set.append(p2_pending_points/total_pending_points if total_pending_points > 0 else
```