Estimating Winning Probability for Texas Hold'em Poker

Wenkai Li and Lin Shang

Abstract—**Among all the technologies in creating a good poker agent, estimating winning probability is a key issue. In this paper, we propose an approach to estimating winning probability for Texas Hold'em poker. We design a data structure using both the observable data from the current board and the history. A Support Vector Machine classifier is trained and 5-fold cross-validation is employed. We create a poker agent with some decision making strategies to compete. Experimental results show that our method has outperformed three other agents in precision of estimating winning probability.**

Index Terms—**Opponent modeling, support vector machine, texas hold'em poker, winning probability.**

I. Introduction

Nowadays, many papers have been published and competitions like the Annual Computer Poker Competition have attracted many researchers' interest [1]. The following are some most important properties of poker [2].

1) Imperfect information. This property creates a necessity for using and coping with deception and ensures a theoretical advantage of using randomized mixed strategies.
2) Non-deterministic dynamics. This means that the cards we get are stochastic.
3) Partial observable. We can't always know the opponent's hole cards, even when a game is over.
4) Multi-players. There are two and mostly more than two players.

Poker research has led to the investigation of a wide range of new algorithms and approaches [3].

Knowledge-based systems: It requires experts with domain knowledge to aid the design of the system. There are two typical categories, which are including rule-based expert systems and formula-based methods. For the former, a collection of if-then rules is created for various scenarios. For the latter, inputs of a numerical representation of the hand strength and pot odds are accepted by the system and a probability triple is given for a betting decision making [4].

Monte-Carlo simulation: It involves choice of random samples in the game tree and playing until a leaf node reached where we know the payoff value. It generally requests a lot of time and computing resource in simulation [5].

Game theoretic equilibrium solutions: It's a method of studying strategic decision making and optimal strategies. The field of game theory provides tools to study situations where multiple agents compete and interact within a particular environment. Nash equilibrium is a generally solution in game. In current computer poker research, Counterfactual regret minimization described by M.B. Johanson is a promising algorithm, which has been used in both heads-up limit and no limit poker agents [6]-[7].

Case-base reasoning: It is a type of method using algorithms like the k-Nearest Neighbor algorithm where a set of cases are stored and maintained that encode knowledge of previously situations and solutions [8].

Evolutionary algorithms and Neural Networks: It tries to evolve strong poker agents via evaluating. In these agents, the algorithm gives the next action [9].

Bayesian Network: It is a directed acyclic graph where each node in it represents a variable associated with conditional tables. The goal of the graph is to forecast the win rate in a certain condition [10].

Support Vector Machine Classifier: J. Pfund shows us the use of Support Vector Machine classifier in poker research [11]. They use the classifiers to tell which action to choose.

Apart from these fields, more and more contributions have focused on **opponent modeling** nowadays. The more you know your opponents, the more higher rate you will win. Opponent modeling has been studied since 1998 [12]. However, not until recent years has it obtained so much attention [13].

As is summarized by AAJ van der Kleij, there are two essential tasks: predicting the opponent's action and estimating the winning probability [4]. But past papers as have been listed above mainly focus on the opponent's actions based on styles, distributions or states [13]. Meanwhile, there are two data sources: data from the current inning and the previous innings. However, present contribution mainly uses data from the current inning online. It has been ignored for these two parts integrated. From that we design a data structure using both data from the current inning and the previous innings to estimate the winning probability. We train a Support Vector Machine classifier and do 5-fold cross-validation to see the precision in estimating whether a certain hand in a inning will win. This classifier can achieve the winning probability of a certain hand. With this classifier we create a poker agent with some simple decision-making strategies and use this to compete with other three different poker agents. Results show that our method has a precision higher than 75% in average in estimating whether a certain hand will win. It is effective on building a poker agent.

In Section II, we will give a brief introduction of Texas Hold'em poker. After that, we show the details of our data

Manuscript received November 16, 2012; revised January 8, 2013. This work was supported in part by the National Science Foundation of China (Grant Nos.61035003, 61170180).

The authors are with the State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology at the Nanjing University, Nanjing, China (e-mail: easerene@gmail.com).

DOI: 10.7763/IJMLC.2013.V3.275

structure design, the classifier and agent design. Section IV presents the experiment results with some analysis. The final section contains a conclusion and some directions for future work.

A. Texas Hold'em Poker

Texas Hold'em poker: In Texas Hold'em pokers there are 4 steps in a round, which are pre-flop, flop, turn and river [2]. During the pre-flop all players at the table are dealt two cards face-down called hole cards. Before any betting takes place, two forced bets are contributed to the pot: the small blind and the big blind. The big blind is typically double that of the small blind. Then the first round of betting is followed. Next the dealer places three community cards face-up in the middle of the table, that is the flop step. This is followed by the second round of betting. Then it is the turn step, and a further more card face-up is added to the community cards, following by the third round of betting. At the last step, the river, the fifth face-up card is added to the community cards followed by the last round of betting. A showdown occurs after the river where the remaining players reveal their hole cards and the player with the best hand wins all the wagers in the pot. If two or more players have the same best hand then the pot is split amongst the winners. The possible betting actions are described as follows:

Fold: A player contributes no further wagers to the pot. This will abandon their hand and any right to contest the wagers that have been added to the pot.

Check/Call: A player commits the minimum amount of wagers in order to stay. A check requires a commitment of zero further wagers, whereas a call requires an amount greater than zero.

Bet/Raise: A player commits greater than the minimum amount of wagers necessary to stay. When the player can check, but decides to invest further wagers in the pot, this is known as a bet. When the player can call, but decides to invest further wagers in the pot, this is known as a raise.

Limit game: In a limit game all bets are in increments of a certain amount.

No limit game: In a no limit game players can wager up to the total amount of chips they possess in front of them.

II. Our Method

In this paper, we aim to combine both data from the current board and the history to estimate the winning probability when our agent facing a certain opponent. Firstly, we design a data structure using both the observable data from the current inning and the previous innings. Then we train Support Vector Machines classifier and do 5-fold cross-validation to see the precision in estimating whether a certain hand in a inning will win. This classifier can tell the winning probability of a certain hand in a inning. With this classifier we create a poker agent with some simple decision-making strategies and use this agent to compete with other three different poker agents. While the competition is going on, we use data from previous innings to update our classifier. Fig. 1 shows the relation between the classifier and the agent.

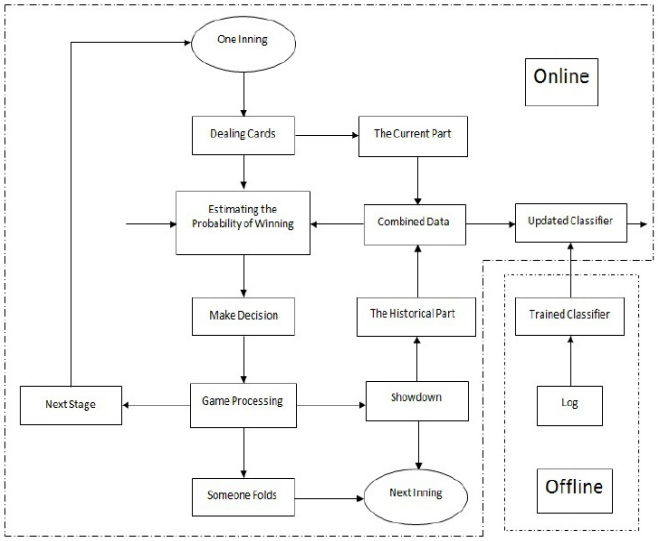

Fig. 1. The flow chart of Our Agent Design

In our agent, the classifier is used to estimate winning probability. We will store data when a game comes to a showdown. This data will be used to construct the input data of our classifier. At the same time, we use this data to update the priori probability of our classifier. In the decision making part of our agent, it chooses a strategy like this:

1) If the winning probability is high enough (In our implementation, if the probability is bigger than 0.7 we think it is high enough. Here 0.7 means our agent will win in seventy percent of cases.), we will choose to raise;
2) If the classifier tells it will win but the winning probably isn't so high(In our implementation, if the probability is smaller than 0.7 but bigger than 0.5, we think it will win but the winning probably isn't so high. That is to say, in fifty to seventy percent of cases, our agent will win.), we will choose to call;
3) In other situations, we will choose to fold.

In the following part of this section, we will show the Classifier and how to estimate Winning Probability at first. After that we will present the details of the data structure design.

A. Estimating Winning Probability and Classifier

In our experiments, we use LIBSVM as our classifier [14]. It is a library for Support Vector Machines developed since the year 2000. It suits our goal quite well. In our experiment, we give it an input data and it will tell us whether we will win with reliability. The reliability is a probability. We use this probability as the winning probability.

B. Data Structure Design

The whole data structure combines two parts: the Current Part and the Historical Part. The Current Part has 129 attributes. It is used to represent the data which can be obtained from the current inning. The Historical Part has 2172 attributes. It is used to represent the data which can be obtained from previous innings. There are 2301 attributes in our final data structure. The final data structure is shown in Fig. 2.

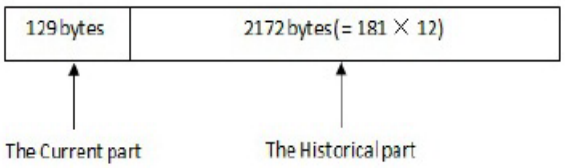

Fig. 2. The Whole Data Structure

1) The current part

In our method, we use the current board data to construct the Current Part. In an inning there are three critical data we can use: our agent's hole cards, the board cards, the actions sequence of all the players.

As there are 52 cards at most, we use 52 bytes to represent them. Each of the byte will be 1 or 0, in which 1 means a certain card exist while 0 means not. For the hole cards and the board cards, we use 104 bytes to represent them.

In a 2 player limit Texas Hold'em, there are at most four raise actions in a single stage. That is to say, adding the probable call actions at the beginning and the end of a stage, there will be six actions at most in a stage. For all the actions in four stages, we use 24 bytes to represent them. Each one of these 24 bytes will be 1 or 0, in which 1 means the action is raise while 0 means call. We don't consider fold action, because once there is an agent takes fold action this inning will be over.

In addition, as the position has some influence on the player's action decision, we use a byte to represent it. This byte takes values of 1 or 0. If this byte is 0, the opponent will be the first to take action. If not, our agent will be the first to take action.

Adding all these bytes together, the Current part has 129 bytes as shown in Fig. 3.

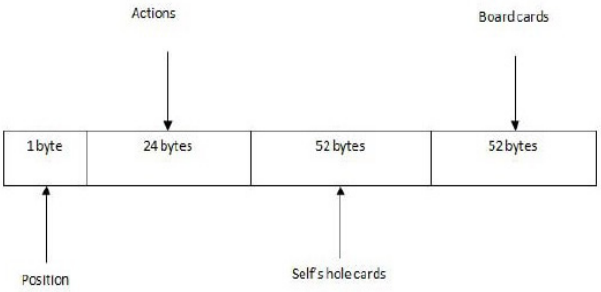

Fig. 3. The Current Part of Data Structure Design

2) The historical part

The Historical Part keeps a record of the historical data about the opponent in the previous innings. It records scenarios when the opponent's cards are in different hand ranks. We classify the hand rank into 12 ranks based on traditional partition of hand ranks which could be seen in Table I. The new ranks could be seen in Table I.

We can represent the Historical Part as a 12 dimensional array. Each dimensional represents a scenario when the opponent has a certain rank of cards. In each dimension, just as what the Current Part has, there are bytes to represent the opponent's position, our agent's hole cards, the board cards and the action sequence. The Historical part has 52 bytes to represent the opponent agent's hole cards, which is what the Current Part doesn't have. That is to say, the Historical part has 2172 (= 181 × 12) bytes together as is shown in Fig. 4.

TABLE I: THE 12 HAND RANKS

Hand	Example
High card Low	♠J♥9♦7♣5♦2
High card Middle	♠A♥J♦7♣5♦2
High card High	♠Q♥J♦10♣9♦2
One pair Low	♦10♥10♠8♣7♦5
One pair High	♦A♥A♠10♣7♦5
Two pair	♦Q♠Q♦9♥9♣4
Three of a kind	♦7♠7♣7♥6♣3
Straight	♦Q♠♦10♣9♥8
Flush	♦A♦Q♦J♦9♦6
Full house	♥Q♦Q♣Q♠5♠5
Four of a kind	♦Q♥Q♣Q♠Q♦9
Straight flush	♦K♦Q♦J♦10♦9

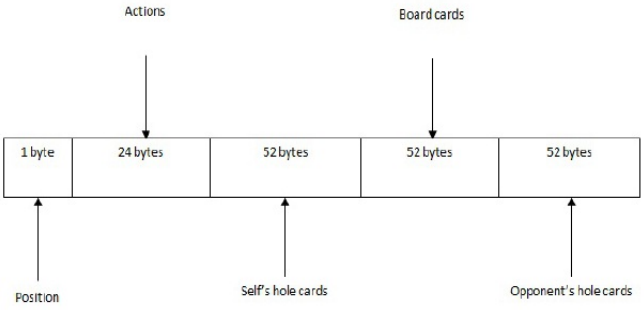

Fig. 4. The Historical Part of Data Structure Design

III. EXPERIMENT RESULTS AND DISCUSSION

A. Experiment Setup

In our experiments, we will do:

1) Firstly, to prove that our data structure design is effective for a certain agent, we use a single agent's log data to train our classifier and do 5-cross validation. It is assumed as usual that if the precision rate is high enough, for example bigger than 70%, our data structure design is effective.
2) Secondly, to prove that our data structure design is effective when facing a new player whose log data can't be obtained ahead, we train a classifier with multiple players' log data that could be obtained in advance. We suppose that a new opponent's strategy should be similar with some agents that we have seen before. So if our data structure is effective the 5-cross validation results of our classifier will good. In this paper, we choose the ACPC(Annual Computer Poker Competition)'s log data for this experiment [1]. The ACPC's log data is described in Fig. 5.

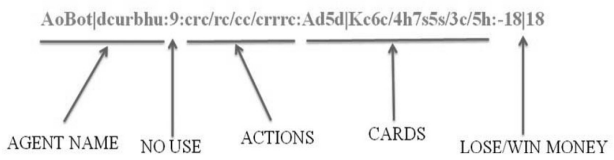

Fig. 5. The ACPC's Log's format

3) At last, an agent integrated with our classifier is necessary. If our agent could be at the same level with or

better than other agents, it will prove that our method is effective. In our experiments, we use three other agents for comparison. These agents are:

A Simple Random Strategy Agent The Simple Random Strategy Agent which we used as an opponent doesn't care what cards it has, it will just choose to fold, call or raise randomly.

A Rule-based Agent In our Rule-based Agent, we use the hand rank of the agent's card hand to make decision. Depending on the psychological confidence on winning or losing of a certain hand rank, the Rule-based Agent will choose to fold, call or raise. In our implementation, if the hand rank of the agent's card hand is stronger than {\bf three of a kind}, it will choose to raise. If the hand rank of the agent's card hand is stronger than one pair, it will choose to call. In other situations, it will choose to fold.

A Bayesain Network Agent The Bayesain Network Agent uses a Bayesian decision network to model the program's poker hand, the opponent's hand and the opponent's playing behavior conditioned upon the hand, and betting curves to randomize betting actions. It is an implementation based on Korb and his partners' work on BPP agent [9].

In our experiments, we use LIBSVM as our classifier [14]. It is a library for Support Vector Machines developed since the year 2000. In our experiments, the classification results will show whether we will win with reliability. The reliability is a probability which is known as the winning probability.

The following parts B and C will show the results of these three tasks.

B. Results and Analysis

1) 5-cross Validation Results

Precision results when using a single player's log for training is shown in Table II.

TABLE II: USING A SINGLE PLAYER'S LOG FOR TRAINING

Rounds	40,000	60,000	200,000
precision	63.57%	82.13%	79.07%

From that, we can see that our method has high performance when using a special agent's log data for training. The results prove that, a Support Vector Machine classifier using our data structure is effective at estimating whether a hand will win or lose. It also proves that our data structure design is effective.

Precision results when using multiple players' log for training is shown in Table III.

TABLE III: USING MULTIPLE PLAYERS' LOG FOR TRAINING

Rounds(1000)	100	400	900	1,000	1,200
Precision(%)	61.20	60.27	60.29	61.22	61.03

As shown in Table III, when using multiple players' log for training, our classifier can make right estimation on win or lose in at least 60% of cases. This shows that will be effective to use plenty of agents log data, which could be collected in advance. This data will be employed to train a classifier and used to estimate winning probability when our agent competes with a new agent.

2) Competition Results

In the following tables, the numbers in the Win rows is a ratio between the money our agent wins and the small blind. If the number is positive/negative, it means our agent wins/loses some money.

Table IV shows the match results between our agent and the Simple Random Strategy Opponent.

TABLE IV: OUR AGENT VS THE SIMPLE RANDOM STRATEGY OPPONENT

Rounds	2,000	20,000	200,000
Win	-0.341	-0.060	+0.312

Table V shows the match results between our agent and the Rule-based Agent.

TABLE V: OUR AGENT VS THE RULE-BASED AGENT

Rounds	2,000	20,000	200,000
Win	-0.630	-0.283	+0.110

Table VI shows the match results between our agent and the Bayesain Network Agent.

TABLE VI: OUR AGENT VS THE BAYESAIN NETWORK AGENT

Rounds	2,000	4,000	6,000	80,000
Win	-0.623	-0.406	-0.175	-0.015

The competition results indicate our method is effective to create an agent and our data structure design makes our classifier more stable. It also justifies our suppose that a new opponent's strategy will be similar with some agents that we have seen before. And these agents' log data could be collected in advance.

C. Discussion

From the 5-cross validation precision results using a single player's log for training we can see that our method has high performance when using a special agent's log data for training. The results prove that, a Support Vector Machine classifier using our data structure is effective at estimating whether a hand will win. It also shows that our data structure design is effective.

The results using multiple players' log for training show that our classifier can achieve right estimation on win or lose in at least 60% of cases. This means it is effective to use plenty agents' log data.

The competition results indicate that our method is effective to create an agent and our data structure design makes our classifier more stable. It also implies that a new opponent's strategy should be similar with some agents that we have seen before, while these agents' log data could be collected in advance.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we design a data structure using both the observable data from the current inning and the previous innings. Using this data structure, we train a Support Vector Machine classifier and do 5-fold cross-validation to see the precision on estimating whether a certain hand in a inning

will win. This classifier can achieve the winning probability of a certain hand in a inning. With this classifier we create a poker agent with some simple decision-making strategies and use this agent to compete with other three different poker agents. As the results show, our data structure design is effective. Using multiple players' log for training, our classifier can make right estimation on win or lose in at least 60% of cases. Our agent design can achieve nearly the same accuracy comparing with other agents. It indicates our method is effective and our data structure design makes our classifier more stable. Meanwhile, it implies that a new opponent's strategy should be similar with some agents that we have seen before, while these agents' log data could be collected in advance.

For the future, we will work on the following:

1) **More comprehensive decision-making strategies** In the implementation of our agent, we just divide the probability of winning into three parts and make a decision to fold, call or raise accordingly. More comprehensive strategies like adding Monte Carlo simulations is essential in order to see the real potential of our method in creating computer poker players.
2) **Real-world training data and competition** Training with real competition data is interesting to compete with human poker experts.

ACKNOWLEDGMENT

We would like to thank Ruobing Li and Yujing Hu for their helps in experiments and Bing Xue for her comments and help for improving the paper.

REFERENCES

[1] The annual computer poker competition. [Online]. Available: http://www.computerpokercompetition.org/, 2010.
[2] D. Billings, "Algorithms and assessment in computer poker," University of Alberta, 2006.
[3] J. Rubin and I. Watson "Computer poker: A review," *Artificial Intelligence*, 2011.
[4] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron "The challenge of poker," *Artificial Intelligence*, pp. 201–240, 2002.
[5] AAJ van der Kleij, *Monte Carlo Tree Search and Opponent Modeling through Player Clustering in no-limit Texas Hold'em Poker*, 2010.
[6] M. B. Johanson "Robust strategies and counter-strategies: Building a champion level computer poker player," in Masters Abstracts International, 2007.
[7] D. P. Schnizlein, State translation in no-limit poker, University of Alberta, 2009.
[8] A. Sandven and B. Tessem "A case-based learner for poker," in *Proc. The Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, Helsinki, Finland, 2006.
[9] H. Quek, C. Woo, K. Tan, and A. Tay "Evolving Nash-optimal poker strategies using evolutionary computation," *Frontiers of Computer Science in China*, pp. 73–91, 2009.
[10] A. E. Nicholson, K. B. Korb, and D. Boulton "Using bayesian decision networks to play texas hold'em poker," *International Computer Games Association (ICGA) Journal (Unveröffentlichter Entwurf)*. Monash University, Victoria, Australien, 2006.
[11] J. Pfund, "Support Vector Machines in the Machine Learning Classifier for a Texas Hold'em Poker Bot," 2007.
[12] D. Billings, D. Papp, J. Schaeffer, and D. Szafron "Opponent modeling in poker," in *Proceedings of the National Conference on Artificial Intelligence*, 1998, pp. 493–499.
[13] T. C. Schauenberg, "Opponent modeling and search in poker," University of Alberta, 2006.
[14] C. C. Chang and C. J. Lin "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, pp. 27:1–27:27, 2011.

Wenkai Li was born on April 20, 1987 in Xiantao of Hubei Province, China. He received his B.Sc. degree in chemistry from School of Chemistry & Chemical Engineering of Nanjing University, China, in 2009. Now he is a third year master major in machine learning and data mining at Department of Computer Science & Technology of Nanjing University, China.