Support Vector Machines in the Machine Learning Classifier for a Texas Hold’em Poker Bot

Jeremy Pfund  
Faculty Advisor: Dr. Mitch Marcus

Classifier Feature Selection

Generally speaking, the strength of a player’s situation in Texas Hold’em is a function of (1) player’s hand quality relative to the threat of better hands provided by the community cards, and (2) the strength of opposition measured by opponents’ actions in the hand. Based on these criteria, the following list constitutes the vector features utilized in the training of my bot: [hand rank, position, number of opponents, betting round, pot size, number of bets per round, number of opponents remaining per round, hole card strength for each hole card]. To determine the hand rank of a given hand, a brute-force, iterative calculation is performed in which for every possible pair of opponent hole cards coupled with the community cards, the best possible five-card hand is determined and compared with the rank of your own best possible five-card hand. This value is incredibly pertinent in that it is naturally normalized across all possible hands, indicating the strength of your hand versus all possible opponents. To determine the hole card strength, a pre-computed table is used. The table dimension is 13 x 13 x 2, representing respectively the rank of the first hole card, the rank of the second hole card, and whether the two are of the same suit or not. When traversing the training data, each cell in this table accumulates values for the following crucial information regarding all pairs of hole cards witnessed: number of times won with these hole cards, number of times lost without folding, and number of times folded. The value in collecting this information about all hole cards comes particularly into play to help the classifier learn whether it is holding a generally strong or generally weak starting hand.

Results

The bot successfully attaches to poker tables, retrieves all vital information in real time, performs the classification process to determine actions, and performs these actions in real time. The performance of the game-play strategy system has proven to be quite impressive. I have run the bot at play-money tables in real time over an extended period in order to collect the performance data displayed in the following figure:

<table>
<thead>
<tr>
<th># Hands</th>
<th>Voluntarily Put $ In The Pot</th>
<th>Won Money When SAW Flop</th>
<th>Avg Win per 100 Hands</th>
<th>Won $ at Showdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>2543</td>
<td>5.6%</td>
<td>17.01%</td>
<td>-3.27 Big Blinds</td>
<td>48.89%</td>
</tr>
</tbody>
</table>

The value of number of big bets won per 100 hands is the best indication for performance of a player. This is a normalized value to the stakes that the user is playing at and it serves to indicate, all other statistics aside, whether that player is consistently winning. According to this result, the bot is losing an average of 3.27 big bets per 100 hands played. Regardless of the fact that it is a losing player, it is quite impressive that this value is so close to zero. The statistic for whether the bot won money if it went all the way to a showdown—i.e. betting completed at the end of the river street and the player with the best hand wins—holds a value of nearly 50%. Since many of these showdowns result in more than two players, with a minimum of two players per showdown, the fact that this value far exceeds 1/4, where n is the number of players at the showdown, is a strong indication that the bot is not excessively bluffing nor excessively playing poor cards to showdowns. The value for how often the bot won money when it saw the flop is greater than 1/4, where n is the number of players playing at a table, indicating that starting hand selection is resulting in positive play, granting confidence in the pre-flop strategy adopted by this project.

Looking Forward: There are plenty of foreseeable improvements, so the decent performance to this point shows promise that it has potential to succeed at these full-ring tables pending future development. As the first stage of a research project, this puts us exactly where we want to be. Anticipated improvements include opponent modeling, reinforcement training and improved training data quality.

Artificial Intelligence Strategy

Pre-flop Strategy: Since the complexity of pre-flop hands is very low, it is justifiable to develop a strong pre-flop system manually. I have devised a simple pre-flop strategy of only playing an elite group of hole cards to the flop and playing them with varying levels of aggression according to strength and position. This provides the bot with a tight enough attitude to spur it into a winning strategy for pre-flop play at a 6-10 person table. I qualify the table size because naturally a player’s hand strength is inversely proportional to the number of opponents, and for sake of discussion, we presume for the remainder of this analysis that the bot will play only at highly populated tables.

Post-flop Strategy: Post-flop, the complexity of game-states increases significantly. This portion of the game is substantially more interesting and warrants machine learning techniques to generalize action strategies. The classification system developed for this bot is to train Support Vector Machines on a database of game states and corresponding actions. This involves identification of a set of important game state attributes, or features, which can sufficiently define all relevant information necessary to make a proper decision. In turn, a vector consisting of values for each of these specified features will constitute a single game state and is passed as input to the classifier. The hand history database utilized in the project is made available for public use by the University of Alberta and is known as Michael Maurer’s IRC poker database.

Testing Platform Implementation

The interaction of the bot with the PappyPoker.net application involves extensive windows programming. I utilize Microsoft Visual C++ coupled with the Microsoft Foundation Classes to program the application. My bot must grasp a handle of the window for a table that the user has joined. It needs to learn when a new hand is dealt out and to learn what two hole cards are dealt to the player. The bot must recognize player actions and discover when it is its turn to act. After deciding the proper action to take, it must grab a handle of the correct button on the table window and send a mouse-click window event to that button. The bot must continue on to recognize all community cards and further player actions on subsequent betting rounds. Finally, it must keep track of its stack size (the amount of money it has available at the table to bet) so that it may recognize when this value reaches excessively low levels and behave accordingly.

General Application Problem

Problem Realization in Poker

- imperfect knowledge: opponents’ hands are hidden
- multiple competing agents: many competing players
- risk management: betting strategies and their consequences
- agent modeling: identifying patterns in opponent’s play and explaining them
- deception: bluffing and varying style of play
- unreliable information: taking into account your opponents’ deceptive plays

Figure: Artificial intelligence problems and their corresponding place in poker

Figure: Flow diagram of bot interaction with poker table and classifier

Figure: The performance data of the bot on live play-money tables

Senior Project Poster Day 2007 Cis Dept. University of Pennsylvania