Date: 12/14/07

Project Members:
Elizabeth Lingg
Alec Go
Bharadwaj Srinivasan

Title: Machine Learning Applied to Texas Hold 'Em Poker

Introduction

Part I

For the first part of our project, we created a Texas Hold'Em poker player that utilizes machine learning techniques (such as Reinforcement learning and Supervised learning schemes). Poker, being a game of incomplete information as opposed to chess or checkers, is a more interesting domain to apply machine learning techniques due to the large amount of uncertainty in the game. Opponent cards are unknown and players may bluff or try to play in unpredictable ways as part of their strategy.

Our goal here is that the player must play optimally (with the given information) in 1 on 1 poker games. We also added personality to the poker application, by allowing the player to bluff and behave differently based on a certain set of heuristics. The player's personality is an important characteristic to induce uniqueness and randomness in its play, so that it cannot be reverse-engineered and beaten trivially.

We investigated opponent modeling and strategy. Some of the variables that we analyzed include strength of hand, playing style of opponent, size of bets, stage of the game, bankroll and our aggression level.

We designed a game interface to play heads up (1-1) Texas Hold'Em poker, with the simplest set of rules (no 'ante', straddling and other complicated events), and created two automated players that learn how to play the game, both with different strategies and playing styles.

Part II

As a second part of our project, we made our game into a Facebook application and opened it up to Facebook users. We collected data from the games that users played against our bot and analyzed the data to find very useful and interesting results about the players, which our bots could capitalize on, to play against players with similar playing styles.

In this paper we outline our techniques and state our results and findings.

The Learning Algorithm

We employed reinforcement learning, with a goal of maximizing the expected value. Our bot learns to play better by repeated training against itself.

We considered game strategy at each step of the poker game. For example, one strategy in poker is to only take the first two cards or "Pre-flop" into consideration.

Our model consisted of states, actions, and rewards. Our states were the different rounds and the sets of all possible outcomes in those rounds (table cards, player cards and pot-sizes):
1. "Pre-flop" (where each player is assigned two cards)
2. "Flop" (where 3 cards are drawn to the table)
3. "Turn" (where another card is drawn to the table)
4. "River" (where the final card is drawn to the table)

Our actions at each state were: bet (continuous action with varying bet amount), fold (discrete), and check (discrete).

Our rewards were calculated based on the expected value of winnings.

Expected value = P(Win| State) * Amount in Pot - (1 - P(Win | State)) * Amount Wagered

Typically we use a Q-learning style algorithm, where we back-propagate rewards (dollar amounts won/lost with respect to the Bankroll and amount wagered) using a fade-out for each step backward, across all states of the game that contributed to that outcome.

The probability of winning with a given hand was updated after each round of game play. In our tests shown below, we had 100,000 rounds in which our player played another instance of itself. Figure 1 (Appendix) shows the results, in which we graphed the strength of hand vs. the probability of winning with that hand. On the X-axis, the hands are ordered from left to right, beginning with a "high card" (the worst hand), then a pair, through to a "royal flush" (the best hand). We also took into account the card values in this ordering. For example, a pair of 3's is lower than a pair of 8's. It is very important to note that the program did not know these probabilities prior to the application of the learning algorithm.

Figure 1 (Appendix) shows clear correlation learned by the bot, between the strength of the hand and the probability of it winning the hand. The erratic dips towards the end of the curve are due to the fact that those high strength hands were relatively rarer occurrences and the updates did not lead to a convergence within the tested time-span. Training with more rounds would resolve this artifact. One interesting finding is that the probability of winning closely resembles a logarithmic function in shape. This is reasonable because after a certain threshold, it is highly likely that the player will win.

We had the bot relearn the probabilities of winning over 100,000 rounds, this time only considering hand type, i.e. "pair" or "straight", as opposed to all possible hand combinations. Figure 2 (Appendix) shows the strength of hand versus the probability of winning at each stage of the game.

Here, we notice how the correlation between hand strength learned varies with respect to the stage of the game (as propagated backwards). Obviously, then hand strength cannot be higher than a 'pair' pre-flop. We see how the correlation is lower at the river than at the flop, consistently, which shows that players do not usually go until the river unless they have quite a good hand. We have tackled this by adjusting out betting amounts and probabilities depending upon the stage of the game, as explained elsewhere in the paper.

We also tried an alternative where we include a combination of learning and direct computation of hand strength in our algorithm. For the first three stages of the game, we learn the probabilities of given hands as before. For the last two stages of the game, when we have more information, we compute the exact probabilities of winning. This is done by evaluating all possible two-card opponent hands, given the table cards and our hand, and evaluating the number of cases where we would lose (by simulating future table cards).

Figure 3 (Appendix) shows the strength of hand versus the probability of winning in this scenario at the final stage of the game. Although the graph is only slightly different, computing exact probabilities in the final stages of the game improved game performance significantly. Before combining learning and heuristics, the bot's average bankroll over 88 games was 742.04 against human players. After adding the probability calculations, the bot's average bank roll against the same two players was 996.83 out of 53 games. The typical reason for this is that it is the later stages of the game that are usually more important, and if we calculate exact probabilities in those stages, we have a more accurate estimate.

We represented several heuristics in the form of a utility function – variables such as opponent's bet, strength of opponent, and pot size were quantified with adjustable weights that were varied as the bot learned and played more games, and also depended upon its mode of play (strategy and aggression). For example, we learned the strength of opponent, based on percentage of wins and the size of bankroll over time and we learned aggressiveness based on the size of the opponents bets. We applied these findings to tweak the bot’s strategy. For instance, when playing against a strong player, the bot plays more conservatively by betting heavy only when it has a monster hand.

In order to determine the parameters for the utility function which when to fold, when to check, and when to bet, we played the bot against another instance of itself for about 100,000 games. If both

instances of the bot ended with about the same bankroll, then we knew the bot was playing consistently and the parameters were working well. If not, we modified the parameters.

As another step in our algorithm, we aim to teach the bot complicated playing and betting strategies which would be hard to learn by playing itself. Here we play the bot against human players which follow different strategies, such as slow-playing good hands and folding bad ones and the bot learns to mimic this kind of behavior. We assign high weights to what the bot learns during this stage since we have a lesser number of examples and also manually change some of the parameters. The bot decides the appropriate strategy to employ based on a whole range of parameters, which is what makes it hard to decipher the bot's strategy and counter it.

Betting & Strategy

An important aspect of creating a poker bot is to give it a 'personality'. The bot must carry its strategy and behave in a non-deterministic way so that it cannot be reverse-engineered and beaten easily. Our algorithm has the bot pick from different levels of aggression/strategies and also by teaching it to bluff, slow-play, etc.

Typically, our player uses Kelly's betting criteria to decide on optimal betting amounts, given the bankroll, pot size and its estimated probability of winning (strength of hand, as explained above). Then it factors in a variable for aggression and bluff/straight play, etc and depending upon the stage of the game, varies its actions. An example of this is betting on a weak hand pre-flop (a bluff) to intimidate the opponent into folding, checking, or calling on a flush which was hit on the flop until the river, again to fool the opponent and maximize gain on that hand. Simple rules like 'Never fold when you can check' are trivially learned by realizing that checking in that case always has an expected value which is higher or equal to folding.

Applying Kelly's criteria, we have an expression for the optimal amount to wager, given the pot size, bankroll and estimated probability of winning, which is: $b = B*p*B / (P + B - p*B)$, where b = optimal bet, B = player's bankroll, P = pot size and p = estimated probability of winning the hand. The probability of winning at any given stage is calculated by computing all possible 2-card opponent hands (Typically 1225 hands pre-flop, 1081 hands at the flop, etc) and calculating how many of those hands 'beat' our hand given the current table cards. That gives us a sense of how many hands can beat us, and thereby a probability measure. This measure is then adjusted, as mentioned previously, based on our risk-aversion and estimate of our opponent's playing style (frequent bluffer or not, etc). Also, we normalize this probability at each stage of game-play, depending on previous actions from the opponent. For example, if we compute a 0.9 probability of winning at the turn, but have seen that our opponent never bluffs, and has been betting very high the last few rounds and continues to bet high on the turn, we adjust our probability of winning *downwards*, accordingly to take that information into effect.

We thus compute the optimal bet amounts as suggested by Kelly's criterion and use that with varying weights (depending on our strategy, performance and past history) to decide upon the best action. We back-propagate our wins and losses along with the loss amounts to tweak the expected value of any particular state. That becomes a parameter for the state's utility function which takes into account other variables such as hand strength and computes probabilities for each action.
For example: For a particular state, the resultant probabilities can be as follows:

Fold	Call	Bet
10%	85%	5%

The actions are then picked based on these computed probabilities.

Figure 4 (Appendix) shows the results of the player's performance over 100000 simulated games. We plotted the average optimal bet amounts for different pot-sizes, averaged across all playing strategies and stages of the game. Figure 4 shows us the trivial result that 'Return is proportional to Risk or amount wagered' and the bot has learned to bet higher if more money is at stake in the pot. More importantly, we see how this amount increases with a higher estimated probability of winning as expected.

Figure 5 (Appendix) shows the bot's optimum bet amounts for a pot-size of $1000, for its different playing strategies. We notice how the amount wagered increases as the strategy gets more aggresive.

Interesting findings here were:
1. In the aggresive mode, the bot wagers a full $1000 for a P(Win) >= 0.806, as seen from the horizontal green line at $1000. This is known to be a good strategy in poker - when you are playing aggresively, an 80% chance is more than enough to go 'all-in' for any pot when the same amount is at stake.
2. While bluffing, the player bets a whole range of randomly oscillating dollar amounts, primarily to confuse the opponent. We still see an increasing trend with respect to the probability of winning - This randomness is what makes the player hard to reverse-engineer or crack.
3. Notice how in 'slow-playing mode', the player bets significantly lower amounts on average than in other modes, for the same estimated chance of winning. This is a sign that it has learned to slow-play consistently. In slow-playing, the idea is to bet lower amounts until the end, to fool the opponent into thinking that we don't have a very strong hand and extract as much money from the opponent as possible.

Figure 6 (Appendix) shows the average amounts wagered by the bot at the different stages of the game. The highlight of this graph is that it shows how effective slow-playing is, in the last column of bars ('slow-play' mode). We notice that while slow-playing, the bot wagers low amounts pre-flop, at the flop and at the turn but bets heavy on the river, essentially executing the slow-play strategy perfectly.

Facebook Integration and Data Analysis

We also tested the bot against humans on a web and Facebook interface. We performed analysis on these games, and modified our parameters based on the bot's performance. For example, in one of the early iterations of our bot, we saw that the bot would lose because the risk level of a human player was very high. Human poker players may be more risky when playing online poker compared to real poker because real money is not at stake. After this discovery, we tuned a parameter to make our bot more aggressive which helped mitigate the effects of a risky player. Playing against human players also allowed us to update the values in reinforcement learning.

As a sanity check, we played the online Bayesian Poker Player (BPP) for fifty games. We found that our player was able to outperform BPP by a net amount of $350. Our player behaved more conservatively and would fold if it did not have a good hand in order to minimize its losses. However, BPP would often bluff and would lose by a significant amount if our player had a better hand. On loosing rounds, our player would lose less money than BPP, and on winning rounds, our player would win more money, due to its conservative strategy. In order to determine statistical significance, we applied Pearson's chi-squared test and found the corresponding p-value. The formula for Pearson's chi-squared test is the summation, from i=1 to 50, of (Oi-Ei)^2/ Ei. Our observed values, Oi, for each round, were the amounts won and lost. Our expected values, Ei, for each round, were the average amount won, 5.1. This gave us a p-value of .00. Because our p-value was less than .05, we were able to reject the null hypothesis that the results happened by chance. Figure 7 (Appendix) shows the results of the fifty matches.

We integrated our application into Facebook, in order to classify user player styles. With the Facebook API, and we turned our machine learning poker application into a Facebook Application. Our goal was to classify users based on response time, level of risk tolerance, strength of play, and bluffing. Specifically, we used K-Means to cluster these feature vectors. As a sanity check, we had one player play under two different identities. Both "identities" ended up in the same cluster. Table 1 (Appendix) shows the results of applying K-Means with k=4.

The interesting result is that four clusters correspond roughly to very good players, good players, bad pl ayers, and very bad players. This clustering was also able to identify outliers. For example, one of the clusters consists of one player, who had an average bankroll of $476 after each game, and bet on avera ge $162.

We also performed analysis to see if we could associate more qualitative attributes of players to their playing style. Attributes include school, sex, age, and degree. We applied the apriori algorithm to learn association rules between these attributes and their cluster. Unfortunately, we could not find any associations between these attributes and their cluster.

We also used linear regression on different attributes of players. We found that the response time is correlated with the player's probability of winning. For example, a player will take a longer time to respond if they have a full house as opposed to one pair. Figure 9 (Appendix) shows this correlation.

This shows a result observed across online poker games, which is the fact that players usually take a longer time to place a bet when they have a good hand. This can be used as a utility function, thereby guessing that if a player takes longer to place a bet, then he/she probably has a better hand.

Conclusions

We made several interesting conclusions:

Broadly, we learned that machine learning techniques, particularly reinforcement learning is effective in tackling partial information games such as poker and in many cases are able to play better than humans, as they can calculate probabilities more effectively.

Adding a personality to the player, making it bluff, slow-play, etc gave the player the necessary randomness and variance in strategy to make it hard to understand and beat.

The player learned the correlation between hand strength and probability of winning as a first step and then went on to choose the optimal action and bet the right amount, based on a whole range of parameters, particular to the game, its performance, and opponent.

From our analysis of the Facebook game data, we were able to cluster people into different categories based on their playing style, using several attributes such as their response time, aggression, and average bet. Our findings reinstated popular beliefs such as 'people take longer to bet with stronger hands'. We were also able to identify behavioral patterns in players.

Future Work

After gathering more data, we could use our Facebook findings more effectively to model the opponent in our player, considering features such as player demographics to better understand players' styles (For example, if we know that males under 25 are most aggressive players, we can use that fact to change the way we would play males under 25. Similarly, if we find that Stanford players are more risk-averse than Berkeley players, we can tweak our player accordingly).

Other frameworks that represent the incomplete information, such as Partially Observable Markov Decision Processes (POMDPs) or Hidden Markov Models, would help state representation and learning on those states. Supervised learning and regression models could be used to make predictions based on player data. Error minimization algorithms can be used to determine the significance of various attributes. Our player can be improved after additional testing and development of new strategies.

Appendix

Figure 1: Strength of Hand vs. Probability of Win

Figure 2: Strength of Hand vs. Probability of Win at each stage of the game

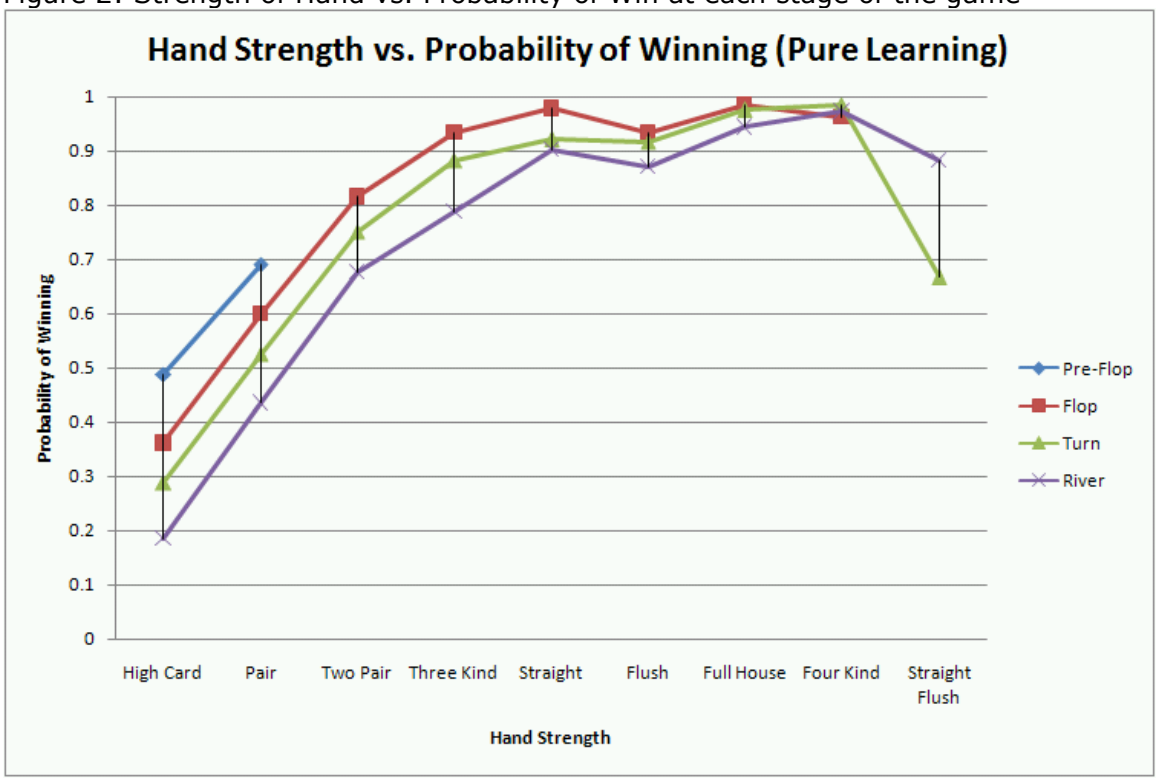

Figure 3: Strength of Hand vs. Probability of Win at River

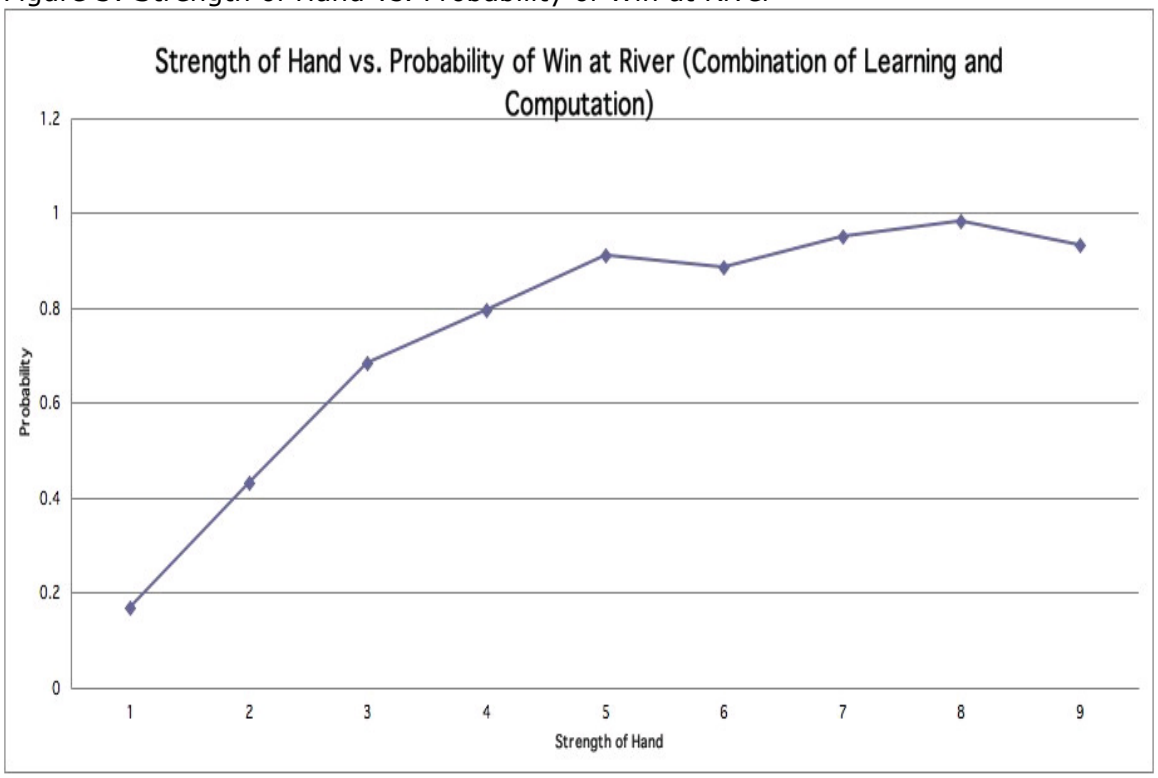

Figure 4: Average Optimal Bet Amounts vs. P(Win) for varying pot-sizes

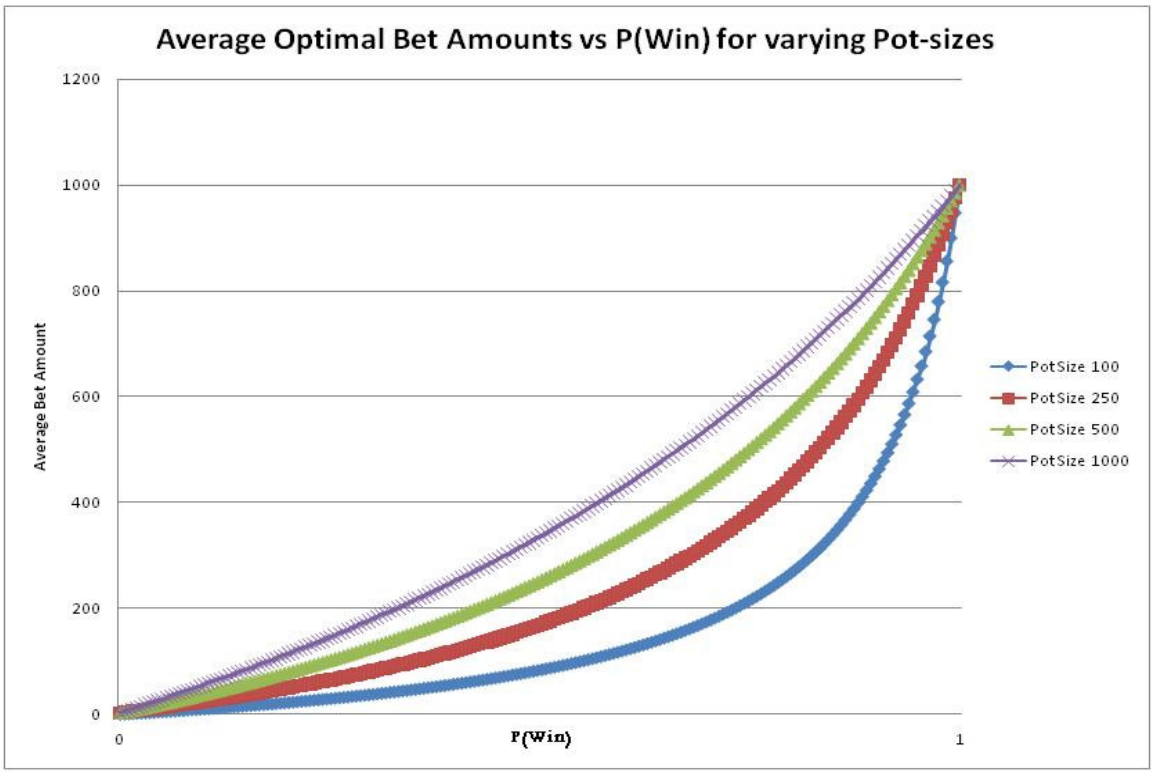

Figure 5: Average Bet Amount vs. P(Win) for varying play-modes

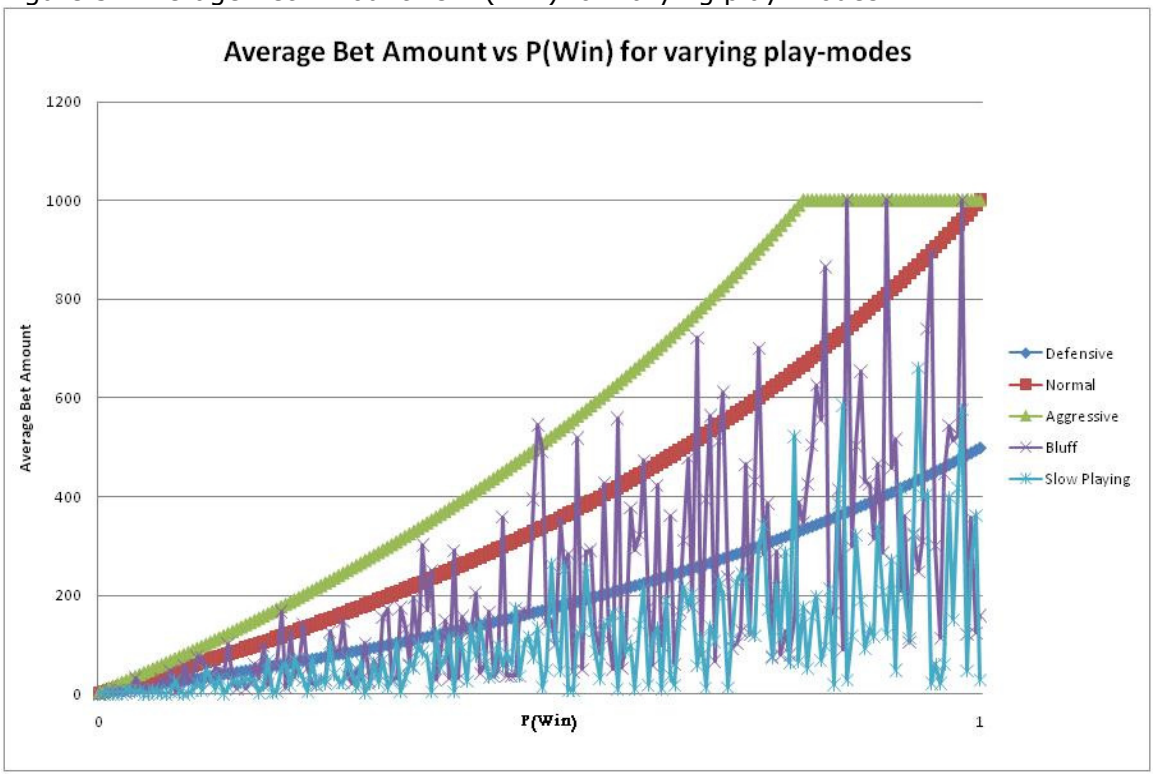

Figure 6: Average Bet Amounts by Stage of Game for Various Play Modes

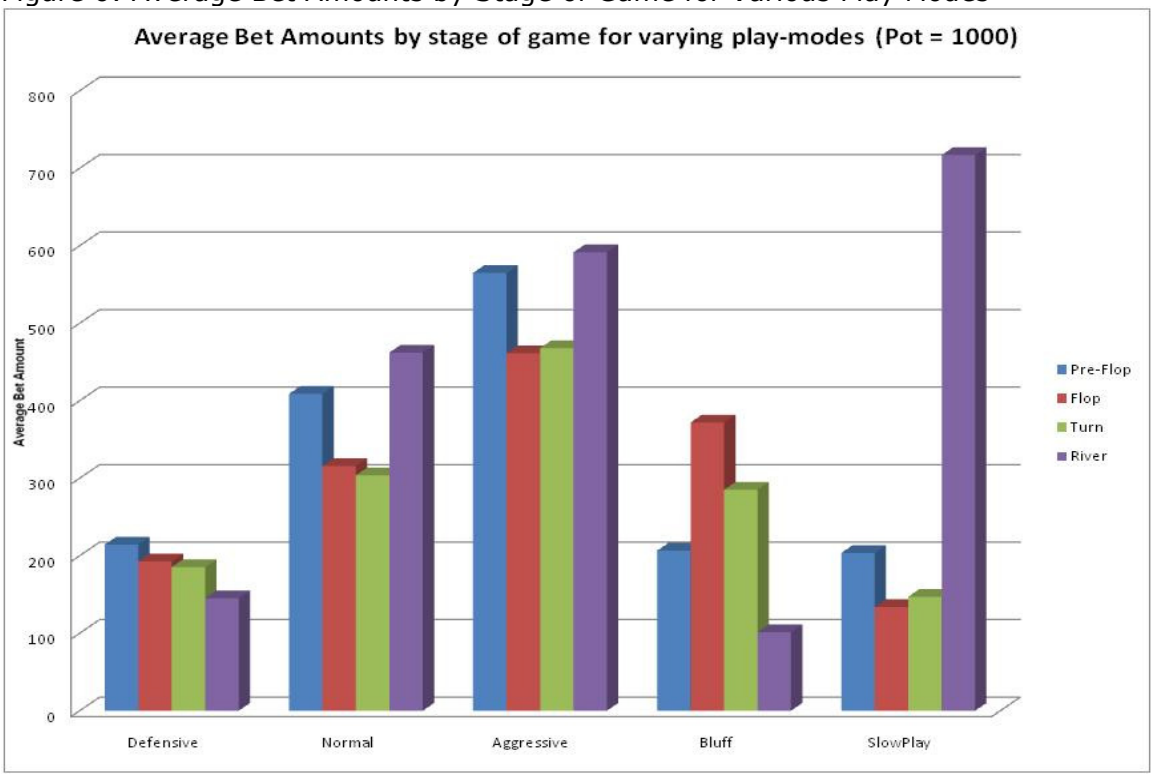

Figure 7: Performance Against Bayesian Poker Player

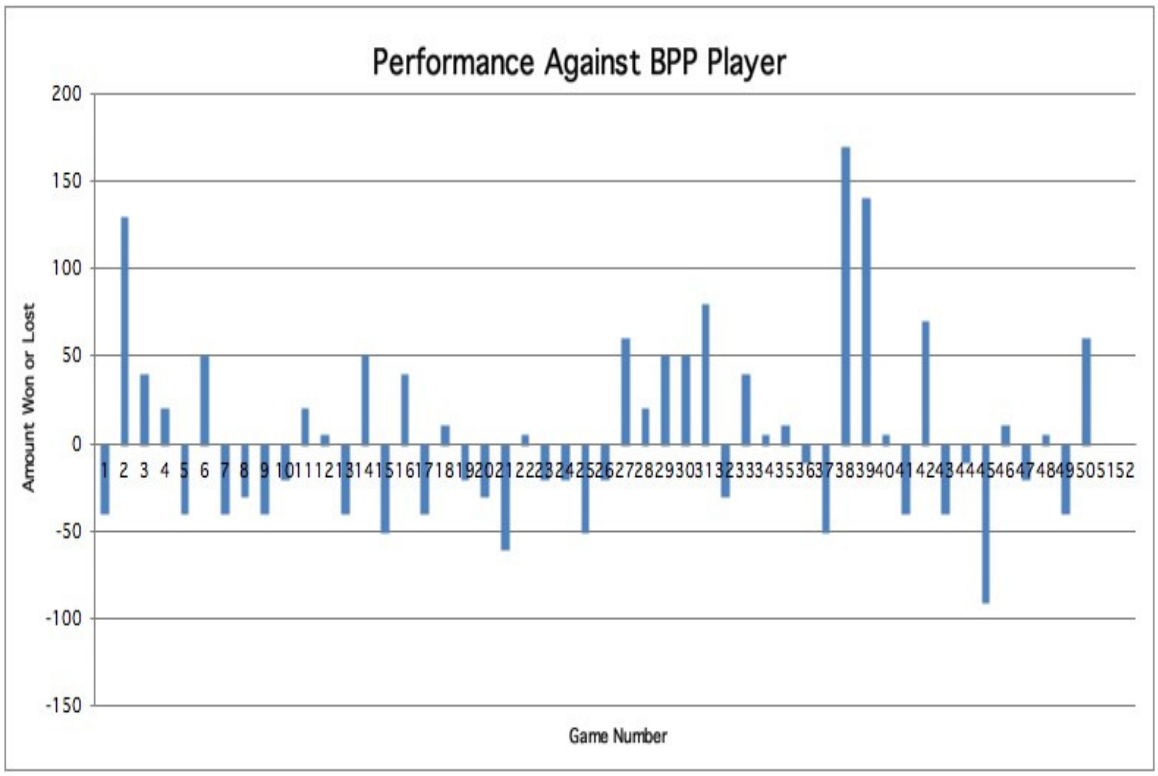

Figure 8: K-Means graphs for each cluster. The x-axis correlates with the 7 features mentioned in Table 1.

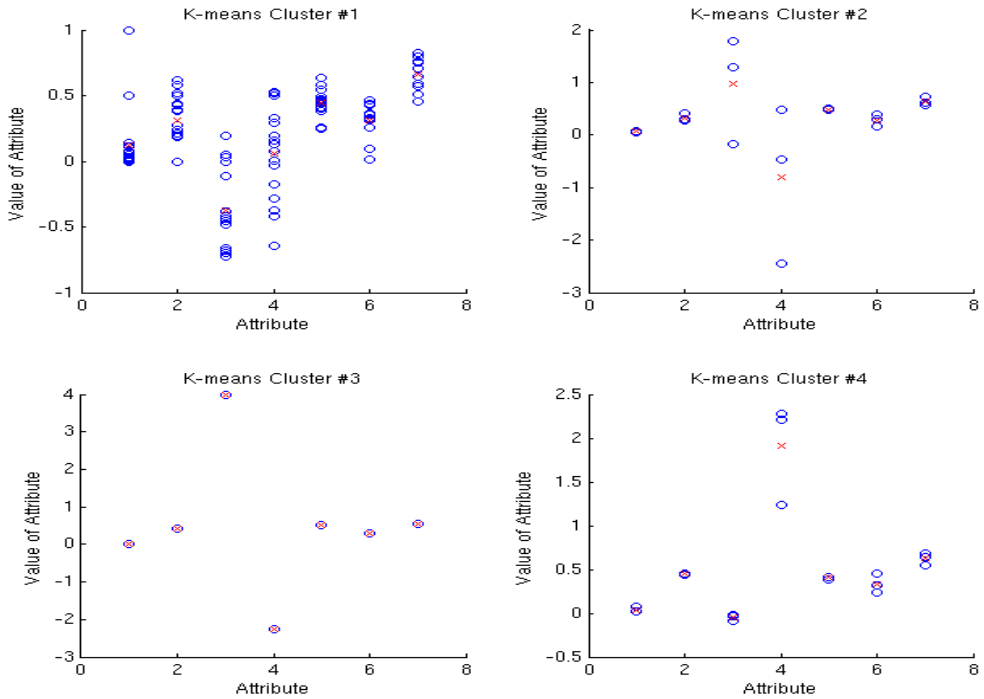

Figure 9: Probability of Winning vs. Response Time for a Human Player

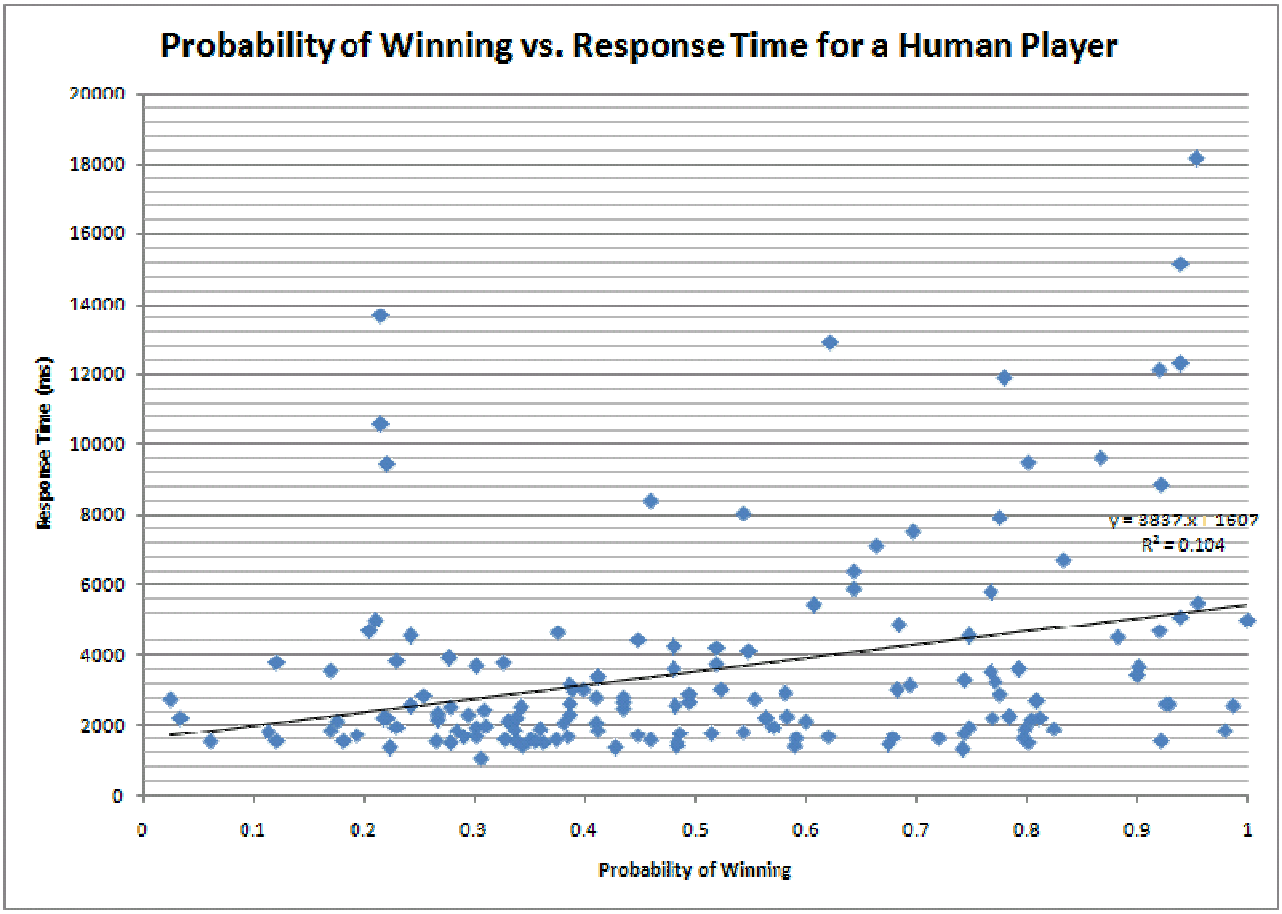

Table 1: K-means applied to playing style of all players

Number of people in cluster	% Hands Folded	% Hands Bet	Average Bet Size ($ Normalized)	Average Bankroll ($ Normalized)	Probability of Win and Checked	Probability of Win and Folded	Probability of Win and Bet
3	0.0364	0.4465	-0.0498	1.9082	0.4075	0.3365	0.625
18	0.1193	0.3107	-0.375	0.0616	0.4444	0.3129	0.6581
3	0.0697	0.328	0.9683	-0.8146	0.4838	0.2817	0.6368
1	0	0.4167	3.9958	-2.266	0.5061	0.31	0.5453
Average	0.098648	0.333323	0.01	0.084943	0.44717	0.311876	0.647037