POKER BOT PROJECT

JOSEPH NASSER

1. Preliminaries

Introduction

This project took place in the Spring of 2014 during a special topics course on Probabilisic Graphical Models. The project involved implementing algorithms to play limit Texas Hold'em Poker. Students were provided with a poker engine which took care of most of the software engineering needed to play computerized poker. The focus was thus on the algorithms necessary to make betting decisions. Students were responsible for providing functions which at each stage of the game would determine whether to call, raise, or fold. Note that this is limit poker, which means that bet sizes are standardized. Thus the algorithms only had to determine whether or not to bet, not how much to bet. The complete game history (includes full betting history for each player, and hole cards revealed in the showdon) is kept by the game engine and is available to use in decision making algorithms. At the end of the semester, student algorithms competed against each other in a computer poker tournament.

Vocabulary

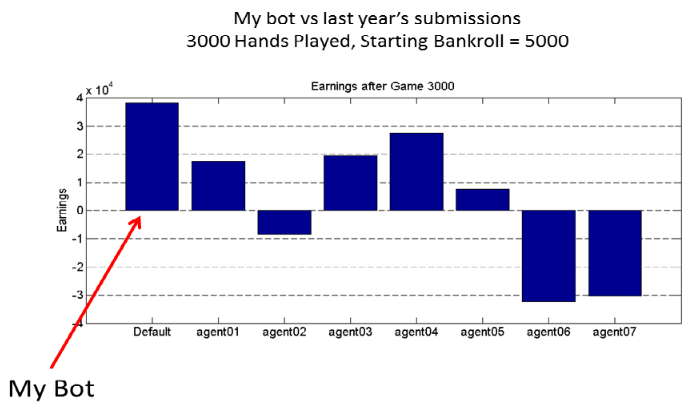
'Agent' refers to the player whose decisions my algorithms are controlling.

Strategy, Score and Results

The strategy I employed was simple but powerful:

1. Calculate the probability that the agent wins the hand as accurately as possible.
2. Use the win probability and the pot-odds to make betting decisions.

Unfortunately the in class computer poker tournament only consisted of 400 hands being played. This is not enough time to fully distinguish good implementations. From a starting bankroll of $10,000, the first place implementation finished with approximately $15,000. My implementation came in third (out of 21) with approximately $14,000. Students were also provided with the top implementations from 2013 to test the performance of their algorithms. I was able to run a longer scale tournament consisting of my algorithms against seven of the best from the previous year's implementations.



It's clear that my algorithms perform well in practice. Yet, as will be described in more detail throughout this document, many model parameters are chosen heuristically and are not optimized. This provides evidence to the general effectiveness of my modelling framework. Thus the algorithms described in this document should be viewed as proof-of-concept as opposed to a finished product.

2. The probability of winning a hand

Assume we are playing against only one opponent. Consider the case of calculating the probability of winning the hand after the flop and the turn. There are nine hand categories in poker. Ordered by increasing strength they are: Junk / No Pair, Pair, Two Pair, Three of a kind, Straight, Flush, Full House, Four of a kind, Straight Flush. Let $A_i, i = 1, \dots, 9$ be the event that the agent will have hand category i if the hand gets to the showdown. Let B_i be the event that the opponent will have hand category i in the showdown. Let W be the event that the agent wins the hand in the showdown against one opponent. Then we have:

$$P(W) = \sum_{i=1}^{9} P(W|A_i)P(A_i) \tag{1}$$

$$P(W|A_i) = P(B_1) + P(B_2) + \cdots + t_i P(B_i) \tag{2}$$

Here, t_i denotes the tie factor. $t_i = x$ implies that if both the agent and the opponent have hand category i in the showdown, the agent has probability x of winning the hand. Note that there are alternatives here. One could consider a separate win probability model in which the hand categories are more granular. For example, the pair category could be subdivided into pair of aces, pair of Kings... Or even on a further granular level: pair of aces with king high, pair of aces with queen high...

Equations (1) and (2) are crucial to the success of my algorithms. My entire strategy hinges on calculating $P(W)$ as accurately as possible. There are three components to consider: $P(A_i), P(B_i)$ and t_i.

In addition to the game engine, students were provided with some helper functions to assist in the development of the algorithms. The most useful of these was a function, hole2finalCat, which takes as input a players hole cards and the board cards. The function outputs the probability distribution over the nine final hand categories. For example, suppose the Agent's hole cards are ace of spades and ten of hearts, and the board cards are 3H, 8H, 10D. Then we have:

$$\text{hand2finalCat(AS,10H,3H,8H,10D)} = \begin{pmatrix} 0 \\ .58 \\ .29 \\ .06 \\ 0 \\ .04 \\ .02 \\ .001 \\ 0 \end{pmatrix}$$

This vector is interpreted as: if the hand were to reach the showdown, the Agent would have 0 probability of having no pair, .58 probability of having a pair, .29 probability of having two pair... In essence this gives us $P(A)$ for free.

Estimating $P(B)$
With the function hole2finalCat in hand (and knowing the board cards) we have

$$P(B) = \sum_j \text{hole2finalCat}(H_j)P(H_j)$$

The sum is over all $\binom{52}{2}$ possible hole card combinations, and H_j is the event that the opponent has hole cards combination j. Note that some of these combinations are impossible (eg, the board cards and the agent's hole cards are known). This will be dealt with soon. Now we need to estimate $P(H_j)$. Recall that we have access to all hole cards that have been revealed in the showdown. With this data in hand, we build a simple histogram distribution over $P(H)$. The construction begins with an initial distribution over $P(H)$, call this H^0. We then have

$$P(H_j) = (1-\alpha)H_j^0 + \alpha\frac{\text{number of observations of hand j}}{\text{number of observed hands}} \tag{3}$$

where $\alpha = \min(1, \frac{\text{number of observed hands}}{M})$. In the first hand of the game, we have no information about our opponent and $P(H) = H^0$. After observing M hands, $P(H)$ is just the empirical observed distribution. If we have observed less than M hands we interpolate linearly between the empirical distribution and H^0.

As board cards are revealed during the course of the hand (and the agent's hole cards are known), we can rule out certain hole card combinations for our opponent. The corresponding entries in $P(H)$ are zero'd out and the probability distribution is normalized.

Continuing the previous example, suppose that the agent has AS, 10H and flop is 3H, 8H, 10D. Assuming a uniform distribution over all possible hole cards for the opponent we have

$$\text{P(B)} = \begin{pmatrix} .27 \\ .46 \\ .16 \\ .04 \\ .03 \\ .03 \\ .01 \\ .001 \\ 0 \end{pmatrix}$$

Calculating the tie factor and using equations (1) and (2) we would arrive at an estimated win probability of .78.

This framework to estimate $P(H)$ does have some drawbacks. One of the more obvious ones is that it is slow to react to changes in opponent strategy. Another is that it is sensitive to choosing M and H^0 properly. An interesting addition to this framework would be to incorporate preflop betting patterns into estimating $P(H)$. For example, we could try to see if raising vs calling preflop is an indicator of the hole cards our opponents have.

Estimating the tie factor
In practice most of the final hand category probability is concentrated around junk, pair and two pair. Having a good estimate of t_i is important. The way t_i is calculated will vary with i. Here is an example calculation for a pair. Assume that the agent has a pair, and there is no pair on the board. We will consider the value of this pair to be based on three factors:

1. The rank of the pair relative to the board cards
2. The value of the pair
3. The kicker in the case of a non-pocket pair

And we have the estimate:

$$t_2 = \alpha_1 \frac{\text{Rank of Pair relative to Board}}{\text{Number of cards on Board}} + \alpha_2 \frac{\text{Value of Pair}}{14} + (1 - \alpha_1 - \alpha_2) \frac{\text{Kicker Value}}{14}$$

where the $\alpha' s$ are weighting parameters. Consider the following specific example. The Agent's hole cards are Queen and Jack and the board cards are Ace, Queen, Four. Then the rank of the pair relative to the board is 2, the value of the pair is 12, and the kicker value is 11. Using $\alpha_1 = .6$ and $\alpha_2 = .3$ results in $t_2 \approx .73$.

There are two questions to ask in evaluating this approach. Is the three factors model effective? If so, how do we choose the $\alpha' s$? Based on some basic poker strategy it seems like this model is a reasonable way to asses the strength of a pair. Choosing the weighting parameters is an interesting problem. In the course of my project, I did not have time to consider this too carefully, and chose the weighting parameters using some heuristic ideas. There is considerable room for improvement here.

The probability of winning on the river
After the river, the agent's final hand is known, which allows for some simplifications. We have

$$P(W_{\text{river}}) = \sum_j W_j P(H_j)$$

where the sum is over all $\binom{52}{2}$ possible hole card combinations for the opponent. W_j equals 1 if the agent wins against hole card combination j and 0 otherwise. $P(H_j)$ is the probability the opponent has hole card combination j as calculated from equation (3).

Evaluating the effectiveness of the win probability algorithm
A simulation is used to evaluate the performance of the win probability algorithm. Suppose n hands with two players and random hole cards are simulated. For each hand the win probability and result of the hand are noted. Thus we have vectors $x \in [0,1]^n$ which represents the win probabilities and $y \in \{0,1\}^n$ which denotes the hand outcome (1 is a win for the agent and 0 is a loss). How well does x predict y? For this project we will measure the accuracy of the win prediction algorithm by considering the averaged L1 distance between x and y.

$$d(x,y) := \frac{1}{n}\sum_{k=1}^{n} |x_k - y_k| \tag{4}$$

This measure has the hueristic interpretation as the average inaccuracy in the prediction. It also has some interesting theoretical properties. For example, even if the prediction algorithm was fully accurate, we would still expect $d(x,y)$ to be positive. In fact, we can calculate this expectation exactly and thus get a lower bound on how well we can expect to do in practice.

Suppose $X_i, i = 1,\ldots,n$ are a sequence of independant (but not identically distributed) Bernoulli random variables with success probability p_i. Then we can calculate the expected value

$$E_{X,p\sim f}\, d(X,p)$$

where we assume the p_i's are iid with probability distribution f. Under this assumption we can calculate

$$\begin{aligned}
E_{X,p}(d(X,p)) &= E_{X,p\sim f}\left(\frac{1}{n}\sum_{i=1}^{n}|X_i - p_i|\right)\\
&= \frac{1}{n}\sum_{i=1}^{n} E_{X,p\sim f}(|X_i - p_i|)\\
&= \frac{2}{n}\sum_{i=1}^{n} E_{p\sim f}(p_i(1-p_i))\\
&= 2(E_{p\sim f}(p - p^2))\\
&= 2\Big(E_{p\sim f}(p) - (E_{p\sim f}(p))^2 - Var_{p\sim f}(p)\Big) \qquad (5)
\end{aligned}$$

What is the correct choice for f? After the river f should be (the discrete analog) of a uniform distribution on [0,1]. Once the board is complete we have a natural order on the resulting final poker hands of length $\binom{52-5}{2}$ indexed by the the possible hole card combinations. When the hole cards of the agent are known, the probability the agent wins is the number of hole card combinations below the Agent's hole cards in this ordering divided by $\binom{52-5}{2}$. This is precisely a uniform distribution.

It is more tricky to think about the proper choice of f on the flop and turn. Let's consider the case of the turn. By conditioning on the choices for the river card, we can see that the probability of winning on the turn is distributed as the averaged sum of discrete uniform random variables. But these random variables are not independant! If they were independant then the resulting distribution would be more 'hat like' which would increase the expected error in (5).

If f is taken to be the uniform on [0,1] then we have an expected lower error bound of $\frac{1}{3}$. If f is taken to be a hat like distribution such as

$$f(t) = \begin{cases} 4t, & \text{if } t \in [0, \frac{1}{2}] \\ 4 - 4t, & \text{if } t \in [\frac{1}{2}, 1] \end{cases}$$

we have an expected lower bound of .416. I expect the true lower bound for the turn to be somewhere in between.

10000 hands were simulated. For each hand the estimated win probability was calculated after the flop, turn and river. The results are

Stage	**Error**
Flop	.436
Turn	.407
River	.329

TABLE 1. The errors in the win probability calculation as measured by (4). Note that the win probability on the river should be close to the expected lower bound since it itself is calculated via a simulation.

Winning against multiple opponents
So far we have considered the probability of winning a hand against a single opponent. To extend this to multiple opponents we assume that winning against each opponent are independant events. So let W_k be the event that the agent wins against k opponents. Then we have

$$P(W_k) = \prod_{i=1}^{k} P(\text{Win against opponent i})$$

Note that the win probabilities will vary across opponents since we have different probability distributions over the hole cards.

3. Making betting decisions

Pre-flop decision making The general strategy pre-flop is to focus on playing premium hands. This condition can be relaxed if there are few opponents in the hand. We partition starting hands into five categories based on strength. For each category we initialize the probability of folding pre-flop. The main impetus for making pre-flop decisions probabilistically is that it would make it harder (compared to a deterministic approach) for the opponent's algorithms to learn the agent's playing style.

Category	Hands	Initial fold probability
1	AA, KK, QQ, AKs, AQs	0
2	JJ, 1010, AJs, KQs, AK	0
3	A10s, KJs, QJs, J10s, AQ	.20
4	99, 88, 77, 66, 55, AJ, A10, KQ, KJ, QJ, K10s, Q10s, J9s, 98s, 87s, 76s	.25
5	Everything else	.95

Table 2. The definition of the hand categories and initial fold probabilities. An 's' next to two cards means they are suited.

With this initialization we are likely to play hole cards falling into categories 1-4 and unlikely to play anything else. However, a reasonable poker strategy is to expand the number of hands the agent is willing to play if there are less players involved in the hand. Suppose there are N players at the table, NActive of which have not folded preflop. We then define a fold dampener function as follows:

$$\text{foldDampener(NActive)} = \frac{A}{1 + \exp(-(\text{NActive} - N/2))} - B$$

A and B are normalizing and scaling constants (dependant on N, but not NActive) chosen so that foldDampener(1) = 0 and foldDampener(N) = 1. The probability of folding preflop is then the initial fold probability multiplied by the dampener. Assuming $N = 6$, here are the values of foldDampener for NActive $= 1, \ldots, 6$.

NActive	Dampener
1	0
2	.18
3	.45
4	.73
5	.91
6	1

If the model decides not to fold, there is still the decision of whether to raise or call. A simple heuristic is employed which prefers to raise with hand categories 1 and 2, and call with hand categories 3-5. This heuristic is not discussed here and is probably not such an important factor in the full performance of the algorithms. An improvement to the pre-flop decision making process would be to make the hand categories themselves a function of N.

Post-flop decision making
Decision making post-flop is a function of two inputs: the win probability and the (implied) pot odds. This can be complicated (or impossible) to get exactly mathematically correct. To simplify matters we take the following approach: If we think the agent has the best hand, then raise. If we think the agent does not have the best hand, but the pot odds are good enough, then call. Otherwise fold.

The raise criteria is simple: if the win probability is greater than $\frac{1}{\text{Number of players in hand}}$ then raise. If we do not raise, we calculate a threshold value β such that if the win probability is greater than β we call, otherwise we fold (or check if it is an option). β is approximated as

$$\beta \approx \frac{\text{Amount agent expects to pay in this round}}{\text{Amount expected in pot at the end of this round}}$$

An important point to note is that the estimated win probability does not change within the round. Thus if the agent calls once, it will very likely always call within the round since the pot-odds tend to improve. So it is important to make the first call decision as accurately as possible. With this in mind we can give some more detail to the β calculation.

$$\beta \approx \frac{\text{Amount to call + (Betting Unit)*(Number of Raises left in the round)}}{\text{Pot size + Amount to call + (Betting Unit)*(\# of players in the hand)}}$$

where Betting Unit is the bet increment (recall that this is limit poker). It's difficult to comment on the effectiveness of this approach. The main way these thresholds were tested was by manually inspecting the win probability and call/raise thresholds for dozens of simulated hands. It's difficult to suggest more comprehensive testing. The call threshold will also depend on the playing/betting strategy of the other players, so it can get complicated. This is an area that requires more thought.

4. Conclusions

We present a framework for making betting decisions in a game of Limit Texas Hold'em poker. Although many model parameters are not optimized, the algorithms perform well in live tests against other implementations. This suggests that the general modelling framework is succesful. Future improvements could try to make use of opponent betting history and optimize several model parameters.