“I don’t play the odds, I play the man”

JJR Strategy Report Pokerbots 2013

Justin Martinez
Jeremy Rubin

Introductions

Our team name, JJR, is an acronym for two permutations “Justin and Jeremy’s Robot” and “Jeremy and Justin’s Robot” the names of our two team members who happen to share a first letter. As two inexperienced freshmen, we decided to put our energy into writing clean, extensible code with a well thought out strategy, although more advanced strategy was beyond our reach with our limited knowledge of Poker and machine learning. We decided to follow the poker philosophy of Harvey Specter from the popular TV series “Suits”: “I don’t play the odds, I play the man.” Here we will outline how we read our opponent to implement our poker strategy.

Check out our code here: https://github.com/JeremyRubin/pokerbots2013

Action Tracking

The following statistics were tracked for each board-card stage and button position. By tracking each stage separately, we were able to adapt our strategy to be more flexible to the current board conditions. We were still able to pull an aggregate sum of all types if necessary however. A key component of our statistic tracking was using a flat strategy for the first 20 hands - this lets us simply watch our opponent and gauge their behavior before we dive in for attack.

- Raises
- All ins
- Folds
- Calls
- Checks
- Three bets
- Continuation Bets

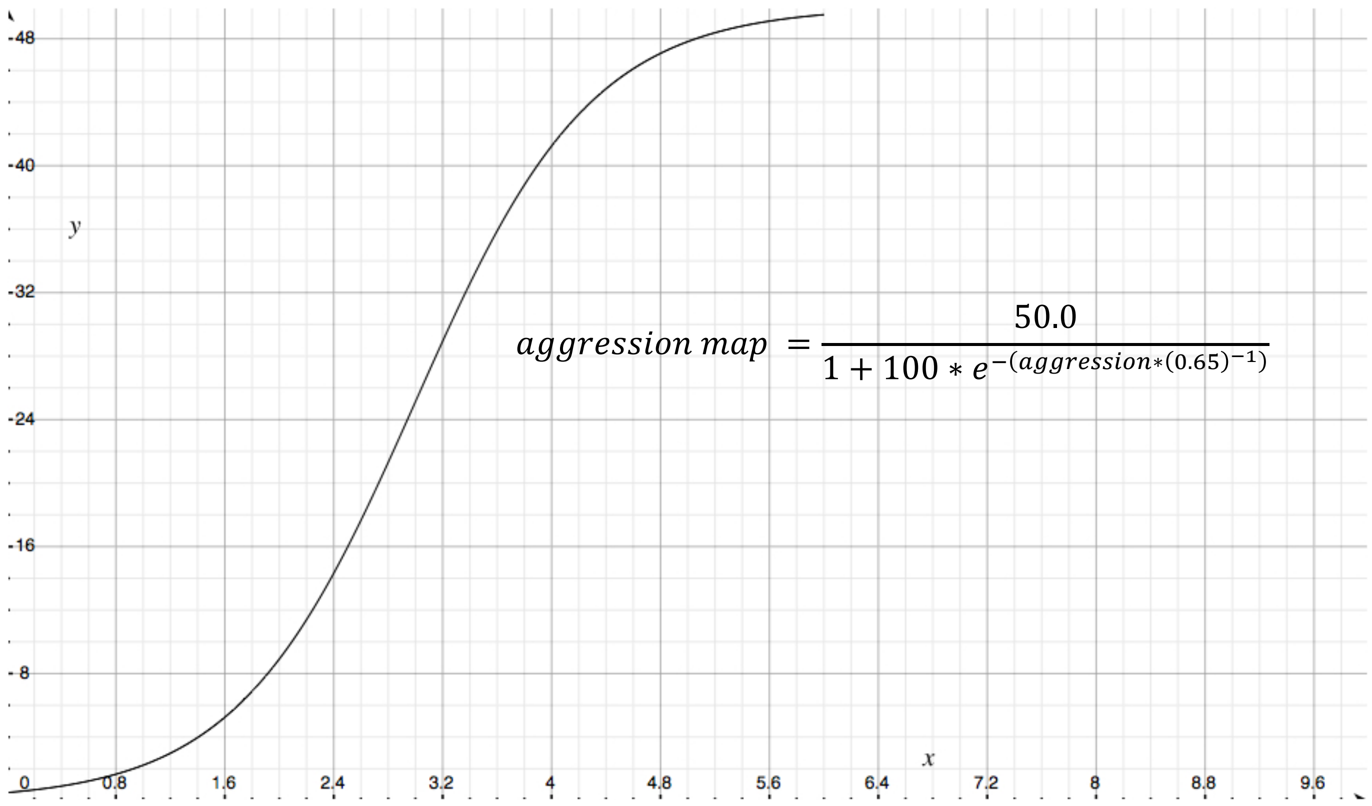
Statistical Analysis

$$aggression = 1.2 * raises + 1.5 * three\ bets + 2.0 * all\ ins + continuation\ bets * 1.2$$

- *aggression* is a weighted sum of the aggressive behaviors

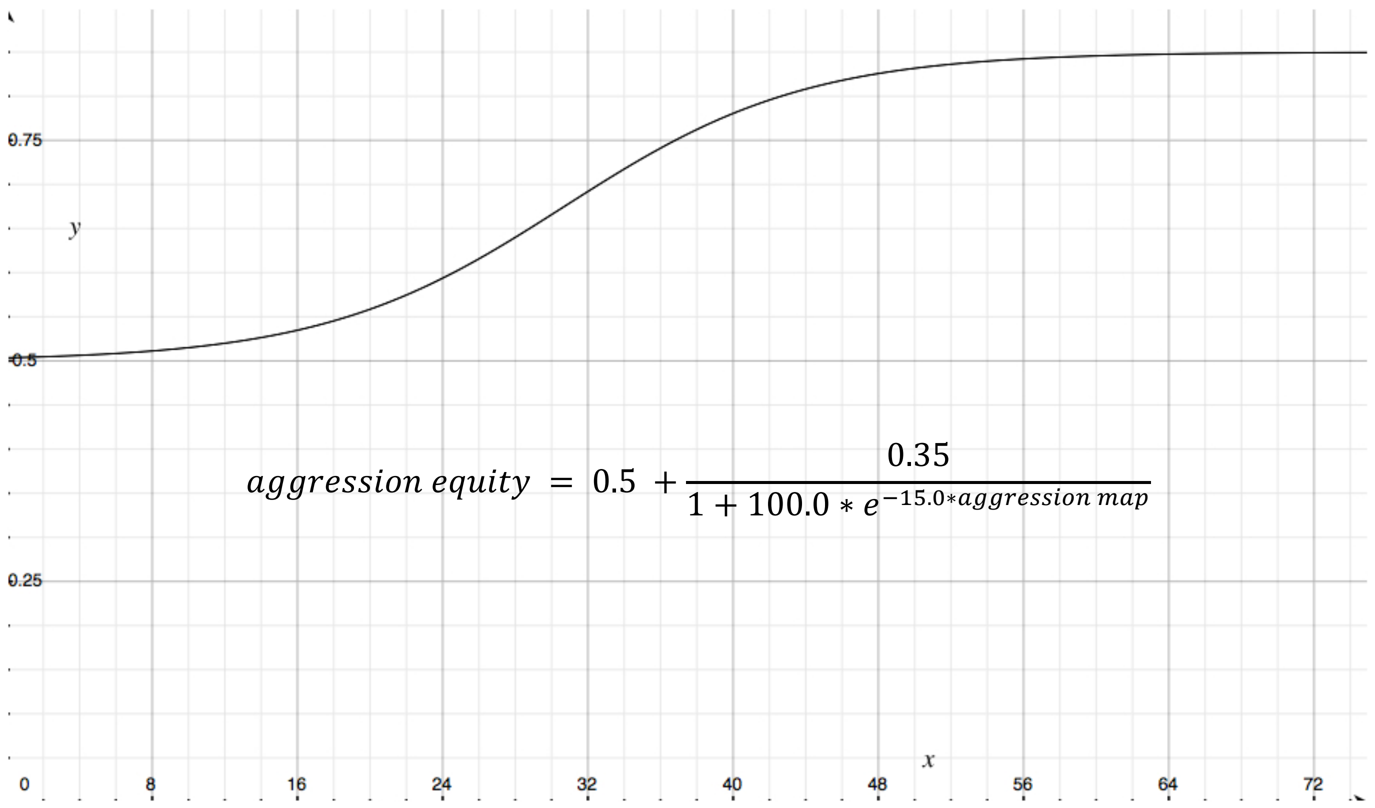
$$looseness = \frac{number\ of\ hands - number\ of\ opponent\ folds - number\ of\ our\ folds}{1 + number\ of\ hands - number\ of\ our\ folds}$$

- *looseness* is a coefficient determined by the total number of non-folded hands over (approximately) the number of hands not folded by us



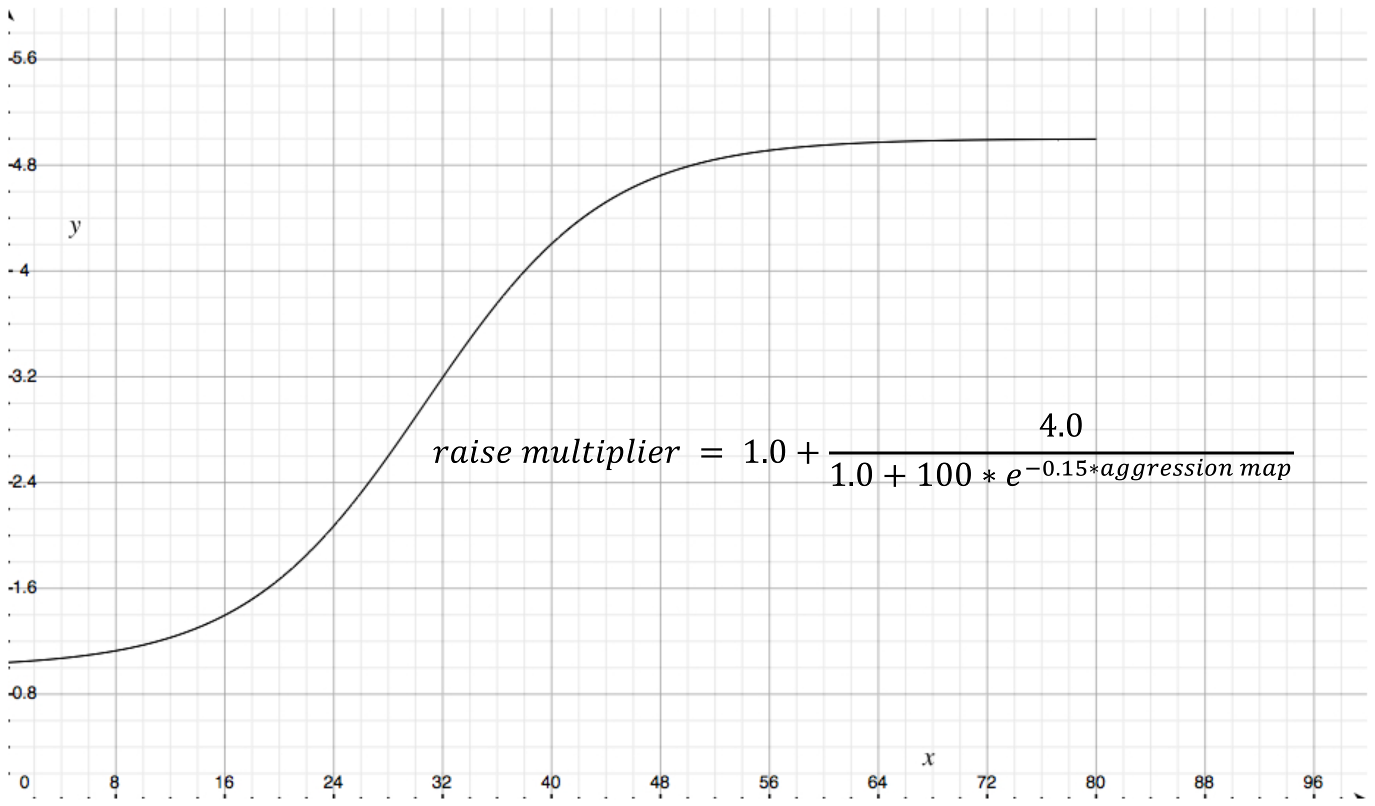
$$aggression\ map = \frac{50.0}{1 + 100 * e^{-(aggression * (0.65)^{-1})}}$$

- *aggression map* is a mapping function to adjust our sum to a range of 0 to 50



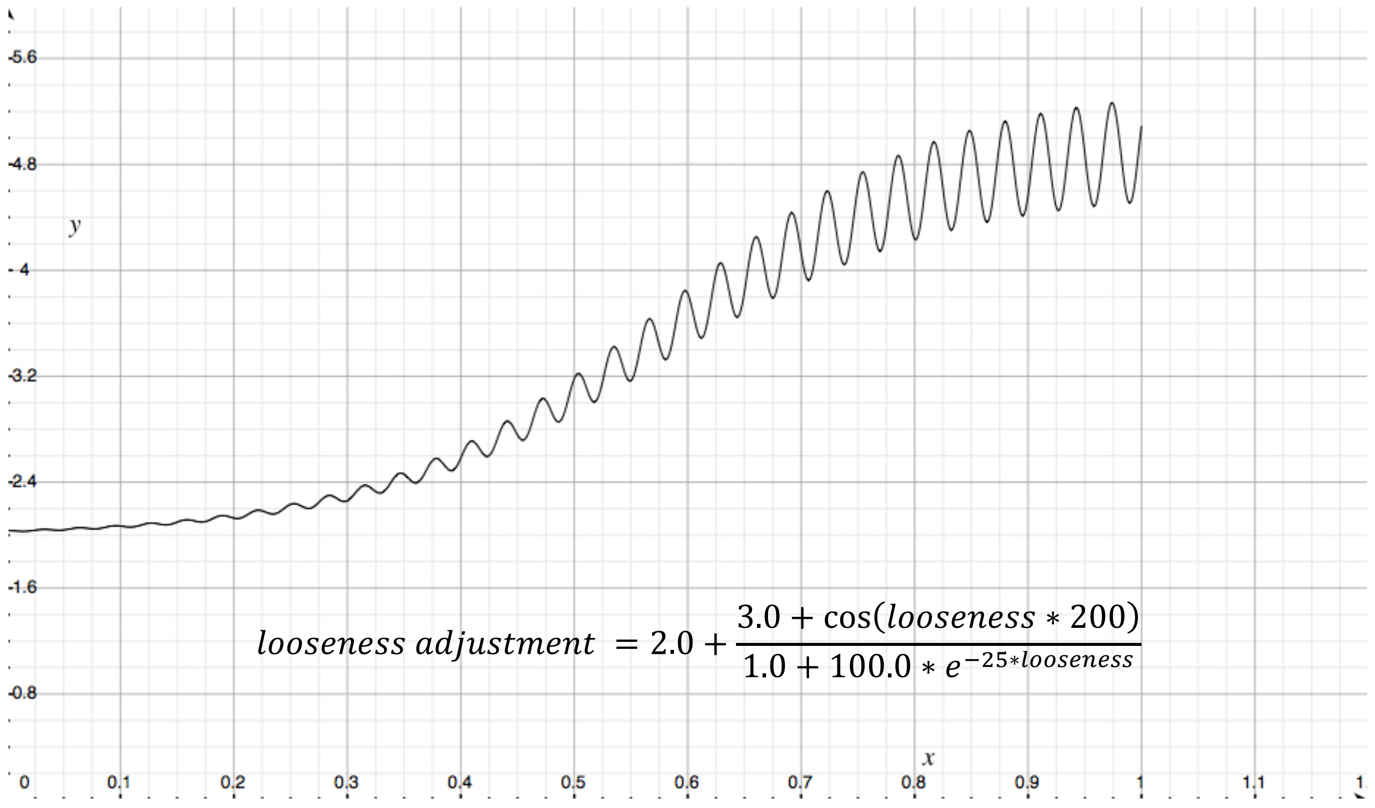
$$aggression\ equity = 0.5 + \frac{0.35}{1 + 100.0 * e^{-15.0 * aggression\ map}}$$

- *aggression equity* is an adjustment equity that will exist in the domain of .5 to .85



$$raise\ multiplier = 1.0 + \frac{4.0}{1.0 + 100 * e^{-0.15 * aggression\ map}}$$

- *raise multiplier* is a coefficient for how many times the minimum raise we would like to raise, between 1x to 5x



$$looseness\ adjustment\ = 2.0 + \frac{3.0 + \cos(looseness * 200)}{1.0 + 100.0 * e^{-25*looseness}}$$

- *looseness adjustment* is a function which lowers our equity based on the number of raises performed. This is an attempt to account for the concept that an opponent who raises will likely has a better hand. The use of a cosine is to attempt to give us a little bit more play in our strategy.

Action Determination

Using the above variables, we use the following conditionals to determine our actions

$$equity \ = \ chance\ our\ hand\ is\ superior\ to\ opponent's\ at\ various\ stages, \\ via\ monte\ carlo\ analysis$$

$$keep\ percentage \ = \ equity\ of\ hands\ cutoff$$

$$equity \ \geq \ 1.0 \ - \ keep\ percentage$$

If our equity is not worth pursuing at a stage, we will check or fold our hand.

$$0.5 \leq 0.5 * \left(aggression\ equity \ + real\ equity \ * \ \left(1.0 - \frac{number\ of\ raises}{looseness\ adjustment}\right)\right)$$

In order for us to raise, this condition must be true: the average of our equity and aggression must be greater than ½. If this is not met, our bot will check/call. As the number of raises increases, we will become less likely to raise according to our looseness adjustment.

Summary

We used statistics tracking to follow opponent's strategy and mold our aggression and looseness to counter theirs. We did not focus highly on attempting to calculate absolute odds - we found that after ten thousand rounds of Monte-Carlo, the percent error is relatively small and this certainly was not our performance bottleneck. We did not use huge look-up tables to determine our moves; we opted to use our opponent's strategy and the equity of our hand to determine our aggressiveness. A lookup table may have been useful to store some really good hands, but we preferred to develop a 'thinking' bot rather than a 'reading' one.

Reflections

As we reached the final week, our code really started to shape up into a pretty decently designed bot. Unfortunately, we missed out on the time to leverage the casino data to help us fine tune our strategy into a money-maker, so in the end it came down to intuition on what should be better. Justin had almost no programming experience at the start, but as we progressed (and with the help of the "Introduction to Python" class) he quickly picked up coding skills. Jeremy came in with quite a bit of coding experience, but left with a better understanding of the nuances of python and how to make code maintainable when collaborating. Given more time, we would have tested out more strategies, developed more functions to determine our player's looseness and aggressiveness, and built a handler to switch between these different types of responsive strategy.