Opponent Modelling in Poker

DME Project Report – s0954584

1. Abstract

Poker is increasingly becoming an area of interest in AI Research.
This is due to certain interesting qualities present in Poker that are absent from games more traditionally studied in AI, such as chess. The imperfect information and stochastic nature of poker are of particular interest to researchers, as these properties make the game's search space prohibitively large for classic AI techniques. The approaches developed to handle these issues promise to prove useful in a large range of practical applications.

Furthermore a fundamental aspect of poker is the necessity of handling the opponent's attempts at disinformation. This adds another layer of complexity and realism to the game, and is the aspect on which this work focuses.

2. Literature Review

There are two general classes of exploitative agents being researched at the moment, static and adaptive. Both of these approaches must incorporate an opponent model, which usually has a large impact on the overall effectiveness.[1]

The static agents rely on a pre-computed exploitative strategy that is stored and used when appropriate during play. The simplest form of this agent uses an opponent model called a "Frequentist best response" which works by analysing a large history of games, and storing the statistical best response at every encountered situation.[2,3] Since the number of different situations possible in poker is huge, the space is usually mapped to a lower dimensional equivalent. One example of such a lower dimensional space that has previously proved successful involves placing all starting poker hands into one of several buckets, each bucket corresponding with a certain probability of winning at showdown.[4,9].

During play the bot will map the current situation to the same lower dimensional space and play the pre-computed "best response" stored there. When presented with an opponent that plays a static strategy and a large database of this opponent's play history, the frequentist best response method will achieve close to maximal possible exploitation. However, this model is quite brittle and if the opponent's strategy changes even slightly the performance can drop drastically. [3]

A solution to this problem is given by mixing the above method with an e-Nash equilibrium bot by simply defining a value p such that the bot will assume the opponent will act according to his model with probability p and that he will act in some other, unknown, way with probability (1-p). The equilibrium strategy can be calculated for this new scenario, which will be a compromise between the pure exploitative and the pure equilibrium extremes.[2,3].

A final refinement to the above solution was explored in [5], where the restriction that the opponent have only one "p" variable is removed. Instead we estimate the probability that the opponent will or will not respect our model based on the current position in the game tree. This is a more natural approach, since we will have more confidence in our opponent model in some situations than others. Therefore, we can have different values of "p" depending on how many observations we have of the opponent acting in the current situation. This method is used in one of the current state of the art bots, "hyperborean", which placed 2nd in the poker competition.

The adaptive agents, rather than using equilibrium strategies, instead do a incomplete information game tree search. The opponent model has two functions here. One is the evaluation function which estimates the probability of winning from a certain node in the game tree. This function is consulted when the search has reached a leaf or a node of sufficient depth and gives the value used in the minmax-style algorithm. The other role of the model is to assign probabilities to the opponent's actions in every node in which the opponent must act. These values are used to

determine the likelihood of visiting each of a node's children.

A key trade-off experienced with this method is that between exploitation and exploration. It is tempting to always pick the most exploitative option, but this may lead to encountering fewer unique situations and therefore having a weaker opponent model. These issues are explored in [6,7]

One main restriction on adaptive models is that they must get decent results after dozens of hands rather than thousands. Because of this, if the frequentist best response method is to be used, the lower dimensionality space to which the world is mapped needs to be exceedingly simple, which limits the effectiveness of the agent. Relatively good results have been obtained by using a series of successively simpler world models and inter-linking them so they share information as to make optimal use of every observed hand. [6,8]

More traditional machine learning methods have also been applied to this problem. For instance neural networks were used in [9]. The author notes positive results when evaluated against human players despite the fact that the large computational cost meant he could only perform the game tree search on the current betting round.

Lastly there are some agents that combine both approaches. They pre-calculate several disjoint exploitative strategies, each corresponding to a class of players and during play they attempt to classify their opponent into one of these classes. Based on how sure they are of their classification they will play some combination of the respective class' exploitative strategy and the equilibrium strategy.[3] Also in this last category is the random forests approach used in[10]. Here each forest stood for one possible opponent starting hand and the probability of each forest being correct was adjusted, as the hand progressed, based on the classification probability of the opponent into one of several basis classes (tight play, aggressive play, etc)

3. Data preparation

I downloaded the logs for the 2012, 2 player, no-limit hold'em competition from http://www.computerpokercompetition.org/downloads/competitions/2012/logs/ .

The dataset for each of the competitions is a folder where every file holds the history of one match. The 2 player no limit competition, for instance, has files with names of the form: "2012-2p-nolimit.player1.player2.run-9.perm-0.log". This file would be the 9th match ("run-9") played between "player1" and "player2". In order to minimize variance in the results every match is repeated with the same cards dealt in the same order, but with the players permutated at the table. Since this is a 2 player competition there's just two permutations possible, shown at the end as "perm-0" or "perm-1".

In every file, every row shows how one hand was played. The rows are of the form: "STATE:4:cc/cr246f:Ts2h|9dQs/3dQh5s:-100|100:player1|player2". This says that the fourth hand("STATE:4") started with two calls ("cc"). In the 2nd round of betting (on the flop) player1 calls again, but player2 raises 246 to which player1 folds and ends the hand ("cr246f"). The players' starting hands are shown, with face value and then suite ("Ts2h" for player1 and "9dQs" for player2). And the community cards are also shown, in this case just the first 3 because of player2's fold on the flop ("3dQh5s"). The monetary outcome of the hand is then shown, here player1 lost 100 and player2 gained 100 ("-100|100") and finally the play order is shown, with player1 being first ("player1|player2").

At the end of the file there is one row showing how much each player has won/lost overall. It is of the form "SCORE:315437|-315437:player1|player2", saying player1 won 315437 here.

A complication arises by the game using "reverse blinds", which means that the first player puts in the big blind and the second puts in the small blind. In turn this means that the second player actually goes first in the pre-flop round. In all subsequent betting rounds the order is the normal one (first player goes first). This needs to be accounted for, since going first or second can change a

player's strategy.

It's also worth mentioning that a further simplifying assumption is made in these games. The assumption is called “Doyle's game” and it means that ,at the start of every hand, the players always have the same number of chips (200 big blinds specifically). Essentially this removes the need for the bots to perform inter-hand bankroll management.

I built a python script that goes through these files and filters just the hands that satisfy our criteria. Specifically, the criteria are that three cards of the same suit come up on the flop and either the first player raises on the flop, or the first player checks and then the second player immediately raises. These cases are then split into two categories. If the player has the flush I call it a negative and if the player has no cards of the needed suit I call it a positive. I ignore cases where he only has one suited card as it’s unclear how much of a bluff it is in that scenario.

The filtered hands are stored per player and per match (i.e. in files of the form “player1_in_match_player1-player2”). This allows us to look at different granularities of the data (i.e. we can try to simulate what player2 might have done in the match, or we can just aggregate all of them by reading all files)

In total there are 376,000 filtered hands, spread across 11 players.

I’ve also managed to get the pokerEval library to work. This provides features such as identifying and naming the best 5 card hand out of 7 cards and estimating the probability of a hand winning when all the cards have not yet been revealed.

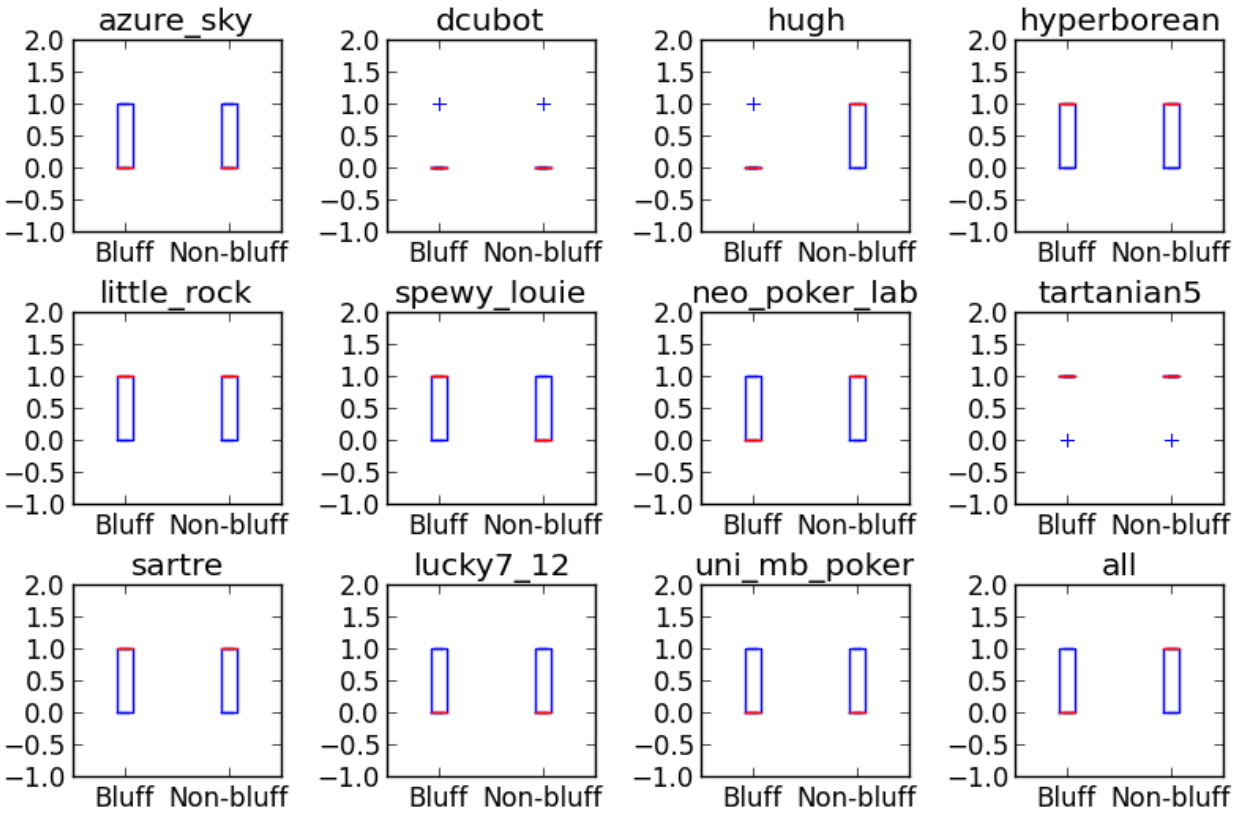
Finally, I’ve used the same dataset for the MASWS course, and there I built a script that converts the poker data into RDF triples. These can be queried relatively easily and are useful for basic exploratory data analysis.

4. Exploratory Data Analysis

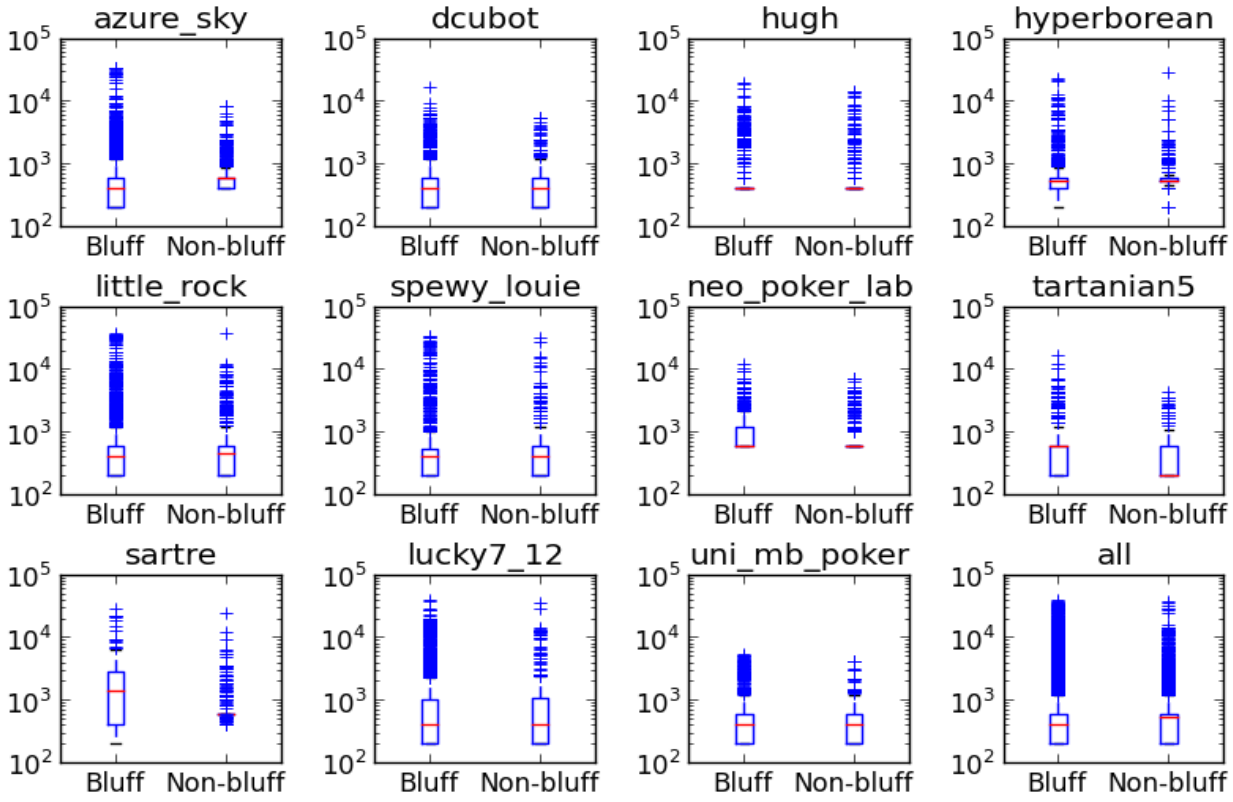
I first queried the RDF data mentioned above (available at https://github.com/RazvanRanca/RDF_Poker). I tried looking for any interesting pattern regarding the number of bluffs or legitimate bets occurring together with various basic game variables. Basically I was looking for things such as whether any of the bots liked to bluff more if the pot was larger, or if they were first to play. I did notice an interesting behaviour regarding one of the player's bet sizes (the player was “azure_sky”). It seemed like the bet sizes gave a lot of information regarding his card's strength. However, here, I realized my RDF data's limitations, in that it doesn't scale well at all. SO I was unable to use the RDF data to actually get statistics for the whole dataset.

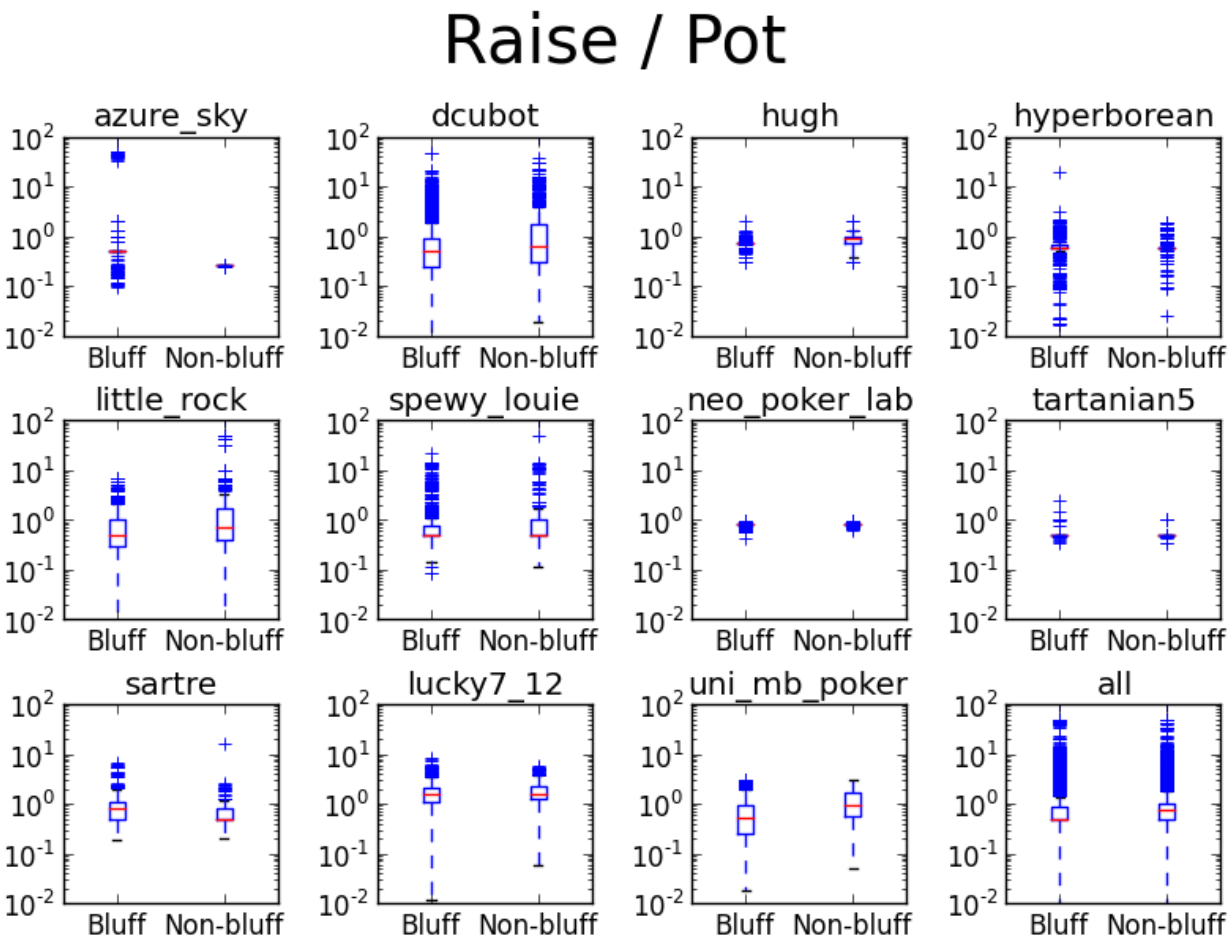
In order to get these statistics I turned back to the raw data and used matplotlib to get a sense of the variability of potential features over the two classes. At first I used features pertaining to the actual betting round in which the action happens. These features where the position in which the player is playing (0 for first, 1 for second), the size of the pot when the flop raise is made, and the amount that is raised, in proportion to the pot (i.e amount raise / pot amount). The results are plotted below for each of the 11 agents individually and for all of them cumulatively.

Position



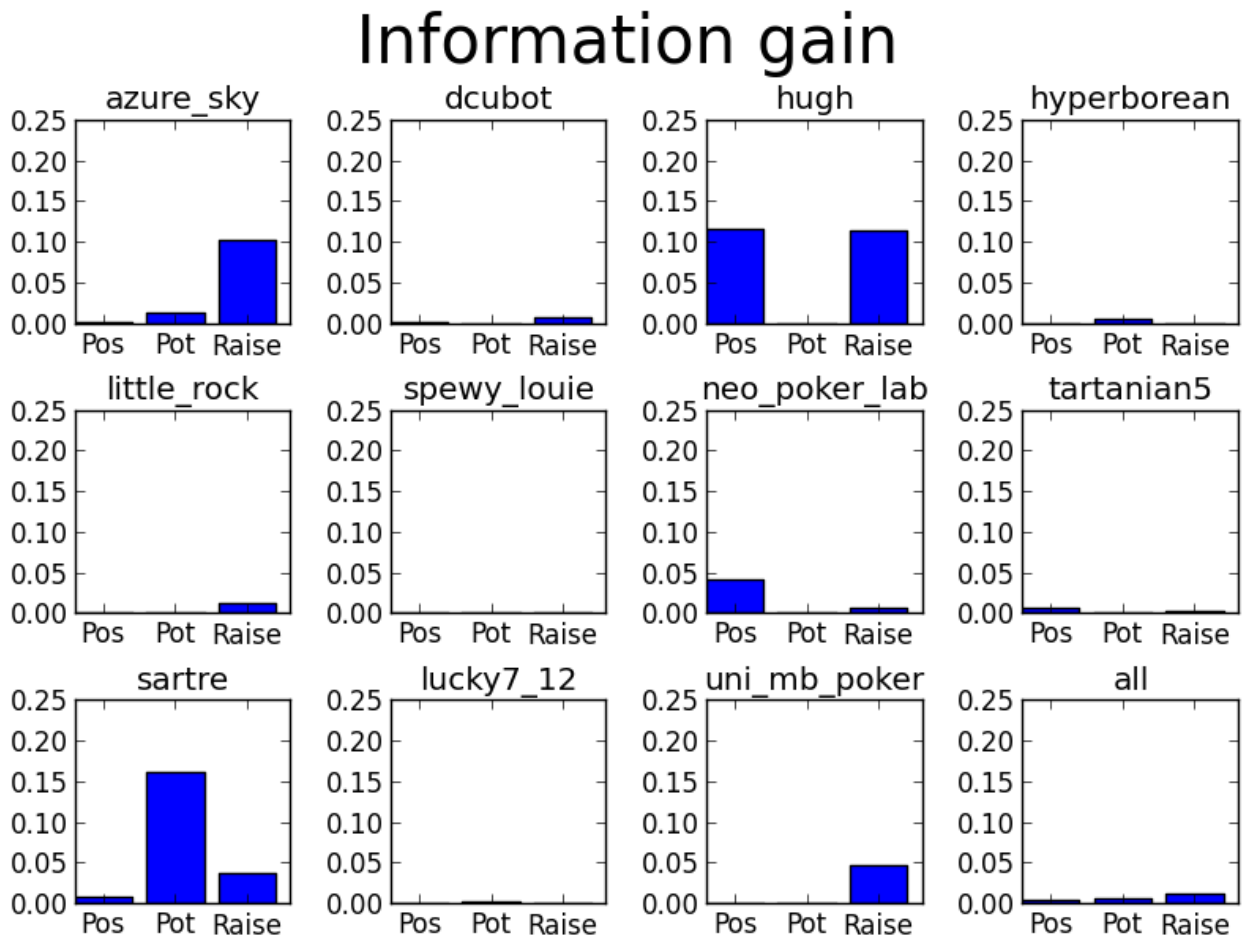
Pot ammount





As can be seen, quite a bit of information can be gained from these distributions. In particular it's interesting to note how different they are for different bots and how some bots seem to have quite simple strategies in certain cases.
For instance, here we can clearly see what I had initially found from the RDF data, namely that "azure_sky" pretty much always bets the same proportion when he actually has a flush.
Also interesting are the results for the "position" factor. It seems that most bots don't really care about their position, but a couple might.

Another way to look at the value of these factors is from an information theoretical perspective. Namely we can try to see the information gained by classifying the hands regarding to one of the factors. For the position feature, the classification is simple, since it is binary. However there are many ways to divide the other two factors into classes. For simplicity I first decided to separate them into classes based on the median values of the bluffs and non-bluffs over that feature.
So, for feature X, we calculate a "div" value which is the average of the two median value of X for bluffs and the median value of X for non-bluffs. We then separate the data based on whether it's X value is above or below the calculated "div" score. This procedure gives the following information gain values:

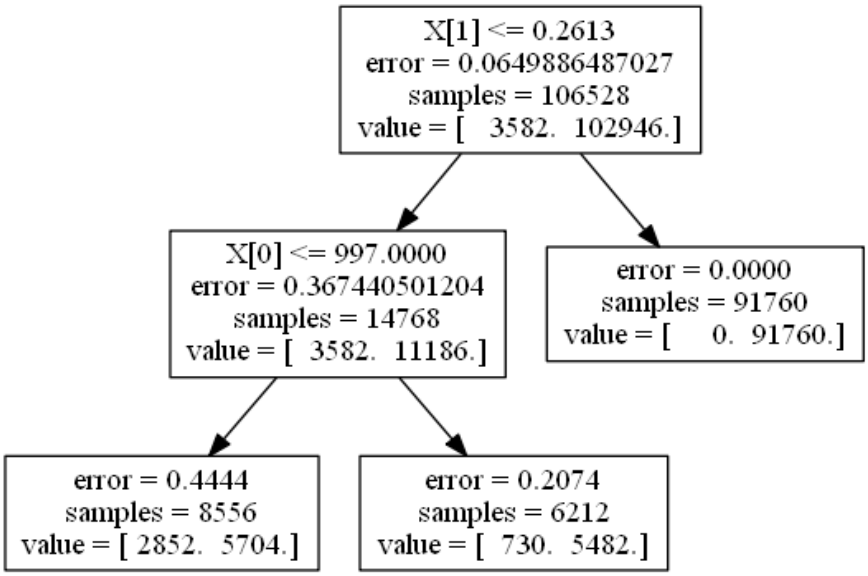


This confirms some of our previous suspicions regarding what features are useful where, though it's unclear how well this would work in general. The problem is that the “div” factor definition is quite ad-hoc. In order to better utilize these features we will have to apply some actual machine learning methods.

5. Learning Method Used

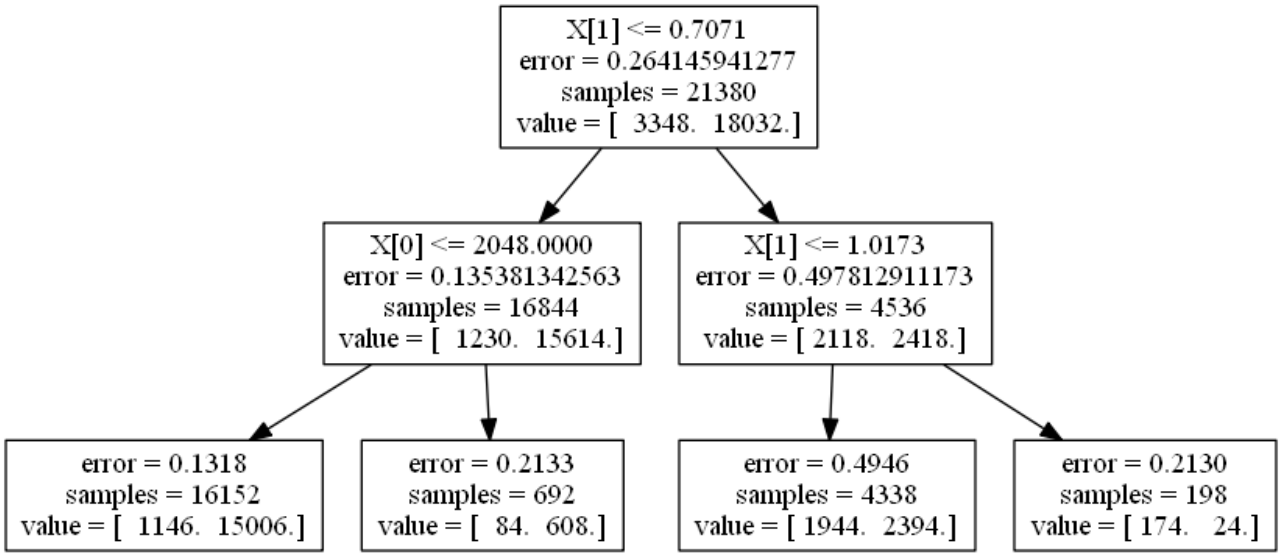
Continuing the line of thought from the previous section, the next step seems to be to apply decision trees on these features and see what the results are. One interesting test is to restrict the decision trees to a shallow depth, this way we can see how easily exploitable these bots really are. One would hope that a bot would not have a play pattern so clear that a decision stump or a very shallow decision tree could easily figure it out. Towards this goal, I generated the optimal decision trees of height 2, for each bot, over these 3 features. I won't show all of these here, as it would take a lot of space, but some interesting results emerge.

This is the tree for “azure_sky”:



It seems this bot really is extremely predictable. We can get 91760/106528 answers with 100% certainty with just 1 test!.

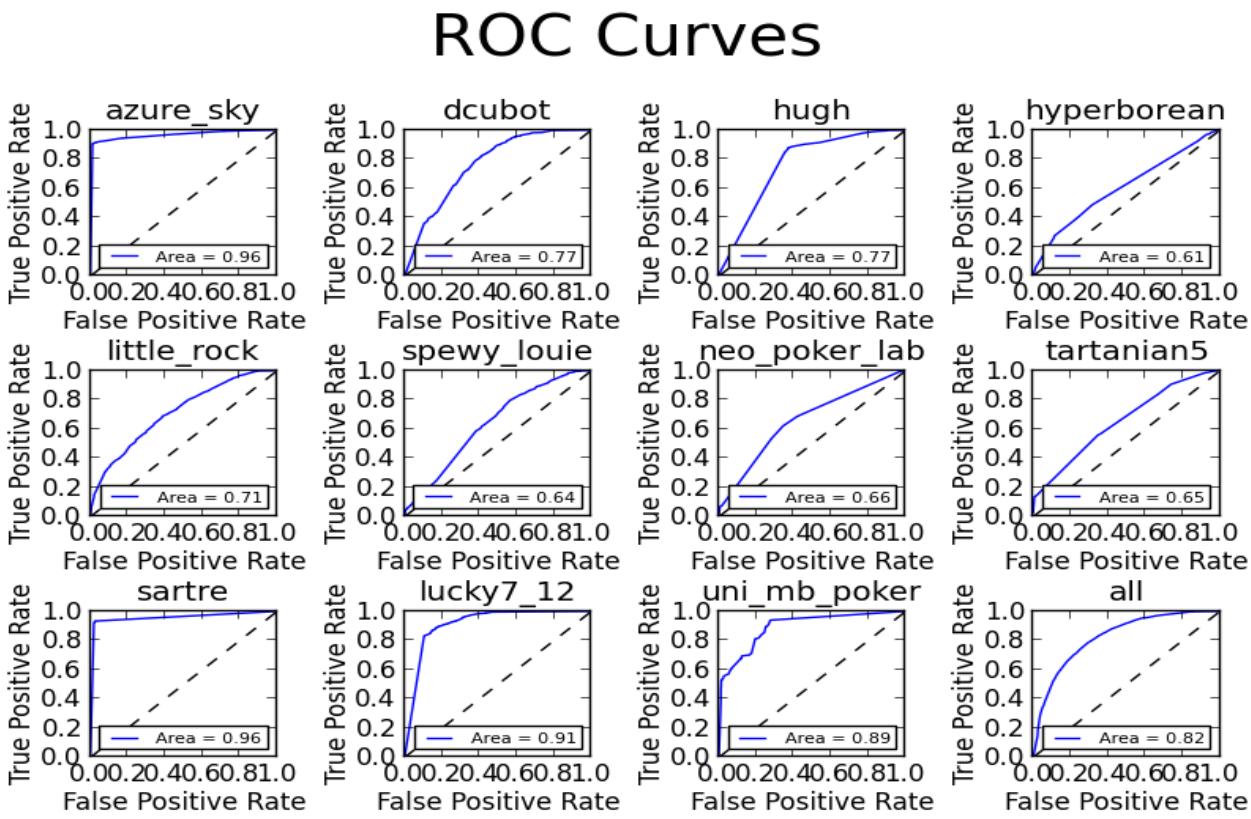
Also as expected, the position parameter isn't used much, with just two exception. One of them hugh: (X[0] is the position parameter).



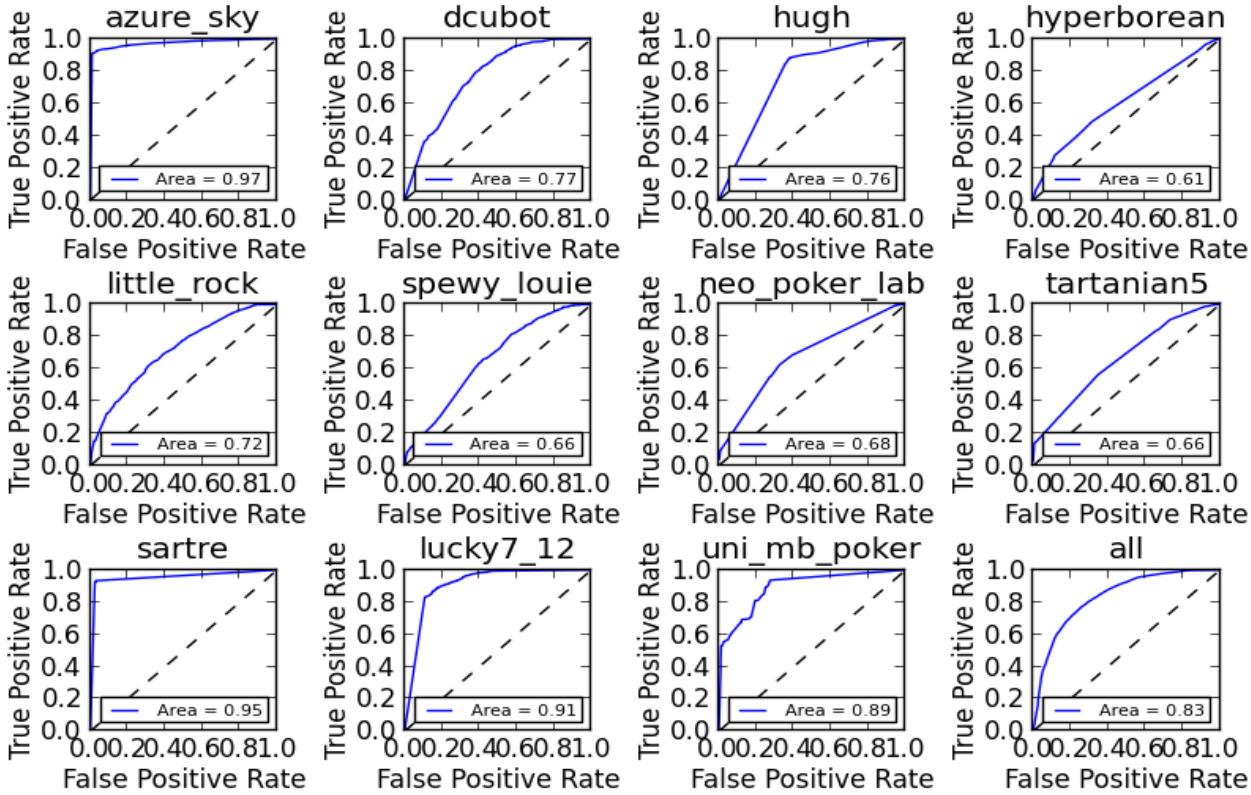
I also attempted to model an SVM on this data, however this meant the real data had to be put into buckets and this seemed to deteriorate the results. I did not have time to experiment with this sufficiently, so I omit those results.

6. Decision Tree Results

Running the basic decision tree algorithm with pruning and the 3 features discussed above gives the following results:



As it turns out, some bots are quite predictable even with such a minimal feature set. Others however are harder to figure out. The performance on all data together is not too bad however, with an AUC of 0.82. Finally, a new feature set was run:



This experiment was done, with the addition of two more features. They were related to the aggressiveness the opponent showed in the pre-flop stage (I.e his number of raises and average amount raised). However, these extra features proved to have little impact on the results, perhaps because the aggresiveness on the pre-flop is already correlated with the size of the pot at the flop.

8. Conclusion & Future Work

It seems that the the ease of modelling an opponent varies dramatically with the opponent's play style. It is surprising that even such a simple model managed to achieve a relatively good performance, but this is perhaps an inaccurate representation of a real situation because the modelling was not done live, and thus did not suffer from the imperfect information you have during play.
One line of future work would be to remedy this problem and see how well modelling can be done on the fly. Another possibility would be to further improve on the above results by investigating longer term trends, such as the bots overall aggressiveness.

9. References

[1] J. Rubin, I. Watson, Computer poker: A review, Artificial Intelligence (204), doi:10.1016/j.artint.2010.6.005
[2] M. Johanson, M. Zinkevich, M.H. Bowling, Computing robust counter-strategies, in: Advances in Neural Information Processing Systems 20, Proceedings
of the Twenty-First Annual Conference on Neural Information Processing Systems, 2007.
[3] M.B. Johanson, Robust strategies and counter-strategies: Building a champion level computer poker player, Master's thesis, University of Alberta, 2007.
[4] J. Shi, M.L. Littman, Abstraction methods for game theoretic poker, in: Computers and Games, Second International Conference, CG, 2000, pp. 333–345.
[5] M. Johanson, M. Bowling, Data biased robust counter strategies, in: Twelfth International Conference on Artificial Intelligence and Statistics, 2009,
pp. 264–271.
[6] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M.H. Bowling, R.C. Holte, J. Schaeffer, D. Szafron, Game-tree search with adaptation in stochastic
imperfect-information games, in: Computers and Games, 4th International Conference, CG 2004, 2004, pp. 21–34.
[7] A. Davidson, Opponent modeling in poker: Learning and acting in a hostile and uncertain environment, Master's thesis, University of Alberta, 2002.
[8] T. Schauenberg, Opponent modelling and search in poker, Master's thesis, University of Alberta, 2006.
[9] P. McCurley, An artificial intelligence agent for Texas hold'em poker, Undergraduate dissertation, University of Newcastle Upon Tyne, 2009.
[10] R. Maîtrepierre, J. Mary, R. Munos, Adaptive play in Texas hold'em poker, in: ECAI 2008 – 18th European Conference on, Artificial Intelligence, 2008,
pp. 458–462.