Opponent Modelling in Poker

Literature Review:

I've done a preliminary literature review(though I've omitted the references for space reasons). 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.

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. Since the number of different situations possible in poker is huge, the space is usually mapped to a lower dimensional equivalent. 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.

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 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.

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 where, when the search has reached a leaf or sufficient depth the opponent model needs to be consulted to get a probability of winning if this node or leaf is reached. The other function is two assign probabilities to the opponent's actions in every node in which the opponent must act.

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, thus limiting 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 maximize the effect of every observed hand.

More traditional machine learning methods have also been applied to this problem. Neural networks were used in one paper, and random forests in another (here each forest stood for one possible opponent starting hand and the probability of each forest was adjusted as the hand progressed)

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.

Progress:

I've downloaded the data and built a script that can filter the hands based on the specified criteria. I've stored a couple of these datasets such as the discussed one where we identify bluffs under the conditions that: three cards of the same suit come up on the flop and one player raises a "large amount" (=pot size currently). Currently 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 bluff. I ignore cases where he only has one suited card as it's unclear how much of a bluff it is in that case.

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.

I'm also using the same dataset for the MASWS course, and there I built a script that converts a the poker data into RDF triples. These can be queried relatively easily and should come in handy for exploratory data analysis, though the process is probably too slow to do mass filtering of the data in this way.

Next, I plan to some machine learning techniques on these datasets and see what kind of results are obtained. For starters I will use the features that were previously used for the neural network design, namely basic game information + a heuristic for "opponent aggressiveness".
If this doesn't work well, I'll try to refine the features by doing some more exploration of the data and trying to spot something else to add. Perhaps some PCA-like methods will be useful as well.

If this works well it could be interesting to try to estimate the the chance of the opponent making a certain move. I think this decision could be represented by trying to predict the value the opponent currently places on his hand and comparing this value with what it will cost him to call or raise.