Harvard Hold'em Poker Bot
Draft Specifications

Prepared by:

Bry Power - bapowers57@gmail.com
Michael Delaney- michaeldelaney@gmail.com
Qi Xiao - xiaoqi.pku@gmail.com
Esty Cohen - aoemom@gmail.com

Brief Overview

The project implements a poker program using a naive Bayesian algorithm to predict poker hand strength. We are planing to create a Texas Hold'Em poker player (bot) that utilizes machine learning techniques such as reinforcement learning. The bot uses the algorithm to predict what an opponent's poker hand likely is based on their past actions. Actions could include checking, raising, betting, folding, and calling. The program is learning as it plays against an opponent, and gets better as it plays the same opponent over time at predicting the probability of hands.

Feature List

1. Implement a naive Bayesian classifier to predict hands of players based on past actions. (core)
2. Learn and adapt probabilities over time as poker hands are played. (core).
3. Implement a player bot to play using the algorithm. (core)
4. Implement the above using python. (core).
5. Use a relational database to record player history. (cool)
6. Implement a user interface to interact with the program. (cool)
7. Compare a decision tree algorithm to the naive Bayesian classifier and analyze which is more accurate. (cool)

Technical Specification

Python is widely used for machine learning programs and includes various packages for computations and visualization – in some instances simple analysis/applications can even be achieved in twenty lines of code or less. In recent years python has become the data science language aside from R, which Professor Morissett has mentioned has much different syntax than other computer science driven languages. With the being said, this gives us the opportunity to skip “reinventing the wheel”. For machine learning purposes python has extensions such as NumPy that support high level mathematical computations. Python also includes libraries and modules that were specifically designed for poker simulations.

Although we could create custom modules for our Poker Bot, one of the packages we intend on integrating is PokerSleuth. This has a built in module that includes an interface for ‘hand evaluation’; that is comparing all hands at the conclusion of a given round to determine the winner. Alternatively, there are many open source hand evaluators available via public repositories which can be customized as needed. MIT has made many skeleton samples available post their Bot Competition this year. In addition to having an efficient hand evaluator, we will implement an abstraction for the deck and cards.

For this representation, we will create multiple classes. Conceptually, we are considering an individual class for the Classifier, Player, Hand, Deck, and Cards. Each class will have a constructor (including the set and get) and various methods. The Hand class will include methods to addACard(), removeACard(), countTheAmount(), and sort() – sort being some functionality that will have the ability to put a given hand in a specific sequence/pattern that will represent a rank in poker. The Desk class will include methods such as clear(), shuffle(), dealCards(), etc. The Card class will include objects for creating the individual cards themselves as there are four potential symbol and numbers that need to be matched. The Cards class much also be able to design the Rank/Strength in the game of Poker. The classifier class will need a train() method, a classify(), a calculateProbabilities() method, and a predict() method. The player class has methods which reflect the actions they may take and a method to calculate the probable had when those actions are taken. The possible actions are:

call : Bet just enough necessary to stay in.
check : Stay in when no bet in needed to do so.
fold : Quit the hand, lose all money
raise : Bet more than the minimum and force all other player who want to stay in to do the same.

In the main() run of the program, we will use the data structure ArrayList to store the actual cards that will be used in the game simulation. Other variables and methods that will be in the main class will be for the purpose of keeping scores of the players, averaging scores, keeping track of bankroll of players, etc. These features don’t necessarily play in the core functionality and so are implemented as ‘add ons’.

The following table display the prior ranks/strengths in poker that are to be implemented in our abstraction design of the cards:

Possible hands and prior probability

Hand	Probability
AKs (or any specific suited cards)	0.00302
AA (or any specific pair)	0.00452
AKs, KQs, QJs, or JTs (suited cards)	0.0121
AK (or any specific non-pair incl. suited)	0.0121
AA, KK, or QQ	0.0136
AA, KK, QQ or JJ	0.0181
Suited cards, jack or better	0.0181
AA, KK, QQ, JJ, or TT	0.0226
Suited cards, 10 or better	0.0302
Suited connectors	0.0392
Connected cards, 10 or better	0.0483
Any 2 cards with rank at least queen	0.0498
Any 2 cards with rank at least jack	0.0905
Any 2 cards with rank at least 10	0.143
Connected cards (cards of consecutive rank)	0.157
Any 2 cards with rank at least 9	0.208
Not connected nor suited, at least one 2-9	0.534

In short, creating such classes, methods, and utilizing a hand evaluation is the process of creating a poker game and should be less than 500 lines of code. Creating the game itself is our first task as what we are

really attempting to do is integrate a predictive analysis algorithm – Naive Bayes. Instead of choosing a SVM, or another decision tree, we choose Naive Bayes because it needs to be highly customized – in comparison to a tree we you simple feed a set of data into. This allows us to pick what features are of priority. In our case the features we would want to select are factors a human player would consider into their strategy while sitting at a table like, how many cards a competitor has, how many times have they raised, folded, or checked, and so on. This allows for categorical predictions and will let us compute a probability of what a player is likely to have in their hand. To test the performance of the algorithm we have the option of historical data from an actual online gaming account, we can download a purified/clean data set from various machine leaning research repositories such as UCI.edu will has Poker Hand Data Sets, or we can simple create a custom data set through means of 'randomization' – although true randomization can be accomplished, extremely complex ones can.

Lastly, are part of the application graphical interface we can include a simple graphics windows. Python has built-in libraries for GUI's. One of the standard libraries is Tkinter. By importing the Tkinter module we can create a simple main window. Using Tkinter is straightforward and simple. To manage the position Tkinter widgets come with organization methods such as pack(), grid(), and place(). Using various widgets such as Canvas, Entry, Frame, Label, Button, text, etc., we can create a simple representation of players, cards, and chips.

Next Steps

Our team has come to an agreement on using python as our chosen language. I think next, we need to decide what tools we intend on using and how much we intend on customizing ourselves. There are dozens of open source poker bots publicly available -- including abstractions, training data, evaluation functionality, and strategy algorithms. Theoretically, its possible to create the whole bot using all the tools, libraries, and sources out there. Therefore what we should decide is what critical to customizing the bot for Naive Bayes and therefore can not be leverage from an open source tool.

Secondly, each member of the team should take some time to become a little more familiar with Python. There are tons of active tutorial online that run through the core features of the language within hours. This way all of us will have Python and its dependencies install on our pcs, and can begin to play with some frameworks that may come handy in our actual design.