Blackjack with Card Counting: Cheating the System

Ryan Worley

Introduction

Background: Card Counting is a Blackjack method used to provide the player information regarding the state of the remaining cards in the deck. Players can leverage this knowledge to change their policy and obtain maximum utility from the game.

Methodology: To calculate the 'True Card Count', the following formula is used: (#(T, J, Q, K, A) - #(2, 3, 4, 5, 6))/#DecksRemaining where #() signifies the total number of the specified card values drawn. #(7, 8, 9) does not change the card count.

Motivation: Can card counting be used as a viable method to gain a high expected utility from Blackjack? If so, what policy should the user be playing with to maximize reward?

Methods

Model: MDP

State: (PlayerState, DealerState, Card Count)
Action: (Draw, Double, Stay, Initial Draw)
Reward: Bet*Modifier (Modified for blackjack (21), winning, losing, tie)
Transitions: Estimated via True Card Count. Transition probabilities assumed constant for entire hand (simplification)

Algorithm: Value Iteration

Solve for the optimum policy and expected utility of given MDP with estimated transition properties. Value iteration was conducted many times for different initial deck conditions, with variable count and mid-card percentage (percentage of 7, 8, 9) values. Both these variables are needed to generate probabilities for drawing card, but don't give information about exact probability of card, only general probability.

Backcheck Algorithm: Simulation

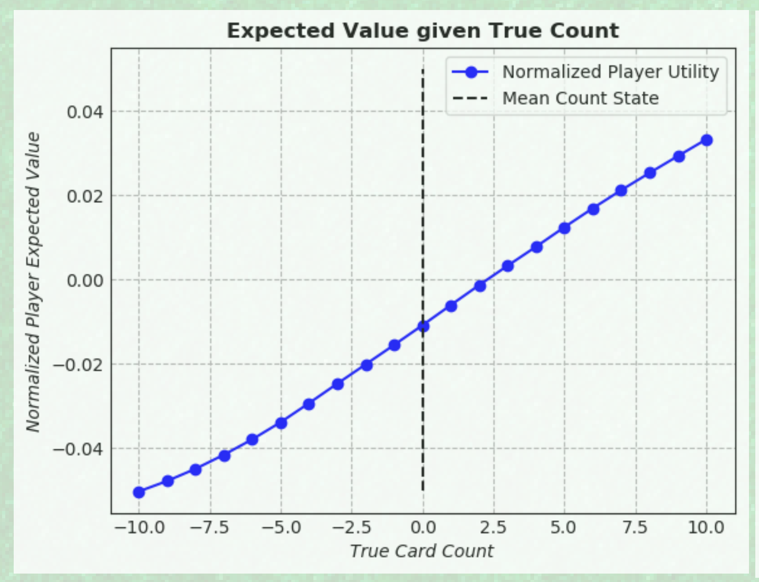
Simulation will be run using the fixed policy learned via value iteration to verify expected utility values. This is needed due to simplifications taken in the Value Iteration process to minimize state space and simplify transition probabilities.

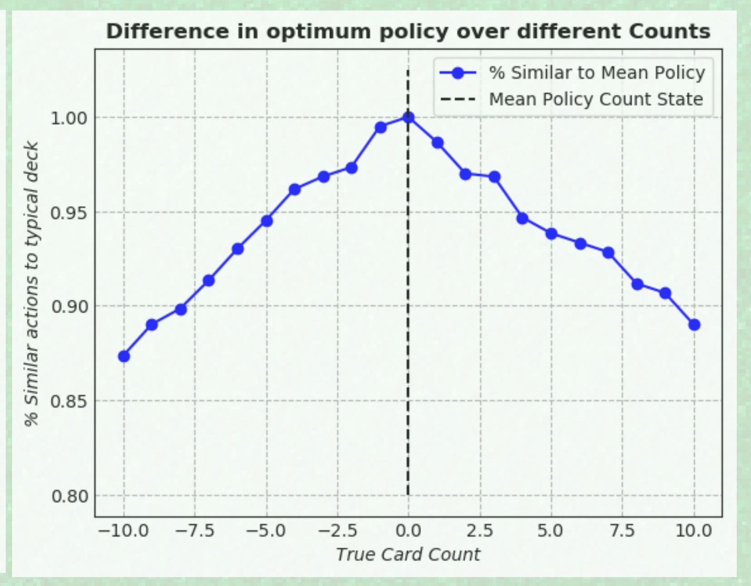
Results & Analysis

Card Count Variability

Value Iteration provides expected utility from start state of MDP for varying card counts. In the first t figure below, we see that expected utility increases linearly with a larger 'True Count' value. Once the count reaches a value of 3, the game shifts to one where the player has the advantage.

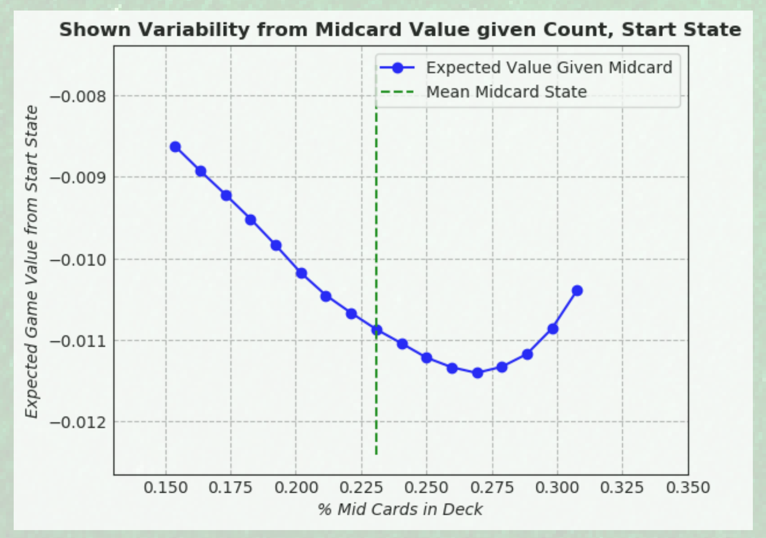
Additionally, count variation produces a change in the optimal policy on how to play the game. By having knowledge of the count, the actions one must take to get the optimal utility changes. See figure 2 below for variation.

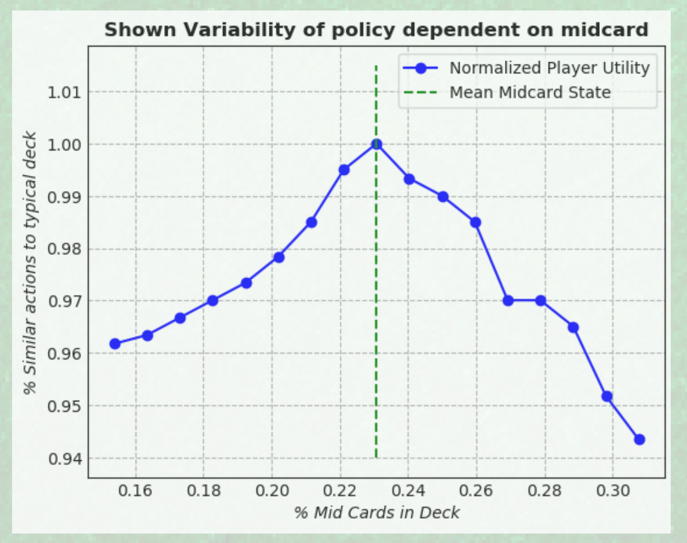




Mid-Card Variability

The number of "Mid-Cards" (cards with value 7, 8, 9) in the deck are not accounted for in the count, and therefore add variability into our value iteration process. Value iteration was run with different mid-card probabilities and variation in both policy and utility were observed.

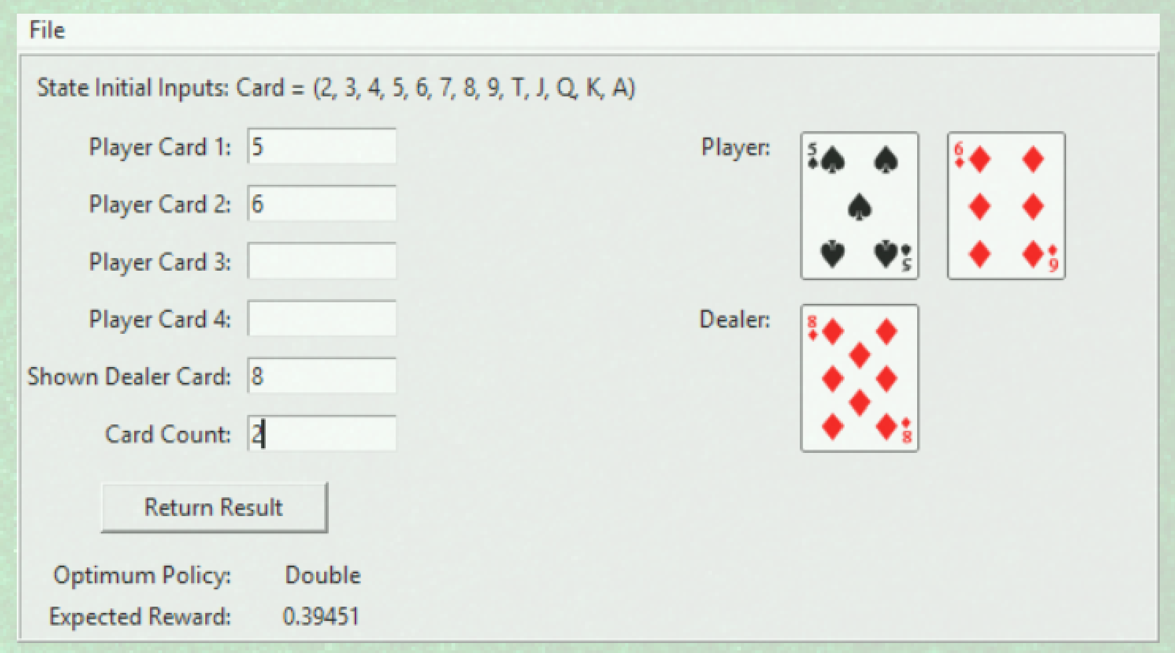




The mid-card density of a deck does have an influence on both the policy and utility of the game, but only very slightly.

Utility and Display

A GUI was created as a method to easily view the optimum policy and an estimation of the expected utility for a single game of Blackjack. The user can input the player cards, dealer cards, and true count, and the GUI will display the optimum policy



State Initial Inputs: Card = (2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A)

Player Card 1:	5
Player Card 2:	6
Player Card 3:	
Player Card 4:	
Shown Dealer Card:	8
Card Count:	2
Optimum Policy:	Double
Expected Reward:	0.39451

Conclusion

Value iteration provides us with a good estimate of the optimum policy for playing Blackjack with card counting. However, there is variability surrounding the deck state that isn't expressed by the value of the true count. Thus, the value iteration is just an approximation of the true MDP, as the transitions associated for a certain count may vary depending on the mid-card weight, as well as variation of other cards. For the final project submission, a simulation will be run testing value iteration policy for actual games of Blackjack in order to refine estimates of the expected utility given a state, and ensure that value iteration results do not make too many simplifications.

Further Analysis

- Implement a simulation method to check expected utility values from value iteration.
- Implement simulation GUI to allow user to practice card counting techniques in a variable setting.
- Different procedure using Q-learning algorithm instead of value iteration, then values of the T(s) need not be approximated.
- Implement splitting action into the algorithm. Splitting is fairly rare optimal move in Blackjack, so it was omitted to reduce the state space of exploration and increase efficiency.