Submission 2 **Ryan Worley**
CS 221 Teammates: None
Prof. Liang Due Date: 10/25/2018

Introduction

Blackjack is marketed as a simple game, one that casino novices should pick up with ease. The premise is simple: draw cards until your total card value is as close to 21 as possible and you beat the dealers hand. Past analysis on the game has found that blackjack has the best odds of any game in the casino, with only a 0.5% house edge when played properly.

While casino novices often decide to risk their money on the blackjack table, they often dont play with the optimum policy. Blackjack is a game with many states (over 80,000 for just a 26-card deck), and knowing how to maneuver through all of them to give yourself the best chance to win can be confusing.

Some companies try to capitalize on this by selling Basic Strategy Cards, which claim to give the player information on how to approach betting at every state. However, these cards dont account for differences in state such as the number of cards left in the deck and the card count at certain points. These additional factors can be leveraged to give you an edge against the casino. I believe that I can make a program that can improve the basic strategy of the starting player, by providing information on how to bet, and what their odds are in every situation on the board.

Project Summary

I plan on creating an MDP that properly plays out the game of blackjack. Using this, we can compute the optimum policy in each of the states associated with a game of blackjack. At the end of the learning process, the user will be able to enter a starting state (card count, player cards, dealer card shown) and be returned with the optimum policy and expected return by which to play with. If the program works correctly, the user should generally see positive returns while playing, especially if they decide to not bet large in games where the odds are not in their favor, and bet bigger in games where they have the edge.

Additionally, I would like to do some additional work regarding the variability involved in playing a concrete number of blackjack games. While value iteration will give us an expected value of return (average), Id like to estimate the dispersion involved in playing blackjack a finite number of times. With this information, we can give the player a better estimate of the variability in draws, at how different factors can affect this as well.

State Space Minimization

Challenges:
The biggest challenge that is faced when creating an algorithm like this is to minimize the state space. While blackjack seems like it would have a relatively small state space given the limited number of cards dealt during every hand, this is not the case. For each state, we must keep track of information regarding the total number of each card left in the deck. This is because we are tracking the probability of each hand with no replacement, meaning that once each card leaves the deck, the probability for each new card being drawn changes. At the end of each action (except those that win or lose the game), the total number of each card is inputted back into the state.

Additionally, the number of cards drawn in a single game of blackjack is quite variable. Since an MDP evaluates every state possible no matter how small the probability, states where both the dealer and user draw only a 2, 3, or 4 will come up, creating an extended state space if all of these cards are included as part of the state.

Approach:
I approached minimizing the state space in the following way. For each state, 3 separate values were stored in a tuple:

1. Players Hand (string with numbers, letters used as code for what exactly is in hand)
2. Dealers Hand (string with numbers, letters used as code for what exactly is in hand)
3. Cards Remaining the deck (tuple with index corresponding to number of cards for certain card)

Both the Players Hand and the Dealers Hand have a specific code that tells the program exactly what is in each hand. The code is as follows:

- First 1 or 2 characters: Value (numerical representation of value that the dealer or player holds)
- Letter D in characters following Value: Double Down allowed on next action
- Letter S in characters following Value: Splitting allowed on next action (duplicate cards in hand)
- Letter A in characters following Value: One ace in hand whos value is taken at 11.

By creating a code to represent the different states while not having to store exact card values or the order in which cards are drawn, I hoped to minimize the state space. For example, if the player draws the 4 and a 5 in the initial draw, this would be represented by 9D state. The 9D state could also represent a draw of 3 and 6, 2 and 7, etc. This minimizes the state space since you do not have to keep track of the individual values of cards drawn.

Actions

For the current code, 4 actions can be taken:

1. Begin: Player and dealer each draw 2 cards
2. Stay: Player stays, dealer draws until value is over 17
3. Draw: Player draws one card
4. Double: Player draws one card, dealer draws until value is over 17

To limit the tree width, some actions were eliminated for certain states that reflect simple blackjack strategy. For example, you arent allowed to stay if your value is less than 11, as there is not chance of losing if you take another card. Additionally, you are only allowed to double down on your bet with certain numbers, mainly you cannot double down on hand value greater than 12 unless you have an ace in your hand. Lastly, you are not allowed to draw another card if you have an 17 or higher with no ace.

Value Iteration

Currently, I have implemented a value iteration algorithm similar to that used in earlier class assignments. The goal of this is to find the optimum policy and expected value associated with a certain deck of cards. As currently oriented, this value iteration on the MDP has full information about the state of the problem, including the total number of the cards in the deck and the dealer and player cards. An initial deck state can be inputted into the program to see how the value changes in regards to playing with different deck states, sizes, and counts.

The run time for a starting deck state of 13 cards is about 10 seconds for the value iteration process, which seems reasonable to me. However, when the number of cards in the deck increases, the state space of the MDP increases exponentially, causing very long run-times for the program. My goal for the next assignment is to increase the efficiency of my code with larger decks.

Playing a game with a start state of 26 cards (each card type having an equal number in the deck) nets an expected value of 0.02 for each unit that you spend. Unfortunately, I believe that this analysis is wrong, since the odds should be against you for a normal game of blackjack. I need to go back and check my succProbReward function, to ensure that all different actions taken in this section give pay out the proper reward.

Q-Learning

Additionally, reinforcement learning will be implemented, since value iteration isnt totally realistic showing of the game. In a real game of blackjack, the player will have knowledge of the count (the state), but no knowledge of the exact probabilities of the reward associated with each action. While the count gives the player a general knowledge of whether a

high or low card is drawn, it does not tell the player exactly what cards are left in the deck.

This portion of the algorithm has not been implemented yet.

Q-Learning Features

: For the current code, 4 actions can be taken:

1. Current Card Count (true count, way to represent deck state as float)
2. Dealers Top Card (only card visible to the player)
3. Player Card Value (total value of players cards)
4. Ace Counter (more aces in players hand provides more flexibility)

Hopefully, these features are sufficient to explore the state space and provide a policy that is very close to optimum.

Next Submission

For the next submission, I plan on working on finishing my algorithm, and completing Q-learning as well.

References

[1] Jensen, Kamron, *The Expected Value of an Advantage Blackjack player*. All Graduate Plan B and other Reports. 524

[2] Wong, Stanford 1994. *Professional Blackjack*, page 31, 1994 ed.