Finding the Optimal Blackjack Strategy Using Monte Carlo Markov Decision Process

Manuel Lopez-Mejia
College of Engineering, North Carolina State University, Raleigh, USA, manuel.lopezme05@gmail.com, 92855

Keywords: Blackjack, Monte Carlo Tree Search, Markov Decision Process, Markov Chains, Dynamic Programming.

Abstract. In this paper, I will introduce the concept of Bayesian Probability and how it can be applied to Blackjack in order to predict the next card in the game deck. We will extend this formulation to calculate the expected value of making a particular decision in a game (e.g., Hit, Stand, Double Down, or Split) in order to find what will maximize the player's return. We will model the player's behavior using a Markov Decision Process and use Monte Carlo Tree Search algorithm to estimate value functions by averaging returns from simulated episodes.

1 Introduction

Blackjack is a popular card game where you try to reach a hand as close to 21 as possible without going over (busting). [cite: 4] If you are able to have a higher hand than the dealers hand, you win. [cite: 5] Cards are ranked with values from 1 to 11, with Ace being worth 1 or 11 (whichever is optimal) and face cards (Kings, Queens, and Jacks) being valued at 10. [cite: 6]

Players and the dealer are initially dealt a pair of cards which are shown to everyone except for one of the dealers cards. [cite: 6] Players move first, choosing to either stand, hit (draw one more card), double down (double one'es wager by drawing one card and then stand), and if dealt a hand with a pair of the exact same cards, split where you can split your current hand into two new hands and wager the same amount on each. [cite: 7] After a player stands, the dealers hidden card is shown and he must hit until he reaches at least 17. For every round of Blackjack played, cards are drawn without replacement until they're reshuffled, typically once 15 to 50 percent of the game deck has been drawn. [cite: 8] My fascination with Blackjack started when I was a kid playing Pirates of the Caribbean: The Curse of the Black Pearl video game, a kids MMORPG based off the self-titled film,where players could play Blackjack in taverns and underground parlors. [cite: 9] Today, with the rising popularity of online gambling among young men[1], and also my personal consumption of clips of the gambling influencer known as "Togi" who believes that "gambling addiction isn't real", I became interested in exploring methods in stochastic simulation and mathematical programming to find what will give you the best chance at winning in a game of blackjack. [cite: 10]

2 Background

2.1 Bayesian Probability

Since the game is played with a finite set of cards (a shoe can consist of any number set of cards, typically 6 decks in Casino games) and we can have some information of what cards have been played, we can estimate the probability of what the next card in the shoe will be and what the final dealer hand's will be so that we know what actions will give us the best chance at beating the dealer and maximize our winnings. [cite: 11] As the game continues and we continue draw more cards, we gain more information of what the shoe composition looks like and are able to predict with more certainty the likelihood of what cards will be drawn next. [cite: 12]

This is what Bayesian Probability is about, updating our beliefs as new information comes to light. [cite: 13]

2.1.1 Formulation

Let:

- H: the player's current hand (a multiset of card ranks)
- U: the dealer's upcard
- P: the multiset of all cards dealt so far (including player's cards, dealer's cards, and previous rounds)
- $R = \{\mathrm{A}, 2, 3, \ldots, 10, \mathrm{J}, \mathrm{Q}, \mathrm{K}\}$: the set of card ranks
- $m_r = 4$: the number of cards of rank r per deck, for each $r \in R$
- $D \in \mathbb{N}$: the number of decks in the shoe

The initial count of cards of rank ¡span class="math-inline"¿r¡/span¿ in the shoe is

$$x_r^{(0)} = D \cdot m_r, \quad r \in R,$$

and the initial total number of cards is

$$N^{(0)} = \sum_{r \in R} x_r^{(0)} = 52D.$$

Let

$$O = H \cup \{U\} \cup P$$

be the multiset of all observed cards. [cite: 14, 15] For each rank ¡span class="math-inline"¿r¡/span¿, define

$$\Delta_r = \big|\{c \in O : \mathrm{rank}(c) = r\}\big|,$$

i.e., the number of cards of rank ¡span class="math-inline"¿r¡/span¿ removed from the shoe. [cite: 16] Then the remaining count of rank ¡span class="math-inline"¿r¡/span¿ is

$$X_r = x_r^{(0)} - \Delta_r,$$

and the total remaining cards is

$$N = \sum_{r \in R} X_r = 52D - |O|.$$

2.1.2 Probability of Drawing Rank Value

Since draws are without replacement and the remaining cards are identically distributed, the probability that the next card has rank ¡span class="math-inline"¿r¡/span¿ given the current state ¡span class="math-inline"¿H, U, P¡/span¿ is

$$\mathbb{P}\big(\mathrm{rank}(\text{next card}) = r \mid H, U, P\big) = \frac{X_r}{N}.$$

2.1.3 Proof

By symmetry, each of the ¡span class=”math-inline”¿N¡/span¿ remaining cards is equally likely. Of these, exactly ¡span class=”math-inline”¿X_r¡/span¿ have rank ¡span class=”math-inline”¿r¡/span¿. [cite: 17, 18, 19, 20] Thus the probability of drawing rank ¡span class=”math-inline”¿r¡/span¿ is the fraction of remaining cards of that rank. [cite: 20]

2.2 Expected Value of Player's Hand

For our model, we want to select the action that maximizes the player's expected value (AKA ”expected utility”). [cite: 21] In this case, expected value is the payoff we receive from our wager based on whether we choose to Hit, Stand, Double Down, or Split (if possible). [cite: 22]

2.2.1 Formulation

Let:

- $R = \{A, 2, 3, \ldots, 10, J, Q, K\}$: Set of card ranks. [cite: 23]
- X_r: Number of cards of rank r remaining in the deck(s). [cite: 24]
- $N = \sum_{r \in R} X_r$: Total number of cards remaining. [cite: 25]
- $\Pr(r) = \dfrac{X_r}{N}$: Probability of drawing a card of rank r. [cite: 26]
- H: The player's current hand (a multiset).
- U: The dealer's upcard. [cite: 27, 28, 29, 30, 31, 32]
- P: The multiset of all cards drawn in previous rounds (observable history).
- $v(H)$: The value of hand H, treating Aces optimally.
- $\in \{0, 1, 2, 3\}$: Number of splits remaining.
- $P_D(y \mid U, P)$: Probability that the dealer ends with value y, given upcard U and card history P.

Payoff Function

For simplicity denote *x* as $V(H_F)$ and *y* denote the final hand value of the dealer, the final value of the hand of the player. [cite: 32, 33] The payoff function is represented by

$$g(x, y) = \begin{cases} +1, & x \leq 21 \text{ and } (y > 21 \text{ or } x > y) \\ 0, & x \leq 21,\ y \leq 21,\ x = y \\ -1, & \text{otherwise} \end{cases}$$

Expected Value Formulas

Expected Values for Each Action

1. **Standing**: Take no more cards

$$\mathrm{EV}_{\text{stand}}(H) = \sum_{y=17}^{22} P_D(y \mid U, P) \cdot g(v(H), y) \tag{1}$$

2. **Hitting**: Draw 1 card

$$\mathrm{EV}_{\text{hit}}(H,) = \sum_{r \in R} \frac{X_r}{N} \cdot \mathrm{EV}^*(H \cup \{r\},) \tag{2}$$

   You draw one card rank r) with probability x_r/N. [cite: 33, 34, 35] Update your hand to $H \cup \{r\}$. Choose the best action given the same number of splits remaining

3. **Doubling Down**: Double the wager, draw once and stand

$$\mathrm{EV}_{\text{dd}}(H) = \sum_{r \in R} \frac{X_r}{N} \cdot 2 \cdot \mathrm{EV}_{\text{stand}}(H \cup \{r\}) \tag{3}$$

4. **Splitting** (if ¡span class=”math-inline”¿H ¯
   {r, r
   }¡/span¿ and ¡span class=”math-inline”¿
   geq 1¡/span¿)

   Let ¡span class=”math-inline”¿X_{r_2}{r_1}¡/span¿ denote the updated count of ¡span class=”math-inline”¿r_2¡/span¿ after removing ¡span class=”math-inline”¿r_1¡/span¿ from the deck. [cite: 35, 36, 37, 38, 39]

$$\mathrm{EV}_{\text{split}}(H = \{r, r\},) = \sum_{r_1 \in R} \sum_{r_2 \in R} \frac{X_{r_1}}{N} \cdot \frac{X_{r_2}^{(-r_1)}}{N-1} \cdot [\mathrm{EV}^*(\{r, r_1\}, -1) + \mathrm{EV}^*(\{r, r_2\}, -1)] \tag{4}$$

   $X_{r_2}^{(-r_1)}$ is the count of r_2 after removing the r_1 drawn from the first split. You draw 2 new cards, which couple with ¡span class=”math-inline”¿r_1¡/span¿ and ¡span class=”math-inline”¿r_2¡/span¿ separately to form 2 new hands: a right hand $\{r_1, r\}$ and left hand $\{r_2, r\}$. Each hand is played optimally with ¡span class=”math-inline”¿1¡/span¿ remaining splits. Since you now have 2 hands, their expected values are added together.

Optimal Action Value

At every game state we choose the expected value that maximizes our return

$$\mathrm{EV}^*(H,) = \max\left\{\mathrm{EV}_{\text{stand}}(H), \mathrm{EV}_{\text{hit}}(H,), \mathrm{EV}_{\text{dd}}(H), [\mathrm{EV}_{\text{split}}(H,)]_{H=\{r,r\},\geq 1}\right\} \tag{5}$$

The objective function defines a recursive dynamic program that can be solved backwards from terminal hands (bust or stand) upwards. [cite: 39, 40, 41, 42, 43, 44, 45]

Method: Markov Decision Process

A Markov Process refers to a process where the possibility of arriving at a certain state depends on the conditions of the current state. For example, the probability of us being able to split in the next state depends on whether or not we've exhausted all our possible splits in the current state.

However since the probability of the next state depends on the players decision and we want to maximize the probability obtaining the state with the highest expected value, we will be using a special case known as the Markov Decision Process (MDP) where outcomes are partly random and partly under the agents (e.g, the player) control

Let

States: $s = (H, U, X,)$, where:

- H: Player's current hand (multiset of ranks)
- U: Dealer's upcard
- $X = (X_r)_{r \in R}$: Remaining counts of each rank
- $\in \{0, 1, 2, 3\}$: Number of splits remaining

Actions: $A(s) = \{\text{Stand},\ \text{Hit},\ \text{Double},\ \text{Split if } H = \{r, r\} \text{ and } \sigma \geq 1\}$

The probability of reaching a particular next state s' from $A(s)$ can be formulated by the following transition functions

Transitions:

- **Stand:** Dealer plays out. Transition to terminal state.
  $P(s' \mid s, \text{Stand}) = P_D(y \mid U, X)$, where s' encodes terminal outcome with dealer total y
- **Hit:** Draw one card r with probability $\frac{X_r}{N}$.
  $s' = (H \cup \{r\}, U, X^{(-r)},)$
- **Double:** Draw one card r, then stand. Terminal transition.
  $s' =$ terminal with doubled reward, dealer plays out
- **Split:** If allowed, draw two cards r_1, r_2 without replacement.
  Create two hands: $(\{r, r_1\}, U, X^{(-\{r,r_1\})}, -1)$ and $(\{r, r_2\}, U, X^{(-\{r,r_2\})}, -1)$

Rewards: 0 for non-terminal steps. Terminal reward:

$$R = g(\text{player total},\ \text{dealer total})$$

Doubled if action = Double. Intermediate transitions have reward 0. Terminal transitions deliver $R = g$(Player Total, Dealer Total) or doubled if action a = Double Down

Discount: $\gamma = 1$ Assume a risk factor of 1 since the game is finite and we care only about the total undiscounted payoff.

Bellman Optimality Equations

Our goal can therefore be written as maximizing the expected value of state S

$$V^*(s) = \max_{a \in A(s)} Q(s, a)$$

where

$$Q(s, a) = \sum_{s'} P(s' \mid s, a)\,[R(s, a, s') + V^*(s')]$$

The expected value for each action can be given as:

- **Stand:**

$$Q(s, \text{Stand}) = \sum_{y=17}^{22} P_D(y \mid U, X)\ g(v(H), y)$$

- **Hit:**

$$Q(s, \text{Hit}) = \sum_{r \in R} \frac{X_r}{N}\ V^*(H \cup \{r\}, U, X^{(-r)}, s_{\text{split}})$$

- **Double:**

$$Q(s, \text{Double}) = \sum_{r \in R} \frac{X_r}{N} \cdot 2 \sum_{y=17}^{22} P_D(y \mid U, X)\ g(v(H \cup \{r\}), y)$$

2.3 Steps to Solve with Dynamic Programming

1. Initialize $V_0(s) = 0$ for all s, or to known terminal values.

2. Repeat until convergence:

$$V_{k+1}(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)[R(s, a, s') + V_k(s')]$$

3. Policy extraction:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)[R(s, a, s') + V^*(s')]$$

2.4 Limitations

This requires we pre-compute $P_d(y|U, x)$ for each dealer upcard U and card-count x, iterate over all reachable states of $(H, U, x,)$ and store their values. So that in a given state we lookup the optimal policy to take.

However, this is gravely computationally intensive because there can be $\binom{52D}{k}$ possible states for a given hand of size K. For example, suppose a game with a shoe of 6 decks, the number of possible combinations for the players initial hand is $\binom{312}{2}$=$48,516$. This count skyrockets as you allow k to grow as the player increases his hand count. [12pt]article amsmath, amssymb, amsfonts enumitem geometry margin=1in

3 Method: Monte Carlo Tree Search for Blackjack MDP

Dealing with the Blackjack MDP can get tricky because figuring out the perfect strategy involves looking at tons of possible game states. Doing this directly is often just too computationally overwhelming. This is where ideas like Monte Carlo simulations come in handy, famously used by Stanislaw Ulam who faced a similar puzzle: figuring out the odds of winning a game of solitaire without checking every single card combination. The clever idea was to simulate the game many times (say, 100 trials on a computer) to get a good estimate. Since this relied on chance and simulation, Ulam fittingly named it after the Monte Carlo casino [**?**].

For our Blackjack problem, we'll use a specific kind of Monte Carlo method called Monte Carlo Tree Search (MCTS). It's a smart approach for games like Blackjack where you have a finite number of moves and the game eventually ends. Instead of blindly exploring, MCTS builds a search tree, focusing its simulation power on the moves that look most promising, while still checking out less-explored options now and then.

So, rather than trying to calculate exact values for every state and action in the full MDP, we use MCTS to get a good approximation. It estimates the quality of taking an action 'a' from a state 's' (let's call this $Q(s, a)$) by running many simulated games down the tree. After enough simulations, the action MCTS picks from the current state is a solid approximation of the best possible move, giving us something close to an optimal policy (ϵ-optimal) without needing to solve the whole MDP explicitly.

Here's the basic loop MCTS runs over and over to build its search tree:

- **Selection:** Start at the root (the current game state). Move down the tree by picking child nodes that look good. We use a rule like UCT (Upper Confidence bounds applied to Trees) to balance choosing nodes we know are good (exploitation) versus nodes we haven't tried much (exploration). Keep going until we hit a node at the edge of our current tree (a leaf node).
- **Expansion:** If the leaf node isn't a game-over state, add one or more new child nodes representing moves we haven't explored yet from this state. Pick one of these new nodes.
- **Simulation (Rollout):** From this new node, play out the rest of the game randomly (or using some simple strategy) until it ends (win, lose, or draw). This quick playthrough gives us an idea of how good this new node (and the path to it) might be.
- **Backpropagation:** Take the result from the simulation (win/loss/draw) and update the statistics (like win counts and visit counts) of all the nodes visited on the way back up from the new node to the root. This helps refine the tree's estimates for the next iteration.

We let MCTS run this loop many times. Once we're out of time or have run enough iterations, we look at the children of the very first node (our starting state) and pick the move that leads to the child with the best stats (usually the highest win rate or the most visited). This is the move MCTS recommends for our current situation in the Blackjack game.

Let

- State $s = (H, U, X, s_{\text{split}})$, where:
  - [leftmargin=1.5cm]

    H: Player hand (multiset of ranks)
  - U: Dealer upcard
  - $X = (X_r)_{r \in R}$: Remaining card counts
  - $s_{\text{split}} \in \{0, 1, 2, 3\}$: Splits remaining
  -
- Actions $A(s) = \{\text{Stand}, \text{Hit}, \text{Double}\}$ plus Split if allowed.

Node Statistics

For each node s and action $a \in A(s)$:

[leftmargin=1.5cm]$N(s)$: visits of state s $N(s, a)$: times action a taken $W(s, a)$: total reward from those simulations $Q(s, a) = W(s, a)/N(s, a)$: average action value

Algorithm Phases

Selection

Recursively select

$$a^* = \arg\max_{a \in A(s)} \left[Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} \right]$$

until leaf or unexpanded node.

Expansion

Add new node s_L with all $N(s_L) = N(s_L, a) = W(s_L, a) = 0$, expand one action to child.

Simulation (Rollout)

From new node, play rollout policy π_{rollout} to terminal, collect reward $G = g(\text{player total}, \text{dealer outcome})$.

Backpropagation

For each visited (s, a) in path:

$$N(s) \leftarrow N(s)+1, \quad N(s,a) \leftarrow N(s,a)+1, \quad W(s,a) \leftarrow W(s,a)+G, \quad Q(s,a) = \frac{W(s,a)}{N(s,a)}.$$

Deck Integration

Use exact draw probabilities:

$$\Pr(r) = \frac{X_r}{N}, \quad X \to X^{(-r)}, \quad N \to N-1.$$

Actions map to MDP transitions: Hit = sample one card, Stand = terminal + dealer simulation, Double = one draw then stand, Split = two subtrees.

Decision Rule

After M iterations,

$$\hat{Q}(s_0, a) = \frac{W(s_0, a)}{N(s_0, a)}, \quad a^* = \arg\max_a \hat{Q}(s_0, a).$$

This action approximates one that maximizes the true $Q^*(s_0, a)$ in the full MDP

Limitations

[leftmargin=1.5cm]Memory grows with visited nodes (much smaller than full state space). Time per iteration is tree depth + rollout length. With $M \to \infty$ and UCT, MCTS converges to optimal action.