Jake Seary
Statistics 486

The House Always Wins

An Analysis of Blackjack using Monte-Carlo Simulation

By Jake Seary

Jake Seary
Statistics 486

I. Abstract

Via a Monte-Carlo simulation of the popular casino game Blackjack, the long-term impact of various factors on the amount that a gambler could earn playing the game are observed. In this simulation, a strategy of standing above a specific threshold value and hitting otherwise is considered for every threshold value between 11 and 20, and the long-term winnings for each of these values is calculated. Furthermore, variance analysis is used to verify the veracity of the conclusions of the simulation. Additionally, the effect of the number of decks used in the game on long-term winnings is investigated.

II. Introduction

Blackjack is a very simple card game often played at casinos. The premise is rather simple; all players are dealt two cards, and then the dealer is dealt two cards. A player may hit to receive an additional card or stand to keep their current hand. The objective is for the total card values to be as close as possible to 21 without exceeding it. If the sum of the values exceeds 21 after a hit, that player busts, the round ends, and the dealer wins their bet. If the sum of the dealer's hand is closer to 21 than a player's, then the dealer wins their bet. If the sum of the player's hand is closer to 21, the player wins twice their bet. If both hands are equally close, neither the dealer nor the player gain or lose money. For the purposes of the game, all face cards have a value of 10, and each ace can have a value of either 1 or 11.

The interesting part of Blackjack is whether one should hit or stand given their total and what is known about any other cards that have been dealt. To keep things ostensibly fair, the dealer will always choose to stand at a sum of 17 or higher and hit otherwise. While the most optimal strategy would be to tailor decisions to the highest expected payout, such calculations are often difficult to perform on the fly. As such, many players will enter with specific schemes of playing. One such scheme is to employ a dealer-like strategy of standing when the hand exceeds a specific threshold value and hit otherwise. If a player were to employ such a strategy, what threshold would beget the best long-term payout? That is one question this Monte-Carlo simulation will attempt to answer.

One other consideration to make is that at a casino, to avoid tactics such as card counting, dealers in Blackjack typically do not use only one deck of cards, instead dealing from a larger deck made up of multiple standard decks, often six. The number of decks in use definitely impacts optimal play, but what about the dealer-like strategy of relying on a threshold? This simulation will also allow us to see if this parameter makes a significant difference.

For the sake of simplicity, all bets are $1. This makes a player's loss have a net profit of -1, a tie have a next profit of 0, and a win have a profit of $1. Furthermore, the simulation assumes that only one player and the dealer are participating in the game.

III. Materials and Methods

To run the simulation, a few R functions[1] are written in a script and then called as appropriate. The script in question is appended. One function is designated for counting the sum of the card values in both the player's and the dealer's hand, accounting for the multiple possible cases with aces. One function takes in a deck, i.e. a random permutation of the possible card values, as input and gives cards to the player and the dealer until an outcome is reached. Finally, a series of loops is used to repeatedly play the game and calculate the mean winnings of a specified number of consecutive rounds. The script is run in R with the command `>source('blackjack.r')`, provided the script is in the working directory.

In this simulation, a set of rounds is considered 12 rounds. The mean winnings of each set is considered one observation. This is demonstrated in the first few lines of output. To test each standing threshold, 1000 sets are simulated for each of the possible thresholds from 11 to 20, i.e. 1000 observations per threshold. Then, a one-way ANOVA test[2] is performed to test the hypothesis that all of the thresholds have the same mean winnings; the threshold is considered a categorical variable with ten different levels. If the test shows significance, then there is a very low probability that the observed data would occur if the true means were the same, so we would reject the hypothesis that the choice of betting threshold has no impact on winnings. In order to see which thresholds in particular differ from one another, Tukey's procedure[3] is then used, assuming the ANOVA gave evidence for the existence of significant differences.

Once this test is done, the next test fixes the standing threshold to either the most optimal strategy if the ANOVA finds differences between them or a threshold of 17 if there is no evidence they are any different. The variable that changes in this test is the number of decks that are dealt from. All integers from 1 to 12 are simulated for 1000 observations each, and then two different statistical tests are performed. The first treats the number of decks as a numerical variable to see if there exists a linear relationship between average winnings and the number of decks used. The next treats the number of decks as a categorical variable instead and runs another ANOVA test on this data.

IV. Results

Full output is pasted in the appendix.
Each time the script is run, the results of the first set of rounds that are printed differ because no RNG seed is set. This acts as an example observation so that the nature of the data that will guide the rest of these results is clear.

With set RNG seed 4444, the threshold with the highest mean winnings is 15 with earnings of -0.06775 per round. The worst threshold is 20, with mean winnings of -0.36833333.

An ANOVA test finds that there exist significant differences between the average earnings between different standing thresholds, with an astronomically low p-value in the order of 10^{-16}. Tukey's procedure then illuminates where the differences lie. There are many pairs to consider, but particularly noteworthy is that the best strategy, standing at 15, differs significantly from thresholds of 11, 18, 19, and 20, but not enough from the rest to be considered significantly better.

The simulation for different deck numbers therefore used a strategy of standing at 15. Of these deck numbers, 9 decks had the highest average earnings with -0.05808333, while 7 decks had the lowest with -0.08591667. A test for linear correlation between number of decks and average winnings found no correlation between the two, with an R-squared value in the order of 10^{-5}. Furthermore, an ANOVA failed to find any significant differences between the average winnings for different numbers of decks.

V. Discussion

Notably, but likely not surprisingly, even the best betting strategy of the ones tested results in negative winnings on average. Long-term, the dealer will always make more money than the player. The reason for this is a quirk in the design of the game: since the player hits before the dealer, the player will have the first opportunity to bust, which means the dealer wins the round on the spot even if the next card would have hypothetically busted the dealer. This edge is enough that even the most ideal strategy will favor the dealer, and as with all of their games, casinos have meticulously planned the payouts of Blackjack so that they are expected to win long-term.

So the game of Blackjack ultimately comes down to mitigating losses more than maximizing returns. According to this simulation, if a standing threshold is the desired strategy, then the best way to lose as little as possible is to stand if the hand value adds to 15 or greater and hit otherwise. Although Tukey's procedure deemed this value not significantly better than 12, 13, 14, 16, or 17, looking at the averages clearly shows a pattern; the average winnings keep increasing until 15, after which they start decreasing. More simulations, perhaps with more observations and trying out different set sizes, would help to confirm the edge that 15 has above the other thresholds.

The results regarding deck number are not particularly surprising; since only about four to eight cards will normally be in play at a time, the cards being dealt without replacement makes only a small impact with one deck, and this impact nearly disappears with the addition of only one more deck. Any number of decks greater than two change the probabilities at play a negligible amount, so the fact that the same strategy begets roughly the same results (with variance) is to be expected. Having multiple decks seems to be a way to discourage card counting and certain forms of cheating more than a way to impact the odds.

VI. Literature Cited

[1] https://rpubs.com/jt_rpubs/279263 - much of the simulation script was adapted from the material here
[2] https://www.r-bloggers.com/one-way-analysis-of-variance-anova/ - a resource used to help recall the syntax for running an ANOVA
[3] https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/TukeyHSD - resource about running Tukey's procedure in R

VII. Appendix

The following is the script used to run this experiment:

```
##########################################################################
# Function sum_cards: sums the values of the cards in a blackjack hand. Aces
# are counted as either 11 or 1 depending on the sum total of non-ace cards in the hand
#
# Parameters:
# - hand: a blackjack hand that has been dealt to either a player or dealer.
#
# NOTE: Blackjack hands in this simulation consist solely of integer values since the
#      cards do not need to be displayed graphically. Suits of cards also are
irrelevant
#      for similar reasons
# ==========================================================================

sum_cards <- function(hand) {

  aces <- FALSE
  ace_locs <- numeric()
  sum <- 0
```

Jake Seary
Statistics 486

```
        # if the hand is only 2 cards, just sum and exit
        if (length(hand) == 2) {
             sum <- hand[1] + hand[2]
             return(sum)
        }

        # find aces
        for (k in 1:length(hand)) {
             if(hand[k] == 11) {
             aces <- TRUE
             ace_locs <- c(ace_locs, k)
             }
        }

        # if no aces were found, sum all items and return
        if (aces == FALSE) {
             for (i in hand) {
             sum <- sum + i
             }
             return(sum)
        }

        # otherwise aces were found so first sum items that are NOT aces
        not_aces <- subset(hand, !(hand %in% c(11)))
        for (i in not_aces) {
             sum <- sum + i
        }

        # if sum of non-aces > 10 all aces must be treated as 1's
        if (sum > 10) {
             for (i in ace_locs) {
             sum <- sum + 1
             }
             return(sum)
        }

        # if sum of non-aces <= 9, need to check count of aces.
        # If > 1 one can be treated as 11 while other must be treated as 1
        if (sum <= 9 & length(ace_locs) > 1) {
             sum <- sum + 11
             for (i in 1:(length(ace_locs) - 1)) {
             sum <- sum + 1
             }
             return(sum)
        }

        # if sum of non-aces == 10, need to check count of aces.
        # if == 1, then add 11 to sum and exit. if > 1, treat all aces as 1's
        if (sum == 10 & length(ace_locs) == 1) {
             sum <- sum + 11
             return(sum)
        } else {
             for (i in 1:length(ace_locs)) {
             sum <- sum + 1
             }
             return(sum)
        }
```

```
}

########################################################################
# Function player_h: implements standard blackjack rules logic for the player's hand
# Parameters:
# - p_hand: the initial 2 cards dealt to the player
# - cards: a vector containing a pre-shuffled set of cards
# - i: an index variable that indicates the position of the card most recently dealt
#
# NOTE: Player strategy will be to take another card any time hand is 15 or less. This
#      implies that the player will stand with any hand of 16 or greater.
# =======================================================================

player_h <- function(p_hand, cards, i, p_stand){

  ncards <- length(cards)
  stand <- FALSE

  while (stand != TRUE) {
       # sum current hand
       current <- sum_cards(p_hand)

       # if card index == number of cards in decks, time to stop
       if (i == ncards) {
       stand <- TRUE

       # else if hand is < p_stand value, take another card
       } else if (current < p_stand) {
       i <- i + 1
       p_hand <- c(p_hand, cards[i])
       } else {
       stand <- TRUE
       }

  } # end while stand != TRUE

  return(c(current, i))
}

########################################################################
# Function dealer_h: implements standard blackjack rules logic for the dealer's hand
# Parameters:
# - d_hand: the initial 2 cards dealt to the dealer
# - cards: a vector containing a pre-shuffled set of cards
# - i: an index variable that indicates the position of the card most recently dealt
#
# NOTE: Dealer MUST stand on hand total of 17 or greater, so any time dealer hand is <=
16
#      another card must be drawn
# =======================================================================

dealer_h <- function(d_hand, cards, i){

  ncards <- length(cards)
  stand <- FALSE
```

Jake Seary
Statistics 486

```
  while (stand != TRUE) {
       # sum current hand
       current <- sum_cards(d_hand)

       # if card index == number of cards in decks, time to stop
       if (i == ncards) {
       stand <- TRUE

       # else if hand is <= 16, take another card
       } else if (current <= 16) {
       i <- i + 1
       d_hand <- c(d_hand, cards[i])
       } else {
       stand <- TRUE
       }

  } # end while stand != TRUE

  return(c(current, i))
}

##########################################################################
# Function play_bj: plays blackjack until hand is over, then records player's
# winnings and losses
#
# Parameters:
# - cards: a pre-shuffled set of cards
# - bet: the dollar amount of bets placeable by the player for each blackjack hand
#
# NOTE: need to sample WITHOUT replacement from cards. Since they are
# pre-shuffled before calling this function, just deal from top of deck and
# use an index to keep track of which cards haven't been dealt
# ========================================================================

play_bj <- function(cards, bet, p_stand){

  # initialize the number of cards to be played
  ncards <- length(cards)

  # initialize index for cards
  i <- 0

  # initialize winnings accummulator
  winnings <- 0

  # New hand needed: check to see whether at least 4 cards remain unused
       if (i <= (ncards - 4)) {

       #### Deal 2 cards to player
       # increment card index
       i <- i + 2
       p_hand <- c(cards[i-1], cards[i])

       ### Deal 2 cards to dealer
       # increment card index
       i <- i + 2
       d_hand <- c(cards[i-1], cards[i])
```

```
        } else {
        # else cards are exhausted so exit
        return(winnings)
        }

        #### logic for player's hand
        p_res <- player_h(p_hand, cards, i, p_stand)

        # update card index
        i <- p_res[2]

        # if player didn't go over 21 then apply dealer hand logic
        if (p_res[1] <= 21) {
        d_res <- dealer_h(d_hand, cards, i)

        # update card index
        i <- d_res[2]
        }

        # --------------------------------------------------------------
        # now compare player's hand to dealer's hand to find out who won

        # if player hand exceeded 21 subtract 1 from winnings
        if (p_res[1] > 21) {
        winnings <- winnings - 1

        # else if dealer went over 21 dealer has lost so add 1 to winnings
        } else if (d_res[1] > 21) {
        winnings <- winnings + 1

        # else if player hand > dealer hand add 1 to winnings
        } else if (p_res[1] > d_res[1]) {
        winnings <- winnings + 1

        # else if player hand < dealer hand subtract 1 from winnings
        } else if (p_res[1] < d_res[1]) {
        winnings <- winnings - 1
        }


    # end hand
  return(winnings)
}
##########################################################################
# This is the actual simulation. This part affects what is calculated and printed when
the script is run.

# NOTE: Suits of cards don't really matter at all. All we really need to track is the
number remaining for each class of card, e.g., how many 2's, how many 3's, etc..

# set number of 2-deck sets to play
N <- 12

# set size of bets
bet <- 1
```

Jake Seary
Statistics 486

```
# set hand total for player to stand at
p_stand <- 16

# generate a deck of cards
deck <- rep(c(2:10, 10, 10, 10, 11), 4)

# create a set of cards consisting of a number of decks
two_d <- rep(deck,2)

# create a vector to store results of play
res <- data.frame(N = 1:12, winnings = numeric(N))

for (i in 1:N){
  # shuffle 2 new decks of cards
  s_decks <-sample(two_d,length(two_d), replace = FALSE)
  s_decks

  # play until 2 decks are exhausted
  res$winnings[i] <- play_bj(s_decks, bet, p_stand)
}

# create a table of winnings for each iteration
iteration <- 1:N
print('Sample of twelve rounds, standing at 16:')
print(res$winnings)

# average the winnings and print to screen
avg_winnings <- mean(res$winnings)
print('Average Winnings per Hand:')
print(avg_winnings)

# set number of simulation iterations to be run
iters <- 1000

# create a vector to store results of play
res2 <- data.frame(N = 1:iters, avg_winnings = numeric(iters))

for (k in 1:iters) {
  for (i in 1:N){
        # shuffle 2 new decks of cards
        s_decks <-sample(two_d,length(two_d), replace = FALSE)
        s_decks

        # play until 2 decks are exhausted
        res$winnings[i] <- play_bj(s_decks, bet, p_stand)
  } # for i

  # save the average the winnings to the results vector
  res2$avg_winnings[k] <- mean(res$winnings)

} # for k

set.seed(4444)

# create a vector to store the average winnings for various player "stand" values
playw <- data.frame(stand = 11:20, avg_winnings = numeric(10))
```

Jake Seary
Statistics 486

```
p_stand <- 11

standcol <- NULL
datacol <- NULL

for (z in 1:10) {
  res4 <- data.frame(N = 1:iters, avg_winnings = numeric(iters))

  for (k in 1:iters) {
        for (i in 1:N){
        # shuffle 2 new decks of cards
        s_decks <-sample(two_d,length(two_d), replace = FALSE)

        # play until round ends
        res$winnings[i] <- play_bj(s_decks, bet, p_stand)
        } # for i

  # average the winnings
  res4$avg_winnings[k] <- mean(res$winnings)
  standcol <- c(standcol, p_stand)

  } # for k

  # save the average winnings to the results vector
  playw$avg_winnings[z] <- mean(res4$avg_winnings)
  datacol <- c(datacol, res4$avg_winnings)
  # increase the player's stand value
  p_stand <- p_stand + 1

} # for z

print('Earnings after 1000 12-round sets for various stand values:')
print(playw)

standdata <- data.frame(stand = standcol, winnings = datacol)
standdata$stand <- as.factor(standdata$stand)

print('ANOVA for testing if strategies significantly differ:')

aov.out <- aov(winnings ~ stand, data = standdata)
print(summary(aov.out))

print('Using Tukey\'s procedure to compare between thresholds:')
ttest.out <- TukeyHSD(aov.out)
print(ttest.out)


p_stand <- 15 #sets ideal strategy
maxsize <- 12 #testing all deck sizes up to 12
sizechange <- data.frame(size = 1:maxsize, avg_winnings = numeric(maxsize))

sizecol <- NULL
datacol <- NULL

for (decksize in 1:maxsize)
{
    new_deck = rep(deck, decksize)
```

Jake Seary
Statistics 486

```
            res5 <- data.frame(N = 1:iters, avg_winnings = numeric(iters))
            for (k in 1:iters)
            {
                for (i in 1:N)
                {
                       #shuffles decks
                       shuffled <- sample(new_deck, length(new_deck), replace = FALSE)
                       #play until round ends
                       res$winnings[i] <- play_bj(shuffled, bet, p_stand)
                }
            res5$avg_winnings[k] <- mean(res$winnings)
            sizecol <- c(sizecol, decksize)
            }
            sizechange$avg_winnings[decksize] <- mean(res5$avg_winnings)
            datacol <- c(datacol, res5$avg_winnings)
        }

        print('Comparisons of stand at 15 strategy with different numbers of decks:')
        print(sizechange)

        realdata <- data.frame(size = sizecol, winnings = datacol)
        print(summary(realdata$winnings))

        print('Linear correlation between number of decks and winnings')
        linearfit <- lm(winnings ~ size, data = realdata)
        print(summary(linearfit))

        print('Analysis of variance to find any significant differences between deck counts')
        realdata$size <- as.factor(realdata$size)
        aov.out <- aov(winnings ~ size, data = realdata)
        print(summary(aov.out))
```

The following is an example of output from running the script:

```
> source('blackjack.r')
[1] "Sample of twelve rounds, standing at 16:"
 [1]  1 -1  0 -1 -1  1  1 -1 -1  1 -1 -1
[1] "Average Winnings per Hand:"
[1] -0.25
[1] "Earnings after 1000 12-round sets for various stand values:"
   stand avg_winnings
1      11  -0.12958333
2      12  -0.09841667
3      13  -0.09191667
4      14  -0.08341667
5      15  -0.06775000
6      16  -0.08441667
7      17  -0.09200000
8      18  -0.12708333
9      19  -0.20800000
10     20  -0.36833333
[1] "ANOVA for testing if strategies significantly differ:"
              Df Sum Sq Mean Sq F value Pr(>F)
stand          9   74.7   8.295   111.8 <2e-16 ***
Residuals   9990  741.1   0.074
---
```

Jake Seary

Statistics 486

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "Using Tukey's procedure to compare between thresholds:"
  Tukey multiple comparisons of means
       95% family-wise confidence level

Fit: aov(formula = winnings ~ stand, data = standdata)

$stand
               diff           lwr          upr    p adj
12-11  3.116667e-02 -0.0073773387  0.069710672 0.2376681
13-11  3.766667e-02 -0.0008773387  0.076210672 0.0618919
14-11  4.616667e-02  0.0076226613  0.084710672 0.0058750
15-11  6.183333e-02  0.0232893279  0.100377339 0.0000173
16-11  4.516667e-02  0.0066226613  0.083710672 0.0080018
17-11  3.758333e-02 -0.0009606721  0.076127339 0.0631354
18-11  2.500000e-03 -0.0360440054  0.041044005 1.0000000
19-11 -7.841667e-02 -0.1169606721 -0.039872661 0.0000000
20-11 -2.387500e-01 -0.2772940054 -0.200205995 0.0000000
13-12  6.500000e-03 -0.0320440054  0.045044005 0.9999501
14-12  1.500000e-02 -0.0235440054  0.053544005 0.9674325
15-12  3.066667e-02 -0.0078773387  0.069210672 0.2588546
16-12  1.400000e-02 -0.0245440054  0.052544005 0.9794556
17-12  6.416667e-03 -0.0321273387  0.044960672 0.9999553
18-12 -2.866667e-02 -0.0672106721  0.009877339 0.3544402
19-12 -1.095833e-01 -0.1481273387 -0.071039328 0.0000000
20-12 -2.699167e-01 -0.3084606721 -0.231372661 0.0000000
14-13  8.500000e-03 -0.0300440054  0.047044005 0.9995325
15-13  2.416667e-02 -0.0143773387  0.062710672 0.6103913
16-13  7.500000e-03 -0.0310440054  0.046044005 0.9998334
17-13 -8.333333e-05 -0.0386273387  0.038460672 1.0000000
18-13 -3.516667e-02 -0.0737106721  0.003377339 0.1091911
19-13 -1.160833e-01 -0.1546273387 -0.077539328 0.0000000
20-13 -2.764167e-01 -0.3149606721 -0.237872661 0.0000000
15-14  1.566667e-02 -0.0228773387  0.054210672 0.9569215
16-14 -1.000000e-03 -0.0395440054  0.037544005 1.0000000
17-14 -8.583333e-03 -0.0471273387  0.029960672 0.9994939
18-14 -4.366667e-02 -0.0822106721 -0.005122661 0.0125247
19-14 -1.245833e-01 -0.1631273387 -0.086039328 0.0000000
20-14 -2.849167e-01 -0.3234606721 -0.246372661 0.0000000
16-15 -1.666667e-02 -0.0552106721  0.021877339 0.9368526
17-15 -2.425000e-02 -0.0627940054  0.014294005 0.6055309
18-15 -5.933333e-02 -0.0978773387 -0.020789328 0.0000493
19-15 -1.402500e-01 -0.1787940054 -0.101705995 0.0000000
20-15 -3.005833e-01 -0.3391273387 -0.262039328 0.0000000
17-16 -7.583333e-03 -0.0461273387  0.030960672 0.9998174
18-16 -4.266667e-02 -0.0812106721 -0.004122661 0.0167085
19-16 -1.235833e-01 -0.1621273387 -0.085039328 0.0000000
20-16 -2.839167e-01 -0.3224606721 -0.245372661 0.0000000
18-17 -3.508333e-02 -0.0736273387  0.003460672 0.1111600
19-17 -1.160000e-01 -0.1545440054 -0.077455995 0.0000000
20-17 -2.763333e-01 -0.3148773387 -0.237789328 0.0000000
19-18 -8.091667e-02 -0.1194606721 -0.042372661 0.0000000
20-18 -2.412500e-01 -0.2797940054 -0.202705995 0.0000000
20-19 -1.603333e-01 -0.1988773387 -0.121789328 0.0000000

[1] "Comparisons of stand at 15 strategy with different numbers of decks:"
   size avg_winnings
```

Jake Seary
Statistics 486

```
1      1  -0.07766667
2      2  -0.07216667
3      3  -0.06375000
4      4  -0.08033333
5      5  -0.07925000
6      6  -0.08341667
7      7  -0.08591667
8      8  -0.06383333
9      9  -0.05808333
10   10  -0.08225000
11   11  -0.06166667
12   12  -0.07466667
       Min.  1st Qu.   Median       Mean  3rd Qu.  Max.
-1.00000 -0.25000 -0.08333 -0.07358  0.08333  0.83333
[1] "Linear correlation between number of decks and winnings"

Call:
lm(formula = winnings ~ size, data = realdata)

Residuals:
       Min     1Q   Median    3Q     Max
-0.92454 -0.17775 -0.00895  0.15986  0.90986

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.0770606  0.0053752 -14.336   <2e-16 ***
size           0.0005350  0.0007303   0.732 0.464
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2762 on 11998 degrees of freedom
Multiple R-squared:  4.472e-05,    Adjusted R-squared:  -3.863e-05
F-statistic: 0.5365 on 1 and 11998 DF,  p-value: 0.4639

[1] "Analysis of variance to find any significant differences between deck counts"
              Df  Sum Sq Mean Sq F value Pr(>F)
size           11     1.0 0.09050   1.187   0.29
Residuals   11988  914.2 0.07626
```